z/OS

UNIX System Services Programming: Assembler Callable Services Reference

Version 2 Release 1

SA23-2281-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 1345.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Appendix C. Mapping macros—AMODE 64 1085

Appendix D. Callable services examples—AMODE 31 1123

Appendix E. Callable services examples—AMODE 64 1215

Figures

Tables

About this document

This document describes the features and usage requirements for the z/OS UNIX System Services (z/OS UNIX) callable services. These services are interfaces between the z/OS® operating system and standard (POSIX or Single UNIX Specification) programming functions that require operating system services. For example, programmers creating runtime library programs use these services. This book also describes callable services that are not related to the standard interfaces.

System programmers coding programs in assembler can use these callable services to obtain the z/OS UNIX services they need. This document contains detailed information—such as the function, requirements, syntax, linkage information, parameters, and usage information—that is needed to use the services. In the appendixes you will find information about:

- System control offsets to callable services
- Mapping macros
- Callable service examples
- The relationship of signals to callable services
- Using threads with callable services
- Optimizing performance using process- and thread-level information
- Callable services available to SRB mode routines
- z/OS UNIX process start/end exits
- Accessibility features
- Notices
- An index

Who should use this document

This document is for assembler programmers who want to use z/OS UNIX System Services.

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to the IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

z/OS UNIX courses

For a current list of courses that you can take, go to IBM Education home page (http://www.ibm.com/services/learning/).

z/OS UNIX home page

Visit the z/OS UNIX home page at z/OS UNIX home page (http://www.ibm.com/systems/z/os/zos/features/unix/).

Some of the tools available from the website are ported tools, and some are unsupported tools designed for z/OS UNIX. The code works in our environment at the time we make it available, but is not officially supported. Each tool has a readme file that describes the tool and lists any restrictions.

The simplest way to reach these tools is through the z/OS UNIX home page. From the home page, click on **Tools and Toys**.

The code is also available from **ftp://ftp.software.ibm.com/s390/zos/unix/** through anonymous FTP.

Because the tools are not officially supported, APARs cannot be accepted.

Discussion list

Customers and IBM® participants also discuss z/OS UNIX on the **mvs-oe discussion list**. This list is not operated or sponsored by IBM.

To subscribe to the mvs-oe discussion, send a note to:

listserv@vm.marist.edu

Include the following line in the body of the note, substituting your given name and family name as indicated:

```
subscribe mvs-oe given_name family_name
```

After you have been subscribed, you will receive further instructions on how to use the mailing list.

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R1.0 UNIX System Services Programming: Assembler Callable Services Reference
  SA23-2281-01
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated February, 2015

The following changes are made for z/OS Version 2 Release 2 (V2R1) as updated February, 2015.

New

- Support was added for vectors. See "ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging" on page 537 for information about PT_READ_VR and PT_WRITE_VR.

Changed

- The BPXYPTRC macro was updated as a result of the added vector support. See "BPXYPTRC — Map parameters for ptrace" on page 1018.

Deleted

No content was removed from this information.

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Chapter 1. Invocation details for callable services

As an interface between the z/OS operating system and the functions specified in the Single UNIX Specification and earlier standards, z/OS UNIX System Services (z/OS UNIX) provides access to assembler callable services (syscalls). The z/OS UNIX callable services have a standard set of syntax and linkage requirements, as well as parameter specification details necessary for successful invocation.

Connecting to and disconnecting from z/OS UNIX System Services

To connect to the kernel for z/OS UNIX System Services, you make an address space known to it. This process is called *dubbing*. After it has been dubbed, the address space is considered to be a *process*. Address spaces that are created by fork are automatically dubbed when they are created. Other address spaces become dubbed if they invoke a z/OS UNIX service. Dubbing also applies to MVS™ tasks. A dubbed task is considered a *thread*. Tasks created by pthread_create are automatically dubbed threads; other tasks are dubbed if they invoke a z/OS UNIX service.

Undub is the inverse of *dub*. Normally, a task (dubbed a thread) is undubbed when it ends. An address space (dubbed a process) is undubbed when the last thread ends.

If, when a thread or process is being dubbed, the calling task has a task-level ACEE that does not have a USP connected to it, an INITUSP is done against the task-level ACEE. This causes z/OS UNIX security information to be associated with the task-level ACEE.

Syntax conventions for the callable services

A callable service is a programming interface that uses the CALL macro to access system services. To code a callable service, code the CALL macro followed by the name of the callable service and a parameter list. A syntax diagram for a callable service follows.

```
CALL Service_name,(Parm_1,
                   Parm_2,
                   .
                   .
                   Return_value,
                   Return_code,
                   Reason_code)
```

This format does not show the assembler column conventions (columns 1, 10, 16, and 72) or parameter list options (VL and MF). The exact syntax is shown in the examples in Appendix D, "Callable services examples—AMODE 31," on page 1123.

When you code a callable service you must:

- Code all the parameters in the parameter list, because parameters are positional in a callable service interface. The function of each parameter is determined by

its position with respect to the other parameters in the list. Omitting a parameter, therefore, assigns the omitted parameter's function to the next parameter in the list.

- Place values explicitly into all supplied parameters, because callable services do not set defaults.

CALL

CALL is the assembler macro that transfers control and passes a parameter list.

Service_name

For AMODE 31 callers, the name that the assembler understands is the name of a module in the form BPX1xxx, where *xxx* is a three-character symbol unique to the service. (In a few cases, where both standard and nonstandard versions of a service exist, the standard version of the service is in the form BPX2xxx.) AMODE 64 callers use the name of a module in the form BPX4xxx. (See "Using callable services in a 64-bit environment" on page 9.)

Modules are invoked in one of the following ways:

- A program can load a module, and then branch to the address where it was loaded.
- When you link-edit a program, you can link to the linkage stub. The program can issue a call.

  The linkage stubs are contained in SYS1.CSSLIB. You can specify SYS1.CSSLIB in the //SYSLIB statement of the JCL that is used to invoke the linkage editor. This causes the addresses of all required linkage-assist routines to be automatically resolved, and saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

  For BPX4xxx stubs, you need 64-bit binder support to do the link-edit. See "Using callable services in a 64-bit environment" on page 9.
- You can include in the code the system control offset to the callable service. See Appendix A, "System control offsets to callable services," on page 939 for information on how to use this linkage.

For information about using callable services in AMODE 64, see "Using callable services in a 64-bit environment" on page 9.

Parm parameters

Parm_1, Parm_2, and so on are placeholders for variables that may be part of a service's syntax.

Return_value

The Return_value parameter is common to many callable services, and indicates the success or failure of the service. If the callable service fails, it returns a -1 in the Return_value. For most successful calls to z/OS UNIX services, the return value is set to 0. However, some services, such as "getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID" on page 223 and "getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name" on page 226, return zeros instead of -1 when the service fails.

Some callable services, such as "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185, return a positive return value to indicate success. Other

services, such as "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150, are unique in that they do not return when successful.

Some services do not have a return value, because under normal conditions they do not fail. System failures, however, may cause these services to fail, in which case, the process that issues the call ends abnormally. See "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217 for an example.

Return_code

The Return_code parameter is referred to as the *errno* in the POSIX and X/Open C interfaces. The Return_code is returned only if the service fails.

In the callable service descriptions, some of the possible return codes are listed for services that have return codes. The return codes are described in each service if they help to describe its function.

For each return code, any reason code that may accompany it is identified.

The return codes and their descriptions can be found in *z/OS UNIX System Services Messages and Codes*.

Some Return_code values may occur for any callable service: the return codes that are unique to z/OS UNIX. They are not always listed under each callable service. See *z/OS UNIX System Services Messages and Codes* for a description of these return codes.

The following five return codes can occur with any callable service, and are not listed with each service because the failure may occur before the syscall gets control:

Return_code	**Explanation**
EFAULT	An address is incorrect, usually because it is a zero pointer, an uninitialized pointer, or a pointer to read-only storage (for example, a program constant of zero) for a parameter that is (or could be, in a different context) an output parameter.
EMVSINITIAL	A process initialization error has occurred.
EMVSERR	An MVS environmental or internal error has occurred.
EMVSPARM	Bad parameters were passed to the service.
ENOMEM	Not enough space is available to fill the request.

Reason_code

The Reason_code parameter usually accompanies the Return_code value when the callable service fails, and further defines the return code. Reason codes do not have an equivalent in the POSIX or X/Open standards.

The reason codes and their descriptions can be found in *z/OS UNIX System Services Messages and Codes*. Reason codes are listed by name and numerically by value. The value is the lower half of the reason code.

Determining the callable service release level

New callable services may be added with each new z/OS UNIX release. Depending on the operating environment, the caller may have to determine the release level of z/OS UNIX before a new callable service can be issued.

The release information is indicated in the CVT feature flags. For z/OS V1R1, the feature flag is:

```
CVTJ7713 EQU   X'20'         JBB7713
```

Linkage conventions for the callable services

Callers must use the following linkage conventions for all z/OS UNIX callable services:

- Register 1 is set up by the CALL macro. In 31-bit mode, it contains the address of a parameter list, which is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit. In 64-bit mode, register 1 is 8 bytes long, and contains a 64-bit address that points to a list of 64-bit addresses. See "Using callable services in a 64-bit environment" on page 9.
- Register 14 is set up by the CALL macro; it contains the return address.
- Register 15 is set up by the CALL macro; it contains the entry point address of the service stub that is being called.

On return from a callable service, general and access registers 2 through 13 are restored. General and access registers 0, 1, 14, and 15 are not restored.

The caller must be running with 31-bit or 64-bit addressing (AMODE=31 or AMODE=64), because the linkage code uses control blocks that reside above the 16-MB line.

See *z/OS MVS Program Management: Advanced Facilities* for detailed linkage information.

Parameter descriptions for the callable services

All the parameters of the callable services are required positional parameters. When you specify a call, you must specify all the parameters in the order listed.

Note: Some parameters do not require values, and allow you to substitute zeros for the parameter. The descriptions of the parameters identify those that can be replaced by zeros, and when to do so.

In the descriptions of the calls, each parameter is described as *supplied* or *returned*:

- **Supplied** means that you supply a value for the parameter in the call.
- **Returned** means that the service returns a value in the named parameter when the call is finished (for example, *Return_code*).
- Some parameters are both supplied and returned.

Each parameter is also described in terms of its *data type* and *length*:

- **Data type** is one of the following: integer, address, character string, or structure.
- **Length** depends on the data type of the parameter:
  - For an address item, the length is a fullword (for AMODE 31 callers) or a doubleword (for AMODE 64 callers).
  - For an integer item, the length indicates the size of the field in bytes or fullwords: bytes are 1, 2, 3, 4, or 8.
  - For a character string parameter, the length indicates the number of characters that can be contained in a character-type parameter.

– For a structure parameter, the length indicates the size of the structure in bytes or fullwords, or refers to a label in the structure's mapping macro that defines the length.

Call parameter lists

Every callable service is called with a parameter list. As shown in Figure 1, when a service is called:

- Register 1 points to a parameter address list.
- Each field in the parameter address list points to a field containing a parameter.
- The "parameter list" is the set of those parameters; however they are arranged in storage. For AMODE 31 callers, the last parameter pointer in the list must have the high-order bit set to 1. For AMODE 64 callers, there is no end-of-parameter list indicator; the high-order bit is part of the 64-bit address.

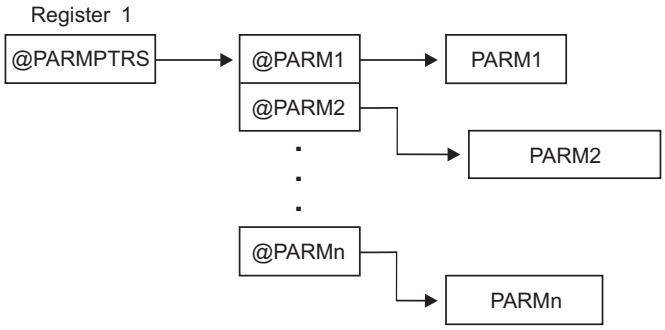


Figure 1. Call parameter list

Mapping macros

In many callable services, mapping macros map the parameter options. A complete list of the options for each macro is listed in the macro in “Macros mapping parameter options” on page 945.

Most of the mapping macros can be expanded with or without a DSECT statement. The invocation operand DSECT=YES is the default.

AMODE 64 callers using the 64-bit versions of the macros must issue SYSSTATE AMODE64=YES to identify the addressing mode. See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for information about the SYSSTATE macro.

Examples

An invocation example for each callable service is in Appendix D, “Callable services examples—AMODE 31,” on page 1123. These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry. The examples are arranged alphabetically, and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

Reentrant coding versus nonreentrant coding

See “BPX1GTH (__getthent) example” on page 1152 for an example of the __getthent service, which uses reentrant code. Compare this example with an

example of nonreentrant code for the same service in “Example of nonreentrant entry linkage—AMODE 31” on page 1307, and note the following:

- Placement of the standard 18-word register save area
- Use of program and dynamic storage base registers
- @DYNAM DSECT in the reentrant version
- Different forms of the CALL macro
- Several variables (such as PGPSCONTTYBLEN) that are initialized by the assembler in the nonreentrant version (see “BPXYPGPS — Map the response structure for w_getpsent” on page 1007 for the DCs), and at execution time with moves and stores in the reentrant version.

Environmental restrictions

Callers must be aware of the following restrictions for all z/OS UNIX callable services:

Functional recovery routines (FRR)
Except for callable services that are supported in service request block (SRB) mode, do not invoke a callable service with an FRR set because doing so bypasses callable services recovery and can severely damage the system. (If a callable service can run in SRB mode, that is stated in its description.)

Linkage stack
The use of the system linkage stack with PC or BAKR instructions prevents signals from being delivered.

Locks Do not call z/OS UNIX with system locks held. Testing is not done for locks held, and your call might fail.

Nested callable services
You cannot issue "nested" callable services. That is, if a program running on a request block (RB) issues a z/OS UNIX callable service and is then interrupted by a program running on an interrupt request block (IRB), any additional z/OS UNIX callable services that the IRB attempts to issue are not supported. Additionally, if a z/OS UNIX callable service invokes an exit during the processing of the callable service, invoking z/OS UNIX callable services from the exit program is not supported. In most cases, the nested callable service invocation is detected and flagged as an error. In some cases, however, the nested invocation is not detected and can lead to failure of the original callable service invocation.

Task structure
When you invoke callable services in task control block (TCB) mode, the calling TCB must be either the initial job step task or a subtask of the initial JST. The initial JST is the JST that is directly attached by the operating system initiator task to run a user requested program. z/OS UNIX does not support the direct attachment of multiple JSTs from the initiator task. The behavior of z/OS UNIX callable services in an environment where multiple job step tasks are attached directly from the initiator task is unpredictable.

Restrictions in a multiprocess, multiuser environment

Programs that change the security environment cannot run in a multiprocess, multiuser environment. A multiprocess, multiuser environment is an environment in which there are multiple z/OS UNIX processes in an address space (enabled by the environment variable _BPX_SHAREAS=YES.) Each process has a different MVS identity; that is, it has its own process-level ACEE anchored at the TCB (TCBSenv) level. To prevent a user running under one MVS identity from affecting all the other processes in the address space, or creating a new process with an identity other than the one the user is running under, certain callable services are restricted.

These z/OS UNIX callable services are restricted in a multiprocess, multiuser environment, and will fail with JRMpMuProcess:

- BPX1ATM/BPX4ATM (attach_execMVS) — ASM only
- BPX1ATX/BPX4ATX (attach_exec) — ASM only
- BPX1SEG/BPX4SEG (setegid)
- BPX1SGI/BPX4SGI (setgid)
- BPX1SPN/BPX4SPN (spawn family)
- BPX1SRG/BPX4SRG (setregid)

See the descriptions of these callable services for further information about the restrictions.

Abend conditions and environments

Callers must be aware of the following conditions that can cause an abnormal end:

- When the _exit service, BPX1EXI/BPX4EXI, is called in any environment except single task, single RB, and no linkage stack, the system issues an abend EC6. This abend ends the calling task and all of its subtasks. The subtasks receive a 442 abend. If the caller is a thread task created with the pthread_create service, the initial pthread creating task abends with a 422 abend code. All subtasks of the initial pthread creating task receive a 442 abend.
- Some POSIX services are defined as always successful, yet the kernel can get program checks or other MVS abends. When these failures occur, the user receives an EC6 abend code.
- There are SLIP traps that recognize z/OS UNIX abends as normal exec service and _exit service processing. Dumps are suppressed, and the new tasks for the exec service are created. These SLIP traps are shipped as part of IEASLP00. If your system does not use IEASLP00 as provided by z/OS, you will need to copy the SLIP commands for EC6 and 422 abends into their SLIP command parmlib member. Otherwise, your system will generate an excessive number of dumps.
- Condition codes (cc) seen by the next step in a multistep job cause an abnormal end:
  - **Case 1:**
    1. The step invokes the C main program.
    2. The C main program invokes the exit or _exit service, specifying the return code.
    3. The return code surfaces as the step condition code.
  - **Case 2:**

    When you return from the main program, the condition code is in R15 at the time of exit.

- Signals that are not caught often cause a task to end abnormally. z/OS UNIX defines which signals generate dumps. Terminating signals that do not require user dumps have an abend code of EC6 with a reason code 0000FFxx, where *xx* is the signal number. Parmlib member IEASLP00 has a statement to suppress all dumps that match this profile. Terminating signals that require that a user dump be taken (if requested) have an abend code of EC6 with a reason code 0000FDxx, where *xx* is the signal number. Parmlib member IEASLP00 has a statement to suppress all SDUMPs that match this profile but that allow user dumps to be taken.
- If a process abends while it is being debugged with ptrace by a debugger program such as **dbx**, the debugger may be notified of the abend. The notification occurs if the tested program's recovery calls ptrace. This is normally true for C programs, because the C runtime library establishes the necessary recovery environment to call ptrace.

Callable service failures

A typical application that receives an unexpected return code from a callable service usually exits. If an application is written to handle unexpected errors, you need to understand the following information:

Services can fail for a number of reasons: bugs in the system, user code that causes failure return codes, or abend conditions. Depending on when the failure occurs in the service path, the requested function may or may not have been performed. For example, if the application provides an address for a file descriptor that does not exist, the open service (BPX1OPN/BPX4OPN) completes the open processing and then fails on the return path when trying to set the file descriptor. If an EFAULT return code is returned, the user may assume that the file was not opened, even though it was.

If the return value parameter is not in valid storage, a service can complete successfully, yet not return normally to the caller. Because the service cannot set the return value, it abends. It is possible for the C runtime library to convert the return value into a **SIGABND** or **SIGSEGV** signal, which can be caught and handled by the user signal action defined in sigaction. You should be aware that functions that abend in this way may have completed their processing. For example, a call to sigaction could modify the state of signal information and then fail on the return to the caller. In this case, the caller should not make any assumptions about the state of the signal environment.

Authorization

Users authorized to perform special functions are defined as having *appropriate privileges*, and are called *superusers*. Users with appropriate privileges are also those with:

- A user ID of zero
- RACF-supported user privileges *trusted* and *privileged*, regardless of their user ID

The ability to change the MVS identity of an address space is reserved for a subset of superusers who control daemons. A daemon is a process that verifies the identity of a user before creating a process to run work on behalf of the user. This approach allows the installation to have superusers whose job is to maintain the file system and user processes, but who do not have the ability to change their user identity. See Setting up the BPX.* FACILITY class profiles in *z/OS UNIX System Services Planning* for a description of the BPX.DAEMON resource profile in

the RACF® FACILITY class and how it is created. This information also describes additional BPX.xxxxxxxx resource profiles in the FACILITY class that are used to provide selective permission to certain restricted functions.

Also, superusers are said to have *daemon authority* if the BPX.DAEMON resource profile is defined and they have access to it. If BPX.DAEMON is not defined, the users have daemon authority if they are a superuser.

Note that aliases can be supplied for user IDs. Callable services that pass or receive user ID parameters may need to use the userid alias table. Its use is described in USERIDALIASTABLE in *z/OS UNIX System Services Planning*.

Note: This information assumes that your operating system contains Resource Access Control Facility (RACF). You could use an equivalent security product updated to handle z/OS UNIX security.

Using callable services in a 64-bit environment

The z/OS UNIX callable services can be called in 31-bit or 64-bit addressing mode (AMODE 31 or AMODE 64). RMODE 64 is not supported. AMODE 31 callers use the BPX1xxx services; AMODE 64 callers use the BPX4xxx services.

Calling programs using 64-bit addressing must be compiled AMODE 64, and clear the upper half of the branch register before branching to the syscall layer. 64-bit binder support is needed to do the link-edit.

An AMODE 64 caller using the 64-bit versions of the macros must issue SYSSTATE AMODE64=YES before calling the service. See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for information about the SYSSTATE macro.

There is a BPX4xxx stub for each BPX1xxx service, except for those services that were replaced in functionality by other services:

Table 1. Callable services with no BPX4xxx counterparts

Callable service	Replaced by
BPX1GPS	BPX1GTH
BPX1TYN	BPX2TYN
BPX1ITY	BPX2TYN
BPX1MNT	BPX2MNT
BPX1RMS	BPX2RMS
BPX1SMS	BPX2SMS

The kernel continues to support the BPX1xxx versions of these syscalls.

Except for the services in Table 1, the last three characters of the stub names are the same for the 31-bit and 64-bit stubs.

Call parameter lists

AMODE 31 callers of the BPX1xxx service provide a standard 31-bit parameter list. Register 1 contains a 31-bit address that points to a list of 31-bit addresses, which point to the parameters. The last parameter pointer in the list must have the high-order bit set to 1. All storage is below the bar.

AMODE 64 callers of the BPX4xxx service provide a 64-bit address in register 1 that points to a list of 64-bit addresses, which point to the parameters. There is no end-of-parameter list indicator; the high-order bit is part of the 64-bit address. The parameter list and the parameters may or may not exist above the bar.

Parameters

For most of the callable services, the parameters for the 31-bit and 64-bit versions are the same; the only difference between the versions is the AMODE of the caller. There is no change in the number or length of the parameters. An example of this type is setuid():

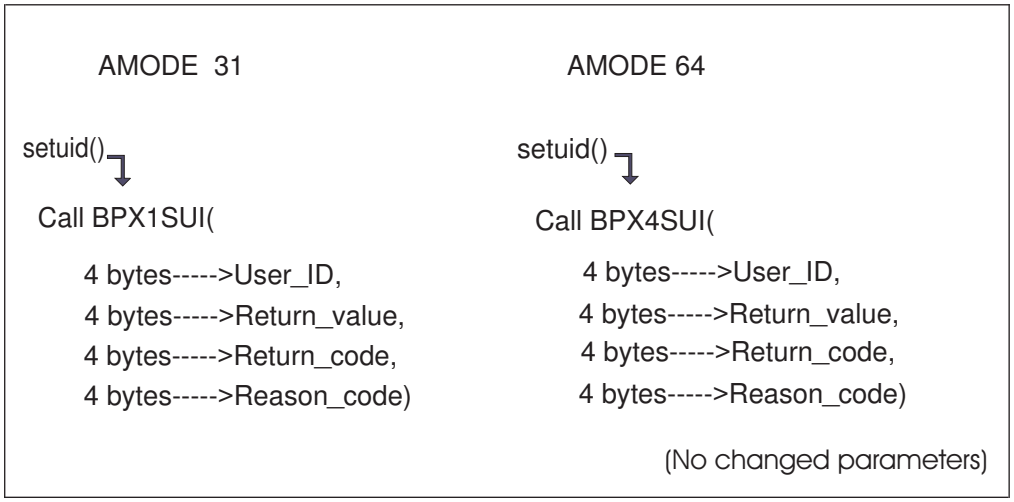


Figure 2. setuid() parameters — AMODE 31 and AMODE 64

The other callable services can be divided into three groups:

1. **Callable services that have doubleword instead of fullword fields for parameter addresses in the 64-bit version**

   Some callable services, such as shmget(), have doubleword instead of fullword fields for parameter addresses in the 64-bit version. As shown in Figure 3 on page 11, the 64-bit version of the shmget() service has a change in the size of the second parameter. For the 31-bit version (BPX1MGT), the Shared_Memory_Size parameter is 4 bytes long. For the 64-bit version (BPX4MGT), the Shared_Memory_Size parameter is 8 bytes long, to accommodate a possible new size of 16 petabytes:

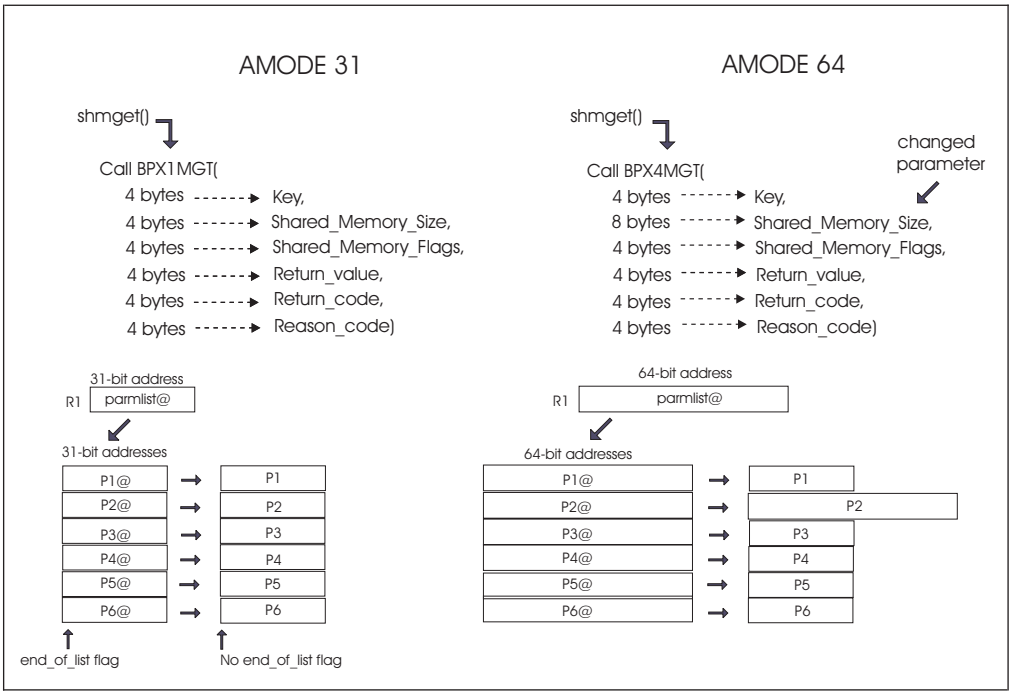


Figure 3. shmget() parameters — AMODE 31 and AMODE 64

2. **Callable services that have an additional parameter in the 64-bit version, such as loadhfs()**

   Other callable services, such as loadhfs(), have an additional parameter in the 64-bit version. Where addresses or lengths are passed back in the Return_value parameter, an 8-byte parameter is added to the 64-bit version of the service. (For compatability issues, the Return_value, Return_code, and Reason_code will always remain 4-byte fields.)

   The 64-bit version of the loadhfs() service has an additional parameter. For BPX1LOD, the entry point address of the loaded HFS executable is returned in the Return_value parameter. For BPX4LOD, the 64-bit entry point address is returned in the 8-byte entry_point parameter.

Invocation details

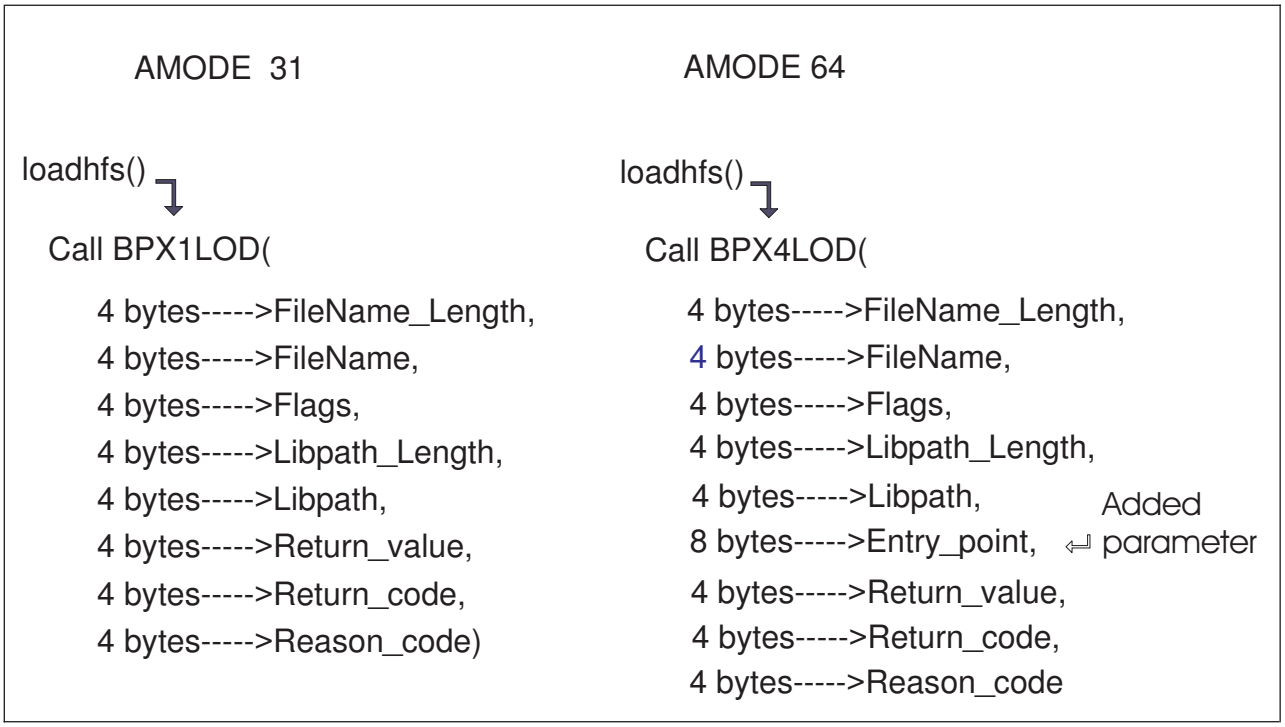


Figure 4. loadhfs() parameters — AMODE 31 and AMODE 64

3. **Callable services that use parameter structures with address fields that are at different offsets in the 64-bit version**

   A subset of callable services use parameter structures whose 64-bit address fields are at different offsets from their 31-bit counterparts; or whose address fields, because of their increased size, have caused other fields in the structure to be at different offsets for AMODE 64 callers. These are:

   - BPX4AIO, with the AIOCB structure
   - BPX4RDV and BPX4WRV, with the iov structure
   - BPX4SMS and BPX4RMS, with the MSGH and iov structures

   For example, in the 64-bit version of the readv service, BPXB4RDV, the 64-bit address fields in the iov structure are at different offsets within the iov from their 31-bit counterparts:

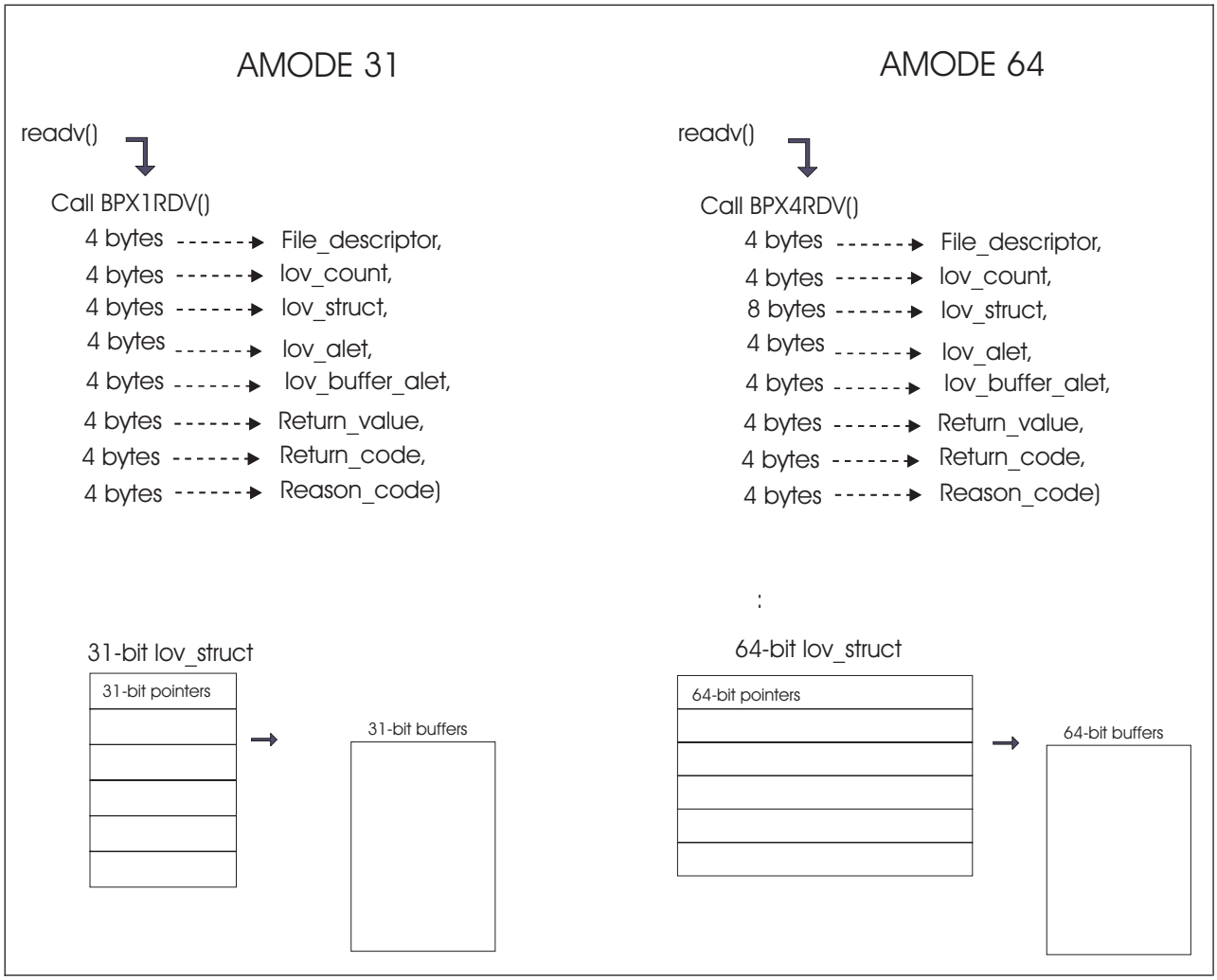


Figure 5. readv iov structure — AMODE 31 and AMODE 64

The descriptions of the individual callable services contain detailed information about using the services in 64-bit AMODE.

System control offsets

The offsets into the callable services table for the BPX4xxx calls are the same as the offsets for the BPX1xxx calls. The kernel reacts to the AMODE of the caller, which is saved in the linkage stack at the time of the PC to the kernel, and not to the stub. The kernel doesn't know whether a program is calling BPX1xxx or BPX4xxx. If you invoke a BPX4xxx stub in AMODE 31, the kernel will process the parameters of its BPX1xxx counterpart. If the parameters are different, an EFAULT exception will probably occur.

Support for multiple AMODES in a single process

At the assembler level, the kernel supports multiple AMODES in a single process, and switching back and forth between AMODES on a single thread. Different threads in a single process can have different AMODES.

Support for SRB callers

See Appendix J, "Callable services available to SRB mode routines," on page 1333 for the list of services that are supported for SRB mode callers running in AMODE 64.

Chapter 2. Callable services descriptions

This topic describes each of the callable services. These services are arranged in alphabetic order. A sample invocation of each service is in Appendix D, "Callable services examples—AMODE 31," on page 1123.

If you are unfamiliar with the conventions used to describe the system calls, refer to Chapter 1, "Invocation details for callable services," on page 1.

accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket

Function

The accept callable service allows a server to accept a connection request from a client. It extracts the first connection on the queue of pending connections, creates a new socket with the same properties as the specified socket, and allocates a new descriptor for that socket. If there are no connections pending, the service either blocks until a connection request is received, or fails with an EWOULDBLOCK, depending on whether the specified socket is marked as blocking or nonblocking.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1ACP):	31-bit task or SRB mode
AMODE (BPX4ACP):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ACP,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ACP with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the accept is to be done.

Sockaddr_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The field can be zero if no value is passed for Sockaddr. The size of the field should be less than 4096 bytes (4KB) in length.

Sockaddr
Supplied and returned parameter

Type: Structure

Length:
Length specified by Sockaddr_length

The name of a field that contains the socket address of the connecting client. The format of Sockaddr is determined by the domain in which the client resides. This field can be null if the caller is not interested in the client address. For more information about the format of this structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the accept service returns one of the following:

- A socket descriptor, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the accept service stores the return code. The accept service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The accept service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, or JRFileNotOpen.
ECONNABORTED	Software-caused connection termination.
EINTR	A signal interrupted the accept service before any connections were available. The following reason code can accompany the return code: JRSignalReceived.
EINVAL	The socket is not accepting connections. A listen must be done prior to the accept. The following reason code can accompany the return code: JRListenNotDone.
EIO	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
ENFILE	Too many files are open in the system. The following reason code can accompany the return code: JRMaxSockets.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EOPNOTSUPP	The referenced socket is not a type that supports the requested function.
EWOULDBLOCK	The socket file descriptor is marked nonblocking, and no connections are present to be accepted.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the accept service stores the reason code. The accept service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The Socket_descriptor that is passed refers to the socket that was created with the socket callable service, was bound to an address with the bind callable service, and that has issued a successful call to the listen callable service.

   Before calling the accept service, you can find out if the socket is pending a connection by doing a read select with the select callable service.
2. In order for Sockaddr to be returned for a UNIX domain socket, the client application doing the connect must bind a unique local name to the socket using the bind service before running the connect service.
3. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets" on page 31
- "bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor" on page 71

- "listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients" on page 330
- "select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues" on page 618
- "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777

Characteristics and restrictions

There are no restrictions on the use of the accept service.

Examples

For an example using this callable service, see "BPX1ACP (accept) example" on page 1125.

---

accept_and_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data

Function

The accept_and_recv callable service accepts the next connection on a socket and receives the first block of data. The new socket's descriptor, the peer's remote address, and the caller's local address are also returned. The service does not return until some data has arrived.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1ANR):	31-bit
AMODE (BPX4ANR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ANR,(Socket_desc,
              Accepted_socket,
              Remote_addr_len,
              Remote_addr,
              Local_addr_len
              Local_addr,
              Buffer_len,
              Buffer,
              Buffer_alet,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ANR with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Socket_desc
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket descriptor for which the accept_and_recv() is to be done. This is the server's "listen socket."

Accepted_socket
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one of the following on input:

- -1, indicating that the system is to assign a new descriptor to the accepted connection. The new descriptor is returned in this parameter. Note that a valid accepted socket descriptor is returned for partial success cases.
- If supported by the system, the value of a reusable socket descriptor with which the accepted connection is to be associated. Socket descriptors are reused after they have been used on a send_file that specified SF_REUSE. Reusable socket descriptors are created initially through an accept or an accept_and_recv. (See “send_file (BPX1SF, BPX4SF) — Send a file on a socket” on page 643.)

Remote_addr_len
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword (doubleword) that contains the length of Remote_addr. This field is updated with the length of the socket address that is returned in Remote_addr. If you do not want the Remote_addr, specify `0` for Remote_addr_len.

Remote_addr
Supplied and returned parameter

Type: Structure

Length:
Remote_addr_len

The name of an area that contains the sockaddr structure that is returned for the client that is connecting.

Local_addr_len
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword (doubleword) that contains the length of Local_addr. This field is updated with the length of the socket address that is returned in Local_addr. If you do not want the Local_addr, specify `0` for Local_addr_len.

`Local_addr`
Supplied and returned parameter

Type: Structure

Length:
Local_addr_len

The name of an area that contains the sockaddr structure that is returned for the server's port on which the connection arrives.

`Buffer_len`
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Buffer. If this value is zero, no receive is done, and the accept_and_recv request completes when a connection is available.

`Buffer`
Returned parameter

Type: Area

Length:
Buffer_len

The name of an area that contains the received data.

`Buffer_alet`
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the alet of the buffer. For buffers in the caller's primary address space, this value should be `0`.

`Return_value`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the accept_and_recv service returns one of the following:

- The number of bytes (zero or greater) that are received into the buffer, if the request is successful. Zero bytes can occur if the client closed the socket without sending any data, if a value of zero was specified for Buffer_len, or if no data was received within the active timeout interval. See “Usage notes” on page 22 for more information.

- -1 with a Return_code of EINTRNODATA, if the request was interrupted by a signal in the time between the arrival of the connection and the arrival of the first data. The connection is established, and Accepted_socket returns the new socket descriptor.
- -1 with a Return_Code of EWOULDBLOCK, if the request was interrupted because the SO_RCVTIMEO value expired before data was received. The connection is established, and Accepted_socket returns the new socket descriptor.
- -1 with any other Return_Code, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the accept_and_recv service stores the return code. The accept_and_recv service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The accept_and_recv service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	A file descriptor that was not valid was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	The connection was reset by a peer. The following reason code can accompany the return code: JRSockNotCon.
ECONNABORTED	The connection has been dropped.
EFAULT	An address that was passed cannot be accessed in the key of the caller.
EINTR	A signal interrupted the accept_and_recv service before a connection had arrived. The following reason code can accompany the return code: JRSignalReceived.
EINTRNODATA	A signal interrupted the accept_and_recv service after a connection had been established but before any data had arrived. This is a partial success, and the session has been established. A new socket descriptor is returned in Accepted_socket.
EIO	An I/O error occurred on one of the descriptors.
EINVAL	The socket is not accepting connections.
EISCONN	Accepted_socket is either bound or already connected.
EMFILE	OPEN_MAX descriptors are currently open in the calling process.
ENOBUFS	The service could not obtain a buffer. The following reason code can accompany the return code: JROutofSocketCells.
ENOMEM	The service could not obtain memory to complete the operation.
ENOREUSE	Socket descriptor reuse is not supported.
ENOSR	Insufficient STREAMS resources were available for the operation to complete.
ENOTSOCK	Socket_desc does not refer to a valid socket descriptor.The following reason code can accompany the return code: JRMustBeSocket.
EOPNOTSUPP	The socket type of the specified socket does not accept connections; or O_NONBLOCK is set for this socket. Nonblocking mode is not supported for this function.

Return_code	Explanation
EWOULDBLOCK	A new connection has been established, but the SO_RCVTIMEO timeout value was reached before data was available. This is a partial success, and the session has been established. A new socket descriptor is returned in Accepted_socket.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the accept_and_recv service stores the reason code. The accept_and_recv service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Nonblocking mode is not supported for this function. If O_NONBLOCK is set on the Socket_desc parameter, the function fails with an EOPNOTSUPP error.
2. If accept (BPX1ACP, BPX4ACP) and accept_and_recv (BPX1ANR, BPX4ANR) calls are both used on the same socket, it cannot be predicted which calls will be satisfied and in which order. Note also that a mixture of accept and accept_and_recv is discouraged as it may result in a reduced performance benefit that is achieved using accept_and_recv exclusively.
3. SO_SNDTIMEOUT and SO_RCVTIMEOUT values are propagated from the server to the new, accepted connections.
4. If SO_RCVTIMEOUT is specified on the server socket, the timeout for new connections is started when the connection is first established, rather than when the BPX1ANR service is issued. When the RCV_TIMEOUT occurs the service completes with a Return_code of -1 and Reason_Code of EWOULDBLOCK. A new or reused socket descriptor is returned in Accepted_Socket.
5. If SO_RCVTIMEOUT is not specified on the server socket, an internal timer is started when the new connection is first established. If data from the client is not received within the internal timeout interval, the BPX1ANR completes successfully with a Return_Value of zero. A new or reused socket descriptor is returned in Accepted_Socket.
6. The accept_and_recv function is designed to work with the send_file function to provide an efficient file transfer capability for a connection-oriented server with short connection times and high connection rates.
7. See Appendix J, “Callable services available to SRB mode routines,” on page 1333 for more information about programming considerations for SRB mode.

Related services

- “accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket” on page 15
- “recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer” on page 597

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1ANR (accept_and_recv) example" on page 1126.

access (BPX1ACC, BPX4ACC) — Determine if a file can be accessed

Function

The access callable service determines whether the caller can access a file. You identify the file by its pathname.

Requirements

Operation	Authorization
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ACC):	31-bit
AMODE (BPX4ACC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ACC,(Pathname_length,
              Pathname,
              Access_mode,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ACC with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file to be checked for accessibility. The length of this field is specified in Pathname_length.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory. The search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Access_mode
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates the accessibility to be tested. This field is mapped by the BPXYACC macro. The values for this field are:

ACC_F_OK
Test for file existence. This is the default value.

ACC_R_OK
Test for permission to read.

ACC_W_OK
Test for permission to write.

ACC_X_OK
Test for permission to execute or search.

ACCWAIT
If an asynchronous mount is in progress, wait for it to complete.

ACCDEVNO
Return the devno of the file in Return_value.

ACCEFFID
Use the effective ID rather than the real ID to check for permission.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the access service returns `0` if the request completes successfully (that is, the file exists or access is permitted), or -1 if the request is not successful, or the file cannot be accessed in the specified way.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the access service stores the return code. The access service returns Return_code only if Return_value is -1. See *z/OS UNIX*

System Services Messages and Codes for a complete list of possible return code values. The access service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have appropriate permissions to access the file in the ways specified by the Access_Mode parameter, or does not have search permission for some component of the Pathname prefix.
EINVAL	The Access_Mode parameter is incorrect. The following reason code unique to the access service can accompany the return code: JRInvalidAMODE.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or some component of the pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	No file named Pathname was found, or no Pathname was specified. The following reason code unique to the access service can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the Pathname prefix is not a directory.
EROFS	The Access_Mode parameter is testing for write access to a read-only file system. The following reason code unique to the access service can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the access service stores the reason code. The access service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Testing for file permission is based on the real user ID (UID) and real group ID (GID), unless the ACCEFFID bit has been set on. In that case, the effective ID is used for the test.
2. The caller can test for the existence of a file, or for access to the file, but not for both.
3. In testing for permission, the caller can test for any combination of read, write, and execute permission. If the caller is testing a combination of permissions, Return_value indicates failure if any one of the accesses is not permitted.
4. If the caller has appropriate privileges (see "Authorization" on page 8), the access test is successful even if the permission bits are off, except when testing for execute permission. When the caller tests for execute permission, at least one of the execute permission bits must be on for the test to be successful.
5. If the Access_mode parameter is zero, the service performs the existence test, ACC_F_OK.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the access service.

Examples

For an example using this callable service, see "BPX1ACC (access) example" on page 1124.

aio_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request

Function

The aio_suspend callable service suspends the calling thread until a specified asynchronous I/O event, specified timeout, or signal occurs.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ASP):	31-bit
AMODE (BPX4ASP):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ASP,(Aiocb_Ptr_List,
              Aiocb_Ptr_Count,
              Seconds,
              Nanoseconds,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ASP with the same parameters.

Parameters

Aiocb_Ptr_List
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of an Aiocb. Each Aiocb represents a previously submitted asynchronous I/O operation that the thread is to wait on for completion. The number of Aiocb pointers in the list is represented by the Aiocb_Ptr_Count parameter.

Aiocb_Ptr_Count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the Aiocb_Ptr_List. If you do not want to wait on any asynchronous I/O requests, define Aiocb_Ptr_Count as the name of a fullword that contains `0`.

Seconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that is the maximum number of seconds the calling program is willing to wait for one of specified asynchronous I/O events to occur.

Note:

1. Seconds can be any value greater than or equal to 0 and less than or equal to 4 294 967 295.
2. The Seconds and Nanoseconds values are combined to determine the timeout value. A combined value of zero indicates that the aio_suspend service will not wait at all. A value of AIO#NO_ASP_TIMEOUT (see "BPXYAIO — Map asyncio parameter list" on page 946) indicates that no timeout value is set.

Nanoseconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that is the number of nanoseconds to be added to the value specified by the Seconds parameter.

Note:

1. Nanoseconds can be any value greater than or equal to 0 and less than or equal to 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the aio_suspend service returns `0` if the request is successful, or -1 if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the aio_suspend service stores the return code. The aio_suspend service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The aio_suspend service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One or more of the specified parameters are not valid. The following reason codes unique to the aio_suspend service can accompany the return code: JrNanoSecondsTooBig, JrMaxAiocbECB.
EFAULT	One of the parameters specified contains the address of a storage area that is not accessible to the caller. The following reason codes unique to the aio_suspend service can accompany the return code: JrOK, JrBadAioEcb.
EINTR	The service was interrupted by a signal. One or more of the specified asynchronous I/O requests may have completed.
EAGAIN	The service timed out before any of the specified asynchronous I/O requests had completed.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the aio_suspend service stores the reason code. The aio_suspend service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The number of pointers to Aiocbs that use application-supplied ECB pointers for invocations of the aio_suspend service is limited to 254 when the Seconds and Nanoseconds parameters are both set to zero, and to 253 if either is nonzero. See “asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets” on page 31 for information on how to supply user-defined ECBs in the Aiocb data area.
2. If the Aiocbs are specified without application-supplied ECB pointers, there is no limit on the number of Aiocb pointers.

3. The Aiocbs that are represented by the list of Aiocb pointers must reside in the same storage key as the caller of the aio_suspend service. If the Aiocb Pointer List or any of the Aiocbs represented in the list are not accessible by the caller, an error of EFAULT may occur.
4. Aiocb pointers in the list with a value of zero are ignored.
5. A timeout value of zero (Seconds + Nanoseconds) means that the aio_suspend service does not wait at all, but checks for any completed asynchronous I/O requests. If it finds none, it returns with an error of EAGAIN; otherwise, it returns with a Return_value of `0`.
6. A passed timeout value of AIO#NO_ASP_TIMEOUT (see "BPXYAIO — Map asyncio parameter list" on page 946) means that no timeout value is set. The aio_suspend service waits until an asynchronous I/O request completes or until a signal is received.
7. The Aiocbs that are passed to the aio_suspend service must not be freed or reused by other threads in the process while this service is still in progress. The service may use the Aiocbs even after the asynchronous I/O completes. This restriction prevents multiple threads from doing aio_suspend()s on the same Aiocb at the same time. The results of modifying the Aiocb during an aio_suspend are unpredictable.
8. If the aio_suspend service is being called in AMODE 64 (BPX4ASP), the Aiocb_Ptr_List must contain 64-bit pointers only.

Related services

- "asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets" on page 31

Characteristics and restrictions

None.

Examples

For an example that uses this callable service, see "BPX1ASP (aio_suspend) example" on page 1126.

alarm (BPX1ALR, BPX4ALR) — Set an alarm

Function

The alarm call generates a **SIGALRM** signal after the number of seconds specified by the Seconds parameter have elapsed. The **SIGALRM** signal delivery is directed to the calling thread.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ALR):	31-bit
AMODE (BPX4ALR):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4ALR with the same parameters.

Format

```
CALL BPX1ALR,(Seconds,
              Return_value)
```

Parameters

Seconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of an unsigned fullword that contains the minimum number of seconds that are to pass between receipt of this request and generation of the **SIGALRM** signal. If the value is zero, any outstanding alarm request is canceled; no new alarm call time is set. Processor scheduling delays can cause the delivery of the **SIGALRM** signal to occur after the desired time.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of an unsigned fullword return value field. If there is a previous alarm request with time remaining, the alarm service returns a nonzero value that is the number of seconds until the previous request would have generated a **SIGALRM** signal. The return value is rounded to the nearest second, except when the time remaining is less than a half second. When the remaining time is less than a half second and greater than zero, Return_value is set to 1. If there is no previous alarm request with time remaining, Return value is set to zero.

Usage notes

1. The alarm service is always successful, and no return value is reserved to indicate an error.
2. An abnormal end is generated when failures are encountered that prevent the alarm service from completing successfully.
3. Alarm requests are not stacked; only one **SIGALRM** generation is scheduled in this manner. If **SIGALRM** was not generated, the call reschedules the time that **SIGALRM** is generated.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185

- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time" on page 771

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1ALR (alarm) example" on page 1125.

MVS-related information

Both the alarm service (BPX1ALR or BPX4ALR) and the sleep service (BPX1SLP or BPX4SLP) use the MVS STIMERM macro. It is possible that two STIMERM SET requests can be set by the alarm service and the sleep service. If the task invokes both STIMERM SET and the alarm service, the limit of concurrent STIMERM SET requests for a task can be exceeded, which results in an abnormal end.

asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets

Function

The asyncio callable service performs I/O operations against a socket asynchronously. It also provides synchronous operations for compatibility with the regular functions.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1AIO):	31-bit
AMODE (BPX4AIO):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1AIO,(Aiocb_length,
              Aiocb,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4AIO with the same parameters. All addresses in the Aiocb structure are doublewords.

Parameters

Aiocb_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Aiocb control block that is being passed in the next parameter. To determine the value of Aiocb_length, use the BPXYAIO macro (see "BPXYAIO — Map asyncio parameter list" on page 946).

Aiocb
Supplied parameter and returned parameter

Type: Structure

Length:
Specified by the Aiocb_length parameter.

The name of an Aiocb structure to be used to control this I/O operation. See usage note 3 on page 35 for information about on setting the Aiocb fields.

The BPXYAIO macro (see "BPXYAIO — Map asyncio parameter list" on page 946) maps the Aiocb.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the asyncio service returns the results of the request:

- `0` : indicates an asynchronous request has been successfully scheduled.

  When the I/O completes, the return value, return code, and reason code of the requested function are returned in the Aiocb, and the application is notified. See usage note 2 on page 34 for more information about asynchronous input/output.
- `-1`: indicates the system could not schedule the request, or the request itself failed immediately, for reasons such as parameter errors. Refer to Return_code and Reason_code for more details. There is no I/O completion notification.

  When the I/O function itself is rejected immediately, the return code and reason code are specific to that function. They are documented with the description of the regular version of the function.
- `+1` : indicates the operation successfully completed synchronously, meaning one of the following occurred:
  - AioOk2CompImd is specified, and the operation is able to be completed immediately.
  - AioSync is specified.
  - The function is Aio#Cancel, and AioCancelNoWait is not specified.

The system returns the return value, return code, and reason code of the requested function in the Aiocb. There is no I/O completion notification.

Note: These values are returned upon successful completion only. Immediate failures are always reported with a Return_value of -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the asyncio service stores the return code. The asyncio service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The asyncio service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	One of the following occurred: • The maximum number of queued signals was exceeded for this process (JrMaxQueuedSigs). This limit is specified with the MAXQUEUEDSIGS parameter of the BPXPRMxx parmlib member. • The maximum number of outstanding asynchronous requests that are permitted for this process was exceeded (JrMaxAsyncIO). The Async I/O maximum is twice the sum of MAXQUEUEDSIGS and the process's file limit. The file limit is taken from RLIMIT_NOFILE or the BPXPRMxx MAXFILEPROC parameter.
EALREADY	The Aiocb has already been canceled.
EBADF	The AioFd field does not contain a valid descriptor or the descriptor of a socket. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EFAULT	A supplied data area cannot be referenced.
EINVAL	A parameter is not valid. For example, AioBuffSize is negative, or AioCmd or AioNotifyType are unsupported values. The following reason codes can accompany the return code: JrAsyncBadAiocbLen, JrAsyncBadOffset, JrAsyncBadNotifyType, JrAsyncBadMsgHdrLen, JrAsyncBadSockAddr, JrAsyncBadCmd.
EIO	There was a network or transport failure.
EMVSINITIAL	Support for unauthorized user exits failed to initialize.
ENOSYS	The socket transport or physical file system does not support asynchronous I/O. Possible value: JrAsyncOpNotSupp.
EOPNOTSUP	If the return code is JrMsgFlagInvalidFlag, the TCPIP stack does not support the specified AioPosixFlags value. If the return code is JrAsyncAnr, a previous accept() operation was processed on this server socket, and because of this the use of asynchronous accept_and_receive is not supported.
EPERM	The caller is not authorized. Consult Reason_code to determine the exact reason the error occurred. If the return code is JrAsyncAuthErr, one of the following flags was set by an unauthorized caller: AioCallB4, AioUseUserKey, or AioCommBuff.

Refer also to the regular versions of the various functions for errors that might be detected before the system schedules the request,or if the request was processed asynchronously (AioSync was specified).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the asyncio service stores the reason code. The asyncio service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value.

The following reason codes may be reported: JrAsyncAuthErr, JrAsyncBadAiocbLen, JrAsyncBadCmd, JrAsyncBadMsgHdrLen, JrAsyncBadNotifyType, JrAsyncBadOffset, JrAsyncBadSigNo, JrAsyncBadSockAddr, JrAsyncExitModeTCB, JrAsyncOpNotSupp, JrAsyncSigKey0Err, JrReadUserStorageFailed, JrWriteUserStorageFailed, JrSyscallAbend, JrMsgInvalidFlag, JrAsyncANR.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. **Asynchronous input/output.** The asyncio service provides the capability to asynchronously perform those functions that are potentially blocking. These include the accept, connect, and receive and send types of functions.

   The general flow of an asynchronous request is as follows:

   a. All the parameters that are normally used on the regular version of the function are specified through the Aiocb structure. Parameters necessary to control the features of an asynchronous request are also specified here.
   b. After some preliminary checking, the system schedules the request and returns control to the caller. The AioRc field is set to EINPROGRESS. The application is free to continue with other work until it is notified that the I/O has completed. See usage note 3 on page 35 for more information about AioNotifyType.

      I/O completion usually occurs under the following conditions:

      - For reads, data is available or arrives from the network.
      - For writes, system buffers are available to hold the caller's data. This is the point at which the caller's buffers can be reused or freed. It does not imply anything about the progress of the actual data transmission.
      - For accept, a connection request is available or arrives.
      - For connect, this depends on the socket type and specific transport. It is usually the point at which you can start sending and receiving on the socket. This does not necessarily mean that the server has accepted this connection.
      - For accept_and_recv, connection request and the initial data from the client are available.

      The Aiocb and any areas pointed to from the Aiocb, such as a receive buffer, must remain valid until the I/O has completed.
   c. When it can complete the I/O, the system schedules an SRB to the caller's address space to perform the following sequence:

1) Optionally, call the exit program for preprocessing. See AioCallB4 in this topic for more information.
2) If the operation has been successful up to this point, transfer the I/O data from or to the caller's buffers.
3) Update the AioRv, AioRc, and AioRsn fields of the Aiocb with the status of the operation.
4) Perform the I/O completion notification as specified by AioNotifyType and other fields in the Aiocb:
   - Send a signal
   - Send a message
   - Call the exit program, passing the Aiocb
   - Post the ECB

   Only one type of notification is issued.

   If no notification is requested, the application can check the AioRc field periodically until it changes from EINPROGRESS.

The asyncio service supports AF_INET and AF_INET6 sockets; it cannot be used with AF_UNIX sockets.

3. **Aiocb control block.** The values set into this control block control the asyncio operation. The BPXYAIO macro (see "BPXYAIO — Map asyncio parameter list" on page 946) maps the Aiocb. The caller is responsible for setting the following fields.

   Field Description

   Function-specific fields

   AioCmd

   Specifies the function to be performed:
   - Aio#Accept for accept (BPX1ACP, BPX4ACP)
   - Aio#Anr for accept_and_recv (BPX1ANR, BPX4ANR)
   - Aio#Connect for connect (BPX1CON, BPX4CON)
   - Aio#Read for read (BPX1RED, BPX4RED)
   - Aio#Write for write (BPX1WRT, BPX4WRT)
   - Aio#ReadV for readv (BPX1RDV, BPX4RDV)
   - Aio#WriteV for writev (BPX1WRV, BPX4WRV)
   - Aio#Recv for recv (BPX1RCV, BPX4RCV)
   - Aio#Send for send (BPX1SND, BPX4SND)
   - Aio#RecvFrom for recvfrom (BPX1RFM, BPX4RFM)
   - Aio#SendTo for sendto (BPX1STO, BPX4STO)
   - Aio#RecvMsg for recvmsg (BPX1RMS, BPX4RMS)
   - Aio#SendMsg for sendmsg (BPX1SMS, BPX4SMS)
   - Aio#SelPoll for select (BPX1SEL, BPX4SEL) or poll (BPX1POL, BPX4POL).
   - Aio#Cancel to cancel a prior asyncio request. See the usage notes for more information about canceling operations.

   For details on their semantics and returned information, refer to the descriptions of the regular versions of these functions.

   AioFd The socket descriptor.

AioBuffPtr
The address of the buffer for the particular operation:

- For read/write, recv/send, recvfrom/sendto, and accept_and_recv — the address of the data buffer.
- For readv/writev — the address of the iov, BPXYIOV (see "BPXYIOV — Map the I/O vector structure" on page 986).
- For recvmsg/sendmsg — the address of the msghdr, BPXYMSGH (see "BPXYMSGH — Map the message header" on page 999).
- For selpoll — the address of a PollFD array, BPXYPOLL (see "BPXYPOLL — Map poll syscall parameters" on page 1014).
- For cancel — the address of the Aiocb to be canceled, or `0` to cancel all outstanding asyncio requests on the descriptor.

In 64-bit mode, AioBuffPtr is a doubleword pointer field, and is at a different offset within the Aiocb.

AioBuffSize
Specifies the size of whatever AioBuffPtr points to:

- For read/write, recv/send, recvfrom/sendto, and accept_and_recv — the length of the data buffer.
- For readv/writev — the number of elements in the iov array.
- For recvmsg/sendmsg — the length of the msghdr.
- For selpoll — the number of elements in the PollFD array.
- For cancel — this field is ignored.

AioBuffAlet
For read/write, recv/send, recvfrom/sendto, readv/writev, and accept_and_recv operations, this field contains the ALET of whatever is pointed to by AioBuffPtr. For all other operations, this field is ignored. See usage note 20 on page 49 for more information about using ALETs.

AioSockAddrPtr
Contains the address of a sockaddr structure area in the caller's primary address space. The sockaddr contains the address of the remote partner.

The sockaddr structure itself is supplied to the sendto and connect functions and returned by the recvfrom, accept_and_recv, and accept functions.

In 64-bit mode, AioSockAddrPtr is a doubleword pointer field, and is at a different offset within the Aiocb.

AioSockAddrLen
Contains the length of the sockaddr structure pointed to by AioSockAddrPtr.

This field is supplied to all functions that use AioSockAddrPtr. It is updated with the returned sockaddr length by the recvfrom, accept, and accept_and_recv functions. The following functions allow a value of 0 to be specified, indicating that no sockaddr structure is to be returned: Aio#Accept, Aio#ANR and Aio#Connect.

AioLocSockAddrPtr
Contains the address of a sockaddr structure area in the caller's primary address space. In 64-bit mode, AioLockSockAddrPtr is a

doubleword pointer field and is at a different offset within the Aiocb. The accept_and_recv function updates this field with the local sockaddr structure.

AioLocSockAddrLen
Contains the length of the sockaddr structure asyncio pointed to by AioLocSockAddrPtr. If you do not want the local sockaddr structure, specify 0 for AioLocSockAddrLen.

AioAnrSocket
Used by the accept_and_recv function. On input contains one of the following:

- -1, indicating that the system is to assign a new descriptor to the accepted connection. The new descriptor is returned in this parameter. Note that a valid accepted socket descriptor is returned for partial success cases as defined by the accept_and_recv service. (See "accept_and_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data" on page 18.)
- If supported by the system, the value of a reusable socket descriptor with which the accepted connection is to be associated. Socket descriptors are reused after they have been used on a send_file that specified SF_REUSE. Reusable socket descriptors are created initially through an accept, an accept_and_recv, or through the asyncio commands AIO#ACCEPT and AIO#ANR. (See "send_file (BPX1SF, BPX4SF) — Send a file on a socket" on page 643.)

AioMsgIovAlet
Specifies the ALET of the recvmsg/sendmsg msghdr's iov. See the usage notes for more information about using ALETs.

AioIovBufAlet
Specifies the ALET of all buffers pointed to from the iov that is used with the readv/writev and recvmsg/sendmsg functions. See the usage notes for more information about using ALETs.

AioPosixFlags
On input, contains the MSG_FLAGS value for the recv/send, recvfrom/sendto, recvmsg/sendmsg, and accept_and_recv functions (such as MSG_OOB and MSG_PEEK). For more information about the format of this field, see "BPXYMSGF — Map the message flags" on page 997

Several of the MSG_FLAGS are particularly useful with asyncio:

MSG_CONNTERM
The asyncio will not complete until the socket session ends. AioBuffSize must be set to 0 and the other MSG_FLAGS cannot be used.

- This flag provides a way to be notified at the time that a TCP socket session ends, independent of any actual receive-type operations outstanding on the socket.
- Any other outstanding requests are also be completed at connection termination.

MSG_WAITALL
The asyncio will not complete normally until the requested amount of data has arrived.

MSG_PEEK
Just the front of the data will be received, without being removed from the socket. For example, if your messages contain a fixed header that contains the length of the full message, just the header can be received asynchronously while the whole message is later received synchronously by the worker thread that will process the message. This also allows the asyncio to be done with a relatively small buffer and the worker thread will know how large a buffer needs to be allocated to hold the whole message.

Asynchronous feature fields

AioNotifyType
Specifies the type of asynchronous notification:

- Aio#Posix — Sets the return value, code, and reason fields of the Aiocb. Optionally, sends the signal specified in the AioSigEvent structure.
- Aio#MVS — Sets the return value, code, and reason fields of the Aiocb. Optionally, calls the exit program or posts the ECB.
- Aio#MsqQ (AIO_MSGQ) -- Sets the return value, code, and reason fields of the Aiocb, and sends the message specified in the AioMsgEvent structure.

Default: Aio#Posix.

AioExitPtr
Specifies the address of a program that the system is to call when the I/O completes. The Aiocb from the original request is passed to the exit. See 6 on page 42 for details. In 64-bit mode, AioExitPtr is a doubleword pointer field, and is at a different offset within the Aiocb.

AioExitData
An eight-byte area that is reserved for use by the application and the exit program. The system does not inspect or change this area.

AioExitModeTcb
For authorized TCB callers only, this specifies the mode (SRB or TCB) in which the exit program is to be called:

- `0` — on an SRB in the caller's address space. This is the default.
- `1` — on the caller's TCB.

For non-authorized callers the exit is always run on the caller's TCB. If the caller's TCB ends before the I/O completes, the system does not run the exit.

AioECBPtr
Specifies the address of an ECB in the caller's home address space that the system is to post when the I/O completes.

AioSigEvent
For Aio#Posix, this is a SigEvent structure that controls the signal generation. It contains the following fields:

- Sigev_Notify - Set to Sigev_Signal (0) to send the signal, or Sigev_none (1) to not send any signal. Default: Sigev_Signal.
- Sigev_Signo - Set to the signal number to be sent.

In 64-bit mode, SigEvent is a larger structure with several doubleword pointer fields; consequently, AioSigEvent is larger and at a different offset within the Aiocb.

AioMsgEvent
For Aio#MsgQ, this is a structure that specifies the I/O completion message. See "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399 for more information about messages and message queues. This structure overlays the AioSigEvent area and contains fields which correspond directly to the parameters in the msgsnd function:

- For AioMsgev_QID, specifies the message queue identifier that will receive the message.
- For AioMsgev_Addr/AioMsgev_Addr64, specifies the address of the message buffer. This buffer is defined by the Msgbuf/Msgbuf64 structure in BPXYMSG and contains the message type and message text. For C programs the name of this address field is the same for both 31-bit and 64-bit amodes, and the msgbuf structure is user-defined as described in the **msgsnd()** function in *z/OS XL C/C++ Runtime Library Reference*.
- For AioMsgev_Size, specifies the length of the message text. This value does not include the length of the message type field that precedes the text in the message buffer. The message text length is limited to 240 bytes.
- For AioMsgev_Flag, specifies the action to take if the message queue fills up. Specify 0 to wait, or IPC_NOWAIT, defined in BPXYIPCP, to not wait.

AioSiCode
A halfword signal code value that is to be associated with this I/O. This is only meaningful for I/O completion notification via signals. When the I/O completes and the completion signal is delivered through sigwaitinfo() or sigtimedwait(), the Si_code field of the resulting siginfo structure contains the value specified here. The halfword specified here occupies the lower half of the fullword Si_code. Normally the Si_code is set to SI_ASYNCIO# for asynchronous I/O completions, but if AioSiCode is not zero, that value will be used when the signal is sent. The meaning of this value is up to the application, and is not interpreted by the system.

AioTimeOut
A word that contains the timeout value for SelPoll. It can also be used to set a time limit for other synchronous operations (AioSync), such as Aio#Recv, Aio#Accept, and Aio#ANR. This value is expressed in milliseconds (1000ths of a second). If the operation times out, it fails with a return code of ETIMEDOUT.

- A value of Aio#Forever (0) (the default) means to wait forever.
- For SelPoll only, a value of Aio#NoWaiting (-1) means to not wait at all.

Note: The SO_RCVTIMEO socket option can be used to achieve a timeout function for asynchronous receive type operations. For more information see the **setsockopt()** function in *z/OS XL C/C++ Runtime Library Reference*.

AioOk2CompImd
Specifies that the system may complete an asynchronous request

immediately if it can do so without waiting, and without making any task switches. Otherwise, the system schedules the request for normal asynchronous processing.

If the request completes successfully and immediately, the Return_Value from asyncio is 1. The system returns the results of the function itself in the Aiocb. In this case there is no I/O completion notification.

On an asynchronous read (or on accept_and_recv) , if data has already arrived, this option avoids the extra overhead of scheduling the SRB and performing the notification. You must code the program to handle the received data in two places: after the call to asyncio and after the notification.

For the best performance, you should always set AioOk2CompImd and be able to handle I/O completion at the point of the call.

Default: Off, or it is not all right to complete immediately. The system issues the I/O completion notification.

AioSync
Specifies that the system is to run the request synchronously. The caller will wait or block, as necessary, subject to the current value of the nonblocking state of the socket.

If the request is successful, the Return_value from asyncio is 1. The system returns the results of the function itself in the Aiocb. There is no I/O completion notification.

This option provides equivalence with the regular versions of the functions. It is useful for synchronous operations that must be cancelable, for operations whose waits should be limited by AioTimeOut, and for calling the select() function with the much more efficient poll() interface.

Default: Off, or asynchronous.

AioTcbAffinity
Specifies that the I/O request should be canceled if the caller's TCB terminates. This field should be set if the Aiocb or buffer areas are in task-related storage and therefore will be freed when the task terminates.

Default: Off, or do not cancel the I/O for any type of I/O complete notification other than TCB Exit. For TCB Exits, the exit cannot be run after the TCB terminates, so the I/O will be canceled.

AioCancelNoWait
Specifies that a cancel operation is not to wait for all I/O completion notifications to finish before it returns to the caller. See usage note 13 on page 46 for more information about canceling operations.

Default: Off, or wait.

AioCancelNoNotify
Specifies that a cancel operation is to skip the I/O completion notifications that have not already been issued. See usage note 13 on page 46 for more information about canceling operations.

Default: Off, or issue the notifications.

AioCallB4
For authorized callers only. When on, this specifies that the exit program is to be called on the SRB for preprocessing before arrived data is transferred to the user's buffer. This provides a way to defer read buffer allocation until after the data has arrived. This call is in addition to the call that is made after the I/O has completed. See usage note 6 on page 42 for more information about I/O completion exits. Usage note 9 on page 44 has information about preprocessing.

Default: Off, or do not call the exit for preprocessing.

AioUseUserKey
For authorized callers only. When on, this specifies that the storage key in AioUserKey is to be used for all references to the functional parameters and data buffers. Only the Aiocb will be referred to with the caller's key.

Default: Off, or use the caller's key for all storage references.

AioUserKey
The key to be used for all references to the functional parameters and data buffers. This is only used when AioUseUserKey is on.

AioCommBuff
For authorized callers only. Specifies that the I/O buffers for this request reside in common storage, and may be addressed from any address space. Examples of common storage are ECSA and CADS data spaces. Having I/O buffers in common storage allows the system to copy data to them without having to have the caller's address space present, which improves the overall performance of asynchronous I/O. This flag is processed only for stream sockets; only for the operations of Aio#Read, Aio#ReadV, or Aio#Recv; and only with I/O complete notification via an SRB exit or an ECB. The Aiocb and the iov for Aio#ReadV do not have to be in common. It is critical that any outstanding I/O that has specified AioCommBuff be canceled explicitly before the descriptor is closed.

Default: Off, or the buffers reside in the caller's address space.

AioACEE
For SRB-mode callers only. Specifies the address of a security environment (ACEE) in the caller's home address space that is to be used for any multilevel security checks that may be done by TCP/IP during this I/O. This provides SRBs with a capability similar to the task-level security that is available with the TCBSENV field of a task's TCB. This field is ignored for TCB-mode callers, for whom the TCBSENV field will be used if it is nonzero, and if multilevel security checks are necessary. Before the specified ACEE can be freed or invalidated, you must ensure that this I/O has completed or been canceled; or that the socket is closed.

Default: Off, or use task-level security.

The following fields, which are set by the system, pass back the results of the requested function, as defined for the regular version of that function:

- AioRv — Return_Value
- AioRc — Return_Code
- AioRsn — Reason_Code

These fields are meaningful only after a successfully scheduled asynchronous request has completed, or when the asyncio service has a Return_value of 1.

The AioRc field is set to EINPROGRESS when a request is successfully scheduled. This value is changed to reflect the final results of the operation when the I/O completes.

The AioRc field is set to ETIMEDOUT for any function that times out because the AioTimeOut field is used.

The AioRc field is set to ECANCELLED for a request that is subsequently canceled. See usage note 13 on page 46 for more information about canceling operations.

Note: There are two ways to request that there be no notification: Assuming that the Aiocb has been initialized to zeros, you can set Sigev_Notify to Sigev_none, or you can set AioNotifyType to Aio#MVS. If no notification is used, the program can occasionally check the AioRc field until it is no longer equal to EINPROGRESS.

Callers of BPX1AIO (BPX4AIO) are considered authorized if the program is running in supervisor state or a system key, or if the program is APF-authorized.

4. **Unauthorized callers.** Unauthorized callers are restricted in the following ways:
   - Exits are run on the caller's TCB. If that TCB ends before the I/O has completed, the exit is not run.

     There are restrictions on the use of exits. See this topic for more information.
   - Authorized TSO commands are not permitted while any asynchronous I/O is outstanding in a TSO address space.
   - The AioCallB4 and AioUseUserKey options are not available.
   - The AioCancelNoWait option is not available if exits are pending for the TCB from which the cancel call is made.
5. **Using message queues for I/O completion notifications.** These messages can be received with the BPX1QRC **(msgrecv)** function. If the message buffer address is 0, a default message is sent. The default message type will be SIGIO#, which is the number 23 and is defined in BPXYSIGH. The default message text will be eight bytes long and contain the Aiocb address. For 31-bit callers the high word of the eight bytes is 0 and the aiocb address is in the lower word.

   The program should be designed to respond to the message queue in a timely manner so that waiting is avoided. These messages are sent from a system SRB, and waiting will tie up this critical resource and impact overall performance. Using small messages and a large queue can also help avoid waiting. If IPC_NOWAIT is specified and the queue fills up, or there is any other error in trying to send the message, the message will be lost. The system SRB will issue an EC6 abend with reason code FsAioMsgQError. The dump or logrec record produced will contain the return and reason codes from the internal call to BPX1QSN **msgsnd()** and potentially other diagnostic information about the problem.

   The length of the message text is limited to 240 bytes.
6. **I/O completion exit** You specify an exit program by setting AioNotifyType to Aio#MVS and putting the exit's address into the AioExitPtr field.

   The exit is called in the AMODE of the caller, AMODE 31 for BPX1AIO and AMODE 64 for BPX4AIO.

   The exit is passed the original Aiocb to correlate this completion with the original request. The Aiocb contains an application area (AioExitData) that can

be used to communicate with the exit. Note that since the application allocates the Aiocb in the first place, the areas before and after the control block are also available for related use. The Aiocb can be embedded in a larger application control block that can easily be reached from the Aiocb.

The exit is usually called to process received data, or free the storage that has been tied up with this request. At this point, the Aiocb contains the final return value, return code, and reason code of the function.

When a request has been canceled, the AioRv field is -1, and the AioRc field is set to ECANCELED. See usage note 13 on page 46 for more information about canceling operations.

7. **Authorized exits.** When the calling program is authorized, the exit is run on a system SRB; it is authorized and in key 0.

   Because the exit program is running on an SRB, the rules for SRB-mode callers must be followed if the exit makes any calls to z/OS UNIX functions. In particular, register 2 must be set before a call, to identify the process to which the exit belongs. See Appendix J, “Callable services available to SRB mode routines,” on page 1333 for details. Note, though, that the discussion on recovery for user SRBs is not relevant, because the exit is running on a z/OS UNIX SRB.

   Guideline: There is an upper limit to the number of SRBs that are allowed to run at the same time. If this limit is reached, other I/O completions remain queued until an SRB becomes available.

   The AioExitModeTCB flag can be used to run the exit on the original TCB rather than on the system SRB. In this case the restrictions that are listed for Unauthorized Exits apply. The exit is entered in the key of the first caller of BPX1AIO (BPX4AIO) in this process. This key cannot be changed.

8. **Unauthorized exits** When the calling program is not authorized, the exit is run on the caller's TCB in the caller's state and key. There are some restrictions:
   - A C program calling BPX1AIO (BPX4AIO) with an exit specified must be POSIX(OFF). POSIX signal handling and POSIX threading, as provided by Language Environment®, are not supported for any task in the program's process.
   - A program may not have invoked the BPX1MSS (BPX4MSS) service to register a signal interrupt routine.
   - The exit program is not in any way an extension of the main program. A C exit must establish its own C environment on entry. (This is significantly different from C signal handling.) In order for the I/O interrupt to be delivered, the thread that calls BPX1AIO (BPX4AIO) must remain dubbed.
   - All callers on all threads of a given process that are doing BPX1AIO (BPX4AIO) calls must be running with the same storage key.
   - The I/O interrupt targets the RB that made the original call to BPX1AIO (BPX4AIO) for a given thread. If the target RB is not the top RB, the interrupt is deferred until the target RB becomes the top RB.
   - A program must not have blocked the SIGIO signal, because the system uses this signal to schedule the exits.
   - If the exit program ends abnormally, the system cleans up that request and continues with other exits that are waiting to be run on that TCB. There is no dump, and the originator of that request is not notified of the problem. To have these abends percolated to the TCB, so that the TCB's mainline recovery will be run or the TCB will be terminated, set the ThliTcbExitPerc bit on before you issue the call to BPX1AIO (BPX4AIO).

The exit is free to do whatever is supported within the environment from which it is called. It may issue another call to asyncio.

Guideline: The exit should not issue any blocking calls, and should not enter into long delays. This ties up the system SRB on which the exit is running. Unauthorized exits are blocking the TCB that made the original request.

9. **Preprocessing exit.** Authorized callers can call the exit for preprocessing before the data is transferred, by using the AioCallB4 flag. This call is on the system SRB, and at a point before arrived data is moved into the application's receive buffer. It provides a way for the application to defer committing the necessary storage until just before it is actually needed.

   This deferred allocation applies to the receive buffers only. All other structures that are related to the call must be present, and the total requested data length must be specified correctly. For read, recv, recvfrom, and accept_and_recv, AioBuffPtr may be 0; but AioBuffSize must be set to the amount that is being requested. For readv, the following conditions must be met:

   - AioBuffPtr must point to an iov.
   - AioBuffSize must contain a nonzero number of iov entries.
   - The sum of the length fields in those entries must equal the amount being requested.
   - The iov buffer pointers may be zero.

   For recvmsg, there must be a valid msghdr structure with its associated sockaddr area, and an iov structure as described for readv. You can specify a simple one-element iov on the initial call to carry the length information, and replace this with another iov to be used for the data transfer.

   Note that you cannot use deferred allocation with AioOk2CompImd.

   When the preprocessing exit is called, the AioRv value usually contains the amount of data that is available, up to the requested amount. You can allocate smaller buffers when they will be sufficient. You should be prepared for cases in which AioRv is zero, when you should allocate buffers for the original requested amount. The actual amount of data that is received is returned in AioRv on the I/O completion call to the exit.

   The preprocessing call is only made when the operation has, up to this point, been successful. The preprocessing call is in addition to, not a replacement of, the call that is made after the I/O completes. The exit can use the AioExitData area to record its entry, and thus distinguish between the first and second calls during a successful operation. If AioRv is -1, this is the only call that is made to the exit.

   The preprocessing exit can change the following Aiocb fields to affect subsequent processing: AioBuffPtr, AioBuffAlet, AioExitPtr, and AioECBPtr. You may not change the function.

10. **Effect of process termination on exits.** When the caller's process terminates:

    - Exits for requests that have not yet completed are not called.
    - SRB exits that are about to be called or are already running may continue to completion, with some exceptions. If the process' address space abnormally ends, the exit is not able to finish. If the exit suspends, or calls a system service that suspends, it can be abnormally ended with a 47B abend code. If you have recovery, you should not take a dump or write an error record for 47B abends, but retry and return.

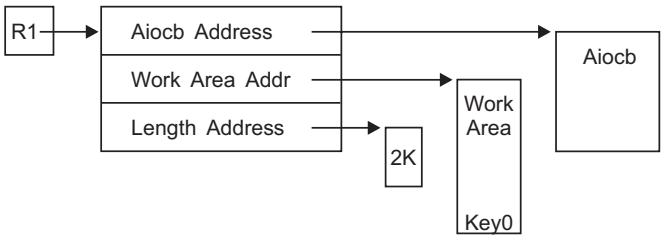
      SRB-mode routines should not call asyncio after it has entered process termination. See Appendix J, “Callable services available to SRB mode routines,” on page 1333.

11. **Environment at entry to the exit.** The exit program receives control in the asyncio caller's address space and in the following environment:

Operation	Environment
Authorization:	Same as the caller of asyncio
Dispatchable unit mode:	SRB or TCB
Cross memory mode:	PASN = HASN
AMODE:	Same as the caller of asyncio
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters are addressable in the primary address space in key 0 storage.

On entry, register 1 points to a parameter list that contains:

- The address of the Aiocb that was specified on the asyncio call now completing
- The address of a 2K work area for the exit's use
- The address of the length of the work area



For authorized exits on the SRB, the work area is in key 0. For unauthorized exits, or authorized exits running on the TCB, it is in the key of the caller of asyncio.

Registers at entry: The contents of the registers on entry to the exit are:

Register
Contents

0 Undefined

1 Parameter list address

2-12 Undefined

13 Address of a 136-byte save area. The first two words are reserved for standard save area conventions, and must not be used.

14 Return address

15 Entry address

AR 0-15
Undefined

12. **Environment at return from the exit.** On return from the exit, the entry environment must be restored.

Registers at exit: On return from the exit, the register contents must be:

Register
Contents

2-13 Restored from the entry values

0,1,14,15
Undefined

AR 0-15
Restored from the entry values

No return of status information is defined for the exit program.

13. **Canceling an operation.** You can cancel prior requests to asyncio with the Aio#Cancel function of asyncio.

    AioFd is set to the descriptor of the original operation, and:

    - To cancel a specific request, AioBuffPtr is set to the address of the Aiocb of the request that is to be canceled.

      Your program does not have to worry about timing or serialization with regard to the Aiocb that is being canceled. If that request has already finished, this attempt to cancel it will be ignored.
    - To cancel all outstanding asyncio requests on a specific descriptor, AioBuffPtr is set to zero.

      Note: SelPoll operations cannot be canceled this way. You cancel them by specifying the Aiocb of the SelPoll request.

    Cancel releases any blocked requests from their waits, and drives through the I/O completion notifications as it does for a failed request. The original AioRc is set to ECANCELED. Any requests that are not currently blocked are allowed to complete normally. If they attempt to enter a blocked wait they are failed with ECANCELED. Exits that are about to be called are still called normally, and those that are running are not interrupted. Aio#Cancel only "cancels" blocked requests, causing them to fail with ECANCELED.

    Synchronous requests (those with AioSync) are also broken out of their blocking waits. They are returned with an AioRc of ECANCELED.

    Usually the Aio#Cancel function waits until all I/O completion notifications have finished before asyncio returns to the caller. When asyncio returns, all exits have run, ECBs have been posted, and signals have been sent. The system is finished with the original request Aiocbs and buffers, and they can be freed by the application, subject to its own design. Note that for ECB and signal notifications, there is no coordination with the waiter or signal receiver, so there may still be application code running that is dealing with the request that has just been canceled. Because of timing, you can never tell which requests will finish normally and which requests will be canceled. You know, however, that the system is finished with the request and that any I/O complete notifications that are to be issued have been issued when asyncio returns to your program.

    In this case the successful Return_value is 1, and AioRv contains one of the following values:

    - AIO_CANCELED (1), if the requested operations were canceled.
    - AIO_NOTCANCELED (2), if at least one of the requested operations cannot be canceled because it is in progress.
    - AIO_ALLDONE (3), if all of the operations have already completed; that is, nothing was found to be canceled.

    If you do not want to wait for all I/O completion notifications, you can set the AioCancelNoWait flag. In this case the Return_value is 0 if any requests were found to be canceled. Your program must maintain its own tally of requests still outstanding if this is significant to it. If no requests were found to be canceled, the Return_value is 1, and AIO_ALLDONE is returned in AioRv.

Note: A program cannot wait for a cancel operation if it is running on the same TCB that pending exits would run on.

If you do not want the I/O completion notification to be issued, you can set the AioCancelNoNotify flag. If the request is still outstanding at the time of the cancel, the I/O completion notification is suppressed. This means that a specified exit program that has not already run will not be run. Setting this flag also stops the system from updating the Aiocb with the results of the operation, so that the AioRc field tends to remain with a value of EINPROGRESS. TCB exits that were scheduled to run at the time of an I/O completion but that have not yet run when the cancel is issued are skipped.

Note: The effectiveness of this flag is unpredictable, because the I/O completion notification may be in progress, or it may already have been made.

Canceling all requests on a given descriptor does not stop new requests from being made, or otherwise affect the descriptor. The program can start afresh or close the descriptor, depending on why it issued the cancel.

The asynchronous features of asyncio do not apply to Aio#Cancel; that is, you cannot specify a signal, an exit program, or an ECB. AioTimeOut does not apply to Aio#Cancel.

Cancel succeeds regardless of whether any outstanding requests have been found to cancel.

An individual request can be canceled only once. Subsequent attempts to explicitly cancel the same request fail with EALREADY.

Aio#Cancel cannot be used to cancel any operations other than those that are started with asyncio. You cannot cancel a read(), for example.

A cancel operation itself is not cancelable.

14. **Effect of close**

    Closing a descriptor deletes all requests that are still cancelable on that descriptor. I/O completion notifications are not issued for these requests. If you need exits to be run or ECBs to be posted, you must issue cancel for the descriptor before you issue close for the descriptor.

    In most cases, close() will flush out and wait for requests that are still in progress to be deleted. However, it cannot wait for requests that are already in the I/O complete exit programs; or that are just about to call these exits, post the I/O complete ECB, or send the I/O complete signal. Consequently, application code related to asyncio requests on the just-closed descriptor may still be in progress when the close() function returns.

    Descriptors that are part of an Aio#SelPoll request are removed from that operation. The request remains outstanding, and may complete as a result of activity on one of the other descriptors or when it times out. If all the descriptors for a particular SelPoll happen to be closed, no special action is taken; the request either times out or hangs forever.

15. **Multiple asynchronous operations on a single socket**

    Not all asynchronous operations support being called to start another I/O before the prior I/O has completed on that same socket. First of all, and most important, each call must have its own Aiocb and buffer or data areas; otherwise a serious and immediate error occurs for all of the operations, and the results are very unpredictable if two operations are using the same areas. In general, starting two or more asynchronous operations on a single socket is analogous to having two or more threads calling the regular synchronous versions of these operations at the same time, and the results are pretty much the same.

Aio#Accept and Aio#ANR may be called more than once. Each inbound connection request will complete a distinct call.

Aio#Connect: Stream (TCP) sockets may not be connected more than once. It does not make sense to connect UDP sockets several times simultaneously, because each connection replaces the previous one, and results will be unpredictable. The results of issuing requests that depend on the connection, such as Aio#Write, before the connection has completed are unpredictable.

Aio#Read, Aio#ReadV, Aio#Recv, Aio#RecvMsg, Aio#RecvFrom: For stream (TCP) sockets, the receive-type operations should not be called more than once before each call completes, as the results are unpredictable. The main reason for this is that the arrival of any data from the network can start the completion of one of these requests while the actual data movement occurs later, and so the data on the stream can be received by different threads out of order.

Aio#Write, Aio#WriteV, Aio#Send, Aio#SendTo, Aio#SendMsg: For stream (TCP) sockets, the send-type operations should not be called more than once before each call completes. Data may be transmitted on the network out of order, and, in general, results are unpredictable. For datagram (UDP) sockets, the send-type operations may be called more than once, because each distinct call defines a single datagram, and there is no implied order of arrival in UDP for these datagrams. Beware of sending too much data, though. If there is network congestion, or the receiver is slow, you can tie up a large amount of system storage with uncontrolled asynchronous sends, and eventually the BPX1AIO calls will start to fail with ENOBUFS.

Aio#SelPoll may be called more than once, but be aware that any one event will complete all the calls at the same time.

Aio#Cancel is not an asynchronous operation.

16. **Blocking and nonblocking**

    A socket must not be set to nonblocking state if you want I/O completion to wait for data.

    If the socket is in nonblocking state and there is no data available, either the asyncio request has its I/O completion driven very quickly with an AioRc of EWOULDBLOCK, or the asyncio call fails with a Return_code of EWOULDBLOCK.

    Note that Aio#ANR does not support nonblocking I/O.

17. **Signal considerations**

    Signals do not interrupt asynchronous operations unless they lead to the termination of the caller's process.

18. **Asynchronous select and poll**

    The Aio#SelPoll command can be used for either an asynchronous select() function or an asynchronous poll() function. The poll() interface structure is used in both cases. AioBuffPtr contains the address of a PollFD array, from BPXYPOLL, and AioBuffSize contains the number of elements in the array.

    - For the poll function, the PollFD structure is used in the same way as for poll (BPX1POL, BPX4POL).
    - For the select function, the SelFlags member of the Sel structure from BPXYSEL is mapped over the PollEvents and PollRevents members of the PollFD structure for input and output, respectively. The select event bits have the same meaning as they do for select (BPX1SEL, BPX4SEL), but they are input and output with the technique used by poll events. The triple bit map scheme of select (BPX1SEL, BPX4SEL) is not used.

These bits occupy different bytes in the PollEvents field, and the intended function is determined according to which bits are used. If no bits are set, the operation is considered to be a poll for nothing, rather than a select for nothing.

The entire PollFD array must consistently use only one type of bit. You cannot use select and poll bits for the same file descriptor, nor can you use select bits for one descriptor and poll bits for another. For the sake of performance, the input array is not checked to enforce this rule, and results are unpredictable if the rule is broken. The first occurrence of select bits that are turned on causes the operation to be a select() rather than a poll().

The AioTimeOut field can be used to specify a timeout value for the operation.

Aio#SelPoll can only be used with socket descriptors.

Aio#SelPoll operations cannot be canceled by descriptor; the specific aiocb must be canceled.

Negative descriptors in the PollFd array are ignored, as documented for poll(). Otherwise, the first bad descriptor causes the whole operation to fail at that point in the array. This is a little different from the behavior of poll().

Guideline: For performance reasons, do not use asynchronous select or poll if you can use any other asynchronous operation on each descriptor.

For example, doing Aio#Read for each of five sockets is much faster and more efficient than doing one Aio#SelPoll for that same set of sockets. This is because when an Aio#Read completes for one socket you have the data; the other sockets are unaffected and remain ready for inbound data. On the next Aio#Read only that one socket has to be readied again. When Aio#SelPoll completes for any socket, all the others are taken out of their prepared state. You still have to issue another call to actually get the data. On the next Aio#SelPoll all the sockets must be "put back" into their prepared state again.

19. **asyncio vs. synchronous (regular) select**

    Asynchronous I/O is similar to the select() and poll() functions in that you can wait for data from many different descriptors at the same time. Asynchronous I/O, though, is much faster and much more efficient for large numbers of descriptors. With the asyncio service you also have control over when you wait for the next event.

20. **Using ALETS**

    ALETS are generally usable only for synchronous requests (AioSync), with the exception of recvmsg/sendmsg. A preprocessing exit (AioCallB4) could update the SRB it is running on with an ALET for a data space, but this would add too many instructions to the operation to be practical for the general read or write. You could, however, consider using a Common Area Data Space (CADS).

Related services

- “accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket” on page 15
- “connect (BPX1CON, BPX4CON) — Establish a connection between two sockets” on page 121
- “poll (BPX1POL, BPX4POL) — Monitor activity on file descriptors and message queues” on page 488
- “read (BPX1RED, BPX4RED) — Read from a file or socket” on page 572

- "readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers" on page 590
- "recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer" on page 597
- "recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer" on page 600
- "recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers" on page 604
- "send (BPX1SND, BPX4SND) — Send data on a socket" on page 640
- "select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues" on page 618
- "sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket" on page 648
- "sendto (BPX1STO, BPX4STO) — Send data on a socket" on page 652
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928
- "writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers" on page 933
- "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399

Characteristics and restrictions

The asyncio service supports AF_INET and AF_INET6 sockets; it cannot be used with AF_UNIX sockets.

Examples

For an example that uses this callable service, see "BPX1AIO (asyncio) example" on page 1125.

attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program

Function

The attach_exec callable service attaches a task to run a z/OS UNIX executable program in a newly created child process of the caller. The child process that is created has the same attributes that a child process would have if it were created by the fork service and followed immediately by a call to the exec service. The new process is created in the same address space as the caller, and is a subtask of the caller's task.

Requirements

Operation	**Environment**
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ATX):	31-bit
AMODE (BPX4ATX):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ATX,(Pathname_length,
              Pathname,
              Argument_count,
              Argument_length_list,
              Argument_list,
              Environment_count,
              Environment_data_length,
              Environment_data_list,
              Exit_routine_address,
              Exit_parameter_list_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ATX with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the path name of the file. The length can be up to 1023 bytes long.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the fully qualified path name of the file to be run. Each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

Argument_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument_length_list and the Argument_list. If the program needs no arguments, define Argument_count as the name of a fullword that contains `0`.

Argument_length_list
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an argument to be passed to the specified program. If the program needs no arguments, define Argument_length_list as the name of a fullword (doubleword) that contains `0`.

`Argument_list`
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an argument to be passed to the specified program. Each argument is of the length that is specified by the corresponding element in the Argument_length_list. If the program needs no arguments, define Argument_list as the name of a fullword (doubleword) that contains `0`.

`Environment_count`
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for the Environment_data_length and the Environment_data. If the program needs no environment data, define Environment_count as the name of a fullword that contains `0`.

`Environment_data_length`
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an environment variable to be passed to the specified program. If the program does not use environment variables, define Environment_data_length as the name of a fullword (doubleword) containing `0`.

`Environment_data_list`
Supplied parameter

Type: Structure

Length:
Variable, specified by the Environment_data_length

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length that is specified by the corresponding element in the Environment_data_length. If the program does not use environment variables, define Environment_data_list as the name of a fullword (doubleword) that contains `0`. If the target executable file is a

program enabled by Language Environment, the environment variables that are supplied to this service must include the null terminator as part of the data string and length.

Exit_routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define Exit_routine_address as the name of a fullword (doubleword) that contains `0`. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

Exit_parameter_list_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define Exit_parameter_list_address as the name of a fullword (doubleword) that contains `0`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_exec service returns the process ID of the created child process, if it is successful. If it is not successful, the service returns -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_exec service stores the return code. The attach_exec service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The attach_exec service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The resources required for another process to be created are not available now; or you have already reached the maximum number of processes you are allowed to run. The following reason codes can accompany the return code: JRMaxChild, JRMaxProc, JRMaxUIDs.

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory that is named in the Pathname parameter; it may lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type. The following reason code can accompany the return code: JRExecNotRegFile.
EFAULT	A bad address was received as an argument of the call, or the user exit program checked. The following reason codes can accompany the return code: JRExecParmErr and JRExitRtnError.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMVSSAF2ERR	The executable file is a set_user_ID or set_group_ID file, and the file, owner's UID, or GID is not defined to RACF.
ENAMETOOLONG	Pathname is longer than 1023 characters, or some component of the Pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	No Pathname was specified, or one or more of the components of the specified Pathname were not found. The following reason codes can accompany the return code: JRExecNmLenZero and JRQuiescing.
ENOEXEC	The specified file has execute permission, but it is not in the proper format to be a process image. Reason_code contains the loader reason code for the error.
ENOMEM	The new process requires more memory than is permitted by the hardware or the operating system. The following reason codes can accompany the return code: JRExecFileTooBig and JRNoSpace.
ENOTDIR	A directory component of Pathname is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_exec service stores the reason code. The attach_exec service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the majority of reason codes, see *z/OS UNIX System Services Messages and Codes*. For the ENOEXEC Return_code, Reason_code contains the loader reason code for the error:

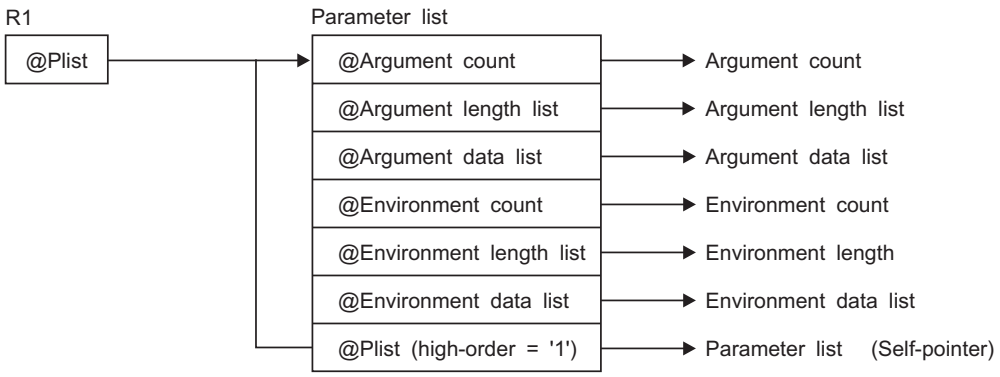
Reason code	Explanation
X'xxxx0C27'	The target file is not in the correct format to be an executable file.
X'xxxx0C31'	The target file is built at a level that is higher than that supported by the running system.

Usage notes

1. The new process (called the *child process*) has similarities to the process that calls attach_exec (called the *parent process*), except for the following:

- The child process has a unique process ID (PID) that does not match any active process group ID.
- The child has a different parent process ID (namely, the process ID of the process that called attach_exec).
- The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.
- If a file has its FCTLCLOFORK or FTCLOEXEC flag set on, it is not inherited by the child process. These flags are set with the fcntl service. For more information, see the fcntl service "Parameters" on page 175.
- Directories opened via a call to the opendir (BPX1OPD, BPX4OPD) service in the parent process are not inherited by the child process.
- The process and system utilization times for the child are set to zero.
- Any file locks previously set by the parent are not inherited by the child.
- The child process has no interval timers (for example, alarms) set. This is similar to a call to the alarm service with Wait_time specified as zero.
- The child process has no pending signals.
- The child process does not get a copy of the parent's storage, as it would if it were created via a call to the fork service.
- The child process can address a shared memory segment only while the parent process maintains its attachment.
- The semaphore adjustment values (semadj) in the child process will be zero.
- The child process created by this service is terminated when its parent terminates.

2. The executable file to be run receives control with the following attributes:
   - Problem program state
   - TCB key of caller
   - AMODE=31(64), taken from the executable
   - Primary ASC mode
3. The information that the service passes to the executable file that is to be run is a parameter list pointed to by register 1. The parameter list consists of the parameter addresses listed in this topic. In the last parameter address, the high-order bit is 1.



For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always *n* parameters, passed with no end-of-parameter-list indicator.

The last parameter that attach_exec passed to the executable file identifies the caller of the file as the attach_exec or exec service.

4. The user exit receives control with the following attributes:
   - Problem program state
   - PSW key of caller
   - AMODE=31(64), same as the invoker of BPX1ATX (BPX4ATX)
   - Primary ASC mode

   See "Characteristics and restrictions" on page 58 for more information about the execution of the user exit.
5. The register usage on entry to the user exit in AMODE 31 is:
   - R0: Undefined.
   - R1: Address of the user exit parameter list, as specified by the caller of the attach_exec service.
   - R2–R12: Undefined.
   - R13: Address of a 96-byte work area in the same key as the caller of the attach_exec service.
   - R14: The return address from the user exit to the attach_exec service. This address must be preserved by the user exit.
   - R15: Address of the user exit.
6. The register usage on entry to the user exit in AMODE 64 is:
   - R0: Undefined.
   - R1: 64-bit address of the user exit parameter list, as specified by the caller of the attach_exec service.
   - R2–R12: Undefined.
   - R13: 64-bit address of a 96-byte work area in the same key as the caller of the attach_exec service. Bits 0–32 of this address are `0`.
   - R14: The 64-bit return address from the user exit to the attach_exec service. This address must be preserved by the user exit. Bits 0–32 of this address are `0`.
   - R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.
7. To support the creation and propagation of a STEPLIB environment to the new process image, attach_exec allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable, and the actions taken for each value:
   a. STEPLIB=NONE. No STEPLIB DD is to be created for the new process image.
   b. STEPLIB=CURRENT. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the attach_exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
   c. STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image. The actual name of the DD is not STEPLIB, but a system-generated name that has the same

effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. Those data sets found to be in violation of this standard are ignored. If the data sets follow the standard, but:

- The caller does not have the proper security access to a data set, or
- A data set is uncataloged or is not in load library format

the data set is ignored. Because the data sets in error are ignored, the executable file may run without the proper STEPLIB environment. If a data set is in error due to improper security access, a X'913' abend is generated. The dump for this abend can be suppressed by your installation.

If the STEPLIB environment variable is not specified, the default behavior of the attach_exec service is the same as if STEPLIB=CURRENT were specified.

For information about STEPLIB performance considerations, see the section on tuning performance in *z/OS UNIX System Services Planning*.

8. A prior loaded copy of an HFS program is reused by this service under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
   - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
   - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the HFS program found is in storage modifiable by the caller, the prior copy is not reused.
9. If the specified file name resolves to an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is used only if the name is eight characters or less; otherwise the caller receives an error from the attach_exec service. For a sticky bit program, the specified file name is used if it is eight characters or less. Otherwise, the program is loaded from the z/OS UNIX file system. If the attach_exec caller is running APF authorized and the specified sticky bit file or link resolves to an MVS program link-edited AC=1 located in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation. For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file. For an external link, it must be installed with an owning UID of 0. Also, a file with the APF extended attribute is not allowed if found in a file system mounted as NOSETUID. If the specified file name represents a symbolic link to a sticky bit file that has the set-user-id attribute, the symbolic link must have an owning UID of 0 or an owning UID equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning UID of 0 or an owning GID equal to that of the sticky bit file. A file or link found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this set up will cause the task attached to run the MVS program to end abnormally with a EC6-xxxxC04A abend when the MVS program is invoked via the attach_exec service.
10. If the calling parent task is in a WLM enclave, the child task is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.

11. If the target executable program is a Language Environment-enabled program, the environment variables supplied to the service must include the null terminator as part of the string and length.
12. If the _BPX_PTRACE_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs loaded into storage during the execution of the target program are also loaded into user-modifiable storage, with the exception of modules loaded from the LPA.
13. If the BPXK_SIGDANGER environment variable is set to YES, the process will receive a SIGDANGER signal rather than a SIGTERM signal when an OMVS shutdown is initiated.
14. A thread that issues an attach_exec or attach_execmvs may receive an A03 abend if any attached children are still running. To avoid the A03 abend, the thread that issued the attach can use waitpid (BPX1WAT) to determine when the attached process has completed, then call mvsprocclp (BPX1MPC) to allow time for a full MVS subtask termination to occur after the child process has terminated.
15. If the caller specifies _BPXK_DISABLE_SHLIB=YES then future loadhfs() and loadhfs_extended() system calls will ignore the st_sharelib attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information, see Commonly used environment variables in *z/OS UNIX System Services Planning*

Related services

- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program" on page 59
- "spawn (BPX1SPN, BPX4SPN) — Spawn a process" on page 780

Characteristics and restrictions

The user exit is given control in the newly created child process on the attached task before the invocation of the specified program. The user exit should not attempt to use any kernel services. Signals cannot be delivered while in the user exit because the attach_exec service is still in progress and signal delivery is inhibited.

The setuid, setgid, setegid and seteuid services, if invoked from a process created by this service, affect the calling process and any other processes that exist in the address space. In a multiprocessing environment, however, when a process created by this service attempts to change the security environment, the request is rejected.

If exec or execmvs is invoked from a process that was created via the attach_exec service, the initial thread of the process and all of its subtasks are terminated, and a new task is attached to run the specified program. This does not result in the ending of any other tasks in the calling job step, nor does it end other processes in the address space. Because of this behavior, only unauthorized, non-privileged programs are supported on the invocation of exec and execmvs.

Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:

- Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST_PROGCTL attribute is ON for the z/OS UNIX program file). Running a z/OS UNIX program with the ST_PROGCTL attribute set to OFF prevents future invocations of authorized programs like Program Access to Data Sets (PADS) programs. These are programs given special authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.
- System key, supervisor state, and APF-authorized callers should not attempt to execute a program from z/OS UNIX, unless the executable file has the APF attribute turned on.
- set-user-ID programs can only be called by processes running with the same effective user ID as the user ID of the executable file.
- set-group-ID programs can only be called by processes that are running with the same effective group ID as the group ID of the executable file.

Sticky bit programs that are link-edited as APF-authorized may be called only by callers that run APF-authorized.

The newly attached task created for the child process does not share user storage subpools 0-127 with the caller.

Examples

For an example using this callable service, see "BPX1ATX (attach_exec) example" on page 1127.

MVS-related information

Because the newly created child process runs on a subtask in the same address space as the caller, it has access to the same MVS environment as the caller. This includes the same allocation (DDs) and storage environment. Because of this, programs that run on each of these tasks should be careful not to interfere with other programs running in the same environment. Although the child subtask has access to the same storage as the calling task, it does not share any user subpools with the calling task; thus it cannot free any user storage that is obtained by the calling task.

attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program

Function

The attach_execmvs service attaches a task to run an MVS executable program in a newly created child process of the caller. The new process is run in a subtask in the same address space.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN

Operation	Environment
AMODE (BPX1ATM):	31-bit
AMODE (BPX4ATM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ATM,(Program_name_length,
              Program_name,
              Argument_length,
              Argument,
              Exit_routine_address,
              Exit_parameter_list_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ATM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Program_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length, in the range of 1 to 8 bytes, of the name of the MVS program.

Program_name
Supplied parameter

Type: Character string

Character set:
Conforms to naming conventions for members of MVS PDSs

Length:
Specified by the Program_name_length parameter

The name of a field that contains the name of the MVS program to be run. The MVS program name must conform to the naming conventions for members of MVS partitioned data sets (PDSs). The program name is from 1 to 8 characters long; the program name is the member name without any qualifiers. The specified Program_name must be in uppercase.

Argument_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the argument that is to be passed to the program. The argument can be from 0 to 4096 bytes long.

Argument
Supplied parameter

Type: Character string

Length:
Specified by the Argument_length parameter

The name of a field of length Argument_length that contains the argument that is to be passed to the MVS program.

Exit_routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define Exit_routine_address as the name of a fullword (doubleword) that contains `0`. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

Exit_parameter_list_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define Exit_parameter_list_address as the name of a fullword (doubleword) containing `0`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_execmvs service returns the process ID of the created child process, if it is successful. If it is not successful, the service returns -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_execmvs service stores the return code. The attach_execmvs service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The attach_execmvs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
E2BIG	The number of bytes used by the new process image's argument list is greater than the system-imposed limit of 4096 bytes. The following reason code can accompany the return code: JRMVSArgTooBig.
EFAULT	The user exit program checked. The following reason code can accompany the return code: JRExitRtnError.
ENAMETOOLONG	The specified MVS program name is too long. The length specified by Program_name_length is longer than 8 bytes.
ENOENT	The specified MVS program was not found in the link pack area (LPA) or in a link list data set (LNKLST); or the program name argument points to an empty string. The following reason code can accompany the return code: JRExecNmLenZero.
ENOMEM	The new process requires more memory than is permitted by the hardware or the operating system. The following reason codes can accompany the return code: JRExecFileTooBig and JRNoSpace.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the attach_execmvs service stores the reason code. The attach_execmvs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The result of a call to the attach_execmvs service is that a subtask is attached to the calling task to run the specified program in a newly created child process. The newly created subtask becomes the initial thread of the newly created child process.
2. The new child process has similarities to the process that calls attach_execmvs (called the *parent process*), except for the following:
   - The child process has a unique process ID (PID) that does not match any active process group ID.
   - The child has a different parent process ID (namely, the process ID of the process that called attach_execmvs).
   - The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.

     Tip: BPX1ATM only propagates file descriptors, not streams. If the target program is MVS-style, there is no consideration for fd0, fd1, and fd2 being used as the standard streams when the child program starts.
   - If a z/OS UNIX file has its FCTLCLOFORK or FCTLCLOEXEC flag set on, it is not inherited by the child process. These flags are set with the fcntl service. For more information, see the fcntl service “Parameters” on page 175.
   - The child has its own copy of the parent's open directory streams. Each open directory stream in the child can share directory stream positioning with the corresponding directory stream of the parent.
   - The process and system utilization times for the child are set to zero.

- Any file locks previously set by the parent are not inherited by the child.
- The child process has no interval timers set. This is similar to the results of a call to the alarm service with Wait_time specified as zero.
- The child has no pending signals.
- The child process does not get a copy of the parent's storage, as it would if it were created via a call to the fork service.
- The child process created by this service is terminated when its parent terminates.

3. The input passed to the MVS executable program by the service is consistent with the input passed to MVS programs. On input, the MVS program receives a single-entry parameter list pointed to by register 1. The high-order bit of the sole parameter entry is set to 1.

   The sole parameter entry is the address of a 2-byte length field followed by an argument string. The length field describes the length of the data that follows it. If a null argument and argument length are specified in the call, the length field specifies `0` bytes on input to the executable program.
4. The MVS program to be run receives control with the following attributes:
   - Problem program state
   - TCB key of caller
   - AMODE=31(64), taken from the executable
   - Primary ASC mode

   The specified program can be located in the link pack area (LPA), in a link list data set, job library, step library, or task library. The program search order that is followed is identical to that of the MVS Attach service when the EP parameter is specified.
5. The user exit receives control with the following attributes:
   - Problem program state
   - PSW key of caller
   - AMODE=31(64)
   - Primary ASC mode
6. The register usage on entry to the user exit in AMODE 31 is:
   - R0: Undefined.
   - R1: Address of the user exit parameter list as specified by the caller of the exec service.
   - R2–R12: Undefined
   - R13: Address of a 96-byte work area in the same key as the caller of the exec service.
   - R14: The return address from the user exit to the exec service. This address *must* be preserved by the user exit.
   - R15: Address of the user exit.
7. The register usage on entry to the user exit in AMODE 64 is:
   - R0: Undefined.
   - R1: 64-bit address of the user exit parameter list as specified by the caller of the exec service.
   - R2–R12: Undefined
   - R13: 64-bit address of a 96-byte work area in the same key as the caller of the exec service. Bits 0–32 of this address are `0`.

- R14: The 64-bit return address from the user exit to the attach_exec service. This address must be preserved by the user exit. Bits 0–32 of this address are 0.
- R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.

8. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the attach_execmvs service are propagated to the new process image. This causes the program that is invoked to run with exactly the same MVS program search order as its invoker.
9. To support the creation and propagation of a STEPLIB environment to the new process image, attach_execmvs allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable and the actions taken for each value:
   a. STEPLIB=NONE. No Steplib DD is to be created for the new process image.
   b. STEPLIB=CURRENT. The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
   c. STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.

      Note: The actual name of the DD is not STEPLIB, but a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. Data sets found to be in violation of this standard are ignored. If the data sets do follow the standard, but:

      - The caller does not have the proper security access to a data set
      - A data set is uncataloged, or is not in load library format

      then the data set is ignored. Because the data sets in error are ignored, the executable file may run without the proper STEPLIB environment. If a data set is in error due to improper security access, a X'913' abend is generated. The dump for this abend can be suppressed by your installation.

   If the STEPLIB environment variable is not specified, the default behavior of the attach_execmvs service is the same as if STEPLIB=CURRENT were specified.

   If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For detailed information about the sanction list, see *z/OS UNIX System Services Planning*. For information about STEPLIB performance considerations, see the section on tuning performance in *z/OS UNIX System Services Planning*.

10. If the calling parent task is in a WLM enclave, the child task is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.
11. A thread that issues an attach_exec or attach_execmvs may receive an A03 abend if any attached children are still running. To avoid the A03 abend, the thread that issued the attach can use waitpid (BPX1WAT) to determine when the attached process has completed, then call mvsprocclp (BPX1MPC) to allow time for a full MVS subtask termination to occur after the child process has terminated.

Related services

- “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185
- “execmvs (BPX1EXM, BPX4EXM) — Run an MVS program” on page 144
- “attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program” on page 50
- “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 780

Characteristics and restrictions

The user exit is given control in the newly created child process on the attached task before the invocation of the specified program. This exit can be used by the caller to alter the environment of the child process, similarly to the way in which a program would alter the child's environment after a call to fork, but before the call to execmvs. The user exit should not attempt to use any kernel services from the exit. Signals cannot be delivered while in the user exit, because the attach_execmvs service is still in progress and signal delivery is inhibited.

The setuid, setgid, setegid and seteuid services, if invoked from a process created by this service, affect the calling process and any other processes that exist in the address space.

If exec or execmvs is invoked from a process that was created via the attach_execmvs service, the initial thread task of the process and all of its subtasks are terminated, and a new task is attached to run the specified program. The initial thread task in such a process is the task that was created as a result of the call to the attach_execmvs service. The call to exec or execmvs does not result in the ending of any other tasks in the calling jobstep, nor does it end other processes in the address space. Because of this behavior, only unauthorized, non-privileged programs are supported on the invocation of exec and execmvs.

APF-authorized programs can be invoked from this service if the caller is APF authorized.

Examples

For an example using this callable service, see “BPX1ATM (attach_execmvs) example” on page 1127.

MVS-related information

Because the newly created child process runs on a subtask in the same address space as the caller, it has access to the same MVS environment as the caller. This includes the same allocation (DDs) and storage environment. Because of this, programs that run on each of these tasks should be careful not to interfere with

other programs running in the same environment. Although the child subtask has access to the same storage as the calling task, it does not share any user subpools with the calling task. For this reason, it cannot free user storage obtained by the calling task.

auth_check_resource_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF-protected resource

Function

The auth_check_resource_np service checks the authority of a RACF-defined user to access a RACF-defined resource. Resources in the DATASET class cannot be checked. The authorization required to invoke this service is one of the following:

- Read access to the BPX.SERVER resource in the FACILITY class
- A UID of 0 when the BPX.SERVER resource is not defined in the FACILITY class

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ACK):	31-bit
AMODE (BPX4ACK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ACK,(Cell_UUID,
              Principal_UUID,
              Userid_Length,
              Userid,
              Security_Class_Length,
              Security_Class,
              Entity_Name_Length,
              Entity_Name,
              Access_Type,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ACK with the same parameters.

Parameters

Cell_UUID
Supplied parameter

Type: Character string

Length:
36 bytes

The name of a 36-byte area that contains the cell DCE UUID. If the cell DCE UUID is not specified, the first byte of this 36-byte area must contain NUL (X'00').

Principal_UUID
Supplied parameter

Type: Character string

Length:
36 bytes

The name of a 36-byte area that contains the principal DCE UUID. If the principal DCE UUID is not specified, the first byte of this 36-byte area must contain NUL (X'00').

Userid_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Userid parameter. Userid_Length can be in the range of 0 to 8. If a user ID is not required, specify the name of a fullword that contains zero.

Userid
Supplied parameter

Type: Character string

Character set:
The XPG4 portable character set, which includes upper and lowercase letters (A-Z,a-z), numerics (0-9), period (.), dash (-) and underbar(_). In addition, the special characters $, %, and # may be specified. (Since these characters are not part of the XPG4 portable character set, however, you should consider the future possibility of program portability before using these characters.)

Length:
Specified by the Userid_Length parameter

The name of an area, 0 to 8 characters in length, that contains a user ID. If a user ID is not required (Userid_Length is zero), this parameter is ignored.

Security_Class_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Security_Class. The Security_Class_Length must be in the range of 1 to 8.

Security_Class
Supplied parameter

Type: Character string

Character set:
Uppercase alphanumeric

Length:
Specified by the Security_Class_Length parameter

The name of an area, 1 to 8 characters in length, that contains the Security_Class. The Security_Class parameter cannot specify DATASET. For systems using RACF, the class name specified must be in the RACF class descriptor table.

Entity_Name_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Entity_Name. The Entity_Name_Length can be in the range of 1 to 246.

Entity_Name
Supplied parameter

Type: Character string

Character set:
Uppercase alphanumeric

Length:
Specified by the Entity_Name_Length parameter

The name of an area, 1 to 246 characters in length, that contains the Entity_Name.

Access_Type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a numeric value that identifies the type of access to check for. The following Access_Type constants are defined by the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Access
ACK_READ#	check READ authority
ACK_UPDATE#	check UPDATE authority
ACK_CONTROL#	check CONTROL authority
ACK_ALTER#	check ALTER authority

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the auth_check_resource_np service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the auth_check_resource_np service stores the return code. The auth_check_resource_np service returns Return_code only if Return_value is -1. For a list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The auth_check_resource_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One or more of the following conditions were detected: • Access_Type specified is undefined • Userid_Length is outside allowable range (0-8) • Security_Class_Length is outside allowable range (1-8) • Entity_Name_Length is outside allowable range (1-246) The following reason codes can accompany the return code: JRAccessUndefined, JRUserNameLenError, JRClassLenErr, or JREntityLenErr.
ESRCH	One or more of the following conditions were detected: • The user ID is not defined to the security product • No mapping to a user ID exists for the specified UUIDs • The resource is not defined to the security product • The DCEUUIDS class is not active The following reason codes can accompany the return code: JRSAFNoUser,JRSAFNoUUIDtoUser, JRSAFResourceUndefined, or JRSAFNoDCEClass.
ENOSYS	One or more of the following conditions were detected: • No security product is installed • SAF support for this function is not installed The following reason codes can accompany the return code: JRNoSecurityProduct, or JRSNoSAFSupport.
EMVSSAF2ERR	An error occurred in the security product. One or more of the following conditions were detected: • An internal error occurred in the security product • An error was detected in the parameter list • There was an undefined return code or reason code The following reason codes can accompany the return code: JRSAFInternal, JRSAFParmListErr, or JRUnexpectedError.
EPERM	One or more of the following conditions were detected: • If BPX.SERVER is defined, the caller does not have update permission to BPX.SERVER. If BPX.SERVER is not defined, the caller is not a superuser. • The user does not have the access specified to the resource. • The caller's address space has done a load from an uncontrolled library The following reason codes can accompany the return code: JRNotServerAuthorized, JRNoResourceAccess, or JREnvDirty.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the auth_check_resource_np service stores the reason code. The auth_check_resource_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The ability to query a user's access to protected resources is a privileged operation. An installation has the following ways of allowing an application to use this service:
   - For the highest level of security, the installation can define the BPX.SERVER resource in the FACILITY class. In order for the application to access this service, it must have at least read access to this profile. In addition, all load modules executing in the application's address space must be defined to RACF. For more information about setting up this security, see the section on establishing UNIX security in *z/OS UNIX System Services Planning*.
   - For a lower security arrangement, assign a UID of 0 to the user ID with which the application is run, so that it operates as a superuser.
2. This service cannot be used to determine access to POSIX resources, such as HFS files.
3. The access check can be made with several forms of identity. The first identity specified in the following list is used to make the authorization check:
   a. User ID
   b. Principal/Cell UUIDs
   c. Caller's task level ACEE
   d. Caller's address space level ACEE
4. When no identity is specified by the caller and the caller's task has an ACEE created with pthread_security_np (BPX1TLS, BPX4TLS) for a SURROGATE (non-password) client, both the task and address space level ACEEs are used in determining the type of access permitted to a resource.
5. Both the principal and cell UUIDs are in string form. A UUID string is 36 characters long. The string must contain the delimiter - in character positions 9, 14, 19, and 24. The general form of a UUID string is xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where x represents a valid numeric or hexadecimal character.

Related services

- "pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create|delete thread-level security" on page 518

Characteristics and restrictions

The auth_check_resource_np service is restricted to users that have the appropriate privileges.

Examples

For an example using this callable service, see "BPX1ACK (auth_check_resource_np) example" on page 1124.

bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor

Function

The bind callable service binds a unique local name to a socket descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1BND):	31-bit task or SRB mode
AMODE (BPX4BND):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1BND,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4BND with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the bind is to be done.

Sockaddr_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr.

Sockaddr
Supplied parameter

Type: Character

Length:
Length specified by Sockaddr_length.

The name of a field that contains the name to be bound to the socket descriptor. The format of Sockaddr is determined by the domain in which the socket descriptor was created. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for additional information on the format of Sockaddr.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind service stores the return code. The bind service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The bind service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EADDRINUSE	The specified address is already in use. The following reason code can accompany the return code: JRNameExists.
EAFNOSUPPORT	The address family specified in the address structure is not supported.
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINVAL	One of the input parameters was not valid. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoname.
EIO	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPERM	The user is not permitted to bind to the specified port. The following reason code can accompany the return code: JRUserNotPrivileged.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind service stores the reason code. The bind service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. An application can retrieve the assigned socket name with the getsockname service.
2. Sockets in the AF_UNIX domain create a name in the file system that must be deleted by the application (using unlink) when it is no longer needed.
3. For Sockaddr to be returned on an accept request for an AF_UNIX domain socket, the client application doing the connect must bind a unique local Sockaddr to the socket with the bind request before issuing the connect request.
4. Server applications issue the bind request to register their addresses with the system. Both connection and connectionless servers must do this before accepting requests from clients.
5. For network sockets, the user must have appropriate privileges (see "Authorization" on page 8) to bind to a port in the range from 1 to 1023.
6. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket" on page 15
- "listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients" on page 330
- "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777

Characteristics and restrictions

There are no restrictions on the use of the bind service.

Examples

For an example using this callable service, see "BPX1BND (bind) example" on page 1128.

bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address

Function

The bind with source address selection callable service binds the best source address for the provided destination IP address to an AF_INET6 socket descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1BAS):	31-bit task or SRB mode
AMODE (BPX4BAS):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1BAS,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4BAS with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the AF_INET6 socket file descriptor for which the bind with source address selection is to be done.

Sockaddr_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr.

Sockaddr
Supplied parameter

Type: Character

Length:
Length specified by Sockaddr_length.

The name of a field that contains the destination IP address. The best source address for the provided destination address will be selected and bound to the AF_INET6 socket descriptor.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind with source address selection service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind with source address selection service stores the return code. The bind with source address selection service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The bind with source address selection service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAFNOSUPPORT	The address family specified in the address structure is not supported.
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINVAL	One of the input parameters was not valid. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoname, JrInAddrAnyNotAllowed.
EIO	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EOPNOTSUPP	The socket domain type is not supported. The following reason code can accompany the return code: JrIncorrectSocketType.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the bind with source address selection service stores the reason code. The bind with source address selection service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. An application can retrieve the assigned socket name with the getsockname service.
2. An application's suggestion of “best” source address is set by the IPv6 setsockopt() call socket option SOCK#IPV6_ADDR_PREFERENCES.

3. In CINET environment, bind with source address selection service is routed to the TCPIP stack that is most appropriate for the provided destination IP address.
4. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor" on page 71
- "accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket" on page 15
- "listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients" on page 330
- "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777

Characteristics and restrictions

1. The bind with source address selection service is for IPv6 and IPv4-mapped IPv6 addresses only.
2. Only TCP and UDP sockets are supported. RAW sockets are not supported.
3. Port number is irrelevant for bind with source address selection service and will not be validated.
4. Destination IP address of IN6ADDR_ANY is not supported. The bind with source address selection service will fail with EINVAL return code and JrInAddrAnyNotAllowed reason code.

Examples

For an example using this callable service, see "BPX1BAS (bind with source address selection) example" on page 1128.

chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory

Function

The chattr service modifies the attributes that are associated with a file. It can be used to change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, file format and size, and file tag. It can also be used to set the initial security label for a file or directory. Identify the file by its path name.

For the corresponding service using a file descriptor, see "fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor" on page 156.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CHR):	31-bit

Operation	Environment
AMODE (BPX4CHR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHR,(Pathname_length,
              Pathname,
              Attributes_length,
              Attributes,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHR with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the path name of the file whose attributes you want to change.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname_length.

Attributes_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area containing the attributes that you want to change.

Attributes
Supplied parameter

Type: Structure

Length:
Specified by the Attributes_length parameter

The name of the area that contains the attributes you want to change. The area is mapped by BPXYATT. For information about the content of this area, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chattr service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chattr service stores the return code. The chattr service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The chattr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process did not have appropriate permissions. Possible reasons include: • The calling process was attempting to set access time or modification time to current time, and the effective UID of the calling process does not match the owner of the file; the process does not have write permission for the file; or the process does not have appropriate privileges (see "Authorization" on page 8). • The calling process was attempting to truncate the file, and it does not have write permission for the file.
EBUSY	The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
EFBIG	The calling process was attempting to change the size of a file, but the specified length is greater than the maximum file size limit for the process. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRWriteBeyondLimit.
EINVAL	The length of the Attributes parameter is too small, or the Attributes structure containing the requested changes is not valid. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JrInvalidAtt, JrNegativeValueInvalid, JrTrNotRegFile, JrTrNegOffset, JrFileNotEmpty, and JrInvalidFileTag.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.

Return_code	Explanation
EMVSERR	An MVS environmental error has been detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the path name is longer than 255 characters. (File name truncation is not supported.)
ENOENT	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOSYS	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
ENOTDIR	Some component of Pathname is not a directory.
EPERM	The operation is not permitted for one of the following reasons: • The calling process was attempting to change the mode or the file format, but the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see "Authorization" on page 8). • The calling process was attempting to change the owner, but it does not have appropriate privileges. • The calling process was attempting to change the general attribute bits, but it does not have write permission for the file. • The calling process was attempting to set a time value (not current time), but the effective user ID does not match the owner of the file, and it does not have appropriate privileges. • The calling process was attempting to set the change time or reference time to current time, but it does not have write permission for the file. • The calling process was attempting to change auditing flags, but the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. • The calling process was attempting to change the security auditor's auditing flags, but the user does not have auditor authority. • Attributes indicate that the security label is to be set, and one or more of the following conditions apply: – The calling process does not have RACF SPECIAL authorization and appropriate privileges. – There is already a security label that is associated with the file.
EROFS	Pathname specifies a file that is on a read-only file system. Consult Reason_code to determine the exact reason that the error occurred. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chattr service stores the reason code. The chattr service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

Table 2. Attribute fields modifiable by chattr

Set flags	Attribute fields input	Description
ATTMODECHG	ATTMODE	Set the mode according to the value in ATTMODE. See "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90.
ATTOWNERCHG	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93.
ATTSETGEN	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits are unchanged.
ATTTRUNC	ATTSIZE	Change the file size to ATTSIZE bytes. See "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203.
ATTATIMECHG	ATTATIME	If ATTLP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
ATTATIMETOD	None	Set the access time of the file to the current time.
ATTMTIMECHG	ATTMTIME	If ATTLP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
ATTMTIMETOD	None	Set the modification time of the file to the current time.
ATTMAAUDIT	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84.

Table 2. Attribute fields modifiable by chattr (continued)

Set flags	Attribute fields input	Description
ATTMUAUDIT	ATTUSERAUDIT	Set the User's auditing flags to the value specified in ATTUSERAUDIT. See "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84.
ATTCTIMECHG	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.
ATTCTIMETOD	None	Set the change time of the file to the current time.
ATTREFTIMECHG	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
ATTREFTIMETOD	None	Set the reference time of the file to the current time.
ATTFILEFMTCHG	ATTFILEFMT	Set the File Format of the file to the value specified in ATTFILEFMT.
ATTCHARSETIDCHG	ATTFILETAG	Set the file tag. See BPXYSTAT ("BPXYSTAT — Map the response structure for stat" on page 1057) for file tag mapping.
ATTSECLABELCHG	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes are to be updated. To set an attribute, turn the corresponding **Set Flag** on, and set the corresponding **Attributes Field** according to Table 2 on page 80. Multiple attributes may be changed at the same time.

   The **Set Flag** field should be cleared before any bits are turned on. It is considered an error if any of the reserved bits in the flag field are turned on.
2. Some of the attributes that are changed by the chattr service can also be changed by other services. See the related service (listed in Table 2 on page 80) for a detailed description.
3. Changing mode (ATTMODECHG = ON):
   - The file mode field in Attributes is mapped by the BPXYMODE macro (see "BPXYMODE — Map the mode constants of the file services" on page 996). For information on the values for file type, see "BPXYFTYP — File type definitions" on page 967.
   - File descriptors that are open when the chattr service is called retain the access permission they had when the file was opened.

- The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges (see "Authorization" on page 8).
- Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec, attach_exec, or spawn service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

  The set-group-ID-on-execution permission is set to zero if both of the following are true:
  - The caller does not have appropriate privileges.
  - The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.
- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

4. Changing owner (ATTOWNERCHG = ON):
   - To change the owner UID of a file, the caller must have appropriate privileges.
   - To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:
     - The effective UID of the caller matches the file's owner UID.
     - The Owner_UID value that is specified in the change request matches the file's owner UID.
     - The Group_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
   - When the owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
   - When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other can be set to its present value or to -1 to remain unchanged.
5. Changing General Attribute bits (ATTSETGEN = ON):
   - For General Attribute bits to be changed, the calling process must have write permission for the file.
6. Changing the file size (ATTTRUNC = ON):
   - The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was originally larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was originally shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

     Full blocks are returned to the file system so that they can be used again.

     The file offset is not changed.
   - When a file size is changed successfully, it clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file, unless the caller has appropriate privileges.
   - The resizing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, fails with EFBIG, and the SIGXFSZ signal is generated for the process.

- A file's size cannot be changed if it is open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to "open (BPX1OPN, BPX4OPN) — Open a file" on page 447 for details about the NFS share reservations.

7. Changing times:
   - All time fields in Attributes are in POSIX format.
   - For the access time or the modification time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
   - For the access time or modification time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
   - For the change time or the reference time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
   - For the change time or reference time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.
   - For any time field (atime, mtime, ctime, reftime), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time is set.
   - When any attribute field is changed successfully, the file's change time is also updated.
8. Changing auditor audit flags (ATTMAAUDIT = ON):
   - For auditor audit flags to be changed, the user must have auditor authority. Users with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

     You establish auditor authority by issuing the TSO/E command ALTUSER Auditor.
9. Changing user audit flags (ATTMUAUDIT = ON):
   - For the user audit flags to be changed, the user must have appropriate privileges (see "Authorization" on page 8) or be the owner of the file.
10. Changing the file format (ATTFILEFMTCHG = ON):
    - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
    - The attribute that is specified in ATTFILEFMT is the same attribute that is set by the FILEDATA=TEXT parameter on a DD statement.
11. Changing the file tag (ATTCHARSETIDCHG=ON):
    - A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.
    - The tagging of `/dev/null`, `/dev/random`, `/dev/urandom`, and `/dev/zero` is ignored.
12. Changing the security label (ATTSECLABELCHG=ON):
    - For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see "Authorization" on page 8), and no security label must currently exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The

function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.

Related services

- "fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor" on page 156
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805
- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93
- "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path"
- "utime (BPX1UTI, BPX4UTI) — Set file access and modification times" on page 879
- "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203
- "truncate (BPX1TRU, BPX4TRU) — Change the size of a file" on page 859
- "lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link" on page 315

Characteristics and restrictions

1. The ATTEXTLINK flag in the ATTGENVALUE field of BPXYATT cannot be modified with BPX1CHR (BPX4CHR).
2. The general attribute fields (set by ATTSETGEN, ATTGENMASK, and ATTGENVALUE fields) are not intended as a general-use programming interface to BPX1CHR (BPX4CHR).
3. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges (see "Authorization" on page 8). It cannot be used to change an existing security label; it can only be used to set an initial security label on a file.

Examples

For an example that uses this callable service, see "BPX1CHR (chattr) example" on page 1131.

chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path

Function

The chaudit service changes the types of access to a file to be audited for the security product. The chaudit service identifies the file by its path name.

For the corresponding service using a file descriptor, see "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN

Operation	Environment
AMODE (BPX1CHA):	31-bit
AMODE (BPX4CHA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHA,(Pathname_length,
              Pathname,
              Audit_flags,
              Option_code,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHA with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the path name of the file.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the path nme of the file for which auditing is to be changed.

Path names can begin with or without a slash:

- A path name that begins with a slash is an absolute pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative pathname. The search for the file starts at the working directory.

Audit_flags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that indicates the access to be audited. This field is mapped by the BPXYAUDT macro; see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. Valid values for this field include any combination of the following:

Value	Description
AUDTREADFAIL	Audit failing read requests.
AUDTREADSUCCESS	Audit successful read requests.
AUDTWRITEFAIL	Audit failing write requests.
AUDTWRITESUCCESS	Audit successful write requests.
AUDTEXECFAIL	Audit failing execute or search requests.
AUDTEXECSUCCESS	Audit successful execute or search requests.

Option_code
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates whether you are changing the auditing for the user or for the security auditor. When this field has the value:

- `0`, the user's auditing is being changed.
- `1`, the security auditor's auditing is being changed. A superuser who is not the auditor cannot change the auditor's authority.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chaudit service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chaudit service stores the return code. The chaudit service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a list of possible return code values. The chaudit service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have search permission for some component of the Pathname prefix.
EINVAL	The Option_code parameter is incorrect. The following reason code can accompany the return code: JRBadAuditOption.

Return_code	Explanation
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or some component of the path name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file named Pathname was found, or no path name was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the Pathname prefix is not a directory.
EPERM	The effective UID of the calling process does not match the file's owner UID; the calling process does not have appropriate privileges (see "Authorization" on page 8); or if Option_code indicated that the auditor audit flags were to be changed, the user does not have auditor authority.
EROFS	The file exists on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chaudit service stores the reason code. The chaudit service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If Option_code indicates that the auditor audit flags are to be changed, the user must have auditor authority for the request to be successful. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

   You can get auditor authority by entering the TSO/E command ALTUSER Auditor.
2. If Option_code indicates that the user audit flags are to be changed, the user must have appropriate privileges (see "Authorization" on page 8) or be the owner of the file.

Related services

- "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the chaudit service.

Examples

For an example using this callable service, see "BPX1CHA (chaudit) example" on page 1129.

chdir (BPX1CHD, BPX4CHD) — Change the working directory

Function

The chdir service changes your working directory from the current one to a new one. The working directory is the starting point for path searches of pathnames that do not begin with a slash.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CHD):	31-bit
AMODE (BPX4CHD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHD,(Pathname_length,
              Pathname,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHD with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the directory that is to become your new working directory.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the new directory. This field has the length specified in Pathname_length.

Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chdir service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chdir service stores the return code. The chdir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The chdir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search one of the components of Pathname.
EINVAL	The Pathname parameter is not valid; it contains nulls.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of Pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	No directory named Pathname was found, or no Pathname was specified. The following reason codes can accompany the return code: JRChdNoEnt and JRQuiescing.
ENOTDIR	Some component of Pathname is not a directory. The following reason code can accompany the return code: JRChdNotDir.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chdir service stores the reason code. The chdir service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Related services

- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "chroot (BPX1CRT, BPX4CRT) — Change the root directory" on page 100
- "fchdir (BPX1FCD, BPX4FCD) — Change the working directory" on page 167
- "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615
- "realpath (BPX1RPH, BPX4RPH) — Resolve a pathname" on page 594
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the chdir service.

Examples

For an example using this callable service, see "BPX1CHD (chdir) example" on page 1130.

chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory

Function

The chmod service modifies the permission bits that are used to control the owner access, group access, and general access to a file. You can use it to set flags that modify the user ID (UID) and group ID (GID) of the file when it is executed. You can also use it to set the sticky bit to indicate from where the file should be fetched. You identify the file by its pathname.

For the corresponding service using a file descriptor, see "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CHM):	31-bit
AMODE (BPX4CHM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHM,(Pathname_length,
              Pathname,
              Mode,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHM with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file whose mode you want to change.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file. This field has the length that is specified in Pathname_length.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that describes the access. This field, which is mapped by BPXYMODE, specifies the file type and permissions for the caller, for the callers group, and for any others. For more information, see "BPXYMODE — Map the mode constants of the file services" on page 996.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chmod service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chmod service stores the return code. The chmod service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The chmod service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search some component of Pathname.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters. Filename truncation is not supported.
ENOENT	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of Pathname is not a directory.
EPERM	The effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	Pathname specifies a file that is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chmod service stores the reason code. The chmod service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. File descriptors that are open when the chmod service is called retain the access permission they had when the file was opened.

2. For mode bits to be changed, the effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges (see "Authorization" on page 8).
3. A user with READ authority to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class can use the chmod service to change the permission bits of any file.
4. When the mode is changed successfully, the file's change time is also updated.
5. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec, spawn, or attach_exec service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file rather than that of the actual invoker.

   The set-group-ID-on-execution permission is set to zero if both of the following are true:

   - The caller does not have appropriate privileges.
   - The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.
6. Setting the set-user-ID-on-execution permission means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

Related services

- "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory"
- "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the chmod service.

Examples

For an example using this callable service, see "BPX1CHM (chmod) example" on page 1130.

chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory

Function

The chown service changes a file's owner, group, or both owner and group. The owner is identified by a user ID (UID) and a group ID (GID).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CHO):	31-bit
AMODE (BPX4CHO):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHO,(Pathname_length,
              Pathname,
              Owner_UID,
              Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHO with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file whose owner or group is to be changed.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file. This field has the length that is specified in Pathname_length.

Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Owner_UID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new owner UID that is assigned to the file. If there is no change, this field contains the present value or -1. This parameter must be specified.

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new owner GID that is assigned to the file. If there is no change, this field contains the present value or -1. This parameter must be specified.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chown service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chown service stores the return code. The chown service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The chown service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search some component of the Pathname prefix.
EINVAL	The Owner_UID or Group_ID parameter is incorrect.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters.
ENOENT	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of the Pathname prefix is not a directory.
EPERM	The calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chown service stores the reason code. The chown service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The chown service changes the owner UID and owner GID of a file. Only a caller with appropriate privileges can change the owner UID of a file. Refer to "Authorization" on page 8 for information on appropriate privileges.
2. The owner GID of a file can be changed by a caller if the caller has appropriate privileges, or if a caller meets all of these conditions:
   - The effective UID of the caller matches the file's owner UID.
   - The Owner_UID value that is specified in the change request matches the file's owner UID.
   - The Group_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner_UID and Group_ID must be specified. To change only one of these values, set the one that is to remain unchanged to its present value or to -1.

Related services

- "fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor" on page 171
- "lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link" on page 324
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196
- "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the chown service.

Examples

For an example using this callable service, see "BPX1CHO (chown) example" on page 1130.

chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process

Function

The chpriority callable service changes the scheduling priority of a process, process group, or user.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CHP):	31-bit
AMODE (BPX41CHP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CHP,(Which,
              Who,
              PriorityType,
              Priority,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CHP with the same parameters.

Parameters

Which
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

- PRIO_PROCESS = Indicates that the Who parameter is to be interpreted as a process ID
- PRIO_PGRP = Indicates that the Who parameter is to be interpreted as a process group ID
- PRIO_USER = Indicates that the Who parameter is to be interpreted as a user ID

The PRIO_ constants are defined in the BPXYCONS macro.

Who
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID, or user ID whose priority is to be changed. The Which parameter indicates how this parameter is to be interpreted. If this parameter is interpreted as a process group ID or user ID, all processes with the specified process group ID or user ID are to have their priority changed. A value of zero for this parameter specifies the current process, process group, or user ID.

PriorityType
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates how the Priority parameter is to be interpreted. This parameter can have one of the following values:

- CPRIO_ABSOLUTE = Indicates that the Priority parameter is to be interpreted as an absolute value. This causes the priority value of the target process(es) to be set to the value specified by the Priority parameter.
- CPRIO_RELATIVE = Indicates that the Priority parameter is to be interpreted as a relative value. This causes the priority value of the target process(es) to be incremented or decremented by the value that is specified by the Priority parameter.

The CPRIO_ constants are defined in the BPXYCONS macro.

Priority
Supplied parameter

Type: Signed Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the priority value that the specific process or group of processes is to be set to or changed by.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chpriority service returns -1 if it is not successful. If it is successful, the chpriority service returns a value of zero.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chpriority service stores the return code. The chpriority service returns Return_code only if Return_value is -1. See *z/OS*

UNIX System Services Messages and Codes for a complete list of possible return code values. The chpriority service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The priority is being changed to a lower value, and the current process does not have the appropriate privilege (see "Authorization" on page 8) to do so.
EPERM	A process was located, but the saved set-user-ID of the calling process does not match the saved set-user-ID of the process whose priority is being changed.
EINVAL	The value of the Which parameter was not recognized; the value of the Who parameter is not a valid process ID, process group ID or user ID; or the value of the PriorityType parameter is not supported.
ESRCH	No process could be located using the Which and Who parameter values specified.
EMVSSAF2ERR	A Security product internal error has occurred. Consult the Reason_code parameter for the exact reason for the error.
ENOSYS	The system does not support this function. Your installation has chosen not to enable this function.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chpriority service stores the reason code. The chpriority service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the supplied Who and Which values specify more than one process, each of the specified processes has its priority values set to the supplied value. If at least one of the specified processes has its priority value successfully changed, the chpriority service returns successfully.
2. The priority value of a process is an integer that can be in the range of -20 to 19. If the priority value that is supplied causes the priority value of a process to be outside this range, the priority of the process is set to the corresponding limit value. The default priority value for all processes is 0.
3. An increase in the priority value of a process results in a lower CPU priority for the process. A decrease in the priority value of a process results in a higher CPU priority for the process.
4. If the supplied priority value would result in a lower priority value for the specified processes, the caller must have appropriate privileges. Refer to "Authorization" on page 8 for information about appropriate privileges. In addition to being able to lower the priority value, a caller with appropriate privileges can change the priority of any other process, regardless of the saved set-user-ID value of the process.
5. The setting of the priority value of a process has a corresponding effect on its nice value, as they both represent the relative CPU priority of the process. For example, if you use the chpriority service to change the priority value of a process to its maximum value (19), the nice value of the process is changed to

its maximum value (2*NICE_ZERO)-1. This is reflected on the nice, getpriority, chpriority and setpriority services. The NICE_ZERO constant is defined in BPXYCONS.

6. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
7. If the supplied Who and Which values specify a process in a multiple—process address space, each of the processes in the address space will have their priority values set to the supplied value.
8. For information about the necessary system setup for this service, see the documentation for the BPXPRMxx parmlib member in Enabling nice(), setpriority(), and chpriority() support in *z/OS UNIX System Services Planning*.

Related services

- "nice (BPX1NIC, BPX4NIC) — Change the nice value of a process" on page 432
- "getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process" on page 255
- "setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process" on page 688

Characteristics and restrictions

There are no restrictions on the use of the chpriority service.

Examples

For an example using this callable service, see "BPX1CHP (chpriority) example" on page 1131.

chroot (BPX1CRT, BPX4CRT) — Change the root directory

Function

The chroot service changes the root directory from the current one to a new one. The root directory is the starting point for path searches of pathnames beginning with a slash. The working directory of the process is unaffected by chroot().

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CRT):	31-bit
AMODE (BPX4CRT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CRT,(Pathname_length,
              Pathname,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CRT with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the directory that is to become your root directory.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the new directory. This field has the length that is specified in Pathname_length.

Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the current root directory, and the search for the file starts at the current root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chroot service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chroot service stores the return code. The chroot service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The chroot service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search one of the components of Pathname.
EINVAL	The Pathname parameter is not valid; it contains nulls.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of Pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	No directory named Pathname was found, or no Pathname was specified. The following reason codes can accompany the return code: JRChdNoEnt and JRQuiescing.
ENOTDIR	Some component of Pathname is not a directory. The following reason code can accompany the return code: JRChdNotDir.
EPERM	The calling process is not a superuser. The following reason code can accompany the return code: JRUserNotPrivileged.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the chroot service stores the reason code. The chroot service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Upon completion of the chroot, the specified directory is now the logical root of the file system for the process. All searches for pathname beginning with slash (/) start from this directory, and all attempts to use dot dot (..) over the root remain in the new root.
2. A new child process inherits a parent's changed root directory.
3. If the current working directory is above the new root, chroot(.) can be used to reset the root directory to equal the current working directory. However, when the current working directory is above the root directory, getcwd() fails with ENOENT return code.

Related services

- "chdir (BPX1CHD, BPX4CHD) — Change the working directory" on page 88
- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452

- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

To change the root directory, the caller must have appropriate privileges (see "Authorization" on page 8).

Examples

See "BPX1CRT (chroot) example" on page 1133 for an example using this callable service.

close (BPX1CLO, BPX4CLO) — Close a file

Function

The close callable service closes a file. You identify the file by its file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB. If SRB, AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1CLO):	31-bit task or SRB mode
AMODE (BPX4CLO):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CLO,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CLO with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the file descriptor of the file or socket the caller wants closed. The file descriptor is returned by the open service (see

"open (BPX1OPN, BPX4OPN) — Open a file" on page 447) or by the socket service (see "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the close service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the close service stores the return code. The close service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The close service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The service did not complete, because the file descriptor specified is currently in use by another thread in the same process.
EBADF	The File_descriptor does not identify a valid, open file. The following reason codes can accompany the return code: JRClNeedClose and JRNotForDir.
EINTR	The service was interrupted by a signal while it was processing the close request. The file may or may not be closed.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the close service stores the reason code. The close service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Closing a file closes, or frees, the file descriptor by which the file was known to the process. The system can then reassign the file descriptor to the same file or to another file when it is opened.
2. Closing a file descriptor also unlocks all outstanding byte range locks that a process has on the associated file.
3. If a file has been opened by more than one process, each process has a file descriptor. When the last open file descriptor is closed, the file itself is closed. If the file's link count is zero at that time, the file's space is freed and the file

becomes inaccessible. When the last open file descriptor for a pipe or FIFO special file is closed, any data remaining in the file is discarded.

4. The close callable service is for files or sockets.
5. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481
- "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the close service.

Examples

For an example using this callable service, see "BPX1CLO (close) example" on page 1132.

closedir (BPX1CLD, BPX4CLD) — Close a directory

Function

The closedir callable service closes a directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CLD):	31-bit
AMODE (BPX4CLD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CLD,(Directory_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CLD with the same parameters.

Parameters

Directory_file_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the closedir service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the closedir service stores the return code. The closedir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The closedir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The Directory_file_descriptor parameter does not represent an open directory.
EINTR	The service was interrupted by a signal while it was processing a closedir request. The directory may or may not be closed.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the closedir service stores the reason code. The closedir service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613

Characteristics and restrictions

There are no restrictions on the use of the closedir service.

Examples

For an example using this callable service, see "BPX1CLD (closedir) example" on page 1132.

cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events

Function

The cond_cancel callable service allows the thread to cancel the effects of a call to the cond_setup service (BPX1CSE).

Requirements

Condition	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CCA):	31-bit
AMODE (BPX4CCA):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CCA,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CCA with the same parameters.

Parameters

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a `0` to indicate that the interest in event notifications has been canceled, or `-1` if it has not.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service stores the return code. The cond_cancel service stores a return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The cond_cancel service stores a reason code only when the return value is -1. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. A program can use the cond_cancel service to clean up when it uses the cond_setup service, but does not call cond_wait or cond_timed_wait. The cond_setup service causes the thread to be eligible to receive event notifications. If the program running on the thread is no longer interested in these events, it should call cond_cancel to tell the system that event notifications are no longer required.
2. If you intend to call cond_wait or cond_timed_wait at a later time to wait until some event occurs, use the cond_setup service to make your program eligible to receive event notifications. The system notes that your program will be waiting for some other thread, either to send it a signal or to use the cond_post service to send an event notification. Both of these require the use of z/OS UNIX services. If z/OS UNIX determines that it has become impossible to send a signal or event notification to your program, it checks to see whether your program is or will be calling the cond_wait or cond_timed_wait services. If so, z/OS UNIX abnormally terminates your program to prevent it from waiting for something that cannot occur. For this reason, if your program uses the cond_setup service but does not subsequently call either cond_wait or cond_timed_wait, it should use the cond_cancel service to cancel the setup to receive event notifications.
3. When the program cannot determine whether cond_wait or cond_timed_wait has been called, it should call cond_cancel to ensure that the thread is not eligible to receive event notifications.

Related services

- "cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications" on page 111
- "cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event" on page 114
- "cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event" on page 118

Characteristics and restrictions

There are no restrictions on the use of the cond_cancel service.

Examples

For an example that uses this callable service, see "BPXICCA (cond_cancel) example" on page 1129.

cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event

Function

The cond_post callable service notifies another thread in the process that an event has occurred.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CPO):	31-bit
AMODE (BPX4CPO):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CPO,(Thread_ID,
              Event,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CPO with the same parameters.

Parameters

Thread_ID
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be notified of the event. The target thread must be in the same process as the caller.

Event
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an integer value that determines which event notification is to be sent to the target thread. The Event value represents an event for which the thread identified by Thread_ID may be waiting. If the target thread is waiting, the cond_post service notifies it that the event has occurred.

The value that is specified by Event must be one of the following two event values, which are defined by the BPXYCW macro:

- **CW_CONDVAR** causes the target thread to resume processing if it is waiting for a CW_CONDVAR event.
- **CW_TIMEOUT** causes the target thread to resume processing if it is waiting for a timeout notification.

Note:

1. You must specify exactly one event.
2. Use of cond_post to send a CW_TIMEOUT notification is restricted to programs that run in supervisor state with protect key 0.

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a 0 if an event notification was sent to the target thread, or -1 if it was not.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service stores the return code. This service routine returns the return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The cond_post service may return one of the following values in the return code parameter:

Error	Explanation
EINVAL	The value specified by Thread_ID is not valid. Either the Event parameter contains an incorrect value, or Thread_ID contains a lightweight thread ID. The following reason codes unique to the cond_post call can accompany this return code: JRLightWeightThID, JRNoEvents, JRTimeOutNotAuth, JRTooMany, JRUndefEvents.
ESRCH	The system determined that the value that was specified by Thread_ID does not refer to a thread that currently exists in the caller's process. The following reason codes can accompany this return code: JRThreadNotFound, JRAlreadyTerminated.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

The cond_post service attempts to send an event notification to the target thread. Event notifications are delivered to a target thread only when the thread is set up to receive them. If the target thread is not set up to receive it, the event notification is discarded. The cond_post service does not check whether the target thread is set up to receive the event, so the cond_post service can return a value of `0` even though the event notification was discarded. Therefore, if you use the cond_wait and cond_post services to synchronize threads, you must be certain that the target thread is set up for the wait or in the wait before you use cond_post to send the notification.

Related services

- "cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event" on page 114
- "cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event" on page 118

Characteristics and restrictions

The target thread must be in the same process as the caller.

Examples

For an example using this callable service, see "BPX1CPO (cond_post) example" on page 1132.

cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications

Function

The cond_setup callable service makes the calling thread eligible to receive event notifications from other threads.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CSE):	31-bit
AMODE (BPX4CSE):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CSE,(Event_list,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CSE with the same parameters.

Parameters

Event_list
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that specifies which events are of interest to the thread. The value contained in Event_list is the inclusive OR of one or more of the following event values, which are defined by the BPXYCW macro:

CW_INTRPT
The program that is running on the thread needs to know about signals sent to the thread.

CW_CONDVAR
The program that is running on the thread needs to suspend processing until some other thread uses the cond_post service to send this thread a notification of a CW_CONDVAR event.

You must specify at least one event; you may specify both.

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a `0` upon normal completion, or -1 otherwise.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the cond_setup call stores the return code. The cond_setup call stores return code only if return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The cond_setup call can return one of the following values in the return code parameter:

Error	Explanation
EINVAL	The system determined that the event list that was passed to the service is in error. The following reason codes unique to the cond_setup call can accompany the return code: JRAlreadySetup, JRNoEvents, JRUndefEvents.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The cond_setup service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The effects of the cond_setup request remain until the next service is requested. The cond_setup service is intended to used to set up for a subsequent call to cond_wait or cond_timed_wait. If the program invokes other callable services between cond_setup, on the one hand, and cond_wait or cond_timed_wait, on the other hand, cond_wait or cond_timed_wait may fail with a return value of -1, a reason code of EINVAL, and a reason code of JRNotSetup.

   The only exception to this is the queue_interrupt service. You can use the queue_interrupt service to "put back" the last signal delivered to the signal interface routine.
2. If you use cond_setup to specify the events that cause the thread to resume processing, you must repeat the setup before each call to cond_wait or cond_timed_wait.
3. If you use cond_setup with cond_timed_wait, do not specify the CW_TIMEOUT condition on the call to cond_setup. The cond_timed_wait service provides setup for the CW_TIMEOUT event.
4. Calling the cond_setup service before the cond_wait and cond_timed_wait services is optional. If the thread does not need to do any additional processing between the time it becomes eligible to request event notification and the time it suspends, you can specify the events on cond_wait or cond_timed_wait instead of using cond_setup.
5. If a thread has called cond_setup but has not called cond_wait or cond_timed_wait, any cond_post services to it are remembered, and processed following the setup. When the cond_wait or cond_timed_wait service is called, the pending cond_post prevents the caller from waiting.

Related services

- “cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events” on page 107
- “cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event” on page 109
- “cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event” on page 114
- “cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event” on page 118
- “queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered” on page 568

Characteristics and restrictions

The program running on the thread should eventually call one of the cond_wait, cond_timed_wait, or cond_cancel services.

Examples

For an example using this callable service, see "BPX1CSE (cond_setup) example" on page 1133.

cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event

Function

The cond_timed_wait callable service suspends the calling thread until any one of a set of events has occurred, or until a specified amount of time has passed.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CTW):	31-bit
AMODE (BPX4CTW):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CTW,(Seconds,
              Nanoseconds,
              Event_list,
              Seconds_remaining,
              Nanoseconds_remaining,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CTW with the same parameters.

Parameters

Seconds

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that is the maximum number of seconds that the calling program is willing to wait for one of the specified events to occur.

Note:

1. Seconds can be any value greater than or equal to 0, and less than or equal to 4 294 967 295. The value specified for Seconds is an unsigned integer.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

Nanoseconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that is the number of nanoseconds to be added to the value specified by Seconds.

Note:

1. Nanoseconds can be any value greater than or equal to 0, and less than or equal to 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

Event_list
Supplied parameter

Type: Integer

Length:
Fullword

Event_list specifies the name of a fullword that contains a value that determines which events are to cause the thread to resume processing.

The value that is contained in the event list is the inclusive OR of one or more of the following event values, which are defined by the BPXYCW macro:

CW_INTRPT
Suspends processing until a signal is sent to the thread. This is a cancelation point that is described in the usage notes of "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527.

CW_CONDVAR
Suspends processing until some other thread in the process sends this one a CW_CONDVAR notification.

If the event list is zero, the caller has used the cond_setup service to specify the events, and the thread is already eligible to be notified of events. In this case, the cond_timed_wait service sets the timer for the specified interval, and suspends thread processing until an event occurs, a signal arrives, or the time limit is reached.

Seconds_remaining
Supplied returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the cond_timed_wait returns an unsigned integer that is the number of seconds of unexpired time remaining in the time interval.

Note: The Seconds_remaining value is valid only when the return value is `0` or EINTR.

`Nanoseconds_remaining`
Supplied returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the cond_timed_wait returns an unsigned integer that is the number of nanoseconds of unexpired time remaining in the time interval.

Note:

1. Nanoseconds_remaining can be any value greater than or equal to 0, and less than or equal to 1 000 000 000.
2. The nanoseconds remaining value is valid only when the return value is `0` or EINTR.

`Return_Value`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a `0` if a CW_CONDVAR event occurred, or -1 if it has not.

`Return_Code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service stores the return code. The cond_timed_wait service stores a return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. cond_timed_wait may return one of the following values in the Return_code parameter:

Error	Explanation
EAGAIN	No signal or event notification arrived within the specified timeout period. The thread resumed processing because the time interval expired. **Note:** If you specify a value of zero for both Seconds and Nanoseconds, and no event notification is pending when you call cond_timed_wait, the service returns this error.

Error	Explanation
EINTR	A signal caused the cond_timed_wait service to resume processing of the thread. **Note:** The signal handler has already run.
EINVAL	The system determined that one or more of the parameters that were passed to the service are in error. The following reason codes unique to the cond_timed_wait call can accompany the return code: JRAlreadySetup, JRNanoSecondsTooBig, JRNotSetup, JRUndefEvents.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The cond_timed_wait service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The cond_timed_wait service is similar to the POSIX function **nanosleep()**. (Refer to the POSIX standard for a description of **nanosleep()**.) If you need the **nanosleep()** function, you can use cond_timed_wait to implement your own version.
2. If your program uses cond_timed_wait to wait for events that it specified by calling cond_setup, it must not call any other z/OS UNIX services between the calls to cond_setup and cond_timed_wait. If the program invokes other callable services between cond_setup and cond_timed_wait, the cond_timed_wait callable service fails with a return value of -1, a return code of EINVAL, and a reason code of JRNotSetup.

   The only exception to this is the queue_interrupt service. You can use the queue_interrupt service to "put back" the last signal delivered to the signal interface routine. A signal can arrive after the program that is running on the thread has called cond_setup, and before it gets a chance to call cond_timed_wait. The program may choose to "put back" the signal to defer handling of it until a later time.
3. If you use cond_setup to specify the events that are to cause the thread to resume processing, you must repeat the setup before each call to cond_wait or cond_timed_wait.
4. If the caller has a PSW key of 0 or a key that is different from the one that was in effect when the process was created, cond_timed_wait gives a return value of -1 with a return code of EMVSERR and a reason code of JRPswKeyNotValid.
5. If the thread has been set up for signals, the cond_timed_wait service must run on the same request block (RB) that was used when the setup for signals was performed.
6. If you do not include the CW_INTRPT event when you use cond_timed_wait, some services that are used by other threads or processes cannot cause the waiting thread to resume processing. In particular, the following services do not cause an event notification unless CW_INTRPT is specified in the event list:
   - kill
   - pthread_cancel

- pthread_kill
- pthread_quiesce

Related services

- "cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events" on page 107
- "cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event" on page 109
- "cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications" on page 111
- "cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event"
- "queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered" on page 568

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1CTW (cond_timed_wait) example" on page 1133.

cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event

Function

The cond_wait callable service allows the caller's thread to suspend processing until any one of a set of events has occurred.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CWA):	31-bit
AMODE (BPX4CWA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CWA,(Event_list,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CWA with the same parameters.

Parameters

Event_list
Supplied returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that determines which events will cause the thread to resume processing.

The value contained in Event_list is the inclusive OR of one or more of the following event values defined by the BPXYCW macro.

CW_INTRPT
Suspends processing until a signal is sent to the thread.

CW_CONDVAR
Suspends processing until some other thread in the process sends this one a CW_CONDVAR event notification.

An Event_list of zero means that the caller has used the cond_setup service to specify the events, and the thread is already eligible to be notified of events. In this case, the cond_wait service suspends thread processing until an event occurs or a signal arrives.

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a 0 a CW_CONDVAR event occurred, or -1 otherwise.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service stores the return code. The cond_wait service stores a return code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The cond_wait service may return one of the following values in the Return_code parameter:

Error	Explanation
EINTR	A signal caused the cond_wait service to resume processing of the thread. **Note:** The signal handler has already run.
EINVAL	The system determined that one or more of the parameters that were passed to the service are in error. The following reason codes unique to the cond_wait call can accompany the return code: JRAlreadySetup, JRNotSetup, JRUndefEvents.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The cond_wait service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. If your program uses cond_wait to wait for events that it specified by calling cond_setup, it must not call any other z/OS UNIX services between the calls to cond_setup and cond_wait. If the program invokes other callable services between cond_setup and cond_wait, the cond_wait callable service fails with a return value of -1, a return code of EINVAL, and a reason code of JRNotSetup.

   The only exception to this is the queue_interrupt service. You may use the queue_interrupt service to "put back" the last signal delivered to the signal interface routine. A signal may arrive after the program that is running on the thread has called cond_setup and before it gets a chance to call cond_wait. The program may choose to "put back" the signal to defer handling it until a later time.

   If you use cond_setup to specify the events that will cause the thread to resume processing, you must repeat the setup before each call to cond_wait or cond_timed_wait.
2. If the caller has a PSW key of 0 or a key that is different from the one that was in effect when the process was created, cond_wait gives a return value of -1, a return code of EMVSERR, and a reason code of JRPswKeyNotValid.
3. If the thread has been set up for signals, the cond_timed_wait service must run on the same request block (RB) that was used when the setup for signals was performed.
4. If you do not include the CW_INTRPT event when you use cond_wait, some services that are used by other threads or processes cannot cause the waiting thread to resume processing. In particular, the following services do not cause an event notification unless CW_INTRPT is specified in the event list:
   - kill
   - pthread_cancel
   - pthread_kill
   - pthread_quiesce

Related services

- “cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events” on page 107
- “cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event” on page 109
- “cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications” on page 111
- “cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event” on page 114
- “queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered” on page 568

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

See "BPX1CWA (cond_wait) example" on page 1134 for an example using this callable service.

connect (BPX1CON, BPX4CON) — Establish a connection between two sockets

Function

For stream sockets, the connect callable service establishes a connection from a client socket to a socket at a server. For UDP (Universal Datagram Protocol) sockets, the connect callable service specifies the peer for a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1CON):	31-bit task or SRB mode
AMODE (BPX41CON):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CON,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CON with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the connect is to be done.

Sockaddr_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr.

Sockaddr
Supplied parameter

Type: Character

Length:
Length specified by Sockaddr_length.

The name of a field that contains the address of the socket or the name of the peer to which a connection is to be attempted.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the connect service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the connect service stores the return code. The connect service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The connect service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	For AF_UNIX sockets, search permission is denied for a component of the path prefix, or write access to the named socket is denied.
EAFNOSUPPORT	The address family that was specified in the address structure is not supported.
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNREFUSED	The attempt to connect was rejected. The connect request may exceed the backlog count of the target socket, or the target socket may be closed. The following reason codes can accompany the return code: JRSocketNotFound, JRExceedsBacklogCount, JRListenNotDone.

Return_code	Explanation
EINVAL	The length that is specified in the Sockaddr_length or in the name length field in the Sockaddr is not valid. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoName.
EINTR	A signal interrupted the connect service before this connection was accepted. The following reason code can accompany the return code: JRSignalReceived.
EIO	There has been a network or transport failure. The following reason codes can accompany the return code: JRPrevSockError, JRTransportDriverNotAccessible.
EISCONN	The socket is already connected.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EOPNOTSUPP	The socket is ready to accept connections. An accept request was expected. The following reason code can accompany the return code: JRListenAlreadyDone.
EPROTOTYPE	The address specifies a socket that is not the correct type for this request. The following reason code can accompany the return code: JRIncorrectSocketType.
EWOULDBLOCK	The socket is marked nonblocking, and the connection cannot be completed immediately.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the connect service stores the reason code. The connect service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. For connectionless sockets, the connect service may be advantageous because the destination address need not be specified for every datagram sent. Once a UDP (connectionless) socket is connected, the read, write, recv, and send system calls can be used for I/O on those sockets. Otherwise, only the sendto/recvfrom system calls can be used. Once a UDP socket is connected, only datagrams from the specified sockaddr are received on the socket. To disconnect a UDP socket from a previous connection, issue the connect system call with an invalid (null) sockaddr.
2. The connect callable service can be used to test whether a target socket is available for the connect. If the socket is not available, an ECONNREFUSED is returned.
3. The connect callable service will always either immediately succeed or fail, depending on the condition of the queue of pending connections, or backlog queue. If the backlog queue is not full, the connect request will immediately succeed. If the backlog queue is full, the connect request will fail with Return_code of ECONNREFUSED.

4. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Characteristics and restrictions

There are no restrictions on the use of the connect service.

Examples

For an example using this callable service, see "BPX1CON (connect) example" on page 1132.

__console() (BPX1CCS, BPX4CCS) — Communicate with console (modify/stop/WTO/DOM)

Function

The __console() service sends messages to the console and waits on a modify/stop request from the console. Additional functions available under __console2() allow you to specify routing and descriptor codes for messages sent to the console and delete held messages from the console, using message IDs or tokens. These functions are activated under the expanded BPXYCCA structure in the Version 2 section. See the usage notes for information about using the __console2() functions. Additionally, the BPXYCCA structure Version 3 section provides the ability to specify a user-supplied CART and console ID. This function can be used by applications that communicate with more than one console to ensure that the messages go to the expected console.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1CCS):	31-bit
AMODE (BPX4CCS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CCS,(MsgAttributes_length,
              MsgAttributes,
              Modify_buffer_ptr,
              Modify_string_length,
              Console_command,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4CCS with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

MsgAttributes_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area that contains the message attributes of the message that is to be sent to the console. If the length is zero, the MsgAttributes parameter is ignored, and no message is sent to the console.

MsgAttributes
Supplied parameter

Type: Structure

Length:
Specified by the MsgAttributes_length parameter.

The name of the area that contains the message attributes of the message that is to be sent to the console. Included in this macro mapping are the address and length of the message to be sent. The area is mapped by BPXYCCA. For information about the content of this area, see "BPXYCCA — Map input/output structure for __console()" on page 950.

Modify_buffer_ptr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

A fullword (doubleword) field that contains the address of a 128-byte buffer that is to be used to receive a string of EBCDIC data from the console modify command. All characters that appear to the right of the APPL= are placed into this buffer, left-aligned. The length of the string copied is returned in the Modify_string_length parameter. The data returned is folded to uppercase. If this parameter is zero, this service does not wait for or process any console modify/stop commands.

Modify_string_length
Returned parameter

Type: Integer

Character set:
No restriction

Length:
Fullword

The name of a fullword in which the __console() service returns the length of the modify string that is returned at the location specified by Modify_buffer_ptr. If the Modify_buffer_ptr is zero, this parameter is unchanged.

Console_command
Returned parameter

Type: Integer

Character set:
No restriction

Length:
Fullword

The name of a fullword in which the __console() service returns the type of command that was issued. The values are CONSOLE_MODIFY and CONSOLE_STOP.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __console() service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __console() service stores the return code. The __console() service returns Return_code only if Return_value is 0. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __console() service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	A message attribute was not valid. An error was detected in one of the fields described by BPXYCCA (JrMsgLength, JRMsgMaxLines, or JrMsgAttrErr).
EINTR	The syscall was interrupted by a signal.
EFAULT	User storage that was passed in could not be accessed. The reason code identifies the bad user storage (JRMsgIdList, JrDescList, JrRoutingList, and JrMsgId).
EPERM	The specified routing code requires the user to have superuser authority (JrAuthRoutingCode).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __console() service stores the reason code. The __console() service returns Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Only one thread per address space is allowed to wait on console commands. If the Modify_buffer_ptr is nonzero, there can be no other instance of the

__console() service waiting for console input. This restriction applies to both the multithread and the multiprocess models. Subsequent attempts fail with an EMVSERR and JrNoMulti.

2. Messages sent to the console go to the last console that issued a modify command to this job. If no modify has been issued to this job, the message goes to the console that started this job. If this job was not started (that is, invoker created by the fork service), the message goes to the default console route code.

   Routing codes specified in the message attribute area override the current message routing.
3. An invoker is deemed to have appropriate privileges for this service if the BPX.CONSOLE resource in the FACILITY class is defined and the invoker is permitted to that resource, or if the invoker is running with an effective UID of 0.
4. If the invoker does not have appropriate privileges (see "Authorization" on page 8), a message ID (BPXM023I) and the invoker's login name are prefixed to the specified message text. If the invoker has appropriate privileges, the invoker is responsible for its own message headers. Any message sent to the console should comply with MVS message guidelines. See *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for more information about how to prefix messages with the correct message header. These guidelines are not enforced by this service.
5. The length of the message must be between 1 and 17850 characters for invokers with appropriate privileges, and between 1 and 17780 for invokers without appropriate privileges. The number of lines written to the console is limited to 255. In the case of an unprivileged user, one of those lines is used for the message ID and the invoker's login name. If the message length is exceeded, no lines are written and the service returns an EINVAL. If the number of lines is exceeded, the service returns an EINVAL, but the first 255 lines are written to the console.
6. The __console() service provides limited formatting in that it recognizes the NEWLINE character and attempts to break on word boundaries. If a blank is found within the last 10 characters of the line, the __console() service breaks the line there. If no blanks are found within the last 10 characters, the line break occurs after the 70th character.
7. Use of QEDIT and console service control blocks to listen to console commands, in combination with this service, may result in failures of EMVSERR JrUnexpectedErr.
8. Although the modify string buffer is 128 bytes, the maximum modify string that can be received from the console is less. The largest string that can be typed in from the console is 126 bytes, and this must include the modify command, job name, and APPL= parameters. For example, F SERVER01,APPL= consumes 16 characters of the 126-character string.
9. If the modify_buffer_ptr is specified, the invoking thread waits until either a modify command is issued to this thread's job, or a caught or terminating signal is generated to this thread. The __console() service is also an interrupt point for pthread_cancel.
10. If the console operator enters nothing after the APPL=, the Modify_buffer is unchanged and a Modify_string_length of zero is returned.
11. If the Console_command type returned is CONSOLE_STOP, the Modify_string_length is set to zero. Console stop commands do not pass string data. It is up to the application to handle the stop command; the system takes no action against the process in response to a stop command. The

application may choose to ignore the stop command, or terminate the process through services such as BPX1EXI (BPX4EXI).

12. To use the functions available under console2(), specify the new version (CCA_#Ver02) and the correct length (CCA#Ver2Len) in the CCA when invoking the __console() service.
13. The three __console() operations (WTO, DOM, and WAIT) can be performed in a single request. The order of operations is WTO (issue messages), DOM (delete messages), and WAIT (for a MODIFY or STOP command).

Characteristics and restrictions

There are no restrictions on the use of the __console() service.

Examples

For an example using this callable service, see "BPX1CCS (__console()) example" on page 1129.

__cpl (BPX1CPL) — CPL interface service

Function

The __cpl callable service calculates coupling facility structure sizes required by the CFRM (Coupling Facility Resource Manager) policy through a Web interface.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1CPL,(FunctionCode,
              Bufferlen,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

FunctionCode
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value indicating the type of CPL function requested. The following are the supported values:

1 Request data from available coupling facilities

2 Request a structure size

3 Request a structure size and CFLevel

4 Request a structure size and CFLevel with variable-length parameter list

These values are defined in __cpl.h.

Bufferlen
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the input length of the buffer.

Buffer
Supplied parameter

Type: Structure

Length:
Length specified by Bufferlen

The name of a fullword that represents the buffer in which the __cpl service receives the input parameters from the Web and returns the results of the call.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __cpl service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __cpl service stores the return code. The __cpl service stores a return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The __cpl service may return the following values in the Return_code parameter:

Return code	Explanation
EMVSCPLERROR	A __cpl service request failed. Consult reason_code, which will contain the reason code from the failing z/OS service, to determine the reason the error occurred.
EFAULT	One of the parameters contained an address that was not accessible to the caller.

Return code	Explanation
EINVAL	The FunctionCode parameter contains a value that is not correct, or the input parameter list is built incorrectly. The following reason codes can accompany the return code: JRCPLInvFcnCode, JRCPLInvBuffLen, JRCPLBuffTooSmall, JRCPLInvStrucType, JRCPLFcnReq, JRCPLParmVer.
EPERM	The calling thread's address space is not permitted to the BPX.CF resource in the FACILITY class. The caller's address space must be permitted to the BPX.CF resource profile in the FACILITY class. The following reason code can accompany the return code: JRCPLNotAuth.
ENOSYS	The __cpl service request failed because the system is not at the correct level. The following reason code can accompany the return code: JRCPLCFNotFound.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __cpl service stores the reason code. The __cpl service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS MVS Programming: Sysplex Services Reference* for the reason codes.

Usage notes

There is no 64-bit version of the __cpl callable service.

Characteristics and restrictions

The __cpl service is a privileged service; the caller must have read access to the BPX.CF resource profile in the FACILITY class.

deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage

Function

The deletehfs service deletes a previously loaded program from the storage of the caller's process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1DEL):	31-bit
AMODE (BPX4DEL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1DEL,(Entrypt_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4DEL with the same parameters. The Entrypt_address parameter is a doubleword.

Parameters

Entrypt_address
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

A fullword (doubleword) pointer field that contains an entry point address that was returned by the loadhfs service for a z/OS UNIX program that was loaded into the caller's process.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the deletehfs service returns -1 if it is not successful. If it is successful, the deletehfs service returns zero.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the deletehfs service stores the return code. The deletehfs service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The deletehfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The entrypt_address parameter contains an entry point address that is not valid. The entry point address does not represent a currently loaded program in the caller's process.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the deletehfs service stores the reason code. The deletehfs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A call to BPX1DEL (BPX4DEL) to delete a program from storage may not actually cause the program to be removed from storage. If the program has been loaded more than once, the program remains in storage until BPX1DEL (BPX4DEL) has been called the same number of times that the program was loaded.
2. If a program that is loaded into storage with the loadhfs service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates,
3. When the calling process is being debugged via the Ptrace service, a call to the deletehfs service generates a WastStopFlagDelete Ptrace event to the debugger process.

Related services

- "loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name" on page 333

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1DEL (deleteHFS) example" on page 1134.

exec (BPX1EXC, BPX4EXC) — Run a program

Function

The exec callable service runs a z/OS UNIX executable file that is either a program object or a REXX exec. The exec callable service replaces the current process image that calls the exec service with a new process image for the executable file that is being run.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1EXC):	31-bit
AMODE (BPX4EXC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1EXC,(Pathname_length,
              Pathname,
              Argument_count,
              Argument_length_list,
              Argument_list,
              Environment_count,
              Environment_data_length,
              Environment_data_list,
              Exit_routine_address,
              Exit_parameter_list_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4EXC with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname parameter. The length can be up to 1023 bytes long.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the fully qualified path name of the file to be run. Each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

The path name can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory; the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

Argument_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument_length_list and the Argument_list. If the program needs no arguments, define Argument_count as the name of a fullword that contains 0.

Argument_length_list
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an argument that is to be passed to the specified program. If the program needs no arguments, define Argument_length_list as the name of a fullword (doubleword) that contains 0.

Argument_list
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an argument to be passed to the specified program. Each argument is of the length specified by the corresponding element in the Argument_length_list. If the program needs no arguments, define Argument_list as the name of a fullword (doubleword) that contains 0.

If the target executable file arguments require null terminators, the arguments that are supplied to this service must include the null terminator as part of the data string and the length.

Environment_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for Environment_data_length and Environment_data. If the program needs no environment data, define Environment_count as the name of a fullword that contains 0.

Environment_data_length
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a fullword that gives the length of an environment variable to be passed to the specified program. If the program does not use environment variables, define Environment_data_length as the name of a fullword (doubleword) that contains 0.

Environment_data_list
Supplied parameter

Type: Structure

Length:
Variable, specified by Environment_data_length

The name of a list of 31(64)-bit pointers. Each pointer in the list is the address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length specified by the corresponding element in Environment_data_length. If the program does not use environment variables, define Environment_data_list as the name of a fullword (doubleword) that contains `0`. If the target executable file is a Language Environment-enabled program, the environment variables that are supplied to this service must include the null terminator as part of the data string and length.

`Exit_routine_address`
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be called, define Exit_routine_address as the name of a fullword (doubleword) that contains `0`.

`Exit_parameter_list_address`
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value that is contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be called or does not require parameters, define Exit_parameter_link_address as the name of a fullword (doubleword) that contains `0`. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

`Return_value`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the exec service returns -1 if it is not successful. If it is successful, the exec service does not return.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the exec service stores the return code. The exec service returns Return_code only if Return_value is -1. See *z/OS UNIX*

System Services Messages and Codes for a complete list of possible return code values. The exec service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory named in the Pathname parameter; it may lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type. The following reason code can accompany the return code: JRExecNotRegFile.
EFAULT	A bad address was received as an argument of the call, or the user exit program checked. The following reason code can accompany the return code: JRExecParmErr and JRExitRtnError.
ELOOP	A loop exists in symbolic links encountered during resolution of the Filename argument. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
EMVSSAF2ERR	The executable file is a set-user-ID or set-group-ID file, and the file owner's UID or GID is not defined to RACF.
ENAMETOOLONG	File_name is longer than 1023 characters, or some component of the file name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file name was specified, or one or more of the components of the specified Filename were not found. The following reason codes can accompany the return code: JRExecNmLenZero and JRQuiescing.
ENOEXEC	The specified file has execute permission, but it is not in the proper format to be a process image. Reason_code contains the loader reason code for the error.
ENOMEM	The new process requires more memory than is permitted by the hardware or the operating system. The following reason code can accompany the return code: JRExecFileTooBig.
ENOTDIR	A directory component of Filename is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the exec service stores the reason code. The exec service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For most of the reason codes, see *z/OS UNIX System Services Messages and Codes*. For the ENOEXEC Return_code, Reason_code contains the loader reason code for the error:

Reason code	Explanation
X'xxxx0C27'	The target file is not in the correct format to be an executable file.
X'xxxx0C31'	The target file is built at a level that is higher than that supported by the running system.

Usage notes

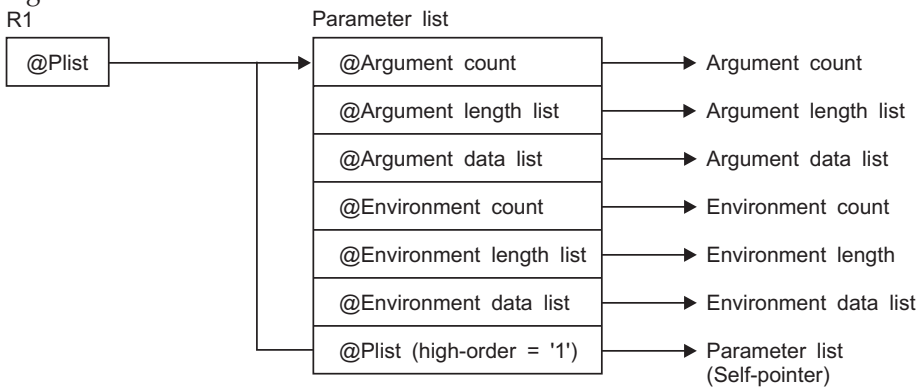
1. The following characteristics of the calling process are changed when the service gives control to the new executable file:
   - The current process image is replaced with a new process image for the executable file to be run.

- All directories that are opened via a call to the opendir service are closed in the new process image.
- All open file descriptors remain open unless the close-on-exec flag is set.
- Signals set to be caught are reset to their default.

  If the SSTFNOSUID bit is set for the file system containing the new process image file, the effective user ID, effective group ID, saved set-user-ID and saved set-group-ID are unchanged in the new process image. Otherwise, if the `setuid` bit of the new process image file is set, the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the `setgid` bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image is saved (as the saved `set-user-ID` and the saved `set-group-ID`) for use by the setuid and setgid functions. For more information, see "BPXYMODE — Map the mode constants of the file services" on page 996.

2. The executable file to be run receives control with the following attributes:
   - Problem program state
   - PSW key 8
   - AMODE=31(64), taken from the executable
   - Primary ASC mode
3. The new process image inherits the following from the calling process image:
   - Process ID
   - Parent process ID
   - The time left until an alarm signal is generated
   - File mode creation mask
   - Process signal mask
   - Pending signals
   - Time accounting information

   For more information, see "times (BPX1TIM, BPX4TIM) — Get process and child process times" on page 856 and "BPXYTIMS — Map the response structure for times" on page 1064.
4. The information that the service passes to the executable file to be run is a parameter list, which is pointed to by register 1. The parameter list consists of the following parameter addresses. In the last parameter address, the high-order bit is 1.



For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always *n* parameters, passed with no end-of-parameter-list indicator.

The last parameter that the exec service passed to the executable file identifies the caller of the file as the exec service. The exit gets control in the same AMODE as the caller.

5. The register usage on entry to the user exit in AMODE 31 is:
   - R0: Undefined.
   - R1: Address of the user exit parameter list, as specified by the caller of the exec service.
   - R2–R12: Undefined.
   - R13: Address of a 96-byte work area in the same key as the caller of the exec service.
   - R14: The return address from the user exit to the exec service. This address must be preserved by the user exit.
   - R15: Address of the user exit.
6. The register usage on entry to the user exit in AMODE 64 is:
   - R0: Undefined.
   - R1: 64-bit address of the user exit parameter list, as specified by the caller of the exec service.
   - R2–R12: Undefined.
   - R13: Address of a 96-byte work area in the same key as the caller of the exec service.
   - R14: The return address from the user exit to the exec service. This address must be preserved by the user exit.
   - R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.
7. When the exec or execmvs service is called in any environment except single task, single RB, and no linkage stack, z/OS UNIX issues an abend EC6. This takes down the calling task and all of its subtasks. The subtasks receive a 33E abend. All other thread tasks in the address space receive a 422 abend with a reason code of 00000181, and their subtasks receive a 33E abend.
8. To support the creation and propagation of a STEPLIB environment to the new process image, the exec service allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable and the actions taken for each value:
   a. STEPLIB=NONE. No STEPLIB DD is to be created for the new process image.
   b. STEPLIB=CURRENT. The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
   c. STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.

Note: The actual name of the DD is not STEPLIB, but a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. The specified data sets must follow standard MVS data set naming conventions. Data sets found to be in violation of this standard are ignored. If the data sets do follow the standard, but:

- The caller does not have the proper security access to a data set
- A data set is uncataloged or is not in load library format

the data set is ignored. Because the data sets in error are ignored, the executable file may run without the proper STEPLIB environment. If a data set is in error due to improper security access, a X'913' abend is generated. The dump for this abend can be suppressed by your installation.

If the STEPLIB environment variable is not specified, the exec service's default behavior is the same as if STEPLIB=CURRENT were specified.

If the program to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For detailed information about the sanction list, see *z/OS UNIX System Services Planning*. For information about STEPLIB performance considerations, see *z/OS UNIX System Services Planning*.

9. The _BPX_JOBNAME environment variable can be used to change the job name of the new process image. The job name change is allowed only if the invoker has appropriate privileges (see "Authorization" on page 8) and is running in an address space that is created by fork. If these conditions are not met, the environment variable is ignored. Accepted values are strings of 1–8 alphanumeric characters. Incorrect specifications are ignored.
10. The _BPX_ACCT_DATA environment variable can be used to change the account data of the new process image. Specifying this environment variable will trigger a new job. The rules for specifying the account data are:
    - Up to 142 actual account data characters are allowed, including any commas.
    - Subparameters must be separated by commas.
    - There is no restriction on the character set.

    If the account data is greater than 142 characters, the data will be ignored. No other validity or syntax checking will be done.
11. Each shared-memory segment attached to the calling process is detached, and the value of the number of processes attached to each detached segment (shm_nattch) is decremented by 1. If this is the last process attached to a shared memory segment and shmctl (BPX1SCT, BPX4SCT) IPC_RMID has been issued for the shared memory segment, the segment will be removed from the system.
12. The semaphore adjustment value, semadj, will be inherited by the new process.
13. A prior loaded copy of a program in the same address space is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
    - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.

- If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the program found is in storage modifiable by the caller, the prior copy is not reused.

14. If the specified file or link resolves to a MVS program that is link-edited AC=1 located in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation. For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file. For an external link, it must be installed with an owning UID of 0. Also, a file with the APF extended attribute is not allowed if found in a file system mounted as NOSETUID. If the specified file name represents a symbolic link to a sticky bit file that has the set-user-id attribute, the symbolic link must have an owning uid of 0 or an owning UID equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning uid of 0 or an owning GID equal to that of the sticky bit file. A file or link found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this set up will cause the executing job to end abnormally with a EC6-xxxxC04A abend when invoking the MVS program via the exec service.
15. If the specified file is not in the correct format to be an executable, but contains the "magic number" (#!) in the first line, the program that is specified in the magic number header is executed. The expected format of the magic number header is as follows:

    ```
    #! Path String
    ```

    `#!` is the file magic number. It identifies the first line of the file as a special header that contains the name of the program to be run and any argument data to be supplied to it.

    The *Path* parameter specifies the path name of the file that is to be run. It is separated by blank or tab characters from the `#!` characters, or can immediately follow the characters.

    The *String* parameter is an optional character string that can be used to pass options to a target command interpreter (shell) that is to run the script. It must be separated from the *Path* parameter by tab or blank characters, and cannot itself contain tab or blank characters.

    The argument data list and argument length list that are passed to the magic number file are to contain the following argument data and corresponding argument data lengths:

    - The magic number path name, ended by a null character (X'00')
    - The string, if one is supplied, ended by a null character (X'00')

    The remaining arguments in the list are to contain the list of arguments that are specified by the caller of the exec service.

    If the path name that is specified in the magic number header cannot be executed for some reason, the exec request fails with return code ENOEXEC, regardless of the error. ENOEXEC is returned for compatibility purposes, so that existing scripts can continue to run successfully when invoked from an application such as a command interpreter (shell). The reason code indicates the exact reason the magic number file could not be executed.
16. If the calling task is in a WLM enclave, the resulting task in the new process image is joined to the same WLM enclave. This allows WLM to manage the old and new process images as one "business unit of work" entity for purposes of system accounting and of management.

17. If the target executable program is an IBM Language Environment-enabled program, the environment variables supplied to the service must include the null terminator as part of the string and length.
18. If the program being executed is APF-authorized, link-edited with AC=1, and is being executed on the job step task, the address space is marked as APF-authorized.
19. If the _BPX_PTRACE_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs loaded into storage during the execution of the target program are also loaded into user-modifiable storage, with the exception of modules loaded from the LPA.
20. The _BPXK_MDUMP environment variable can be used to specify where a SYSMDUMP is to be written. The following are the allowable values:

    Value **Description**

    OFF The dump is to be written to the current directory. This is the default. This dump is only written if the user allocates a SYSMDUMP data set for the TSO/E session. The system creates a file named coredump.*pid* in the user's working directory (where *pid* is the process ID for the process being dumped) and writes the core dump (SYSMDUMP) in hexadecimal format.

    MVS data set name
    The dump is to be written to an MVS data set. The data set name must be fully qualified, and can be up to 44 characters. It can be specified in uppercase, lowercase, or both; it is folded to uppercase.

    z/OS UNIX file name
    The dump is to be written to a z/OS UNIX file. The file name can be up to 1024 characters and must begin with a slash. The slash refers to the root directory, in which the file is created. This specification is ignored for the MODIFY BPXOINIT,DUMP command and for the SIGDUMP signal.
21. The _BPXK_JOBLOG environment variable can be used to specify that WTO messages are to be written to an open job log file. The following are the allowable values:

    Value **Description**

    nn Job log messages are to be written to open file descriptor *nn*.

    STDERR
    Job log messages are to be written to the standard error file descriptor, 2.

    NONE
    Job log messages are not to be written. This is the default.

    The file that is used to capture messages can be changed at any time by calling the oe_env_np service and specifying _BPXK_JOBLOG with a different file descriptor.

    Message capturing is turned off if the specified file descriptor is marked for close on a fork or exec.

    Message capturing is process-related. All threads under a given process share the same job log file. Message capturing may be initiated by any thread under that process.

Multiple processes in a single address space can each have different files active as the JOBLOG file; some or all of them can share the same file; and some processes can have message capturing active while others do not.

When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. Failure to do this causes unpredictable results.

Only files that can be represented by file descriptors may be used as job log files; MVS data sets are not supported.

Message capturing is propagated on a fork() or spawn(). If a file descriptor is specified, the physical file must be the same in order for message capturing to continue in the forked or spawned process. If STDERR is specified, the file descriptor may be remapped to a different physical file..

Message capturing may be overridden on exec() or spawn() by specifying the _BPXK_JOBLOG environment variable as a parameter on the exec() or spawn().

Message capturing only works in forked (BPXAS) address spaces.

This is not true joblog support: messages that would normally go to the JESYSMSG data set are captured, but messages that go to JESMSGLG are not.

22. If the BPXK_SIGDANGER environment variable is set to YES, the process will receive a SIGDANGER signal rather than a SIGTERM signal when an OMVS shutdown is initiated. This may be advantageous for an application that uses the SIGTERM signal for other purposes.
23. When the executable file to be run is a REXX exec, the first argument should be the path name of the REXX exec. Subsequent arguments for the exec can follow this. Each argument should be a string terminated by a null character; that is, the last byte should be X'00'. Each argument length should include this last byte.
24. An environment variable, AUTHPGMLIST, has been created to work with this system call. This environment variable specifies a list that identifies the sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being instantiated is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For details about the sanction list, see the topic on using sanction lists in *z/OS UNIX System Services Planning*.
25. The _BPXK_SUID_FORK environment variable specifies whether the setuid indicator is propagated to child address spaces created by the fork service. For more information, see Commonly used environment variables in *z/OS UNIX System Services Planning*.
26. If the caller specifies _BPXK_DISABLE_SHLIB=YES then future loadhfs() and loadhfs_extended() system calls will ignore the st_sharelib attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information, see the section on commonly used environment variables in *z/OS UNIX System Services Planning*

Related services

- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29
- “chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory” on page 90
- “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 174
- “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185

- "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "spawn (BPX1SPN, BPX4SPN) — Spawn a process" on page 780
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805
- "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866

Note: The exec service is not related to the **exec** shell command.

Characteristics and restrictions

1. The exec service dynamically inserts into a job a new job step that has no allocations associated with it, with the exception of the MVS data sets that may be built into the STEPLIB environment for the new process image.
2. If the exec service is invoked from a process that contains one task, one request block (RB), and no linkage stack entries, the process is ended. This action results in a normal return to the operating system. Almost all forked processes run in this manner. In all other cases, the calling task receives a nonretryable EC6 abend with reason code 0000FFFE to cause it to end. All other thread tasks in the address space that are not subtasks of the calling task receive a 422 abend with reason code 00000181.
3. The user exit is given control while the exec service is still in progress. The user exit should not attempt to use any z/OS UNIX service that alters or terminates the current process (that is, the exec, exit, and kill services). If such services are attempted, the results are unpredictable. Signals cannot be delivered while in the user exit, because the exec service is still in progress and signal delivery is inhibited.
4. If you intend to run a program in an APF-authorized environment, the program that is being run by the exec service should have the APF extended attribute turned on and should be linked AC=1. DLLs that are loaded by APF-authorized applications should have the APF extended attribute set on and should be linked AC=0.
5. Any shared memory segments attached to the caller will not be attached to the newly created process image. Any shared memory segments attached to the caller will be detached and the value of shm_nattch decremented by the number of shared memory segments attached to the caller. If this is the last process attached to a shared memory segment and a shmctl IPC_RMID has been issued, the segment is removed from the system.
6. For semaphore users, when the process exec is issued, the SemAdj values are inherited by the new process image.
7. Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST_PROGCTL attribute is ON for the z/OS UNIX program file). Running a z/OS UNIX program with the ST_PROGCTL attribute set to OFF prevents future invocations of authorized programs like Program Access to Data Sets (PADS) programs. These are programs given special authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.

8. The _BPXK_TIMEOUT environment variable is processed for this callable service. It allows applications to specify whether a specific process waiting for terminal input should be timed out. For more information about the _BPXK_TIMEOUT environment variable, see *z/OS UNIX System Services Planning*.

Examples

For an example using this callable service, see "BPX1EXC (exec) example" on page 1135.

MVS-related information

If the exec service is invoked from an address space containing a single process, it tears down the existing process image by ending the currently running job step and then inserting a new step for the specified file to run in. Any MVS task-related resources that existed in the old job step are cleaned up. The new job step that is created has no allocations associated with it, except for the MVS data sets that may be built into the STEPLIB environment for the new process image. When the newly created job step ends, the flow of the job continues, as it normally does, to the next sequential step in the job, depending on the completion code of the ending step.

If the exec service is invoked from an address space containing multiple processes, the following characteristics apply:

- If the calling process does not have any subtasks that are part of another process, and if the calling process was created via a call to the attach_execmvs or attach_exec service, only the initial thread task of the process and all of its subtasks are terminated, and a new task is attached to the parent process creator task to run the specified program. The initial thread task in such a process is the task that was created as a result of the call to the attach_execmvs or attach_exec service. This call to the exec service does not result in the ending of any other tasks in the calling job step, nor does it end other processes in the same address space.
- If the calling process has any subtasks that are part of another process, or if the calling process was not created via a call to the attach_exec or attach_execmvs service, the exec invocation is not allowed to prevent the unexpected termination of other processes in the address space. The caller receives a return code and reason code detailing the error.

If the exec service is invoked after a successful setuid that changes the MVS identity and the _BPX_JOBNAME environment variable was not specified, the job name of the new process image is set to the user ID associated with the new UID specified on the setuid invocation.

execmvs (BPX1EXM, BPX4EXM) — Run an MVS program

Function

The execmvs service runs an MVS executable program that is in the link pack area (LPA) or LNKLST concatenation. If it is invoked from an address space that contains multiple processes, the program can come from a STEPLIB.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1EXM):	31-bit
AMODE (BPX4EXM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1EXM,(Program_name_length,
              Program_name,
              Argument_length,
              Argument,
              Exit_routine_address,
              Exit_parameter_list_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4EXM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Program_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the name of the MVS program.

Program_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Program_name_length parameter

The name of a field that contains the name of the MVS program that is to be run. The MVS program name must conform to the naming conventions for members of MVS partitioned data sets (PDSs). The program name is from 1 to 8 characters long; the program name is the member name without any qualifiers.

The specified Program_name must be in uppercase.

Argument_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the argument that is to be passed to the program. The argument can be from 0 to 4096 bytes long except for unauthorized callers calling authorized programs. For unauthorized callers calling authorized programs, the argument can be from 0 to 100 bytes long. If you want to allow an unauthorized caller to pass an argument greater than 100 bytes to a program, a BPX.EXECMVSAPF.program_name FACILITY class profile needs to be defined for that program.

Argument
Supplied parameter

Type: Integer

Length:
Specified by the Argument_length parameter

The name of a field of length Argument_length that contains the argument that is to be passed to the MVS program.

The data that is contained in the Argument parameter should not include pointers to private storage. The execmvs service frees all private storage while cleaning up the previous job step.

Exit_routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user's exit routine. If a user exit is not to be invoked, define Exit_routine_address as the name of a fullword (doubleword) that contains `0`.

Exit_parameter_list_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user exit parameter list. The value that is contained in this fullword (doubleword) is in register 1 when the user exit receives control. If the user exit is not to be invoked or does not require parameters, define Exit_parameter_list_address as the name of a fullword (doubleword) that contains `0`. Currently the exit must be RMODE 31, and therefore the address must reside below the 2-gigabyte bar.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the execmvs service returns -1 if it is not successful. If it is successful, the execmvs service does not return.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the execmvs service stores the return code. The execmvs service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The execmvs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
E2BIG	The number of bytes used by the new process image's argument list is greater than the system-imposed limit of 4096 bytes. The following reason code can accompany the return code: JRMVSArgTooBig.
EFAULT	The user exit program checked. The following reason code can accompany the return code: JRExitRtnError.
ENAMETOOLONG	The specified MVS program name is too long. The length that is specified by Program_name_length is longer than 8 bytes.
ENOENT	The specified MVS program was not found in the link pack area or in a link list data set, LNKLST; or the program name argument points to an empty string. STEPLIB needs to be included in a multiprocess environment. The following reason code can accompany the return code: JRExecNmLenZero.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the execmvs service stores the reason code. The execmvs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The following characteristics of the calling process are changed when the new executable is given control by the execmvs service:
   - The prior process image is replaced with a new process image for the executable program that is to be run.
   - All open files that are marked close-on-exec and all open directory streams are closed.
   - All signals that have sigaction settings are reset to their default actions.
2. The input that is passed to the MVS executable file by the service is consistent with the input that is passed to MVS programs. On input, the MVS program receives a single-entry parameter list that is pointed to by register 1. The high-order bit of the sole parameter entry is set to 1.

   The sole parameter entry is the address of a 2-byte length field followed by an argument string. The length field describes the length of the data that follows

it. If a null argument and argument length are specified in the call, the length field specifies 0 bytes on input to the executable file.

3. The call can invoke both unauthorized and authorized MVS programs:
   - Unauthorized programs receive control in problem program state, with PSW key 8.
   - Authorized programs receive control in problem program state, with PSW key 8 and APF authorization.
4. The register usage on entry to the user exit in AMODE 31 is:
   - R0: Undefined.
   - R1: Address of the user exit parameter list, as specified by the caller of the execmvs service.
   - R2–R12: Undefined
   - R13: Address of a 96-byte work area in the same key as the caller of the execmvs service.
   - R14: The return address from the user exit to the execmvs service. This address *must* be preserved by the user exit.
   - R15: Address of the user exit.
5. The register usage on entry to the user exit in AMODE 64 is:
   - R0: Undefined.
   - R1: 64-bit address of the user exit parameter list, as specified by the caller of the execmvs service.
   - R2–R12: Undefined
   - R13: Address of a 96-byte work area in the same key as the caller of the execmvs service.
   - R14: The return address from the user exit to the execmvs service. This address *must* be preserved by the user exit.
   - R15: R15: Information about the caller. Bit 61 is on and bit 62 is off, indicating an AMODE 64 caller. Bit 63 is also off, indicating that the addressing mode should not be changed on return to the caller, and that a BRANCH ON CONDITION (BCR) should be used for the return. The other bits in R15 are not relevant. Because R15 does not contain the address of the exit routine on entry, BRANCH RELATIVE instructions should be used for branching within the user exit.
6. When the exec or execmvs service is called in any environment except single task, single RB, and no linkage stack, z/OS UNIX issues a quiesce_force to terminate all of its subtasks. The subtasks receive a 422 abend with a reason code of 000001A0.
7. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the execmvs service are propagated to the new process image if the data sets that they represent are found to be cataloged. Uncataloged data sets are not propagated to the new process image. This causes the program that is invoked to run with the same MVS program search order as its invoker.
8. To support the creation and propagation of a STEPLIB environment to the new process image, the execmvs service allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable and the actions taken for each value:
   a. STEPLIB=NONE. No Steplib DD is to be created for the new process image.

b. STEPLIB=CURRENT. The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the exec service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.

c. STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.

Note: The actual name of the DD is not STEPLIB, but a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order that is specified. The specified data sets must follow standard MVS data set naming conventions. Data sets that are found to be in violation of this standard are ignored. If the data sets follow the standard, but:

- The caller does not have the proper security access to a data set, or
- A data set is uncataloged or not in load library format

the data set is ignored. Because the data sets that are in error are ignored, the executable file may run without the proper STEPLIB environment. If a data set is in error because of improper security access, a X'913' abend is generated. The dump for this abend can be suppressed by your installation.

If the STEPLIB environment variable is not specified, the default behavior of the execmvs service is the same as if STEPLIB=CURRENT were specified.

If the program that is to be invoked is a set-user-ID or set-group-ID file and the user-ID or group-ID of the file is different from that of the current process image, the data sets that are to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. For detailed information about the sanction list, see Using sanction lists in *z/OS UNIX System Services Planning*. For information about STEPLIB performance considerations, see Tuning performance in *z/OS UNIX System Services Planning*.

9. If the calling task is in a WLM enclave, the new process image task is joined to the same WLM enclave. This allows WLM to manage the old and new process images as one "business unit of work" entity for system accounting and management purposes.

10. If an unauthorized caller attempts to pass an argument greater than 100 bytes to an authorized program, the caller will abend with ABENDEC6 reason code 0B26C048 pr ABEND306 reason code 44. If the caller needs to be allowed to execute the program with an argument greater than 100 bytes, then a BPX.EXECMVSAPF.program_name profile should be defined or have the program owner rebind the program specifying the LONGPARM option.

Related services

- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132

Characteristics and restrictions

- When the execmvs service is called from any process except one that was created via the attach_exec or attach_execmvs service, the program must be located either in the link pack area (LPA) or in a link list data set (LNKLST).
- When the execmvs service is called from a process that was created via the attach_exec or attach_execmvs service, the specified program can be located in

the link pack area, in a link list data set, job library, step library, or task library. The program search order that is followed is identical to that of the MVS Attach service when the EP parameter is specified.

- If the execmvs service is invoked from a process that contains one task, one request block (RB), and no linkage stack entries, the process is ended by an SVC 3 instruction. This action results in a normal return to the operating system. Almost all forked processes run in this manner. In all other cases, the system ends all tasks (threads) in the caller with a nonretryable 422 abend, reason code 000001A0.
- The user exit cannot invoke any z/OS UNIX services. If it attempts to invoke a z/OS UNIX service, the service fails or the caller is abended, depending on the service that is attempted. Signals cannot be delivered to the caller of the exec service while the user exit is in control.
- The program that is invoked by the execmvs service must be enabled to run in 31-bit addressing mode (AMODE=31).

Examples

For an example using this callable service, see "BPX1EXM (execmvs) example" on page 1136.

MVS-related information

Because the service must create a new process image for the specified program to run within, the prior process image is completely cleaned up. In MVS terms, the system ends a step within a job and then inserts a new step for the specified program to run in. Any MVS task-related resources that existed in the old job step are cleaned up. The new job step that is created has no allocations associated with it, with the exception of the MVS data sets that may be built into the STEPLIB environment for the new process image. When the newly created job step ends, the flow of the job continues, as it normally does, to the next sequential step in the job, depending on the completion code of the ending step.

_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup

Function

The _exit callable service ends the calling thread task and all its subtasks. In most environments, this results in the ending of the process, with the specified status being reported to its parent.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1EXI):	31-bit
AMODE (BPX4EXI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1EXI,(Status_field)
```

AMODE 64 callers use BPX4EXI with the same parameter.

Parameters

Status_field
Supplied parameter

Type: Structure

Length:
4 bytes

The name of a 4-byte status field. If the call to _exit results in a process ending and contents of the status field conform to the allowable exit status values, the service provides the contents to the parent when a wait service is issued. For a mapping of the status field and a description of the conforming status values, see "BPXYWAST — Map the wait status word" on page 1069.

Usage notes

1. A call to _exit results in the ending of the calling task and all its subtasks, and the cleaning up of their associated MVS and z/OS UNIX resources. In most environments, this results in the ending of the calling process.
2. In some environments the call to _exit does not result in a process ending. An example of such an environment is the TSO/E TMP environment, where multiple MVS tasks can be concurrently dubbed as threads. A call to the _exit service from one of these threads results only in the ending of the calling thread task and its subtasks. In such an environment, if only one task is currently dubbed as a thread, a call to the _exit service from this thread task ends the process.
3. The ending of a process results in the following actions:
   - All file descriptors and directory streams that are open in the ending process are closed. Open file descriptors are inherited by the child. Literally speaking, the child did not open the file, yet it will still be closed.
   - If the parent of the ending process has issued a wait call and is waiting for the ending process to end, has not used sigaction to set its SA_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the status is returned to the parent at once.

     If the parent of the ending process is not waiting, has not used sigaction to set its SA_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the status is saved. It is returned to the parent if the parent later issues a wait call for the now-ended child.
   - If the parent of the ending process has set the SA_NOCLDWAIT flag for the SIGCHLD signal, or has set the action for SIGCHLD to ignore, the status is discarded and will not be seen by the parent if the parent issues a wait. The ending process is assigned the parent process ID of the initialization process (whose process ID is 1) that frees the PID and system resources associated with the ending process.

     If the parent of the ending process does not later wait for the ending process, and has not used sigaction to set its SA_NOCLDWAIT flag for the SIGCHLD signal, and has not set the action for SIGCHLD to ignore, the ending

process's ID (PID) remains in use until the parent ends. Because the number of process IDs is a limited system resource, user and system availability for process IDs may be affected.

- If the ending process is a session leader, the controlling terminal is disassociated from the session. The controlling terminal can then be acquired by a new controlling process.
- Child processes of a process that ends are assigned the parent process ID of the initialization process (whose process ID is 1). The status of these child processes is reported to the initialization process that frees the PID and system resources associated with the ending process.
- A **SIGCHLD** signal is sent to the parent of the ending process.
- Ending a process does not end its child processes directly, however; under the following circumstances a **SIGHUP** signal is sent to a child process that can cause a child process to end:
  - If the ending process is a controlling process, a **SIGHUP** signal is sent to each process in the foreground process group of the controlling terminal belonging to the caller.
  - If the ending process is a dubbed process that has not been a controlling process of a terminal session—for example, a batch job step that has issued z/OS UNIX service calls—a **SIGHUP** signal is sent to each process in the ending process's process group.
  - If ending a process leaves a process group orphaned and any member of that process group is stopped, each member of the process group is sent a **SIGHUP** signal followed by a **SIGCONT** signal.

4. If the ending of the calling task results in the ending of a job step, the specified status code is used as the completion code for the ending job step.
5. The _exit service does not return to the caller. If it cannot complete its processing successfully, the caller receives an EC6 abend.
6. If the caller specifies an incorrect exit status value, the caller receives an EC6 abend with an appropriate reason code identifying the error.
7. If you are going to use this service in a multiple-pthread environment, see Appendix H, “Using threads with callable services,” on page 1321.
8. Each shared-memory segment attached to the calling process is detached, and the value of the number of processes attached to each detached segment (shm_nattch) is decremented by 1. If this is the last process attached to a shared memory segment and shmctl IPC_RMID has been issued for the shared memory segment, the segment is removed from the system.
9. When the process is terminated, the semadj values are applied to the semaphores. Adjustments to each semaphore set are made atomically.

Related services

- “close (BPX1CLO, BPX4CLO) — Close a file” on page 103
- “mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources” on page 418
- “wait (BPX1WAT, BPX4WAT) — Wait for a child process to end” on page 882

Note: The _exit service is not related to the **exit** shell command and is different from the **exit()** ANSI C routine.

Characteristics and restrictions

If the _exit service is invoked with a normal exit status completion code from a task that has no subtasks, one request block (RB), and no linkage stack entries, the

task ends with an SVC 3 instruction. This action results in a normal return to the operating system. Almost all forked processes end in this manner. In all other cases, the calling task receives a nonretryable EC6 abend with a reason code that varies with the type of exit status specified. If the exit status value indicates that the process is to end with:

- A normal exit status code, an abend reason code of 0000FFFF is received.
- An ending signal, an abend reason code of 0000FFxx is received, where *xx* is the signal number specified in the exit status.
- A terminating signal with a core dump to be taken, an abend reason code of 0000FDxx is received, where *xx* is the signal number specified in the exit status.

All subtasks of the calling thread task receive a 33E abend when the calling thread task is abended.

If the calling thread task was created with the pthread_create service, the initial pthread-creating task abends with a 422 abend code, and reason code 000001xx. The value of *xx* is the signal number if signal exit status is specified, or 82 if a normal exit status is specified.

For a detailed description of the conforming exit status values see “BPXYWAST — Map the wait status word” on page 1069.

Examples

For an example using this callable service, see “BPX1EXI (_exit) example” on page 1136.

extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link

Function

The extlink_np service creates a symbolic link to an external name. A file named Link_name, of type "symbolic link" is created. The content of the symbolic link file is the external name specified in Ext_name.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1EXT):	31-bit
AMODE (BPX4EXT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1EXT,(Ext_name_length,
              Ext_name,
              Link_name_length,
              Link_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4EXT with the same parameters.

Parameters

Ext_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the length of Ext_name. The Ext_name can be up to 1023 bytes long.

Ext_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Ext_name_length parameter

The name of a field containing the external name for which you are creating a symbolic link. An external name is the name of an object outside the hierarchical file system.

Link_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the length of Link_name. The Link_name can be up to 1023 bytes long; each component of the name (between delimiters) can be up to 255 bytes long.

Link_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by Link_name_length parameter

The name of a field containing the symbolic link being created.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the extlink_np service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the extlink_np service stores the return code. The extlink_np service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The extlink_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search a directory in the Link_name, or does not have permission to write in the directory to contain the symbolic link file.
EEXIST	Link_name already exists.
EINVAL	Parameter error. Possible reasons are: • Ext_name_length exceeds the maximum allowed. • Ext_name_length is zero. • Link_name has a slash as its last component, which indicates that the preceding component is a directory. A symbolic link cannot be a directory. The following reason codes can accompany the return code: JRInvalidSymLinkLen, JREndingSlashSymLink.
ELOOP	A loop exists in symbolic links encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
ENAMETOOLONG	Link_name is longer than 1023 characters, or some component of that name is longer than 255 characters. Name truncation is not supported.
ENOSPC	The directory in which the entry for the symbolic link is being placed cannot be extended; not enough space remains in the file system.
ENOTDIR	A component of the path prefix of Link_name is not a directory.
EROFS	The requested operation requires writing in a directory on a read-only file system.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the extlink_np service stores the reason code. The extlink_np service returns a Reason_code only if Return_value is -1.

Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The extlink_np service creates an external symbolic link (Link_name) with the object you specify by Ext_name.
2. The object identified by Ext_name need not exist when the symbolic link is created, and refers to an object outside a hierarchical file system.
3. The external name contained in an external symbolic link is not resolved. The Link_name cannot be used as a directory component of a pathname.

Related services

- "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349
- "readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link" on page 587
- "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1EXT (extlink_np) example" on page 1136.

fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor

Function

The fchattr service modifies the attributes that are associated with a file. It can be used to change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, file size, and file tag. It can also be used to set the initial security label for a file or directory. You identify the file by its file descriptor.

For the corresponding service using a pathname, see "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCR):	31-bit
AMODE (BPX4FCR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation
Control parameters:

Environment
All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCR,(File_descriptor,
              Attributes_length,
              Attributes,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCR with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the file descriptor of the file whose attributes you want to change.

Attributes_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the length of the area containing the attributes you want to change.

Attributes
Supplied parameter

Type: Structure

Length:
Specified by the Attributes_length parameter

The name of the area containing the attributes you want to change. The area is mapped by BPXYATT. For information on the content of this area, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchattr service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fchattr service stores the return code. The fchattr service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fchattr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process did not have appropriate permissions. Possible reasons include: • The calling process was attempting to set access time or modification time to current time; the effective UID of the calling process does not match the owner of the file; the process does not have write permission for the file; or the process does not have appropriate privileges (see "Authorization" on page 8). • The calling process was attempting to change the file size; the calling process does not have write permission for the file.
EBADF	The File_descriptor parameter is not a valid file descriptor.
EFBIG	Attempting to change the size of a file, the specified length is greater than the maximum file size limit for the process. The following reason code can accompany the return code: JRWriteBeyondLimit.
EINVAL	The length of the Attributes parameter is too small, or the Attributes structure containing the requested changes is not valid. The following reason codes can accompany the return code: JrInvalidAtt, JrNegativeValueInvalid, JrTrNotRegFile, JrTrNegOffset, JrFileNotEmpty, and JrInvalidFileTag.
EMVSERR	An MVS environmental error has been detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
ENOSYS	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.

Return_code	Explanation
EPERM	The operation is not permitted for one of the following reasons: • The calling process was attempting to change the mode or the file format; the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see "Authorization" on page 8). • The calling process was attempting to change the owner, and the calling process does not have appropriate privileges. • The calling process was attempting to change the general attribute bits, and the calling process does not have write permission for the file. • The calling process was attempting to set a time value (not current time); the effective user ID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. • The calling process was attempting to set the change time or reference time to current time, and the calling process does not have write permission for the file. • The calling process was attempting to change auditing flags; the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. • The calling process was attempting to change the security auditor's auditing flags, and the user does not have auditor authority. • Attributes indicate that the security label is to be set, and one or more of the following conditions apply: – The calling process does not have RACF SPECIAL authorization and appropriate privileges. – There is already a security label associated with the file.
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchattr service stores the reason code. The fchattr service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

Table 3. Attribute fields modifiable by fchattr

Set flags	Attribute fields input	Description
ATTMODECHG	ATTMODE	Set the mode according to the value in ATTMODE. See "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169.

Table 3. Attribute fields modifiable by fchattr (continued)

Set flags	Attribute fields input	Description
ATTOWNERCHG	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93.
ATTSETGEN	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits will be unchanged.
ATTTRUNC	ATTSIZE	Change the file size to ATTSIZE bytes. See "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203.
ATTATIMECHG	ATTATIME	If ATTLP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
ATTATIMETOD	None	Set the access time of the file to the current time.
ATTMTIMECHG	ATTMTIME	If ATTLP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
ATTMTIMETOD	None	Set the Modification time of the file to the current time.
ATTMAAUDIT	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164.
ATTMUAUDIT	ATTUSERAUDIT	Set the User's auditing flags to the value specified in ATTUSERAUDIT. See "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164.
ATTCTIMECHG	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.

Table 3. Attribute fields modifiable by fchattr (continued)

Set flags	Attribute fields input	Description
ATTCTIMETOD	None	Set the Change Time of the file to the current time.
ATTREFTIMECHG	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
ATTREFTIMETOD	None	Set the Reference Time of the file to the current time.
ATTFILEFMTCHG	ATTFILEFMT	Set the File Format of the file to the value specified in ATTFILEFMT.
ATTCHARSETIDCHG	ATTFILETAG	Set the file tag. See BPXYSTAT (“BPXYSTAT — Map the response structure for stat” on page 1057) for file tag mapping.
ATTSECLABELCHG	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes should be updated. To set an attribute, turn the corresponding **Set Flag** on, and set the corresponding **Attributes Field** according to Table 3 on page 159. Multiple attributes may be changed at the same time.

   The **Set Flag** field should be cleared before any bits are turned on. It is considered an error if any of the reserved bits in the flag field are turned on.
2. Some of the attributes changed by the fchattr service can also be changed by other services. See the related service (listed in Table 3 on page 159) for a detailed description.
3. Changing mode (ATTMODECHG = ON):
   - The file mode field in Attributes is mapped by the BPXYMODE macro (see “BPXYMODE — Map the mode constants of the file services” on page 996). For information on the values for file type, see “BPXYFTYP — File type definitions” on page 967.
   - File descriptors that are open when the fchattr service is called retain the access permission they had when the file was opened.
   - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
   - Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

     The set-group-ID-on-execution permission is set to zero if both of the following are true:
     – The caller does not have appropriate privileges.
     – The GID of the file's owner does not match the effective GID or one of the supplementary GIDs of the caller.

- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

4. Changing owner (ATTOWNERCHG = ON):
   - To change the owner UID of a file, the caller must have appropriate privileges.
   - To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:
     – The effective UID of the caller matches the file's owner UID.
     – The Owner_UID value specified in the change request matches the file's owner UID.
     – The Group_ID value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
   - When owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
   - When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other can be set to its present value or to -1 to remain unchanged.
5. Changing general attribute bits (ATTSETGEN = ON):
   - For General Attribute bits to be changed, the calling process must have write permission for the file.
6. Changing the size of a file (ATTTRUNC = ON):
   - The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was previously larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was previously shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

     Full blocks are returned to the file system so that they can be used again.

     The file offset is not changed.
   - When a file is changed successfully, it clears the set-user-ID, the set-group-ID and the save-text (sticky bit) attributes of the file unless the caller has appropriate privileges.
   - The changing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, will fail with EFBIG and the SIGXFSZ signal will be generated for the process.
   - If write access is removed at some time after the File_descriptor was opened for writing, a change request will fail with EACCES. In such a case, a call to “ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file” on page 203 could be used to change the file size.
7. Changing times:
   - All time fields in Attributes are in POSIX format.
   - For the Access Time or the Modification Time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
   - For the Access Time or Modification Time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.

- For the Change Time or the Reference Time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the Change Time or Reference Time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.
- For any time field (atime, mtime, ctime, reftime), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time will be set.
- When any attribute field is changed successfully, the file's change time is updated as well.

8. Changing auditor audit flags (ATTMAAUDIT = ON):
   - For auditor audit flags to be changed, the user must have auditor authority. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

     Auditor authority is established by issuing the TSO/E command ALTUSER Auditor.
9. Changing user audit flags (ATTMUAUDIT = ON):
   - For the user audit flags to be changed, the user must have appropriate privileges (see "Authorization" on page 8) or be the owner of the file.
10. Changing file format (ATTFILEFMTCHG = ON):
    - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
    - The attribute that is specified in ATTFILEFMT is the same attribute that is set by the FILEDATA=TEXT parameter on a DD statement.
11. Changing the file tag (ATTCHARSETIDCHG=ON):
    - A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.
12. Changing the security label (ATTSECLABELCHG=ON):
    - For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see "Authorization" on page 8), and no security label must currently exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.

Related services

- "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76
- "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164
- "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169
- "fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor" on page 171
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196
- "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203

- "truncate (BPX1TRU, BPX4TRU) — Change the size of a file" on page 859
- "utime (BPX1UTI, BPX4UTI) — Set file access and modification times" on page 879
- "lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link" on page 315

Characteristics and restrictions

1. The ATTEXTLINK flag in the ATTGENVALUE field of BPXYATT cannot be modified with fchattr.
2. The General Attribute bits (set by ATTSETGEN, ATTGENMASK, and ATTGENVALUE fields) are not intended as a general-use programming interface to fchattr.
3. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges. It cannot be used to change an existing security label; it can only be used to set an initial security label on a file.

Examples

For an example using this callable service, see "BPX1FCR (fchattr) example" on page 1138.

fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor

Function

The fchaudit callable service changes the types of access to a file to be audited for the security product. You identify the file by its file descriptor.

For the corresponding service using a pathname, see "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCA):	31-bit
AMODE (BPX4FCA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCA,(File_descriptor,
              Audit_flags,
              Option_code,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCA with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the file descriptor of the file to be changed.

Audit_flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword indicating the access to be audited. This field is mapped by the BPXYAUDT macro; see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. Values for this field include any combination of the following:

Value	Description
AUDTREADFAIL	Audit failing read requests.
AUDTREADSUCCESS	Audit successful read requests.
AUDTWRITEFAIL	Audit failing write requests.
AUDTWRITESUCCESS	Audit successful write requests.
AUDTEXECFAIL	Audit failing execute or search requests.
AUDTEXECSUCCESS	Audit successful execute or search requests.

Option_code
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates whether you are changing the auditing for flags of the user or of the auditor. When this field has the value:

- 0: User audit flags are changed.
- 1: Auditor audit flags are changed.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchaudit service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

Return_code	Explanation
EBADF	The File_descriptor parameter is not a valid file descriptor.
EINVAL	The Option_code parameter is incorrect, or File_descriptor refers to an unnamed pipe and fchaudit is not allowed on such a file.
EPERM	The effective user ID of the calling process does not match the owner of the file, the calling process does not have appropriate privileges (see "Authorization" on page 8), or if Option_code indicated that the auditor audit flags were to be changed, then the user may not have had auditor authority.
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchaudit service stores the reason code. The fchaudit service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If Option_code indicates that the auditor audit flags are to be changed, the user must have auditor authority for the request to be successful. The user with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

   You can get auditor authority by issuing the TSO/E command ALTUSER Auditor.
2. If Option_code indicates that the user audit flags are to be changed, the user must have appropriate privileges (see "Authorization" on page 8), or be the owner of the file.

Related services

- "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the fchaudit service.

Examples

See “BPX1FCA (fchaudit) example” on page 1137 for an example using this callable service.

fchdir (BPX1FCD, BPX4FCD) — Change the working directory

Function

The fchdir service changes your working directory from the current one to a new one. The working directory is the starting point for path searches of pathnames not beginning with a slash.

For corresponding service using a pathname, see “chdir (BPX1CHD, BPX4CHD) — Change the working directory” on page 88.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCD):	31-bit
AMODE (BPX4FCD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCD,(Directory_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCD with the same parameters.

Parameters

Directory_file_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the directory file descriptor that was returned when the directory was opened (see “opendir (BPX1OPD, BPX4OPD) — Open a directory” on page 452), which is to become the new working directory. It may also be specified as the name of a fullword containing the file descriptor of an open directory (see “open (BPX1OPN, BPX4OPN) — Open a file” on page 447).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchdir service returns `0` if the request is successful, or -1 if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fchdir service stores the return code. The fchdir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fchdir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have search permission for the directory referenced by the file descriptor.
EBADF	The file descriptor parameter is not a valid file descriptor.
ENOTDIR	The open file descriptor does not refer to a directory. The following reason code can accompany the return code: JRChdNotDir.
EINTR	A signal was caught during the execution of fchdir().
EIO	An I/O error occurred while reading from or writing to the file system. The following reason codes can accompany the return code: JRQuiescing.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchdir service stores the reason code. The fchdir service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Related services

- "chdir (BPX1CHD, BPX4CHD) — Change the working directory" on page 88
- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the fchdir service.

Examples

For an example using this callable service, see "BPX1FCD (fchdir) example" on page 1137.

fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor

Function

The fchmod service modifies the permission bits used to control the owner access, group access, and general access to a file. It can be used to set flags that modify the user ID (UID) and group ID (GID) of the file when it is executed. It can also be used to set the sticky bit to indicate where the file should be fetched from. You identify the file by its file descriptor.

For the corresponding service using a pathname, see "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCM):	31-bit
AMODE (BPX4FCM):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCM,(File_descriptor,
              Mode,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCM with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the name of a fullword containing the file descriptor of the file whose mode you want to change.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

Specifies the name of a fullword in which the mode field is specified. The mode field, mapped by BPXYMODE, specifies the file type and the permissions you grant to yourself, to your group, and to any user. See "BPXYMODE — Map the mode constants of the file services" on page 996 for the parameter options.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

Specifies the name of a fullword to which the fchmod service returns `0` if successful, or -1 if not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fchmod service stores the return code. The fchmod service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fchmod service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter is not a valid file descriptor.
EPERM	The effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fchmod service stores the reason code. The fchmod service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. File descriptors open at the time of the call to the fchmod service retain the access permission they had at the time the file was opened.
2. For mode bits to be changed, the effective UID of the calling process must match the file's owner UID, or the process must have appropriate privileges (see "Authorization" on page 8).
3. When the mode is changed successfully, the file's change time is updated as well.
4. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec call, the effective GID of the process is set to the file's owner GID, so that the process seems to be running under the GID of the file, rather than that of the actual invoker.

   The set-group-ID-on-execution permission is suppressed (the bit is turned off) if both of the following are true:

   - The calling process does not have appropriate privileges.
   - The file's owner GID does not match the effective GID or one of the supplementary GIDs of the calling process.
5. Setting the set-user-ID-on-execution permission means that when this file is run the process's effective UID will be set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the fchmod service.

Examples

For an example using this callable service, see "BPX1FCM (fchmod) example" on page 1138.

fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor

Function

The fchown callable service changes the owner, group, or both owner and group of a file. You identify the file by its file descriptor.

For the corresponding service using a pathname, see "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCO):	31-bit
AMODE (BPX4FCO):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCO,(File_descriptor,
              Owner_UID,
              Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCO with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the file for which you wish to change the owner, group, or both owner and group.

Owner_UID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new owner UID assigned to the file, or the present value or -1 if there is no change. This parameter must be specified.

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new group ID (GID) to be assigned to the file, or the present value or -1 if there is no change. This parameter must be specified.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the fchown service returns `0` if the request is successful, or -1 if it is unsuccessful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fchown service stores the return code. The fchown service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fchown service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter is not a valid file descriptor.
EINVAL	The Owner_UID or Group_ID parameter is incorrect; or File_descriptor refers to an unnamed pipe, and fchown is not allowed on such a file.
EPERM	The calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fchown service stores the reason code. The fchown service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The fchown service changes the owner UID and owner GID of a file. Only a process with appropriate privileges (see "Authorization" on page 8) can change the owner UID of a file.
2. The owner GID of a file can be changed by a process if the process has appropriate privileges, or if a process meets all of these conditions:
   - The effective UID of the process matches the file's owner UID.
   - The Owner_UID value specified in the change request matches the file's owner UID.
   - The Group_ID value specified in the change request is the effective GID, or one of the supplementary GIDs, of the calling process.

3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner_UID and Group_ID must be specified. If you want to change only one of these values, you must set the other to its present value or to -1 in order for it to remain unchanged.

Related services

- "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93
- "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196

Characteristics and restrictions

There are no restrictions on the use of the fchown service.

Examples

See "BPX1FCO (fchown) example" on page 1138 for an example using this callable service.

fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors

Function

The fcntl callable service performs general control functions for open files: it retrieves or sets file descriptor flags, file status flags, locking information, and file tags. It also controls the automatic conversion of text data within files.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FCT):	31-bit
AMODE (BPX4FCT):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FCT,(File_descriptor,
              Action,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FCT with the same parameters. However, for AMODE 64 callers, the Argument parameter may be either a 64-bit pointer or a 4-byte value, depending upon the Action parameter.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the file. This parameter must specify an opened file descriptor, except when the Action parameter is F_CLOSFD, in which case this file descriptor is not expected to be in use.

Action
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains an integer value, mapped in the BPXYFCTL macro, that indicates the action to be performed. For a list of actions, see "BPXYFCTL — Command values and flags for fcntl" on page 966.

Argument
Parameter supplied and returned

Type: Structure

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains an argument, or zero. The type of argument depends upon the action requested:

Action Argument

F_CLOSFD
File_descriptor_2

F_CONTROL_CVT
fcntl convert (F_CVT) structure

F_DUPFD
File_descriptor_2

F_DUPFD2
File_descriptor_2

F_GETFD
0

F_GETFL
0

F_GETLK
Lock_information

F_GETOWN
0

F_SETFD
File_descriptor_flags

F_SETFL
File_status_flags

F_SETLK
Lock_information

F_SETLKW
Lock_information

F_SETOWN
Pid

F_SETTAG
File_Tag

For AMODE 64 callers using F_SETLK, F_GETLK, F_SETLKW, F_SETTAG, or F_CONTROL_CVT, the Argument is a 64-bit pointer. For AMODE 31 callers using F_SETLK, F_GETLK, F_SETLKW, F_SETTAG, or F_CONTROL_CVT, the argument is a 31-bit pointer.

Argument Options
The options you can use as an argument follow:

File_descriptor_2
The name of a fullword that contains a file descriptor.

When Action is F_DUPFD, fcntl returns the lowest file descriptor equal to or greater than File_descriptor_2 that is not already associated with an open file. File_descriptor is duplicated.

When Action is F_DUPFD2, the file descriptor that is returned is equal to File_descriptor_2. File_descriptor_2 is closed if it is already in use. F_CLOEXEC is cleared.

File_descriptor is duplicated. If File_descriptor is equal to File_descriptor_2, the F_DUPFD2 action returns File_descriptor_2 without closing it. F_CLOEXEC is not cleared.

When Action is F_CLOSFD, File_descriptor_2 specifies the upper limit for the range of file descriptors to be closed, and File_descriptor specifies the lower limit. If a -1 is specified for File_descriptor_2, all file descriptors greater than or equal to the lower limit are closed.

File_descriptor_flags
The name of a fullword that contains the file descriptor flags that are to be set or retrieved for File_descriptor.

To get File_descriptor_flags, specify action F_GETFD. If the action is successful, Return_value maps to the bit settings of File_descriptor_flags

Similarly, to set File_descriptor_flags, specify action F_SETFD and use the mapping to set or reset File_descriptor_flags to the desired value.

Note: After the FCTLCLOFORK flag has been set on, it cannot be set off again.

File descriptor flags are mapped by the BPXYFCTL macro; see "BPXYFCTL — Command values and flags for fcntl" on page 966.

File_status_flags
The name of a fullword that contains the file status flags to be set or retrieved for File_descriptor.

To get File_status_flags, specify action F_GETFL. If the action is successful, Return_value maps to the bit settings of File_status_flags

Similarly, to set File_status_flags, specify action F_SETFL and use the mapping to set or reset File_status_flags to the desired value. Only the O_ASYNCSIG, O_APPEND, O_NONBLOCK, and O_SYNC flags are set when Action is F_SETFL; any other flags specified are ignored.

File status flags are used to set some of the open flags that are mapped by the BPXYOPNF macro. For the mapping of the file status flags, see "BPXYOPNF — Map flag values for open" on page 1004.

Two masks are available for use with the return value from an F_GETFL request. You can use the O_ACCMODE mask to extract the file access mode flags from the return value, or you can use the O_GETFL mask to extract both the file access mode and the file status flags.

Lock_information
The name of a fullword (doubleword) that contains a pointer to a structure that contains information about a file segment for which locks are to be set, cleared, or queried.

The Lock_information is mapped by the BPXYBRLK macro as follows:

Word	Description
0	l_type: Bytes 0–1 specify the type of lock that is being set, cleared, or queried. For more information, see "File Locking" in the usage notes.
0	l_whence: Bytes 2–3 specify how the lock offset is to be determined. For more information, see "File Locking" in the usage notes.
1–2	l_start specifies the starting byte offset of the lock that is to be set, cleared, or queried. This is a doubleword value.
3–4	l_len specifies the length of the byte range that is to be set, cleared, or queried. This is a doubleword value.
5	l_pid: On return from a F_GETLK request, this field contains the process ID of the process that is holding the blocking lock, if one was found.

For more information, see "File Locking" in the usage notes.

Pid
The name of a fullword that contains either the process ID or the process group ID that is to receive the SIGIO or SIGURG signals for the socket that is associated with `File_descriptor`.

Every socket has an associated process group number, which is initialized to zero. You set it by calling the fcntl service and specifying the `F_SETOWN` action. This value can also be set using the w_ioctl callable service. The `Argument` value for the `F_SETOWN` can be a positive integer, specifying a process ID, or a negative integer (other than -1), specifying a process group ID. The `F_GETOWN` command returns in the return value field either the process ID or the process group ID that is associated with the socket. The difference between specifying a process ID and specifying a process group ID is that in the first case only a single process receives the signal, while in the second case all processes in the process group receive the signal. The `F_SETOWN` and `F_GETOWN` actions are only available for AF_INET stream sockets.

File_Tag
The name of a fullword (doubleword) that contains a pointer to a file tag. The file tag is mapped in BPXYSTAT ("BPXYSTAT — Map the response structure for stat" on page 1057).

When Action is F_SETTAG, the fcntl service sets the file tag attributes for the file. The file must be a regular, FIFO, or character special file and must be opened in write mode. The file must be empty. If the file is not empty and the DeferTag bit is set, no error is returned and no processing occurs, assuming that the command would otherwise have worked. This allows the caller to issue F_SETTAG without checking the file size, but not incur an error. If you use F_SETTAG to set a tag that is already tagged and opened, O_TRUNC is ignored.

When the DeferTag bit is off, the file tag is set immediately. When the DeferTag bit is on, the setting of the file tag is deferred until the first write by a call to BPX1WRT (BPX4WRT). The file tag is lost if no write ever occurs and the file is closed. If the write fails, file tagging might or might not have occurred. When the file is a FIFO or pipe, the file tag is deferred until the first read (BPX1RED/BPX4RED) or first write (BPX1WRT/BPX4WRT), whichever comes first. This is because a read can precede a write when blocking is enabled, even for an empty file.

If the file is `/dev/null`, `/dev/random`, `/dev/urandom`, or `/dev/zero`, the file tag is not hardened to disk.

Recommendation: Using F_SETTAG multiple times with deferred tagging before the first write to the file is not recommended. Be aware that there are C-RTL environment options that may cause F_SETTAG with deferred tagging (such as FILETAG(,AUTOTAG)).

fcntl convert structure
The name of a two-word structure that describes how conversion is to occur for this file. The two-word structure is mapped in BPXYFCTL ("BPXYFCTL — Command values and flags for fcntl" on page 966; see F_CVT). The first word is one of four possible subcommands, followed by a 2-byte program CCSID and a 2-byte file CCSID.

When Action is F_CONTROL_CVT, the fcntl service controls how conversion occurs when the opened file is being read from (via BPX1RED or BPX4RED) or written to (via BPX1WRT or BPX4WRT). The file must be a regular, FIFO, or character special file.

The subcommands are:

Subcommand
Description

SetCvtAll
Behaves the same as SetCvtOn, except automatic conversion is set to ALL, which enables UNICODE conversion. ThliCcsid is not used. SetCvtAll is ignored if I/O for the file has already started. A thread can set different program CCSIDs for each open file. However, an I/O error will result if any two threads have different program CCSIDs for the same open file that is shared by those two threads.

SetCvtOff
Turns off any conversion that may be in effect. The CCSID values are ignored. If automatic conversion is set to ALL and I/O has already started for the file, this command is ignored.

SetCvtOn
Turns automatic conversion on for the stream and, optionally, sets the program CCSID or file CCSID, or both. A hex value of 0 for program-ccsid indicates using the current program CCSID at the time of each read or write. The current program CCSID is initially 1047, but can be reset directly by the program, or indirectly by setting the appropriate runtime option or environment variable.

A hex value of 0 for the file CCSID indicates that the current setting is not to be changed. The values do not affect the stored file tag or program CCSID; they only change the values that are being used to control conversion on this data stream.

Setting or referencing ThliCcsid is still valid, but no longer recommended.

SetAutoCvtAll
If conversion is enabled for the environment (by BPXPRMxx parmlib statement AUTOCVT setting of ALL or with the appropriate environment variable), this subcommand behaves identically to SetCvtAll. Otherwise, it has no effect.

SetAutoCvtOn
If conversion is enabled for the environment (by AUTOCVT in BPXPRMxx or with the appropriate environment variable), this subcommand behaves identically to SetCvtOn. Otherwise, it has no effect.

QueryCvt
Returns information about whether conversion is in effect, and the program and file CCSIDs that are being used. On input, the subcommand is QueryCvt. On output, the subcommand is reset to SetCvtOn, SetCvtAll, or SetCvtOff, indicating that conversion is ON or ALL or OFF, respectively. The current CCSIDs are returned in their respective positions in the F_CVT structure.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fcntl service returns 0 or greater, if the request is successful; or -1, if it is not successful. The following table lists the possible values of Return_value for each action specified:

Action	Argument	Return_value
F_CLOSFD	File_descriptor_2	0
F_CONTROL_CVT	F_CVT	0
F_DUPFD	File_descriptor_2	File_descriptor
F_DUPFD2	File_descriptor_2	File_descriptor
F_GETFD	0	File_descriptor_flags
F_GETFL	0	File_status_flags
F_GETLK	Lock_information	Lock_information
F_GETOWN	0	Pid
F_SETFD	File_descriptor_flags	0
F_SETFL	File_status_flags	0
F_SETLK	Lock_information	0

Action	Argument	Return_value
F_SETLKW	Lock_information	0
F_SETOWN	Pid	0
F_SETTAG	File_Tag	0

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fcntl service stores the return code. The fcntl service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fcntl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The calling process asked to set a lock, but the lock conflicts with a lock on an overlapping part of the file that is already set by another process.
EBADF	The request was not accepted, for one of these reasons: • The File_descriptor parameter does not specify a valid, open file descriptor. • The request was to set a read lock, but the file is open for writing only. • The request was to set a write lock, but the file is open for reading only. • File_descriptor was opened with an opendir request. Many of the other requests are rejected for an opendir filedes. • If the action requested was F_DUPFD2, this error indicates that File_descriptor_2 was negative, or was equal to or greater than the highest file descriptor value allowed for the process. The MAXFILEPROC parmlib option is used to specify the largest file descriptor value for the system. The following reason code can accompany the return code: JRFdTooBig.
EDEADLK	The action requested was F_SETLKW; the potential for deadlock was detected.
EINTR	While processing a F_SETLKW request, fcntl was interrupted by a signal.

Return_code	Explanation
EINVAL	The request was not accepted, for one of these reasons: • If the action requested was F_DUPFD, File_descriptor_2 was negative, or it was equal to or greater than the highest file descriptor value that is allowed for the process. The MAXFILEPROC parmlib option is used to specify the largest file descriptor value for the system. • If the action requested was F_SETLK or F_SETLKW, the file that is specified by File_descriptor does not support locking, or the Lock_information parameter contains incorrect values. • The action requested was F_CLOSFD and the file descriptor that is specified by File_descriptor_2 was less than File_descriptor, but not equal to -1. • The action requested was F_SETTAG or F_CONTROL_CVT, and either incorrect input data was supplied, or the file was inappropriate for this use. • An incorrect action was requested. The following reason codes can accompany the return code: JRFdTooBig, JRFd2TooSmall, JrBrlmBadFileType, JrBrlmBadL_Type, JrBrlmInvalidRange, JrBrlmBadL_Whence, JrNotsupportedForFileType, JrBadInputBufAddr, JrFileNotEmpty, JrWFildeRdOnly, JrInvalidFileTag, JrInvalidCcsid, JrBadOptCode.
EMFILE	The action requested was F_DUPFD. The process has already reached its maximum number of file descriptors, or there is no file descriptor available greater than File_descriptor_2.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPERM	The action requested was F_CLOSFD, and at least one of the file descriptors in the specified range remains open. For a description of the file descriptors that cannot be closed with F_CLOSFD, see the “Usage notes.”

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fcntl service stores the reason code. The fcntl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

When closing files:

- A process can use the fcntl service to close a range of file descriptors. File_descriptor_2 must be greater than or equal to File_descriptor, or it can be -1, which indicates that all file descriptors greater than or equal to File_descriptor are to be closed.
- Use of F_CLOSFD is meant to be consistent with use of the close service (BPX1CLO, BPX4CLO). You cannot close file descriptors that could not also be closed using the close service.

- If a file descriptor cannot be closed, it is considered an error, but the request continues with the next file descriptor in the range. File descriptors that are not in use are ignored.

When locking files:

- A process can use the fcntl service to lock out other cooperating processes from part of a file, so that the process can read or write to that part of the file without interference from others. This ensures data integrity when several processes are accessing a file concurrently.
- File locking can only be performed on file descriptors that refer to regular files. Locking is not permitted on file descriptors that refer to directories, FIFO files, pipes, character special files, or any other type of file.
- Locking operations are controlled with a structure mapped by BPXYBRLK, whose format is described in "Parameters" on page 175. This structure is needed whether the request is for setting a lock, releasing a lock, or querying a particular byte range for a lock. The following is a more detailed description of the BPXYBRLK structure.
- The l_type field is used to specify the type of lock that is to be set, cleared, or queried. Valid values for l_type are as follows:

  Type Description

  F_RDLCK
  A *read lock*. Specified as a halfword integer value of 1, this is also known as a *shared lock*. This type of lock specifies that the process can read the locked part of the file, and other processes cannot write on that part of the file while it is doing so. A process can change a held write lock, or any part of it, to a read lock, thereby making it available for other processes to read. Multiple processes can have read locks on the same part of a file simultaneously. To establish a read lock, a process must have the file that is accessed for reading.

  F_WRLCK
  A *write lock*. Specified as a halfword integer value of 2, this is also know as an *exclusive lock*. This type of lock indicates that the process can write on the locked part of the file, without interference from other processes. If one process puts a write lock on part of a file, no other process can establish a read lock or write lock on that same part of the file. A process cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that process is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock that is being requested. To establish a write lock, a process must have the file that is accessed for writing.

  F_UNLCK
  Indicates unlock. Specified as a halfword integer value of 3, this is used to unlock all locks held on the given range by the requesting process.
- The use of the l_whence and l_start fields for the fcntl service parallels their processing for the lseek service (BPX1LSK, BPX4LSK). See "lseek (BPX1LSK, BPX4LSK) — Change a file's offset" on page 345 for more information.
- The l_whence field is used to specify how the byte range offset is to be found within the file. Valid values for l_whence are as follows:

  Value Description

SEEK_SET
Stands for the start of the file, and is specified as a halfword integer value of 0.

SEEK_CUR
Stands for the current file offset in the file, and is specified as a halfword integer value of 1.

SEEK_END
Stands for the end of the file, and is specified as a halfword integer value of 2.

- The l_start field is used to identify the part of the file that is to be locked, unlocked, or queried. The part of the file that is affected by the lock begins at this offset from the location specified by the l_whence field. For example, if l_whence is SEEK_CUR and l_start is the value 10, a F_SETLK request attempts to set a lock beginning 10 bytes past the current cursor position. The l_start value may be negative, provided that when it is added to the offset indicated by the l_whence position, the resulting offset does not extend beyond the beginning of the file.

  Note: Although you cannot request a byte range that begins or extends beyond the beginning of the file, you can request a byte range that starts or extends beyond the end of the file.
- The l_len field is used to give the size of the locked part of the file, in bytes. The value specified for l_len may be negative. If l_len is positive, the area affected begins at l_start and ends at l_start+l_len-1. If l_len is negative, the area affected begins at l_start+l_len and ends at l_start-1. If l_len is zero, the locked part of the file begins at the position specified by l_whence and l_start, and extends to the end of the file.
- The l_pid field identifies the process ID of the process that holds the lock found on an F_GETLK request, if one was found.

When obtaining locks:

- You can set locks by specifying F_SETLK as the Action parameter for the BPX1FCT (BPX4FCT) service. If the lock cannot be obtained, a Return_value of -1 is returned along with an appropriate Return_code and Reason_code. You can also use F_SETLK to release locks that are already held, by setting l_type to F_UNLCK.
- You can also set locks by specifying F_SETLKW as the Action parameter for the BPX1FCT (BPX4FCT) service. If the lock cannot be obtained because another process has a lock on all or part of the requested range, the F_SETLKW request waits until the specified range becomes free and the request can be completed. You can also use F_SETLKW to release locks that are already held, by setting l_type to F_UNLCK.
- If a signal interrupts a call to the fcntl service while it is waiting in an F_SETLKW operation, the function returns with a Return_value of -1 and a Return_code of EINTR.
- F_SETLKW operations can encounter deadlocks. This happens when process A is waiting for process B to unlock a region, and process B is waiting for process A to unlock a different region. If the system detects that an F_SETLKW might cause a deadlock, the fcntl service returns with a Return_value of -1 and a Return_code of EDEADLK.

When determining lock status:

- A process can determine locking information about a file by using F_GETLK as the Action parameter for the fcntl service. In this case, Argument should specify a pointer to a structure that is mapped by the BPXYBRLK macro. This structure should describe a lock operation that the caller would like to perform. When the fcntl service returns, the structure is modified to describe the first lock found that would prevent the proposed lock operation from completing successfully.
- If a lock is found that would prevent the proposed lock from being set, the F_GETLK request returns a modified structure whose:
  - l_whence value is always SEEK_SET
  - l_start value gives the offset of the locked portion from the beginning of the file
  - l_len value is set to the length of the locked portion of the file
  - l_pid value is set to the process ID of the process that is holding the lock

  If there are no locks that would prevent the proposed lock operation from completing successfully, the returned structure is modified to have an l_type of F_UNLCK, but otherwise remains unchanged.

When there are multiple lock requests:

- A process can have several locks on a file simultaneously, but it can have only one type of lock set on any given byte. If a process puts a new lock on part of a file that it has previously locked, the process has only one lock on that part of the file and the lock type is the one given by the most recent locking operation.

When releasing locks:

- If an F_SETLK or F_SETLKW request is made to unlock a byte region of a file, all locks that are held by that process within the specified region are released. In other words, each byte specified on an unlock request is freed from any lock that is held against it by the requesting process.
- All of a process's locks on a file are removed when the process closes a file descriptor for that file. Locks are not inherited by a child process created with the fork service. See “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185 for more information about the fork service.

Note: All locks are advisory only. Processes can use locks to inform each other that they want to protect parts of a file, but locks do not prevent I/O on the locked parts. A process that has appropriate permissions on a file can perform any I/O it chooses, regardless of which locks are set. Therefore, file locking is only a convention, and it works only when all processes respect the convention.

Related services

- “close (BPX1CLO, BPX4CLO) — Close a file” on page 103
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185
- “lseek (BPX1LSK, BPX4LSK) — Change a file's offset” on page 345
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 447

Characteristics and restrictions

There are no restrictions on the use of the fcntl service.

Examples

For an example that uses this callable service, see "BPX1FCT (fcntl) example" on page 1139.

fork (BPX1FRK, BPX4FRK) — Create a new process

Function

The fork callable service creates a new process, called a *child process*.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW key 8, TCB key 8.
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FRK):	31-bit
AMODE (BPX4FRK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FRK,(Process_ID,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FRK with the same parameters.

Parameters

Process_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fork service places the process ID of the newly created child process, 0, or -1.

Upon successful completion, fork returns the process ID of the newly created child to the calling (parent) process.

Because the child is a duplicate, it contains the same service request to the fork service as the parent. Execution of the child begins with this fork service returning a process ID value of zero; the child then proceeds with normal execution.

If Process_ID is returned as -1, no child process was created, for the reason shown by Return_code.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fork service stores the return code. The fork service returns Return_code only if Process_ID is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fork service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The resources required to let another process be created are not available now; or you have already reached the maximum number of processes you are allowed to run. The following reason codes can accompany the return code: JRForkExitRcChildNoStorage, JRForkExitRcParentBadEnv, JRForkExitRcParentNoRoom, JRForkNoAccess, JRForkNoResource, JRForkVsmListTooLarge, JRKernelReady, JRMaxChild, JRMaxProc, JRMaxUIDs, JRNoSecurityProduct, JRNotKey8, and JRWlmWonErr.
EINVAL	The following reason code can accompany the return code: JRJsrRacXtr.
ENOMEM	The process requires more space than is available.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fork service stores the reason code. The fork service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The new process (called the *child process*) is a duplicate of the process that calls the fork service (called the *parent process*), except for the following:
   - The child process has a unique process ID (PID) that does not match any active process group ID.
   - The child has a different parent process ID (namely, the process ID of the process that called the fork service).
   - The child has its own copy of the parent's file descriptors. Each file descriptor in the child refers to the same open file as the corresponding file descriptor in the parent.
   - If an HFS file has its FCTLCLOFORK flag set on, it is not inherited by the child process. This flag is set with the fcntl service. For more information, see "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174.
   - The child has its own copy of the parent's open directory streams. Each open directory stream in the child can share directory stream positioning with the corresponding directory stream of the parent.
   - The process and system utilization times for the child are set to zero.
   - Any file locks previously set by the parent are not inherited by the child.

- The child process has no interval timers set (similar to the results of a call to the alarm service with Wait_time specified as zero).
- The child has no pending signals.

In other respects, for z/OS UNIX the child is identical to the parent.

2. The child process inherits all key 8 shared memory segments attached to the calling process. The internal values of the number of processes attached to each shared memory segment (shm_nattch) are incremented.

   BPX1FRK only supports the propagation of key 8 storage; therefore, the fork service does not propagate to the child any shared memory segments that reside in a storage key other than key 8.
3. If the calling address space uses the macro IARVSERV to capture storage, these pages are not copied to the child address space.
4. The semaphore adjustment values (semadj) are cleared in the child process.
5. PSW Key 2 mmap storage areas are not propagated to the child.
6. For AMODE 64 callers, high-memory storage is copied to the child process in the following cases:
   - All storage that is obtained by an IARV64 request made by the forking thread is copied to the child process.
   - All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and the parent process's PID in bits 32-64 is copied to the child process. In the child process, the user token is changed to the value of the child process's PID in bits 32-64.
   - All storage that is obtained by an IARV64 request with a user token that contains zeros in bits 0-31 and a nonzero value that matches ThliParentTkn in bits 32-64 (when ThliChildTkn is nonzero) is copied to the child process. In the child process, the user token is changed to the value of ThliChildTkn (from the parent process). This value is also used to initialize ThliParentTkn on the child process.
   - All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the parent process's PID in bits 0-31 is copied to the child process. In the child process, the user token is changed to the value of the child process's PID in bits 0-31.
   - All authorized storage that is obtained by an IARV64 request with a user token that contains zeros in bits 32-64 and the value of PSALAA in bits 0-31 is copied to the child process. In the child process, the user token is changed to the value of the child process's LAA in bits 0-31.

   The child process inherits the MEMLIMIT of the parent.

The child address space inherits the following address space attributes of the parent address space:

1. Region size
2. Time limit

Related services

- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 174
- “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304
- “setrlimit (BPX1SRL, BPX4SRL) — Set resource limits” on page 698

- "times (BPX1TIM, BPX4TIM) — Get process and child process times" on page 856
- "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882

Characteristics and restrictions

Following is a list of characteristics or restrictions for the fork call:

- The fork service can be requested from either an MVS or kernel address space.
- The fork service is supported from programs running in PSW key 8 only. An additional requirement is that the storage protection key value in the TCBPKF field of the task control block (TCB) must be 8. The fork service from authorized or problem-state programs with a PSW key other than 8 or a TCBPKF value other than 8 is rejected with an error code.
- Only the following storage subpools are copied by fork: 0–127, 129–132, and 251–252.
- With the exception of subpool 252, which is all key-0 storage, only the caller's key-8 storage is copied to the child. For subpools that support multiple keys—that is, subpool 129 to subpool 132—only storage obtained with a key of 8 is copied.
- When the fork service is called from a single-process address space, all storage obtained by all the tasks in the calling jobstep in the given subpools are copied to the child address space.

  When the fork service is called from a multiple-process address space, only storage obtained by the tasks in the calling process in the subpools identified previously are copied to the child address space.
- The child process always runs in problem program state key of 8, even when it is forked by an APF-authorized MVS process.
- One task (thread) and one request block (RB) are present in the child address space after the fork service request.

  If the parent was single-task with multiple RBs, only a single RB is created in the child address space after the fork service request. If multiple tasks exist in the parent process, only the task issuing the fork service request is replicated. There is no serialization among the different tasks.
- The TCB address and the addresses of other MVS control blocks are likely to be different in the child.
- The fork service does not copy any system subpools or MVS control blocks from the parent to the child, except as noted.

  For example, the task I/O table (TIOT) is not copied. This means that MVS data sets that were allocated in the parent are not allocated to the child, with the exception of the propagated TASKLIB, STEPLIB, or JOBLIB DD data sets. Because user data in user subpools are copied, it is possible that some of those control blocks can point to system control blocks that are no longer present in the child.

  As another example, a user's data control block (DCB) that has been opened in the parent still appears as an opened DCB in the child, but the corresponding system control blocks pointed to by the DCB are not present in the child.

  Only services that are specifically documented as supported can be used across the fork service. For further details, see "MVS-related information" in this topic.
- There is a limit on the total number of "living" or "zombied" children the parent can have at a time. This limit is set with the MAXPROCUSER parameter in a BPXPRMxx parmlib member. You can retrieve this count with the sysconf service (BPX1SYC, BPX4SYNC).

Although the child process resembles the parent process in many ways, it has specific differences from the parent process. Besides the differences described in POSIX.1 (under fork), the following are some examples of elements in the parent process that are not propagated to the child process:

- **Linkage stack.** The caller can have a linkage stack, but the child does not inherit it. If the caller intends to do an exec service request in the child, the loss of the linkage stack is not a problem. It is a problem only if the child process executes a PR (Program Return) instruction that requires the linkage stack.
- **Access list (that is, PASN-AL, DU-AL**). The parent's access lists are not propagated to the child.
- **Access registers.** Access registers are not propagated to the child, because the child process does not inherit the parent's access list, which would be needed to use the access registers.
- **Virtual pages.** Virtual pages that were page-fixed in the parent are not page-fixed in the child.
- **Dynamic resource managers (RESMGRs).** Dynamic resource managers that were established for the parent are not propagated to the child.
- **MVS files.** Any MVS files that were opened for the parent are not opened for the child process, with the exception of the TASKLIB, STEPLIB or JOBLIB DD data sets that were propagated from the parent process. Only z/OS UNIX files are opened in the child process.
- **Mutexes and condition variables.** Ownership of mutexes and condition variables is on a single-thread basis; therefore, these attributes cannot be propagated on fork. Where a mutex or condition variables exists, the thread that is created in the child has access to the shared memory and can use the mutex or condition variable. When it begins running, however, it will not own any mutexes or consume any condition variables.

Examples

For an example using this callable service, see “BPX1FRK (fork) example” on page 1140.

MVS-related information

1. Following is a list of services in the child that relate to the services done in the parent:
   - GETMAIN or FREEMAIN, or STORAGE. If the parent process has issued a GETMAIN macro for a storage block, the child process can issue a FREEMAIN macro for the same storage block.
   - LOAD or DELETE. If a problem state parent process issues a LOAD macro for a module, the child process can issue a DELETE macro to remove the module from storage. If the child process issues a LOAD macro for the same module that was loaded in the parent, the copied version of the module is used and the use count is incremented.

     If a supervisor state parent process issues a LOAD macro for a module, the child process cannot issue a DELETE macro for the module, and it cannot use a LOAD macro to load a new copy of the module.

     A LOAD macro for global storage, however, is not reflected in the child; the child cannot issue a DELETE macro to remove a module that was loaded to a common storage by the parent.
   - CSVQUERY. The EPTOKEN (entry point token) returned as OUTEPTKN on a CSVQUERY macro in the parent can be used by the child as the INEPTKN parameter on a CSVQUERY macro to refer to the same module.

- ESTAE. The child process can issue an ESTAE macro with a `0` parameter to delete an ESTAE routine established by the parent process.
- ESPIE. The child process can delete an ESPIE routine established by the parent process.

  Note: No other MVS services are carried across fork. They can be freely used in either the parent process or the child process, as long as it is understood that the result of these services (if performed in the parent process) cannot be available to the child process.

2. The system propagates the contents directory related information (including extent lists) for the job pack queue for the job step task related to the task issuing the fork call. It also propagates the information on all modules (whether private or in the LPA) that have been loaded by the task issuing the fork call.
3. The system propagates the current task's SPIE or ESPIE and STAE or ESTAE status to the child process.
   - STAE or ESTAE control blocks representing the current RB are propagated to the child process. Control blocks associated with older RBs are not propagated, nor are STAI or ESTAI control blocks.
   - SPIE or ESPIE control blocks representing the current RB are propagated to the child process. SPIE or ESPIE control blocks associated with older RBs are not propagated.
4. Security information from the parent's address space is propagated to the child's address space. As a result, the child has a security environment equivalent to that of the parent.
5. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the current task are propagated to the child's address space. This causes the child address space to have the same MVS program search order as the calling parent task.
6. The accounting information of the parent's address space is propagated to the child's address space. (See Managing accounting work in *z/OS UNIX System Services Planning*.)

   If the ThliForkAcctg bit is set on in “BPXYTHLI — Thread-level information” on page 1060, the fork service creates the child with the accounting data from the RACF WORKATTR of the user ID that is associated with the last setuid call. If no setuid call has been performed, the accounting information from the parent is used. No error is returned to the caller.
7. The job name of the parent is propagated to the child and appended with a numeric value in the range of 1–9 if the job name is 7 characters or fewer. If the job name is 8 characters, the job name is propagated as is. When a job name is appended with a numeric value, the count wraps back to 1 when it exceeds 9.
8. If the calling parent task is in a Work Load Manager (WLM) enclave, the child is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.
9. z/OS UNIX sets a default message class of “A” for all forked or spawned processes. Unlike JES, z/OS UNIX does not have a method for accepting a user-supplied default message class, and a default had to be supplied to the converter interpreter. Message class A was chosen as the default for BPXAS initiators. There is currently no way to dynamically change this default value.

The MSGCLASS for the joblog (JESMSGLG, JESJCL, JESYSMSG) is set to class A before the fork or spawn that associates the process with the BPXAS initiator is begun.

10. The user syscall trace setting is propagated to the child process.

fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor

Function

The fpathconf callable service determines the current values of a configurable limit or option (variable) that is associated with a file or directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FPC):	31-bit
AMODE (BPX4FPC):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FPC,(File_descriptor,
              Name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FPC with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the file.

Name
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates which configurable limit or option (variable) is to be returned in the Return_value. Use the

BPXYPFC macro (see "BPXYPCF — Command values for pathconf and pathconf" on page 1005) to specify the path name variable you want returned. The following table shows the variables that can be returned:

Variable returned	Description
PC_CHOWN_RESTRICTED	The change ownership ("chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93) function is restricted to processes with appropriate privileges (see "Authorization" on page 8). The group ID (GID) of a file can be changed only to the effective group ID of the process, or to one of its supplementary group IDs.
PC_LINK_MAX	The maximum value of a file's link count.
PC_MAX_CANON	The maximum number of bytes in a terminal canonical input line.
PC_MAX_INPUT	The minimum number of bytes for which space will be available in a terminal input queue. This is the maximum number of bytes a portable application may require to be typed as input before it reads them.
PC_NAME_MAX	The maximum number of bytes in a file name (not a string length; the count excludes a terminating null).
PC_NO_TRUNC	Path name components longer than 255 bytes generate an error.
PATH_MAX	The maximum number of bytes in a path name (not a string length; the count excludes a terminating null).
PIPE_BUF	The maximum number of bytes that can be written atomically when writing to a pipe.
_POSIX_VDISABLE	Terminal special characters maintained by the system can be disabled using this character value. For information on querying and setting these special characters, see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831 or "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842.
PC_ACL	The security product supports access control lists.
PC_ACL_ENTRIES_MAX	The maximum number of entries that can be placed in an access control list for the specified file.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fpathconf service returns the current value of the Pathname variable that corresponds to the Name specified, or -1 if the request is not successful.

If the named Pathname variable does not have a limit for the specified file, Return_value is set to -1 and the Return_code and Reason_code remain unchanged.

If PC_CHOWN_RESTRICTED is specified for Name, and PC_CHOWN_RESTRICTED is active, Return_value is set to 1.

If PC_CHOWN_RESTRICTED is specified for Name, and PC_CHOWN_RESTRICTED is not active, Return_value is set to 0.

If PC_NO_TRUNC is specified for Name, and PC_NO_TRUNC is active, Return_value is set to 1.

If PC_NO_TRUNC is specified for Name, and PC_NO_TRUNC is not active, Return_value is set to 0.

If PC_ACL is specified for Name, and PC_ACL is supported, Return_value is set to 1.

If PC_ACL is specified for Name, and PC_ACL is not supported, Return_value is set to 0.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fpathconf service stores the return code. The fpathconf service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*.

If the named Pathname variable does not have a limit for the specified file, Return_value is -1 and Return_code is unchanged. Otherwise, the fpathconf service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor argument is not a valid file descriptor.
EINVAL	Refer to the usage notes for situations in which this is returned.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fpathconf service stores the reason code. The fpathconf service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If Name refers to MAX_CANON, MAX_INPUT, or _POSIX_VDISABLE, the following applies:
   - If File_descriptor does not refer to a terminal file, the function returns -1 in Return_value and sets the Return_code to EINVAL.
2. If Name refers to NAME_MAX, PATH_MAX, or _POSIX_NO_TRUNC, the following applies:
   - If File_descriptor does not refer to a directory, the function still returns the requested information using the parent directory of the specified file.
3. If Name refers to PC_PIPE_BUF, the following applies:

- If File_descriptor refers to a pipe or a FIFO, the value returned applies to the referred-to object itself. If File_descriptor refers to a directory, the value returned applies to any FIFOs that exist or that can be created within the directory. If File_descriptor refers to any other type of file, the function returns -1 in Return_value and sets the Return_code to EINVAL.

4. If Name refers to PC_LINK_MAX, the following applies:
   - If File_descriptor refers to a directory, the value returned applies to the directory.

Related services

- "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464

Characteristics and restrictions

There are no restrictions on the use of the fpathconf service.

Examples

For an example using this callable service, see "BPX1FPC (fpathconf) example" on page 1140.

freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures

Function

The freeaddrinfo callable service frees the Addr_Info structure(s) that are obtained by the getaddrinfo callable service ("getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location" on page 205).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FAI):	31-bit
AMODE (BPX4FAI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FAI,(Addr_Info_Ptr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FAI with the same parameters.

Parameters

Addr_Info_Ptr
Supplied parameter

Type: Pointer

Length:
Fullword

The name of a fullword field that contains a pointer to an Addr_Info structure or a linked list of Addr_Info structures returned by the getaddrinfo callable service. See Addr_Info – AddrInfo Data Structure in the EZBREHST assembler macro for more information about the format of this structure. The EZBREHST macro is shipped in the installation's MACLIB SMP/E DDEF location.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the freeaddrinfo service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the freeaddrinfo service stores the return code. The freeaddrinfo service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS Communications Server: IP and SNA Codes*. The freeaddrinfo service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAI_AGAIN	The resolver address space has not been started. Try the request later.
EAI_FAIL	An unrecoverable error occurred.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the freeaddrinfo service stores the reason code. The freeaddrinfo service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS Communications Server: IP and SNA Codes*.

Usage notes

1. The freeadrinfo service supports a thread-safe environment.

2. The pointer that is returned in the Results_Ptr parameter of the getaddrinfo callable service can be specified with the Addr_Info_Ptr parameter on the freeaddrinfo callable service.
3. When the Addr_Info_Ptr parameter points to a linked list of Addr_Info structures, the linked list of Addr_Info structures is freed with one invocation of the freeaddrinfo callable service.

Related services

- "getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location" on page 205
- "getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address" on page 246

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1FAI (freeaddrinfo) example" on page 1137.

fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor

Function

The fstat callable service obtains status information about a file. You identify the file by its file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FST):	31-bit
AMODE (BPX4FST):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FST,(File_descriptor,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FST with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the file.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area to which the fstat call returns Status_area. To determine the value of Status_area_length, use the BPXYSTAT macro (see "BPXYSTAT — Map the response structure for stat" on page 1057).

Status_area
Parameter supplied and returned

Type: Structure

Length:
The length of BPXYSTAT or Status_area_length, whichever is less.

The name of an area to which the fstat call returns the status information for the file. Status_area is mapped by the BPXYSTAT macro. For information on the contents of this macro, see "BPXYSTAT — Map the response structure for stat" on page 1057.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fstat service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fstat service stores the return code. The fstat service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fstat service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not identify a known file. One possible reason for this is that the file descriptor specified is from an opendir instead of an open, in which case JrNotForDir is returned as the reason code.
EINVAL	Parameter error; for example, a zero-length buffer was passed. The following reason code can accompany the return code: JRBuffTooSmall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fstat service stores the reason code. The fstat service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. All time fields in the Status_area are in POSIX format.
2. The file mode field in the Status_area is mapped by BPXYMODE, and the file type field within the mode area is mapped by BPXYFTYP. For information about these fields, see "BPXYMODE — Map the mode constants of the file services" on page 996 and "BPXYFTYP — File type definitions" on page 967.
3. When the mode of an open file is changed using a service such as chmod(), an fstat() reflects the change in mode. However, no change in access authorization is apparent when the file is accessed through a previously opened file descriptor.
4. If no security label (SECLABEL) exists for the file, the security label field in the Status_area contains binary zeros.

Related services

- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the fstat service.

Examples

For an example using this callable service, see "BPX1FST (fstat) example" on page 1140.

fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status

Function

The fstatvfs callable service obtains status information about a file system. The file system is specified by a file descriptor that refers to a file from the desired file system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FTV):	31-bit
AMODE (BPX4FTV):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FTV,(File_descriptor,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FTV with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the file.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area to which the service returns status information.

Status_area
Parameter supplied and returned

Type: Structure

Length:
Specified by the Status_area_length parameter

The name of an area of length Status_area_length to which the service returns the status information for the file system. The BPXYSSTF macro maps this area. For information on this macro, see "BPXYSSTF — Map response structure for file system status" on page 1055.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fstatvfs service returns the length of the status written to the Status_area if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fstatvfs service stores the return code. The fstatvfs service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fstatvfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	Information is temporarily unavailable. This can occur because the mount process for the file system is incomplete.
EBADF	The File_descriptor parameter does not specify a valid, open file descriptor.
EINVAL	Parameter error; for example, Status_area_length is too small. The following reason code can accompany the return code: JRBuffTooSmall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the fstatvfs service stores the reason code. The fstatvfs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the passed Status_area_length is not less than or equal to zero, it is not considered an error for the Status_area_length to be insufficient to hold the requested information. (In other words, future expansion is allowed for.) As much information as can fit is written to Status_area, and this amount is returned.

2. The amount of valid data returned in the Status_area is indicated by the Return_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

Related services

- "statvfs (BPX1STV, BPX4STV) — Get the file system status" on page 809
- "w_statvfs (BPX1STF, BPX4STF) — Get the file system status" on page 926

Characteristics and restrictions

There are no restrictions on the use of the fstatvfs service.

Examples

For an example using this callable service, see "BPX1FTV (fstatvfs) example" on page 1141.

fsync (BPX1FSY, BPX4FSY) — Write changes to permanent storage

Function

The fsync callable service writes changes on the permanent storage device that holds the file.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FSY):	31-bit
AMODE (BPX4FSY):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FSY,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FSY with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the file for which changes are to be written to permanent storage.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fsync service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fsync service stores the return code. The fsync service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The fsync service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not specify a valid, open file.
EINVAL	The file is not a regular file.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the fsync service stores the reason code. The fsync service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The fsync service causes all modified data in the specified file to be written to the permanent storage device that holds the file. On return from a successful call, all updates have been saved on the permanent storage device that holds the file.
2. If the file represented by the file_descriptor was opened with synchronous updates specified, there is no need to use the fsync callable service, because each write causes all updates to be written to permanent storage.

Related services

- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

The file identified by File_descriptor must be open for writing when the fsync service is called.

When automatic conversion is enabled, a fsync operation cannot write to permanent storage an untranslated partial character that was cached by z/OS. Untranslated partial characters occur when translating multibyte characters sets and z/OS is given data that does not end on a character boundary. Untranslated partial characters are resolved during a subsequent write operation (BPX1WRT/BPX4WRT) when the remaining part of the character is supplied.

Examples

For an example using this callable service, see "BPX1FSY (fsync) example" on page 1140.

ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file

Function

The ftruncate service changes the size of a file. The file is identified by its file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1FTR):	31-bit
AMODE (BPX4FTR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1FTR,(File_descriptor,
              File_length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4FTR with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the file whose size is to be changed.

File_length
Supplied parameter

Type: Integer

Length:
Doubleword

The name of a doubleword that contains the number of bytes the file is to contain after its size has been changed.

This field is a doubleword to accommodate large files. For normal processing with a singleword value, propagate the sign bit through the second word, so that the final doubleword value has a valid sign. The ftruncate service accepts only positive values.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ftruncate service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ftruncate service stores the return code. The ftruncate service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The ftruncate service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not specify a valid, open file.
EINVAL	The file is not a regular file; it is opened Read Only; or the File_length specified is negative. The following reason codes can accompany the return code: JRTrNegOffset, JRTrNotRegFile, and JRTrOpenedRO.
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRTrMountedRO.
EFBIG	The File_length parameter is greater than the maximum file size limit for the process. The following reason code can accompany the return code: JRWriteBeyondLimit.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ftruncate service stores the reason code. The ftruncate service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The ftruncate service changes the file size to File_length bytes, beginning from the first byte of the file. If the file was originally larger than File_length bytes, the data from File_length to the original end of the file is removed. If the file was originally shorter than File_length, bytes between the old and new lengths are read as zeros.
2. If File_length is greater than the soft file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.
3. Full blocks are returned to the file system so that they can be used again.
4. The file offset is not affected by an ftruncate request.

Related services

- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "truncate (BPX1TRU, BPX4TRU) — Change the size of a file" on page 859

Characteristics and restrictions

The file specified must be a regular file, open for writing.

Examples

For an example using this callable service, see "BPX1FTR (ftruncate) example" on page 1141.

getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location

Function

The getaddrinfo callable service translates the name of a service location (for example, a host name) or a service name (for example, FTP) into a set of socket addresses and other associated information. This information can be used to open a socket and connect to, or to send a datagram to, the specified service. The TCP/IP Services resolver attempts to resolve the host name through a name server, if one is present, or through the local data sets.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GAI):	31-bit
AMODE (BPX4GAI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GAI,(Node_Name,
              Node_Name_Length,
              Service_Name,
              Service_Name_Length,
              Hints_Ptr,
              Results_Ptr,
              Canonical_Length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GAI with the same parameters. Hints_Ptr and Results_Ptr are doubleword pointer fields.

Parameters

Node_Name
Supplied parameter

Type: Character

Character set:
EBCDIC

Length:
Specified by Node_Name_Length

Node_Name can be specified as one of the following:

1. An EBCDIC character string, up to 255 characters long, set to the node name (host name) that is being queried.
2. An EBCDIC character string set to the IP address of the node (host) where the service resides.

Also, Node_Name can include scope information in the form *host name%scope information* or *IP address%scope information*. The scope information can be an interface name or an interface index, and must be specified as an EBCDIC character string. The combined character string of host name and scope information cannot exceed 255 characters in length.

You must specify Node_Name or Service_Name, or both.

Node_Name_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Node_Name parameter.

Service_Name
Supplied parameter

Type: Character

Character set:
EBCDIC

Length:
Specified by Service_Name_Length

Service_Name can be specified as one of the following:

1. An EBCDIC character string, up to 32 characters long, set to the service name that is being queried.
2. An EBCDIC character string set to the port number of the required service.

You must specify Node_Name or Service_Name, or both.

Service_Name_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Service_Name parameter.

Hints_Ptr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a field that contains a pointer to an input Addr_Info structure. The following information can be specified in the input Addr_Info:

- A set of flags (ai_flags) for interpreting the request. The flag settings are: AI_PASSIVE, AI_CANONNAMEOK, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.
- The address family (ai_family) that the caller expects to be returned by the resolver. Valid settings are AF_UNSPEC, AF_INET, and AF_INET6.
- The socket type (ai_socktype) that the caller can accept as a response.
- The protocol (ai_protocol) that the caller can accept as a response.

All other fields in the Addr_Info structure must be set to zero.

See Addr_Info – AddrInfo Data Structure in the EZBREHST assembler macro for more information about the format of this structure. The EZBREHST macro is shipped in the installation's MACLIB SMP/E DDEF location.

If the Hints_Ptr parameter is not specified (zero), the invocation is treated as if ai_family=AF_UNSPEC, ai_socktype=0, ai_protocol=0, and all the ai_flags are specified as off.

Results_Ptr
Returned parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a field that contains a pointer to an output Addr_Info structure. If more than one address is returned, this field contains a linked list of output Addr_Info structures. Each output Addr_Info structure contains the following information:

- A set of flags (ai_flags) for interpreting the address that is returned in this Addr_Info structure. For output Addr_Info structures, this value is unpredictable.
- The address family (ai_family) for the address returned in this Addr_Info structure.
- The socket type (ai_socktype) for the address returned in this Addr_Info structure.
- The protocol (ai_protocol) for the address returned in this Addr_Info structure.
- The length (ai_addrlen) of the sock_inet_sockaddr or sock_inet6_sockaddr structure returned in the ai_addr field.
- The canonical name (ai_canonname) associated with the input Node_Name, if this was requested using the input AI_CANONNAMEOK flag. If more than one Addr_Info structure is returned, the canonical name is supplied in only the first Addr_Info structure.

  The length of the canonical name is returned in the Canonical_Length parameter. If no canonical name exists, this field contains the input value that was passed in the Node_Name parameter. If AI_CANONNAMEOK in the input Addr_Info structure was zero, ai_canonname in the output Addr_Info structure is set to zero.
- The socket address (ai_addr) returned by the resolver in this Addr_Info structure, in the form of a sock_inet_sockaddr or sock_inet6_sockaddr address structure. The length of the address returned is supplied by ai_addrlen.
- The next Addr_Info structure (ai_next) returned by the resolver. If this is the last Addr_Info structure returned as part of the reply, this value is X'00000000'.

See Addr_Info – AddrInfo Data Structure in EZBREHST for more information about the format of this structure.

Canonical_Length
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getaddrinfo service returns the length of the canonical name that was returned in the first Addr_Info structure pointed to by the Results_Ptr parameter.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getaddrinfo service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getaddrinfo service stores the return code. The getaddrinfo service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS Communications Server: IP and SNA Codes*. The getaddrinfo service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAI_NONAME	One of the following occurred: 1. The name does not resolve for the specified parameters. 2. A Name or Service operand was not specified. At least one of the Name or Service operands must be specified.
EAI_AGAIN	The name specified by the Node_Name or Service_Name parameter could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.
EAI_FAIL	An unrecoverable error occurred.
EAI_SOCKTYPE	The intended socket type was not recognized.
EAI_SERVICE	The service that was passed was not recognized for the specified socket type.
EAI_BADFLAGS	The ai_flags parameter had an incorrect setting.
EAI_FAMILY	The ai_family parameter had an incorrect setting.
EAI_MEMORY	A memory allocation failure occurred during an attempt to acquire an Addr_Info structure.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getaddrinfo service stores the reason code. The getaddrinfo service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See*z/OS Communications Server: IP and SNA Codes* for the reason codes.

Usage notes

1. When you specify Node_Name as an EBCDIC IP address, you must use the conventional forms for expressing IPv4 and IPv6 addresses as text strings. For example, the IPv4 address 1.1.1.1 would be specified as `F14BF14BF14BF1`, and the IPv6 address 1:1:1:1:1:1:1:1 would be specified as `F17AF17AF17AF17AF17AF17AF17AF1`.
2. When you specify the AI_NUMERICHOST flag in the input Addr_Info structure pointed to by the Hints_Ptr parameter, Node_Name must be an IP address specified as an EBCDIC character string.
3. When you specify Node_Name as an IP address, the address returned in the different structures is in different formats:
   - The IP address returned in the ai_canonname field of the first Addr_Info structure pointed to by Results_Ptr is in its EBCDIC format.

- The IP address returned in the sock_inet_sockaddr or sock_inet6_sockaddr structure of each returned Addr_Info structure pointed to by Results_Ptr is in numeric form (hexadecimal).

4. When the AI_NUMERICSERV flag is specified in the input Addr_Info structure pointed to by the Hints_Ptr parameter, Service_Name must be a port number specified as an EBCDIC character string.
5. The gettaddrinfo service supports a fully thread-safe environment. The Addr_Info structure or structures are allocated by the resolver and returned to the invoking application. The storage is subsequently returned to the resolver task, to be freed by the freeaddrinfo service (“freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures” on page 194). The storage for the Addr_Info structures is allocated in the caller's TCB key, and can be accessed in any key. To free the Addr_Info structures using the freeaddrinfo service, or change the contents of the structures, the application must be in their TCB key.
6. To get the most useful set of IP addresses available for the requested host name, applications that are enabled for IPv6 processing should specify AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG in the ai_flags field; and AF_UNSPEC for the ai_family field in the input Addr_Info structure pointed to by the Hints_Ptr parameter. When the stack has IPv6 capability, requests that are coded with AF_UNSPEC are treated as if the request is for AF_INET6, and all addresses are returned using sock_inet6_sockaddr structures (with the IPv4 addresses mapped appropriately, based on the AI_V4MAPPED setting). If there is no IPv6 capability, IPv4 addresses are returned in sock_inet_sockaddr structures. This frees the application, to some extent, from having to decide what format works for the stack.
7. Applications are encouraged to attempt all returned addresses, in order, when using the getaddrinfo results to open a socket and connect, or to send a datagram.
8. Scope information specified as part of the Node_Name input parameter is resolved into a zone index value and is returned in all the sock_inet6_sockaddr structures that represent IPv6 link-local addresses. Scope information is ignored for resolved IPv4 addresses and for IPv6 addresses that are not link-local scoped addresses.
9. These are the flag descriptions specified with the Addr_Info_Structure parameter:

Flag	Description
AI_PASSIVE (X'00000001')	Specifies how to fill in the NAME pointed to in Results_Ptr. If this flag is specified, the returned address information will be suitable for use in binding a socket for accepting incoming connections for the specified service (such as the bind syscall). If the Node_Name parameter is not specified, the IP address portion of the socket address structure pointed to in Results_Ptr will be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY for an IPv6 address. If this flag is not specified, the returned address information will be suitable for the connect syscall (for a connection-mode protocol); or for a connect, sendto, or sendmsg syscall (for a connectionless protocol). In this case, if the Node_Name parameter is not specified, the IP address portion of the socket address structure pointed to by Results_Ptr will be set to the default loopback address for an IPv4 address (127.0.0.0) or the default loopback address for an IPv6 address (::1). This flag is ignored if the Node_Name parameter is specified.
AI_CANONNAMEOK (X'00000002')	If this flag is specified and the Node_Name parameter is also specified, the getaddrinfo call attempts to determine the canonical name that corresponds to the Node_Name.
AI_NUMERICHOST (X'00000004')	If this flag is specified, the Node_Name parameter must be an IP address specified in EBCDIC format, or an error (EAI_NONAME) is returned.
AI_NUMERICSERV (X'00000008')	If this flag is specified, the Service_Name parameter must be a port number specified in EBCDIC format, or an error (EAI_NONAME) is returned.
AI_V4MAPPED (X'00000010')	If this flag is specified when ai_family=AF_INET6 or AF_UNSPEC in the input Addr_Info structure pointed to by the Hints_Ptr parameter and when IPv6 is supported on the system, the caller will accept IPv4-mapped IPv6 addresses. When the AI_ALL flag is not also specified and no IPv6 addresses are found, a query is made for IPv4 addresses. If any IPv4 addresses are found, they are returned as IPv4-mapped IPv6 addresses. This flag is ignored if ai_family is not AF_INET6, or if ai_family is AF_UNSPEC but IPv6 is not supported on the system.

Flag	Description
AI_ALL (X'00000020')	When the ai_family field has a value of AF_INET6 and AI_ALL is set, the AI_V4MAPPED flag must also be set to indicate that the caller will accept all addresses (IPv6 and IPv4-mapped IPv6 addresses). When the ai_family field has a value of AF_UNSPEC when the system supports IPv6 and AI_ALL is set, the caller accepts IPv6 addresses and either IPv4 (if AI_V4MAPPED is not set) or IPv4-mapped IPv6 (if AI_V4MAPPED is set) addresses. A query is first made for IPv6 addresses and if it is successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any found are returned as IPv4 addresses (if AI_V4MAPPED was not set) or as IPv4-mapped IPv6 addresses (if AI_V4MAPPED was set). If the ai_family field does not have the value of AF_INET6, or the value of AF_UNSPEC when the system supports IPv6, the flag is ignored.
AI_ADDRCONFIG (X'00000040')	If this flag is specified, a query for IPv6 on the Node_Name will occur if the resolver determines whether either of the following is true: • If the system is IPv6 enabled and has at least one IPv6 interface, the resolver will make a query for IPv6 (AAAA or A6 DNS) records. • If the system is IPv4 enabled and has at least one IPv4 interface, the resolver will make a query for IPv4 (A DNS) records. The loopback address is not considered in this case as a valid interface.

Related services

- “freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures” on page 194
- “getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address” on page 246

Characteristics and restrictions

None.

Examples

For an example using this callable service, see “BPX1GAI (getaddrinfo) example” on page 1141.

getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier

Function

The getclientid callable service obtains the calling program's identifier.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GCL):	31-bit
AMODE (BPX4GCL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GCL,(FunctionCode,
              Domain,
              Clientid,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GCL with the same parameters.

Parameters

FunctionCode
Supplied parameter

Type: Integer

Length:
Fullword

Specify a 1 to have the caller's name and task identifiers returned in the Clientid parameter. Specify a 2 to have the caller's process id returned in the Clientid parameter.

Domain
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the communications domain in which the sockets are to be given and taken. See “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043 for more information about the values defined for this field.

Clientid
Returned parameter

Type: Structure

Length:
Length of BPXYCID

The name of a structure that is to be returned with information that identifies the calling program.

If the FunctionCode parameter is 1, the returned Clientid is filled in as follows:

CIdDomain
Input Domain

CIdName
Calling program's address space name, left-justified, and padded with blanks

CIdTask
Calling program's subtask identifier

CIdReserved
Binary zeros

If the FunctionCode parameter is not 1, the returned Clientid is filled in as follows:

CIdDomain
Input Domain

CIdName
A fullword of binary zeros followed by the calling program's process id

CIdTask
Blanks

CIdReserved
Binary zeros

See "BPXYCID — Map the returning structure for getclientid()" on page 951 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getclientid service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getclientid service stores the return code. The getclientid service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getclientid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	Using the Clientid parameter as specified would result in an attempt to access storage outside the caller's address space.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the getclientid service stores the reason code. The getclientid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The Clientid output of getclientid is intended to be used as the input Clientid of the givesocket and takesocket services.
2. The output Clientid that is returned with an input FunctionCode of 2 provides optimal performance and integrity when used as the input Clientid on the givesocket and takesocket services. The input FunctionCode of 1 is only provided for existing applications that may have been using the output of getclientid for purposes other than as input on the givesocket or takesocket services.

Related services

- "givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program" on page 285
- "takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program" on page 821

Characteristics and restrictions

There are no restrictions on the use of the getclientid service.

getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory

Function

The getcwd callable service gets the pathname of the working directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN

Operation	Environment
AMODE (BPX1GCW):	31-bit
AMODE (BPX4GCW):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GCW,(Buffer_length,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GCW with the same parameters.

Parameters

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the buffer to which the getcwd service returns the pathname of the directory. Buffer_length must be large enough to accommodate the actual length of the pathname plus one (for the terminating null).

Buffer
Parameter supplied and returned

Type: Character string

Character set:
No restrictions

Length:
Specified by the Buffer_length parameter

The name of the buffer that is to hold the pathname of the working directory.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getcwd service returns the length of the pathname that is in the buffer, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getcwd service stores the return code. The getcwd service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getcwd service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process did not have permission to read or search a component of the working directory's pathname.
EINVAL	Buffer_length was specified as zero. The following reason code can accompany the return code: JRBufLenInvalid.
EIO	An input/output error occurred.
ENOENT	A component of a pathname does not exist. This will be returned if a component of the working directory pathname was deleted.
ERANGE	The specified Buffer_length is less than the length of the pathname of the working directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getcwd service stores the reason code. The getcwd service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "chdir (BPX1CHD, BPX4CHD) — Change the working directory" on page 88

Characteristics and restrictions

There are no restrictions on the use of the getcwd service.

Examples

For an example using this callable service, see "BPX1GCW (getcwd) example" on page 1142.

getegid (BPX1GEG, BPX4GEG) — Get the effective group ID

Function

The getegid callable service gets the effective group ID (GID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1GEG):	31-bit
AMODE (BPX4GEG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GEG,(Effective_group_ID)
```

AMODE 64 callers use BPX4GEG with the same parameter.

Parameters

Effective_group_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the getegid service returns the effective group ID of the calling process.

Usage notes

If this service fails, the process ends abnormally.

Related services

- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "getgid (BPX1GID, BPX4GID) — Get the real group ID" on page 220
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "setegid (BPX1SEG, BPX4SEG) — Set the effective group ID" on page 670
- "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

There are no restrictions on the use of the getegid service.

Examples

For an example using this callable service, see "BPX1GEG (getegid) example" on page 1142.

geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID

Function

The geteuid callable service gets the effective user ID (UID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GEU):	31-bit
AMODE (BPX4GEU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GEU,(Effective_user_ID)
```

AMODE 64 callers use BPX4GEU with the same parameter.

Parameters

Effective_user_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the geteuid service places the effective user ID of the calling process.

Usage notes

If this service fails, the process ends abnormally.

Related services

- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

There are no restrictions on the use of the geteuid service.

Examples

For an example using this callable service, see "BPX1GEU (geteuid) example" on page 1144.

getgid (BPX1GID, BPX4GID) — Get the real group ID

Function

The getgid callable service gets the real group ID (GID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GID):	31-bit
AMODE (BPX4GID):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GID,(Real_group_ID)
```

AMODE 64 callers use BPX4GID with the same parameter.

Parameters

Real_group_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgid service returns the real group ID.

Usage notes

If this service fails, the process ends abnormally.

Related services

- "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217
- "setegid (BPX1SEG, BPX4SEG) — Set the effective group ID" on page 670
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674

Characteristics and restrictions

There are no restrictions on the use of the getgid service.

Examples

For an example using this callable service, see "BPX1GID (getgid) example" on page 1146.

getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database

Function

The getgrent callable service gets information about a group and its members. Each time you use the getgrent service, you get information about the next group entry in the group database.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GGE):	31-bit
AMODE (BPX4GGE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GGE,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GGE with the same parameters. The address returned is a fullword (below the bar).

Parameters

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the getgrent service returns an address, or 0. If no more group entries exist in the group database, or if an error is encountered, Return_value is set to 0.

If an entry is found, Return_value is set to the address of a data area mapped by the BPXYGIDS macro. The first area contains the fullword length of the group name, followed by the group name, padded with blanks. See "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. The address returned is 31 bits for both AMODE=31 and AMODE 64 callers.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgrent service stores the return code or 0. The getgrent service returns Return_code only if Return_value is 0. Return_code is 0 when no more group entries exist in the database. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getgrent service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSSAF2ERR	The system authorization facility (SAF) or RACF Get GMAP service had an error.
EMVSSAFEXTRERR	The SAF or RACF RACROUTE EXTRACT service had an error.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the getgrent service stores the reason code or 0. The getgrent service returns Reason_code only if Return_value is 0. Reason_code is 0 when no more group entries exist in the database. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery
8	20	The group is incompletely defined.

Usage notes

1. The getgrent service is intended to be used to search the group database sequentially. The first call to this service from a given task returns a pointer to the first group entry in the group database. Subsequent calls from the same task return a pointer to the next group entry found, until no more entries exist. At this point a null pointer is returned.
2. The setgrent service can be used to reset this sequential search. The next getgrent service used from the same task after a call to setgrent returns a pointer to the first group entry. The next getgrent service used after an end of file indication (a null pointer) has been returned also returns a pointer to the first group entry. The use of setgrent after end of file is therefore optional.
3. The return value points to data that may change or go away after the next getgrgid, getgrnam, or getgrent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
4. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.

Related services

- "getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID"
- "getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name" on page 226
- "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245
- "setgrent (BPX1SGE, BPX4SGE) — Reset the group database" on page 677

Characteristics and restrictions

There are no restrictions on the use of the getgrent service.

Examples

For an example using this callable service, see "BPX1GGE (getgrent) example" on page 1144.

getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID

Function

The getgrgid callable service gets information about a group and its members. You specify the group by the group ID (GID).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GGI):	31-bit
AMODE (BPX4GGI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GGI,(Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GGI with the same parameters. The address returned is a fullword (below the bar).

Parameters

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the ID of the group you want information about.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the getgrgid service returns an address, or 0. If no entry for the specified group ID is found, Return_value is set to 0. If an entry is found, Return_value is set to the address of the BPXYGIDS macro. The first area contains the fullword length of the group name, followed by the group name padded with blanks. See "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969.

If an entry is found and function code ThliEP_GidNameSet was set in ThliEP_FunctionCode prior to making this call, then Return_value is set to the address of the area in the BPXYTHLI where group name length and group name are set, as per the BPXYGIDS macro.

The address returned is 31 bits for both AMODE 31 and AMODE 64 callers.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgrgid service stores the return code. The getgrgid service returns Return_code only if Return_value is 0. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The getgrgid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSSAFEXTRERR	The system authorization facility (SAF) RACROUTE EXTRACT service had an error.
EMVSSAF2ERR	The (SAF) Get GMAP service had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgrgid service stores the reason code. The getgrgid service returns Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	If the search is by GID, the GID is not defined. If the search is by group name, the current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

Usage notes

1. The return value points to data that may change or go away after the next getgrgid, getgrnam, or getgrent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. Performance degradation can occur if you use this service when Virtual Lookaside Facility (VLF) is not active. For more information, see Tuning performance and the section on establishing UNIX security in *z/OS UNIX System Services Planning*.
4. You can request to have just the group name of the specified GID returned instead of the group name and all of the members of the group. This will improve performance because just one group name look-up is done. To request this function, set the ThliEP_FunctionCode field with function code ThliEP_GIDNameSet prior to issuing the syscall. On successful return, the following values are set:
   - ThliEP_GIDNameLen is set to the length of the returned group name; valid values are 1-8.
   - ThliEP_GIDName is set to the group name of the specified GID.
   - ThliEP_GIDLen is always set to 4, the length of a GID.
   - ThliEP_GID is the GID specified on input.
   - ThliEP_GroupCount is always set to 0.

   Return_value is set to the output area. This is mapped by both the ThliExtendedGIDName structure in “BPXYTHLI — Thread-level information” on page 1060, and by “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 969. If no entry for the specified group ID is found, Return_value is set to 0.

   The ThliEP_FunctionCode is cleared prior to returning to the caller; so, the invoker will need to set the ThliEp_FunctionCode before each invocation of the BPX1GGI/BPX4GGI service if the ThliEP_GIDNameSet function is to be used.

Related services

- “getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database” on page 221
- “getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name” on page 226
- “getlogin (BPX1GLG, BPX4GLG) — Get the user login name” on page 245

Characteristics and restrictions

There are no restrictions on the use of the getgrgid service.

Examples

For an example using this callable service, see "BPX1GGI (getgrgid) example" on page 1144.

getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name

Function

The getgrnam callable service gets information about a group and its members. You specify the group by name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GGN):	31-bit
AMODE (BPX4GGN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GGN,(Group_name_length,
              Group_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GGN with the same parameters. The address returned is a fullword (below the bar).

Parameters

Group_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Group_name.

Group_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Group_name_length parameter

The name of the field that contains the name of the group you want information about.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword where the getgrnam service returns an address, or 0. If no entry for the specified group name is found, or no GID is specified, Return_value is set to 0. If an entry is found, Return_value is set to the address of the BPXYGIDS macro structure. The first area contains the fullword length of the group name followed by the group name padded with blanks.

If an entry is found and function code ThliEP_GidNameSet was set in ThliEP_FunctionCode prior to making this call, then Return_value is set to the address of the area in the BPXYTHLI where the GID is set, as per the BPXYGIDS macro.

The address returned is 31 bits for both AMODE 31 and AMODE 64 callers. See "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgrnam service stores the return code. The getgrnam service returns Return_code only if Return_value is 0. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The getgrnam service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	Group name length is not valid.
EMVSSAFEXTRERR	The system authorization facility (SAF) RACROUTE EXTRACT service had an error.
EMVSSAF2ERR	The SAF Get GMAP service had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the getgrnam service stores the reason code. The getgrnam service returns Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. For the reason codes, see

z/OS UNIX System Services Messages and Codes. The reason code in the case of EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF Return Code	RACF Reason Code	Explanation
8	4	If the search is by GID: the GID is not defined. If the search is by group name: The current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

Usage notes

1. The return values point to data that can change or go away after the next getgrgid, getgrnam, or getgrent call from that task. Each task manages its own storage separately. Move data to your own dynamic storage if you need it for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. You can request to have just the GID of the specified group name returned without any information about the other group members. This will improve performance because just one group name look-up is done. To request this function, set the ThliEP_FunctionCode field with function code ThliEP_GIDNameSet prior to issuing the syscall. On successful return, the following values are set:
   - ThliEP_GIDNameLen is set to the length of the returned group name; valid values are 1-8.
   - ThliEP_GIDName is set to the specified group name.
   - ThliEP_GIDLen is always set to 4, the length of a GID.
   - ThliEP_GID is the GID associated with the input group name.
   - ThliEP_GroupCount is always set to 0.

   Return_value is set to the output area. This is mapped by both the ThliExtendedGIDName structure in "BPXYTHLI — Thread-level information" on page 1060, and by "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. If no entry for the specified group ID is found, Return_value is set to 0.

   The ThliEP_FunctionCode is cleared prior to returning to the caller; so, the invoker will need to set the ThliEp_FunctionCode before each invocation of the BPX1GGI/BPX4GGI service if the ThliEP_GIDNameSet function is to be used.

Related services

- "getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database" on page 221
- "getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID" on page 223
- "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245

Characteristics and restrictions

There are no restrictions on the use of the getgrnam service.

Examples

For an example using this callable service, see "BPX1GGN (getgrnam) example" on page 1145.

getgroups (BPX1GGR, BPX4GGR) — Get a list of supplementary group IDs

Function

The getgroups callable service gets the number of supplementary group IDs (GIDs) for the calling process. It optionally gets a list of those supplementary group IDs.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GGR):	31-bit
AMODE (BPX4GGR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GGR,(Group_ID_list_size,
              Group_ID_list_pointer_address,
              Number_of_group_IDs,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GGR with the same parameters. The Group_ID_list_pointer_address parameter is a doubleword.

Parameters

Group_ID_list_size
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies the number of fullword entries in the group ID list. This number must be at least as great as the total number of group IDs for the process, or must be 0.

If you specify 0, the program receives only a count of the actual number of group IDs for the calling process, and not a list of those IDs.

Group_ID_list_pointer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to a storage area in which the getgroups service is to place the list of supplementary group IDs. If Group_ID_list_size is specified as 0, Group_ID_list_pointer_address is ignored, and does not have to be set to a valid address. When the request is successful, the storage area is an array of fullwords, each containing a supplementary group ID for the calling process.

Number_of_group_IDs
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroups service returns a number that represents a count of supplementary group IDs. A -1 is returned if an error is detected.

- If Group_ID_list_size is specified as 0, the number is the total number of supplementary group IDs for the process.
- If Group_ID_list_size is specified as greater than 0 and the request was successful, the number is the actual number of group IDs that were put into the area specified by Group_ID_list_pointer_address.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroups service stores the return code. The getgroups service returns Return_code only if Number_of_group_IDs is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getgroups service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Group_ID_list_size parameter was greater than 0 but less than the number of supplementary group IDs.
EMVSSAF2ERR	System authorization facility (SAF) had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroups service stores the reason code. The getgroups service returns Reason_code only if Number_of_group_IDs is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

The reason code in the case of EMVSSAF2ERR contains the Resource Access Control Facility (RACF) return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF GETGRPS service return and reason code values, see the following table:

RACF Return Code	RACF Reason Code	Explanation
8	4	Group count is less than the number of supplemental groups
8	8	Invalid grouplist address
8	12	Internal error during RACF processing

Related services

- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674

Characteristics and restrictions

There are no restrictions on the use of the getgroups service.

Examples

For an example using this callable service, see "BPX1GGR (getgroups) example" on page 1145.

getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs by user name

Function

The getgroupsbyname service gets the number of supplementary group IDs (GIDs) and, optionally, gets a list of those supplementary group IDs for a specified user name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GUG):	31-bit
AMODE (BPX4GUG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GUG,(User_name_length,
              User_name,
              Group_ID_list_size,
              Group_ID_list_pointer_address,
              Number_of_group_IDs,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GUG with the same parameters. The Group_ID_list_pointer_address parameter is a doubleword.

Parameters

User_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of User_name.

User_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the User_name_length parameter

The name of a field of length User_name_length that contains the name of the user that you want information about. The name is specified in the Resource Access Control Facility (RACF) command that defined the user to the system.

Group_ID_list_size
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies the number of fullword entries in the group ID list. This number must be at least as great as the total number of group IDs for the process, or must be `0`.

If you specify `0`, the program receives only a count of the actual number of group IDs for the calling process, and not a list of those IDs.

Group_ID_list_pointer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to a storage area where the getgroupsbyname service is to place the list of supplementary group

IDs. If Group_ID_list_size is specified as 0, Group_ID_list_pointer_address is ignored, and does not have to be set to a valid address. When the request is successful, the storage is an array of fullwords, each containing a supplementary group ID for the calling process.

Number_of_group_IDs
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroupsbyname service returns the number of supplementary group IDs. A -1 is returned if an error is detected.

- If Group_ID_list_size is specified as 0, the number is the total number of supplementary group IDs for the process.
- If Group_ID_list_size is specified as greater than 0 and the request is successful, the number is the actual number of group IDs that are put into the area specified by Group_ID_list_pointer_address.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroupsbyname service stores the return code. The getgroupsbyname service returns Return_code only if Number_of_group_IDs is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getgroupsbyname service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Group_ID_list_size parameter was greater than 0 but less than the number of supplementary group IDs; or the User_name or User_name_length fields were incorrect.
EMVSSAF2ERR	A system authorization facility (SAF) service had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getgroupsbyname service stores the reason code. The getgroupsbyname service returns Reason_code only if Number_of group_IDs is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, the reason code contains the Resource Access Control Facility (RACF) return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF GETGNAME service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	Group count is less than the number of supplemental groups
8	8	Incorrect group list address
8	12	Internal error during RACF processing
8	16	Unable to establish recovery
8	20	Internal error verifying user ID. The user ID might have been revoked.
8	24	User ID is not defined to RACF

Related services

- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674

Characteristics and restrictions

There are no restrictions on the use of the getgroupsbyname service.

Examples

For an example using this callable service, see "BPX1GUG (getgroupsbyname) example" on page 1153.

gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address

Function

The gethostbyaddr callable service returns the alias names and the internet addresses of a host whose address is specified as input. The TCP/IP Services resolver tries to resolve the host address through a name server, if one is present. If a name server is not present, the resolver searches for the HOSTS.ADDRINFO data set (or **/etc** hosts data set) until a matching host address is found, or until an EOF marker is reached.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GHA):	31-bit
AMODE (BPX4GHA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GHA,(Address,
              Address_length,
              Hostent_ptr,
              Domain,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GHA with the same parameters. Hostent_ptr is a doubleword pointer field.

Parameters

Address
Supplied parameter

Type: Hexadecimal string

Length:
Length specified by Address_length

The name of a hexadecimal string that contains the IP address of the host being queried. This is a fullword field for IPv4 addresses. (IPv6 addresses are not supported.)

Address_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the address that is being passed in the Address parameter. This is 4 for IPv4 addresses. No other addresses are currently supported.

Hostent_ptr
Returned parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a field that contains a pointer to the Hostent structure. The Hostent structure contains the following fields:

h_name
The address of the host name returned by the service. The host name is a variable-length field that is ended by X'00'.

h_aliases
The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer X'00000000'. Each alias name is a variable-length field that is ended by X'00'.

h_addrtype
The value 2, which signifies AF_INET.

h_length
The length of the host internet addresses pointed to by h_addr_list.

h_addr_list
The address of a list of addresses that point to the host internet addresses returned by this service. This list is ended by the pointer X'00000000'.

Domain
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the numeric value of the domain for this query. Only the value of 2 (AF_INET) is currently supported.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyaddr service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyaddr service stores the return code. The gethostbyaddr service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The gethostbyaddr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
HOST_NOT_FOUND	The host name specified by the Address parameter was not found.
TRY_AGAIN	The host address specified by the Address parameter could not be resolved within the configured time interval. The request can be retried later.
NO_RECOVERY	An unrecoverable error occurred.
NO_DATA	The requested Address parameter is valid, but it does not have a record at the name server.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyaddr service stores the reason code. The gethostbyaddr service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value.

Reason codes lower than decimal 4096 are z/OS UNIX System Services return codes, and are documented in *z/OS UNIX System Services Messages and Codes*. Reason codes greater than decimal 4096 are returned by the resolver, and are described in *z/OS Communications Server: IP and SNA Codes*.

An assembler macro (EZBREHST) that contains the hostent structure, gethostbyxxxx return codes, and reason codes is shipped in the installation's MACLIB SMP/E DDEF location.

Related services

- "gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names"

Characteristics and restrictions

The gethostbyaddr service does not support a fully reentrant environment. The Hostent structure that is returned is allocated at a task level. This area will be reused on subsequent gethostbyaddr calls. Therefore, within a task only one call can be occurring at a time. For example, if the mainline task has issued a gethostbyaddr call that has not completed, a signal handler that interrupts that thread's processing should not invoke the gethostbyaddr service.

Examples

For an example using this callable service, see "BPX1GHA (gethostbyaddr) example" on page 1145.

gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names

Function

The gethostbyname callable service returns the alias names and the internet addresses of a host whose domain name is specified as input. The TCP/IP Services resolver tries to resolve the name through a name server, if one is present. If a name server is not present, the resolver searches for the HOSTS.SITEINFO data set (or **/etc** hosts data set) until a matching host name is found, or until an EOF marker is reached.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GHN):	31-bit
AMODE (BPX4GHN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GHN,(Name,
              Name_length,
              Hostent_ptr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GHN with the same parameters. Hostent_ptr is a doubleword pointer field.

Parameters

Name
Supplied parameter

Type: Character

Length:
Length specified by Name_length

A string, up to 255 characters long, set to the host name that is being queried.

Name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Name parameter.

Hostent_ptr
Returned parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a field that contains a pointer to the Hostent structure. The Hostent structure contains the following fields:

h_name
The address of the host name returned by the service. The host name is a variable-length field that is ended by X'00'.

h_aliases
The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer X'00000000'. Each alias name is a variable-length field that is ended by X'00'.

h_addrtype
The value 2, which signifies AF_INET.

h_length
The length of the host internet addresses pointed to by h_addr_list.

h_addr_list
The address of a list of addresses that point to the host internet addresses returned by this service. This list is ended by the pointer X'00000000'.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyname service returns one of the following:

- 0, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyname service stores the return code. The gethostbyname service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The gethostbyname service can return one of the following values in the Return_code parameter:

Return_code	Explanation
HOST_NOT_FOUND	The host name specified by the Name parameter was not found.
TRY_AGAIN	The host name specified by the Name parameter could not be resolved within the configured time interval. The request can be retried later.
NO_RECOVERY	An unrecoverable error occurred.
NO_DATA	The requested Name parameter is valid, but it does not have a record at the name server.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostbyname service stores the reason code. The gethostbyname service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value.

Reason codes lower than decimal 4096 are z/OS UNIX System Services return codes, and are documented in *z/OS UNIX System Services Messages and Codes*. Reason codes greater than decimal 4096 are returned by the resolver, and are described in *z/OS Communications Server: IP and SNA Codes*.

An assembler macro (EZBREHST) that contains the hostent structure, gethostbyxxxx return codes, and reason codes is shipped in the installation's MACLIB SMP/E DDEF location.

Related services

- "gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address" on page 234

Characteristics and restrictions

The gethostbyname service does not support a fully reentrant environment. The Hostent structure that is returned is allocated at a task level, and is reused on subsequent gethostbyname calls. Therefore, at any time only one call can be occurring within a task. For example, if the mainline task has issued a gethostbyname call that has not completed, a signal handler that interrupts that thread's processing should not invoke the gethostbyname service.

The Hostent structure is freed when the task is terminated.

Examples

For an example using this callable service, see "BPX1GHN (gethostbyname) example" on page 1146.

gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host

Function

The gethostid or gethostname callable service obtains the ID or the name of the socket host.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1HST):	31-bit task or SRB mode
AMODE (BPX4HST):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1HST,(Domain,
              Name_length,
              Name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4HST with the same parameters.

Parameters

Domain
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number that represents a domain. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for valid Domain values.

Name_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Name. If this field is zero, the information that is returned is the host ID. If this field is nonzero, the value that is supplied is the maximum length of the host name that is to be returned.

On return, this field contains the length of the name that is returned, including the trailing null. The size of this field should be less than 4096 bytes (4KB) in length.

Name
Returned parameter

Type: Character

Length:
Length specified by Name_length.

The name of a field that contains the host name on successful return, if the request was gethostname. This name is terminated by a null character if there is sufficient room in the buffer.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostid or gethostname service returns one of the following:

- The host id, if a zero—length Name_length is supplied.
- 0, if a nonzero Name_length is supplied and the name is successfully returned.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostid or gethostname service stores the return code. The gethostid or gethostname service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The gethostid or gethostname service can return one of the following values in the Return_code parameter:

Return_code	Explanation
ENOENT	The domain that was specified was found to be not active. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRDomainNotSupported.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the gethostid or gethostname service stores the reason code. The gethostid or gethostname service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Characteristics and restrictions

These functions work only for AF_INET sockets, and not for AF_UNIX.

Examples

For an example using this callable service, see "BPX1HST (gethostid or gethostname) example" on page 1154.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer

Function

The getitimer callable service stores the current value of the timer specified into a structure.

Requirements

Operation	Environment
Authorization:	Problem Program or Supervisor State, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GTR):	31-bit
AMODE (BPX4GTR):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GTR,(Interval_Type,
              Interval_Value_Adr
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GTR with the same parameters. The Interval_Value_Adr parameter is a doubleword.

Parameters

Interval_Type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a numeric value that identifies the interval timer and format of the structure that is pointed to by Interval_Value_Adr. This parameter can have the following values:

- ITIMER_REAL = Real time (the default if VIRTUAL and PROF are not specified)
- ITIMER_VIRTUAL = Virtual time (CPU time minus system time)
- ITIMER_PROF = CPU time
- ITIMER_MICRO = Initial and reload times are in microseconds (the default if NANO is not specified)
- ITIMER_NANO = Initial and reload times are in nanoseconds

The ITIMER_ constants are defined in the BPXYITIM macro.

Interval_Value_Adr
Supplied parameter

Type: address

Length:
Fullword (doubleword)

A fullword (doubleword) field that contains an address that points to a structure that is defined by the BPXYITIM macro. This structure contains the time remaining and reload values, in seconds and either microseconds or nanoseconds.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getitimer service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getitimer service stores the return code. The getitimer service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getitimer service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value specified for *Interval_Type* is not valid. (JRIntervalTypeInvalid).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getitimer service stores the reason code. The getitimer service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value.

Usage notes

1. The number of seconds that is returned is unsigned and may exceed the setitimer allowable limit. This can happen if alarm is set for up to X'FFFFFFFF' seconds.
2. In 31-bit mode, the first two words returned are seconds, and then micro or nanoseconds. In 64-bit mode, the first doubleword in the structure is seconds, the next word is reserved, and the next word is the micro or nanoseconds. Although the structure returned in 64-bit mode for seconds is a doubleword, the value is the same as if it were a single word.
3. The three interval timers are:
   - ITIMER_REAL, which decrements in real time. A SIGALRM signal is delivered when this timer expires.
   - ITIMER_VIRTUAL, which decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.
   - ITIMER_PROF, which decrements both in process virtual time, and when the system is running on behalf of the process. A SIGPROF signal is delivered when it expires.
   - Nanosecond values are subject to rounding.
   - Reload values may be changed to a system-imposed minimum.

MVS-related information

- “setitimer (BPX1STR, BPX4STR) — Set the value of the iterval timer” on page 680
- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29

Characteristics and restrictions

There are no restrictions on the use of the getitimer service.

Examples

For an example using this callable service, see "BPX1GTR (getitimer) example" on page 1152.

getlogin (BPX1GLG, BPX4GLG) — Get the user login name

Function

The getlogin callable services gets the user login name that is associated with the current process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GLG):	31-bit
AMODE (BPX4GLG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GLG,(Return_value)
```

AMODE 64 callers use BPX4GLG. The address returned is a fullword (below the bar).

Parameters

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword to which the getlogin service returns a pointer to a login name field, or `0`. If a login name is not found, Return_value is set to `0`. If a login name is found, Return_value is set to the address of a field that contains the length of the login name and the login name. The address returned is 31 bits for both AMODE 31 and AMODE 64 callers. The login name length is a fullword. Batch processing has a user name that is associated with a process; this user name is used as the login name. For example:

Usage notes

If this service fails, the process ends abnormally.

Related services

- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name" on page 260
- "getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID" on page 263
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282

Characteristics and restrictions

There are no restrictions on the use of the getlogin service.

Examples

For an example using this callable service, see "BPX1GLG (getlogin) example" on page 1147.

getpeername or getsockname (BPX1GNM, BPX4GNM) — Get the name of a socket or of the peer connected to a socket

See "getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer" on page 272.

getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address

Function

The getnameinfo callable service resolves a socket address into a host name and a service name. The TCP/IP Services resolver attempts to resolve the socket address through a name server, if one is present, or through the local data sets.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GNI):	31-bit
AMODE (BPX4GNI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GNI,(SockAddr,
              SockAddr_Length,
              Service_Buffer,
              Service_Buffer_Length,
              Host_Buffer,
              Host_Buffer_Length,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GNI with the same parameters.

Parameters

SockAddr
Supplied parameter

Type: Structure

Length:
Specified by SockAddr_Length

The name of a field that contains the socket address to be resolved. The socket address consists of an address family, a port number, and an IP address. If the IP address is a link-local IPv6 address, the socket address can also contain a zone index field.

The IP address is resolved to a host name and returned in the Host_Buffer parameter. The port number is resolved to a service name and returned in the Service_Buffer parameter. The zone index field is resolved to an interface name and appended to the host name in the form *hostname%scope information*.

The format of SockAddr is determined by the domain in which the socket descriptor was created. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for additional information on the format of SockAddr.

SockAddr_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the SockAddr parameter.

Service_Buffer
Supplied and returned parameter

Type: Character

Character set:
EBCDIC

Length:
Specified by Service_Buffer_Length

The name of a field into which the service name, resolved from the port number that was specified as part of the SockAddr parameter, is returned as an EBCDIC string. The maximum length of the returned service name is 32

bytes. If the storage specified is inadequate to contain the resolved service name, the service name is returned only up to the specified storage, and truncation can occur.

Service_Buffer_Length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Service_Buffer parameter. Upon return from the getnameinfo service, Service_Buffer_Length contains the length of the name returned in the Service_Buffer parameter.

If Service_Buffer_Length is zero, nothing is returned in Service_Buffer.

Host_Buffer
Supplied and returned parameter

Type: Character

Character set:
EBCDIC

Length:
Specified by Node_Buffer_Length

The name of a field into which the host name, resolved from the IP address that was specified as part of the SockAddr parameter, is returned as an EBCDIC string. The maximum length of the returned host name is 255 bytes. If the storage specified is inadequate to contain the resolved host name, the host name is returned only up to the specified storage, and truncation can occur.

If the IP address specified as part of the SockAddr parameter represents a link-local IPv6 address and the zone index specified as part of the SockAddr parameter is nonzero, then the information returned includes scope information in the form *hostname%scope information*. When the NI_NUMERICSCOPE flag is specified with the Flags parameter, the scope information returned is the zone index value in numeric form (EBCDIC decimal); otherwise, the scope information returned is the interface name associated with the zone index. The maximum length for the combined hostname and scope information remains 255 bytes.

If the NI_NUMERICHOST flag is specified with the Flags parameter, or the host name cannot be located, the IP address, specified as part of the SockAddr parameter, is returned in Host_Buffer in numeric form (EBCDIC decimal).

Host_Buffer_Length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Host_Buffer parameter.

Upon return from the getnameinfo service, Host_Buffer_Length contains the length of the name returned in the Host_Buffer parameter.

If Host_Buffer_Length is zero, nothing is returned in Host_Buffer.

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains flags for controlling the resolution of the socket address.

Flag	Value	Description
NI_NOFQDN	X'00000001'	Only the host name portion of the FQDN is to be returned for local hosts.
NI_NUMERICHOST	X'00000002'	The numeric form of the host's address is to be returned, instead of its name.
NI_NAMEREQD	X'00000004'	If the host name cannot be located, an error or NULL character is to be returned.
NI_NUMERICSERV	X'00000008'	The numeric form of the service name is to be returned (its port number), instead of its name.
NI_DGRAM	X'00000010'	The service is a datagram service (SOCK_DGRAM). The default behavior is to assume that the service is a stream service.
NI_NUMERICSCOPE	X'00000020'	The numeric form of the scope information is to be returned (the zone index), rather than the interface name.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getnameinfo service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getnameinfo service stores the return code. The getnameinfo service returns Return_code only if Return_value is -1.

For a complete list of possible return code values, see *z/OS Communications Server: IP and SNA Codes*. The getnameinfo service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAI_NONAME	The host name does not resolve for the supplied parameters. One of the following conditions occurred: 1. NI_NAMEREQD is set, and the host name cannot be located. 2. Both host name and service name were null.
EAI_BADFLAGS	The flags parameter had an incorrect value.
EAI_FAMILY	The address family was not recognized, or the address length was not valid for the specified family.
EAI_MEMORY	A memory allocation failure occurred.
EAI_AGAIN	The specified host address could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.
EAI_FAIL	An unrecoverable error occurred.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getnameinfo service stores the reason code. The getnameinfo service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS Communications Server: IP and SNA Codes*.

Usage notes

1. The getnameinfo service supports a fully thread-safe environment.
2. You must specify either Service_Buffer and Service_Buffer_Length, or Host_Buffer and Host_Buffer_Length.

Related services

- "freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures" on page 194
- "getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location" on page 205

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1GNI (getnameinfo) example" on page 1148.

getpgid (BPX1GEP, BPX4GEP) — Get the process group ID

Function

The getpgid callable service gets the process group ID of the process whose process ID is equal to the input process ID. If the input process ID is 0, getpgid

returns the process group ID of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GEP):	31-bit
AMODE (BPX4GEP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GEP,(PID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GEP with the same parameter.

Parameters

PID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the process ID for which to get the process group ID. If PID is 0, the process group ID of the calling process is returned.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpgid service returns a process group ID or, if it is not successful, a -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpgid service stores the return code. The getpgid service returns Return_code only if Return_value is -1. For a

complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getpgid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EPERM	The process whose process ID is equal to PID is not in the same session as the calling process.
ESRCH	There is no process with a process ID equal to PID.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpgid service stores the reason code. The getpgid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader" on page 270
- "getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID"
- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253
- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686

Characteristics and restrictions

There are no restrictions on the use of the getpgid service.

Examples

For an example using this callable service, see "BPX1GEP (getpgid) example" on page 1142.

getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID

Function

The getpgrp callable service gets the process group ID (PGID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPG):	31-bit
AMODE (BPX4GPG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPG,(Group_ID)
```

Parameters

Group_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpgrp service places the caller's process group ID.

Usage notes

If getpgrp fails, the process ends abnormally.

Related services

- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702

Characteristics and restrictions

There are no restrictions on the use of the getpgrp service.

Examples

For an example using this callable service, see "BPX1GPG (getpgrp) example" on page 1148.

getpid (BPX1GPI, BPX4GPI) — Get the process ID

Function

The getpid callable service gets the process ID (PID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPI):	31-bit
AMODE (BPX4GPI):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPI,(Process_ID)
```

Parameters

Process_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the getpid service places the caller's process ID.

Usage notes

1. If the getpid service fails, the process abends.
2. To optimize performance, see Appendix I, "Optimizing performance using process- and thread-level information," on page 1329.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "getppid (BPX1GPP, BPX4GPP) — Get the parent process ID"
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304

Characteristics and restrictions

There are no restrictions on the use of the getpid service.

Examples

For an example using this callable service, see "BPX1GPI (getpid) example" on page 1149.

getppid (BPX1GPP, BPX4GPP) — Get the parent process ID

Function

The getppid callable service gets the parent process ID (PPID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN

Operation	Environment
AMODE (BPX1GPP):	31-bit
AMODE (BPX4GPP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPP,(Return_value)
```

AMODE 64 callers use BPX4GPP.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getppid service returns the parent process ID of the calling process.

Usage notes

If the getppid service fails, the process ends abnormally.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304

Characteristics and restrictions

There are no restrictions on the use of the getppid service.

Examples

For an example using this callable service, see "BPX1GPP (getppid) example" on page 1150.

getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process

Function

The getpriority callable service gets the scheduling priority of a specific process or group of processes.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPY):	31-bit
AMODE (BPX4GPY):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPY,(Which,
              Who,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPY with the same parameters.

Parameters

Which

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

- PRIO_PROCESS = Indicates that the Who parameter is to be interpreted as a process ID
- PRIO_PGRP = Indicates that the Who parameter is to be interpreted as a process group ID
- PRIO_USER = Indicates that the Who parameter is to be interpreted as a user ID

The PRIO_ constants are defined in the BPXYCONS macro (see "BPXYCONS — Constants used by services" on page 952).

Who

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID or User ID whose priority is to be obtained. The Which parameter indicates how this parameter is to be interpreted. A value of zero for this parameter specifies the current process, process group or User ID.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpriority service returns the priority value of the specified process, or -1 if it is not successful.

Because the getpriority service can return the value -1 on successful completion, it is necessary to set the Return_code parameter to 0 before a call to the getpriority service. If getpriority returns the value -1, the Return_code parameter can be checked to see if the service was successful, or if an error occurred.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpriority service stores the return code. The getpriority service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getpriority service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of the Which parameter was not recognized; or the value of the Who parameter is not a valid process ID, process group ID or user ID.
ESRCH	No process could be located using the Which and Who parameter values that were specified.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpriority service stores the reason code. The getpriority service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the supplied Who and Which values specify more than one process, the lowest priority value found among the specified processes is returned.
2. The setting of a process's priority value has an equivalent effect on a process's nice value, as they both represent the process's relative CPU priority. For example, setting the priority value of a process via the setpriority service to its maximum value (19) has the effect of increasing its nice value to its maximum value (2*NICE_ZERO)-1, and is reflected on the nice, getpriority and setpriority services. The NICE_ZERO constant is defined in BPXYCONS. (See “BPXYCONS — Constants used by services” on page 952.)

Related services

- "setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process" on page 688
- "nice (BPX1NIC, BPX4NIC) — Change the nice value of a process" on page 432

Characteristics and restrictions

There are no restrictions on the use of the getpriority service.

Examples

See "BPX1GPY (getpriority) example" on page 1151 for an example using this callable service.

getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database

Function

The getpwent callable service gets information about a user. Each time you use the getpwent service, you get information about the next user entry in the user database.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPE):	31-bit
AMODE (BPX4GPE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPE,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPE with the same parameters.

Parameters

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the getpwent service returns an address, or 0.

If no more user entries exist in the user database, or if an error is encountered, Return_value is set to 0.

If an entry is found, Return_value is set to the address of a data area that is mapped by the BPXYGIDN macro. The first area contains the fullword length of the user name, followed by the user name padded with blanks. See "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwent service stores the return code, or 0. The getpwent service returns Return_code only if Return_value is 0. Return_code is 0 when no more user entries exist in the database. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getpwent service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSSAF2ERR	The system authorization facility (SAF) or RACF Get GMAP service had an error.
EMVSSAFEXTRERR	The SAF or RACF RACROUTE EXTRACT service had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwent service stores the reason code, or 0. The getpwent service returns Reason_code only if Return_value is 0. Reason_code is 0 when no more user entries exist in the database. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF Return Code	RACF Reason Code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery
8	20	The current user is incompletely defined.

Usage notes

1. The getpwent service is intended to be used to search the user database sequentially. The first call to this service from a given task returns a pointer to the first user entry in the user database. Subsequent calls from the same task return a pointer to the next user entry found that is a defined OMVS user. When the end of the data base is reached, a null pointer is returned (RV and RS are both set to 0).

A user entry is not returned for users that are not defined as OMVS users. This includes:

- Users that do not have a RACF OMVS segment defined
- Users with a RACF OMVS segment defined but with no UID defined
- Users whose DFLTGRP does not have a valid OMVS GID defined

If a user does not have a RACF OMVS segment but is given access to the OMVS DEFAULT user, no entry is returned on a getpwent call. This is because the user is not defined as an OMVS user in the user database.

2. The setpwent service can be used to reset this sequential search. The next getpwent service used from the same task after a call to the setpwent service returns a pointer to the first user entry. The next getpwent service used after an end-of-file indication (a null pointer) is returned also returns a pointer to the first user entry. The use of setpwent after end-of-file is therefore optional.
3. The return value points to data that may change or go away after the next getpwuid, getpwnam, or getpwent service request from that task. Each task manages its own storage separately. Move data to the program's storage if it is needed for future reference.
4. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.

Related services

- "getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name"
- "getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID" on page 263
- "setpwent (BPX1SPE, BPX4SPE) — Reset the user database" on page 691

Characteristics and restrictions

There are no restrictions on the use of the getpwent service.

Examples

For an example using this callable service, see "BPX1GPE (getpwent) example" on page 1149.

getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name

Function

The getpwnam callable service gets information about a user. You specify the user by user name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPN):	31-bit
AMODE (BPX4GPN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPN,(User_name_length,
              User_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPN with the same parameters.

Parameters

User_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of User_name.

User_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the User_name_length parameter

The name of a field of length User_name_length that contains the name of the user that the program wants information about. The name is specified in the Resource Access Control Facility (RACF) command that defines the user to the system.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the getpwnam service returns an address, or `0`.

If no entry for the specified group name is found, Return_value is set to `0`.

If an entry is found, Return_value is set to the address of the BPXYGIDN macro. See “BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 969.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwnam service stores the return code. The getpwnam service returns Return_code only if Return_value is `0`. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The getpwnam service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	User_name_length is incorrect; or the user name has an illegal first character (JRUserNameBad).
EMVSSAF2ERR	The system authorization facility (SAF) Get GMAP service had an error.
EMVSSAFEXTRERR	The SAF RACROUTE EXTRACT service had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwnam service stores the reason code. The getpwnam service returns Reason_code only if Return_value is `0`. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR or EMVSSAFEXTRERR, the reason code contains the RACF return and reason codes, respectively, in the two low-order bytes, as follows:

- For EMVSSAF2ERR, the reason code contains the return code and reason code from the RACF getGMAP service. For a detailed description of the return code and reason code values for the getGMAP service, see *z/OS Security Server RACF Callable Services*.
- For EMVSSAFEXTRERR, the reason code contains the return code and reason code from the RACROUTE REQUEST=EXTRACT service. For a detailed description of the return code and reason code values for the RACROUTE service, see *z/OS Security Server RACROUTE Macro Reference*.

RACF Get GMAP service return and reason code values include the following:

RACF Return Code	RACF Reason Code	Explanation
8	0	No profile found.
8	4	If the search is by GID: The GID is not defined. If the search is by group name: The current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

For a more detailed description of the RACF Get GMAP service return and reason code values, see *z/OS Security Server RACROUTE Macro Reference*.

Usage notes

1. If an entry for the specified User_name is not found in the user database, an address of 0 is returned as the Return_value parameter.
2. Return_value points to data that may change or go away after the next getpwuid, getpwnam, or getpwent service request from that task. Each task manages its own storage separately. Move data to your own dynamic storage if you need it for future reference.
3. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX services.
4. If the BPX.UNIQUE.USER profile is defined, a call to BPX1GPN for a user ID that does not have an OMVS segment configured as part of its security profile causes BPX1GPN to assign OMVS segments to the user ID. See Setting up default OMVS segments in *z/OS UNIX System Services Planning* .

Related services

- "getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database" on page 258
- "getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID"
- "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245

Characteristics and restrictions

There are no restrictions on the use of the getpwnam service.

Examples

For an example using this callable service, see "BPX1GPN (getpwnam) example" on page 1149.

getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID

Function

The getpwuid callable service gets information about a user. You specify the user by user ID (UID).

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPU):	31-bit
AMODE (BPX4GPU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPU,(User_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPU with the same parameter.

Parameters

User_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the user ID of the user you want information about.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword to which the getpwuid returns an address, or 0. If no entry for the specified user ID is found, Return_value is set to 0. If an entry is found, Return_value is set to the address of the BPXYGIDN mapping macro. See "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwuid service stores the return code. The getpwuid service returns Return_code only if Return_value is 0. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The getpwuid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSSAF2ERR	The system authorization facility (SAF) or RACF Get GMAP or Get UMAP service had an error.
EMVSSAFEXTRERR	The SAF or RACF RACROUTE EXTRACT call had an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getpwuid service stores the reason code. The getpwuid service returns Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR or EMVSSAFEXTRERR contains the RACF return and reason codes, respectively, in the two low-order bytes.

For a more detailed description of the RACF Get GMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	For a search by GID: the GID is not defined. For a search by group name: the current group is not defined.
8	8	The group name is not defined.
8	12	There was an internal error during RACF processing.
8	16	Recovery could not be established.
8	20	The current group is incompletely defined.

For a more detailed description of the RACF Get UMAP service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	For a search by UID: the UID is not defined. For a search by user ID: The user is not defined.
8	8	The user ID is not defined.
8	12	An internal error occurred during RACF processing.
8	16	Unable to establish recovery.
8	20	The user is incompletely defined.

Usage notes

1. Return_value points to data that can change or go away after the next getpwuid, getpwnam, or getpwent service request from the task. Each task manages its own storage separately. Move data to the program's dynamic storage if the program needs it for future reference.
2. The storage is key 0 nonfetch-protected storage that is managed by z/OS UNIX.
3. Most systems have multiple user IDs defined to have UID=0. It is impossible to predict which user ID will be returned on a call to getpwuid with a UID=0.
4. Performance degradation can occur if you use this service when the Virtual Lookaside Facility (VLF) is not active. For more information, see Tuning performance and the section on establishing UNIX security in *z/OS UNIX System Services Planning*.

Related services

- “getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database” on page 258
- “getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name” on page 260
- “getlogin (BPX1GLG, BPX4GLG) — Get the user login name” on page 245

Characteristics and restrictions

There are no restrictions on the use of the getpwuid service.

Examples

For an example using this callable service, see "BPX1GPU (getpwuid) example" on page 1151.

getrlimit (BPX1GRL, BPX4GRL) — Get resource limits

Function

The getrlimit callable service gets hard and soft resource limits for the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GRL):	31-bit
AMODE (BPX41GRL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GRL,(Resource,
              Rlimit,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GRL with the same parameters.

Parameters

Resource

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the resource for which to get the hard and soft limits. This parameter can have one of the following values:

Constant Name	Description
RLIMIT_MEMLIMIT	Limit of 1-megabyte segments above the 2-gigabyte addressing range

Constant Name	Description
RLIMIT_CORE	Limit size of core dump
RLIMIT_CPU	Limit CPU time per process
RLIMIT_FSIZE	Limit file size
RLIMIT_NOFILE	Limit number of open files
RLIMIT_AS	Limit address space size

The RLIMIT_ constants are defined in the BPXYCONS macro. (See "BPXYCONS — Constants used by services" on page 952.)

Rlimit
Supplied parameter

Type: Structure

Length:
The length of the rlimit structure

The name of an rlimit structure in which the hard (maximum) and soft (current) limit values for the resource that is identified by the resource parameter are to be placed. Macro BPXYRLIM defines the rlimit structure. (See "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033.) Each limit value contains two fullwords. For all resources except RLIMIT_FSIZE, the upper fullword for each limit value is set to zero before returning to the caller.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrlimit service returns a value of zero if it is successful, and -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrlimit service stores the return code. The getrlimit service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getrlimit service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	An incorrect resource was specified. The following reason code can accompany the return code: JrInvalidResource.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrlimit service stores the reason code. The getrlimit service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "setrlimit (BPX1SRL, BPX4SRL) — Set resource limits" on page 698
- "getrusage (BPX1GRU, BPX4GRU) — Get resource usage"

Characteristics and restrictions

The current high-memory limit that is returned in RLIM_CUR_DW is the exact current memory limit. When the MEMLIMIT value is set by SMF, the maximum number of bytes that can be returned in RLIM_MAX_DW is X'FFFFFFFF00000000'. When the MEMLIMIT value is set by a z/OS UNIX service, the highest value that is supported is 16383 petabytes, or X'FFFC00000000000000'. It is therefore possible for the current MEMLIMIT to be larger than the maximum MEMLIMIT supported by z/OS UNIX System Services.

Examples

For an example using this callable service, see "BPX1GRL (getrlimit) example" on page 1151.

getrusage (BPX1GRU, BPX4GRU) — Get resource usage

Function

The getrusage callable service gets information about resources that are used by the calling process or its terminated and waited-for child processes.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GRU):	31-bit
AMODE (BPX4GRU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GRU,(Who,
              Rusage,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GRU with the same parameters.

Parameters

Who
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates for whom to get the resource usage. This parameter can have one of the following values:

Constant Name	Description
RUSAGE_SELF	Rusage for current process
RUSAGE_CHILDREN	Rusage for terminated children

The RUSAGE_ constants are defined in the BPXYCONS macro (see "BPXYCONS — Constants used by services" on page 952).

Rusage
Supplied parameter

Type: Integer

Length:
Fullword

The name of an rusage structure that is to contain the values for resource usage. Macro BPXYRLIM defines the rusage structure (see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrusage service returns a value of zero if it is successful, and -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrusage service stores the return code. The getrusage service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getrusage service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	An incorrect Who value was specified. The following reason code can accompany the return code: JrInvalidWho.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getrusage service stores the reason code. The getrusage service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Resource information returned for multithreaded processes may be inaccurate.

Related services

- "setrlimit (BPX1SRL, BPX4SRL) — Set resource limits" on page 698
- "getrlimit (BPX1GRL, BPX4GRL) — Get resource limits" on page 266

Characteristics and restrictions

There are no restrictions on the use of the getrusage service.

Examples

For an example using this callable service, see "BPX1GRU (getrusage) example" on page 1152.

getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader

Function

The getsid callable service gets the process group ID of the session leader of the process whose process ID is equal to the input process ID. If the input process ID is 0, the service returns the process group ID of the session leader of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GES):	31-bit
AMODE (BPX4GES):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GES,(PID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GES with the same parameter.

Parameters

PID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the process ID that identifies the process whose session leader's process group ID should be obtained. If PID is 0, the process group ID of the calling process's session leader is returned.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsid service returns a process group ID, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsid service stores the return code. The getsid service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getsid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EPERM	The process whose process ID is equal to PID is not in the same session as the calling process.
ESRCH	There is no process with a process ID equal to PID.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsid service stores the reason code. The getsid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702
- "getpgid (BPX1GEP, BPX4GEP) — Get the process group ID" on page 250
- "getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID" on page 252
- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253
- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686

Characteristics and restrictions

There are no restrictions on the use of the getsid service.

Examples

For an example using this callable service, see "BPX1GES (getsid) example" on page 1143.

getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer

Function

The getsockname or getpeername callable service obtains the name of a socket or the name of a peer connected to a socket.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1GNM):	31-bit task or SRB mode
AMODE (BPX4GNM):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4GNM with the same parameters.

Format

```
CALL BPX1GNM,(Socket_descriptor,
              Operation,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the service is to be performed.

Operation
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the operation option. The value of this field determines whether the service to be performed is getsockname or getpeername. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for valid Operation values.

Sockaddr_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The size of this field must be less than 4096 bytes (4KB) in length. The size of the buffer that is specified must be the maximum length that the sockaddr could be on output.

Sockaddr
Supplied and returned parameter

Type: Character

Length:
Length specified by Sockaddr_length.

The name of a field in which the socket name or peer name is to be returned. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for valid Operation values.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockname or getpeername service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockname or getpeername service stores the return code. The getsockname or getpeername service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getsockname or getpeername service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINVAL	The length that is specified by the sockaddr_length operand is too small to allow the name to be returned. The following reason code can accompany the return code: JRSocketCallParmError.
ENOBUFS	Unable to obtain a buffer. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	**getpeername()** was specified and the socket is not connected.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockname or getpeername service stores the reason code. The getsockname or getpeername service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Characteristics and restrictions

There are no restrictions on the use of the getsockname or getpeername service.

Examples

For an example using this callable service, see "BPX1GNM (getpeername or getsockname) example" on page 1148.

getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket

Function

The getsockopt or setsockopt callable service gets or sets options that are associated with a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1OPT):	31-bit task or SRB mode
AMODE (BPX4OPT):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1OPT,(Socket_descriptor,
              Operation,
              Level,
              Option_name,
              Option_data_length,
              Option_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4OPT with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the service is to be performed.

Operation
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the operation option. The value of this field determines whether the service to be performed is getsockopt, setsockopt, or setibmsockopt. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for valid Operation values.

Level
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the level for which the option is set or being set.

Option_name
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field in which the value of the option name is provided. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for valid Option_name values.

Option_data_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Option_data. On return from getsockopt, this field contains the size of the data that was returned in Option_data. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer specified should be the maximum length that the option_data could be on output.

Option_data
Supplied and returned parameter

Type: Character

Length:
Length specified by Option_data_length.

The name of a field that contains the data that is associated with or is to be associated with the socket. On return from getsockopt, this field contains the data that is associated with the socket. For setsockopt, this field provides the data that is to be associated with the socket.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockopt or setsockopt service returns one of the following:

- `0` if the request is successful.
- `-1` if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockopt or setsockopt service stores the return code. The getsockopt or setsockopt service returns Return_code. only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getsockopt or setsockopt service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EDOM	An argument that is too large was supplied on the call.
EINVAL	An incorrect argument was supplied on the call. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRLevelNotSupp.
ENOBUFS	A buffer could not be obtained. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutofSocketCells.
ENOPROTOOPT	An option_name that was specified for getsockopt is not supported. An incorrect value was specified on the Level parameter. SOL_SOCKET must be specified. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRLevelNotSupp, JRInvalOpOpt, JROptNotSupp.
ENOSYS	For AF_UNIX, setsockopt was specified; it is not supported. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSetNotSupp.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRMustBeSocket.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getsockopt or setsockopt service stores the reason code. The getsockopt or setsockopt service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The socket descriptor must refer to an open socket.
2. For AF_UNIX sockets, the **getsockopt()** service supports the following option_names only: so_acceptconn, so_type, and so_secinfo.
3. The level of support for this service depends on the particular socket stack you have installed. Some options might not be defined by the BPXYSOCK macro. Refer to the documentation for the product you are using to determine the socket options it supports. For example, see *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference* for the z/OS Communications Server socket stack, and *z/OS Communications Server: IPv6 Network and Application Design Guide* for information on IPv6 socket options.
4. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Characteristics and restrictions

There are no restrictions on the use of the getsockopt or setsockopt service.

Examples

For an example using this callable service, see "BPX1OPT (getsockopt or setsockopt) example" on page 1169.

__getthent (BPX1GTH, BPX4GTH) — Get thread data

Function

The __getthent callable service obtains data that describes the status of a process and its threads. This data includes, but is not limited to, running time, reasons for waiting, syscalls made, files open, and signal information. The caller can access one process on each request, with from none to all of its threads.

You can invoke this service in several ways:

- For the first accessible (by SAF standards) process (the lowest relative process in the system)
- For a specific process, if the process ID is known and the process is accessible
- For a specific thread within a specific accessible process, if both IDs are known
- For the next accessible process or thread after one just returned
- For a specific address space ID or user ID

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GTH):	31-bit
AMODE (BPX4GTH):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	No latches should be held
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1GTH,(Input_length,
               Input_address,
               Output_length,
               Output_address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4GTH with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Input_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the value PGTHA#LEN.

Input_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of an area mapped by PGTHA; see "BPXYPGTH — Map the __getthent input/output structure" on page 1009. The input area must be initialized to hex zeros and then the requested options must be set in the PGTHA section.

Output_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the length of the output buffer. Some requests could be satisfied by the minimum buffer size of 128 bytes; whereas a request for all options of a process with maximum resources could exceed half a million bytes.

Output_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of an area mapped by PGTHB and other PGTH sections; see "BPXYPGTH — Map the __getthent input/output structure" on page 1009.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __getthent service returns `0` if the request is successful, or -1 if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __getthent service stores the return code. The __getthent service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __getthent service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The input_addr (to input_length) or the output_addr (to output_length) contains the address of storage that the caller is not authorized to access (JrBadInputError or JrBadOutputError).
EINVAL	One of the following occurred: • The input area (PGTHA) contains a value that is not valid (JrPidBad, JrBadOptions) • The input_address is zero (JrBadInputBuffAddr) • The output_address is zero (JrBadOutputBuffAddr) • The input_length is incorrect (JrInvParmLength) • The output_length is too small (JrBuffTooSmall)
EAGAIN	PGTHAPID is undergoing changes, and the z/OS UNIX control blocks are not properly connected (JrBlocksInFlux).
EACCES	PGTHAPID is not accessible to the caller (JrInaccessible).
ESRCH	PGTHAPID was not found (JrPIDNotFound); or PGTHATHID was not found (JrThreadNotFound).
ENOBUFS	The __getthenent service could not obtain a local work area (JrNoBufStorage).

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __getthent service stores the reason code. The __getthent service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The system extracts fields PgthJTTime and PgthJLoginName from the TCB and OTCB of the target process by scheduling an SRB to run in that process's address space. If the address space is terminating, the SRB might not run, and these fields are returned as zero.

If this information is not required, you can set option PgthAThreadFast. This action can improve performance on a very busy system because the SRB will not be scheduled. An indication of the thread being in an MVSWAIT is also extracted from the TCB. If the thread is in an MVS WAIT, then PgthJStatus is set to Y. When the SRB is not scheduled, then this field is not seen as Y. If the thread is not in any other kernel wait, then PgthJStatus2 is set to R, which indicates running or non-kernel wait.

2. Typically a user starts with PGTHAPID=PGTH#FIRST, processes the data, and sets PgthAContinue=PgthBContinue to continue with the next thread or the next process, until a Return_value of -1 is reached.
3. The setting of PgthBContinue steps the caller to the next process or thread. If this action is not desirable, do not use PgthAContinue.
4. The Output_length required varies with the PGTHAFLAGs selected and the characteristics of the process and its threads. Most processes should fit in 4000 bytes. If a process has 65 000 files opened, 3/4 of a million bytes would be needed. An arbitrary minimum size of 128 bytes is necessary to avoid an error of EINVAL (JrBuffTooSmall).
5. EINVAL (JrBuffTooSmall) can also indicate that there is insufficient room for at least one PgthJ area. This could happen even with a buffer in excess of 4000 bytes.
6. Field PgthJWTime indicates how long the thread has been in most waits that are internal to z/OS UNIX. It is meaningful only if it is nonzero.
7. The high-memory values for the process that are returned in PgthCMemUsage and PgthCMemPages are described in bytes. Each value can be displayed as a 4-byte number or as a 3-byte value followed by a qualifier (for example, 50M for fifty megabytes). High-memory values are described with the truncated value of the exact system state; the real values might be slightly higher. The maximum number of bytes that can be returned in PgthCMemUsage and PgthCMemPages is 16383 petabytes.
8. Flag PGTHCRESPAWN indicates that the process was started with the respawn attribute. See “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 780 for more information about the respawn attribute.
9. Flag PGTHCBLOCKING indicates a shutdown-blocking process and flag PGTHCPERM indicates a permanent process.
10. When the command or argument data is not terminated by a null character, the command or argument is terminated with a null character by the __getthent service.
11. If PgthACommand is ON and PgthACommandLong is OFF, the maximum length returned in the PGTHF area is 1024. If PgthACommand is OFF and PgthACommndLong is ON, the length returned in the PGTHF area can be greater than 1024. If both bits are on, the new setting, PgthACommndLong, is honored.
12. Together, the PgthE output area (for path) and the PgthF output area (for the command and arguments) can take up to 2048 bytes space. When PgthACommand is specified, 1024 bytes are reserved for the PgthE section (path name) and 1024 bytes are reserved for the PgthF section (the command and arguments). Up to only 1024 bytes of data are returned for the PgthF section.

    When PgthACommandLong is specified, it indicates that the PgthF section can be greater than 1024 bytes. If path is requested (PgthAPath), then the PgthE section is filled in and the remainder of the 2048 bytes can be used for

the PgthF section. (The PgthF section length will be 2048 bytes less the length of the PgthE section returned). If path is not requested, then the PgthF section can be up to 2048 bytes.

If PgthACommand and PgthACommandLong are both specified, PgthACommandLong is honored.

13. PgthaFilePath must be specified with PgthaFileData. If it is not, then PgthaFilePath is ignored. If PgthaFilePath is specified, a significant amount of data might be returned if the process has many open files. Information is not returned if more than 16MB is required or if the input buffer is too small. The path names that are returned are full, relative, or partial and is intended for diagnostic use. If a partial path name is returned, the PgthHPathTrunc bit is set.

Related services

- "w_getpsent (BPX1GPS) — Get process data" on page 897

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1GTH (__getthent) example" on page 1152.

getuid (BPX1GUI, BPX4GUI) — Get the real user ID

Function

The getuid callable service gets the real user ID (UID) of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GUI):	31-bit
AMODE (BPX4GUI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GUI,(User_ID)
```

AMODE 64 callers use BPX4GUI.

Parameters

User_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the getuid service returns the real user ID of the calling process.

Usage notes

If the getuid service fails, the process ends abnormally.

Related services

- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

There are no restrictions on the use of the getuid service.

Examples

For an example using this callable service, see "BPX1GUI (getuid) example" on page 1153.

getwd (BPX1GWD, BPX4GWD) — Get the pathname of the working directory

Function

The getwd callable service gets the pathname of the working directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GWD):	31-bit
AMODE (BPX4GWD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GWD,(Buffer_length,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GWD with the same parameters.

Parameters

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the buffer to which the getwd service returns the pathname of the directory. Buffer_length must be large enough to accommodate the actual length of the pathname plus one (for the terminating null). Length of zero has special meaning; see the usage notes.

Buffer
Parameter supplied and returned

Type: Character string

Character set:
No restrictions

Length:
Specified by the Buffer_length parameter

The name of the buffer that will hold the pathname of the working directory.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getwd service returns the length of the pathname that is in the buffer, if the request is successful, or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getwd service stores the return code. The getwd service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The getwd service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process did not have permission to read or search a component of the working directory's pathname.
EINVAL	The Buffer_length specified was not valid. The following reason code can accompany the return code: JRBufLenInvalid.
EIO	An input/output error occurred.
ENOENT	A component of a pathname does not exist. This is returned if a component of the working directory pathname was deleted.
ERANGE	The specified Buffer_length is less than the length of the pathname of the working directory. The specified Buffer_length is zero, and the length of the pathname of the working directory is larger than PATH_MAX bytes.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the getwd service stores the reason code. The getwd service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. PATH_MAX plus 1 for the terminating null character is a reasonable size for the Buffer.
2. If a Buffer_length of zero is passed to this service, the generated null terminated pathname is stored in the named buffer up to a maximum of PATH_MAX + 1 bytes. Buffer is assumed to be of sufficient size to contain the pathname derived by the getwd service. If the generated pathname is larger than PATH_MAX bytes, the return value is -1 and Return_code is ERANGE.

Related services

- "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215

Characteristics and restrictions

There are no restrictions on the use of the getwd service.

Examples

For an example using this callable service, see "BPX1GWD (getwd) example" on page 1153.

givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program

Function

The givesocket callable service makes a specified socket available to a takesocket call to be issued by another program.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GIV):	31-bit
AMODE (BPX4GIV):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GIV,(Socket_descriptor,
              Clientid,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GIV with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the givesocket is to be done.

Clientid
Supplied and returned parameter

Type: Structure

Length:
Length of BPXYCID

The name of a structure that contains Clientid information identifying the (slave) program to which the socket is to be given. This information is typically obtained with the getclientid service issued by the slave and passed to the server. Clientid input may be as follows:

- CIdDomain - domain of the socket being given. See “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043 for more information about the values defined for this field.
- CIdName - one of the following:
  - Blanks - allows any program to take the socket using the takesocket service.
  - The slave program's address space name, left-justified, and padded with blanks.
  - A fullword of binary zeroes followed by the slave program's process id.

- CIdTask - used only if an address space name was supplied in the CIdName field. One of the following:
  - Blanks - allows any subtask in the address space to take the socket.
  - The slave program's subtask identifier.
- CIdReserved - one of the following:
  - The CIdType field of the CIdReserved area set to CId#Close. This results in the givesocket service doing a close of the input socket and returning a unique socket token in the CIdSockToken field of the CIdReserved area.
  - The CIdType field of the CIdReserved area set to CId#Select. This indicates that the application intends to block on **select()** for exceptions, waiting for the takesocket call to occur before closing the socket. It allows a select call to return exception status even if that call is made after the takesocket call, as long as the socket was not closed. It also results in the connection being severed if the giver closes the socket before it has been taken.
  - The CIdType field of the CIdReserved area set to zeros. The program will not be calling **select()** to coordinate with the taker of the socket. Either the socket is not going to be closed or the giver and taker have some other method of coordination for the giver to know when the taker has called **takesocket()**.

    Note: If **select()** for exception is called before **takesocket()** the select will return when **takesocket()** is called but if **select()** is called after **takesocket()** it will hang. CidType of Cid#Select should be used if **select()** is going to be called by the giver.

    If the given socket is closed before the takesocket() is issued it is possible for that socket descriptor number to be reused in the giver's process so a sequence of accept(), givesocket(), and close() calls issued several times before any takesocket() calls can result in several sockets with the same descriptor number waiting to be taken. In this case the oldest given socket will be taken and first in first out order is used.

    Note: If **select()** is called when there are several given sockets with the same descriptor number waiting to be taken then **select()** will operate on the current active socket for that descriptor so it effectively waits for the last (newest) given socket to be taken and the order becomes last in first out.

The Clientid is a returned parameter only if the CIdType field in the CIdReserved area is set to CId#Close. A unique token for the given socket is then returned in the CIdToken field of the CIdReserved area. This token, instead of the socket descriptor, is to be passed to the slave program to be used on the takesocket service. The token must be used, rather than the socket descriptor, because the socket being given will be closed, and the socket descriptor may be reused. See “BPXYCID — Map the returning structure for getclientid()” on page 951 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the givesocket service returns one of the following:

- 0 if successful.
- -1 if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the givesocket service stores the return code. The givesocket service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The givesocket service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The Socket_descriptor is not valid, or the socket has already been given.
EFAULT	Using the Clientid parameter as specified would result in an attempt to access storage that is outside the caller's address space.
EINVAL	The Clientid parameter does not specify a valid client identifier; or the CIdDomain in the Clientid parameter does not match the actual domain of the input Socket_descriptor.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the givesocket service stores the reason code. The givesocket service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The program that does the givesocket will always be able to do a takesocket for the given socket, even if its identity does not match that of the Clientid input of the givesocket.
2. The Clientid output of the getclientid service (issued by the slave program and passed to the server) is intended to be used as the input Clientid of the givesocket service. If you use a FunctionCode of 2 on the getclientid service to obtain Clientid information that will then be used as the Clientid input of the givesocket service, you will ensure the best performance of the givesocket service, and the most secure identification of the validity of the taker.
3. Setting the CIdType field of the CIdReserved area in the Clientid structure to CId#Close improves performance, by allowing the givesocket service to automatically close the socket, rather than requiring the application to do a select and a close.
4. If the given socket is not closed, it can still be used, even after the takesocket() has been done. The socket can be shared between the giver and taker in the same way that an inherited socket can be shared between parent and child after a fork() has been issued.

Related services

- "getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier" on page 213
- "takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program" on page 821

Characteristics and restrictions

There are no restrictions on the use of the givesocket service.

grantpt (BPX1GPT, BPX4GPT) — Grant access to the slave pseudoterminal

Function

The grantpt callable service changes the mode and ownership of the slave pseudoterminal device that is identified by the file descriptor. The file descriptor must be the file descriptor of the corresponding master pseudoterminal. The user ID of the slave is set to the real UID of the calling process. The group ID is set to the group ID that is associated with the group name that was specified by the installation in the TTYGROUP initialization parameter. The permission mode of the slave pseudoterminal is set to be readable and writable by the owner, and writable by the group.

You can provide secure connections either by using grantpt and unlockpt, or by issuing the first open against the slave pseudoterminal from the user ID or process that opened the master terminal.

Requirements

Operation	**Environment**
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GPT):	31-bit
AMODE (BPX4GPT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GPT,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GPT with the same parameter.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the grantpt service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the grantpt service stores the return code. The grantpt service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The grantpt service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The slave pseudoterminal was opened before grantpt, or a grantpt has already been issued. In either case, slave pseudoterminal permissions and ownership have already been updated. If you use grantpt to change slave pseudoterminal permissions, you must issue grantpt between the master open and the first pseudoterminal open. The grantpt service can only be requested once.
EBADF	The File_descriptor parameter does not specify a valid open file descriptor.
EINVAL	The file descriptor is not associated with a master pseudoterminal device.
ENOENT	During lookup, the slave pseudoterminal device was not found.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the grantpt service stores the reason code. The grantpt service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal master/slave pair" on page 875

Characteristics and restrictions

There are no restrictions on the use of the grantpt service.

IPCSDumpOpenClose (BPXGMCDE, BPXGMCD4) — MVS IPCS dump open/close service

Function

The IPCSDumpOpenClose service opens (and closes) a dump that has been captured with an SVC dump, a SYSMDUMP, or the DUMP command. Once the dump has been opened, it can be processed with the BPXGMPTR callable service, which reads storage, registers, program attributes, and other dump-related information.

Requirements

Operation	**Environment**
Authorization:	Problem state, user PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPXGMCDE):	31-bit
AMODE (BPXGMCD4):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPXGMCDE,(Dcor_Open,
               LevelIndicator,
               DumpDataSetName,
               LogDataSetName,
               ExecDataSetName,
               ClistDataSetName,
               DDIRStr,
               ErrorMsgPtr,
               Return_code,
               Return_Value1,
               Return_Value2,
               Return_Value3)

CALL BPXGMCDE,(DCOR_Close,
               OpenToken)
```

AMODE 64 callers use BPXGMCD4 with the same parameters.

Parameters

Dcor_Open
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the constant for an open request, DCOR_OPEN. The value of this constant is defined in the BPXYDCOR macro (see "BPXYDCOR — dbx cordump cache information" on page 959). If the open request is completed successfully, the BPXGMCDE (BPXGMCD4) service returns a nonzero open token in register 15. This token is used by the BPXGMCDE (BPXGMCD4) close function and the BPXGMPTR (BPXGMPT4) callable service.

If the open request is not successful, the BPXGMCDE (BPXGMCD4) service returns a token value of zero, with explicit failure information in the Return_code, Return_value1, Return_value2, and Return_value3 fields.

LevelIndicator
Parameter supplied and returned

Type: Address

Length:
Fullword

The name of a fullword that contains the release level of the DCOR services. The level number is defined in BPXYDCOR.

DumpDataSetName
Supplied parameter

Type: Character string

Length:
Variable

The name of a required null (X'00'-terminated) character string that provides the name of the dump that is to be opened. The name may be an MVS data set name or a z/OS UNIX file name. An MVS data set name must begin with a double slash (//); otherwise the name is considered to be the name of a file. To indicate that an MVS data set name is fully qualified, quotes should be used on each side of the data set name (//'MVS.DATA.SET'). When quotes are not used to fully qualify the data set name, the login user ID is prefixed to the data set name (userid.MVS.DATA.SET).

LogDataSetName
Supplied parameter

Type: Character string

Length:
Variable

The name of an optional null (X'00'-terminated) character string that provides the name of a log data set. The name must be an MVS data set name; a z/OS UNIX file cannot be used as a log data set. The data set name is considered to be fully qualified; quotes may be used but they are not necessary.

TSO messages that are generated from running IPCS are written to the log data set. This log is useful in problem determination, especially when the IPCS environment does not get established.

ExecDataSetName
Supplied parameter

Type: Character string

Length:
Variable

The name of an optional null (X'00'-terminated) character string that provides the name of the MVS PDS data set that is to be used in place of SYS1.SBPXEXEC, which is the default. SYS1.SBPXEXEC contains the REXX exec BPXTIPCS, which is used to create a dump directory and establish the IPCS environment.

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: quotes may be used, but they are not necessary.

ClistDataSetName
Supplied parameter

Type: Character string

Length:
Variable

The name of an optional null (X'00'-terminated) character string that provides the name of the MVS PDS data set that is to be used in place of SYS1.SBLSCLI0, which is the default. SYS1.SBLSCLI0 contains IPCS CLISTs, including BLSCDDIR, which is used to allocate a temporary or permanent dump directory.

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: quotes may be used, but they are not necessary.

DDIRStr
Supplied parameter

Type: Character string

Length:
Variable

The name of an optional null (X'00'-terminated) character string that is used to tailor the use of the IPCS dump directory on the invocation of the BLSCDDIR command. It may contain any of the parameters that are accepted by BLSCDDIR. (See *z/OS MVS IPCS Commands*.)

The name must be an MVS data set name; z/OS UNIX files are not supported for this parameter. The data set name is considered to be fully qualified: quotes may be used, but they are not necessary.

The BLSCDDIR command uses a VOLSER of VSAM01 to allocate a new dump directory if the VOL parameter is not provided here.

ErrorMsgPtr
Returned parameter

Type: Character string

Length:
Variable (Fullword)

The name of a required fullword area that will be set to point to a string terminated by a null (X'00'-terminated) character string containing one or more messages that describe certain types of errors that can occur. This string can be sent to a standard error device or file by the caller, and used to inform the end

user of the specific reasons for certain failures. If no messages are returned, the string is the null chaacter. Not all error cases return a message string.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the IPCSDumpOpenClose service returns the reason for the failure of the open request. This field is meaningful only when the open token returned in register 15 is 0. The value and meaning of Return_value1, Return_value2, and Return_value3 are dependent upon the nonzero value returned in the Return_code field. See "BPXYDCOR — dbx cordump cache information" on page 959 for detailed information about these fields.

Return_code	Explanation
Dcor_CDErc_OK	The specified function completed successfully.
Dcor_CDErc_ParmErr	A parameter error was detected. Return_value1 determines the specific reason for the failure: • Dcor_R1_ParmErr_FuncCodeErr — The function code is not supported. • Dcor_R1_ParmErr_DumpDsnReq — The dump data set name is required.
Dcor_CDErc_ProcErr	A DCORE processing error occurred. Return_value1 determines the specific reason for the failure: • Dcor_R1_ProcErr_SystemErrATC — An unexpected system error occurred while the IPCS environment was being established. Return_value2 contains the ABEND reason code.
Dcor_CDErc_IKJTSOEVErr	The system encountered an error while trying to establish a TSO environment with the IKJTSOEV service. See the return values for more information.
Dcor_CDErc_IKJEFTSRErr	The system encountered an error while trying to run the REXX exec with the IKJEFTSR service. See the return values for more information.
Dcor_CDErc_AllocateErr	The system encountered an error while trying to allocate one of the specified data sets. Return_value1 identifies the data set that caused the failure; Return_value2 contains the return code from dynamic allocation (DYNALLOC); and Return_value3 contains the reason code. • Dcor_R1_AllocateErr_LogDsn — There was an error allocating the log data set. • Dcor_R1_AllocateErr_ExecDsn — There was an error allocating the EXEC data set.

Return_value1, Return_value2, and Return_value3
Returned parameters

Type: Integer

Length:
Fullword

The names of fullwords in which the IPCSDumpOpenClose service returns details of the error indicated by Return_code. See the mapping of BPXYDCOR for detailed information about these fields.

Dcor_Close
Supplied parameter

Type: Integer

Length:
Fullword

The names of a fullword that contains the constant for a close request, DCOR_CLOSE. The value of this constant is defined in the BPXYDCOR macro see "BPXYDCOR — dbx cordump cache information" on page 959).

OpenToken
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the open token returned by the BPXGMCDE (BPXGMCD4) open request.

Usage notes

1. The routine to be executed receives control with the following attributes:
   - Problem program state
   - Key of the target pthread task
   - AMODE 31
   - Primary ASC mode
2. The register usage on entry to the specified routine is:
   - R0: Undefined
   - R1: Address of the parameter list, as specified by the caller of the IPCSDumpOpenClose service.
   - R2–R12: Undefined.
   - R13: Address of a 72–byte work area in the key that the routine gains control under.
   - R14: The return address from the specified routine to the IPCSDumpOpenClose service. This address must be preserved by the invoked routine.
   - R15: Address of the invoked routine.
3. Only tasks created with pthread_create or the IPT can invoke this service. If a task that is not an IPT or a pthread-created task requests this service, it receives an EACCES return code.
4. At any given time only one pthread can have this service request pending for a given target pthread. If a pthread requests this service for a given target pthread when another pthread already has this service pending for that target pthread, the last pthread receives an EAGAIN return code. It is the caller's responsibility to serialize the invocation of IPCSDumpOpenClose, or contain retry logic for cases in which the EAGAIN return code is received.
5. The EXITRTN assembler routine cannot issue callable services after it gains control under the target pthread.

6. The specified routine can establish its own recovery environment. However, even if recovery is not established, the IPCSDumpOpenClose service establishes its own recovery environment while running under the target pthread. For all recoverable errors, this recovery routine retries, returning the EFAULT return code to the requester. It also ensures that any recovery routine established by the target pthread itself is not entered unexpectedly.

Related services

- "IPCSDumpAccess (BPXGMPTR, BPXGMPT4) — PTRACE IPCS dump access service"

Characteristics and restrictions

There are no restrictions on the use of the IPCSDumpOpenClose service.

IPCSDumpAccess (BPXGMPTR, BPXGMPT4) — PTRACE IPCS dump access service

Function

The IPCSDumpAccess service reads storage, registers, program attributes, and other information related to a process or thread in a dump that has been opened with the IPCSDump Open/Close service (BPXGMCDE, BPXGMCD4).

Requirements

Operation	Environment
Authorization:	Problem state, user PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPXGMPTR):	31-bit
AMODE (BPXGMPT4):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPXGMPTR,(Dcor_Request,
               OpenToken,
               Parm1Address,
               Parm2Address,
               Parm3Address,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPXGMPT4 with the same parameters.

Parameters

Dcor_Request
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the integer value for the function requested. The functions are explained in the usage notes. The request integer values are defined in the BPXYDCOR macro. See "BPXYDCOR — dbx cordump cache information" on page 959.

OpenToken
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the open token returned by the BPXGMCDE (BPXGMCD4) Dcor_Open request.

Parm1Address
Supplied parameter

Type: Address or Integer

Length:
Variable

The name of a required value that contains the first parameter described by the function requested. See "BPXYDCOR — dbx cordump cache information" on page 959.

Parm2Address
Supplied parameter

Type: Address or Integer

Length:
Variable

The name of a required value that contains the second parameter described by the function requested. If a second parameter is not required, this value may be zero. See "BPXYDCOR — dbx cordump cache information" on page 959.

Parm3Address
Supplied parameter

Type: Address or Integer

Length:
Variable

The name of a required value that contains the third parameter described by the function requested. If a third parameter is not required, this value may be zero. See "BPXYDCOR — dbx cordump cache information" on page 959.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the IPCSDumpAccess service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The address of a fullword in which the IPCSDumpAccess service stores the return code. The IPCSDumpAccess service returns Return_code only when the Return_value is -1 and the Reason_code is DcorPTR_RsnDcorError. The IPCSDumpAccess service can return one of the following values in the Return_code parameter:

Return_code	Explanation
Dcor_PTRrc_OK	The specified function completed successfully.
Dcor_PTRrc_AsidNotFound	An address space could not be found in the dump to satisfy this request.
Dcor_PTRrc_AsidNotSet	An ASID or PID has not been established for this session.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the IPCSDumpAccess service stores the reason code. The IPCSDumpAccess service returns Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. The following reason codes can accompany the return code:

Reason_code	Explanation
RsnOkValue	The specified function completed successfully.
RsnDcorError	See Dcor return codes.
RsnMVSError	A getmain error probably occurred.
RsnIPCSError	Use the log data set to determine the cause of the error.

Usage notes

This table shows the constant options you can select for the Dcor_Request parameter. See "BPXYDCOR — dbx cordump cache information" on page 959.

Table 4. Dcor_Request options

Function request	Explanation
Dcor_ASID_LIST#	Return a list of ASIDs, and the number of ASIDs contained in the list. This list is described in BPXYDCOR as the AsidList_Map.
Dcor_SET_ASID#	Set the current address space ID to view in the dump. If a null parameter is provided, the home address space at the time the dump was taken will be returned. Changing the ASID may alter other values, such as the PID or the current thread.

Table 4. Dcor_Request options (continued)

Function request	Explanation
Dcor_PID_LIST#	Return a list of PIDs, and the number of PIDs contained the list. This list is described in BPXYDCOR as the PidList_Map.
Dcor_SET_PID#	Set the current process ID to view in the dump. If a null parameter is provided, the active process at the time the dump was taken will be returned. The process requested must exist in the current address space. Changing the PID may also cause the current thread to change.
Dcor_LDINFO#	Return the loader data from the current thread.
Dcor_THREAD_LIST#	Return the list of threads contained in the current PID, and the number of threads in the list. The Thread_list mapping is described in BPXYPTRC as PtPxInfo.
Dcor_THREAD_CURRENT#	Return the value of the current thread.
Dcor_SET_THREAD#	Set the current thread ID to view in the dump.
Dcor_PSW#	Return the 16-byte PSW for the current thread.
Dcor_ERROR_PSW#	Return the 16-byte PSW that caused the dump to be taken.
Dcor_GPR_LIST#	Return the 64-bit GPRs for the current thread.
Dcor_ERROR_GPR_LIST#	Return the 64-bit GPRs active at the time of error.
Dcor_FLT_LIST#	Return the 64-bit FLTs for the current thread.
Dcor_ERROR_FLT_LIST#	Return the 64-bit FLTs active at the time of error.
Dcor_THREAD_STATUS#	Return the Thread_list entry of the current thread. The Thread_list mapping is described in BPXYPTRC as PtPxInfo.
Dcor_READ_D#	Retrieve dump data and place it in a buffer provided by the caller.
Dcor_CAPTURE#	Return the address of a buffer containing the requested dump data.
Dcor_CONDINFO#	Return the current abend information at the time of error. CondInfo is described in BPXYDCOR by the CondInfo_Map.
Dcor_IPCSCMD#	Pass a command to IPCS. The output is stored in an MVS sequential data set named 'userid.BPXGCOR.IPCSPRNT'.

This table shows the PTRACE service options for the Dcor_Request parameter. For each option, the meanings of the Parm1, Parm2, and Parm3 parameters are shown. The terms in the table are described in “BPXYDCOR — dbx cordump cache information” on page 959.

Table 5. PTRACE service options for the Dcor_Request parameter

Function request	Parm1	Parm2	Parm3
Dcor_ASID_LIST#	The address of a fullword location to receive the list address	The address of a fullword location to receive the count of ASIDs	0
Dcor_SET_ASID#	The address of a 16-bit location that contains 0 or an ASID value	0	0
Dcor_PID_LIST#	The address of a fullword location to receive the list address.	The address of a fullword location to receive the count of PIDS.	0
Dcor_SET_PID#	The address of a fullword location that contains 0 or a PID value.	0	0
Dcor_LDINFO#	The address of a fullword location to receive the address of the loader information	0	0
Dcor_THREAD_LIST#	The address of a fullword location to receive the address of the GPR list	0	0
Dcor_THREAD_CURRENT#	The address of an 8-byte location that contains a null value or a thread ID	0	0
Dcor_SET_THREAD#	The address of an 8-byte location that contains a null value or a thread ID	0	0
Dcor_PSW#	The address of a 16-byte location to receive the PSW	0	0
Dcor_ERROR_PSW#	The address of a 16-byte location to receive the PSW	0	0
Dcor_GPR_LIST#	The address of a fullword location to receive the address of the GPR list	The address of a fullword location to receive the length of the GPR list	0
Dcor_ERROR_GPR_LIST#	The address of a fullword location to receive the address of the GPR list	The address of a fullword location to receive the length of the GPR list	0
Dcor_FLT_LIST#	The address of a fullword location to receive the address of the FLT list	The address of a fullword location to receive the length of the FLT list	0
Dcor_ERROR_FLT_LIST#	The address of a fullword location to receive the address of the FLT list	The address of a fullword location to receive the length of the FLT list	0

Table 5. PTRACE service options for the Dcor_Request parameter (continued)

Function request	Parm1	Parm2	Parm3
Dcor_THREAD_STATUS#	The address of a fullword location to receive the address of the thread list	The address of a fullword location to receive the length of the thread list	
Dcor_READ_D#	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) input value of virtual storage in the dump	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) input value of the number of bytes to return	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address that contains the address of the caller's input buffer
Dcor_CAPTURE#	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) value of virtual storage in the dump	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) value of the number of bytes to return	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the output buffer address
Dcor_CONDINFO#	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the address of the CondInfo data	The fullword (for AMODE 31 callers) or doubleword (for AMODE 64 callers) address to return the length of the CondInfo data	0
Dcor_IPCSCMD#	The address of the text of an IPCS command, or the word 'LOG'. LOG causes the diagnostic information that is active in the session to be written to an MVS sequential output data set.	A fullword input value, of the length of the IPCS command.	A fullword input value that contains the LRECL to use when allocating the sequential MVS data set `userid.BPXGCORE.IPCSPRNT`, which contains the output of the IPCS command.

Related services

- "IPCSDumpOpenClose (BPXGMCDE, BPXGMCD4) — MVS IPCS dump open/close service" on page 291

Characteristics and restrictions

There are no restrictions on the use of the IPCSDumpAccess service.

isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal

Function

The isatty callable service determines whether a file is a terminal.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ITY,(File_descriptor,
              Return_value)
```

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the file descriptor.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the isatty service returns 1 if File_descriptor is a terminal, or `0` if it is not a terminal.

Usage notes

This function does not return -1. If the file descriptor is not valid, a zero is returned. If this service fails for other reasons, the process abends.

Related services

- "ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal" on page 862
- "isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal" on page 303

Characteristics and restrictions

There are no restrictions on the use of the isatty service.

Examples

For an example using this callable service, see "BPX1ITY (isatty) example" on page 1155.

isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal

Function

The isatty callable service determines whether a file is a terminal.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX2ITY):	31-bit
AMODE (BPX4ITY):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX2ITY,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ITY with the same parameter.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the isatty service returns `1` if File_descriptor is a terminal, or `0` if it is not a terminal.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the isatty service stores the return code. The isatty service may return Return_code only if Return_value is 0. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The isatty service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The **filedes** argument is not a valid open file descriptor.
ENOTTY	The **filedes** argument is not associated with a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the isatty service stores the reason code. The isatty service may return Reason_code only if Return_value is 0. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This version of isatty is XPG4 compliant.
2. This function does not return -1. If the file descriptor is not valid, a zero is returned.

Related services

- "ttyname (BPX2TYN, BPX4TYN) (X/Open version) — Get the name of a terminal" on page 863
- "isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal" on page 301

Characteristics and restrictions

There are no restrictions on the use of the isatty service.

Examples

For an example using this callable service, see "BPX2ITY (isatty) example" on page 1155.

kill (BPX1KIL, BPX4KIL) — Send a signal to a process

Function

The kill callable service sends a signal to a process, a process group, or all processes in the system to which the caller has permission to send a signal.

CAUTION:

Note that when a caller with appropriate privileges (see "Authorization" on page 8) specifies a pid equal to -1, the signal will normally be sent to all processes in the system, excluding the init process (process ID 1). If the signal action is to terminate the process, all processes will terminate. This may not be the desired action, considering that some processes may be necessary for the continued operation of the system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1KIL):	31-bit
AMODE (BPX4KIL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1KIL,(Process_ID,
              Signal,
              Signal_Options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4KIL with the same parameters.

Parameters

Process_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword whose value specifies the process or processes to which the signal is to be sent:

- If Process_ID is greater than 0, it is assumed to be a process ID. The signal is sent to the process with that specific process ID.
- If Process_ID is equal to 0, the signal is sent to all processes with a process group ID equal to that of the caller, and for which the caller has permission to send a signal.
- If Process_ID is -1, the signal is sent to all processes for which the caller has permission to send the signal.
- If Process_ID is less than -1, its absolute value is assumed to be a process group ID. The signal is sent to all processes with a process group ID equal to that absolute value, and for which the caller has permission to send a signal.

For more information, see "Characteristics and restrictions" on page 308.

Signal
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the signal number to be sent to the processes that are indicated by the Process_ID parameter. The signal number must be defined in the BPXYSIGH macro, or 0.

If the signal is 0, error checking takes place, but no signal is sent. Use a signal value of 0 to verify that the Process_ID parameter is correct before actually sending a signal. This method does not verify permission to send the signal to the specified Process_ID.

Signal_Options
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the binary flags that describe how the signal is to be handled by z/OS UNIX and the user-supplied signal interface routine (SIR). This byte of user information is passed to the SIR in a data structure that is mapped by the BPXYPPSD macro. See "BPXYPPSD — Map signal delivery data" on page 1014. Signal_Options are mapped as follows:

First 2 bytes
User-defined bytes that are delivered with the signal to the SIR in the signal information control block. These bytes are mapped by PPSDKILDATA.

Last 2 bytes
Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - signal code specified in first 2 bytes, set by the application
- Fourth bit - reserved
- Fifth bit - enhanced SIGKILL deliverability (superkill)
- Sixth bit - Indicates whether to override the SIGTRACE action. If this bit is ON, then the seventh bit indicates the override action. If this bit is OFF, the SIGTRACE action is to toggle the current user syscall trace setting for the target processes.
- Seventh bit - If this bit is ON, the SIGTRACE signal turns user syscall tracing ON. If OFF, then SIGTRACE turns user syscall tracing OFF.
- Remaining bits - reserved.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the kill service returns `0` if it has permission to send the specified signal to any of the processes specified by the Process_ID parameter. A return value of `0` means that a signal was sent (or could have been sent, if the signal value was `0`) to at least one of the specified processes.

If no signal is sent, `-1` is returned.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the kill service stores the return code. The kill service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The kill service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of Signal is incorrect, or is not the number of a supported signal.
EMVSSAF2ERR	The caller does not have the appropriate RACF permissions.
EPERM	The caller does not have permission to send the signal to any process that was specified by the Process_ID parameter.
ESRCH	No processes or process groups that correspond to Process_ID were found.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the kill service stores the reason code. The kill service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, the reason code contains the security product return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the security product Check Privilege service return and reason code values, see the following table:

Security Product Return Code	Security Product Reason Code	Explanation
8	4	The caller is not the owner of the target process.
8	12	There was an internal error during security product processing.

Related services

- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253

- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746

Characteristics and restrictions

1. A caller can send a signal if the real or effective user ID of the caller is the same as the real or saved set user ID of the intended recipient. A caller can also send signals if it has appropriate privileges.

   Permissions are discussed in "Authorization" on page 8.
2. Regardless of user ID, a caller can always send a SIGCONT signal to a process that is a member of the same session as the sender.
3. A caller can also send a signal to itself. If the signal is not blocked, at least one pending unblocked signal is delivered to the sender before the service returns control. Provided that no other unblocked signals are pending, the signal delivered is the signal sent. See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313 for more information.
4. The thread-scoped signals (SIGTHSTOP and SIGTHCONT) cannot be issued by the kill callable service. They can be issued only by the pthread_kill service.
5. An enhanced SIGKILL (superkill) can be sent by setting the PPSDSUPERKILL bit on in the BPXYPPSD ("BPXYPPSD — Map signal delivery data" on page 1014). The superkill will break through most of the signal deterrents that can be an obstacle to the normal delivery of a SIGKILL and the resulting termination of the target process.

   Note:

   a. You cannot use pthread_kill() or sigqueue() to do a superkill. The superkill option is ignored for these services.
   b. You cannot do a superkill to a group, or specify a PID of -1 (kill everyone).
   c. When a target process has blocked all signals with the set_dub_default (BPX1SDD/BPX4SDD) service, superkills are deferred. The kill does not fail; it is simply ignored by the target process.
   d. Before a process can be superkilled, a regular SIGKILL must be sent to it, or the attempt will result in EINVAL/JRSigkillNotSent. This is analogous to the required "cancel" before a "force arm".

   If the environment is valid, the target process is abended with a X'422' abend and reason code X'0109'.

Examples

For an example using this callable service, see "BPX1KIL (kill) example" on page 1155.

MVS-related information

For signal information, see Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

__login, __login__applid, __certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product

Function

The BPX1SEC/BPX4SEC callable service provides an interface to the security product to allow the calling process to obtain security-related services.

No special authority is required to use this service to register or deregister a certificate that has the current identity of the calling process.

The C functions _login and _certificate result in a call to this service.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SEC):	31-bit
AMODE (BPX4SEC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1SEC,(Function_code,
               Identity_type,
               Identity_length,
               Identity,
               Pass_length,
               Pass,
               Certificate_length,
               Certificate,
               Option_flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SEC with the same parameters.

Parameters

Function_code

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value identifying the function that is to be performed. The following Function_code constants are defined by the BPXYCONS macro. See “BPXYCONS — Constants used by services” on page 952.

Constant	Description
SECURITY_CREATE#	Create the security environment for the caller's process.
SECURITY_CERTREG#	Register the passed certificate with the user ID that is associated with the current security environment.
SECURITY_CERTDEREG#	Deregister the passed certificate from the user ID that is associated with the current security environment.
SECURITY_CERTAUTH#	Authenticate the passed certificate for the caller. The certificate must have been registered.

Identity_type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that identifies the format of the Identity parameter and the Pass parameter. Constants are defined by the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Description
SECURITY_USERID#	The user identity is in the format of a 1-to 8-character user ID that is passed as input.

Identity_length
Supplied parameter for SECURITY_USERID#

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Identity parameter. The specified length must be consistent with the allowable Identity types: for SECURITY_USERID#, the length is 1-to-8 characters.

Identity
Supplied parameter for SECURITY_USERID#

Type: Character string

Character set:
For SECURITY_USERID#, the identity is a USERID that follows the XPG4 naming convention portable character set. This includes upper and lower-case letters (A-Z, a-z), numerics (0–9), period (.), dash (-), and underbar (_).

Length:
Specified by the Identity_length parameter

The name of a field that contains the user identity in the specified format.

Pass_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored.

Pass
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pass_length parameter

The name of a field, of length Pass_length, that contains, left-justified, the password, PassTicket or password phrase that is to be verified.

Certificate_length
Supplied parameter

Type: Integer

Length:
Fullword

For SECURITY_CERTREG#, SECURITY_CERTDEREG#, and SECURITY_CERTAUTH#, the name of a fullword that contains the length of a certificate structure as defined by the Certificate parameter. This parameter is ignored for all other function codes.

Certificate
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Variable

For SECURITY_CERTREG#, SECURITY_CERTDEREG#, and SECURITY_CERTAUTH#, the name of an area that consists of a digital certificate. See the information on the initACEE callable service in *z/OS Security Server RACF Callable Services* for a description of the formats for a digital certificate. This parameter is ignored for all other function codes.

Option_flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword binary field that contains the BPX1SEC/BPX4SEC options. If no options are required, specify the name of a fullword field that contains `0`. No options are currently defined.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPX1SEC/BPX4SEC service returns `0` if the request is successful, or -1 if it is not successful. For SECURITY_CERTAUTH#, this field returns an address to read-only storage that contains the 8-character user ID. If the request is not successful, the service returns -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPX1SEC/BPX4SEC service stores the return code. The BPX1SEC/BPX4SEC service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The BPX1SEC/BPX4SEC service can return one of the following values in the Return_code parameter:

Return_Code	Explanation
EINVAL	A parameter is not valid, or a certificate was not specified, or no security product is installed. The following reason codes can accompany the return code: JrFunctionCode, JrIdentityType, JrBadOptions, JrUserNameLenError, JrPasswordLenError, JrNewPasswordLenError, JrCertificate, JrNoSecurityProduct.
EPERM	The operation is not permitted, or the calling task has a task level ACEE that was not created by a prior call to this service. The following reason codes can accompany the return code: JrNotServerAuthorized, JrSecurityEnv, JrEnvDirty, JrMultiThreaded, JrUnexpectedError.
ESRCH	The USERID cannot become an OMVS process. The following reason codes can accompany the return code: JrOK, JrNoCertforUser.
EMVSSAF2ERR	An error occurred in the security product, or there was a parameter list error on a call to initACEE. The following reason codes can accompany the return code: JrCertInvalid, JrSafInternal, JrSafGroupNoMVS, JrSafNoGid, JrSafNoUid, JrSafUserNoMVS, JrSafParmListErr, JrCertInvalid, JrCertDoesNotMeetReq, JrCertAlreadyDefined, JrUnexpectedError, JrOK, X' '0814' '.
ENOSYS	The function is not implemented. The following reason codes can accompany the return code: JrNoSecurityProduct, JrNoInitACEE.
EACCES	Permission is denied. The following reason codes can accompany the return code: JrOK, JrNoResourceAccess.
EMVSEXPIRE	The password for the resource that was specified has expired. The following reason code can accompany the return code: JrOK.
EMVSPASSWORD	The new password that was specified is not valid. The following reason code can accompany the return code: JrOK.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which this service stores the reason code. The BPX1SEC/BPX4SEC service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

The following table shows the Return_code and Reason_code values that are returned when the BPX1SEC/BPX4SEC service is called to register or deregister a certificate and initACEE has a return code of 8.

Table 6. BPX1SEC/BPX4SEC return values for certificate registration/deregistration with initACEE return code 8

initACEE reason code	BPX1SEC/BPX4SEC return code	BPX1SEC/BPX4SEC reason code	Explanation
4	EMVSSAF2ERR	JrSafParmListErr	There was a parameter list error.
8	EMVSSAF2ERR	JrSafInternal	There was an internal RACF error.
12	EMVSSAF2ERR	JrSafInternal	RACF recovery environment could not be established.
16	EACCES	JrNoResourceAccess	The user is not authorized.
20	EMVSSAF2ERR	JrCertDoesNotMeetReq	The certificate does not meet RACF requirements.
24	EMVSSAF2ERR	JrCertAlreadyDefined	The certificate is already defined for another user.
36	EMVSSAF2ERR	JrCertInvalid	The certificate is not valid.

Usage notes

1. The following table shows the BPX1SEC/BPX4SEC parameters that are used with each function.

Table 7. BPX1SEC/BPX4SEC parameter usage based on function requested

Parameter	Login as a new user	Register a certificate	Deregister a certificate	Authenticate a certificate
Function_Code	_CREATE#	_CERTREG#	_CERTDEREG#	_CERTAUTH#
Identity_Type	SECURITY_USERID#	Not applicable	Not applicable	Not applicable
Identity_Length	Input	Not applicable	Not applicable	Not applicable
Pass_Length	Input (optional)	Not applicable	Not applicable	Not applicable
Pass	Input (optional)	Not applicable	Not applicable	Not applicable
Cert_Length	Not applicable	Input	Input	Input
Certificate	Not applicable	Input	Input	Input
Option_Byte	Not applicable	Not applicable	Not applicable	Not applicable
Return_value	Output	Output	Output	Output (address)
Return_code	Output	Output	Output	Output
Reason_code	Output	Output	Output	Output

For the SECURITY_CERTREG# and SECURITY_CERTDEREG# functions, the certificate is passed in the Certificate parameter, and not the Identity parameter. The certificate does not necessarily define the identify of the caller; these functions could be called with a user ID and password.

For the SECURITY_CERTAUTH# function, the certificate is passed in the Certificate parameter. The certificate contains the identity of the caller, and can be used instead of a user ID/password combination.

2. When this service is called for function code SECURITY_CERTAUTH#:
   - A certificate is passed for authentication. It is possible that the USERID associated with the certificate was valid at the time the certificate was created, but is no longer valid when this call is made. For example, the USERID could have been revoked or its profile could have been changed so that it no longer has any USS information associated with it. However, the original ACEE is still valid and can be cached. Because of this there are different return/reason codes issued for revoked USERIDs and USERIDs with no OMVS segment, depending on the ACEE that is used during this call. That is:

     If the USERID has no OMVS segment, when using a cached ACEE you will receive ESRCH and JrOK. When there is no cached ACEE, you will receive EMVSSAF2ERR/JrSafUserNoMVS

     If the USERID has been revoked, when using a cached ACEE you will receive ESRCH/JrOK. When there is no cached ACEE you will receive EMVSSAF2ERR/JrOK.

     If the USERID has an OMVS segment, but no UID defined, when using a cached ACEE you will receive EMVSSAF2ERR/X' '0814' '. When there is no cached ACEE you will receive EMVSSAF2ERR/JrSafNoUid.
3. Mixed case passwords and PassTickets are supported when the installed security product (such as RACF) supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

   The contents of the password phrase string are passed unchanged to the installed security product.

   Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation or the installed security product can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.
4. The BPX1SEC/BPX4SEC service allows a process to assume an identity that is different from that of the address space. It is assumed that the process will either terminate or select a new user ID, but not try to revert back to the original address space identity. The user could issue the BPX1SEC/BPX4SEC request again with the original user identity; however, at this point the user has its own security environment, at the task level, rather than the address space level.
5. For SECURITY_CREATE# and SECURITY_CERTAUTH#, if BPX.DAEMON is defined, then the address space must be program-controlled.
6. When calling the BPX1SEC/BPX4SEC service with function code SECURITY_CREATE#, the caller can change identities under any of the following conditions:
   - The caller specifies the password for the requested identity.
   - If no password is specified and the BPX.DAEMON profile is not defined in the FACILITY class, the caller must be a superuser. If no password is specified and the caller has read access to the BPX.SRV.*userid* SURROGAT profile, where *userid* is the user ID specified in the User_name parameter.
   - If no password is specified and the BPX.DAEMON profile is defined in the FACILITY class, the caller must be permitted to that profile with at least READ access and must be a superuser.
7. Only a single-threaded process can call the BPX1SEC/BPX4SEC service with function code SECURITY_CREATE#.

8. The purpose of the BPX1SEC/BPX4SEC register/deregister service is to provide a way for the caller to associate or disassociate its user ID with a certificate. No new security environment is created, and no authentication of the user is done.
9. The ability to call the BPX1SEC/BPX4SEC service to register or deregister a certificate with a user ID is not a privileged operation. The user does not need any special authority above that required by RACF to register or deregister certificates. The caller does not, for example, have to be a DAEMON or a SUPERUSER. RACF requires that the caller have access to the RACDCERT FACILITY class definitions (IRR.DIGTCERT.ADD and IRR.DIGTCERT.DELETE) for registration and deregistration.
10. The BPX1SEC/BPX4SEC authenticate service provides the caller with a way to authenticate a security environment using a certificate. The certificate must already be registered. If the certificate is not registered, an error is returned.
11. The __login__applid() function is equivalent to __login() with the added feature that __login__applid() allows an application identifier (applid) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the applid is also used in conjunction with the USERID to verify the PassTicket. If an application is not using the __login__applid() function but still wants to pass an applid to this service, the application can set the applid value in the BPXYTHLI. Also:
    - THLIEP_FunctionCode is set with ThliEP_ApplSet.
    - THLIEP_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, the ThliEP_APPLID value is ignored.
    - ThliEP_APPLID is set to the APPLID value.

    If there is no applid value passed, the applid value defaults to OMVSAPPL.
12. If environment variable BPXK_MIN_PWFOLD=YES is set then non-graphic characters will not be changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set

Related services

None.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see “BPX1SEC (__login, __login__applid, __certificate) example” on page 1185.

lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link

Function

The lchattr service modifies the attributes that are associated with a file. It is similar to the chattr service, and can be used to modify certain attributes associated with a symbolic or external link, as well as for regular files and directories.

If the pathname is a symbolic link, the requested change occurs on the attributes of the symbolic link, and only those attributes that can apply to a symbolic link are updated. These are limited to the owner, the time values, and the security label. All other requested attribute changes have no effect for the symbolic link. See "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LCR):	31-bit
AMODE (BPX4LCR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

AMODE 64 callers use BPX4LCR with the same parameters.

Format

```
CALL BPX1LCR,(Pathname_length,
              Pathname,
              Attributes_length,
              Attributes,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file whose attributes you want to change.

Pathname
Supplied parameter

Type: Character string

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file. The length of this field is specified in Pathname_length.

If Pathname specifies a symbolic link file, the lchattr service changes the attributes of the symbolic link file itself, provided that the attributes requested

can apply to a symbolic link. Only the owner, times, and security label can be changed for a symbolic link. All other attributes do not apply and will be ignored.

Attributes_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area containing the attributes you want to change.

Attributes
Supplied parameter

Type: Structure

Length:
Specified by the Attributes_length parameter

The name of the area that contains the attributes you want to change. The area is mapped by BPXYATT. For information on the content of this area, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948.

If Pathname specifies a symbolic link file, the lchattr service changes the attributes of the symbolic link file itself, provided that the attributes requested can apply to a symbolic link. Only the owner, times, and security label can be changed for a symbolic link. All other attributes do not apply and will be ignored.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchattr service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchattr service stores the return code. The lchattr service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The lchattr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process did not have appropriate permissions. Possible reasons include: • The calling process was attempting to set access time or modification time to current time, and the effective UID of the calling process does not match the owner of the file; the process does not have write permission for the file; or the process does not have appropriate privileges(see "Authorization" on page 8). • The calling process was attempting to truncate the file, and it does not have write permission for the file.
EFBIG	The calling process was attempting to change the size of a file, but the specified length is greater than the maximum file size limit for the process. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRWriteBeyondLimit.
EINVAL	The length of the Attributes parameter is too small, or the Attributes structure containing the requested changes is not valid. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JrInvalidAtt, JrNegativeValueInvalid, JrTrNotRegFile, JrTrNegOffset, JrFileNotEmpty, and JrInvalidFileTag.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMVSERR	An MVS environmental error has been detected. The following reason code can accompany the return code: JrSeclabelClassInactive.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters. Filename truncation is not supported.
ENOENT	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOSYS	The function is not supported for the specified file. The following reason code can accompany the return code: JrNotSupportedForFileType.
ENOTDIR	Some component of Pathname is not a directory.

Return_code	Explanation
EPERM	The operation is not permitted for one of the following reasons: • The calling process was attempting to change the mode or the file format, but the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges (see "Authorization" on page 8). • The calling process was attempting to change the owner, but it does not have appropriate privileges. • The calling process was attempting to change the general attribute bits, but it does not have write permission for the file. • The calling process was attempting to set a time value (not current time), but the effective user ID does not match the owner of the file, and it does not have appropriate privileges. • The calling process was attempting to set the change time or reference time to current time, but it does not have write permission for the file. • The calling process was attempting to change auditing flags, but the effective UID of the calling process does not match the owner of the file, and the calling process does not have appropriate privileges. • The calling process was attempting to change the security auditor's auditing flags, but the user does not have auditor authority. • The calling process was attempting to set the security label, but one or more of the following conditions apply: – The calling process does not have RACF SPECIAL authorization and appropriate privileges. – There is already a security label associated with the file.
EROFS	Pathname specifies a file that is on a read-only file system. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchattr service stores the reason code. The lchattr service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

Table 8. Attribute fields modifiable by lchattr

Set flags	Attribute fields input	Description
ATTMODECHG	ATTMODE	Set the mode according to the value in ATTMODE. See "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90.

Table 8. Attribute fields modifiable by lchattr (continued)

Set flags	Attribute fields input	Description
ATTOWNERCHG	ATTUID ATTGID	Set the owner user identifier (UID) and group identifier (GID) to the values specified in ATTUID and ATTGID. See "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93.
ATTSETGEN	ATTGENVALUE ATTGENMASK	Only the bits corresponding to the bits set ON in the ATTGENMASK are set to the value (ON or OFF) in ATTGENVALUE. Other bits are unchanged.
ATTTRUNC	ATTSIZE	Change the file size to ATTSIZE bytes. See "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203.
ATTATIMECHG	ATTATIME	If ATTLP64TIMES is not set, set the access time of the file to the value specified in ATTATIME. If ATTLP64TIMES is set, set the access time of the file to the value specified in ATTATIME64, which is a doubleword field.
ATTATIMETOD	None	Set the access time of the file to the current time.
ATTMTIMECHG	ATTMTIME	If ATTLP64TIMES is not set, set the modification time of the file to the value specified in ATTMTIME. If ATTLP64TIMES is set, set the modification time of the file to the value specified in ATTMTIME64, which is a doubleword field.
ATTMTIMETOD	None	Set the modification time of the file to the current time.
ATTMAAUDIT	ATTAUDITORAUDIT	Set the security auditor's auditing flags to the value specified in ATTAUDITORAUDIT. See "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84.
ATTMUAUDIT	ATTUSERAUDIT	Set the user's auditing flags to the value specified in ATTUSERAUDIT. See "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84.
ATTCTIMECHG	ATTCTIME	If ATTLP64TIMES is not set, set the change time of the file to the value specified in ATTCTIME. If ATTLP64TIMES is set, set the change time of the file to the value specified in ATTCTIME64, which is a doubleword field.

Table 8. Attribute fields modifiable by lchattr (continued)

Set flags	Attribute fields input	Description
ATTCTIMETOD	None	Set the change time of the file to the current time.
ATTREFTIMECHG	ATTREFTIME	If ATTLP64TIMES is not set, set the reference time of the file to the value specified in ATTREFTIME. If ATTLP64TIMES is set, set the reference time of the file to the value specified in ATTREFTIME64, which is a doubleword field.
ATTREFTIMETOD	None	Set the reference time of the file to the current time.
ATTFILEFMTCHG	ATTFILEFMT	Set the file format of the file to the value specified in ATTFILEFMT.
ATTCHARSETIDCHG	ATTFILETAG	Set the file tag. See BPXYSTAT ("BPXYSTAT — Map the response structure for stat" on page 1057) for file tag mapping.
ATTSECLABELCHG	ATTSECLABEL	Set the initial security label for a file or directory.

1. Flags in the Attributes parameter are set to indicate which attributes are to be updated. To set an attribute, turn the corresponding **Set Flag** on, and set the corresponding **Attributes Field** according to Table 2 on page 80. Multiple attributes may be changed at the same time.

   The **Set Flag** field should be cleared before any bits are turned on. It is considered an error if any of the reserved bits in the flag field are turned on.
2. Some of the attributes that are changed by the lchattr service can also be changed by other services. See the related service (listed in Table 8 on page 319) for a detailed description.
3. Changing mode (ATTMODECHG = ON):
   - The file mode field in the Attributes parameter is mapped by the BPXYMODE macro (see "BPXYMODE — Map the mode constants of the file services" on page 996). For information on the values for file type, see "BPXYFTYP — File type definitions" on page 967.
   - File descriptors that are open when the lchattr service is called retain the access permission they had when the file was opened.
   - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
   - Setting the set-group-ID-on-execution permission (in mode) means that when this file is run through the exec, attach_exec, or spawn service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

     The set-group-ID-on-execution permission is set to zero if both of the following are true:
     - The caller does not have appropriate privileges.
     - The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.

- Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

4. Changing owner (ATTOWNERCHG = ON):
   - To change the owner UID of a file, the caller must have appropriate privileges.
   - To change the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:
     - The effective UID of the caller matches the file's owner UID.
     - The Owner_UID value that is specified in the change request matches the file's owner UID.
     - The Group_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
   - When the owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
   - When the owner is changed, both UID and GID must be specified as they are to be set, or set to -1 if the value is to remain unchanged. If only one of these values is to be changed, the other can be set to its present value or to -1 to remain unchanged.
5. Changing general attribute bits (ATTSETGEN = ON):
   - Changing the general attributes of a file, directory, symbolic link, or external link is not supported with BPX1LCR. Refer to “chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory” on page 76 for information on setting the general attributes of a file or directory.
6. Changing the file size (ATTTRUNC = ON):
   - The resizing of a file to ATTSIZE bytes changes the file size to ATTSIZE, beginning from the first byte of the file. If the file was originally larger than ATTSIZE bytes, the data from ATTSIZE to the original end of file is removed. If the file was originally shorter than ATTSIZE, bytes between the old and new lengths are read as zeros.

     Full blocks are returned to the file system so that they can be used again.

     The file offset is not changed.
   - When a file size is changed successfully, it clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file, unless the caller has appropriate privileges.
   - The resizing of a file to ATTSIZE bytes, where ATTSIZE is greater than the soft file size limit for the process, fails with EFBIG, and the SIGXFSZ signal is generated for the process.
7. Changing times:
   - All time fields in Attributes are in POSIX format.
   - For the access time or the modification time to be set explicitly (ATTATIMECHG = ON or ATTMTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
   - For the access time or modification time to be set to the current time (ATTATIMETOD = ON or ATTMTIMETOD = ON), the effective ID must match the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.

- For the change time or the reference time to be set explicitly (ATTCTIMECHG = ON or ATTREFTIMECHG = ON), the effective ID must match the file's owner, or the process must have appropriate privileges.
- For the change time or reference time to be set to the current time (ATTCTIMETOD = ON or ATTREFTIMETOD = ON), the calling process must have write permission for the file.
- For any time field (atime, mtime, ctime, reftime), if both current time and specific time are requested (for example, ATTCTIMETOD = ON and ATTCTIMECHG = ON), the current time is set.
- When any attribute field is changed successfully, the file's change time is also updated.

8. Changing auditor audit flags (ATTMAAUDIT = ON):
   - For auditor audit flags to be changed, the user must have auditor authority. Users with auditor authority can set the auditor options for any file, even those for which they do not have path access or authority to use for other purposes.

     You establish auditor authority by issuing the TSO/E command ALTUSER Auditor.
9. Changing user audit flags (ATTMUAUDIT = ON):
   - For the user audit flags to be changed, the user must have appropriate privileges (see "Authorization" on page 8) or be the owner of the file.
10. Changing file format (ATTFILEFMTCHG = ON):
    - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
11. Changing the file tag (ATTCHARSETIDCHG=ON):
    - A file tag can be set for regular, FIFO, and character special files. If the DeferTag bit is on in the file tag, the file must be empty.
12. Changing the security label (ATTSECLABELCHG=ON):
    - For the security label to be changed, the user must have RACF SPECIAL authorization and appropriate privileges (see "Authorization" on page 8), and no security label must currently exist on the file. Only an initial security label can be set. An existing security label cannot be changed. The function will successfully set the security label if the RACF SECLABEL class is active. If the SECLABEL class is not active, a return code of EMVSERR will be returned.

Related services

- "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76

Characteristics and restrictions

1. The ATTGENVALUE field of BPXYATT cannot be modified with BPX1LCR.
2. The General Attribute fields (set by ATTSETGEN, ATTGENMASK, and ATTGENVALUE fields) are not intended as a general-use programming interface to BPX1LCR.
3. The security label (ATTSECLABELCHG) flag requires RACF SPECIAL authorization and appropriate privileges (see "Authorization" on page 8). It cannot be used to change an existing security label; it can only be used to set an initial security label on a file.
4. When Pathname refers to a symbolic link, any attributes that are requested for change other than owner, times, and security label will be ignored.

Examples

For an example using this callable service, see "BPX1LCR (lchattr) example" on page 1156.

lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link

Function

The lchown service changes the owner or group (or both) of a file or a directory. The owner is identified by a user ID (UID) and a group ID (GID).

The lchown service is identical to the chown service, except when the Pathname specified is a symbolic link (a pointer to another file or directory). If the Pathname is a symbolic link, the UID and/or the GID of the symbolic link are updated, rather than the UID or GID of the file to which the symbolic link refers. See "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LCO):	31-bit
AMODE (BPX4LCO):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LCO,(Pathname_length,
              Pathname,
              Owner_UID,
              Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LCO with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the path name of the file for which the owner or group is to be changed.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the path name of the file. The length of this field is specified in Pathname_length.

Pathnames can begin with or without a slash.

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name. The search for the file starts at the working directory.

If the path name specifies a symbolic link file, the lchown service changes the ownership of the symbolic link file itself.

Owner_UID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new owner UID that is assigned to the file; or the present value or -1, if there is no change. This parameter must be specified.

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the new owner GID that is assigned to the file; or the present value or -1, if there is no change. This parameter must be specified.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchown service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchown service stores the return code. The lchown service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The lchown service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search some component of the Pathname prefix.
EINVAL	The Owner_UID or Group_ID parameter is incorrect.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or a component of the path name is longer than 255 characters.
ENOENT	No file named Pathname was found; or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of the Pathname prefix is not a directory.
EPERM	The calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lchown service stores the reason code. The lchown service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The lchown service changes the owner UID and owner GID of a file. Only a caller with appropriate privileges (see "Authorization" on page 8) can change the owner UID of a file.
2. The owner GID of a file can be changed by a caller if the caller has appropriate privileges, or if the caller meets all of these conditions:
   - The effective UID of the caller matches the file's owner UID.
   - The Owner_UID value that is specified in the change request matches the file's owner UID.
   - The Group_ID value that is specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both Owner_UID and Group_ID must be specified. To change only one of these values, set the other to its present value or to -1.

Related services

- "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196
- "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the lchown service.

Examples

For an example using this callable service, see "BPX1LCO (lchown) example" on page 1156.

link (BPX1LNK, BPX4LNK) — Create a link to a file

Function

The link callable service creates a link to a file. The link is a new name that identifies an existing file. The new name does not replace the old one; it provides an additional way to refer to the file. To rename an existing file, see "rename (BPX1REN, BPX4REN) — Rename a file or directory" on page 607.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LNK):	31-bit
AMODE (BPX4LNK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LNK,(Filename_length,
              Filename,
              Link_name_length,
              Link_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LNK with the same parameters.

Parameters

Filename_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Filename of the existing file.

Filename
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Filename_length parameter

The name of a field of length Filename_length that contains the name of the existing file to which a link is to be established.

Link_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Link_name.

Link_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Link_name_length parameter

The name of a field that contains the link name by which the file is to be known.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the link service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the link service stores the return code. The link service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The link service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process did not have appropriate permissions to create the link. Possible reasons include: • No search permission for a pathname component of Filename or Link_name • No write permission for the directory intended to contain the link • No permission to access Filename
EEXIST	A file, directory, or symbolic link named Link_name already exists. The following reason code can accompany the return code: JRLnkNewPathExists.
EINVAL	The Filename or Link_name is incorrect because it contains a null.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Filename or Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Filename or Link_name.
EMLINK	Filename already has its maximum number of links. The maximum number is LINK_MAX. The value of LINK_MAX can be determined through "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464, or "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191.
ENAMETOOLONG	Filename or Link_name is longer than 1023 characters; or some component of the pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	A component of the pathname that was specified by Filename or Link_name was not found; the file specified by Filename was not found; or one of the two arguments is missing. The following reason code can accompany the return code: JRLnkNoEnt.
ENOSPC	The directory intended to contain the link cannot be extended to contain another entry.
ENOTDIR	A pathname component of one of the arguments is not a directory.
EPERM	Filename is the name of a directory; links to directories are not allowed. The following reason code can accompany the return code: JRLnkDir.
EROFS	Creating the link would require writing on a read_only file system. The following reason code can accompany the return code: JRLnkROFileset.
EXDEV	Filename and Link_name are on different file systems. z/OS UNIX does not support links between file systems. The following reason code can accompany the return code: JRLnkAcrossFilesets.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the link service stores the reason code. The link service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The link service creates a link named Link_name to an existing file named Filename. This provides an alternate pathname for the existing file; the file can be accessed by the old name or the new name. The link can be stored in the same directory as the original file, or in a different directory.
2. If the link is created successfully, the service increments the link count of the file. The link count shows how many links exist for a file. (If the link is not created successfully, the link count is not incremented.)
3. Links are allowed only to files, not to directories.
4. If the link is created successfully, the change time of the linked-to file is updated. The change and modification times of the directory that holds the link are also updated.

Related services

- "rename (BPX1REN, BPX4REN) — Rename a file or directory" on page 607
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the link service.

Examples

For an example using this callable service, see "BPX1LNK (link) example" on page 1159.

listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients

Function

The listen callable service creates a connection request queue for a server socket to queue incoming connection requests from a client.

Listen is used for connection-oriented sockets only. If a connection request arrives with the backlog queue full, the client may receive an ECONNREFUSED.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1LSN):	31-bit task mode or SRB mode
AMODE (BPX4LSN):	64-bit task mode only

Operation	Environment
ASC mode:	AR mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LSN,(Socket_descriptor,
              Backlog,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LSN with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the listen is to be done.

Backlog
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the maximum length of the connection queue. For network sockets, if Backlog is greater than SOMAXCONN, this field is set to SOMAXCONN. For AF_UNIX sockets, there is no maximum value for this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the listen service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the listen service stores the return code. The listen service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The listen service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINVAL	An incorrect argument was supplied. The socket is not named (a bind has not been done); or the socket is ready to accept connections (a listen has already been done). The following reason code can accompany the return code: JRListenNotAccepted.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EOPNOTSUPP	The socket descriptor specified a *datagram* socket. The listen service is valid only for *stream* sockets. The following reason code can accompany the return code: JRListenNotStream.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the listen service stores the reason code. The listen service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If a bind is not called before the listen request, the listen callable service returns an EINVAL.
2. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Characteristics and restrictions

There are no restrictions on the use of the listen service.

Examples

For an example using this callable service, see "BPX1LSN (listen) example" on page 1159.

loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name

Function

The loadhfs service loads an executable program by path name into the caller's process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LOD):	31-bit
AMODE (BPX4LOD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LOD,(Filename_length,
              Filename,
              Flags,
              Libpath_length,
              Libpath,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers need an additional parameter, Entry_point:

```
CALL BPX4LOD,(Filename_length,
              Filename,
              Flags,
              Libpath_length,
              Libpath,
              Entry_point,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

Filename_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Filename parameter. The length can be a value in the range 1 to 1023.

Filename
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Filename_length parameter

The name of a field that contains the file name of the program that is to be loaded. If the Filename parameter does not contain a slash (/), it is treated as a base name; it should be in one of the directories listed in the supplied Libpath parameter. If the Libpath parameter is null, the file must be in the current directory. If the file name is not a base name (that is, it contains at least one slash), the name is used as is; the Libpath parameter is not used to locate the file.

If the file name is a base name, it can be up to 255 characters long.

If the Filename parameter represents a path name, each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The Flags parameter is a fullword field that contains option flags that the loadhfs service uses in determining the optional processing to be performed on behalf of the caller. These constants are defined in the BPXYCONS macro.

Constant	Description
Lod_Error_St_ExLink	Indicates that LOAD processing is to be bypassed if the file is an external link or has the sticky bit set on. If the file is sticky or is an external link, the request fails with return code EPERM (the operation is not permitted) and a reason code of JrExternalLink or JrStickyBit.
Lod_Ignore_Sticky	Indicates that the sticky bit for a file is to be ignored. If the file is sticky, it is loaded from the z/OS UNIX file system.

Note: If both Lod_Ignore_Sticky and Lod_Error_St_ExLink are specified, the Lod_Ignore_Sticky option is honored, and Lod_Error_St_ExLink is ignored.

Libpath_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the library path parameter. If a value of zero is specified, the library path parameter is ignored.

Libpath
Supplied parameter

Type: Structure

Length:
Specified by the Libpath_length parameter

The name of a field that contains the library path to be searched to determine the fully qualified path name of the file that is specified. The library path can contain a series of path names separated by colons. The path names in the list are searched one at a time until the specified file name is located. If the list of path names begins or ends with a colon, the working directory of the calling process is used to locate the file. Each path name in the list can have a maximum length of 1021 bytes.

The following is an example of a valid library path:

- **/usr1/bin:/grp1/bin:/bin**

Entry_point
Returned parameter (BPX4LOD only)

Type: Structure

Length:
Doubleword

The name of a field that contains the entry point.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the loadhfs service returns -1 if it is not successful. If it is successful, the loadhfs service returns the entry point address of the program that was loaded into storage. If the loaded program is an AMODE 31 program, the high-order bit of the return value is turned on. For this reason, applications that test for a failure condition must explicitly check for a -1 return value. Checking for a value of less than zero will not produce the desired results.

For AMODE 64 programs, if the return value is `0`, the entry point address of the loaded program is returned in the Entry_point parameter.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the loadhfs service stores the return code. The loadhfs service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The loadhfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory named in the Pathname parameter; it may lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type.
EAGAIN	The file changed during load processing (JrFileChangeDuringLoad).
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Filename parameter. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
EMVSERR	An error occurred while loading a z/OS UNIX program (JrMVSLoadFailure or JrMVSPgmNotFound). Or an error occurred checking the caller's environment against the authorization of the file (JrNoListAuthPgmPath, JrNoListPgmCntlPath, JrProgCntl, JrAuthCaller).
ENAMETOOLONG	Filename is longer than 1023 characters; or some component of the file name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file name was specified, or one or more of the components of the specified Filename parameter were not found.
ENOEXEC	The specified file has execute permission, but it is not in the proper format to be a process image file.
ENOMEM	The file that is to be loaded requires more memory than is permitted by the hardware or the operating system.
ENOTDIR	A directory component of Filename is not a directory.
EINVAL	An invalid parameter value was specified. The invalid parameter might be one of the following: Filename_length.
EPERM	The operation is not permitted. The Flags parameter was set to Lod_Error_St_ExLink, and either the file is an external link (JrExternalLink), or it has the sticky bit set on (JrStickyBit).

Note: In addition to the return codes listed here, the loadhfs service can return additional errors for other failures that can occur on a stat or an open syscall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the loadhfs service stores the reason code. The loadhfs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS LOAD service, with the following exceptions:
   - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
   - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the HFS program found is in storage that is modifiable by the caller, the prior copy is not reused.

2. If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is only used if the name is eight characters or less, otherwise the caller receives an error from the loadhfs service. For a sticky bit program, the file name is used if it is eight characters or less. If the file name is greater than eight characters, or the MVS program is not found, the program is loaded from the z/OS UNIX file system.
3. When it is running from a pthread_created thread (pthread), the specified file is loaded into storage and associated with the Initial Pthread Creating Task (IPT). This allows the program to be shared across multiple threads, without the problem of its disappearing unexpectedly when a thread terminates.
4. When the calling process is being debugged via the ptrace service, the following applies:
   - Programs that are loaded using this service are loaded into storage that is modifiable by the caller of the loadhfs service.
   - A call to this service generates a WastStopFlagLoad Ptrace event to the debugger process.
5. Because this service does not cause the specified program to be executed, the set-user-ID and set-group-ID flags have no impact on the process.
6. Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:
   - Loading a program from the z/OS UNIX file system causes the program environment to become uncontrolled unless the executable file has the program control attribute turned on (ST_PROGCTL). Having the program control attribute on prevents future invocations of authorized programs like PADS programs. In addition, PADS programs should not attempt to load programs from the z/OS UNIX file system; the z/OS UNIX file system is considered an unauthorized library and can potentially be modified by users that do not have the same level of authorization as the PADS program.
   - System key, supervisor state and APF-authorized callers should not attempt to load a program from the z/OS UNIX file system, unless the executable file has the APF attribute turned on.
7. If a program that is loaded into storage with this service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates.
8. The AUTHPGMLIST environment variable works with this system call. The environment variable specifies a list of sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being loaded is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For details about the sanction list, see the topic on using sanction lists in *z/OS UNIX System Services Planning*.

   The following usage notes apply for shared library programs:
9. Executables that have the ST_SHARELIB extended attribute turned on are considered system shared library programs. System shared library programs are the most optimal way to share large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage; it does not interfere with the virtual storage used by the application program.

10. If the program to be loaded is determined to be a shared library program (that is, if the ST_SHARELIB extended attribute is on), the loadhfs service queries the shared library region to determine if the target program is there.

    When a shared library program is loaded anew into the shared region or reloaded from the shared region, the program is mapped from the shared region into the private area of the calling address space. It is important to note that, because the program is not actually reloaded from DASD into the private area of each calling address space, but only remapped from the shared region, shared library programs are more efficient in their utilization of system resources than normal private area programs. For this reason, programs that are to be shared across several address spaces in the system are good candidates for identification as shared library programs.

    If a target program is not in the shared library region and cannot be loaded into the region because of its attributes, the program is treated like a private area program and is loaded into the caller's private area storage.

    Additionally, if the calling address space cannot accommodate the target address for the shared library program, the program is treated like a private area program.
11. In order for a program to be honored as a shared library program, certain conditions must be met:
    - The program must be a z/OS UNIX program module; MVS library modules cannot be loaded into the shared region.
    - A sticky bit program that is found in the MVS search order is not honored as a shared library program.
    - The program cannot be a multiple-segment (split RMODE) load module; multiple-segment load modules are not supported in the shared library region.
    - The program must have read "other" permission and be link-edited as REENTRANT.
12. A shared library program can reside in a file system that was mounted with the NOSETUID operand.

Related services

- "deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage" on page 130

Characteristics and restrictions

There are no restrictions on the use of the loadhfs service.

Examples

For an example using this callable service, see "BPX1LOD (loadHFS) example" on page 1158.

loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage

Function

The loadhfs extended service loads an executable program by path name into the caller's process. This service provides all the functions of "loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name" on page 333 and also allows authorized users to load an executable program into common storage.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key unless the Lod_Directed flag is specified. When this flag is specified, the caller must be APF authorized, PSW Key 0-7, or Supervisor State.
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LDX):	31-bit
AMODE (BPX4LDX):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LDX,(Filename_length,
              Filename,
              Flags,
              Libpath_length,
              Libpath,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers need an additional parameter, Entry_point:

```
CALL BPX4LDX,(Filename_length,
              Filename,
              Flags,
              Libpath_length,
              Libpath,
              Entry_point,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

Filename_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Filename parameter. The length can be a value in the range 1 to 1023.

Filename
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Filename_length parameter

The name of a field that contains the file name of the program that is to be loaded. If the Filename parameter does not contain a slash (/), it is treated as a base name. This parameter should be in one of the directories listed in the supplied Libpath parameter. If the Libpath parameter is null, the file must be in the current directory. If the file name is not a base name (that is, it contains at least one slash), the name is used as is; the Libpath parameter is not used to locate the file.

If the file name is a base name, it can be up to 255 characters long.

If the Filename parameter represents a path name, each component of the path name (directory name, subdirectory name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending null character.

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The Flags parameter is a fullword field. The first three bytes contain option flags. The last byte can be data as defined by an option flag. These constants are defined in the BPXYCONS macro.

Constant	Description
Lod_Directed	Indicates that the target program is to be loaded into the supplied storage subpool. When this option flag is specified, the storage subpool is supplied as the last byte of the FLAGS parameter. This flag is only supported for authorized system callers (APF authorized or system key or supervisor state). Unauthorized callers specifying this flag receive a EPERM error return code. When this flag is specified, it is the responsibility of the caller to free the program storage. Only subpool 241 is currently supported; any other subpool specified results in an EINVAL error return code. The storage obtained for the target program is key 0 storage. Lod_Directed takes precedence over Lod_Ignore_Sticky, which in turn takes precedence over Lod_Error_St_ExLink.
Lod_Error_St_ExLink	Indicates that LOAD processing is to be bypassed if the file is an external link or has the sticky bit set on. If the file has the sticky bit set or is an external link, the request fails with return code EPERM (the operation is not permitted) and a reason code of JrExternalLink or JrStickyBit.
Lod_Ignore_Sticky	Indicates that the sticky bit for a file is to be ignored. If the file is sticky, it is loaded from the z/OS UNIX file system.

Note: If both Lod_Ignore_Sticky and Lod_Error_St_ExLink are specified, the Lod_Ignore_Sticky option is honored, and Lod_Error_St_ExLink is ignored.

Libpath_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the library path parameter. If a value of zero is specified, the library path parameter is ignored.

Libpath
Supplied parameter

Type: Structure

Length:
Specified by the Libpath_length parameter

The name of a field that contains the library path to be searched to determine the fully qualified path name of the file that is specified. The library path can contain a series of path names separated by colons. The path names in the list are searched one at a time until the specified file name is located. If the list of path names begins or ends with a colon, the working directory of the calling process is used to locate the file. Each path name in the list can have a maximum length of 1021 bytes.

The following is an example of a valid library path:

- **/usr1/bin:/grp1/bin:/bin**

Entry_point
Returned parameter (BPX4LDX only)

Type: Structure

Length:
Doubleword

The name of a field in which either an entry point address or the address of a structure is returned. If the Lod_Directed flag is specified, this service returns the address of a 24-byte structure that contains the length of the loaded program storage, followed by the start address of the loaded program, followed by the entry point address of the loaded program. The returned structure is mapped in the BPXYCONS macro.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The return value for this service is as follows:

- For an AMODE(31) caller, the name of a fullword in which the loadhfs extended service returns -1 if it is not successful. If it is successful, the loadhfs extended service returns the entry point address of the program that was loaded into storage, unless the Lod_Directed flag is specified. If the Lod_Directed flag is specified, this service returns the address of a 24-byte structure that contains the length of the loaded program storage, followed by the start address of the loaded program, followed by the entry point address of the loaded program. If the loaded program is an AMODE(31)

program, the high-order bit of the entry point address is ON. The returned structure is mapped in the BPXYCONS macro

- For an AMODE(64) caller, the Return_value is returned as either `0` if successful or `-1` if not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the loadhfs extended service stores the return code. The loadhfs extended service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The directed loadhfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory named in the Pathname parameter; it may lack execute permission for the file to be run; or the file to be run is not a regular file, and the system cannot run files of its type.
EAGAIN	The file changed during load processing (JrFileChangeDuringLoad).
EINVAL	An invalid parameter value was specified. The invalid parameter can be Filename_length, or FLAGS. If FLAGS is incorrect, a reason code of either JrOptionFlagsErr (unsupported FLAGS parameter value), or JrLodDirectedSubpoolError (unsupported value for the directed loadhfs subpool passed in the FLAGS parameter).
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Filename parameter. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
EMVSERR	An error occurred while loading a z/OS UNIX program (JrMVSLoadFailure or JrMVSPgmNotFound). Or an error occurred checking the caller's environment against the authorization of the file (JrNoListAuthPgmPath, JrNoListPgmCntlPath, JrProgCntl, JrAuthCaller).
ENAMETOOLONG	The Filename parameter is longer than 1023 characters; or some component of the file name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file name was specified, or one or more of the components of the specified Filename parameter were not found.
ENOEXEC	The specified file has execute permission, but it is not in the proper format to be a process image file.
ENOMEM	The file that is to be loaded requires more memory than is permitted by the hardware or the operating system, or a storage request failed for the directed load target (JrLodDirectedNoStorage).
ENOTDIR	A directory component of the Filename parameter is not a directory.
EPERM	The operation is not permitted. The Flags parameter was set to Lod_Error_St_ExLink. If the file has the sticky bit set or is an external link, the request fails with reason code of JrStickyBit or JrExternalLink, respectively. Or an unauthorized caller specified the Lod_Directed option flag (JrLodDirectedAuthErr).

Note: In addition to the return codes listed here, the loadhfs extended service can return additional errors for other failures that can occur on a stat or an open syscall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the loadhfs extended service stores the reason code. The loadhfs extended service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Note that usage notes 1–9 do not apply if you specify the Lod_Directed flag.

1. A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS LOAD service, with the following exceptions:
   - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
   - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the HFS program found is in storage that is modifiable by the caller, the prior copy is not reused.
2. If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is only used if the name is eight characters or less, otherwise the caller receives an error from the loadhfs service. For a sticky bit program, the file name is used if it is eight characters or less. If the file name is greater than eight characters, or the MVS program is not found, the program is loaded from the z/OS UNIX file system.
3. When it is running from a pthread_created thread (pthread), the specified file is loaded into storage and associated with the Initial Pthread Creating Task (IPT). This allows the program to be shared across multiple threads, without the problem of its disappearing unexpectedly when a thread terminates.
4. When the calling process is being debugged via the ptrace service, the following applies:
   - Programs that are loaded using this service are loaded into storage that is modifiable by the caller of the loadhfs service.
   - A call to this service generates a WastStopFlagLoad Ptrace event to the debugger process.
5. Because this service does not cause the specified program to be executed, the set-user-ID and set-group-ID flags have no impact on the process. These flags have meaning only for an execed or spawned program.
6. Because the z/OS UNIX file system is not an authorized library, the following restrictions apply:
   - Loading a program from the z/OS UNIX file system causes the program environment to become uncontrolled unless the executable file has the program control attribute turned on (ST_PROGCTL). Not having the program control attribute on prevents future invocations of authorized

programs like PADS programs. In addition, PADS programs should not attempt to load programs from the z/OS UNIX file system; the z/OS UNIX file system is considered an unauthorized library and can potentially be modified by users that do not have the same level of authorization as the PADS program.

- System key, supervisor state and APF-authorized callers receive an EMVSERR with reason code JrAuthCaller if the caller attempts to load a program from the z/OS UNIX file system, unless the executable file has the APF attribute turned on.

7. If a program that is loaded into storage with this service is not deleted from storage, the program remains in storage until the calling task terminates, if it is not a pthread. If the caller is a pthread, the program remains in storage until the Initial Pthread Creating Task (IPT) terminates.
8. The AUTHPGMLIST system parameter applies to this system call. AUTHPGMLIST specifies a z/OS UNIX file that contains a list of sanctioned directories or authorized program names. If activated, an additional level of security checking will be performed to ensure that the program being loaded is coming from an authorized directory in the z/OS UNIX file system or is an authorized MVS program name. For details about the sanction list, see the topic on using sanction lists in *z/OS UNIX System Services Planning*.
9. The following apply to shared program libraries:
   - Executables that have the ST_SHARELIB extended attribute turned on are considered system shared library programs. System shared library programs are the most optimal way to share large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage.
   - If the program to be loaded is determined to be a shared library program (that is, if the ST_SHARELIB extended attribute is on), the loadhfs service queries the shared library region to determine if the target program is there.

     When a shared library program is loaded anew into the shared region or reloaded from the shared region, the program is mapped from the shared region into the private area of the calling address space. It is important to note that, because the program is not actually reloaded from DASD into the private area of each calling address space, but only remapped from the shared region, shared library programs are more efficient in their utilization of system resources than normal private area programs. For this reason, programs that are to be shared across several address spaces in the system are good candidates for identification as shared library programs.

     If a target program is not in the shared library region and cannot be loaded into the region because of its attributes, the program is treated like a private area program and is loaded into the caller's private area storage.

     Additionally, if the calling address space cannot accommodate the target address for the shared library program, the program is treated like a private area program.
   - In order for a program to be honored as a shared library program, certain conditions must be met:
     – The program must be a z/OS UNIX program module; MVS library modules cannot be loaded into the shared region.
     – A sticky bit program that is found in the MVS search order is not honored as a shared library program.

  - The program cannot be a multiple-segment (split RMODE) load module; multiple-segment load modules are not supported in the shared library region.
  - The program must have read "other" permission and be link-edited as REENTRANT.
- A shared library program can reside in a file system that was mounted with the NOSETUID operand.

10. When the Lod_Directed flag is specified:
    - It is the responsibility of the caller to manage the storage associated with the loaded program. When Lod_Directed is specified, deletehfs cannot be used to remove the executable from storage. The executable will stay in storage until freed. The storage can be freed using the returned storage length and program start address.
    - It is the responsibility of the caller to use the CSVDYLPA ADD BYADDR(YES) service to create a CDE in order to provide serviceability information for the loaded program. Without this, serviceability functions, such as SLIP LPAMOD and IPCS WHERE, are not available for the loaded program.
    - The caller must save a copy of the returned program information after each call. The returned data structure is reused for each syscall by a given task. The returned program information structure is cleared if the call is made and an error occurs.
    - A program loaded with the Lod_Directed flag cannot be debugged using Ptrace debug mode.
    - The shared library program attribute, st_Sharelib, is ignored.
    - The sticky bit for a file is ignored whether or not Lod_Ignore_Sticky is specified.
    - If the file is an external link, the request will fail with return code of EPERM (the operation is not permitted) and a reason code of JrExternalLink whether or not Lod_Error_St_ExLink is specified.

Related services

None.

Characteristics and restrictions

There are no restrictions on the use of the loadhfs extended service.

Examples

For an example using this callable service, see “BPX1LDX (loadHFS extended) example” on page 1156.

lseek (BPX1LSK, BPX4LSK) — Change a file's offset

Function

The lseek callable services changes the file offset of a file to a new position. The file offset is the position in a file from which data is next read, or to which data is next written.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LSK):	31-bit
AMODE (BPX4LSK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LSK,(File_descriptor,
              Offset,
              Reference_point,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LSK with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the file whose file offset is to be changed. The file descriptor is returned from "open (BPX1OPN, BPX4OPN) — Open a file" on page 447.

Offset
Parameter supplied and returned

Type: Integer

Length:
Doubleword

The name of a doubleword that contains a signed number. The numeric part of the value is the amount (number of bytes) by which you want to change the offset. The sign indicates whether you want the offset to be moved forward or backward in the file.

This field is a doubleword, to accommodate large files. For normal processing with a singleword value, propagate the sign bit through the second word, so that the final doubleword value has a valid sign.

On successful completion, this field returns the new file offset.

Reference_point
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that represents an option. Reference_point indicates the point from which the offset is calculated. These values are mapped by the BPXYSEEK macro. For information on the contents of the macro, see "BPXYSEEK — Constants for lseek" on page 1036.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lseek service returns `0` if the request is successful, or -1 if it is not successful. Offset returns the new file offset if the request is successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lseek service stores the return code. The lseek service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The lseek service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not specify a valid, open file.
EINVAL	The Reference_point parameter contained something other than one of the three options; or the combination of the Offset and Reference_point parameters would have placed the file offset before the beginning of the file. The following reason codes can accompany the return code: JRLskOffsetIsInvalid, JRLskWhenceIsInvalid.
ESPIPE	The File_descriptor refers to a pipe, a FIFO special file, or a socket. The following reason code can accompany the return code: JRLskOnPipe.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lseek service stores the reason code. The lseek service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The Offset parameter gives the length and direction of the offset change. Reference_point parameter states where the change is to start. For example, assume that a file is 2000 bytes long, and that the current file offset is 1000:

Offset specified	Reference point	New file offset
80	SEEK_CUR	1080
1200	SEEK_SET	1200
-80	SEEK_END	1920
132	SEEK_END	2132

2. The file offset can be moved beyond the end of the file. If data is written at the new file offset, there is a gap between the old end of the file and the start of the new data. A request to read data from anywhere within that gap completes successfully, and returns bytes with the value of zero in the buffer and the actual number of bytes read.

   Seeking itself, however, does not extend the file. Only if data is written at the new offset does the length of the file change.

Related services

- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

When automatic conversion is enabled to ALL (that is, Unicode conversion is in effect), a lseek operation for a file containing multibyte characters can cause a subsequent read (BPX1RED/BPX4RED) or write (BPX1WRT/BPX4WRT) operation to fail due to the following cases:

1. The cursor jumps to another code page in a file tagged with a CCSID that has multiple code pages. The subsequent read will fail. However, using lseek to position the cursor to the beginning of the file is acceptable. The code page will be reset to the beginning default defined for the CCSID.
2. The previous write operation caused LFS to internally cache an incomplete multibyte character which, as a result of the lseek, is no longer convertible. The subsequent read or write will fail.
3. The cursor jumps to a position which is not on a character boundary. The subsequent read will fail.

Examples

For an example using this callable service, see "BPX1LSK (lseek) example" on page 1159.

lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name

Function

The lstat callable service obtains status information about a file. The lstat service is identical to the stat service, except when the path name specified is a symbolic link (a pointer to another file or directory). In this case, the status information that is returned relates to the symbolic link, rather than to the file to which the symbolic link refers. The stat service is explained in "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805.

For the corresponding service using a file descriptor, see "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1LST):	31-bit
AMODE (BPX4LST):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1LST,(Pathname_length,
              Pathname,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4LST with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Pathname.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of an area of length Pathname_length that contains the path name of the file for which you want to obtain status. The Pathname can be a pathname to a file, a linkname to a file (as returned by "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327), or a symbolic link name (as returned by "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812).

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area to which the service returns Status_area. To determine the value of Status_area_length, use macro BPXYSTAT; see "BPXYSTAT — Map the response structure for stat" on page 1057.

Status_area
Parameter supplied and returned

Type: Structure

Length:
Length of BPXYSTAT macro

The name of an area of length Status_area_length to which the service returns the status information for the file. Status_area is mapped by the BPXYSTAT macro; see "BPXYSTAT — Map the response structure for stat" on page 1057.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lstat service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lstat service stores the return code. The lstat service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The lstat service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process does not have permission to search some component of the Pathname prefix.
EINVAL	Parameter error—for example, a zero-length buffer. The following reason code can accompany the return code: JRBuffTooSmall.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or some component of the path name is longer than 255 characters. This could happen if a symbolic link was encountered during the resolution of Pathname, and the substituted string was longer than 1023 characters.
ENOENT	No file named Pathname was found, or Pathname was not specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the Pathname prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the lstat service stores the reason code. The lstat service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. All time fields in Status_area are in POSIX format.
2. The File Mode field in Status_area is mapped by BPXYMODE; see "BPXYMODE — Map the mode constants of the file services" on page 996. For information on the values for file type, see "BPXYFTYP — File type definitions" on page 967.
3. If no security label (SECLABEL) exists for the file, the security label field in the Status_area contains binary zeros.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196

- "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572
- "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872
- "utime (BPX1UTI, BPX4UTI) — Set file access and modification times" on page 879
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

To obtain information about a file, you need not have permissions for the file itself; however, you must have search permission for all of the directory components of Pathname.

Examples

For an example on the use of this system call, see "BPX1LST (lstat) example" on page 1160.

__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area

Function

The __map_init callable service creates a mapped megabyte area in the private area of the calling address space to hold a fixed number of the application's data blocks. This map area is divided into map blocks, each of which is a view onto a data block that is maintained in the kernel data space. The application can set the number of map blocks contained in the map area and the size, in megabytes, of each map block.

Once it has created the map area with the __map_init service, an application can use the __map_service (BPX1MMS, BPX4MMS) callable service to connect and disconnect blocks of storage in the map area.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MMI):	31-bit
AMODE (BPX4MMI):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MMI (FunctionCode,
              ParmListPtr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MMI with the same parameters. ParmListPtr is a doubleword pointer field.

Parameters

FunctionCode
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value indicating the type of map function the caller is requesting. The following is the only supported value:

Constant	Description
MMG_INIT	Create mapped megabyte area

This constant is defined in the BPXYMMG macro. See "BPXYMMG — Map interface for _map_init and _map_service" on page 991.

ParmListPtr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list. See "BPXYMMG — Map interface for _map_init and _map_service" on page 991 for the mapping of the parameter list.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __map_init service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __map_init service stores the return code. The __map_init service stores a return code only if the return value is -1. See

z/OS UNIX System Services Messages and Codes for a complete list of possible return code values. The __map_init service may return one of the following values in the Return_code parameter:

Return code	Explanation
EEXIST	An attempt was made to create more than one map area for the process (JRMapAlreadyActive).
ENOMEM	A request to initialize a map area failed for one of the following reasons: • There was insufficient storage in the caller's address space to obtain the map area needed to contain the map blocks (JRNoUserStorage). • All or part of the area defined by the address that was provided by the caller in MMG_AREAADDR was already allocated (JRStorNotAvail).
EPERM	One of the following errors occurred: • The caller is not permitted to the BPX.MAP resource in the FACILITY class. Superuser status (UID=0) is not sufficient (JRNotAuthMAP). • The BPX.MAP resource in the FACILITY class is not defined, and the user is not a superuser (JROK).
EMVSSAF2ERR	An error occurred in the security product.
EINVAL	One of the following errors occurred: • The FunctionCode parameter contains a value that does not represent a supported function (JRMapBadFunction). • The number of blocks specified (_MMG_NUMBLKS) was either negative or zero (JRNegativeValueInvalid). • The number of megabytes per block specified (_MMG_MEGSPERBLK) was either negative or zero (JRNegativeValueInvalid). • A reserved field contains nonzero data (JRReservedValueInvalid). • The request specified a map address (_MMG_AREAADDR) that was not above the line, or that was not on a megabyte boundary (JRBadAddress).
EFAULT	An argument of this service contained an address that was not accessible to the caller (JRMapBadStorage).
EMVSERR	One of the following occurred: • There was an unexpected error (JRMapUnexpectedErr). • An attempt to process the new map area failed in RSM (JRIarvServ).

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __map_init service stores the reason code. The __map_init service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The __map_init and __map_service callable services allow applications to manage an unlimited number of data blocks, each of which can hold some number of megabytes of data. They provide a fast way to connect up to persistent memory for applications that need more shared memory than will fit in the address space.
2. It is intended that an application will call the __map_init service once to create the map area. The map area should be large enough for the biggest expected usage.
3. A process may have one, and only one, map active at a given time. There is currently no way to terminate a map area once it has been established without terminating the establishing process.
4. At any point in time, an application can view as many data blocks as were specified at initialization of the map area, and it can have many times this number of data blocks defined and residing in kernel data spaces.
5. The map area may be shared among one or more processes. Sharing may only be between a parent and any children that were created after the parent created the map area with a call to the __map_init service. Children that were created before the call do not have access to the map area, nor can they gain access to it through any service.
6. A map area is not propagated across a spawn or preserved across an exec. Unlike most attributes on fork, the map area that is inherited by a child is empty; none of the map blocks are connected to data blocks, regardless of how many data blocks are currently connected to the parent's map area.
7. A map area persists until the process that created it terminates. Once that process terminates, all map activity against the data blocks is shut down. Currently connected blocks may continue to be used until they are disconnected. New blocks cannot be created, nor can a process connect to an existing data block. Once all data blocks have been disconnected by all processes, the map area is ended. A process that has been detached from a map area by disconnecting from all data blocks may create a new map area.
8. Each process that is sharing a map (parent, child, or grandchild) gets a map area that is located at the same virtual storage address as the map originator and that consists of map blocks that are the same size and number as those of the originator. Each process that is sharing a map manages its own map area in terms of the data blocks that are connected, and each process determines which data block is viewed through which map area block.
9. The initial process forks worker processes, which inherit the map area at the same virtual address. Because the map area is at the same virtual address, storage blocks can be connected to the same block in map areas of different worker processes, and pointers can be used to point to data in this and other blocks. (This assumes that they are always connected at the same location in the map area.)
10. As worker processes perform their tasks, they can request that new blocks of storage be created in the map area. Each block has a token associated with it, which allows other worker processes to connect to the same block. In this respect, the map area acts like shared memory.
11. The worker processes can connect as many blocks to their map area as will fit.
12. When the worker process has no further need for a data block, it can disconnect it from the map area. Following a delete request for a block, the block is actually freed when the last worker process disconnects from it.

13. When a worker process has finished using a data block, the storage can be freed. The data is actually freed when the last worker process disconnects from that block.
14. Using the __map_init and __map_services, an application could create multiple gigabytes of storage, of which only certain blocks are mapped into the worker processes at a given time.
15. There is no explicit call to delete the map area.

Related services

- "__map_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services"

Characteristics and restrictions

Users of __map_service can create and manage a tremendous amount of data, causing the kernel to consume a large amount of system resources. To prevent abuse of such power, the __map_init service requires that the user be permitted to the BPX.MAP resource in the FACILITY class. (The __map_service callable service does not check for authority to BPX.MAP, because it does not perform any functions without first completing a __map_init request.)

Examples

For an example using this callable service, see "BPX1MMI (__map_init) example" on page 1162.

__map_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services

Function

The __map_service callable service performs the following operations on one or more data blocks in a memory map area created by the __map_init service:

- Creates a new data block
- Connects to an existing data block
- Disconnects from a data block
- Frees the backing storage for a data block
- Changes the read or write permission for a data block

Before an application can use this service, it must invoke the __map_init callable service to create a mapped megabyte area to hold its data blocks. See "__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area" on page 352.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MMS):	31-bit
AMODE (BPX4MMS):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MMS (FunctionCode,
              ParmListPtr,
              ArrayCount,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MMS with the same parameters. ParmListPtr is a doubleword pointer field.

Parameters

FunctionCode
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value indicating the type of map function the caller is requesting. The following is the only supported value:

Constant	Description
MMG_SERVICE	Perform one or more operations on map blocks: • Activate a new data block (MAP_NEWBLOCK) • Connect to a data block (MAP_CONN) • Disconnect from a data block (MAP_DISCONN) • Free the backing storage for a data block (MAP_FREE) • Change the read or write permissions for a data block (MAP_CNTL)

These constants are defined in the BPXYMMG macro. See "BPXYMMG — Map interface for _map_init and _map_service" on page 991.

ParmListPtr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list for the specified function. See "BPXYMMG — Map interface for _map_init and _map_service" on page 991 for the mapping of the parameter lists.

ArrayCount
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of entries in the array that is contained in the parameter list provided by ParmListPtr. The value specified in the ArrayCount parameter must be greater than or equal to 1 and less than or equal to 1000.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which __map_service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __map_service callable service stores the return code. The __map_service callable service stores a return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The __map_service callable service may return one of the following values in the Return_code parameter:

Return code	Explanation
EEXIST	A request was made to perform a service on a block, but either a map area is not currently active for the process, or the map area is in the process of being shut down (JRMapNotActive).
ENOMEM	A request to create a new block or connect to an existing block was made with a zero block address, specifying that the __map service is to locate the address of a free map block, but there are no unused blocks in the map area to satisfy the request (JRMapOutOfBlocks).

Return code	Explanation
EINVAL	One of the following errors occurred: • The FunctionCode parameter contains a value that is not a supported function, or the service call parameter list field MMG_SERVICETYPE contains an unsupported value (JRMapBadFunction). • A request was made to connect to a block, free the backing storage for a block, or change the access state (control operation) for a block, but the token provided does not match that of any allocated block in the backing storage (JRMapTokenNotFound). • A MAP_NEWBLOCK or MAP_CONN request specified a map area block that is already in use (JRMapBlockInUse). • A request was made to connect to a block in the backing storage that is currently marked to be freed. The connection is not permitted (JRMapBlockFreePending). • A request was made to disconnect from a map block, but the block is not currently in use in the map area for this process (JRMapBlockNotInUse). • A reserved field contains nonzero data (JRReservedValueInvalid). • A block address was provided, but either it is not in the map area or it is not on a map block boundary (JRBadBlkAddr). • The array count was negative, zero, or greater than the maximum number of array elements permitted (1000) (JRMapArrayCountErr).
EFAULT	An argument of this service contained an address that was not accessible to the caller (JRMapBadStorage).
EMVSERR	One of the following occurred: • There was an unexpected error (JRMapUnexpectedErr). • A request to create a new block, connect to an existing block, disconnect from an existing block, or change the read or write permissions for a block failed in RSM (JRIarvServ).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __map_service callable service stores the reason code. The __map_service callable service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The __map_service callable service is designed to perform storage connects and disconnects very quickly. No data movement occurs.
2. Input to the __map_service callable service is an array of requests. Each request is processed in order until all requests have been successfully processed, or until an error occurs. When an error occurs, some requests may have been processed and some may not. An output flag on the array elements, _mmg_ReqFail, indicates the requests that have and have not been processed.

The flag is off for array elements that have been processed successfully. The flag is on for the request that failed and all requests that had not yet been processed at the time of the failure.

3. The __map_service callable service allows an application to create a new data block and specify which map area block is to be used to view this data block. The map area block that is to contain the new data block must be free, that is, not currently connected to another data block. The kernel assigns a unique token to the new data block and returns this token to the application. The token is later used to identify the data block to subsequent calls to __map_service. The application may modify the new data block contained within its map block in any way it chooses.
4. Storage blocks are initially connected in write mode. When a block is in write mode, all worker processes that have the block connected have the block in write mode. If the block access is changed to read-only, all worker processes that have the block connected have the block in read-only mode.
5. Any areas within the map area that do not have a block connected are in the hidden state. Any reference to storage in the hidden state triggers a SIGSEVG signal.
6. If the initial process or a worker process forks, the child process inherits a map area that is initialized to the hidden state.
7. When an application has finished using a data block, it may do one of several things:
   - If it no longer needs the data block, it can disconnect it from the map block and request that the kernel free the data block. Once the data block has been freed and its use count has gone to zero, the data no longer exists in the kernel data space and is no longer available for processing.
   - If the data is still valuable, but is not currently needed, the application can request that the map area block be disconnected from the data block (without freeing it). This leaves the data block in a kernel data space for later use, while freeing the map area block for use in processing other data blocks. The map area block is hidden as part of the disconnect, and an 0C4 abend occurs if the application attempts to reference any storage in the map area block.

     Later, when a disconnected data block needs to be processed, the application can call __map_service with a connect request, specifying the token for the data block and the address of the map block it is to be attached to for processing. The __map_service callable service attaches the specified data block to the appropriate map block for use by the application. The block is read-only or read/write based on its state as of the last control operation.
   - A data block may be freed without having first been connected by a call to __map_service with a free request, specifying the token of the data block.
   - An application can control the access state (read or read/write) of a connected data block by calling __map_service with a control request and specifying the desired target state. Because special mechanisms are used for the sharing of a data block between several processes, a state change is against the data block and affects all users of the data block (not just the current user's data block). State changes persist across disconnects. If a data block is made read-only and all users disconnect from the data block, the next user to connect to the data block obtains the block read-only.

See the description of the __map_init callable service, “Usage notes” on page 355, for more information about using these two related services.

Related services

- "__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area" on page 352

Characteristics and restrictions

Users of __map_service can create and manage a tremendous amount of data, causing the kernel to consume a large amount of system resources. To prevent abuse of such power, the __map_init service requires that the user be permitted to the BPX.MAP resource in the FACILITY class. (The __map_service callable service does not check for authority to BPX.MAP, because it does not perform any functions without first completing a __map_init request.).

Examples

For an example using this callable service, see "BPX1MMS (__map_service) example" on page 1163.

mkdir (BPX1MKD, BPX4MKD) — Make a directory

Function

The mkdir callable service creates a new, empty directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MKD):	31-bit
AMODE (BPX4MKD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MKD,(Pathname_length,
              Pathname,
              Mode,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MKD with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Fullword

The name of a fullword that contains the length of the full Pathname of the directory. The name can be up to 1023 bytes long. Each component of the name (between delimiters) can be up to 255 bytes long.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field, of length Pathname_length, that contains the full name of the directory.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions you grant to yourself, to your group, and to any user.

The file type is identified using the BPXYFTYP mapping macro and permissions that are specified with the BPXYMODE mapping macro. See "BPXYFTYP — File type definitions" on page 967 and "BPXYMODE — Map the mode constants of the file services" on page 996.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mkdir service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mkdir service stores the return code. The mkdir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The mkdir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process did not have search permission on some component of Pathname, or did not have write permission on the parent directory of the directory to be created.
EEXIST	There is already a file or directory with the given Pathname. The following reason code can accompany the return code: JRMkDirExist.
EFBIG	A request to create a directory is prohibited because the file size limit for the process is set to 0.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMLINK	The link count of the parent directory has already reached the maximum defined for the system. Refer to the LINK_MAX in "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464, or to "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191.
ENAMETOOLONG	Pathname contains more than 1023 characters; or a component of the name is longer than 255 characters.
ENOENT	Some component of Pathname does not exist; or the Pathname parameter is blank.
ENOSPC	The file system does not have enough space to contain a new directory; or the parent directory cannot be extended.
ENOTDIR	A component of Pathname is not a directory.
EROFS	The parent directory of the directory to be created is on a read-only file system. The following reason code can accompany the return code: JRMkDirROnly.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mkdir service stores the reason code. The mkdir service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The file permission bits that are specified through the Mode parameter are modified by the file creation mask of the calling process (see "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866). They are then used to set the file permission bits of the new directory.
2. The new directory's owner ID is set to the effective user ID (UID) of the calling process.

3. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
   - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
   - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
4. The mkdir service sets the access, change, and modification times for the new directory. It also sets the change and modification times for the directory that contains the new directory.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805
- "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866

Characteristics and restrictions

There are no restrictions on the use of the mkdir service.

Examples

For an example using this callable service, see "BPX1MKD (mkdir) example" on page 1161.

mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file

Function

The mknod callable service creates a new directory, a regular file, a character special file, or a FIFO special file (named pipe).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MKN):	31-bit
AMODE (BPX4MKN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MKN,(Pathname_length,
              Pathname,
              Mode,
              Device_identifier,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MKN with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname of the special file to be created.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file. The length of this field is specified in Pathname_length.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword in which the mode field is specified. The mode field specifies the file type and the permissions you grant to yourself, to your group, and to any user. Specify the file type with the BPXYFTYP mapping macro, and specify permissions with the BPXYMODE mapping macro. See “BPXYFTYP — File type definitions” on page 967 and “BPXYMODE — Map the mode constants of the file services” on page 996.

Device_identifier
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains a device identifier, or 0. The high-order 16 bits of Device_identifier is the device major number. The device major number corresponds to a device driver that supports a class of devices—for example, interactive terminals. The low-order 16 bits of Device_identifier is the device minor number. The device minor number corresponds to a specific device within the class of devices that are referred to by the device major number. Specify Device_identifier if you are creating a character special file.

If a FIFO, directory, or regular file is being created, Device_identifier is ignored.

The following device major numbers are currently defined:

Device major numbers	Device class
1	Master pseudoterminal
2	Slave pseudoterminal
3	/dev/tty
4	/dev/null, /dev/zero, and /dev/random
5	/dev/fd*n*
6	Sockets
7	OCSRTY
8	OCSADMIN
9	/dev/console

For device major numbers 1, 2, and 7, the device minor numbers refer to specific pseudoterminal pairs and the values range from 0 and one less than the maximum number of pseudoterminal pairs defined by the installation.

For device major numbers 3, 6, 8, and 9, the device minor number is ignored.

For device major number 4, device minor numbers represent files as follows:

Device minor numbers for device major number 4	File
0	/dev/null
1	/dev/zero
2	/dev/random and /dev/urandom

For device major number 5, the device minor number value represents the file descriptor to be referred to. For example, device minor 0 refers to file descriptor 0.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mknod service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mknod service stores the return code. The mknod service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The mknod service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process does not have permission to search some component of Pathname; or does not have write permission for the directory of the file to be created.
EEXIST	A file or directory named Pathname already exists. The following reason code can accompany the return code: JRSpFileExists.
EFBIG	A request to create a new file is prohibited because the file size limit for the process is set to 0.
EINVAL	The file type specified in the Mode parameter is not 1, 2, 3 or 4. The following reason code can accompany the return code: JRMknodInvalidType.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or a component Pathname has a name longer than 255 characters.
ENOENT	A component of Pathname was not found; or no pathname was specified. The following reason code can accompany the return code: JREndingSlashMknod.
ENOTDIR	A component of Pathname is not a directory.
EPERM	The operation is not permitted. The operation requested requires a superuser authority. The following reason code can accompany the return code: JrUserNotPrivileged.
EROFS	The directory of the file is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFilesetMknodReq.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mknod service stores the reason code. The mknod service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The file permission bits of Mode are modified by the process's file creation mask (see "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866). They are then used to set the file permission bits of the file being created.

2. The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
   - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
   - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
3. The mknod service sets the access, change, and modification times for the new file. It also sets the change and modification times for the directory that contains the new file.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805
- "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866

Characteristics and restrictions

When the mknod service is requested to create a character special file, a directory or a regular file, it is a privileged operation and requires superuser authority.

Examples

For an example using this callable service, see "BPX1MKN (mknod) example" on page 1162.

mmap (BPX1MMP, BPX4MMP) — Map pages of memory

Function

The mmap callable service establishes a mapping between a process's address space and a HFS file.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW Key 2 or PSW Key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MMP):	31-bit
AMODE (BPX4MMP):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MMP,(Map_address,
              Map_length,
              Protect_options,
              Map_type,
              File_descriptor,
              File_offset,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers need an additional parameter, Returned_map_address:

```
CALL BPX4MMP,(Map_address,
              Map_length,
              Protect_options,
              Map_type,
              File_descriptor,
              File_offset,
              Returned_map_address
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

Map_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains zero, or the address of an area within the address space at which the system is to attempt to map the requested file.

If the value of map_address is zero, the system has complete freedom in selecting the location within the address space at which the requested file is mapped.

If the value of map_address is not zero, the value that is specified is taken to be a suggestion of an address near which the mapping is to be placed. For non-MAP_FIXED requests, the system attempts to create the mapping at the address specified by map_address. The address is truncated to the nearest page boundary when a map type of MAP_SHARED or MAP_PRIVATE is specified, and to the nearest segment or megabyte boundary when a map type of MAP_MEGA is specified. If it is unsuccessful, it proceeds as if a map_address value of zero were specified.

For MAP_FIXED requests, the value of map_address must be a multiple of the page size when MAP_PRIVATE or MAP_SHARED is specified, and a multiple of the segment size when MAP_MEGA is specified. (If MAP_MEGA is specified, the value that is specified in map_address must be equal to zero or equal to or greater than 16 megabytes, or the request is failed with EINVAL.) The MAP_FIXED request fails with an EINVAL if any portion of the requested range is already in use for any reason (including a previous mapping).

The map_address supplied by the caller cannot be above the 31-bit addressability bar (X'7FFFFFFF'), or the request will fail (EINVAL).

Map_length
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of a fullword (doublword) field that contains the size (in bytes) of the memory mapping that is to be created. The length that is specified must be less than or equal to the size of the file, and must not cause the address space REGION to be exceeded. Mapping operations are performed over whole pages, or whole segments when MAP_MEGA is specified. If the length is not a multiple of the page size or segment size, the entire trailing portion of the page or segment (up to the end of the file) is also mapped into the user storage. The trailing portion of the page or segment in which an end of file occurs contains binary zeros.

Protect_options
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the value of the memory access protection flags. The protect_options parameter indicates whether read, write, execute, or some combination of accesses are permitted to the mapped data. It can be set to either PROT_NONE, or a combination (using, for example, an inclusive OR) of one or more of the other access protection flags. The constant values for these flags are defined in the BPXYCONS macro. (See "BPXYCONS — Constants used by services" on page 952.) For MAP_MEGA mappings, the value that is specified for protect_options has a global effect on all current maps to the same file-offset range. For example, if PROT_READ is specified, all active maps have their protection for the same file-offset range changed to a protection of read.

Constant	Description
PROT_READ	Mapped data can be read. The file descriptor must have been previously opened with at least read access.
PROT_WRITE	Mapped data can be written and read. To select the PROT_WRITE option, if a map_type of MAP_SHARED is specified, the file descriptor must have been previously opened with Read/Write access. If MAP_PRIVATE is specified, the file descriptor only needs to have been opened with read access.
PROT_EXEC	Mapped data can be executed. This option is treated as if PROT_READ has been specified.
PROT_NONE	Mapped data cannot be accessed.

Map_type
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the mapping type. The constant values for map_type are defined in the BPXYCONS macro.

Constant	Description
MAP_SHARED	All changes to the mapped data are shared. Modifications to the mapped data are visible to all other processes that map the same file-offset range.
MAP_PRIVATE	All changes to the mapped data are private. Modifications to the mapped data are visible only to the calling process, and do not change the underlying file. To use this option, the hardware must provide the suppression-on-protection support.
MAP_MEGA	All changes to the mapped data are shared. Modifications to the mapped data are visible to all other processes that map the same file-offset range. The protection attributes of file-offset ranges are common among all active maps. Changes to the protection option of a file-offset range are global, and immediately affect all active maps.
MAP_FIXED	The mapping must be placed at exactly the location specified by the map_address parameter.

You must specify MAP_SHARED, MAP_PRIVATE, or MAP_MEGA, but you cannot specify more than one. MAP_FIXED is optional when any of the other map options is specified. To specify both MAP_FIXED and MAP_SHARED, for example, use a map_type value equal to the inclusive OR of these two constants.

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of an open file that is to be mapped to process storage. The file descriptor is returned by "open (BPX1OPN, BPX4OPN) — Open a file" on page 447. You can only specify the file descriptor of a regular file.

For a MAP_MEGA mapping, if this is the first map to the file that is represented by the specified file descriptor, the protect_options that can be specified for this file by this map request (and by all future map or mprotect requests, by this or any other process mapping to the same file) are determined by whether the file was opened for read or for read and write. If the file was opened for read but not write, only PROT_READ, PROT_EXEC, or PROT_NONE are allowed. If the file was opened for write, any of the protection options are accepted. Once PROT_WRITE is allowed for a file, all map requests must provide a file descriptor that was opened for write, or the map request is failed.

File_offset
Supplied parameter

Type: Integer

Length:
Doubleword

The name of a doubleword that defines which part of the file is to be mapped. It contains the offset into the file at which the map_length is to begin. The value of file_offset must be a multiple of the page size when MAP_PRIVATE or MAP_SHARED is specified, and a multiple of the segment size when MAP_MEGA is specified. The offset plus the map_length must fall within the current size of the file.

Returned_map_address
Returned parameter (BPX4MMP only)

Type: Address

Length:
Doubleword

The name of a doubleword in which the mmap service returns the 64-bit address where the mapping was placed, if the request is successful.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the mmap service returns the 31-bit address at which the mapping was placed, if the request is successful; or -1, if it is not successful. In AMODE 64, if mmap is successful, 0 is returned in this field and the 64-bit address is returned in the Returned_map_address parameter.

Upon successful completion, the mmap service has established a mapping between the process's address space, at an address returned in the Return_value parameter, for map_length bytes, to the file that is represented by the file_descriptor, at the specified file_offset, for a length of map_length bytes. The specified access protections and mapping type are set for the mapped range.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mmap service stores the return code. The mmap service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

The mmap service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: • The file descriptor is not open for read, regardless of the protection specified. (JRRFileNoRead) • The file descriptor is not open for write, and PROT_WRITE was specified for a MAP_SHARED type mapping. (JRWFileRDOnly) • A MAP_MEGA request specified PROT_WRITE, but the first active map to a file was done with a file descriptor that was not open for write. (JRWFileMapRDonly)
EAGAIN	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)
EBADF	One of the following conditions occurred: • The file specified by the file_descriptor parameter does not represent a standard file. (JRNotStdFile) • The file specified by the file_descriptor parameter is not a valid open file descriptor. (JRs belong to fstat() or w_ioctl())
EINVAL	One of the following conditions occurred: • MAP_FIXED was specified, and the requested range was not available. The range could be previously allocated, or it could be outside the address space region. (JRAddressNotAvailable) • MAP_FIXED was specified, and the value of the map_address parameter is not a multiple of the page size. (JRNotPage) • The value of the file_offset parameter is not a multiple of the page size. (JRNotPage) • The value specified in the map_type parameter is incorrect. (JRMmapBadType) • The value specified in the protect_options parameter is incorrect. PROT_NONE cannot be specified in combination with any other options. (JROptNotSupp) • The file was extended and subsequently mapped beyond the original EOF point while an existing memory map containing the original EOF point was outstanding. (JRMmapOverEof) • The file_offset value must be zero or larger. (JRNegativeValueInvalid) • An attempt was made to map a file that is already mapped, but with a different specification of MAP_MEGA. At any point in time, a file may be mapped with or without the MAP_MEGA option, but not both with and without the MAP_MEGA option. • The file was already mapped by another process into a storage key that does not match the PSW key of the caller. (JrKeyMismatch) • In 64-bit mode, an address greater than 31 bit addr was passed in map_address (JrAddressNotAvailable). • In 64-bit mode, a length greater than X'7FFFFFFF' was passed in map_length (JrInvParmLength).
EMFILE	The number of mapped regions would exceed a system limit: • The system-wide limit on the amount of memory consumed by memory-mapped areas was exceeded. (JRMmapStgExceeded) • The per-process limit on the number of outstanding memory-mapped areas was exceeded. This limit is the same as the limit on the number of files a process can have open at any given time. (JRProcMaxMmap)

Return_code	Explanation
ENODEV	The file descriptor refers to a file for which mmap is not supported (for example, a terminal). (JRNotSupportedForFileType)
ENOMEM	One of the following conditions occurred: • MAP_FIXED was specified, and the requested range (map_address, map_address + map_length) exceeds that allowed for the address space of a process. (JRAddressNotAvailable) • There is insufficient room in the address space to effect the mapping. (JRNoUserStorage) • There is insufficient shared storage available in the system to satisfy this request. (JRShrStgStorage)
ENOSYS	MAP_PRIVATE was specified, but the required suppression-on-protection hardware support was not available. (JRHardware)
ENXIO	The addresses in the range (file_offset, file_offset + map_length) are not valid for the specified file descriptor. (JRMmapFileAddress)

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mmap service stores the reason code. The mmap service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The mmap service supports only regular files. Any other type of file is not processed.
2. The mmap resources are maintained at a process level. This means that the termination of the thread that invoked the mmap service does not cause the associated mapping to be removed. The mmap resources are freed when the process ends.
3. The mmap service adds an extra reference to the file that is associated with the specified file descriptor that is not removed by a subsequent close on that file descriptor. This reference is removed when there are no more mappings to the file. The access level (read/write) that was established when the file was opened is enforced for the life of the memory-mapped area, independent of subsequent activity that occurs upon that file descriptor.
4. The storage that is allocated by the mmap service is allocated in fetch-protected key 2 or 8 storage, depending on the key of the caller mapping the file for the first time. It is allocated with memory that can have both virtual addresses and real addresses above the 16-MB line. The storage cannot be freed by an unauthorized user. The allocated storage comes out of the user region.
5. Specifying a target Map_address can have a negative impact on the address space. For example, specifying a Map_address at the top of the private area, below the 16MB line, could prevent system code from successfully obtaining below-the-line storage.

6. All tasks and SRBs within the address space that issued the mmap request can access the memory allocated by the mmap service, but only threads within the process that created the mmap area are permitted to invoke any subsequent memory map services against that mmap instance. The protection level that is established by this process is enforced for all accesses that are made to that range within the address space.
7. All memory-mapped areas, along with their mapping types and mprotect established access levels, are propagated to the child process during fork processing. The user is responsible for serialization across multiple threads.
8. If MAP_PRIVATE is specified, the initial write reference to the memory-mapped region creates a private copy of the memory-mapped page, and redirects the mapping to the copy. Note that the copy is not created until the first write. Until the first write, updates that are made to that region by other processes that are mapped by MAP_SHARED with the same file-offset range are visible.
9. Applications that use the MAP_PRIVATE support may need to be aware of page boundaries when updates are performed, because an update to a single byte causes an entire page to no longer receive updates that are made by other processes mapped with the same file-offset range.
10. To serialize access to a file-offset range that is being accessed by multiple processes, you can use lockf, fcntl, or semaphores. Serialization should be obtained when the incore copy of the data is being updated, or when the file is being updated using msync.
11. If a sparse file is memory-mapped, accessing a page that has never been written to in the file causes a page of binary zeros to be generated.
12. The mmap service allows access to HFS files through address space manipulation, instead of through the read/write services. After the file is mapped, the process can access it by using the data at the address to which the file was mapped.

    The following code sample illustrates how an existing program might be changed to use the mmap service:

```
fd = open(...)
lseek(fd, file_offset)
read(fd, buffer, length)

/* ...(use data in buffer) ... */

becomes

fd = open(...)
address = mmap (0, length, PROT_READ, MAP_PRIVATE, fd, file_offset)

/* ...(use data at address) ... */
```

13. Constants used for this callable service are defined in the BPXYCONS macro. See “BPXYCONS — Constants used by services” on page 952.
14. The mmap service is not enabled to map storage above the 2-gigabyte addressing range. It is enabled only to be called from a 64-bit program with a 64-bit parameter list.

Related services

- “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 174
- “ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file” on page 203
- “mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping” on page 384

- "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403
- "munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses" on page 407
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores" on page 631
- "setrlimit (BPX1SRL, BPX4SRL) — Set resource limits" on page 698
- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819

Characteristics and restrictions

1. The MAP_PRIVATE support requires the suppression-on-protection hardware feature.
2. The same file-offset range can be mapped multiple times within a given address space (to different virtual addresses), each with unique protection levels. A memory-mapped file-offset range can partially or fully overlap other existing mapped file-offset ranges. This support also holds true across multiple processes.
3. The mmap service can never be used to extend or truncate the size of a file. If a page is updated beyond the EOF mark of the original memory-mapped file, the portion beyond the EOF mark is not written to the file.
4. A file that is memory-mapped can be appended by another process while the memory map is active; no overlays will occur. However, the newly created area cannot be mapped across the original EOF point, unless either the EOF point falls on a 4K boundary, or the original memory mapping is unmapped.
5. When a given file-offset is memory-mapped, unpredictable results will occur if the file is truncated to a point which resides within the memory mapped range. These results may include the abnormal termination of the task that is accessing the memory-mapped area.
6. If other processes modify the contents of the file that is using the write service while mapped ranges are active for that file-offset, results will be unpredictable, unless specific serialization actions are taken by the user. See "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403 for details.
7. There is a limit on the number of active memory maps that a process can have outstanding at any given time. The system administrator defines this limit by specifying the maximum number of files a process can have open. Even though a single value is set that limits both files and mmaps, the two limits are enforced independently of one another.
8. Memory maps with the MAP_MEGA option use storage in units of megabytes. Extensive use of MAP_MEGA on very small files, or on small ranges of larger files, can be wasteful. MAP_MEGA is best used on large files.
9. Memory maps of very large files by several processes can realize substantial savings of system common area usage when you use the MAP_MEGA option.

Examples

For an example using this callable service, see "BPX1MMP (mmap) example" on page 1163.

mount (BPX1MNT) — Make a file system available

Function

The mount callable service mounts a file system, making the files in it available for use.

Note: There is no 64-bit version of the mount callable service. To get equivalent function, use "__mount (BPX2MNT, BPX4MNT) — Make a file system available" on page 381 in 64-bit mode.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MNT,(MountPoint_length,
              MountPoint_name,
              File_system_name,
              File_system_type,
              Mount_mode,
              Parm_length,
              Parm,
              Return_value,
              Return_code,
              Reason_code)
```

Parameters

MountPoint_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of MountPoint_name.

MountPoint_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the MountPoint_length parameter

The name of a field that contains the name of the mount point. The length of this field is specified in MountPoint_length.

File_system_name
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
44 bytes

The name of a 44-character field that identifies the file system to be mounted. The name must be left-justified and padded with blanks.

File_System_type
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
8 bytes

The name of a field that contains the 8-character file system type. This corresponds to the type of file system that was defined by a FILESYSTYPE parameter of the BPXPRMxx parmlib member.

Mount_mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains binary flags. The flags can indicate:

- The mount mode (read or read/write)
- Whether the mount request must complete synchronously
- Whether SETUID is not allowed
- NOSECURITY.

This parameter is mapped by the macro BPXYMTM; see "BPXYMTM — Map the modes for mount and unmount" on page 1000 for details.

Parm_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the file system-specific parameters (Parm). The maximum length is 500 bytes.

Parm
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Parm_length bytes

The name of a field, of length Parm_length, that contains the file-system-specific parameters. These have a maximum of 500 bytes.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mount service returns `0` or `1` if the request is successful, or -1 if it is not successful. A Return_value of 1 indicates that the mount will complete asynchronously.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mount service stores the return code. The mount service always returns Return_code if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The mount service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBUSY	The file system to mount is quiesced; or no more locks are available. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced.
EINVAL	There was a parameter error. Verify the Mount_mode and File_system_type. Other reasons for this error include: • The mount point is a root of a file system. • The file system is already mounted. • parm_length is too long. • A mounted file system has a real or alias name that conflicts with this mount request. One of these situations occurred: – A file system was previously mounted using an alias data set name, and the corresponding real data set name conflicts with the file system name specified by this mount request. – The file system name specified on this mount request is an alias data set name that has a real data set name which conflicts with the name of a previously mounted file system. Resolve the duplicate file system names. Resolve the duplicate file system names and reissue the mount request. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced, JRIsMountedRealName.
EIO	An I/O error occurred.

Return_code	Explanation
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENOENT	The mount point does not exist. The following reason code can accompany the return code: JRMountPt.
ENOMEM	There is not enough storage space available to mount this file system.
ENOTDIR	The mount point is not a directory. The following reason code can accompany the return code: JRMountPt.
EPERM	Insufficient authority to do the mount.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mount service stores the reason code. The mount service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The mount service effectively creates a virtual file system. After a file system is mounted, references to the pathname that is mounted refer to the root directory on the mounted file system.
2. A file system can be mounted at only one point.
3. The physical file system might complete the mount operation asynchronously, which is indicated by a Return_value of 1. The w_getmntent callable service can then be used to determine if the file system has been mounted.

Related services

- "umount (BPX1UMT, BPX4UMT) — Remove a virtual file system" on page 867
- "w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems" on page 894

Characteristics and restrictions

1. In order to mount a file system, the caller must be an authorized program, or must be running for a user with appropriate privileges (see "Authorization" on page 8).
2. Typically, an EBUSY error condition is returned when a file system is quiesced. In a sysplex, however, the mount syscall is suspended until the file system becomes unquiesced.

Examples

For an example using this callable service, see "BPX1MNT (mount) example" on page 1163.

__mount (BPX2MNT, BPX4MNT) — Make a file system available

Function

The __mount callable service mounts a file system, making the files in it available for use.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX2MNT):	31-bit
AMODE (BPX4MNT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX2MNT,(Mnte_length,
              Mnte,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MNT with the same parameters.

Parameters

Mnte_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Mnte and its associated data structures, such as parameter string length.

Mnte
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Mnte_length parameter

The MNTE data structure. This is composed of a header field, the body field and an additional area for the parameter string if one is being used. This structure is mapped by BPXYMNTE (see "BPXYMNTE — Map response and element structure of w_getmntent" on page 993). See the usage notes for the fields in this data structure that must be set for the different __mount requests.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __mount service returns 0 or 1 if the request is successful, or -1 if it is not successful. A Return_value of 1 indicates that the mount will complete asynchronously.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __mount service stores the return code. The __mount service always returns Return_code if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __mount service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCESS	The user who is doing this nonprivileged mount does not have access to either the mount point or root. The following reason code can accompany the return code: JRNoMntPtAccess, JRNoRootAccess.
EBUSY	The file system to be mounted is quiesced; or no more locks are available. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced.
EINVAL	There was a parameter error. Verify the Mount_mode and File_system_type (specified in the MNTE data structure). Other reasons for this error include: • The mount point is a root of a file system. • The file system is already mounted. • parm_length is too long. • A mounted file system has a real or alias name that conflicts with this mount request. One of these situations occurred: – A file system was previously mounted using an alias data set name, and the corresponding real data set name conflicts with the file system name specified by this mount request. – The file system name specified on this mount request is an alias data set name that has a real data set name which conflicts with the name of a previously mounted file system. Resolve the duplicate file system names. Resolve the duplicate file system names and reissue the mount request. The following reason codes can accompany the return code if it is a nonprivileged user mount: JRNotSupportedForFileType, JRFileSystemMigrated, JRSecurityConflict, JRNoSetUID, JRSysNameNotAllowed, JRPFSNotSupported. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced, JRIsMountedRealName.

Return_code	Explanation
EIO	An I/O error occurred.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENOENT	The mount point does not exist. The following reason code can accompany the return code: JRMountPt.
ENOMEM	There is not enough storage space available to mount this file system.
ENOTDIR	The mount point is not a directory. The following reason code can accompany this return code: JRMountPt.
ENOTEMPTY	The mount point directory is not empty for the nonprivileged user mount.
EPERM	Insufficient authority to do the mount.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __mount service stores the reason code. The __mount service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The __mount service effectively creates a virtual file system. After a file system is mounted, references to the pathname that is mounted refer to the root directory on the mounted file system.
2. A file system can be mounted at only one point.
3. Parameter specifics for the HFS physical file system:
   - The File_system_name value must be uppercase, and must be the name of the data set.
4. The physical file system may complete the __mount operation asynchronously, which is indicated by a Return_value of 1. The w_getmntent callable service can then be used to determine if the file system has been mounted.
5. The MNTE eye-catcher must be set to MNT2. Additionally, the length of the body following the header must be set into MNTEHBLEN, the body length field in the header.
   - When setting character string values like SYSNAME or FROMSYSNAME, in general, these fields are set with the strings left justified and blank padded. Should data in these fields be unrequired or absent, the values should be set to nulls. Consult the data structure definition for specifics.
   - When requesting a change to an already mounted file systems, the MNTENTCHANGE bit must be set on. Additionally, when requesting that the AUTOMOVE setting be changed, the MNTENTNEWAUTO bit must be set on.
   - When requesting a mount of a file system, as opposed to a change, none of the MNTENTRFLAGS are expected to be set on.
   - When requesting that a collection of file systems be moved from one system to another, the following fields must be set: FROMSYS (to indicate where the

file systems are to be moved from); SYSNAME (to indicate where the file systems are to be relocated); and Rflags (to indicate that this is a change mount request). The other fields will be ignored.

- When requesting a single file system move, the mount point *or* the file system name must be specified. Do not specify both. Additionally, the name of the system that the file system should be moved to should be specified in SYSNAME. If you plan to change an AUTOMOVE setting, set the new value in the bit of the FSmode word. The Rflags setting will specify that MNTENTCHANGE=ON, which indicates that the change is a **chmount** request. You should set MNTENTNEWAUTO only if the request intends to change the AUTOMOVE setting to what is reflected in the MNTENTFSNOAUTOMOVE value.

  For more information about SYSNAME and AUTOMOVE, see Customizing BPXPRMxx for a shared file system in *z/OS UNIX System Services Planning*. The **chmount** command is explained in the **chmount** command description in*z/OS UNIX System Services Command Reference*.
- When requesting a mount on a system other than the one the mount command is executed on, the MNTENTSYSNAME field will denote the system that will "own" the file system. Fields that must be set to request a mount are: Filemode settings (such as read or write), FILESYSNAME, FILESYSTYPE, and pathname. Other fields that can be optionally set are: parameter string and systemname.

Related services

- "umount (BPX1UMT, BPX4UMT) — Remove a virtual file system" on page 867
- "w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems" on page 894

Characteristics and restrictions

1. In order to mount a file system, the caller must be an authorized program, or must be running for a user with mount authority. For more information about mount authorization, see *z/OS UNIX System Services Planning*.
2. A file system cannot be moved while it is being exported by the DFS server. It must first be unexported from DFS.
3. Typically, an EBUSY signal is returned when a file system is quiesced. In a sysplex, however, the __mount() syscall suspends until the file system becomes unquiesced.

Examples

For an example using this callable service, see "BPX2MNT (__mount) example" on page 1164.

mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping

Function

The mprotect callable service changes the access protection of a memory mapping for the caller's address space.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW Key 2 or PSW Key 8

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MPR):	31-bit
AMODE (BPX4MPR):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MPR,(Map_address,
              Map_length,
              Protect_options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MPR with the same parameters. The Map_address and Map_length parameters are doublewords.

Parameters

Map_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the starting address in the address space at which the access protection of the mapping is to be changed. The value of Map_address must be a multiple of the page size.

Map_length
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of the fullword (doubleword) that contains the size (in bytes) of the mapping that is to have its access protection modified. The length can be the size of the whole mapping, or a part of it. If the specified length is not in multiples of the page size, it is rounded up to a page boundary.

Protect_options
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the new value of the access protection flags for the specified mapping. The access protection flags can be changed to either PROT_NONE or a combination (for example, by using an inclusive OR)

of one or more of the other flags (such as PROT_READ, PROT_WRITE, or PROT_EXEC). These flags are defined in the BPXYCONS macro. (See "BPXYCONS — Constants used by services" on page 952.)

Constant	Description
PROT_READ	Mapped data can be read.
PROT_WRITE	Mapped data can be written and read.
PROT_EXEC	Mapped data can be executed. PROT_EXEC is treated in the same way as PROT_READ.
PROT_NONE	Mapped data cannot be accessed.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mprotect service returns the value of `0` if the request is successful, or -1 if it is not successful.

Upon successful completion, the mprotect service has changed the access protections on the mapping specified by the range (map_address, map_address + map_length) to those specified by the protect_options parameter.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mprotect service stores the return code. The mprotect service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

The mprotect service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The access protection value is incorrect; it violates the access permission of the process to the underlying file. The following condition occurred: • The original file is not open for write, and PROT_WRITE is specified for a MAP_SHARED type mapping.
EAGAIN	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)

Return_code	Explanation
EINVAL	One of the following conditions occurred: • The value of map_address is not a multiple of the page size. (JRNotPage) • The input address or length is negative. (JRNegativeValueInvalid) • The protection options specified are not valid. (JROptNotSupp) • The caller's PSW key does not match the key of the memory mapped storage segment that is being operated against. (JrKeyMismatch) • In 64-bit mode, the value of map_address specified was greater than X'7FFFFFFF' (JrAddressNotAvailable). • In 64-bit mode, the value of map+length was greater than X'7FFFFFFF' (Jr_InvParmLength).
ENOMEM	One of the following conditions occurred: • Addresses in the range (map_address, map_address + map_length) are not valid for the address space. (JRAddressNotAvailable) • One or more specified pages are not mapped. (JRNotMapped)

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mprotect service stores the reason code. The mprotect service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Access protection only acts on full pages. If the map_length parameter contains a value that is not a multiple of the page size, the length is rounded up to a full page.
2. The protection level that is established by the mprotect service is address-space wide in scope, not just process specific. The scope is system-wide when the protection is changed for a MAP_MEGA map. All active maps to the same file-offset range are affected by the request.
3. Constants used for this callable service are defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Related services

- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819
- "mmap (BPX1MMP, BPX4MMP) — Map pages of memory" on page 368
- "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403
- "munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses" on page 407

Characteristics and restrictions

The range specified (map_address, map_address + map_length) must not contain any areas that are not currently memory mapped. It may, however, contain areas that have been unmapped, in which case no action will be taken against the unmapped areas.

Examples

For an example using this callable service, see "BPX1MPR (mprotect) example" on page 1166.

msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations

Function

The msgctl service provides a variety of message control operations as specified by the Command parameter. These functions include reading and changing message variables within the MSQID_DS data structure, and removing a message queue from the system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QCT):	31-bit
AMODE (BPX4QCT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QCT,(Message_Queue_ID,
              Command,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QCT with the same parameters. The Buffer parameter is a doubleword.

Parameters

Message_Queue_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the message queue identifier.

Command

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates the message command to be executed. For the structure that contains these constants, see "BPXYIPCP — Map interprocess communication permissions" on page 987. The values for Command are:

Ipc_STAT
Obtain status information about the message queue that is identified by the Message_Queue_ID parameter, if the current process has read permission. This information is stored in the area that is pointed to by argument Buffer and mapped by area MSQID_DS data structure. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997, MSQID_DS DSECT.

Ipc_SET
Set the value of the IPC_UID, IPC_GID, IPC_MODE and MSG_QBYTES for associated Message_queue_ID. The values that are to be set are taken from the MSQID_DS data structure that is pointed to by argument Buffer. Any value for IPC_UID and IPC_GID may be specified. Only mode bits that are defined by msgctl under Message_Flag argument may be specified in the IPC_MODE field. This Command can only be executed by a task that has an effective user ID equal either to that of a task with appropriate privileges (see "Authorization" on page 8), or to the value of IPC_CUID or IPC_UID in the MSQID_DS data structure that is associated with Message_Queue_ID. This information is taken from the buffer that is pointed to by the Buffer parameter. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997, MSQID_DS DSECT.

Ipc_RMID
Remove the message identifier that is specified by Message_Queue_ID from the system, and destroy the message queue and MSQID_DS data structure that are associated with it. This Command can only be executed by a process that has an effective user ID equal either to that of a process with appropriate privileges (see "Authorization" on page 8), or to the value of IPC_CUID or IPC_UID in the MSQID_DS data structure that is associated with Message_Queue_ID.

Buffer

Parameter supplied and returned

Type: Address

Length:
Fullword (doubleword)

The name of the fullword (doubleword) that contains the address of the buffer into which or from which the message queue information will be copied. This buffer is mapped by MSQID_DS. (See "BPXYMSG — Map interprocess communication message queues" on page 997.)

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgctl service returns -1 or 0.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgctl service stores the return code. The msgctl service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The msgctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The command specified was Ipc_STAT, and the calling process does not have read permission. The following reason code can accompany the return code: JRIpcDenied.
EINVAL	One of the following occurred: • Message_Queue_ID is not a valid Message queue identifier. • The Command parameter is not a valid command. • The mode bits were not valid (SET). The following reason codes can accompany the return code: JRIpcBadFlags, JRMsqQBytes, or JRIpcBadID.
EPERM	One of the following occurred: • The command specified was Ipc_RMID or Ipc_SET. The effective user ID of the caller is not that of a process with appropriate privileges (see "Authorization" on page 8), and is not the value of IPC_CUID or IPC_UID in the MSQID_DS data structure that is associated with Message_Queue_ID. • The command specified was Ipc_SET, and an attempt is being made to increase MSG_QBYTES. The effective user ID of the caller does not have superuser privileges. The following reason codes can accompany the return code: JRIpcDenied or JRMsqQBytes.
EFAULT	The Buffer parameter specified an address that caused the syscall to program check. The following reason code can accompany the return code: JRBadAddress.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgctl service stores the reason code. The msgctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Changing the access permissions only affects message queue syscall requests that occur after the msgctl service has returned. The msgsnd and msgrcv services, which are waiting while the permission bits are changed by msgctl, are not affected.
2. Ipc_SET can change permissions, and may affect the ability of a thread to use the next message queue syscall.
3. Quiescing a message queue stops additional messages from being added, while allowing existing messages to be received. You can quiesce a message queue by clearing (Ipc_SET) write permission bits.
4. You can also quiesce a message queue by reducing MSG_QBYTES (Ipc_SET) to zero. (Note: It would take a superuser to re-raise the limit.) Requesters are told EAGAIN or wait.
5. When a message queue ID is removed (Ipc_RMID) from the system, all waiting threads regain control with RV=-1, RC=EIDRM, and RC=JRIpcRemoved.
6. If you do not wish to change all the fields, first initialize (Ipc_STAT) the buffer, change the desired fields, and then make the change (Ipc_SET).
7. For Command Ipc_RMID, the remove is complete by the time control returns to the caller.

Related services

- "msgget (BPX1QGT, BPX4QGT) — Create or find a message queue"
- "msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue" on page 395
- "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399

Characteristics and restrictions

The invoker is restricted by ownership, read and read-write permissions defined by msgget and msgctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1QCT (msgctl) example" on page 1176.

msgget (BPX1QGT, BPX4QGT) — Create or find a message queue

Function

The msgget function returns a message queue ID that it created or that the user is allowed to access.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QGT):	31-bit
AMODE (BPX4QGT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QGT,(Key,
              Message_Flag,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QGT with the same parameters.

Parameters

Key

Supplied parameter

Type: Integer

Length:
Fullword

Identification for this message queue. This can be a user-defined value that serves as a lookup value to determine if this message queue already exists, or the reserved value Ipc_PRIVATE.

Message_Flag

Supplied parameter

Type: Integer

Length:
Fullword

Valid values for this field include any combination of the following (additional bits cause an EINVAL):

Ipc_CREAT
Creates a message queue if the key that is specified does not already have an associated ID. Ipc_CREATE is ignored when Ipc_PRIVATE is specified.

Ipc_EXCL
Causes the msgget function to fail if the key that is specified has an associated ID. Ipc_EXCL is ignored when Ipc_CREAT is not specified, or when Ipc_PRIVATE is specified.

Ipc_RcvTypePID
Creates a message queue that can only be read from (by the msgrcv service) when Message_Type is the process ID of the invoker. This restriction does not apply if the caller of the msgrcv service has the same effective UID as the creator of the message queue.

Ipc_SndTypePID
Creates a message queue that can only be written to (by the msgsnd service) when Message_Type is the process ID of the invoker. This restriction does not apply if the caller of the msgsnd service has the same effective UID as the creator of the message queue.

Ipc_PLO1
Use PLO for serialization.

Ipc_PLO2
Use PLO if practical.

S_IRUSR
Permits the process that owns the message queue to read it.

S_IWUSR
Permits the process that owns the message queue to alter it.

S_IRGRP
Permits the group that is associated with the message queue to read it.

S_IWGRP
Permits the group that is associated with the message queue to alter it.

S_IROTH
Permits others to read the message queue.

S_IWOTH
Permits others to alter the message queue.

The values that begin with an "Ipc_" prefix are defined in BPXYIPCP, and are mapped onto S_TYPE, which is in BPXYMODE.

The values that begin with an "S_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgget service returns -1 or the message queue identifier.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgget service stores the return code. The msgget service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The msgget service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	A message queue identifier exists for the Key parameter, but operation permission, as specified by the low-order 9– bits of the Message_Flag parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRIpcDenied.
EEXIST	A message queue identifier exists for the Key parameter, and both Ipc_CREAT and Ipc_EXCL are specified. The following reason code can accompany the return code: JRIpcExists.

Return_code	Explanation
EINVAL	The Message_Flag operand included bits that are not supported by this function. The following reason code can accompany the return code: JRIpcBadFlags.
ENOENT	A message queue identifier does not exist for the Key parameter, and Ipc_CREAT was not set. The following reason code can accompany the return code: JRIpcNoExist.
ENOSPC	The system limit on the number of message queue IDs has been reached. The following reason code can accompany the return code: JRIpcMaxIDs.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgget service stores the reason code. The msgget service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. As long as a thread knows the message queue ID, it may issue a msgctl, msgsnd, or msgrcv request (msgget is not needed).
2. This function returns the message queue identifier that is associated with the Key parameter.
3. This function creates a data structure that is defined by MSQID_DS if one of the following is true:
   - The Key parameter is equal to Ipc_PRIVATE.
   - The Key parameter does not already have a message queue identifier associated with it, and Ipc_CREAT is set.
4. Upon creation, the data structure that is associated with the new message queue identifier is initialized as follows:
   - Ipc_CUID and Ipc_UID are set to the effective user ID of the calling task.
   - Ipc_CGID and Ipc_GID are set to the effective group ID of the calling task.
   - The low-order 9-bits of Ipc_MODE are equal to the low-order 9-bits of the Message_Flag parameter.
   - MSG_QBYTES is set to the system limit that is defined by parmlib.
5. The message queue is removed from the system when msgctl is called with command Ipc_RMID.
6. Users of message queues are responsible for removing them when they are no longer needed. Failure to do so ties up system resources.
7. In a client/server environment, two message queues could be used: one inbound to the server, created with Ipc_SndTypePID, and the other outbound from the server, created with Ipc_RcvTypePID. This arrangement guarantees that the server knows the process ID of the client and that the client is the only process that receives the server's returned message. The server could call the msgrcv service with `PID=0` to see if there are any messages that belong to process IDs that have gone away.

8. Message_Flags Ipc_PLO1 and Ipc_PLO2 are ignored if the PLO (Perform Lock Operation) instruction is not present on the hardware. (See SCCBPLO in IHASCCB and the Ipc_PLOInUse bit in the S_MODE byte returned with w_getipc.)
9. Performance of the PLO instruction for serialization varies with the msgrcv() type, the number of messages on the queue, and the number of tasks that are doing msgsnd and msgrcv requests. A msgrcv request with a message type that is less than zero and that has long message queues is expected to be a poor performer. A msgrcv request with a message type that is greater than zero is expected to be an equivalent or good performer. A msgrcv request with a message type equal to zero is expected to be a very good performer.
10. Message queues that are created with Ipc_RcvTypePID, Ipc_SndTypePID, Ipc_PLO1 and Ipc_PLO2 show these bits, and may show the Ipc_PLOInUse bit in the S_MODE byte that is returned with the w_getipc request.
11. Message queue PLO serialization is not compatible with the use of select() for message queues. When the msgrcv service detects a select() for a message queue, serialization is changed to use traditional latches.
12. Performance runs should be made with Ipc_PLO1, because Ipc_PLO2 could switch to latch serialization, and the user would not be aware of this. Upon the first msgrcv() with a message type that is less than zero, the message queue will attempt to switch to latch serialization.

Related services

- "msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations" on page 388
- "msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue"
- "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399
- "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890

Characteristics and restrictions

1. There is a maximum number of message queues that are allowed in the system.
2. The invoker is restricted by ownership, read, and read-write permissions that are defined by msgget and msgctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1QGT (msgget) example" on page 1177.

msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue

Function

The msgrcv service receives messages from a message queue.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QRC):	31-bit

Operation	Environment
AMODE (BPX4QRC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QRC,(Message_Queue_ID,
              Message_Address,
              Message_Alet,
              Message_Length,
              Message_Type,
              Message_Flag,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QRC with the same parameters. The Message_Address and Message_Type parameters are doublewords.

Parameters

Message_Queue_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the message queue identifier.

Message_Address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of a buffer that is mapped by MSGBUF or MSGXBUF (see "BPXYMSG — Map interprocess communication message queues" on page 997).

Message_Alet
Supplied parameter

Type: Address

Length:
Fullword

The name of the fullword that contains the ALET for Message_Address, which identifies the address space or data space where the buffer resides.

You should specify a Message_Alet of 0 if the buffer is in the user's address space (current primary address space).

You should specify a Message_Alet of 2 if the buffer resides in the home address space.

If a value other than 0 or 2 is specified for the Message_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Message_Length
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the length of the message text that is to be placed in the buffer that is pointed to by Message_Address parameter.

In 31-bit mode, if Msg_Info is specified, this buffer is 20 bytes longer than Message_Length; otherwise this buffer is 4 bytes longer than Message_Length. In 64-bit mode, if Msg_Info is specified, this buffer is 28 bytes longer than Message_Length; otherwise this buffer is 8 bytes longer than Message_Length.

The message that is received may be truncated (see MSG_NOERROR of Message_Flag). A value of zero with MSG_NOERROR is useful for receiving the message type without the message text.

Message_Type
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

Specifies the type of message requested, as follows:

- If Message_Type is equal to zero, the first message on the queue is received.
- If Message_Type is greater than zero, the first message of Message_Type is received.
- If Message_Type is less than zero, the first message of the lowest type that is less than or equal to the absolute value of Message_Type is received.

Message_Flag
Supplied parameter

Type: Integer

Length:
Fullword

MSG_NOERROR specifies that the received message is to be truncated to Message_Length (mapped in BPXYMSG). The truncated part of the message is lost, and no indication of the truncation is given to the caller.

MSG_INFO specifies that the received message is to be of the MSGXBUF and not the MSGBUF format, mapped in BPXYMSG. MSG_INFO specifies that extended information is to be received. This is similar to the msgxrcv() C language function.

Ipc_NOWAIT specifies the action that is to be taken if a message of the desired type is not on the queue, as follows:

- If Ipc_NOWAIT is specified, the caller is to return immediately with an error (ENOMSG).
- If Ipc_NOWAIT is not specified, the calling thread is to suspend execution until one of the following occurs:
  - A message of the desired type is placed on the queue.

- The message queue is removed from the system (EIDRM).
- The caller receives a signal (EINTR).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgrcv service returns -1, or the number of MSG_MTEXT bytes returned.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgrcv service stores the return code. The msgrcv service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The msgrcv service can return one of the following values in the Return_code parameter:

Return_code	Explanation
E2BIG	MSG_MTEXT is greater than Message_Length, and MSG_NOERROR is not set. The following reason code can accompany the return code: JRMsq2Big.
EACCES	Operation permission is denied to the calling task: JRIpcDenied. If the message queue was built with the Ipc_RcvTypePID, and the MSG_TYPE was other than the invoker's process ID, the following reason code accompanies the return code: JRTypeNotPID.
EIDRM	The Message_Queue_ID was removed from the system while the invoker was waiting. The following reason code can accompany the return code: JRIpcRemoved.
EINTR	The function was interrupted by a signal. The following reason code can accompany the return code: JRIpcSignaled.
EINVAL	Message_Queue_ID is not a valid message queue identifier; or the Message_Length parameter is less than 0. The following reason codes can accompany the return code: JRIpcBadID or JRMsqBadSize.
EFAULT	The Message_Address parameter specified an address that caused the syscall to program check. The following reason code can accompany the return code: JRBadAddress.
ENOMSG	The queue does not contain a message of the desired type, and Ipc_NOWAIT is set. The following reason code can accompany the return code: JRMsqNoMsg.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgrcv service stores the reason code. The msgrcv service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Within the type specifications, the longest waiting thread will be reactivated first (FIFO). For example, if there are two threads waiting on message type 3 and one thread waiting on message type 2, when a message send for type 3 occurs, the oldest waiter for message type 3 receive is posted first.
2. Read access to the specified message queue is required.

Related services

- "msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations" on page 388
- "msgget (BPX1QGT, BPX4QGT) — Create or find a message queue" on page 391
- "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue"

Characteristics and restrictions

The caller of the msgrcv service is restricted by ownership, read, and read-write permissions that are defined by msgget and msgctl Ipc_SET.

Examples

See "BPX1QRC (msgrcv) example" on page 1177 for an example using this callable service.

msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue

Function

The msgsnd service sends a message to a message queue.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1QSN):	31-bit task or SRB mode
AMODE (BPX4QSN):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QSN,(Message_Queue_ID,
              Message_address,
              Message_Alet,
              Message_Size,
              Message_Flag,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QSN with the same parameters. The Message_address parameter is a doubleword.

Parameters

Message_Queue_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the message queue identifier.

Message_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the message to be sent. This area is mapped by MSGBUG of MSGXBuf. The message type is the first word of the message. It must be greater than zero.

Message_Alet
Supplied parameter

Type: Address

Length:
Fullword

The name of the fullword that contains the ALET for Message_address that identifies the address space or data space where the buffer resides.

You should specify a Message_Alet of 0 if the buffer resides in the user's address space (current primary address space).

You should specify a Message_Alet of 2 if the buffer resides in the home address space.

If a value other than 0 or 2 is specified for the Message_Alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Message_Size
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the length of the message text that is pointed to by the Message_address parameter. The length does not include the 4-byte type that precedes the message text. For example, a message with a MSG_TYPE and no MSG_MTEXT would have a Message_Size of zero. A zero-length message is accepted.

Message_Flag
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the action that is to be taken if one or more of these conditions are true:

- Placing the message on the message queue would cause the current number of bytes on the message queue (msg_cbytes) to be greater than the maximum number of bytes that are allowed on the message queue (msg_qbytes).
- The total number of messages on the message queue (msg_qnum) is equal to the system-imposed limit.

The actions to be taken are as follows:

- If Ipc_NOWAIT is specified, the caller returns immediately with an error (EAGAIN).
- If Ipc_NOWAIT is not specified, the calling thread suspends execution until one of the following occurs:
  - The message is sent.
  - The message queue is removed from the system (EIDRM).
  - The caller receives a signal (EINTR).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgsnd service returns `-1` or `0`. The message was sent unless a -1 is received.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgsnd service stores the return code. The msgsnd service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The msgsnd service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Operation permission is denied to the calling task: JRIpcDenied. If the message queue was built with Ipc_SndTypePID, and the MSG_TYPE was other than the invoker's process ID, the following reason code accompanies the return code: JRTypeNotPID.
EAGAIN	The message cannot be sent, and Message_Flag is set to Ipc_NOWAIT. The following reason codes can accompany the return code: JRMsqQueueFullMessages, JRMsqQueueFullBytes.
EIDRM	The Message_Queue_ID was removed from the system while the caller was waiting. The following reason code can accompany the return code: JRIpcRemoved.
EINTR	The function was interrupted by a signal, and the message was not sent. The following reason code can accompany the return code: JRIpcSignaled.
EINVAL	Message_Queue_ID is not a valid message queue identifier; the value of MSG _TYPE is less than 1; or the value of Message_Size is less than zero or greater than the system—imposed limit. The following reason codes can accompany the return code: JRIpcBadID, JRMsqBadSize, or JRMsqBadType.
EFAULT	The Message_address parameter specified an address that caused the service to program check. The following reason code can accompany the return code: JRBadAddress.
ENOMEM	There were not enough system storage exits to send the message; the message was not sent. The following reason code can accompany the return code: JrSmNoStorage.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msgsnd service stores the reason code. The msgsnd service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

- Write access to the specified message queue is required.
- See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations" on page 388
- "msgget (BPX1QGT, BPX4QGT) — Create or find a message queue" on page 391
- "msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue" on page 395

Characteristics and restrictions

The caller of this service is restricted by ownership and read and read-write permissions that are defined by msgget and msgctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1QSN (msgsnd) example" on page 1178.

msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage

Function

The msync callable service writes all modified pages over the requested range to their permanent storage locations on disk. It also deletes any in-memory cached pages over the requested range, resetting the contents of those pages to that which resides on disk.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW Key 2 or PSW Key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MSY):	31-bit
AMODE (BPX4MSY):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MSY,(Map_address,
              Map_length,
              Sync_Options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MSY with the same parameters. The Map_address and Map_length parameters are doublewords.

Parameters

Map_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the mapping from which the modified pages are to be written to their permanent storage locations on disk, or invalidated. The value of map_address must be a multiple of the page size.

Map_length
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of the fullword (doubleword) that contains the size (in bytes) of the mapping that is to have all updated pages written out to disk, or invalidated. The length can be the size of the whole mapping, or a part of it. If the specified length is not a multiple of the page size, it is rounded up to a page boundary.

Sync_Options
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the option flags for the service. The specified value can be a combination (for example, using an exclusive OR) of one or more of the following flags, with the limitation that MS_ASYNC and MS_SYNC are mutually exclusive. These constants are defined in the BPXYCONS macro.

Constant	Description
MS_ASYNC	Performs asynchronous writes. MS_ASYNC returns immediately when all write operations are scheduled. If the requestor's intent is to write consistent data to the disk, do not use this option.
MS_SYNC	Performs synchronous writes. MS_SYNC will return after all write operations are completed.
MS_INVALIDATE	Invalidates the cached memory—mapped pages. After the cached copy of the data in memory has been invalidated for a MAP_SHARED mapping, any further references to these pages will be obtained by the system from their permanent storage locations on disk. For a MAP_PRIVATE mapping, only updated (private) pages are invalidated. Any further references to these pages will be obtained from the shared cache.

Note:

1. If MS_INVALIDATE is the only flag specified, the requested cached memory—mapped pages are invalidated without any modified pages first being written to disk.
2. If MS_INVALIDATE is specified with either MS_SYNC or MS_ASYNC, all the modified pages in the requested address range are written to disk before the cached copy of data in memory is invalidated.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msync service returns the value of `0` if the request is successful, or -1 if it is not successful.

Upon successful completion, the msync service writes all modified pages over the range (map_address, map_address+map_length) to their permanent storage locations on disk, invalidates the cached mmap pages, or does both.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msync service stores the return code. The msync service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

The msync service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)
EINVAL	One of the following conditions occurred: • The value of map_address is not a multiple of the page size. (JRNotPage) • The value in the Sync_Options parameter is incorrect. (JROPtNotSupp) • The input address or length is negative. (JRNegativeValueInvalid) • The caller's PSW key does not match the key of the memory mapped storage segment that is being operated against. (JrKeyMismatch) • In 64-bit mode, an address greater than 31 bit addr was passed in map_address (JrAddressNotAvailable). • In 64-bit mode, a length greater than X'7FFFFFFF' was passed in map_length (JrInvParmLength).
EIO	An I/O error occurred while writing to the file system (file system JR). This return code is set only if MS_SYNC is set in the Sync_Options parameter. I/O errors during asynchronous write operations are not reported to the application.
ENOMEM	One of the following conditions occurred: • Some or all of the addresses in the range (map_address, map_address + map_length) are not valid for the address space. (JRAddressNotAvailable) • One or more specified pages are not mapped. (JRNotMapped)

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the msync service stores the reason code. The msync service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The msync service is used by programs that require a file to be in a known state (such as in building transaction-oriented programs).
2. When a request is made to write the cached pages by a process that has mapped the area with the MAP_SHARED option, updates made by all processes sharing the specified file-offset range are written, not just the updates made by the msync requesting process. The same is true for invalidate requests.
3. Only full pages are processed. If the map_length parameter contains a value that is not a multiple of the page size, the length will be rounded up to a full page.
4. In relation to advisory locking mechanisms, there is no difference between sharing a file using the mmap services, and sharing a file using the read/write services. Specifically, before a series of bytes are accessed using either method, a byte range lock is required to ensure the consistency of the data being accessed. It logically follows that if the intent is to write consistent data to the disk when a file is shared using memory map services, an advisory lock should be held on the pages being acted upon, before calling the msync service (with the MS_SYNC option).
5. Constants used for this callable service are defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Related services

- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819
- "mmap (BPX1MMP, BPX4MMP) — Map pages of memory" on page 368
- "mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping" on page 384
- "munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses" on page 407

Characteristics and restrictions

1. The range that is specified (map_address, map_address + map_length) must not contain any areas that are not currently memory mapped. It may, however, contain areas that have been unmapped, in which case no action is taken against the unmapped areas.
2. To successfully write or invalidate MAP_SHARED mappings, the range that is specified must have the PROT_WRITE access level. If any portion of the specified range has either the PROT_NONE or PROT_READ access levels at the time of the msync request, that portion will not be written or invalidated, and no error condition will be raised.
3. Because memory map is implemented using a cached copy of the original data that resides on disk, concurrent updates made using the write callable service to a file that is being memory mapped will produce undefined results. If this type of activity is desired, explicit serialization must be implemented between a process invoking the msync service with the invalidate option, and another process invoking the write service (page-multiple advisory lock).
4. When the msync service is called for MAP_PRIVATE mappings, any data that is modified by that process is not written to the file, and such data is not visible to other processes. The only supported action is to invalidate the pages that were cached exclusively for the use of the requesting process (this has no impact on the MAP_SHARED cache). For the invalidate request to be successful, the range that is specified must have the PROT_WRITE access level. If another process mapping the same file-offset range with the MAP_SHARED

option invalidates the shared cache, then, from the perspective of the MAP_PRIVATE process, only the pages that were not updated by the MAP_PRIVATE process (still shared) are invalidated. The modified (and now private) pages remain intact in the cache. This type of activity could cause inconsistencies within the MAP_PRIVATE mapping.

Examples

For an example using this callable service, see "BPX1MSY (msync) example" on page 1167.

munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses

Function

The munmap callable service removes the mapping for pages in the requested range. It should be used only to unmap regions that have been previously mapped by the application with the mmap callable service.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW Key 2 or PSW Key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MUN):	31-bit
AMODE (BPX4MUN):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MUN,(Map_address,
              Map_length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MUN with the same parameters. The Map_address and Map_length parameters are doublewords.

Parameters

Map_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of an existing mapping that is to be unmapped. The specified address does not have to be

the start of a mapping. The value of map_address must be a multiple of the page size. If the address falls within a MAP_MEGA map, the address that is provided is rounded down to a megabyte multiple so that an entire segment is included in the unmap operation. It is not possible to unmap part of a segment when processing a MAP_MEGA map.

Map_length
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of the fullword (doubleword) containing the size (in bytes) of the mappings that are to be unmapped. The length can be the size of the whole mapping, or a part of it. If the specified length is not in multiples of the page size, it will be rounded up to a page boundary. If map_address plus map_length falls within a MAP_MEGA map, the length is rounded up so that it includes an entire segment (but not necessarily the entire MAP_MEGA mapping).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the munmap service returns the value of `0` if the request is successful, or -1 if it is not successful.

Upon successful completion, the munmap service unmaps all pages in the range (map_address, map_address+map_length).

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the munmap service stores the return code. The munmap service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

The munmap service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The caller is not running in either PSW Key 2 or PSW Key 8. (JRUnsupportedKey)

Return_code	Explanation
EINVAL	One of the following conditions occurred: • The value of map_address is not multiples of the page size. (JRNotPage) • Some or all of the addresses in the range (map_address, map_address + map_length) are not valid for the address space. (JRAddressNotAvailable) • The input address is negative, or the input length is zero or negative. (JRZeroOrNegative) • The caller's PSW key does not match the key of the memory mapped storage segment that is being unmapped. (JrKeyMismatch) • In 64-bit mode, an address greater than 31 bit addr was passed in map_address (JrAddressNotAvailable). • In 64-bit mode, a length greater than X'7FFFFFFF' was passed in map_length (JrInvParmLength).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the munmap service stores the reason code. The munmap service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. An address of 0 with a length of X'7FFFF000' unmaps all the storage that is associated with this process.
2. For both private and shared mappings, if the munmap service unmaps a subset of the range of the original mmap request, further references to those pages result in a program check exception. When the entire range of the original mmap request has been unmapped, the memory allocated by the mmap service is freed.
3. If there are no mappings in the requested address range, the munmap service has no effect. The service returns successfully.
4. The range that is specified (map_address, map_address + map_length) may contain areas that have been unmapped, in which case no action is taken against the unmapped areas.
5. If a mapping to be removed is private, any modifications that are made in the specified address range are discarded.
6. If a mapping to be removed is shared, all modifications that are made in the specified address range since the last msync (if any) are written to disk. If this is not desired, the msync service must be called to invalidate the updates that have been made to the mapped region before the range is unmapped.
7. If a memory-mapped region is not unmapped before the process terminates, process termination does not automatically write out to disk any modified data in the mapped region. Modified private data in a MAP_PRIVATE region is discarded. If the mapped region is MAP_SHARED, the modified data continues to reside in the cache (if the same file-offset range is being shared), and may ultimately be written out to disk by another process via the msync service.

However, if no other processes map the same file-offset range as MAP_SHARED, the modified data is discarded.

8. Only entire pages are unmapped. If the map_length parameter contains a value that is not a multiple of the page size, the length is rounded up to a full page. For MAP_MEGA maps, only entire segments are unmapped. The map_address and map_length are adjusted to ensure that entire segments are unmapped.
9. An unmap request may span MAP_MEGA and non-MAP_MEGA ranges.

Related services

- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819
- "mmap (BPX1MMP, BPX4MMP) — Map pages of memory" on page 368
- "mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping" on page 384
- "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403

Characteristics and restrictions

There are no restrictions on the use of the munmap service.

Examples

For an example using this callable service, see "BPX1MUN (munmap) example" on page 1167.

mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread

Function

The mvsiptaffinity callable service allows a task created with pthread_create to request that a user-defined assembler routine run on its initial pthread-creating thread (IPT). The requesting pthread is blocked until the requested routine has been executed.

This service manages MVS resources under the IPT, instead of under the task created with pthread_create. Some resources that can be managed with this service are:

- Load modules
- Opened data sets
- Other MVS resources with task affinity

Requirements

Operation	**Environment**
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1IPT):	31-bit
AMODE (BPX4IPT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1IPT,(Routine_address,
              Parameter_list,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4IPT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine control is passed to on the pthread's IPT. The requesting pthread is responsible for ensuring that the routine to be run is in memory when it is called and remains there until the call is complete.

Parameter_list
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine parameter list. The value in this fullword is passed in register 1 when the specified routine receives control. If the routine does not require parameters, specify `0`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvsiptaffinity service returns -1 if it is not successful. If it is successful, the mvsiptaffinity service returns `0`.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvsiptaffinity service stores the return code. The mvsiptaffinity service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The mvsiptaffinity service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	A bad address was received as an argument of the call, or the specified routine experienced an abend or program check that was not handled by the routines recovery. The following reason codes can accompany the return code: JRBadAddress and JRRoutineError.
EAGAIN	Another pthread within the process has this call pending. At most one pthread can request this service at a time. The requesting task can try again later when the current pending call is complete.
EACCES	A task other than a pthread-created task is not permitted to perform this service.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the mvsiptaffinity service stores the reason code. The mvsiptaffinity service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The specified routine to be executed receives control with the following attributes:
   - Problem program state
   - Key of the IPT task
   - AMODE 31(64), according to the AMODE of the caller
   - Primary ASC mode
2. The register usage on entry to the specified routine is:
   - R0: Undefined
   - R1: Address of Parameter_list, as specified by the caller of the mvsiptaffinity service
   - R2–R12: Undefined
   - R13: Address of a 72-byte work area with which the routine gains control. For AMODE 64 callers, the work area is 144 bytes long.
   - R14: The return address from the specified routine to the mvsiptaffinity service. This address *must* be preserved by the invoked routine.
   - R15: Address of the invoked routine
3. Only tasks created with pthread_create can invoke this service. If a task that was created using MVS non-POSIX interfaces requests this service, or if it is an IPT itself, it receives an EACCES return code.
4. At most one pthread can have this service request pending at a time. If a pthread already has this service pending, when another pthread requests this

service, the last pthread receives an EAGAIN return code. It is the caller's responsibility to serialize the invocation of mvsiptaffinity, or contain retry logic if the EAGAIN return code is obtained.

5. The EXITRTN assembler routine cannot issue callable services after it gains control under the IPT.
6. The specified routine can establish its own recovery environment. However, even if recovery is not established, the mvsiptaffinity service establishes its own recovery environment while running under the IPT. For all recoverable errors, this recovery routine retries, returning the EFAULT return code to the requestor. It also ensures that any recovery routine established by the IPT itself is not entered unexpectedly.

Related services

There are no related services.

Characteristics and restrictions

There are no restrictions on the use of the mvsiptaffinity service.

Examples

For an example using this callable service, see “BPX1IPT (mvsiptaffinity) example” on page 1154.

mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals

Function

The mvspause callable service allows a thread to suspend until a signal arrives or some application-defined event is posted.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task - No EUT FRRs
Cross memory mode:	PASN = HASN
AMODE (BPX1MP):	31-bit
AMODE (BPX4MP):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MP,( Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MP.

Parameters

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a 0 indicating that an event occurred, or -1 otherwise.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvspause service stores the return code. The mvspause service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The mvspause service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	Error addressing parameters. The parameters on the prior mvspauseinit call were not fully validated at mvspauseinit time. The following reason code unique to the mvspause service can accompany the return code: JRECBStateBad.
EINTR	The mvspause call was interrupted by a signal.
EMVSPARM	Incorrect parameters were passed to an MVS service. The following reason codes unique to the mvspause service can accompany the return code: JRECBListNotSetup, JRECBStateBad.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The mvspause service stores a reason code only when the return value is -1. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The intended use of mvspause is for a program to wait on user ECBs plus signals.
2. The user must first invoke the mvspauseinit service to declare to the system the list of ECB pointers to use. The system makes a copy of the list of ECB pointers to wait on and uses the existing MVS WAIT service to wait for the user events and the signal event.
3. When one of the ECBs in the ECB list has been posted or a signal is received, the mvspause operation concludes and control is returned to the caller. When a signal is received, the mvspause service posts the signal ECB and runs the signal handler before returning to the user.

4. The user has the option of reinvoking the mvspause service without reinvoking the mvspauseinit service. The user should be aware, however, that the system has made a copy of the list of pointers that point to the user's ECBs. Any changes to the caller's copy of the ECB pointer array are not reflected in the system copy unless the mvspauseinit service is invoked again. Furthermore, when the user wishes to reinvoke the mvspause without reinvoking the mvspauseinit service, the user must clear all ECBs that were posted. This includes clearing the signal ECB. If the user does not clear posted ECBs, the mvspause detects the already posted ECB and returns immediately. The user must take care when clearing ECBs, because not all ECBs may have been posted. Asynchronous operations could post an ECB at any time during the user's processing. The user should clear only ECBs that are processed, and not blindly clear all ECBs.

   The following logic example displays one method for processing ECBs:

```
CALL MVSpauseInit(list of ECB addresses);

for(;;)               /* Do forever */
 {
  call MVSpause()
  for(i=1;i<=MaxEcbs;i++)
   {
    Copy value of Ecb(i)
    if (Ecb(i) was posted)
     {
      Clear Ecb(i)
      switch(i)
       {
        case 1:    CALL Signal-Occurred;
            break;
        case 2:    CALL Rtn_for_2nd_Ecb;
            break;
        case 3:    CALL Rtn_for_3rd_Ecb;
            break;
          ...      /* As many as are needed */
        default: no ECBs POSTed
       }               /* end switch */

     }                 /* end if */
   }                   /* end for */
   if (terminating condition occurred)
      break;           /* Exit Do Forever loop */
  }                    /* end do forever */
```

Related services

- "mvspauseinit (BPX1MPI, BPX4MPI) — Set up to wait on user events plus signals" on page 416

Characteristics and restrictions

There are no restrictions on the use of the mvspause service.

Examples

See "BPX1MP (mvspause) example" on page 1164 for an example using this callable service.

mvspauseinit (BPX1MPI, BPX4MPI) — Set up to wait on user events plus signals

Function

The mvspauseinit callable service allows the thread to declare to the system a list of event control blocks (ECBs) the application program will use to receive event notifications. This service is used in conjunction with the mvspause service.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task - No EUT FRRs
Cross memory mode:	PASN = HASN
AMODE (BPX1MPI):	31-bit
AMODE (BPX4MPI):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MPI,( Addr_of_ECBlist,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4MPI with the same parameters. The Addr_of_ECBlist parameter is a doubleword.

Parameters

Addr_of_ECBlist
Address of a list of up to 1018 user-defined event control blocks (ECBs). The system uses the first ECB in the list.

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) from which the service extracts the address of the input ECB list. The mvspauseinit service requires this list to contain a maximum of 1018 ECBs, with the first ECB dedicated to the system. The user is responsible for obtaining the storage for all ECBs.

All pointers in the ECB list are 32-bit pointers for both AMODE 31 and AMODE 64 callers, as ECBs are only supported below the 2 GB bar. The last ECB pointer in the list must have the high-order bit set to 1 (80000000x). This bit indicates that it is the last ECB address in the list.

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns a `0` upon normal completion, or -1 otherwise.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvspauseinit service stores the return code. The mvspauseinit service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The mvspauseinit service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	Error addressing parameters. The following reason codes unique to the mvspauseinit service can accompany the return code: JRECBError, JRECBListBad.

Reason_code
Returned parameter

Type: Integer

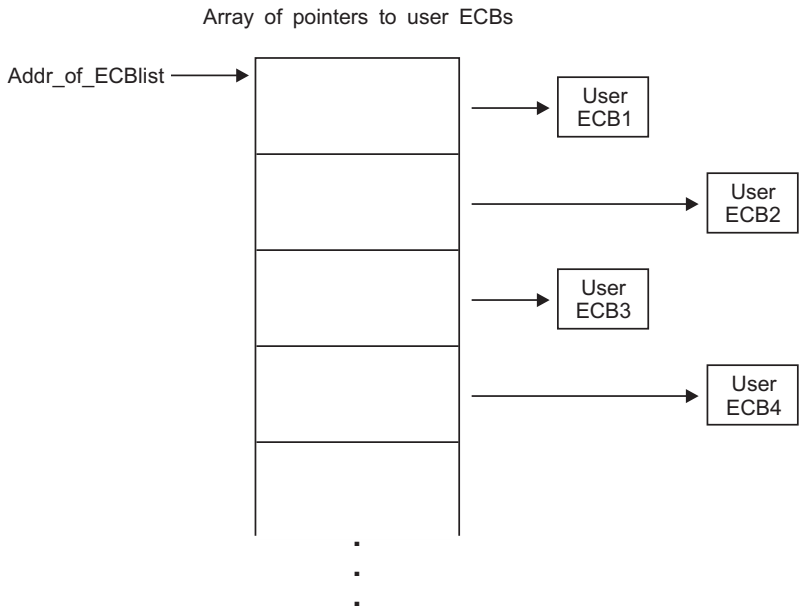
Length:
Fullword

The name of a fullword in which the service routine stores the reason code. The reason code further qualifies the return code value. The mvspauseinit service stores a reason code only when the return value is -1. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The intended use of the mvspauseinit service is for a program to declare to the system a list of pointers to user ECBs in user-managed storage. These ECBs are used by the mvspause function to suspend the thread until a signal arrives or a user-defined event is posted.
2. The user is responsible for initializing all ECBs, including the signal ECB. The first ECB is the signal ECB. The system does not alter the ECBs during mvspauseinit. This means that an asynchronous operation may post an ECB in the ECB list while mvspauseinit is operating.
3. After mvspauseinit returns to the caller, the mvspause service may be invoked as many times as necessary without reinvoking the mvspauseinit service. If the application program needs to change one or more ECB addresses, the application must reinvoke the mvspauseinit service before invoking the mvspause service.

   Note: Only one ECB list is allowed per thread. If a user invokes the mvspauseinit service multiple times, each invocation replaces the ECB list specified on previous invocations of mvspauseinit.



Related services

- "mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals" on page 413

Characteristics and restrictions

There are no restrictions on the use of the mvspauseinit service.

Examples

For an example using this callable service, see "BPX1MPI (mvspauseinit) example" on page 1165.

mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources

Function

The mvsprocclp callable service cleans up the z/OS UNIX-related resources for an entire process or on a thread-by-thread basis. After cleaning up resources, the mvsprocclp service terminates the thread or the entire process with the final thread.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MPC):	31-bit
AMODE (BPX4MPC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MPC,(Status_field,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MPC with the same parameters.

Parameters

Status_field
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword status field. The status field is a one-word area that is mapped by BPXYWAST, the WAIT status word. The WAST area should be initialized to zero. If the caller wants to set a specific exit status, then either WASTEXITCODE or WASTSIGTERM should be set. If the invocation of this service causes a full process cleanup to occur and the contents of the status field conform to the allowable exit status values, the contents are made available to the parent when the wait service is issued. For the mapping of the status field and a description of the allowable exit status values see "BPXYWAST — Map the wait status word" on page 1069.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the mvsprocclp service returns one of the following values:

Value	Explanation
0	Thread-related resources were cleaned up for the calling thread.
1	Process-related resources were cleaned up for the calling process.
-1	The service failed to clean up process resources.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvsprocclp service stores the return code. The mvsprocclp service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The mvsprocclp service can return the following value in the Return_code parameter:

Return_code	Explanation
EMVSERR	The specified terminating status value did not conform to the allowable values. The call failed with a Return_value of -1 and a Reason_Code of JrInvTermStat.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the mvsprocclp service stores the reason code. The mvsprocclp service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The mvsprocclp service normally cleans up only the thread-related data for the calling thread. The two following situations, however, cause full process cleanup to occur:
   - If the call is made from the initial thread of the process and no other threads exist in the process.
   - If the call is made from the last thread that is left in the process and that thread is not the initial thread, and the initial thread has not performed any z/OS UNIX system calls.

   In these two cases, both the thread-related and process-related resources are cleaned up and z/OS UNIX process termination is performed. See the _exit service for a description of z/OS UNIX process ending.
2. An important distinction between this service and the _exit service is that this service does not actually cause the user's tasks to end. The _exit service has the added effect of causing a full MVS-like ending, in that all the tasks in the executing process are ended. The mvsprocclp service cleans up only the process-related resources and causes a process termination to occur, leaving the other MVS-related resources in the address space unaffected.
3. The mvsprocclp service does not trigger a core dump when the dump flag is *on* in the status word.
4. For message queues, each thread is removed from the send and receive waiting chains (the message to be sent is lost). End of memory may require the message queue to be rebuilt.
5. When shared memory is being used, each process is terminated and the shared memory segment attached to the terminating process is detached. If the last attachment is removed and a shmctl RMID had been issued, the segment is removed from the system.
6. If semaphores are being used, each thread is removed from the waiting chain. The adjustment values are associated with the process, not the thread. The adjustments are made to each semaphore set atomically. If an adjustment would cause a semaphore value to overflow a limit (0 or SEM#MAX_VAL), no adjustment is made to that semaphore. Adjustments will continue for the set. No assumptions may be made as to the order in which the semaphore sets will be adjusted.

   When semval changes, the waiting chain is searched and other threads may regain control (as with semop, semctl operations). As adjustments are completed, sem_pid and sem_otime are updated for each semaphore set.

7. Even if full process termination does not occur, mvsprocclp will cause the terminating thread to wait up to 60 seconds for subtask termination. If the caller is the IPT, a reason code will be returned to indicate that a subtask is still attached.
8. By default, when an mvsprocclp call results in a full process cleanup, all processes that are found in the caller's subtask tree are terminated with a sigkill signal.

   If the ThliUndubCallerOnly flag is on in the caller's THLI when the mvsprocclp service is called, mvsprocclp does not attempt to terminate processes in the caller's subtask tree.

   You cannot use the ThliUndubCallerOnly flag to clean up the first process dubbed in the address space while other processes exist in the caller's address space. The mvsprocclp service will fail with return code EMVSERR and reason code JRActiveProcess.
9. WASTEXITCODE is a two-byte field that can be set to any value from 1 to 255. This is considered a user specified value for successful termination.
10. WASTSIGTERM is a two-byte field that can be set to any valid signal number (defined in BPXYSIGH). If the signal specified is SIGKILL (x'09'), then the termination request is treated as an abnormal termination, as if the task was abtermed.

Related services

- "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150
- "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882

Characteristics and restrictions

1. The mvsprocclp service is provided for non-C applications that invoke z/OS UNIX services. As a rule-of-thumb, if your program causes the task to be dubbed, issue mvsprocclp when it is complete. If your program is already dubbed when invoked, do not call mvsprocclp when exiting. If you know the termination of your program will cause the task to terminate, you can allow end-of-task processing to perform mvsprocclp for you. To determine if your program is already dubbed, you can test STCBOTCB in mapping macro IHASTCB. If this field is 0, it is not dubbed. You can also use the querydub callable service ("querydub (BPX1QDB, BPX4QDB) — Obtain the dub status of the current task" on page 566).
2. If a thread issued an attach_exec or an attach_execmvs, mvsprocclp can be called to allow up to 60 seconds for a full MVS subtask termination to complete. Although mvsprocclp would not trigger the termination of a subtask, a thread that exited the system while subtasks were attached would force those subtasks to terminate with an abend.

Examples

For examples of using this callable service, see "BPX1MPC (mvsprocclp) examples" on page 1164.

mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals

Function

The mvssigsetup callable service allows a task to catch or intercept signals. This service also allows a task to intercept cancellation and quiesce interrupts. Only one

mvssigsetup service in a process can be active. If a second mvssigsetup service must be performed in a process, an mvsunsigsetup service must be performed on the thread that issued the mvssigsetup service request before the second invocation of the mvssigsetup service.

Both MVS task termination and the mvsprocclp service (BPX1MPC, BPX4MPC) perform the mvsunsigsetup service.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MSS):	31-bit
AMODE (BPX4MSS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MSS,(Signal_interface_routine_address,
              User_data,
              Default_override_signal_set,
              Default_terminate_signal_set,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MSS with the same parameters. The Signal_interface_routine_address parameter and the User_data parameter are doublewords.

Parameters

Signal_interface_routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the user-supplied signal interface routine (SIR) that gets control when a signal handler needs to be invoked. The signal handler is defined by the sigaction call; see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746. You can also invoke the SIR to process a default signal action, depending on the values specified for Default_override_signal_set. See the usage note on using the upper bit of the SIR address for indirect signal handler addresses.

User_data
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Fullword (doubleword)

The name of a fullword (in 31–bit mode) or doubleword (in 64–bit mode) containing 4 or 8 bytes of user-supplied data that is passed to the signal interface routine on invocation from signal processing.

Default_override_signal_set
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
8 bytes

The name of an 8-byte area containing a 64-bit mask of signals that the SIR processes when their respective default actions take place. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. The signals SIGSTOP, SIGDUMP, and SIGTRACE cannot be intercepted. The bit positions that represent these signals are ignored. Signal 64 represents cancellation or quiesce requests. For more information, see "BPXYSIGH — Signal constants" on page 1039.

Default_terminate_signal_set
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
8 bytes

The name of an 8-byte area containing a 64-bit mask of signals specified in the Default_override_signal_set parameter that also causes the process to end. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. When set to 1, the signal represented results in a task that is either stopped or in a wait state to be interrupted by the signal. It is up to the signal interface routine to end the process. The bit that represents signal 64 of this mask is reserved. For more information, see "BPXYSIGH — Signal constants" on page 1039.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvssigsetup service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvssigsetup service stores the return code. The mvssigsetup service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The mvssigsetup service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSINITIAL	The service failed. The following reason codes can accompany the return code: JRNotPRB, JRPSWKeyNotValid, and JRAlreadySigSetup.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvssigsetup service stores the reason code. The mvssigsetup service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The user-supplied signal interface routine (SIR) is given control when the current PSW key is equal to the signal delivery key for the process. The signal delivery key for the process is defined as the PSW key when the process was dubbed for the first request for a callable service. A process image that results after the exec service or the execmvs service always has a signal delivery key of 8 and is not set up for signals.

If the signal handler addresses specified on the call to sigaction are not the actual handler addresses but pointers to the handler addresses, turn on the upper bit of the SIR address supplied on this service to enable ptrace to set breakpoints at the beginning of the signal handlers.

For information about the BPXYPPSD macro, see "BPXYPPSD — Map signal delivery data" on page 1014.

The SIR receives control with the following register interface:

Register	Contents
Reg 0	0
Reg 1	Address of standard parameter list. PARM1= address of BPXYPPSD; Reg 1 = ADDR(PpsdSirPARMS).
Regs 2–12	0
Reg 13	0 No save area for registers is provided to the SIR. The SIR does not save caller's registers.
Reg 14	0 No return address.
Reg 15	Set to address of the SIR.

The SIR receives control in the following system state:

Operation	Environment
Authorization:	Problem program state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	The same as the caller of mvssigsetup (BPX1MSS, BPX4MSS)
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Signal mask:	All signals that may be blocked by the signal mask are blocked.

Following are the steps that a user-supplied SIR must perform.

1. The SIR must obtain local storage for a local copy of the BPXYPPSD and copy the BPXYPPSD information into this local storage.
2. The PPSD contains the information necessary for the SIR to determine the reason for the interruption. The interruption can be the result of a signal, cancellation, or quiesce request.
3. If the interrupt cannot be processed at this time, possibly due to general register 13 not currently containing the address of a program stack, or the last service called on the current thread was cond_setup, then the queue_interrupt service request is issued (see “queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered” on page 568). Then go to step 11 on page 426.
4. If the interrupt is a signal and the default action is to be performed by the SIR, write the appropriate messages to the terminal and end the process. For more information about how to end the process, see “_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup” on page 150.
5. If the interrupt is a cancellation or a terminating quiesce request, clean up any necessary thread-related resources and end the thread. To end the thread, issue the pthread_get_and_exit service with Options_field set to PTEXITTHREAD. If the interrupt is because of a cancellation, issue the pthread_exit_and_get service with Status_field set to -1. For more information about how to end the thread, see “pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread” on page 505. If the interrupt is a freeze quiesce request, issue the quiesce_threads service to freeze the thread; see “pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 515. The SIR receives these types of interrupts only if bit 64 of the Default_override_signal_set is set on.
6. Obtain language stack storage for the signal handler.
7. Examine the sigaction call flags in the BPXYPPSD for the signal being delivered. Some of these flags, specified on the sigaction call, are intended to allow the user certain options when interfacing with signal catchers, or to provide additional processing. For example, the SA_SIGINFO flag specifies that additional signal information, also present in BPXYPPSD, should be passed to the signal catcher in a siginfo structure. It is up to the SIR to interpret and implement these sigaction flags. Refer to “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action” on page 746 for more information about the function of the sigaction flags. The relevant flags are:
   - SA_ONSTACK
   - SA_RESETHAND
   - SA_RESTART

- SA_SIGINFO
- SA_NODEFER

8. Set the signal processor mask to the appropriate value before invoking the signal handler. This mask is formed by taking the union of the current signal mask (PPSDCATCHERMASK), the value of Sa_mask specified on the sigaction call for the signal being delivered (PPSDSAMASK), and then including the signal being delivered (unless the SA_NODEFER flag is set). The signal processor mask is set by calling the sigprocmask service (BPX1SPM, BPX4SPM). Recursive calls to the SIR can occur after calling the sigprocmask service here to unblock signals. Therefore, the SIR must use the local copy of the BPXYPPSD macro after calling the sigprocmask service.
9. Conform to the language-dependent requirements for invoking signal-handlers.
10. On return from the signal handler, call the sigprocmask service to set the signal processor mask to the interrupted value that was saved in the BPXYPPSD field PPSDCURRENTMASK on entry to this SIR.
11. Use the CSRL16J MVS service to load 16 registers and jump to the address that was interrupted by the signal.

The use of the Default_terminate_signal_set is to indicate to the kernel which signals intercepted by the SIR cause the process to end. For example, a user might wish to intercept the **SIGUSR1** signal, but rather than performing the default of termination, the user might wish to have a message issued and then the signal thrown away (ignored). In this case, the user would turn the corresponding bit on in the Default_override_signal_set and off in the Default_terminate_signal_set. This bit set combination tells the kernel not to interrupt functions that return an EINTR.

Related services

- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304
- “pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread” on page 495
- “pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 515
- “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action” on page 746
- “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask” on page 757
- “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763

Characteristics and restrictions

See Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1313.

Examples

For an example using this callable service, see “BPX1MSS (mvssigsetup) example” on page 1166.

MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service

Function

The MVSThreadAffinity callable service allows a task created with pthread_create to request that a user-defined assembler routine is to be run on a specified target pthread. The requesting and target pthread must have been created with pthread_create, and both threads must be under the same initial pthread-creating thread (IPT). The requesting pthread is blocked until the requested routine has been run. The target pthread may be the IPT.

This service provides the ability for a program to manage MVS resources under the target pthread or IPT, instead of under the requesting pthread. Resources that can be managed with this service include load modules, opened data sets, and other MVS resources with task affinity.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TAF):	31-bit
AMODE (BPX4TAF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TAF,(Routine_address,
              Parameter_list,
              Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TAF with the same parameters. Routine_address and Parameter_list are 64-bit pointer fields.

Parameters

Routine_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing the address of the routine to which control is to be passed on the target pthread. The requesting pthread is responsible for ensuring that the routine to be run is in memory when it is called, and remains there until the call is complete.

Parameter_list
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the routine parameter list. The value in this fullword is passed in register 1 when the specified routine receives control. If the routine does not require parameters, specify 0.

Thread_ID
Supplied parameter

Type: Character string

Character set:
N/A

Length:
8 bytes

The name of an 8-byte field that contains the target pthread under which the routine is to run. This is the value returned by the pthread_self service, or pointed to by the PTXL field PTXLTHIDPTR provided by the pthread_exit_and_get service. (See "BPXYPTXL — Map the parameter list for pthread_create" on page 1032.) A value of all zeros will target the IPT.

Return_value
Returned parameter

Type: Integer

Character set:
N/A

Length:
Fullword

The name of a fullword in which the MVSThreadAffinity service returns `0` if the request completes successfully, or -1 if the request is not successful.

Return_code
Returned parameter

Type: Integer

Character set:
N/A

Length:
Fullword

The name of a fullword in which the MVSThreadAffinity service stores the return code. The MVSThreadAffinity service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The MVSThreadAffinity service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	A bad address was received as an argument of the call; or the specified routine experienced an abnormal end or program check that was not handled by the routine's recovery. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRBadAddress and JRRoutineError.
EAGAIN	Another pthread within the process has this call pending for the specified pthread. At most one pthread can request this service at a time for a given pthread. The requesting pthread can try again later when the current pending call is complete.
EACESS	A task other than a pthread-created task or IPT is not permitted to perform this service.
EINVAL	A thread with the specified thread ID was not found. The reason code accompanying this return code is JRThreadNotFound.
EMVSERR	The passed Routine_address is zero or greater than X'7FFFFFFF'. The reason code accompanying this return code is JrInvalidRoutine.

Reason_code
Returned parameter

Type: Integer

Character set:
N/A

Length:
Fullword

The name of a fullword in which the MVSThreadAffinity service stores the reason code. The MVSThreadAffinity service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The specified routine to be executed receives control with the following attributes:
   - Problem program state
   - Key of the target pthread task
   - AMODE 31
   - Primary ASC mode
2. The register usage on entry to the specified routine is:
   - R0: Undefined
   - R1: Address of Parameter_list, as specified by the caller of the MVSThreadAffinity service
   - R2–R12: Undefined
   - R13: Address of a 72-byte work area with which the routine gains control. For AMODE 64 callers, the work area is 144 bytes long.
   - R14: The return address from the specified routine to the MVSThreadAffinity. This address must be preserved by the invoked routine.
   - R15: Address of the invoked routine
3. Only tasks created with pthread_create or the IPT can invoke this service. If a task that is not an IPT or a pthread-created task requests this service, it receives an EACCES return code.

Related services

- "mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread" on page 410
- "pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread" on page 505

Characteristics and restrictions

There are no restrictions on the use of the MVSThreadAffinity service.

Examples

For an example using this callable service, see "BPX1TAF (MVSThreadAffinity) example" on page 1203.

mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup

Function

The mvsunsigsetup callable service deletes the task's signal set up established by the mvssigsetup service (see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421). The parameters specified in the mvssigsetup service are returned by the mvsunsigsetup service. The signal actions for all signals in the process set by the sigaction service (see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746) are set to default action SIG_DFL.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MSD):	31-bit
AMODE (BPX4MSD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MSD,(Signal_interface_routine_address,
              User_data,
              Default_override_signal_set,
              Default_terminate_signal_set,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MSD with the same parameters. The Signal_interface_routine_address parameter is a doubleword.

Parameters

Signal_interface_routine_address
Returned parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) return area where Signal_interface_routine_address, set by the mvssigsetup service, is returned.

User_data
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword return area where User_data, set by the mvssigsetup service, is returned.

Default_override_signal_set
Returned parameter

Type: Character string

Character set:
No restriction

Length
8 bytes

The name of an 8-byte area where Default_override_signal_set, set by the mvssigsetup service, is returned.

Default_terminate_signal_set
Returned parameter

Type: Character string

Character set:
No restriction

Length:
8 bytes

The name of an 8-byte area where Default_terminate_signal_set, set by the mvssigsetup service, is returned.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the mvsunsigsetup service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvsunsigsetup service stores the return code. The mvsunsigsetup service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The mvsunsigsetup service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMVSINITIAL	The service failed (JRNotSigSetup).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the mvsunsigsetup service stores the reason code. The mvsunsigsetup service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1MSD (mvsunsigsetup) example" on page 1166.

nice (BPX1NIC, BPX4NIC) — Change the nice value of a process

Function

The nice callable service changes the nice value of the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1NIC):	31-bit
AMODE (BPX4NIC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1NIC,(Nice_change,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4NIC with the same parameters.

Parameters

Nice_change
Supplied parameter

Type: Signed Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the relative change in the nice value of the calling process.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the nice service returns -1 if it is not successful. If it is successful, the nice service returns the new nice value minus NICE_ZERO. The constant NICE_ZERO is defined in the BPXYCONS macro (see "BPXYCONS — Constants used by services" on page 952).

Because the nice service can return the value -1 on successful completion, it is necessary to set the Return_code parameter to 0 before a call to nice. If nice returns the value -1, the Return_code parameter can be checked to see if an error occurred or if the service was successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the nice service stores the return code. The nice service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The nice service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EPERM	The nice_change value is negative, and the calling process does not have the appropriate privileges (see "Authorization" on page 8).
EMVSSAF2ERR	A security product internal error has occurred. Consult Reason_code for the exact reason for the error.
ENOSYS	The system does not support this function. Your installation has chosen not to enable it.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the nice service stores the reason code. The nice service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A process's nice value is a non-negative integer for which a more positive value would result in a lower CPU priority. A minimum nice value of 0 and a maximum value of (2*NICE_ZERO)-1 are imposed on all processes. If the specified nice_change value would result in a nice value that is outside this range, the nice value is set to the limit value. The default nice value for all processes is set to the constant value NICE_ZERO, which is defined in BPXYCONS.
2. If the specified nice_change value is negative, the value would result in a lowering of a process's nice value, thus giving the process a higher CPU priority. Only processes with the appropriate privileges (see “Authorization” on page 8) can lower their nice values.
3. The changing of a process's nice value has the same effect on a process's priority value, because they both represent the process's relative CPU priority. For example, increasing the nice value of a process to its maximum value of (2*NICE_ZERO)-1 has the effect of setting its priority value via the setpriority service to its maximum value (19), and will be reflected on the nice, getpriority, and setpriority services.
4. If the calling process is in a multiple-process address space, each of the processes in the address space has its nice value changed by the call to the nice service.
5. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
6. To set up the nice service, see the documentation for parmlib member BPXPRMxx in Enabling nice(), setpriority(), and chpriority() support in *z/OS UNIX System Services Planning*.

Related services

- “setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process” on page 688
- “getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process” on page 255

Characteristics and restrictions

If the calling process is running in a multiple-process address space, the nice values of all the processes in the address space are changed upon successful completion of the nice service.

Examples

For an example using this callable service, see "BPX1NIC (nice) example" on page 1167.

oe_env_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute

Function

The oe_env_np service examines, changes, or examines and changes an environmental attribute. The environmental attribute to be processed is determined by the value that is specified by the Function_code parameter.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1ENV):	31-bit
AMODE (BPX4ENV):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1ENV,(Function_code,
              InArgCount,
              InArgListPtr,
              OutArgCount,
              OutArgListPtr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4ENV with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Function_code
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword specifying a numeric value that identifies the environmental attribute the caller wants to examine, change, or both examine and change. Each environmental attribute has a specific Function_code value; these are defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Function
DFP_CLEANUP_EXIT_REG	Registers a DFP cleanup exit that is to be called during process cleanup processing. No other input parameters are applicable for this function. Upon success, a return value of zero is supplied. No unique error codes apply to this function code.
ENQWAIT_PROCESS	Determines the kernel behavior when pthread_quiesce (freeze or term) and pthread_cancel encounter threads in MVS ENQ waits.
FREEZE_EXIT_REG	Registers/deregisters a user exit that is to be given control when a pthread_quiesce(freeze_exit) call is made.
MVS_USERID	Retrieves the MVS user ID of the invoker.
ENV_TOGGLE_SEC	Toggles the task-level security.
ENV_STOR_SERVICE	Modifies storage attributes of an address space.
SHUTDOWN_REG	Registers the caller for special treatment at OMVS shutdown time.
WRITE_DOWN	Sets, resets, or queries the setting of the write-down privilege in the target ACEE.
PIDXFER_QUERY	Determines if the current process image was the result of a PIDXFER-type exec.
QUERY_MODE	Returns the AMODE, RMODE, and AMODE capability of a target PID.
MUST_STAY_CLEAN	Sets or queries the address space MUST_STAY_CLEAN state. Once this state is set, any loads or execs are prevented to files that reside in uncontrolled libraries.

The value that is specified for the Function_code also determines the number and length of input and output parameters. See the usage notes for details on defined function codes.

InArgCount
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword specifying a numeric value that indicates the number of parameters pointed to by the InArgListPtr parameter. If no input arguments are required, specify the name of a fullword that contains `0`. If `0` is specified, no environmental attributes are changed.

InArgListPtr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing an address that points to an array of addresses that point to parameters. If the value that is specified for InArgCount is `0`, the value that is specified for the InArgListPtr is ignored. See the usage notes for details on how to specify input parameters.

OutArgCount
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword specifying a numeric value that indicates the number of parameters pointed to by the OutArgListPtr parameter. If no output arguments are required, specify the name of a fullword that contains `0`. If `0` is specified, no environmental attributes are examined.

`OutArgListPtr`
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) containing an address that points to an array of addresses that point to parameters. If the value that is specified for OutArgCount is `0`, the value that is specified for the OutArgListPtr is ignored. See the usage notes for details on how to specify input parameters.

`Return_value`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the oe_env_np service returns `0` if the request is successful, or `-1` if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the oe_env_np service stores the return code. The oe_env_np service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The oe_env_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The InArgListPtr, OutArgListPtr or associated parameter lists point to a location that is partially or completely outside of the addressable storage range.
EINVAL	The Function_code value specified is not defined, or the InArgCount or OutArgCount parameter contains an incorrect count for the specified Function_code. The following reason codes can accompany the return code: JRFuncUndefined, JRBadArgCount or JRBadInputValue. If the SHUTDOWN_REG function was requested, the following reason codes can accompany the return code: JRJSTMustBeRegistered, JRAlreadyInShutDown, JRBlockPermAlreadyRegistered, JRBlockPermNotRegistered, JRBadInputValue, or JRAlreadyInShutDown.

Return_code	Explanation
EMVSSAF2ERR	There was an internal error in the security product. The hexadecimal reason code value contains the two-byte security product return code *xx* and reason code *yy*.
EMVSERR	The BPX.DAEMON FACILITY class profile is not defined. Reason code JRNoDaemon can accompany this return code.
EPERM	One of the following conditions occurred: • The calling process does not have the appropriate privilege to perform the requested operation. The reason code JROK can accompany this return code. If the SHUTDOWN_REG function was requested, the caller must be given read permission to the BPX.SHUTDOWN resource in the FACILITY class. • A call was made to register as a permanent process or job but the calling process was started with the respawn attribute. Reason code JRRespawnNotAllowed can accompany this return code.
ENOSYS	The implementation does not support this memory locking interface. Reason code JRNotBpxStorSwap can accompany this return code.

Reason_code
Returned parameter

Type: Integer

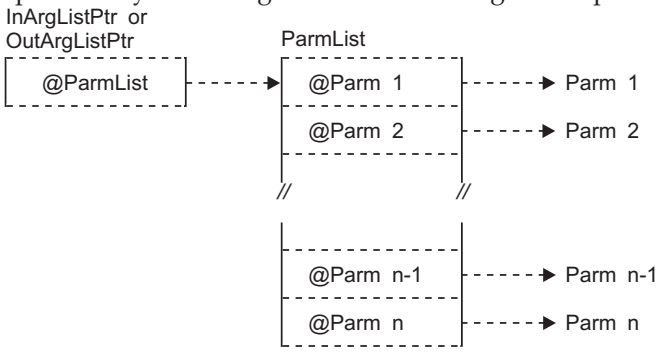
Length:
Fullword

The name of a fullword in which the oe_env_np service stores the reason code. The oe_env_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the value that is specified by InArgCount or OutArgCount is `0`, the corresponding InArgListPtr or OutArgListPtr is ignored. They must still be specified, but the value that is contained in the named field is irrelevant.
2. The oe_env_np service can examine, change, or examine and change environmental attributes based on the argument counts that are specified by the caller. If only an InArgCount is specified, an environmental attribute is changed, but the previous value is not returned. If only an OutArgCount is specified, the current setting of an environmental attribute is examined and returned but not changed. If both an InArgCount and OutArgCount are specified, the environmental attribute is changed and the previous setting is returned. If neither InArgCount nor OutArgCount are specified, no environmental attributes are examined or changed (NOOP), and the oe_env_np service sets Return_value to `0`.
3. The InArgListPtr and OutArgListPtr parameters each contain a fullword address that points to an array. The argument count (InArgCount and OutArgCount) defines the number of elements in each of these arrays. Each element in the arrays contains a fullword address that points to a parameter. The length of each parameter varies according to the Function_code specified.

The following figure is an example of an input or output parameter list as specified by the InArgListPtr and OutArgListPtr parameter.



4. The following table defines the number of input and output arguments (if specified) and the scope of each defined Function_code.

   The scope of an environmental attribute is the range of influence the attribute has in the kernel. The highest level of scope is ADDRESS SPACE: these attributes have influence over all processes and threads in an address space. The next level of scope is PROCESS: these attributes have influence over a single process. The lowest level of scope is THREAD: these attributes have influence over a single thread.

Function_code	# Input Args	# Output Args	Scope
ENQWAIT_PROCESS	1	1	Process
FREEZE_EXIT_REG	1	1	Process
MVS_USERID	0	1	Thread
ENV_TOGGLE_SEC	0	0	Thread
ENV_STOR_SERVICE	1	0	Address space
SHUTDOWN_REG	4	0	Process or address space
WRITE_DOWN	1	0 or 1	Address space or thread
PIDXFER_QUERY	0	1	Process
QUERY_MODE	1	3	Address space (AMODE, RMODE) or process (AMODE capability)
MUST_STAY_CLEAN	1	1	Address space

5. Function_code and argument definitions:
   - ENQWAIT_PROCESS

     The purpose of the ENQWAIT_PROCESS Function_code is to register with the kernel the behavior desired when a pthread_quiesce(freeze or term) or pthread_cancel encounters a thread in an ENQ wait in the caller's process.

     When ENQWAIT_PROCESS is disabled, the kernel does not interrupt threads that are found in ENQ waits. This means that pthread_quiesce(freeze or term) and pthread_cancel events are not delivered to a thread until after the ENQ wait has completed. This is the default behavior for all processes.

     When ENQWAIT_PROCESS is enabled, the kernel interrupts threads that are found in ENQ waits. The kernel delivers pthread_quiesce(freeze) events to

threads by scheduling an IRB on top of the SVRB for the ENQ wait, and freezing the thread from the IRB. For pthread_quiesce(term) or pthread_cancel events, the kernel abends threads in ENQ waits with a retryable 422 abend, reason code X'00000189'.

To get pending pthread_cancel and pthread_quiesce(term) events delivered, applications that invoke ENQ need to do the following:

– Establish an ESTAE before invoking ENQ.
– Have the ESTAE retry and check for the 422 abend with reason code X'00000189'.
– When the abend is detected, call sigprocmask and block all signals. On return from sigprocmask the pthread_cancel or pthread_quiesce(term) events are delivered.

When a pthread_cancel interrupts a thread in an ENQ, the target thread is abended (S0422-189). If an ESTAE has not been established or just percolates, the entire process is terminated. This behavior is required for standards compliance.

– Input arguments:
  - 1st argument (fullword):

Value	Definition
0	Disable ENQ wait interrupt support in the caller's process.
1	Enable ENQ wait interrupt support in the caller's process.

– Output arguments:
  - 1st argument (fullword):

Value	Definition
0	ENQ wait interrupt support is disabled in the caller's process.
1	ENQ wait interrupt support is enabled in the caller's process.

• FREEZE_EXIT_REG

The purpose of the FREEZE_EXIT_REG Function_code is to register with the kernel the address of an exit that is to get control when the Freeze-Exit function code of the pthread_quiesce service is requested. The exit gets control once on each thread processed by the pthread_quiesce(Freeze_Exit) service.

If a FREEZE_EXIT_REG is registered, the kernel gives control to the specified exit as a result of the pthread_quiesce(freeze_exit) call.

The user exit is given control while the pthread_quiesce service is still in progress. The user exit should not attempt to use any service that alters or terminates the current process. No callable services should be requested. If such services are attempted, the results are unpredictable.

The register usage on entry to the user exit is:

– R0: Undefined
– R1: Address of the parameter list defined by PpsdSIRParms. The first word of this parameter list is the address of the Ppsd.
– R2–R15: Undefined

When the exit returns, there are no expected return values or codes. The exit routine should terminate via SVC 3.

– Input arguments:

- 1st argument (fullword):

Value	Definition
address	Pthread_quiesce(Freeze_Exit) exit address.
0	Clear pthread_quiesce(Freeze_exit) address. Specifying zero unregisters an exit address.

– **Output arguments:**
- 1st argument (fullword):

Value	Definition
address	Pthread_quiesce(Freeze_Exit) exit address.
0	No exit has been registered with the kernel.

• MVS_USERID

The purpose of the MVS_USERID Function_code is to query the current MVS identity of the caller. The MVS user ID that is returned can be affected by the presence of a task-level security environment. If a task-level security environment has been created by the pthread_security_np service, the user ID that is associated with the task is returned.

The InArgCount must be 0, and the OutArgCount must be 1.

– Input arguments:
- 1st argument (None)

Value	Definition
N/A	N/A

– Output arguments:
- 1st argument (Char 8)

Value	Definition
The current MVS user ID	8-character user ID padded on the right with blanks.

• ENV_TOGGLE_SEC

The purpose of the ENV_TOGGLE_SEC Function_code is to toggle the task-level security. If the calling task has a task-level security environment, it is saved, and the task security is set back to the process level. If the calling task has a saved security environment and currently has no task-level security, the saved security environment is reinstated. If the calling task has not made a prior call to pthread_security_np, this call has no effect.

There are no additional input or output arguments, so the InArgCount and the OutArgCount must be 0.

• ENV_STOR_SERVICE

The purpose of the ENV_STOR_SERVICE Function_code is to modify the storage attributes of the caller's address space. The caller's address space cannot be made swappable unless it has previously been made non-swappable by this function. If the function is called to make an address space non-swappable and the current address space has already been made non-swappable by this function, the call is ignored and the address space remains non-swappable.

The InArgCount must be 1, and the OutArgCount must be 0.

– **Input arguments:**
- 1st argument (Structure):

ENV_STOR_FLAGS (Supplied Parameter)

ENV_STOR_FLAGS can be set to the following values defined in BPXYCONS macro:

Value	Definition
BPX_SWAP	Makes the address space swappable. The caller needs at least READ access to the BPX.STOR.SWAP resource in the FACILITY class.
BPX_NONSWAP	Makes the address space non-swappable. The caller needs at least READ access to the BPX.STOR.SWAP resource in the FACILITY class. When an application makes an address space non-swappable, it may cause additional real storage in the system to be converted to preferred storage. Preferred storage cannot be configured offline. Use of this service can therefore reduce an installation's ability to reconfigure storage in the future. Any application using this service should warn the customer about this side effect in the installation documentation.

– Output arguments:
  - 1st argument (None)

Value	Definition
N/A	N/A

• SHUTDOWN_REG

The purpose of the SHUTDOWN_REG Function_code is to request special treatment for the caller at OMVS shutdown time.

The SHUTDOWN_REG exit receives control in the caller's address space with it being both the home and primary address space. For a process-level registration, the exit runs in task mode from an IRB running on the initial thread task of the process that called the BPX*n*ENV registration service. For a job-level registration, it runs on the initial thread task of the first process in the address space. The authorization state of the exit will be the same PSW key and PSW state of the caller of BPX*n*ENV. The exit receives no parameters. Because the exit is driven from an IRB, it might not be able to issue other z/OS UNIX callable services; therefore, it should not depend on doing so.

The following is the register usage on entry to the exit:

– R0–R12: Undefined
– R13: Address of a 96-byte work area in the same key as the caller that registered the exit
– R14: The return address from the exit to the operating system. The exit must preserve this address to be used to return to the operating system.
– R15: The address of the exit

There are constants defined in BPXYCONS for use with the SHUTDOWN_REG function. See “BPXYCONS — Constants used by services” on page 952.

The InArgCount must be 4, and the OutArgCount must be 0.

– Input arguments:
  - 1st parameter (fullword):

The first parameter contains a value that indicates the type of shutdown registration being requested:

Value	Definition
1	Register as a blocking process or job
2	Register as a permanent process or job
3	Unregister as a blocking process or job
4	Unregister as a permanent process or job
5	Register for notification of shutdown
6	Unregister for notification of shutdown

- 2nd parameter (fullword):

  The second parameter contains a value that indicates the scope of shutdown registration being requested:

Value	Definition
1	Register for the calling job
2	Register for the calling process

  The values in the first two parameters are mutually exclusive; they cannot be combined.
- 3rd parameter (fullword):

  The third parameter contains a value that indicates the registration options being requested:

Value	Definition
1	Block system calls that are waiting for restart (valid for permanent registration only).
2	Abend system calls during shutdown/restart (valid for permanent registration only). This option is mutually exclusive with option value **1**.
4	Send a SIGTERM signal when shutdown is initiated.
8	Invoke a specified exit when shutdown is initiated. This option is mutually exclusive with option 4 (send a SIGTERM).

- 4th parameter (fullword):

  The fourth parameter contains the address of the exit that is to receive control at shutdown time, or `0`.

– **Output arguments:**
- (None)

• WRITE_DOWN

The purpose of the WRITE_DOWN Function_code is to set, reset, or query the setting of the write-down privilege in the target ACEE.

The InArgCount must be 2, and the OutArgCount must be 0 or 1.

– Input arguments:
- 1st argument (Fullword):

  The first argument contains a value that indicates the type of operation to be performed:

Value	Definition
0	Query the current setting of the write-down privilege.
1	Activate the write-down privilege.
2	Deactivate the write-down privilege.

Value	Definition
3	Reset the write-down privilege to its default value.

- 2nd argument (fullword):

  The second argument contains a value that indicates the scope of the write-down privilege operation to be performed:

Value	Definition
1	Perform write-down operation on address-space level ACEE (WD_SCOPE_AS).
2	Perform write-down operation on task-level ACEE (WD_SCOPE_THD).

– Output arguments:

- 1st argument (fullword):

  This argument is only returned when the OutArgCount is 1. If the OutArgCount is 0, this argument is ignored.

Value	Definition
0	Write-down privilege is inactive for ACEE (WD_IS_INACTIVE).
1	Write-down privilege is active for ACEE (WD_IS_ACTIVE).

• PIDXFER_QUERY

The purpose of the PIDXFER_QUERY Function_code is to determine if the caller's current process image was created as the result of a PIDXFER-type exec.

The InArgCount must be 0, and the OutArgCount must be 1.

– Input arguments:

- 1st argument: None.

– Output arguments:

- 1st argument (fullword):

Value	Definition
0	The current process image was not the result of a PIDXFER exec (PIDXFER_NO).
1	The current process image was the result of a PIDXFER exec (PIDXFER_YES).

• QUERY_MODE

The purpose of the QUERY_MODE Function_code is to query:

– The addressing mode (AMODE) of the target PID

– The residency mode (RMODE) of the target PID

– The ability of the address space in which the PID is running to support an AMODE 64 program; in other words, whether storage can be obtained above the bar

The service records the modes at the time the process was created (dubbed), or at the time of the last executed program. If a process has multiple threads, the service reports on the AMODE of the initial thread of the process.

The InArgCount must be 1, and the OutArgCount must be 3.

– Input arguments:

- 1st argument: PID of the target process.

– Output arguments:

- 1st argument (fullword):

  The first argument contains a value that indicates the AMODE of the target process:

Value	Definition
1	24-bit AMODE
2	31-bit AMODE
3	64-bit AMODE

- 2nd argument (fullword):

  The second argument contains a value that indicates the RMODE of the target process:

Value	Definition
1	24-bit RMODE
2	31-bit RMODE
3	64-bit RMODE

- 3rd argument (fullword):

  The third argument contains a value that indicates the AMODE capability of the target process:

Value	Definition
1	24-bit AMODE capability
2	31-bit AMODE capability
3	64-bit AMODE capability

• MUST_STAY_CLEAN

The purpose of the MUST_STAY_CLEAN Function_code is to ensure that a process and its children are, and will remain program-controlled. A process must be program-controlled in order to enable the MUST_STAY_CLEAN state. If a process is not program-controlled, this service will fail with a return code of EMVSERR and a reason code of JRENVDIRTY. Also, message BPXP015I is written to the console indicating the program that made the process uncontrolled. Once set, a process will not be able to load, execute, or spawn any files that are not from program-controlled libraries. Since this state is also propagated to children during a fork or spawn, the processes involved can fully trust one another.

The typical usage is for a process to set this state, create any children, and then in any child processes query the state. In order to trust the parent the child must issue the query service before any other security-related services are used in the child process. (For a list of these security services see Setting up the BPX.* FACILITY class profiles in *z/OS UNIX System Services Planning*.) In this way the child can be sure that it has inherited the state instead of the state being set by an action of the child. If the child recognizes that the state is enabled, all processes involved can trust one another. Usage of this service to enable the MUST_STAY_CLEAN state requires the BPX.DAEMON FACILITY class profile to be defined. If it is not defined the service will fail with a return code of EMVSERR and a reason code of JRNODAEMON.

– Input arguments:

- First argument (fullword):

Value	Definition
0	Query the current MUST_STAY_CLEAN state. The current state is returned in the first output argument.
1	Enable MUST_STAY_CLEAN state.

– Output arguments:
  - 1st argument (fullword):

Value	Definition
0	MUST_STAY_CLEAN state is disabled.
1	MUST_STAY_CLEAN state is enabled.
2	MUST_STAY_CLEAN state is conditional.

6. If the MUST_STAY_CLEAN service returns EMVSSAF2ERR, then the propagated failing 1-byte return code and 1-byte reason code of the IRRENS00 service is found in the last two bytes of the reason code returned by BPX1ENV/BPX4ENV.
7. A returned state of MSC_ENABLED indicates the state was set by this service and will continue even after **exec()s** that cause a job step to terminate. A returned state of MSC_ENABLED_COND indicates a previous security service, such as BPX1PWD, enabled the clean state and will be reset when the next job step runs.

Related services

- "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495
- "pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process" on page 515
- "pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create|delete thread-level security" on page 518

Characteristics and restrictions

1. Users of the blocking and permanent registration options of the SHUTDOWN_REG function must meet one of the following requirements:
   - The calling address space must be a system started task address space.
   - The caller must be running authorized (APF-authorized, system key 0–7, or supervisor state).
   - The caller must be a privileged z/OS UNIX process. It must have either superuser identity or read permission to the BPX.SHUTDOWN profile in the FACILITY class.
2. For the write-down privilege to be activated, the user ID in the target ACEE must be permitted to the IRR.WRITEDOWN.BYUSER profile in the FACILITY class. The FACILITY class must be active and RACLISTed, and the SETROPTS MLS option must be active.

Examples

For an example using this callable service, see "BPX1ENV (oe_env_np) example" on page 1134.

open (BPX1OPN, BPX4OPN) — Open a file

Function

The open callable service gains access to a file and creates a file descriptor for it.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1OPN):	31-bit
AMODE (BPX4OPN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1OPN,(Pathname_length,
              Pathname,
              Options,
              Mode,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4OPN with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname of the file.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the name of the file to be opened. The length of this field is specified in Pathname_length.

Path names can begin with or without a slash.

- A path name that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Options
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the binary flags that describe how the file is to be opened. For descriptions of the options, see "Usage notes" on page 450.

Options are mapped by the BPXYOPNF macro; see "BPXYOPNF — Map flag values for open" on page 1004.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword in which the mode field is specified. The mode field, which is mapped by BPXYMODE, specifies the file type and the permissions the caller grants to itself, to its groups, and to any user. See "BPXYMODE — Map the mode constants of the file services" on page 996.

If create or exclusive create is not specified on the Options parameter, the Mode parameter is ignored.

The file type is mapped by the BPXYFTYP macro; see "BPXYFTYP — File type definitions" on page 967.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the open service stores the file descriptor if the file was opened successfully, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the open service stores the return code. The open service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The open service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Reasons for being denied access include these: • The calling process does not have permission to search one of the directories specified in the Pathname parameter. • The calling process does not have permission to open the file in the way specified by the Options parameter. • The file does not exist, and write permission for the parent directory in which the file would have been created is denied to the calling process. • The truncate option was specified, but the process does not have write permission for the file.
EAGAIN	Resources were temporarily unavailable.
EBUSY	The path name specifies a master pseudoterminal that is either already in use or for which the corresponding slave is open, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation
EEXIST	The exclusive create option was specified, but the file already exists. The following reason code can accompany the return code: JRFileExistsExclFlagSet.
EFBIG	A request to create a new file is prohibited because the file size limit for the process is set to 0.
EINTR	The open operation was interrupted by a signal.
EINVAL	The Options parameter does not specify a valid combination of the O_RDONLY, O_WRONLY and O_TRUNC bits; or the file type that was specified in the Mode parameter is not valid. The following reason codes can accompany the return code: JRInvOpenFlags and JROpenFlagConflict.
EISDIR	The file specified by Pathname is a directory, and the Options parameter specifies write or read/write access. The following reason code can accompany the return code: JRDirWriteRequest.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMFILE	The process has either reached the maximum number of file descriptors it can have open, or the current pipe limit was exceeded. Refer to the reason code that was provided.
ENAMETOOLONG	The path name is longer than 1023 characters, or a component of the path name is longer than 255 characters. File names cannot be truncated.
ENFILE	The maximum number of file descriptors that can be open has been reached.
ENODEV	Typical causes: • An attempt was made to open a character special file for a device that is not supported. • An attempt was made to open a character special file for a device that is not yet initialized. The following reason code can accompany the return code: JRNoCTTY.

Return_code	Explanation
ENOENT	Typical causes: • The request did not specify that the file was to be created, but the file named by Pathname was not found. • The request asked for the file to be created, but some component of Pathname was not found, or the Pathname parameter was blank. The following reason codes can accompany the return code: JREndingSlashOCreat, JRNoFileNoCreatFlag, and JRQuiescing.
ENOSPC	The directory or file system intended to hold a new file has insufficient space.
ENOTDIR	A component of Pathname is not a directory.
ENXIO	The open request specified write-only and nonblock for a FIFO special file, but no process has the file open for reading. For pseudoterminals, it can mean that the minor number associated with the pathname is too big.
EPERM	The caller is not permitted to open the specified slave pseudoterminal; or the corresponding master is not yet open. EPERM is also returned if the slave is closed with HUPCL set, and an attempt is made to reopen it.
EROFS	The Pathname parameter names a file on a read-only file system, but options that would allow the file to be altered were specified: write-only, read/write, truncate, or—for a new file—create. The following reason codes can accompany the return code: JRReadOnlyFileSetWriteReq and JRReadOnlyFileSetCreatReq.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the open service stores the reason code. The open service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. When a file is created with the `Create` or `Exclusive_create` options of the Options parameter, the file permission bits as specified in the Mode parameter are modified by the process's file creation mask (see "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866), and then used to set the file permission bits of the file that is being created.

   The file's owner ID is set to the process's effective user ID (UID). By default, the owning GID is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
   - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
   - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.
2. Exclusive Create Option: If the exclusive create bit is set and the create bit is not set, the exclusive create bit is ignored.

3. Truncate Option: Turning on the truncate bit opens the file as though it had been created earlier, but never written into. The mode and owner of the file do not change (although the change time and modification time do); but the file's contents are discarded. The file offset, which indicates where the next write is to occur, points to the first byte of the file.
4. Nonblock Option: A FIFO special file is a shared file from which the first data written is the first data read. The Nonblock option is a way of coordinating write and read requests between processes that share a FIFO special file. Provided that no other conditions interfere with opening the file successfully, it works as follows:
   - If a file is opened read-only and Nonblock is specified, the open request succeeds. Control returns to the caller immediately.
   - If a file is opened write-only and Nonblock is specified, the open request completes successfully, provided that another process has the file open for reading. If another process does not have the file open for reading, the request ends with Return_value set to -1.
   - If a file is opened read-only and Nonblock is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for writing.
   - If a file is opened write-only and Nonblock is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for reading.
5. Synchronous Update Option: When this bit is set, the program is assured that all data updates have been written to permanent storage.
6. Sharing files with NFS clients on Windows workstations: Some remote NFS clients can open files on ø in such a way that no one else can open that file until the first program has finished and closed the file. This is done through the use of *share reservations* that prohibit others from concurrently opening the same file in certain ways. The share reservations are: Deny_Read, Deny_Write, Deny_Both, and Deny_None. If a file is open by an NFS client and a subsequent attempt to open it for reading or writing is made which is currently denied by the NFS client's share reservation, the open request will either block or return with EBUSY, depending on the O_NonBlock option that was specified. Conversely, if a file is open by a local program and an NFS client then attempts to open it and deny the type of access that is already established, the client's open request will fail.

Related services

- “close (BPX1CLO, BPX4CLO) — Close a file” on page 103
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 174
- “lseek (BPX1LSK, BPX4LSK) — Change a file's offset” on page 345
- “read (BPX1RED, BPX4RED) — Read from a file or socket” on page 572
- “stat (BPX1STA, BPX4STA) — Get status information about a file by pathname” on page 805
- “write (BPX1WRT, BPX4WRT) — Write to a file or a socket” on page 928
- “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 866

Characteristics and restrictions

See “Usage notes” on page 450.

Examples

See "BPX1OPN (open) example" on page 1168 for an example using this callable service.

MVS-related information

The Execution access requested bit is used by the exec service (see "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132) to verify that the process has permission to run the specified file. When the open service succeeds, the specified file is treated as read-only for this case.

opendir (BPX1OPD, BPX4OPD) — Open a directory

Function

The opendir callable service opens a directory so that it can be read with the readdir service.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1OPD):	31-bit
AMODE (BPX4OPD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1OPD,(Directory_name_length,
              Directory_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4OPD with the same parameters.

Parameters

Directory_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the name of the directory.

Directory_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Directory_name_length

The name of a field that contains the name of the directory. The length of this field is specified in Directory_name_length.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the opendir service stores a directory file descriptor that describes the specified directory, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the opendir service stores the return code. The opendir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The opendir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process does not have permission to search some component of the name that is specified as Directory_name; or it does not have permission to work with the directory itself.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Directory_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Directory_name.
EMFILE	Too many other files are already open for the process.
ENAMETOOLONG	Directory_name is longer than 1023 bytes; or a component of the pathname is more than 255 bytes long.
ENFILE	Too many files are already open.
ENOENT	The specified directory was not found. The following reason codes can accompany the return code: JROpenDirNotFound and JRQuiescing.
ENOTDIR	Some component of the pathname is not a directory. The following reason code can accompany the return code: JRTargetNotDir.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the opendir service stores the reason code. The opendir service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The opendir service opens a directory so that the first readdir service—see "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577—starts reading at the first entry in the directory.
2. Return_value is a file descriptor for a directory only. It can be used only as input to services that expect a directory file descriptor. These services are closedir, rewinddir, and readdir.

Related services

- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613

Characteristics and restrictions

There are no restrictions on the use of the opendir service.

Examples

For an example using this callable service, see "BPX1OPD (opendir) example" on page 1168.

openstat (BPX2OPN, BPX4OPS) — Open a file and obtain status information

Function

The openstat callable service gains access to a file, creates a file descriptor for it, and obtains its status.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX2OPN):	31-bit
AMODE (BPX4OPS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX2OPN,(Pathname_length,
               Pathname,
               Options,
               Mode,
               Status_area_length,
               Status_area,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4OPS with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname of the file that is to be opened.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the name of the file that is to be opened. The length of this field is specified in Pathname_length.

Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an absolute pathname. The slash refers to the root directory. The search for the file starts at the root directory.
- A pathname that does not begin with a slash is a relative pathname. The search for the file starts at the working directory.

Options
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the binary flags that describe how the file is to be opened. For a description of the options, see the options described for the open callable service in “Usage notes” on page 450.

Options are mapped by the BPXYOPNF macro; see “BPXYOPNF — Map flag values for open” on page 1004.

Mode
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword in which the mode field is specified. The mode field, which is mapped by BPXYMODE, specifies the file type and the permissions granted by the caller to itself, to its groups, and to any user. See"BPXYFTYP — File type definitions" on page 967.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the length of the Status_area parameter. To determine the value of Status_area_length, use the BPXYSTAT macro (see "BPXYSTAT — Map the response structure for stat" on page 1057).

Status_area
Supplied and returned parameter

Type: Structure

Length:
The length of BPXYSTAT or Status_area_length, whichever is less.

The name of an area to which the openstat service returns the status information for the file. Status_area is mapped by the BPXYSTAT macro (see "BPXYSTAT — Map the response structure for stat" on page 1057).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the openstat service stores the file descriptor, if the file is opened successfully; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the openstat service stores the return code. The openstat service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The openstat service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process was denied access for one of the following reasons: • The calling process does not have permission to search one of the directories that was specified in the Pathname parameter. • The calling process does not have permission to open the file in the way that was specified on the Options parameter. • The file does not exist, and the calling process does not have permission to write into files in the directory in which the file would have been created. • The truncate option was specified, but the calling process does not have write permission for the file.
EAGAIN	Resources were temporarily unavailable.
EBUSY	The Pathname parameter specified a master pseudoterminal that is already in use, or for which the corresponding slave is open.
EEXIST	The exclusive create option was specified, but the file already exists. The following reason code can accompany the return code: JRFileExistsExclFlagSet.
EFBIG	A request to create a new file is prohibited because the file size limit for the process is set to `0`.
EINTR	The openstat operation was interrupted by a signal.
EINVAL	The Options parameter does not specify a valid combination of the O_RDONLY, O_WRONLY and O_TRUNC bits; or the file type that was specified in the Mode parameter is not valid. The following reason codes can accompany the return code: JRInvOpenFlags and JROpenFlagConflict.
EISDIR	The file that is specified by Pathname is a directory, and the Options parameter specifies write or read/write access. The following reason code can accompany the return code: JRDirWriteRequest.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
EMFILE	The process has reached the maximum number of file descriptors it can have open.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters. Filename truncation is not supported.
ENFILE	The maximum number of file descriptors that can be open has been reached.
ENODEV	Typical causes: • An attempt was made to open a character special file for a device that is not supported. • An attempt was made to open a character special file for a device that is not yet initialized. The following reason code can accompany the return code: JRNoCTTY.

Return_code	Explanation
ENOENT	Typical causes: • The openstat request did not specify that the file was to be created, but the file that was named by Pathname was not found. • The openstat request specified that the file was to be created, but some component of Pathname was not found, or the Pathname parameter was blank. The following reason codes can accompany the return code: JREndingSlashOCreat, JRNoFileNoCreatFlag, and JRQuiescing.
ENOSPC	The directory or file system that was intended to hold a new file has insufficient space.
ENOTDIR	A component of Pathname is not a directory.
ENXIO	The openstat request specified write-only and nonblock for a FIFO special file, but no process has the file open for reading. For pseudoterminals, this could mean that the minor number that is associated with the pathname is too big.
EPERM	The caller is not permitted to open the specified slave pseudoterminal; or the corresponding master is not yet open. EPERM is also returned if the slave is closed with HUPCL set and an attempt is made to reopen it.
EROFS	The Pathname parameter names a file on a read-only file system, but options that would allow the file to be altered were specified: write-only, read/write, truncate, or (for a new file) create. The following reason codes can accompany the return code: JRReadOnlyFileSetWriteReq and JRReadOnlyFileSetCreatReq.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the openstat service stores the reason code. The openstat service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

For information about opening and creating files, see “open (BPX1OPN, BPX4OPN) — Open a file” on page 447.

Related services

- “open (BPX1OPN, BPX4OPN) — Open a file” on page 447
- “fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor” on page 196

Characteristics and restrictions

See the usage notes.

Examples

For an example using this callable service, see "BPX2OPN (openstat) example" on page 1168.

__passwd, __passwd__applid (BPX1PWD, BPX4PWD) — Verify or change security information

Function

The __passwd callable service verifies and/or changes the input user_name's password or password phrase, or verifies the input user_name's PassTicket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PWD):	31-bit
AMODE (BPX4PWD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PWD,(User_name_length,
              User_name,
              Pass_length,
              Pass,
              New_Pass_length,
              New_Pass,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PWD with the same parameters.

Parameters

User_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of User_name.

User_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the User_name_length parameter

The name of a field, of length User_name_length, that contains, left-justified, the name of the user whose Pass value is to be verified and/or changed.

Pass_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored and causes a SURROGAT class check. See "Usage notes" on page 462.

Pass
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pass_length parameter

The name of a field, of length Pass_length, that contains, left-justified, the password, PassTicket or password phrase that is to be verified.

New_Pass_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of New_Pass. This length must be between 1 and 8 characters for a password or between 9 and 100 characters for a password phrase. A length of zero indicates that New_Pass is to be ignored.

New_Pass
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the New_Pass_length parameter

The name of a field, of length New_Pass_length, that contains, left-justified, the new password or password phrase for the specified user.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __passwd service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __passwd service stores the return code. The __passwd service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*.

The __passwd service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	User_name, Pass, or New_Pass length is incorrect; or the user name has an illegal first character. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRUserNameLenError, JRPasswordLenError, JRNewPasswordLenError, and JRUserNameBad.
ESRCH	The user name specified is not defined to OMVS.
EACCES	The password specified is not authorized; access is denied.
EMVSERR	There is an error in the USER definition in the security product data base. The following reason codes can accompany the return code: JREnvDirty, JRPNoSAFUser, JRSAFGroupNoOMVS, JRSAFUserNoOMVS, and JRSAFNoGid. The caller environment is dirty; that is, a program was loaded from an unauthorized library.
EMVSEXPIRE	The password has expired.
EMVSPASSWORD	The new password is not valid.
EMVSSAFEXTRERR	A RACF authorization error has occurred. The reason code contains the RACF return and reason codes, respectively, in the two low-order bytes. See Table 9 on page 462 for more information.
EMVSSAF2ERR	A RACF authorization occurred. The reason code contains the RACF return and reason codes, respectively, in the two low-order bytes. See Table 9 on page 462 for more information.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __passwd service stores the reason code. The __passwd service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Table 9. RACF return and reason codes

RACF Return Code	RACF Reason Code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery
8	20	The user does not have appropriate RACF access to either the SECLABEL, SERVAUTH profile, or APPL.
30	00	The user is not authorized to the port of entry.
30	04	The user is not authorized to access the system on this day, or at this time of day.
30	08	The port of entry cannot be used on this day, or at this time of day.
34	N/A	The user is not authorized to use the application.
38	04	MLACTIVE requires a SECLABEL; none was specified.
38	08	The user is not authorized to the SECLABEL.
38	0C	The system was in a multilevel secure status, and the dominance check failed.
38	10	Neither the user's nor the submitter's security label dominates. They are disjoint.
38	14	The client's security label is not equivalent to the server's security label.

1. All return and reason codes are in hexadecimal.
2. Return codes 30, 34 and 38 are associated with the RACF RACROUTE REQ=VERIFY macro.
3. Return code 8 is associated with the initACEE (IRRSIA00) RACF callable service.

Table 9 is not a complete list of all possible RACF return code and reason code combinations. For RACF codes not listed here, see initACEE (IRRSIA00) callable service in *z/OS Security Server RACF Callable Services*.

Usage notes

1. If a profile is defined in the FACILITY class protecting the BPX.DAEMON resource, all programs that are loaded into the caller's address space must be controlled programs by the installed security product (such as RACF). If the __passwd service detects that a load of a non-program control was done, it fails with an errno of EMVSERR and an errnojr of JRENVDIRTY. See Establishing the correct level of security for daemons in *z/OS UNIX System Services Planning*.
2. New_Pass is ignored if New_Pass_length is 0. If New_Pass is specified for a password, the length must be less than or equal to 8. Further installation requirements may apply; for example, the length may need to be a minimum of 6. For a password phrase, the length must be less than or equal to 100 and greater than or equal to 9. If the Return_code indicates EMVSPASSWORD, the installation exit routine may have failed the request because the New_Pass did not meet some installation standard. If no installation exit is installed on this system, RACF rejected the password.
3. Mixed case passwords and PassTickets are supported if the installed security product supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

   The contents of the password phrase string are passed unchanged to the installed security product.

4. If an entry for the specified User_name is not found in the user database, or if the User_name is not defined to the OMVS segment, an ESRCH error is returned.
5. If the caller of the __passwd service has read access to the BPX.SRV.*userid* SURROGAT class profile, where *userid* is the user ID specified in the User_name parameter, a null Pass value (Pass_length set to zero) can be specified, and the __passwd service will return a successful Return_value. See Defining servers to process users without passwords in *z/OS UNIX System Services Planning* for more information about setting up SURROGAT profiles.

   If, however, a New_Pass is specified and Pass_length is specified as 0, the __passwd service fails with an EINVAL.
6. When no Pass value is specified, a SURROGAT class check is made, ensuring the caller has access to the profile BPX.SRV.*userid* (where *userid* is the value specified on the User_Name parameter). If the *userid* portion of the profile name has blanks in it, then RACROUTE REQUEST=AUTH results in ABEND282 RC5C. The dump is suppressed and the request fails with a return code of EMVSSAF2ERR and a reason code of JrRACFBlankExists
7. The __passwd_applid() function is equivalent to __passwd() with the added feature that it also allows an application identifier (applid) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the applid value is also used in conjunction with the USERID to verify the PassTicket.

   If an application is not using the __passwd_applid() function but still wants to pass an applid to this service, the application can set the applid value in the BPXYTHLI.

   - THLIEP_FunctionCode is set with ThliEP_ApplSet.
   - THLIEP_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, then the ThliEP_APPLID value is ignored.
   - ThliEP_APPLID is set to the APPLID value.

   If there is no applid value passed and the calling process has done a pthread_security_np() call, the applid value defaults to OMVSAPPL.

   If there is no applid value passed and the calling process has NOT done a pthread_security_np() call, the applid defaults to a null value.

   Some applications may need the applid to be specified as the JOBNAME. The application should set the ThliF2_SetApp bit prior to calling the password service. When this bit is on, the password service uses the application JOBNAME as the applid value passed to the security product. This is honored only if the process has not done a pthread_security_np() call. Specification of the applid in the THLIEP_APPLID field or via the __passwd_applid() call overrides the ThliF2_SetApp setting.
8. Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation or the installed security product can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.
9. The __passwd() service supports identity context references (ICR). When __passwd() is used in conjunction with pthread_security_np(), an ICR can be used to authenticate a user and then create a thread-level security context (ACEE), similar to using a user ID and password. Because an ICR is a one-time-use entity, the __passwd() and pthread_security_np() services must be called from the same thread and the ICR specified on pthread_security_np() must be identical to the ICR specified on the preceding __passwd().

Note: An ICR is not a user ID and password. It is a reference in the local identity context cache. See *z/OS Integrated Security Services EIM Guide and Reference* for more information regarding identity context references.

10. If environment variable BPXK_MIN_PWFOLD=YES is set then non-graphic characters will not be changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set

Related services

- "getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name" on page 260

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1PWD (__passwd, __passwd__applid) example" on page 1176.

pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name

Function

The pathconf callable service determines the current values of a configurable limit or option (variable) that is associated with a file or directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PCF):	31-bit
AMODE (BPX4PCF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PCF,(Pathname_length,
              Pathname,
              Name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PCF with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname parameter.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name that contains the path name of the file. The file has the length that is specified in Pathname_length.

Path names can begin with or without a slash:

- A path name that begins with a slash is an *absolute* path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a *relative* path name. The search for the file starts at the working directory.

Name
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains a value that indicates the configurable limit or option (variable) that is to be returned in Return_value. Use the BPXYPCF macro to specify which path name variable you want returned; see "BPXYPCF — Command values for pathconf and pathconf" on page 1005.

Variable returned	Description
PC_CHOWN_RESTRICTED	Change ownership ("chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93) function is restricted to a process with appropriate privileges (see "Authorization" on page 8), and to changing the group ID (GID) of a file only to the effective group ID of the process or to one of its supplementary group IDs.
PC_LINK_MAX	Maximum value of a file's link count.
PC_MAX_CANON	Maximum number of bytes in a terminal canonical input line.
PC_MAX_INPUT	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed as input before reading them.

Variable returned	Description
PC_NAME_MAX	Maximum number of bytes in a filename (not a string length; count excludes a terminating null).
PC_NO_TRUNC	Path name components longer than 255 bytes generate an error.
PATH_MAX	Maximum number of bytes in a path name (not a string length; count excludes a terminating null).
PIPE_BUF	Maximum number of bytes that can be written atomically when writing to a pipe.
_POSIX_VDISABLE	Terminal special characters maintained by the system can be disabled using this character value. For information on querying and setting these special characters, see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831 or "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842.
PC_ACL	The security product supports access control lists.
PC_ACL_ENTRIES_MAX	The maximum number of entries that can be placed in an access control list for the specified file.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pathconf service returns the current value of the path name variable corresponding to the Name specified, or -1 if not successful.

If the named path name variable does not have a limit for the specified file, then Return_value is set to -1, and Return_code and Reason_code remain unchanged.

If _POSIX_CHOWN_RESTRICTED is specified for Name, and _POSIX_CHOWN_RESTRICTED is active, Return_value is set to 1.

If _POSIX_CHOWN_RESTRICTED is specified for Name, and _POSIX_CHOWN_RESTRICTED is not active, Return_value is set to 0.

If _POSIX_NO_TRUNC is specified for Name, and _POSIX_NO_TRUNC is active, Return_value is set to 1.

If _POSIX_NO_TRUNC is specified for Name, and _POSIX_NO_TRUNC is not active, Return_value is set to 0.

If PC_ACL is specified for Name, and PC_ACL is supported, Return_value is set to 1.

If PC_ACL is specified for Name, and PC_ACL is not supported, Return_value is set to 0.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pathconf service stores the return code. The pathconf service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

If the named path name variable does not have a limit for the specified file, Return_value is -1 and Return_code is unchanged. Otherwise the pathconf service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Search permission is denied for a component of the path prefix.
EINVAL	See the usage notes for situations in which EINVAL is returned. The following reason code can accompany the return code: JRNotSupportedForFileType.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or some component of the path name is longer than 255 characters. Name truncation is not supported.
ENOENT	The named file does not exist; or the Pathname argument points to an empty string. The following reason code can accompany the return code: JRNotSupportedForFileType.
ENOTDIR	A component of the path prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pathconf service stores the reason code. The pathconf service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If Name refers to MAX_CANON, MAX_INPUT, or _POSIX_VDISABLE, the following applies:
   - If Pathname does not refer to a terminal file, the function returns -1 in Return_value, and sets Return_code to EINVAL.
2. If Name refers to NAME_MAX, PATH_MAX, or _POSIX_NO_TRUNC, the following applies:
   - If Pathname does not refer to a directory, the function still returns the requested information using the parent directory of the specified file.
3. If Name refers to PC_PIPE_BUF, the following applies:
   - If Pathname refers to a pipe or a FIFO, the value that is returned applies to the referred to object itself. If Pathname refers to a directory, the value that is returned applies to any FIFOs that exist or that can be created within the directory. If Pathname refers to any other type of file, the pathconf service returns -1 in Return_value, and sets the Return_code to EINVAL.
4. If Name refers to PC_LINK_MAX, the following applies:
   - If File_descriptor refers to a directory, the value that is returned applies to the directory.

Related services

- "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191

Characteristics and restrictions

There are no restrictions on the use of the pathconf service.

Examples

For an example using this callable service, see "BPX1PCF (pathconf) example" on page 1170.

pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal

Function

The pause service suspends execution of the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to end the thread.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PAS):	31-bit
AMODE (BPX4PAS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PAS,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PAS with the same parameters.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pause service returns -1 if completion of a signal-handling function causes control to be returned. The pause service does not otherwise return to its caller.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pause service stores the return code. The pause service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The pause service can return the following value in the Return_code parameter:

Return_code	Explanation
EINTR	A signal was received and handled successfully.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pause service stores the reason code. The pause service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A thread that calls the pause service does not resume processing until a signal is delivered with an action to either process a signal-handling function or end the thread. Some signals can be blocked by the thread's signal mask; see "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757 for details.
2. If an incoming unblocked signal ends the thread, pause never returns to the caller.
3. If the signal action is to process a signal-catching function, the signal interface routine (SIR), which is defined by the mvssigsetup call, is given control when the pause service returns.
4. A return code is set when any failures are encountered that prevent this function from completing successfully.
5. The signal interface routine is given control only when the PSW key of the caller is equal to the signal delivery key of the process. The signal delivery key is set to the PSW key of caller of the first callable service that dubbed the process.
6. If the caller has a PSW key that is different from the signal delivery key, or has a PSW key of zero, pause returns a return code of EMVSERR and a reason code of JRPSWKeyNotValid.

Related services

- "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746

- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered" on page 763
- "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1PAS (pause) example" on page 1170.

pfsctl (BPX1PCT, BPX4PCT) — Physical file system control

Function

The pfsctl callable service sends a command and argument to a physical file system (PFS). The meanings of the command and argument are specific to the PFS and are defined by the PFS.

For detailed information about the use of pfsctl, see *z/OS DFSMSdfp Advanced Services*.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB. If SRB, AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1PCT):	31-bit task or SRB mode
AMODE (BPX4PCT):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PCT,(File_System_type,
              Command,
              Argument_length,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PCT with the same parameters.

Parameters

File_System_type
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
8 bytes

The name of a field that contains the 8-character file system type name. The file system type name matches the TYPE operand that was specified on the FILESYSTYPE statement, or the NAME operand of the SUBFILESYSTYPE statement that defined this physical file system in the BPXPRM*xx* parmlib member.

Command
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the command that is to be passed to the physical file system.

Argument_length
Supplied parameter.

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Argument parameter.

Argument
Parameter supplied and returned

Type: Defined by the physical file system

Character set:
No restriction

Length:
Specified by the Argument_length parameter

Specifies the name of a buffer, of length Argument_Length, that contains the argument that is to be passed to the physical file system.

The buffer may be modified by the physical file system to return information to the caller.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pfsctl service returns -1 if the request is not successful.

Depending on the physical file system and the request involved, the length of any returned information that is placed in the Argument buffer may be returned here.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pfsctl service stores the return code. The pfsctl service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The pfsctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The Argument buffer address is not valid; or an address passed in the buffer is not valid.
EINTR	The service was interrupted by a signal.
EINVAL	A supplied parameter is incorrect. One of the following Reason_codes may accompany this Return_code: • JRFilesysNotThere - The File_System_type specified does not exist. • JrIOBufLengthInvalid - The Argument_length specified an incorrect value. • JrInvloctlCmd - The Command value was negative.
EMVSPARM	The command or argument parameters were rejected by the physical file system. In this case the accompanying Reason_code is generated by the physical file system. Refer to its documentation to determine the exact reason the error occurred.
ENOSYS	This function is not supported by the physical file system that was specified. The following reason code can accompany this return code: JRPfsctl.
EPERM	Permission was denied by the physical file system. The calling program does not have sufficient authority for the service that was requested.
EIBMBADTCPNAME	PC#SetIbmOptCmd was used, and the name that was specified did not match any of the transports configured under Common INET. The caller did not succeed in getting affinity to a single transport, and this is probably an error for the application.
ENXIO	PC#SetIbmOptCmd was used. The name that was specified did not match a socket stack, but Common INET is not configured on this system. Because this system does not have multiple socket transports configured, there is already a natural affinity to one single stack, and this failure may not be a problem for the application.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pfsctl service stores the reason code. The pfsctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This service is provided for generic communication between a program that is running in a user process and a physical file system.

   It is similar to w_ioctl, but the command is directed to the physical file system itself, rather than to or for a particular file or device.
2. There is no restriction on the length of the argument buffer. The address and length of the argument buffer are passed to the physical file system in a UIO structure on the vfs_pfsctl operation.
3. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
4. As an example of how a physical file system writer could make use of this function, consider the requirement to display status and performance statistics about the physical file system. You can collect this information in the physical file system, but you need a way to display it to the user.

   With pfsctl, your status utility program can easily fetch the information it needs from the physical file system. The utility needs to know the File_System_type name that the physical file system was started with, and this can be made known to it by the physical file system with the Name/token callable services. (See *z/OS MVS Programming: Assembler Services Guide* for information on the Name/token callable services.)
5. Command values less than X'0x40000000' are considered to be authorized commands. A check for appropriate privileges (see "Authorization" on page 8) is made and the results of this check are passed to the physical file system in the osi_privileged bit.
6. Command values less than zero are reserved by the system.
7. **PC#SetIbmOptCmd** — This pfsctl service chooses a particular sockets transport; this is similar to the setibmopt(IBMTCP_IMAGE) C function.

   The Command value for this function is X'C0000005'. You specify the desired transport with the File_System_type parameter. The Argument parameter is not used, and Argument_length should be 0, unless you are setting persistent address space affinity.

   The PC#SetIbmOptCmd function is used by programs that must connect to a specific socket transport, also known as a specific TCP/IP stack, when z/OS UNIX is configured with multiple transports for the AF_INET or AF_INET6 address families. After a transport is chosen, all subsequent socket requests for address family AF_INET or AF_INET6 create sockets that are exclusively attached to that single transport.

   This is similar to the function provided by ioctl(SIOCSETRTTD), except that ioctl(SIOCSETRTTD) detaches an existing socket from all but the specified transport, while pfsctl(PC#SetIbmOptCmd) causes future sockets to be attached to only the one transport. Using pfsctl for this function is significantly more efficient than using ioctl.

   When there is only one transport configured, all socket requests for that address family go directly to it, regardless of any prior calls to pfsctl(PC#SetIbmOptCmd). A call to pfsctl(PC#SetIbmOptCmd) is therefore not necessary in a single transport configuration, but the call will still fail if the name that is specified does not match that of a socket stack. There could be something wrong in the caller's configuration files that needs to be addressed.

A pfsctl(PC#SetIbmOptCmd) request may be issued more than once to change the chosen transport and affect future sockets that are created. If File_System_type is all blanks, the caller's process is reset to indicate no transports chosen.

The chosen transport is inherited over fork and preserved over exec. If this is not desired, the child process should call pfsctl(PC#SetIbmOptCmd) with a blank name to reset itself.

If Argument_length is four and the Argument value is one, this transport affinity also becomes an address space-level transport affinity. Otherwise, only process-level affinity is established. Address-space affinity persists over job steps within a job and over UNIX process termination and re-dubbing in that address space. It applies to all UNIX processes running within that address space, so long as the MVS JOBID of the address space does not change. Clearing a process's affinity also clears the address space affinity if an argument value of one is passed on that call.

Address-space-level affinity is intended for multiple job-step procedures in which one job step makes a call to pfsctl so that a program in a later job step will be restricted to the one specified transport (where that program does not have its own call to pfsctl or cannot be changed to do so). It may also be used to set affinity for a TSO address space, which affects all the programs and commands invoked afterwards.

To minimize the performance impact of this feature, an address space is checked for address-space level affinity only once in the life of a process, and that check is only made in the socket, gethostid, and gethostname functions. Consequently, the effect of setting address space affinity when other processes are currently running in the address space, or for future programs that have their own calls to pfsctl, is unpredictable. Address-space level affinity is not, strictly speaking, inherited over fork; however, it is applied to a process the first time a call to socket, gethostid, or gethostname is made, so that if the fork occurs after one of those calls, the process's affinity is inherited by its children.

The BPXTCAFF program supplied by IBM may also be used to establish an address-space-level transport affinity for started procedures, submitted job streams, and the TSO CALL command. The BPXTCAFF program takes one parameter, the transport name, and makes a call to pfsctl(PC#SetIbmOptCmd), passing that name with an argument value of one, as follows:

```
//STEP0 EXEC,PGM=BPXTCAFF,PARM=TPNAME
```

See Using specific transports under CINET in *z/OS UNIX System Services Planning* for more information about transport affinity.

8. **PC#SetIbmAsyIO** — This pfsctl service chooses a Sockets transport that supports asynchronous I/O.

   The Command value for this function is X'C0000006'. The File_System_type and Argument parameters are not significant, and Argument_length should be 0.

   This is similar to the function provided by PC#SetIbmOptCmd, except that you do not have to know the name of the TCP/IP stack.

   Note: This function is obsolete and should not be used.

   When there is only one transport, all socket requests for that address family go directly to it, regardless of any prior calls to pfsctl(PC#SetIbmAsyIO). It is not an error to call pfsctl(PC#SetIbmAsyIO) when there is only one transport configured, therefore, programs using this function do not have to be sensitive to how an installation is configured. If the single transport does not support asynchronous I/O, attempts to call asyncio later will fail.

The choice of an asynchronous capable transport can be reset with a call to pfsctl(PC#SetIbmOptCmd) with a File_System_type of all blanks.

The chosen transport is inherited over fork and preserved over exec. If this is not desired, the child process should call pfsctl(PC#SetIbmOptCmd) with a blank name to reset itself.

9. **PC#ErrorText** — This pfsctl command retrieves error text for z/OS UNIX return codes and reason codes and for TCP/IP and zFS reason codes. You can use this service to request the error description for a specified return code or the error description, action text, and issuing module name for a specified reason code.

   The Command value for this function is X'C000000B'.

   On entry, the Argument parameter specifies the buffer in which to pass the request type and the return code or reason code and to receive the requested text. The buffer header contains fields to pass the type of text being requested, the type of error code (return code or reason code), and the return code or reason code and is mapped as follows:

Offset	Length	Field
0	2	Text request type
2	2	Error code type
4	4	Reason code (for PC#EtReason)
6	2	Return code (for PC#EtErrno)

   The following request types indicate the type of text to be returned:

Text request type	Value	Text request description
PC#EtDesc	X'0000'	Get the error description text for a return code or reason code
PC#EtAction	X'0001'	Get the action text for a reason code
PC#EtModname	X'0002'	Get the issuing module name for a reason code

   The following error code types indicate whether the error code you are passing is a return code or a reason code:

Error code type	Value	Error code description
PC#EtReason	X'0000'	Request is for a reason code
PC#EtErrno	X'0001'	Request is for a return code

   The Argument_length includes the length of the buffer header and must be large enough to receive the requested text.

   On return, the requested text starts at the beginning of the buffer, overlaying the header. The Return_value indicates the number of bytes returned in the buffer. If the buffer is not long enough to hold all of the requested text, the service only returns the amount of data that fits in the buffer; there is no explicit indication that data was truncated.

10. **PC#TDNames** — This pfsctl service returns a list of the names of all the transport stacks that are configured under Common INET. These are the names specified with the NAME() parameter of the SUBFILESYSTYPE statements from the BPXPRMxx parmlib member.

    The Command value for this function is X'C000000F'.

The output of this function is a simple array of 8-byte names that are left justified and padded with blanks. The Argument parameter, which is used for the output area, must be large enough to hold all the names. If the Argument parameter is not large enough, it is filled with the number of whole names that will fit, and no indication is given that there are more names.

Tip: The maximum number of stacks configurable under CINET is 32, so an argument that is 256 bytes long will always be large enough.

The Return_value indicates the number of names that have been returned. If CINET is not configured, the Return_value will be zero

The File_System_type parameter is not used with this command.

11. **PC#DirGetHost** — This pfsctl service can be used to direct a BPX1HST gethostid() or gethostname() request to a particular socket transport. The PC#DirGetHost function can be used by programs that need to get the host ID or host name of a specific TCP/IP stack without setting or changing the stack affinity for a process.

    The Command value for this function is 'C0000014'x.

    You use the File_System_type parameter to pass the desired transport and the Argument parameter to pass the BPX1HST request parameters. The Argument is mapped as follows:

Offset	Length	Field
0	4	Input: Domain value
4	4	Input: 0 for gethostid or Length of the Name area
8	*	Output: Name area for gethostname

    The Argument_length is the total length of the structure that is being passed, including the space reserved for the output Name on a gethostname() request. Refer to BPX1HST for details about the parameters and the difference between the gethostid() and gethostname() requests.

    When Common Inet is not configured, the transport name is ignored and the BPX1HST request is directed to the one and only transport.

    The output from a successful call to pfsctl will be the output that would have been received from BPX1HST, as follows:

    - For gethostid() requests, the Return_value parameter contains the ID.
    - For gethostname() requests, the PC_Name field of the PC_DirGetHostArg structure contains the name.

    Refer to *z/OS UNIX System Services Planning* for more information about transport affinity.

12. **PC#IsSrcAddr** — This pfsctl service is used to validate the passed source IP address against a set of input source address selection preference flags.

    The Command value for this function is 'C0000018'x.

    In a Common Inet configuration the File_System_Type parameter can be used to direct the request to a specific TCPIP stack. This will override any process stack affinity that might be established.

    When File_System_Type is all blanks or all zeros the request is routed to the TCPIP stack specified in the scope_id of the sockaddr or to the stack that "owns" the provided source IP address. If the provided IP address is not in the home address list of any active TCPIP stack, then the request will be routed to the default TCPIP stack.

When Common Inet is not configured the File_System_Type parameter is ignored.

Argument value is the IsSrcAddr structure which is defined in the BPXYSOCK macro. This function can be invoked using the inet6_is_srcaddr() C function.

Return Value	Explanation
0	The input address does not match an address that satisfies the source address selection preference flags indicated.
1	The IPv6 address corresponds to a valid address in the node and satisfies the given source address selection preference flags.
-1	The IPv6 address input value does not correspond to any address in the node or the source address selection flags are not one of the valid preference flags or the PFSCTL service failed.

This PFSCTL service is only routed to a single active IPv6 TCPIP stack and not routed by the CINET on failure.

This service is only available for IPv6 addresses and IPv4-mapped IPv6 addresses. If an AF_INET6 NETWORK statement is not specified in the BPXPRMxx parmlib member or at least one AF_INET6 supported TCPIP stacks is not active, then the service is rejected with return code and reason codes.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see “BPX1PCT (pfsctl) example” on page 1170.

__pid_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list

Function

The __pid_affinity service adds or deletes an entry in a process's affinity list. When a process terminates, each process in its affinity list is notified (sent a signal) of the event. The __pid_affinity service dynamically creates or breaks an association between two processes. Its function is similar to the notification mechanism between parent and child processes, except that in this case the processes are not related in any way.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PAF):	31-bit
AMODE (BPX4PAF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PAF,(Function_code,
              Target_Pid,
              Signal_Pid,
              Signal,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PAF with the same parameters.

Parameters

Function_code
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the function to be performed. The following Function_code constants are defined by the BPXYCONS macro (see "BPXYCONS — Constants used by services" on page 952):

Constant	Function
PAF_ADD_PID#	The process and associated signal that are specified by Signal_Pid are to be added to the affinity list of the process that is specified by Target_Pid.
PAF_DELETE_PID#	The process and associated signal that are specified by Signal_Pid are to be deleted from the affinity list of the process that is specified by Target_Pid.

Target_Pid
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the PID (Process ID) of the process whose affinity list is to be altered. See the usage notes for limitations on the PIDs that can be specified.

Signal_Pid
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the PID (Process ID) of the process that, when the Target_Pid process terminates, is to be sent the signal that is specified by the Signal parameter. See the usage notes for limitations on the PIDs that can be specified.

Signal
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the signal that the Signal_Pid process is to receive when the process that is specified by Target_Pid terminates. The signal must be one that is defined by the BPXYSIGH macro.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __pid_affinity service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __pid_affinity service stores the return code. The __pid_affinity service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __pid_affinity service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One or more of the following conditions were detected: • The value of Signal is not a supported signal. • Target_Pid does not contain a value greater than 1. • Signal_Pid does not contain a value greater than 1. • Signal_Pid and Target_Pid are the same. The following reason codes can accompany the return code: JRInvalidSignal, JRTargetPid, JRPidsSame, and JRSignalPid.
EPERM	The caller does not have permission to send a signal to the process that is specified on the Signal_Pid parameter.
EMVSSAF2ERR	A System Authorization Facility (SAF) or RACF call had an error.
ESRCH	One or more of the following conditions were detected: • No process was found that corresponds to Target_Pid. • No process was found that corresponds to Signal_Pid. The following reason codes can accompany the return code: JRTargetPid and JRSignalPid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __pid_affinity service stores the reason code. The __pid_affinity service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

In the case of EMVSSAF2ERR, Reason_code contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF Check Privilege service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	The caller is not the owner of the target process.
8	12	There was an internal error during RACF processing.

Usage notes

1. The PIDs that are specified by the Target_Pid and Signal_Pid parameters must be greater than 1. Specifying a PID that is equal to or less than 1 results in an error.
2. In order for the caller to add an entry to the affinity list of a process (Target_Pid), the Signal_Pid process must exist, and the caller's process must have permission to send it a signal.
3. During process termination, the process attempts to send all the specified signals to the corresponding PID or PIDs in its affinity list. If a signal cannot be sent (for instance, if the process has already terminated), termination continues.
4. If a process changes identity after it has been added to another process's affinity list, the signal is sent upon process termination without permission being reverified.
5. Identical entries that contain the same PID (Signal_Pid) and signal are not allowed in a process's affinity list. If an attempt is made to add a process and an identical entry is found, the service completes successfully without adding another entry.
6. To delete an entry from an affinity list, the PID (Signal_Pid) specified by the caller must be the same as an entry in the Target_Pid process's affinity list.

Related services

- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1PAF (__pid_affinity) example" on page 1170.

pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe

Function

The pipe callable service creates a pipe. A pipe is an I/O channel that a process can use to communicate with another process, with another thread (in this same process or another process), or in some cases with itself. Data can be written into one end of the pipe and read from the other.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PIP):	31-bit
AMODE (BPX4PIP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PIP,(Read_file_descriptor,
              Write_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PIP with the same parameters.

Parameters

Read_file_descriptor
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pipe service stores the file descriptor for the read end of the pipe if the pipe is created successfully.

Write_file_descriptor
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pipe service stores the file descriptor for the write end of the pipe if the pipe is created successfully.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pipe service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pipe service stores the return code. The pipe service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The pipe service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EMFILE	The process has either reached the maximum number of file descriptors it can have open, or the current pipe limit was exceeded. Refer to the reason code that was provided.
ENFILE	Opening the pipe would exceed the number of files that the system can have open simultaneously.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pipe service stores the reason code. The pipe service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Processes can read from the Read_file_descriptor and write to the Write_file_descriptor. Data written will be read first-in, first-out (FIFO).
2. When the pipe call creates a pipe, the O_NONBLOCK and FD_CLOEXEC flags are turned off on both ends of the pipe. You can turn on these flags with the fcntl call; see "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174.

Related services

- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572

- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

There are no restrictions on the use of the pipe service.

Examples

For an example using this callable service, see "BPX1PIP (pipe) example" on page 1171.

__poe() (BPX1POE, BPX4POE) — Port of entry information

Function

The __poe() callable service specifies the port of entry information the system is to use in determining various levels of permission checking in a multilevel-secure system. The authorization that is required to invoke this service is one of the following:

- Read access to the BPX.POE resource in the FACILITY class
- A UID of 0 when the BPX.POE resource is not defined

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1POE):	31-bit
AMODE (BPX4POE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1POE,(Poecb_length,
              Poecb,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4POE with the same parameters.

Parameters

Poecb_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the length of the Poecb control block that is being passed in the next parameter. To determine the value of Poecb_length, use the BPXYPOE macro ("BPXYPOE — Map poe syscall parameters" on page 1013).

Poecb
Supplied and returned parameter

Type: Structure

Length:
Specified by the Poecb_length parameter

The name of a Poecb structure that is to be used to control this port of entry operation. See the section on the Poecb control block in the usage notes for details on setting the fields of the Poecb. The BPXYPOE macro ("BPXYPOE — Map poe syscall parameters" on page 1013) maps the Poecb.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __poe() service returns one of the following:

- `0`, if the request is successful.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __poe() service stores the return code. The __poe() service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __poe() service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Poecb structure is incorrect. Consult Reason_code to determine the exact reason the error occurred. The following reason codes unique to the __poe() service can accompany the return code: JRPoeLenErr, JRPoeScopeErr, JRPoeEntryTypeErr, and JRPoeEntryLenErr.
EPERM	The calling address space does not have the appropriate privileges to set the POE attributes.
EFAULT	A bad address was received in the POEEntryPtr field of the BPXYPOE mapping that was pointed to by the Poecb parameter.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __poe() service stores the reason code. The __poe() service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The ability to register port of entry is a privileged operation. An installation has two ways of allowing an application to use this service:
   a. For the highest level of security, the installation defines the BPX.POE resource in the FACILITY class. For an application to use this service, the user ID it runs under must be given read access to that resource.
   b. For a lower security arrangement, you can assign the user ID under which the application runs a UID of 0 so that it operates as a superuser.
2. **Poecb control block:** When the POEEntryPtr field in the BPXYPOE mapping contains the address of a file descriptor, the caller must indicate in the POEEntryType field what type of file the file descriptor represents. The two supported file types are:
   - POE#EntrySocket — the file descriptor is for a socket file.
   - POE#EntryFile — the file descriptor is for a non-socket file. This includes the following file types:
     – Character special
     – FIFO
     – Regular
     – Symbolic link
     – Directory

   To clear the POE data that has been registered for a thread or process, specify the following data and options in the poecb and call the __poe() service.
   - POE#Attributes set to zeros (X'00'') or blanks (X'40'')
   - POE#ActionWrite
   - POE#ScopeProcess or POE#ScopeThread

   When the POE data for a process or thread has been cleared, that data will no longer be included in the search order that z/OS UNIX services use to determine the POE data to be passed to RACF when creating a security context. For example, when both process and thread POE data has been registered, the thread-level data would take precedence over the process data. Clearing the thread-level data would cause it to be skipped and the process level data would be used if available.

   POE#ScopeThread, POE#ScopeProcess and POE#ScopeSocket are mutually exclusive; only one must be specified. If none or more than one are specified the request will fail with –1/EINVAL/JRPoeScopeErr

   POE#ReadPOE, POE#WritePOE and POE#SetGetPOE are mutually exclusive. If more than one is specified the request will fail with EINVAL/JRPoeActionErr.

   If the POE#ReadPOE, POE#WritePOE or POE#SetGetPOE options are specified the storage obtained for the POE control block should be POE#LenV2 bytes in length. POE#LenV2 must be specified for the Poecb_length parameter on the BPX1POE syscall. If the length is incorrect the request will fail with -1/EINVAL/JrPOELenErr

Table 10. Poecb control block

POE options		POE data		Description
Scope	**Action**	**Source**	**Destination**	
Socket	Read	Socket or file descriptor	POEAttributes area in the BPXYPOE mapping	POE data is extracted from the file/socket descriptor supplied by the caller and returned to the caller via the BPXYPOE area. The caller's thread and process level POE data in the Oaob and Otcb is unchanged.
	Write	n/a	n/a	Request fails with EINVAL/ JrPoeSocketScopeErr
	SetGet	n/a	n/a	Request fails with EINVAL/ JrPoeSocketScopeErr
	None	n/a	n/a	Request fails with EINVAL/ JrPoeSocketScopeErr
Process	R	Process level (OapbPOEAttrs)	POEAttributes area in the BPXYPOE mapping	Process level POE data copied from the Oapb is returned to the caller via the BPXYPOE area
	W	POEAttributes area in the BPXYPOE mapping	Process level (OapbPOEAttrs)	POE data received from the caller in the BPXYPOE area is copied to the process level POE data in the Oapb
	SetGet	Socket or file descriptor	Process level (OapbPOEAttrs) and POEAttributes area in the BPXYPOE mapping	POE data is extracted form the file/socket descriptor supplied by the caller. The data is copied to the process level POE data in the Oapb and returned to the caller via the BPXYPOE area
	None	Socket or file descriptor	Process level (OapbPOEAttrs)	POE data is extracted from the file/socket descriptor supplied by the caller and copied to the process level POE data in the Oapb
Thread	R	Thread level (OtcbPOEAttrs)	POEAttributes area in the BPXYPOE mapping	Thread level POE data copied from the Otcb is returned to the caller via the BPXYPOE area
	W	POEAttributes area in the BPXYPOE mapping	Thread level (OtcbPOEAttrs)	POE data received from the caller in the BPXYPOE area is copied to the thread-level POE data in the Otcb
	SetGet	Socket or file descriptor	Thread level (OtcbPOEAttrs) and POEAttributes area in the BPXYPOE mapping	POE data is extracted form the file/socket descriptor supplied by the caller. The data is copied to the thread-level POE data in the Otcb and returned to the caller via the BPXYPOE area
	None	Socket or file descriptor	Thread level (OtcbPOEAttrs)	POE data is extracted from the file/socket descriptor supplied by the caller and copied to the thread-level POE data in the Otcb

3. The POE data registered via the __poe() service is passed to RACF by z/OS UNIX services that create a new security context (ACEE). The z/OS UNIX services that utilize POE data are:
   - __passwd()
   - pthread_security_np()
   - __login()

- setuid()/seteuid()/setreuid()
- spawn (only with userid change)

4. When z/OS UNIX services call RACF to build a new security context, POE data is passed to RACF using the following parameters for RACROUTE REQUEST=VERIFY ENVIR=CREATE

Table 11. RACROUTE parameters for POE data

POE data	RACROUTE parameter
POELabel	SECLABL
POEProfile	SERVAUTH
POETermid	TERMID (1)
(1) TERMID is only specified by the z/OS UNIX __passwd() and setuid()/seteuid()/setreuid() services.	

The amount and source of POE data passed to RACF varies based on the caller's environment.

- When the caller's address space security label is non-SYSMULTI, only the SECLABL parameter is specified.
- When the caller's address space security label is SYSMULTI, the following RACROUTE parameters are specified when the corresponding POE data is found in the search order. POE data from only one level will ever be specified. When valid POE data is found in the search order the search stops. For example, if a thread-level POELabel (SECLABL) is found the search will continue for POEProfile (SERVAUTH) and POETermid (TERMID) thread-level data but will not search at the process or address space level.
  - SECLABL
    a. Thread level (Otcb)
    b. Process level (Oapb)
    c. Address space seclabel (SYSMULTI)
  - SERVAUTH
    a. Thread level (Otcb)
    b. Process level (Oapb)
  - TERMID (only for the __passwd() and setuid(0/seteuid()/setreuid() services)
    a. Thread level (Otcb)
    b. Process level (Oapb)

5. The Table 12 describes the POE data propagation for z/OS UNIX services that create a new process, a new process image (exec), or a new thread.

Table 12. POE data propagation for z/OS UNIX services

POE data propagation		
Service	**Thread POE data**	**Process POE data**
fork()	Yes	Yes
spawn()	No	Yes
exec()	No	Yes
pthread_create()	No	Yes (*)
attach_exec()	No	Yes
attach_execMVS()	No	Yes

Table 12. POE data propagation for z/OS UNIX services (continued)

POE data propagation		
Service	**Thread POE data**	**Process POE data**
execMVS()	No	Yes
* The newly created thread is in the same process as the thread calling pthread_create(). By default, process level POE data is shared between all thread in the process.		

Characteristics and restrictions

The __poe() service is restricted to users that have the appropriate privileges, as defined under "Function" on page 483.

Examples

For an example using this callable service, see "BPX1POE (__poe) example" on page 1171.

poll (BPX1POL, BPX4POL) — Monitor activity on file descriptors and message queues

Function

The poll service checks the I/O status of multiple open file descriptors and message queues. The file descriptors can be for character special files, pipes, sockets, or files.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1POL):	31-bit
AMODE (BPX4POL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1POL,(PollArrayPtr,
              NMsgsFds,
              Timeout,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4POL with the same parameters. The PollArrayPtr parameter is a doubleword.

Parameters

PollArrayPtr

Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains a pointer to an array of Pollfd structures. The elements of the array must be arranged such that the PollFd structures that contain file descriptors precede the PollFd structures that contain message queue identifiers, if any are specified.

There is one Pollfd structure for each file descriptor or message queue that is being polled. A Pollfd structure specifies the file descriptor or message queue and the events for which it is being polled. On return, the poll service sets the corresponding bit in the response section of the Pollfd structure if the requested condition is true.

The events that can be polled are:

POLLRDNORM
Normal data can be read without blocking.

POLLRDBAND
Data from a nonzero priority band can be read without blocking. For STREAMs, this flag is set in **revents**, even if the message is of zero length.

POLLIN
Same as POLLRDNORM.

POLLWRNORM
Normal data may be written without blocking.

POLLWRBAND
Priority data (priority band greater than 0) may be written.

POLLPRI
Out-of-band data may be received without blocking.

POLLOUT
Same as POLLWRNORM.

POLLNVAL
The specified **fd/msgid** value is not valid. This flag is only valid in the **revents** bitmask; it is ignored in the **events** bitmask.

POLLERR
An error has occurred. This flag is only valid in the **revents** bitmask; it is ignored in the **events** bitmask.

POLLHUP
The device has been disconnected. This event and POLLOUT are mutually exclusive; a stream can never be writable if a hang-up has occurred. However, this event and POLLIN, POLLRDRNORM, POLLRDBAND, or POLLPRI are not mutually exclusive. This flag is valid in the **revents** bitmask. It is ignored in the events member.

For more information about the format of this field, see "BPXYPOLL — Map poll syscall parameters" on page 1014. The flags in the **events** bitmask are defined as:

- POLLEPRI, POLLEWRBAND, POLLEWRNORM, POLLEOUT, POLLEIN, POLLERDBAND, POLLERDNORM

The flags in the **revents** bitmask are defined as:

- POLLRNVAL, POLLRHUP, POLLRERR, POLLRPRI, POLLRWRBAND, POLLRWRNORM, POLLROUT, POLLRIN, POLLRRDBAND, POLLRRDNORM

NMsgsFds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains two numbers, the sum of which gives the total number of PollFd structures pointed to by **PollArrayPtr**.

The first number, which is in the first halfword of the fullword, tells how many message queue PollFd structures were specified. This number must not exceed 32,767. The second number, which is in the second halfword of the fullword, tells how many file descriptor PollFd structures were specified. This number should not exceed 65,535.

Timeout
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains a timeout value, in milliseconds, that controls how the file descriptors/message queues are checked.

1. **No waiting**:

   If the Timeout value is `0`, poll returns immediately after checking the selected descriptors and queues; no waiting is done.

2. **Wait for a specified period of time**:

   If the Timeout value is greater than `0`, it specifies the number of milliseconds to wait for one of the events to occur before returning to the caller. (1000 milliseconds equal 1 second).

3. **Wait forever**:

   If the timeout value is -1, poll blocks until a requested event occurs or until the call is interrupted.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the poll service returns one of the following:

- The number of events that were found to be ready.

  The return_value is similar to **NMsgsFds**. The first halfword of return_value contains the number of message queues with ready events. The second halfword contains the number of file descriptors with ready events.

- `0`, if the timeout value expired before any of the events were met.

- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the poll service stores the return code. The poll service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The poll service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The allocation of internal data structures failed, but a subsequent request may succeed.
EINTR	The select service request was interrupted by a signal for the caller.
EINVAL	One of the parameters specified a value that was not correct. Consult the reason code to determine the exact reason for the error. The following reason codes can accompany this return code: JRWaitForever, JRInvalidNfds, JRNoFdsTooManyQIds.
EIO	One of the descriptors in the poll mask has become inoperative and it is being repeatedly included in a poll, even though other operations against this descriptor have been failing with EIO. A socket descriptor can become inoperative, for example, if TCP/IP is shut down. When a descriptor fails, a failure from poll cannot tell you which descriptor has failed, so generally poll will succeed, and these descriptors will be reported to you as being ready for whatever events were specified on the poll. When the inoperative descriptor is subsequently used on a receive or other operation, you will receive the EIO failure, and can then react to the problem with the individual descriptor. In general, you would close() the descriptor and remove it from the next poll mask. If the individual descriptor's failing return code is ignored, though, and an inoperative descriptor is repeatedly polled and used (even though each time it is used the call fails with EIO), eventually the poll call itself will fail with EIO.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the poll service stores the reason code. The poll service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Poll bits are supported as follows:

Regular Files
Always poll true for reading and writing. This means that all poll read and write bits are supported. They will never return with **POLLERR** or **POLLHUP**.

FIFOs / PIPEs
Do not have the concept of out-of-band data or priority band data. They support **POLLIN**, **POLLRDNORM**, **POLLOUT**, **POLLWRNORM**, and **POLLHUP**. They ignore **POLLPRI**, **POLLRDBAND**, and **POLLWRBAND**. They never return **POLLERR**.

TTYs / OCS
Same support as FIFOs and PIPEs, except that TTYs may return **POLLERR**.

Sockets
Have the concept of out-of-band data. They support **POLLIN**, **POLLRDNORM**, **POLLOUT**, **POLLWRNORM**, and **POLLPRI** for out-of-band data. They ignore **POLLRDBAND** and **POLLWRBAND**. They might return **POLLERR**, but they will never return **POLLHUP**.

If the value of **fd/msgid** is less than `0`, **events** is ignored and **revents** is set to `0` in that entry on return from poll.

In each **pollfd** structure, poll clears the **revents** member, except that where the application requested a report on a condition by setting one of the bits of **events** listed above, the poll service sets the corresponding bit in **revents** if the requested condition is true. In addition, poll sets the **POLLERR** flag in **revents** if the condition is true, even if the application did not set the corresponding bit in **events**.

The poll request is not affected by the *O_NONBLOCK* flag.

A file descriptor for a socket that is listening for connections indicates that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously indicates that it is ready for writing, once a connection has been established.

Characteristics and restrictions

There are no restrictions on the use of the poll service.

Examples

For an example using this callable service, see “BPX1POL (poll) example” on page 1171.

Pread() and Pwrite() (BPX1RW, BPX4RW) — Read from or write to a file without changing the file pointer

Function

The Pread() and Pwrite() callable service reads from or writes to a given position in a file without changing the file pointer.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RW):	31-bit
AMODE (BPX4RW):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RW,(File_descriptor,
             Fuio_Address,
             Fuio_Alet,
             Fuio_Length,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX4RW with the same parameters. The Fuio_Address parameter is a doubleword.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of an open file.

Fuio_Address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the Fuio control block, which contains the user request. This area is mapped by the BPXYFUIO macro (see "BPXYFUIO — Map file system user I/O block" on page 967).

The setting of the FuioAddr64 bit, and not the AMODE of the caller, indicates whether the buffer address is a 31-bit or 64-bit address. If FuioAddr64 is on, the buffer address is in FuioBufferAddr, and is 31-bit. If FuioAddr64 is off, the buffer address is in FuioBuffV64Addr, and is 64-bit.

Fuio_Alet
Supplied parameter

Type: Address

Length:
Fullword

The name of a fullword field that contains the address of Fuio_alet.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the Pread() and Pwrite() service returns the number of bytes that were actually read or written, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the Pread() and Pwrite() service stores the return code. The Pread() and Pwrite() service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. In addition to the return codes listed for the read and write callable services, the Pread() and Pwrite() service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The offset argument is not valid. The value is negative.
ENXIO	A request was outside the capabilities of the device.
EOVERFLOW	The file is a regular file and an attempt was made to read or write at or beyond the offset maximum associated with the file.
ESPIPE	File_descriptor is associated with a pipe or FIFO.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the Pread() and Pwrite() service stores the reason code. The Pread() and Pwrite() service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

BPX1RW/BPX4RW does not support conversion using Unicode Services.

Examples

For an example using this callable service, see "BPX1RW (Pwrite) example" on page 1183.

pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread

Function

The pthread_cancel callable service generates a cancelation request for the target thread.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTB)	31-bit
AMODE (BPX4PTB)	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTB,(Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTB with the same parameters.

Parameters

Thread_ID
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be canceled.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_cancel service returns `0` if the thread is canceled or the cancel is pending, or `-1` if a failure occurs.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_cancel service stores the return code. The pthread_cancel service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_cancel service can return one of the following values in the Return_code parameter:

Return Code	Explanation
EINVAL	The value that was specified by thread ID is not valid. It does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThID.
ESRCH	The value that was specified by Thread_ID does not refer to a thread that currently exists. The following reason codes can accompany this return code: JRThreadNotFound and JRAlreadyTerminated.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_cancel service stores the reason code. The pthread_cancel service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A successful call to pthread_cancel generates a cancelation request for the target thread.
2. Delivery of the cancelation request either causes a nonretryable 422 abend (with reason code 01A0), or causes the signal interface routine (established with BPX1MSS/BPX4MSS) to receive control.
3. If the invoking process sets _BPXK_FORCE_CANCEL=YES, this service is allowed to cancel threads that are not cancelable when this environment variable is set to NO, which is the default. To do this, the **pthread_cancel()** service will wait up to three seconds for the thread cancelation to take effect before terminating the target task with a 422 non-retryable abend, ReasonCode=1A0. The abend occurs only if after three seconds the thread has not terminated. If the target of the **pthread_cancel()** is the invoking thread, the service exits without waiting three seconds and the cancelation occurs upon exit from the **pthread_cancel()** service. If the invoking process sets _BPXK_FORCE_CANCEL=YES and then cancels a large number of threads, the amount of time to complete the cancels may be significantly larger than when the environment variable is NO. This is because the **pthread_cancel()** service may wait for up to three seconds before terminating each thread. For more information, see Commonly used environment variables in *z/OS UNIX System Services Planning*.

4. See the usage notes in "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527 for the definition of thread cancelation points.

Related services

- "pthread_create (BPX1PTC, BPX4PTC) — Create a thread"
- "pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread" on page 505
- "pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread" on page 509
- "pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread" on page 512
- "pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID" on page 526

Characteristics and restrictions

There are no restrictions on the use of the pthread_cancel service.

Examples

For an example using this callable service, see "BPX1PTB (pthread_cancel) example" on page 1172.

pthread_create (BPX1PTC, BPX4PTC) — Create a thread

Function

The pthread_create callable service creates new threads in the calling process. Each thread that is created represents a single flow of control within the process with its own unique attributes.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTC):	31-bit
AMODE (BPX4PTC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTC,(Init_rtn_addr,
              Work_area_addr,
              Attribute_area_addr,
              Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTC with the same parameters. Init_rtn_addr, Work_area_addr and Attribute_area_addr are doublewords.

Parameters

Init_rtn_addr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the initialization routine for the thread that is to be created. This routine is given first control when a new thread task is created to run the thread. In both AMODE 31 and AMODE 64, the actual address of the initialization routine is a 31-bit address.

Work_area_addr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a user-supplied work area that is later passed to the initialization routine. This address is in the parameter list that is returned by pthread_exit_and_get on a thread get request. For a description of this parameter list, see "BPXYPTXL — Map the parameter list for pthread_create" on page 1032.

Attribute_area_addr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the pthread attribute area that is used to define the attributes of the thread to be created. If a zero address is specified, the attributes are set to their default value. For the mapping of the pthread attribute area and the definition and defaults of the supported attributes, see "BPXYPTAT — Map attributes for pthread_exit_and_get" on page 1017. The address of the pthread attribute area is in the parameter list that is returned by pthread_exit_and_get on a thread get request. The BPXYPTXL macro also has a description of this parameter list; see "BPXYPTXL — Map the parameter list for pthread_create" on page 1032.

Thread_ID
Returned parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field in which the service returns the thread ID for the thread that is created. This field is valid only if the service returns successfully with a return value of `0`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_create service returns `0` if the request is successful, or -1 if it is not successful.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_create service stores the return code. The pthread_create service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_create service can return one of the following values in the Return_code parameter:

Return code	Explanation
EINVAL	One of the parameters contains a value that is not correct. Consult Reason_code to determine the exact reason that the error occurred. The following reason codes can accompany this return code: JRPtatEye, JRPtatSysLen, JRPtatSysOff, JRPtatLen, JRInitRtn, JRShSpMask, JRPtatWeight, JRPtatDetachState, and JRPtatSyncType.
EAGAIN	The system lacked the necessary resources to create the new thread.
EINVAL	The pthread_create service was requested in a multiprocess/multiuser process. The following reason code can accompany this return code: JRMultiProcUser.
ENOMEM (132)	Not enough space is available.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_create service stores the reason code. The pthread_create service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The thread initialization routine:

1. The pthread-creating task initialization routine has a user-specified routine to initialize the user environment for each new task that is created to process thread requests, and to control the processing of each thread that is to be run on that task.
2. The pthread-creating task initialization routine is first given control when a new MVS task is created to process a thread request. At this point, the

initialization routine should set up the user environment for the new task. After performing its initialization, the initialization routine can retrieve the first thread to process by invoking the pthread_exit_and_get callable service.

3. This routine performs its own initialization and cleanup processing for each thread that is to be processed.
4. When this routine gains control, signals and cancellation requests are blocked.
5. The environment in which the initialization routine receives control is described in the following table:

Operation	Environment
Authorization:	Problem program, key that is inherited from TCB key of initial pthread creating task
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31(64), the same as the caller of pthread_create
ASC mode:	Primary address space control (ASC) mode
Serialization:	Enabled for interrupts
Locks:	No locks held
Control parameters:	All parameters addressable in Primary

6. Upon entry to the initialization routine, the register contents are as follows:
   - R1 contains the address of a standard MVS parameter list. In AMODE 31 the parameter list consists of two 4-byte pointers. In AMODE 64 the parameter list consists of two 8-byte pointers. The parameter list consists of the following parameters:
     a. The address of an initial work area for use by the initialization routine during its setup processing.
     b. The address of a fullword field that contains the length of the initial work area.
   - R2–R12 are unspecified.
   - R13 contains the address of a 208-byte save area for use by the initialization routine to perform standard save area linkage and save the general and access registers.
   - R14 contains the return address for the initialization routine to return control to the system. This address must be preserved by the initialization routine. When the initialization routine is given control in AMODE 31, the high-order bit (bit 0) of this address is ON. When it is given control in AMODE 64, bits 32 and 33 of the 64-bit R14 are OFF. The initialization routine can always do a simple branch to return to its caller.
   - R15 contains the address of the initialization routine.
7. After the first thread request is received, in order for the initialization routine to process subsequent thread requests, it invokes pthread_exit_and_get within a loop. It can then exit the previous thread and obtain a new thread to process.
8. To provide the most efficient interface with the high-level-language environment, the following characteristics apply to the thread initialization routine:
   a. Only one pthread-creating task initialization routine is allowed per process image. When a process image is cleaned up after an invocation of the exec or execmvs service, the address can be changed. If the specified address is different within a given process image, the pthread_create invocation fails with a return value of -1, a return code of EINVAL, and a reason code of JRInitRtn.

b. Only one shared subpool mask is allowed per process image. When a process image is cleaned up after an invocation of the exec or execmvs service, the subpool mask can be changed. If the specified shared subpool mask is different within a given process image, the pthread_create invocation fails with a return value of -1, a return code of EINVAL, and a reason code of JRShSpMask.

c. The work area and pthread attribute area are passed through from pthread_create to the caller of pthread_exit_and_get without each being copied. The caller of pthread_create must therefore ensure that the storage that is provided for these items is not released or modified before these items are used by the caller of pthread_exit_and_get.

9. A minimum of 256 KB is required in the high private below the line. If this is not available, the pthread_create() terminates with a RC=ENOMEM and a RSNcode=0B510292.

Handling MVS tasks and threads:

Each thread that is created with pthread_create runs as an MVS subtask of the initial pthread-creating task (IPT). The IPT is the task that issued the first pthread_create call within the address space.

Note: The IPT is not the same as the pthread-creating task initialization routine. The IPT refers to the task that the first thread runs on, whereas the pthread-creating task initialization routine is the routine given control when a pthread_create is done.
When all the threads created with pthread_create and the IPT have ended, the next task in the address space to issue a pthread_create call is made the IPT.

Handling thread IDs:

1. Threads that are created by pthread_create are represented by 8-character thread IDs. A thread ID is unique only for a given process; multiple processes can have threads that are represented by the same thread ID.
2. Threads that are to be managed by a user application should also represent their threads with 8-character values. To distinguish between thread IDs that are managed by the system and those that are managed by a user application, the high-order bit of the thread ID indicates the origination of the thread ID. Thread IDs that are managed by a user application must have the high-order bit turned on. Thread IDs that are managed by the system have the high-order bit turned off.
3. Since thread IDs that are managed by the system can represent only mediumweight or heavyweight threads, those that are managed by a user application are considered to be lightweight threads. Any z/OS UNIX service that expects a thread ID as input fails if the thread ID represents a user-application-managed, or lightweight, thread.

When exiting from the initial pthread-creating task (IPT):

When exiting back to the operating system from the IPT, the caller may receive an A03 abend if any pthread_created tasks are still running. These tasks may still be running even if the IPT has called pthread_join for all the threads that it created. To avoid the A03 abend, the IPT should call the _exit service when it is ready to return to the operating system. The _exit service ends the IPT and all of its pthread_created subtasks without causing an A03 abend to occur.

Other usage notes

1. The pthread attribute area is passed as input to the pthread_create callable service to describe the attributes of the thread that is to be created. The area is split into two sections. The first section is the system attribute area, which is used by the system to build the new thread. The second section is the user area, which is intended for use by the pthread-creating task initialization routine that receives the address of the entire pthread attribute area from pthread_exit_and_get.
2. The system offset and user offset fields indicate where the start of each area begins. The system offset field (PTATSYSOFFSET) must be set to (PTATSYSOFFVAL), or pthread_create fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error. The user offset field PTATUSEROFFSET must be set to `0` if no user attributes are specified.
3. The system length and user length fields indicate the length of each area. The system length field (PTATSYSLENGTH) must be set to PTATSYSLENVAL. If it is not, pthread_create fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error. The user length field PTATUSERLENGTH can be set to any length. However, if the sum of PTATUSERLENGTH + PTATSYSLENGTH does not equal PTATLENGTH, pthread_create fails with a -1 return value, a return code of EINVAL, and a reason code that indicates the exact error.
4. The characteristics of each thread attribute and its impact on the pthread_create are as follows:
   - `Detach state` specifies the detach state of the thread that is to be created. A thread that is created in a DETACHED state cannot be joined (with the pthread_join callable service) by other threads, and has its system-obtained storage freed when it exits. A thread that is created in an UNDETACHED state can be joined by other threads, and does not have its system-obtained storage freed until it has been detached with pthread_detach. If the pthread attribute area is not specified on a pthread_create invocation, the default value is UNDETACHED.
   - `Weight` specifies the weight of the thread that is to be created. A thread that is created with the MEDIUMWEIGHT attribute allows the executing task to be reused when the thread exits.

     The thread is assumed to clean up all resources that it used. Due to system limits, the medium weight thread can sometimes be terminated. Pthreads in a wait for greater that 30 seconds are ended. If your application requires obtained resources to remain intact even if a medium weight thread is terminated, then consider using the ipt_affinity() service to associate the resource with the IPT. If the medium weight thread goes away, the resources are still associated with the IPT. If you use spawn() to start your application, you can use environment variable _BPXK_UNUSEDTASKS to keep idle medium weight threads in a wait. If you have hundreds of idle medium weight threads, system resources are used, which can degrade performance. For more information about the _BPXK_UNUSEDTASKS environment variable, see *z/OS UNIX System Services Planning*.

     When a heavyweight pthread exits, the associated MVS task can no longer request threads to process. If the pthread attribute area is not specified on a pthread_create invocation, the default value is HEAVYWEIGHT.
   - `Sync type` specifies the synchronous processing type of the thread to be created. The supported sync types are SYNCHRONOUS and ASYNCHRONOUS. A SYNCHRONOUS thread is one that is created only if the resources are immediately available to create it. An ASYNCHRONOUS thread is one that is queued until resources are available. An EAGAIN return code is received from a pthread_create invocation for a SYNCHRONOUS

thread if the resources are not available. This situation can occur if the thread or task limit was already reached for the calling process. If the task limit was reached, only ASYNCHRONOUS threads can be created. If the thread limit was reached, the service fails regardless of the SYNC TYPE. The thread or task limit is specified by parmlib member BPXPRMxx. If the pthread attribute area is not specified on a pthread_create invocation, the default value is SYNCHRONOUS.

- `Shared Subpool Mask type` specifies the set of subpools that are to be shared between threads. The bit positions of the mask represent the subpool number to be shared. If a bit is on, the subpool is shared. Specify subpools 1-127 by turning on their corresponding bit positions in the mask. Turning on the first bit indicates that subpool 1 is to be shared, and so on, to bit position 127. Bit 128 is the enabling bit; if it is off the subpool mask is ignored and the system default is used. The default shared subpools are 1, 2 and 78. The shared subpools must remain constant within the process image; any variation results in the failure of the pthread_create service.

5. If the calling thread is in a Workload Manager (WLM) enclave, the newly created thread is joined to the same WLM enclave. After the thread is joined, WLM can manage the calling thread and the newly created thread as one "business unit of work" entity for system accounting and management purposes.

Related services

- “pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread” on page 495
- “pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread” on page 505
- “pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread” on page 509
- “pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread” on page 512
- “pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID” on page 526

Characteristics and restrictions

To prevent unauthorized programs from gaining control in an authorized environment, pthread_create does not allow unauthorized callers if the IPT is running in an authorized key (0–7). Unauthorized callers are problem program state, key 8, and not job-step authorized). This restriction is required because the tasks that are created by pthread_create inherit the TCB key of the IPT.

To prevent deadlocking tasks within an MVS address space, pthread_create is supported only from the initial pthread_create task and from any of its daughter tasks. Invocations of pthread_create from any other tasks fail with a -1 return value, an EMVSERR return code, and a reason code of JRPTCNotSupp.

Examples

See “BPX1PTC (pthread_create) example” on page 1173.

pthread_detach (BPX1PTD, BPX4PTD) — Detach a thread

Function

The pthread_detach callable service detaches a thread in the calling process. When a thread is detached, its system storage can be reclaimed when the thread exits.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTD):	31-bit
AMODE (BPX4PTD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTD,(Thread_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTD with the same parameters.

Parameters

Thread_ID
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the thread ID for the thread that is to be detached.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_detach service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_detach service stores the return code. The pthread_detach service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_detach service can return one of the following values in the Return_code parameter:

Return code	Explanation
EINVAL	The value that was specified by thread ID is not valid; it does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThid.
ESRCH	The system has detected that the value that was specified by thread ID refers to a thread that is already detached or that cannot be found. The following reason codes can accompany this return code: JRThreadNotFound and JRAlreadyDetached.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_detach service stores the reason code. The pthread_detach service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497
- "pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread" on page 509

Characteristics and restrictions

There are no restrictions on the use of the pthread_detach service.

Examples

For an example using this callable service, see "BPX1PTC (pthread_create) example" on page 1173.

pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread

Function

The pthread_exit_and_get callable service exits a thread, gets a new thread request to process, or both. To start a new thread request, see "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTX):	31-bit
AMODE (BPX4PTX):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTX,(Status_field,
              Options_field,
              Signal_setup_userdata,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTX with the same parameters. The Status_field and Signal_setup_userdata parameters are doublewords.

Parameters

Status_field
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the status of the exiting thread. This status is available to any other thread that uses the pthread_join service to wait for the termination of this thread.

Options_field
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one of the following option values:

PTEXITTHREAD
Exit the calling thread. This causes the cleanup of system-related resources for the calling thread.

PTGETNEWTHREAD
Exit the last obtained thread and get the next available thread to process. The first invocation of pthread_exit_and_get from the pthread-creating task initialization routine must specify this option.

PTFAILIFLASTTHREAD
Exit the calling thread only if it is not the last thread in the process.

The default option value is PTEXITTHREAD. The option values are defined in the BPXYCONS macro; see "BPXYCONS — Constants used by services" on page 952. You can combine options by specifying a plus between them.

Signal_setup_userdata
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains 4 bytes (8 bytes) of user data that is normally supplied on the signal setup service, mvssigsetup. This field is used only when the PTGETNEWTHREAD option is specified. If this field contains a zero address, the signal setup user data is not changed for this thread. This field is ignored when the PTEXITTHREAD option is specified.

Return_Value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the service stores the return value. The return value varies depending on the options specified, as follows:

- **PTEXITTHREAD option value specified:**

  -1 The caller asked to exit the calling thread, but the thread could not be exited. For an explanation of the error, see Return_code and Reason_code.

  0 The thread was successfully exited.

- **PTGETNEWTHREAD option value specified:**

  -1 The caller asked for a new thread to process, but the thread request could not be satisfied. No new thread requests can be handled by the calling task. For an explanation of the error, see Return_code and Reason_code.

  >0 The address of the parameter list for the new thread request that is to be processed. The parameter list consists of the following:
  - The user work area address that was specified on the pthread_create invocation.
  - The user attribute area address that was specified on the pthread_create invocation.
  - The address of an 8-byte field that contains the thread ID of the thread request.
  - The address of a 4-byte thread run status field. For the possible status values and their definitions, see "BPXYPTXL — Map the parameter list for pthread_create" on page 1032.

- **PTFAILIFLASTTHREAD option value specified:**

  -1 The caller asked to exit the calling thread only if it was not the last thread, but the thread could not be exited. For an explanation of the error, see Return_code and Reason_code.

  0 The thread was successfully exited.

This parameter list is mapped by the BPXYPTXL macro; see "BPXYPTXL — Map the parameter list for pthread_create" on page 1032. The storage for the list is supplied by the system and should not be modified or freed by the caller of pthread_exit_and_get.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_exit_and_get service stores the return code. The pthread_exit_and_get service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The pthread_exit_and_get service can return one of the following values in the Return_code parameter:

Return code	Explanation
EINVAL	One of the parameters contains a value that is not valid. The following reason codes can accompany the return code: JRInvOption, JRGetFirst, JRHeavyWeight, JRQuiesceInProcess, and JRLastThread.

Reason_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_exit_and_get service stores the reason code. The pthread_exit_and_get service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. The pthread_exit_and_get service provides a highly efficient mechanism for processing mediumweight threads. A mediumweight thread is a unit of work that causes reuse of MVS tasks. If a mediumweight thread exits, the task is still capable of processing another mediumweight thread request. The pthread_exit_and_get service provides pthread_exit with an option that obtains a new thread for its caller to process.
2. The first invocation of pthread_exit_and_get from the pthread-creating task initialization routine must specify the PTGETNEWTHREAD option. On the first invocation, a thread request is retrieved without the occurrence of a thread exit. All subsequent invocations result in a thread exit, following which the next available thread request is obtained. If the PTGETNEWTHREAD option is not specified on the first pthread_exit_and_get invocation, the service fails with a -1 return value, an EINVAL return code, and a JRGetFirst reason code.
3. Using the PTGETNEWTHREAD option can cause a failure if the process is being quiesced. If this happens, the pthread_exit_and_get service fails with a -1 return value, an EINVAL return code, and a JRQuiesceInProgress reason code. At this point, the caller should perform its own cleanup and return to the operating system to allow the task to terminate.
4. If the PTFAILIFLASTTHREAD option is specified and the pthread_exit_and_get is issued from the last thread, the thread is not exited and a JrLastThread reason code is returned with a -1 return value and an EINVAL return code. Any thread that has never issued a pthread_create or that was not created with pthread_create is considered the last thread when the PTFAILIFLASTTHREAD option is used.
5. When pthread_exit_and_get is used to get a new thread request, the signal environment is inherited from the creator of the thread. The signal state for the newly created thread is roughly analogous to that of a newly created

process after the fork and exec services have been performed. The one exception is that the new thread inherits the setup state from the creator.

6. A successful invocation of pthread_exit_and_get awakens a thread that is waiting for the exiting thread, through the pthread_join service. The thread exit status that is specified on the pthread_exit_and_get call is made available to the waiting thread.
7. After pthread_exit_and_get is requested with the PTEXITTHREAD option from a given task, that task can no longer request z/OS UNIX services. An exception is the mvsprocclp service (BPX1MPC, BPX4MPC), which can be issued to undub the task. The caller should perform its own cleanup and return to the operating system to allow the task to end.
8. If pthread_exit_and_get fails for any reason (with a return value of -1), the caller should perform cleanup and return to the operating system to allow the task to end.
9. When a thread that specified the PTGETNEWTHREAD option is terminated with pthread_exit_and_get and the maximum allowable task limit is exceeded, a JRMaxTasks reason code is returned.
10. When this service is called from the initial pthread-creating task (IPT), it waits for all threads that were created with pthread_create to end.
11. For information about the pthread attribute area, see "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497.
12. If you are going to use this service in a multiple-pthread environment, see Appendix H, "Using threads with callable services," on page 1321.
13. If the PTEXITTHREAD option is used when a non-pthread_created thread exits, the status of the exiting thread is saved. This status is available to any other thread that uses the pthread_join service.

Related services

- "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497
- "pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread"

Examples

For an example using this callable service, see "BPX1PTX (pthread_exit_and_get) example" on page 1175.

pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread

Function

The pthread_join callable service obtains the termination status for a specific thread. The pthread_join service waits only if the thread has not ended, is not in a detached state, and is not currently joined by another thread.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTJ):	31-bit
AMODE (BPX4PTJ):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTJ,(Thread_ID,
              Status_field_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTJ with the same parameters. The Status_field_address parameter is a doubleword.

Parameters

Thread_ID
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the thread ID for the target thread that is to be waited upon.

Status_field_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of a status field in which to return the exit status of the thread that is specified by the thread ID value. If this field is zero, the thread exit status is not returned.

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_join service returns `0` if the request is successful, or -1 if it is not successful.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_join service stores the return code. The pthread_join service returns Return_code only if Return_value is -1.

For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_join service can return one of the following values in the Return_code parameter:

Return code	Explanation
EINTR	The calling process received a signal before the completion of an event that would cause the pthread_join service to return. The service was interrupted by a signal. In this case, the value contained in Status_field_address is undefined.
EINVAL	The value that was specified by thread ID is not valid; it does not contain a value that is consistent with thread IDs managed by the system. The following reason code can accompany this return code: JRLightWeightThread.
ESRCH	The value that was specified by thread ID does not refer to a thread that is undetached. The following reason codes can accompany this return code: JRThreadNotFound, JRAlreadyJoined, and JRAlreadyDetached.
EDEADLK	A deadlock was detected; or the value specified by thread ID refers to the calling thread. The following reason codes can accompany this return code: JRJoinLoop and JRJoinToSelf.

Reason_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_join service stores the reason code. The pthread_join service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The pthread_join service can be called repeatedly for a thread until the thread is detached. However, a thread can be the target of only one pthread_join at a time.
2. In AMODE 31, the status field pointed to by Status_Field_Addr is 4 bytes. In AMODE 64, the status field is 8 bytes. See “pthread_create (BPX1PTC, BPX4PTC) — Create a thread” on page 497 for further information.
3. When pthread_join is issued for a non-pthread-created thread that has ended, that thread will only be found if it exited via pthread_exit_and_get.

Related services

- “pthread_create (BPX1PTC, BPX4PTC) — Create a thread” on page 497
- “pthread_detach (BPX1PTD, BPX4PTD) — Detach a thread” on page 503

Characteristics and restrictions

There are no restrictions on the use of the pthread_join service.

Examples

For an example using this callable service, see “BPX1PTJ (pthread_join) example” on page 1174.

pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread

Function

The pthread_kill callable service targets a signal to a particular thread. The service is limited to interthread communication within a process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTK):	31-bit
AMODE (BPX4PTK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTK,(Thread_ID,
              Signal,
              Signal_options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTK with the same parameters.

Parameters

Thread_ID
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the target thread that is to receive the signal.

Signal
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that contains the signal number that is to be sent to the thread that is indicated by the Thread_ID parameter. This must be one of the signals defined in BPXYSIGH macro, or 0.

If the signal is 0, error checking takes place, but no signal is sent. The pthread_kill service can be called with a signal value of 0, to verify that Thread_ID parameter is correct before the signal is actually sent.

Signal_options
Supplied parameter

Type: Bit

Length:
Fullword

The name of a fullword field that contains the binary flags that describe how the signal is to be handled by both the kernel and the user-supplied signal interface routine (SIR). The signaling options are passed to the SIR in the signal information control block, which is mapped by BPXYPPSD; see "BPXYPPSD — Map signal delivery data" on page 1014. Signal_options are mapped as follows:

First 2 bytes
User-defined bytes that are delivered with the signal to the SIR in the signal information control block. These bytes are mapped by the BPXYPPSD macro.

Last 2 bytes
Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - signal code specified in the first 2 bytes is set by the application
- Remaining bits - reserved

Return_Value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_kill service returns `0` if the request is successful, or -1 if it is not successful.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_kill service stores the return code. The pthread_kill service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_kill service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One of the following conditions causes this return code: • The value of Signal is not valid, or is not the number of a supported signal. • The thread corresponding to Thread_ID was not found, not valid, or ended. The following reason codes can accompany the return code: JRInvalidSignal, JRLightWeightThid, JRThreadNotFound, and JRThreadTerm.

Reason_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_kill service stores the reason code. The pthread_kill service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The pthread_kill service provides a mechanism for asynchronously directing a signal to a thread in the calling process. This mechanism could be used, for instance, by one thread to cause the processing of other threads within the process.
2. The pthread_kill service is the only function that can issue the thread-scoped signals (**SIGTHSTOP** and **SIGTHCONT**). The **SIGTHSTOP** signal stops a specific thread; other threads in the process are not affected. The **SIGTHCONT** signal can be issued by the pthread_kill service to resume the stopped thread.

   SIGTHSTOP and **SIGTHCONT** can only be issued to threads within the same process. If all the threads in a process are stopped with **SIGTHSTOP**, the process is virtually hung. No other threads can send a **SIGTHCONT** signal to wake them up. The stopped threads must be manually killed.

   The **SIGTHSTOP** and **SIGTHCONT** signals are noncatchable, nonblockable, and cannot be ignored.

Related services

- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746

Characteristics and restrictions

There are no restrictions on the use of the pthread_kill service.

Examples

For an example using this callable service, see "BPX1PTK (pthread_kill) example" on page 1174.

MVS-related information

Delivery of a signal to the signal interface routine occurs only when the PSW key of the caller is equal to the signal delivery key of the process. The signal delivery key is set to the PSW key of the caller of the first callable service that dubbed the process.

pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process

Function

The pthread_quiesce callable service performs quiesce or query functions on threads. Depending on the function that is specified, pthread_quiesce queries the thread environment in the current process, or synchronously quiesces all threads in the current process (except for the calling thread, which returns when all threads have been quiesced).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTQ):	31-bit
AMODE (BPX4PTQ):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTQ (Quiesce_type,
              User_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTQ with the same parameters. User_data is a doubleword.

Parameters

Quiesce_type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one of the following values:

QUIESCE_TERM
Terminates all threads (except the invoking thread) that were created with pthread_create and IPT threads, allowing the signal interface routine to receive control when the quiesce request is delivered.

QUIESCE_FORCE
Terminates all threads (except the invoking thread) that were created with pthread_create, and IPT threads that do not allow the signal interface routine to receive control when the quiesce request is delivered.

PTHREAD_QUERY
Counts the number of threads that were created with pthread_create or IPT threads and returns the count in Return_value.

QUIESCE_FREEZE
Freezes all threads (except the invoking thread) in the process, including threads that were created with pthread_create, IPT, and MVS dubbed tasks. The signal interface routine is allowed to receive control when the quiesce event is delivered.

QUIESCE_UNFREEZE
Continues execution of all threads (except the invoking thread) in the process that are in a frozen state.

FREEZE_THIS_THREAD
Places the invoking thread into a frozen state, in response to a QUIESCE_FREEZE request.

The Quiesce_type values are defined in the BPXYCONS macro; see "BPXYCONS — Constants used by services" on page 952.

User_data
Supplied parameter

Type: Character string

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that is to be passed to the signal interface routine when the quiesce request is delivered.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_quiesce service places the return value. The return value varies depending on the Quiesce_type:

- **PTHREAD_QUERY quiesce type specified:**

  -1 The caller asked to query the number of threads that were created with pthread_create and IPT threads in the process, but the request could not be completed. For an explanation of the error, see the return code and reason code.

  0 The calling thread is the initial pthread-creating task (IPT), and no other threads that were created with pthread_create exist in the current process.

  1 The calling thread is created with pthread_create, not the IPT, and no other threads that were created with pthread_create or IPT threads exist in the current process.

  >1 The value indicates the number of threads that were created with pthread_create and IPT threads in the current process.

- **All other quiesce types specified:**

  -1 The caller asked to quiesce a thread in the current process, but the target threads may not all have been quiesced. For an explanation of the error, see the return code and reason code.

0 The target threads in the current process were successfully quiesced.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_quiesce service stores the return code. The pthread_quiesce service returns a Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_quiesce service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value specified for Quiesce_type was incorrect. The following reason code can accompany the return code: JRQuiesceTypeInvalid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_quiesce service stores the reason code. The pthread_quiesce service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Requesting pthread_quiesce with the QUIESCE_TERM or QUIESCE_FORCE options delivers a quiesce request to the IPT and all pthread_created threads in the process. When Quiesce_type is QUIESCE_TERM, the request is delivered to each thread by the signal interface routine (SIR) if the process is set up to intercept the quiesce request. If the process is not set up for quiesce request interception, or if Quiesce_type is QUIESCE_FORCE, the kernel performs the quiesce request for each thread. For details on how to intercept quiesce requests, see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421.
2. The kernel issues 422 abends when performing the termination quiesce request. If the request is intercepted by the user-defined SIR, it should perform whatever cleanup is necessary, and then issue the pthread_exit_and_get service to end the thread.
3. Requesting pthread_quiesce with the QUIESCE_TERM or QUIESCE_FORCE options from a thread that is not the IPT, or that was not created with the pthread_create service, has no effect on any threads in the process; and pthread_quiesce returns with a `0` return value.
4. The pthread_quiesce service should be requested with one of the terminating options before an exit (BPX1EXI, BPX4EXI) to prevent the other threads in the process from receiving an asynchronous abend.
5. When requested with one of the terminating options, the pthread_quiesce service posts all MVS tasks that are in pthread_exit_and_get (BPX1PTX, BPX4PTX) waiting for more work. The pthread_exit_and_get service returns to

the caller with a -1 return value. The caller can then clean up the task-related resources before the normal end (SVC 3) of the task.

6. If the pthread_quiesce service is invoked when Quiesce_type is PTHREAD_QUERY from a thread that was not created with pthread_create and is not an IPT thread, pthread_quiesce returns with a `0` return value.
7. The use of QUIESCE_FREEZE is not limited to the IPT and pthread_created threads. This option causes a quiesce event to be delivered to every other thread in the process. Upon return from pthread_quiesce, all threads in the process are no longer executing and are in a "frozen state".
8. If the target thread is intercepting quiesce events (see “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421), the signal interface routine gains control and is expected to either issue the queue_interrupt service (this is not a good time to freeze this thread) or issue the pthread_quiesce service with the FREEZE_THIS_THREAD option. However, since the quiescer is waiting for all threads to be placed into a frozen state, the pthread_quiesce service should be issued as soon as possible. If the target thread is not intercepting, the kernel places the thread into a frozen state.
9. The FREEZE_THIS_THREAD function places the thread into a frozen state only if a freeze request is pending on the calling thread. If a request is not pending, the FREEZE_THIS_THREAD function does not suspend execution. Control is immediately returned to the caller with a return code of zero.
10. When you want to restart the process, use the pthread_quiesce service with the QUIESCE_UNFREEZE option. All threads that are found to be in a frozen state are restarted.

Related services

- “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421
- “pthread_create (BPX1PTC, BPX4PTC) — Create a thread” on page 497

Examples

For an example using this callable service, see “BPX1PTQ (pthread_quiesce) example” on page 1174.

pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create|delete thread-level security

Function

This service creates or deletes the thread-level security environment for the caller's thread. The authorization that is required to invoke this service is one of the following:

- Read or update access to the BPX.SERVER resource profile in the FACILITY class.
- Superuser status.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN

Operation	Environment
AMODE (BPX1TLS):	31-bit
AMODE (BPX4TLS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TLS,(Function_code,
              Identity_Type,
              Identity_Length,
              Identity,
              Pass_Length,
              Pass,
              Option_Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TLS with the same parameters.

Parameters

Function_code

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the function that is to be performed. The following Function_code constants are defined by the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Function
TLS_CREATE_THREAD_SEC#	Creates a thread-level security environment for the caller's thread. If a thread-level security environment already exists, it is deleted before the new environment is created.
TLS_DAEMON_THREAD_SEC#	Creates a thread-level security environment for the caller's thread without the need for a password if the caller has READ access to the BPX.DAEMON resource in the FACILITY class.
TLS_DELETE_THREAD_SEC#	Deletes the thread-level security environment for the caller's thread, if one exists. If the security environment was created using the TLS_TASK_ACEE# option, only the POSIX security information is deleted; the task-level ACEE is left alone.

Constant	Function
TLS_TASK_ACEE#	Initializes the UNIX (POSIX) security data for a task that has an existing task-level security environment (task-level ACEE). If the UNIX security data already exists for the calling task, the existing UNIX security data is deleted, and a new set of UNIX security data is established.
TLS_TASK_ACEE_USP#	Takes an existing USP from a task-level ACEE and extracts the UID and GID information. This information is then used to build a complete MVS and POSIX security environment for the caller's thread.

Identity_Type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies a numeric value that identifies the format of the user identity that is provided in the Identity parameter. Constants are defined by the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Identity Format
TLS_IDENTITY_USERID#	The user identity is in the format of a 1-to-8-character user ID.
TLS_IDENTITY_CERT#	The user identity is in the form of a certificate control block.

Identity_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Identity. The specified length must be consistent with the allowable Identity types:

- USERID - 1 to 8 characters
- CERTIFICATE - the length of the constant OCRT_LEN, defined in the BPXYOCRT control block. See "BPXYOCRT — Map the OE certificate support structure" on page 1002.

Identity
Supplied parameter

Type: Character string or number, if using an identity type of USERID; structure, if using an identity type of CERTIFICATE.

Character set:
Not applicable for an identity type of CERTIFICATE. For an identity type of USERID, the XPG4 portable character set that includes upper and lower case letters (A-Z, a-z), numerics (0-9), period (.), dash (-), and underbar(_). In addition, the special characters $, %, and # may be specified. (Since these characters are not part of the XPG4 portable

character set, however, the future possibility of program portability should be considered before using these characters.)

Length:
Specified by the Identity_Length parameter

If the identity type is specified as TLS_IDENTITY_USERID#, this area is the name of a field that contains the user identity in the specified format.

If the identity type is specified as TLS_IDENTITY_CERT#, this area is mapped by the BPXYOCRT macro (see "BPXYOCRT — Map the OE certificate support structure" on page 1002).

Pass_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pass parameter. This length must be between 1 and 8 characters for a password or PassTicket, or between 9 and 100 characters for a password phrase. A length of zero indicates that the Pass parameter is to be ignored.

Pass
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pass_length parameter

The name of a field, of length Pass_length, that contains, left-justified, the password, PassTicket or password phrase that is to be verified.

Option_Flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword binary field that contains the pthread_security_np options. If no options are required, specify the name of a fullword field that contains `0`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword where the pthread_security_np service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_security_np service stores the return code. The pthread_security_np service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The pthread_security_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Permission is denied; the specified password is incorrect. The following reason code can accompany the return code: JROK.
EMVSEXPIRE	The password for the specified identity has expired. The following reason code can accompany the return code: JROK.
EINVAL	One or more of the following conditions were detected: • The Function_Code that was specified is undefined. • The Identity_Type that was specified is undefined. • The Identity_Length that was specified was not valid for the Identity_Type. • The Password_Length that was specified was not in the range 0 to 8. • An undefined option flag was set. The following reason codes can accompany the return code: JRTLSCertIDLenInvalid, JRTLSCertTypeInvalid, JRTLSCertLengthInvalid, JRTLSRequestInvalid, JRTLSIdTypeInvalid, JRTLSIdLengthInvalid, JRTLSAddressLengthInvalid, and JRBadOptions.
EMVSERR	An MVS environmental error has been detected. The following reason codes can accompany the return code: JRTLSCallerIsIPT, JRSecActive, JRTLSNotDoneByOE, JRNoPtraceTaskSec, JRNotWLMACEE, JRUnexpectedError, JRTLSDoneOnIPT, JRNoTaskACEE, JRSAFNoUID, JRSAFNoGID, JRSAFNoUSER JRSAFGroupNoOMVS, JRSAFUserNoOMVS, JRUnexpectedError and JRSAFInternal.
EPERM	One or more of the following conditions were detected: • The calling address space is not authorized to use this service. • A password was not supplied and the RACF SURROGAT class has not been activated; or no SURROGAT class profile has been defined for the specified user identity. • A password was not supplied and the caller's address space does not have READ permission to the specified user identity's RACF SURROGAT class profile. • A load from an unauthorized (not Program Control protected) library was done in the address space. The following reason codes can accompany the return code: JRNotServerAuthorized, JRSurrogateUndefined, JRNoSurrogatePerm, JRNoChangeIdentity, and JREnvDirty.
EMVSSAF2ERR	An error occurred in the security product. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRCertInvalid, JRRACFBlankExits, JRSAFInternal, and JRSAFParmlistError. The reason code can also contain the RACF return and reason codes, respectively, in the two low-order bytes. For more information, see Table 9 on page 462 and *z/OS Security Server RACF Callable Services*.
EMVSSAFEXTRERR	The user's access was revoked.

Return_code	Explanation
ENOSYS	The function is not supported on this system. The following reason code can accompany the return code: JRNoSecurityProduct.
ESRCH	The identity that was specified is not defined to the security product. The following reason code can accompany the return code: JROK and JRNoCertForUser.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_security_np service stores the reason code. The pthread_security_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The ability to create a task-level security environment (ACEE) using the function codes TLS_CREATE_ THREAD_SEC# and TLS_DAEMON_THREAD_SEC# is a privileged operation. An installation has the following two ways of allowing an application to use this service with these two function codes:
   a. For the highest level of security, the installation defines the BPX.SERVER resource profile in the FACILITY class. For the application to access this service, it must be given read access to this profile. In addition, all load modules that are executing in the application's address space must be defined to RACF. See the section on establishing UNIX security in *z/OS UNIX System Services Planning* for more information about setting up this security.
   b. For a lower security arrangement, you can assign the user ID under which the application is run a UID of 0 so that it operates as a superuser.
2. When a task-level security environment is established, the other z/OS UNIX services are divided into two categories:
   - Services that are used to access data in the file system base the permission checks on the task-level security. Therefore, a function like open() will only work if the identity of the user in the task security environment has permission to the file. The pthread_security_np is very useful for creating a file server.
   - Services that tend to be process oriented continue to base the permission checking on the security identity of the process. Functions like kill only work if the process has permission to send the signal to the target process. The IPC functions of shared memory, message queues, and semaphores are also accessed with the security environment of the process.

   The permission checks are done on the first call to this service, and a successful result is remembered so that future calls to the service run faster. Therefore, revoking access to the BPX.SERVER profile in the FACILITY class does not stop a running server from continuing to create task-level security environments.
3. If a thread with a task level security environment issues a spawn (non-local only), the child process is created with the identity of the thread. If a thread

with a task level security environment issues a fork, the child process is created with the identity of the process.

4. Access to most MVS resources is based on the security identity of the thread.
5. Mixed case passwords and password phrases are supported when the installed security product (such as RACF) supports mixed case; otherwise, passwords and PassTickets are folded to uppercase. Non-graphic characters are always folded to blanks.

   The contents of the password phrase string are passed unchanged to the installed security product.
6. The specification of a Pass value is optional. The following are some examples of situations in which a server would want to create a task-level security environment without a Pass value:
   - Some servers allow access to a system with a user ID known as ANONYMOUS. The ANONYMOUS user ID is defined to the system with access to data available to the general public. It is up to the installation to define and manage an ANONYMOUS user ID so that integrity is not compromised.
   - Some servers are connected to global security servers. If other services are used to authenticate a user, it is not necessary to provide a Pass value to this service. It is up to the application and the installation to define the level of user authentication that is acceptable.
7. Debugging in this environment is only allowed for users with read permission to the BPX.DEBUG resource profile in the FACILITY class.
8. This service cannot be called from the Initial Pthread Task (IPT) with function code TLS_CREATE_THREAD_SEC#. The RTL performs certain process-related functions on the IPT that would be adversely affected by a task-level security environment.
9. If the user identity that is specified by the caller has been defined as a SURROGAT user ID (see Defining servers to process users without passwords in *z/OS UNIX System Services Planning* for details on how to define a SURROGAT user), and no password was specified, the task-level ACEE that is created for the calling thread has the CLIENT feature turned on. When RACF encounters a task-level ACEE with the CLIENT feature turned on, authority checking is done using both the task and process-level ACEEs. Both ACEEs must have permission to be able to access the resource.
10. If the identity type is TLS_IDENTITY_CERT#, the user ID is returned to the caller, filled in with the user ID that is associated with the certificate and ended with a null character.
11. If the function code specified is TLS_TASK_ACEE#, the values specified for the Identity_Type, Identity_Length, Identity, Pass_Length, Pass, and Option_Flags parameters are ignored. Because the user had the authority to create a task-level ACEE and attach it to the TCB, no additional credentials are necessary to redub the thread with the POSIX identity associated with the user ID of the task-level ACEE.
12. For the TLS_TASK_ACEE# and TLS_TASK_ACEE_USP# function codes to be used successfully, either the caller must be supervisor state and system key (0–7), or the ACEE for the calling task must have been created by WLM.
13. The POSIX identity established by a TLS_TASK_ACEE# can be deleted in one of three ways:

- Issue another TLS_TASK_ACEE#. This deletes the old thread-level POSIX identity before establishing the new identity. This method fails, however, if the previous thread-level identity was not established by a previous TLS_TASK_ACEE#.
- Issue a TLS_DELETE_THREAD_SEC#. This deletes the POSIX thread-level identity, and the thread takes on the POSIX identity of the process.
- Issue a pthread_exit(). If the thread is heavyweight, the task terminates. If the thread is mediumweight, only the POSIX identity is cleaned up; the task-level MVS identity remains.

14. The pthread_security_applid_np() function is equivalent to pthread_security_np() with the added feature that it allows an application identifier (applid) to be supplied. The applid is used to verify the user's authority to access the application. When a PassTicket is specified, the applid is also used in conjunction with the USERID to verify the PassTicket.

    If an application is not using the pthread_security_applid_np() function but still wants to pass an applid to this service, the application can set the applid value in the BPXYTHLI.

    - THLIEP_FunctionCode is set with ThliEP_ApplSet.
    - THLIEP_ApplidLen is set to the length of the APPLID. If this value is less than 1 or greater than 8, then the ThliEP_APPLID value is ignored.
    - ThliEP_APPLID is set to the APPLID value.

    If there is no applid value passed, the applid value defaults to OMVSAPPL.
15. If the calling task does not have a USP associated with the task-level ACEE, the kernel treats this call as if the TLS_TASK_ACEE# function was called. If a USP is present, the kernel initializes the thread security environment with the UIDs and GIDs of the USP (supp groups are not used). Calling the pthread_security_np service with TLS_DELETE_THREAD_SEC# will return the thread to its original state. It is up to the caller to delete the ACEE or USP and maintain the task's TCBSENV field.
16. If the caller's IPT task has previously called BPX1TLS to create a thread-level security environment, then calling BPX1TLS with TLS_TASK_ACEE# or TLS_TASK_ACEE_USP# function codes from a pthread will fail.
17. If the pthread_security_np service returns a Return_code of EMVSSAF2ERR and the TLS_TASK_ACEE_USP# function code was specified, then the Reason_code will contain the propagated Return_code and Reason_code from the IRRSGE00 service.
18. Using the TLS_DAEMON_THREAD_SEC# function code without specifying a password is similar to using the BPX.SRV.*userid* surrogate support. The difference is that the installation need not set up individual surrogate profiles for each client that desires a thread-level identity in the target server process. The server is allowed to create any identity without authentication as long as it has been given READ permission to the BPX.DAEMON resource in the FACILITY class.
19. When function code TLS_CREATE_THREAD_SEC# is specified without specifying a PASS parameter, a SURROGAT class check is made, ensuring the caller has access to the profile BPX.SRV.*userid* (where *userid* is the value specified on the IDENTITY parameter). If the *userid* portion of the profile name has blanks in it, then the RACROUTE REQUEST=AUTH results in ABEND282 RC5C. The dump is suppressed and the request fails with a return code of EMVSSAF2ERR and reason code of JrRACFBlankExists.
20. Although z/OS UNIX System Services supports password phrases that are 9-100 characters in length, your installation or the installed security product

can have additional rules for password phrase lengths. Ask your security administrator or system programmer if any additional rules apply.

21. The pthread_security_np() will recognize and use the ICR passed if a previous __passwd() had been made that specified the same ICR. When __passwd() is used in conjunction with pthread_security_np(), an ICR can be used to authenticate a user and then create a thread-level security context (ACEE), similar to using a user ID and password. Because an ICR is a one time use entity, the __passwd() and pthread_security_np() services must be called from the same thread and the ICR specified on pthread_security_np() and the preceding __passwd() must be identical.

    Note: An ICR is not a user ID and password. It is a reference in the local identity context cache. See *z/OS Integrated Security Services EIM Guide and Reference* for more information regarding identity context references.

22. If environment variable BPXK_MIN_PWFOLD=YES is set then non-graphic characters will not be changed to blanks before being passed to the security product. This behavior will exist for all threads in the process where the environment variable was set

Related services

- "oe_env_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute" on page 435
- "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245

Characteristics and restrictions

The pthread_security_np service is restricted to users that have the appropriate privileges.

Examples

For an example using this callable service, see "BPX1TLS (pthread_security_np) example" on page 1205.

pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID

Function

The pthread_self callable service gets the thread ID of the calling thread.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTS):	31-bit
AMODE (BPX4PTS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTS,(Thread_ID)
```

AMODE 64 callers use BPX4PTS with the same parameter.

Parameters

Thread_ID
Returned parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field in which the service places the thread ID of the calling thread.

Usage notes

1. The caller should request this service only once when it needs the thread ID of the active thread. It should save a copy of the thread ID in its own storage for repeated usage.
2. If this service fails, the calling thread ends abnormally.

Related services

- "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497

Characteristics and restrictions

There are no restrictions on the use of the pthread_self service.

Examples

For an example using this callable service, see "BPX1PTS (pthread_self) example" on page 1175.

pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state

Function

The pthread_setintr callable service sets the specified interruptability state of the calling thread and atomically returns the previous interruptability state.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PSI,(Interrupt_state,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PSI with the same parameter.

Parameters

Interrupt_state
Supplied parameter

Type: Structure

Length:
Fullword

Specifies the name of a fullword that contains a numeric value that identifies the interrupt state that is to be set. The following constants, which are defined in the BPXYCONS macro, define the valid states (see "BPXYCONS — Constants used by services" on page 952):

Constant	Description
PTHREAD_INTR_ENABLE#	When interruptability is enabled, new or pending cancelation requests are acted upon according to the interruptability type set by the pthread_setintrtype service (BPX1PST, BPX4PST).
PTHREAD_INTR_DISABLE#	When interruptability is disabled, cancelation requests against the target thread are held pending.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns the previous interrupt state, or -1 if the service did not complete successfully.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_setintr service stores the return code. The pthread_setintr service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System*

Services Messages and Codes. The pthread_setintr service can return the following value in the Return_code parameter:

Return code	Explanation
EINVAL	One of the parameters contains a value that is not valid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_setintr service stores the reason code. The pthread_setintr service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Setting the interruptability state allows you to control when cancelation requests sent via the pthread_cancel (BPX1PTB, BPX4PTB) service are handled.
2. The pthread_setintr (BPX1PSI, BPX4PSI) and pthread_intrtype (BPX1PST, BPX4PST) services establish three interruptability states:
   - **Disabled:** Cancellation requests are left pending.
   - **Controlled:** Cancellation requests are left pending until the next cancelation point is reached. Cancellation points are defined as when:
     – The pthread_testinr service is invoked (BPX1PTI, BPX4PTI).
     – A thread is placed in an unbounded wait during a call to a z/OS UNIX service. Some examples of these types of calls are
       - “close (BPX1CLO, BPX4CLO) — Close a file” on page 103
       - “cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event” on page 118
       - “fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors” on page 174
       - “open (BPX1OPN, BPX4OPN) — Open a file” on page 447
       - “pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal” on page 468
       - “pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread” on page 509
       - “pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur” on page 536
       - “read (BPX1RED, BPX4RED) — Read from a file or socket” on page 572
       - “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763
       - “sigwait (BPX1SWT, BPX4SWT) — Wait for a signal” on page 769
       - “sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time” on page 771
       - “tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted” on page 824
       - “tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal” on page 842

  - "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882
  - "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

- **Asynchronous:** Cancellation request can be delivered at any time.

3. The default interrupt state for newly created threads and the initial thread is PTHREAD_INTR_ENABLE#.
4. The default interrupt type for newly created threads and the initial thread is PTHREAD_INTR_CONTROLLED#.
5. The interruption types of controlled and asynchronous are set with pthread_intrtype (BPX1PST, BPX4PST). See "pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type." These states are acted upon only if thread interruption is enabled. If a cancelation request is pending and the interrupt state or type is set to allow asynchronous cancelation requests, the thread is canceled before control is returned to the invoker.
6. See Appendix I, "Optimizing performance using process- and thread-level information," on page 1329.

Related services

- "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495
- "pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type"
- "pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur" on page 536

Characteristics and restrictions

There are no restrictions on the use of the pthread_setintr service.

Examples

For an example using this callable service, see "BPX1PSI (pthread_setintr) example" on page 1172.

pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type

Function

The pthread_setintrtype callable service sets the specified interruptability type of the calling thread and atomically returns the previous interruptability type.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PST):	31-bit
AMODE (BPX4PST):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PST,(Interrupt_type,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PST with the same parameters.

Parameters

Interrupt_type
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword containing a numeric value identifying the interrupt type to be set. The following constants, which are defined in BPXYCONS, define the valid states.

Constant	Description
PTHREAD_INTR_ASYNCHRONOUS#	When interruptability is enabled and the interruptability type is set to PTHREAD_INTR_ASYNCHRONOUS#, cancelation requests can be acted upon at any time.
PTHREAD_INTR_CONTROLLED#	When interruptability is enabled and the interruptability type is set to PTHREAD_INTR_CONTROLLED#, cancelation requests are held pending until a cancelation point is reached. See the usage notes for a definition of cancelation points.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the service returns the previous interrupt type, or -1 if the service did not complete.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_setintrtype service stores the return code. The pthread_setintrtype service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_setintrtype service can return the following value in the Return_code parameter:

Return code	Explanation
EINVAL	One of the parameters contains a value that is not valid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_setintrtype service stores the reason code. The pthread_setintrtype service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The default interrupt type for newly created threads and the initial thread is PTHREAD_INTR_CONTROLLED#. If a cancelation request is pending and the interrupt state is set to PTHREAD_INTR_AYNCHRONOUS#, the cancelation request is acted upon before control is returned to the invoker.
2. For more information about controlling cancelation requests, see the usage notes for "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527.
3. See Appendix I, "Optimizing performance using process- and thread-level information," on page 1329.

Related services

- "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495
- "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527
- "pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur" on page 536

Characteristics and restrictions

There are no restrictions on the use of the pthread_setintrtype service.

Examples

For an example using this callable service, see "BPX1PST (pthread_setintrtype) example" on page 1172.

pthread_tag_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data

Function

The pthread_tag_np service sets, queries, or both sets and queries the 65 bytes of thread tag data that is associated with the caller's thread.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTT):	31-bit
AMODE (BPX4PTT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTT,(New_Tag_Length,
              New_Tag_Ptr,
              Old_Tag_Length,
              Old_Tag_Ptr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

New_Tag_Length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains either 0 or the length of the new tag data that is pointed to by New_Tag_Ptr. If New_Tag_Length contains 0 and New_Tag_Ptr contains a nonzero value, the caller's thread tag data is cleared. If New_Tag_Ptr contains a nonzero value, New_Tag_Length must be in the range of 0 to 65. See the usage notes for more details.

New_Tag_Ptr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains either 0 or the address of a location that contains the new thread tag data. If New_Tag_Ptr contains 0, the caller's thread tag data is left unchanged. See the usage notes for more details.

Old_Tag_Length
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_tag_np service returns the length of the old (current) thread tag data that is returned to the caller.

Old_Tag_Ptr
Returned parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains either 0 or the address of a 65-byte area in which the pthread_tag_np service returns the old (current) thread tag data. If Old_Tag_Ptr contains 0, no thread tag data is returned to the caller and Old_Tag_Length remains unchanged. See the usage notes for more details.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_tag_np service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_tag_np service stores the return code. The pthread_tag_np service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The pthread_tag_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	One or more of the following conditions were detected: • All or part of the location that is specified by New_Tag_Ptr and New_Tag_Length was not addressable by the caller. • All or part of the 66 bytes at the location that is specified by Old_Tag_Ptr was not addressable by the caller. The following reason codes can accompany the return code: JRNewLocationErr, or JROldLocationErr.

Return_code	Explanation
EINVAL	New_Tag_Ptr was nonzero, but New_Tag_Length was not in the range of 0 to 65. The following reason code can accompany the return code: JRNewLenBad.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_tag_np service stores the reason code. The pthread_tag_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If New_Tag_Ptr contains a nonzero value and New_Tag_Length contains 0, the caller's thread tag data is cleared.
2. If New_Tag_Ptr contains 0, the caller's thread tag data is left unchanged and the value specified by New_Tag_Length is not validity checked.
3. If the caller attempts to query the thread tag data and the tag data has never been set or was cleared, no data is stored at the location that is specified by Old_Tag_Ptr and Old_Tag_Length is set to 0.
4. If New_Tag_Ptr is nonzero, Tag_Length must be in the range of 0 to 65. If it is not within range, the tag data is left unchanged and the pthread_tag_np service is unsuccessful.
5. Thread tag data is displayed with the DISPLAY OMVS command when 'PID=' option is specified. The thread tag data should be printable (EBCDIC) data.
6. When Old_Tag_Ptr is nonzero and the caller's thread has tag data associated with it (previously set and not cleared), the pthread_tag_np service stores the tag data (left justified) at the location that is specified by the caller, and Old_Tag_Length contains the length of the data that is stored.

Related services

None.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see “BPX1PTT (pthread_tag_np) example” on page 1175.

pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur

Function

The pthread_testintr callable service causes a cancellation point to occur. If a cancellation request is pending, the cancellation request is acted upon before pthread_testintr returns.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTI):	31-bit
AMODE (BPX4PTI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTI,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTI with the same parameters.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which pthread_testintr returns a `0` if the thread did not have any pending cancellation requests, or `-1` if pthread_testintr did not complete (the cancellation request was not tested).

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_testintr service stores the return code. The pthread_testintr service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the pthread_testintr service stores the reason code. The pthread_testintr service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If a cancellation request is pending when this service is requested, control is not returned.
2. Calling the pthread_testintr service does not affect the interrupt state or type.
3. For more information about this service, see the usage notes for "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527.

Related services

- "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495
- "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527
- "pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type" on page 530

Characteristics and restrictions

There are no restrictions on the use of the pthread_testintr service.

Examples

For an example using this callable service, see "BPX1PTI (pthread_testintr) example" on page 1173.

ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging

Function

The ptrace callable service provides information about another process and controls its running. Use this service in debugger programs to do breakpoint debugging.

Requirements

Operation	Environment
Authorization:	Problem Program, PSW key 8
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PTR):	31-bit
AMODE (BPX4PTR):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PTR,(Request,
              Process,
              Address,
              Data,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PTR with the same parameters. The Address, Data, and Buffer parameters are doublewords.

Parameters

Request
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one of the integer values that indicates the function requested. The request integer values are defined in the BPXYPTRC macro. See “BPXYPTRC — Map parameters for ptrace” on page 1018.

Process
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the process identifier of the process that is the target of the ptrace call, or `0` for the PT_TRACE_ME, PT_EXTENDED_EVENT, and PT_RECOVER requests.

Address
Supplied parameter

Type: Address or Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see Table 16 on page 545.

Data
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see Table 16 on page 545.

Buffer
Supplied parameter

Type: Address or Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a value that is identified by the option selected for the Request parameter. For a mapping of this parameter to the Request parameter options, see Table 16 on page 545.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ptrace service returns `0`; the requested value if the request is successful; or -1 if it is not successful. For more information about values that are returned for specific requests, see Table 17 on page 548. A value of -1 is sometimes returned when the request is successful. For example, if a general-purpose register contains a value of -1, a PT_READ_GPR request returns this value in the Return_value parameter.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ptrace service stores the return code. The ptrace service always returns Return_code, even if Return_value is not -1. A Return_code of `0` is returned for successful completion. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. For a mapping of these values to the various requests, see Table 17 on page 548. The ptrace service can return one of the following values in the Return_code parameter:

Table 13. Return codes for ptrace

Return_code	Explanation
EAGAIN	One or more resources are temporarily unavailable. Reissue the request at a later time.
ECHILD	The debugged process ended while a ptrace service request was running.
EFAULT	An address in the caller's process is incorrect. The following reason codes can accompany the return code: JRBadAddress, JRPtInvDbrAddress.
EINTR	The ptrace service request was interrupted by a signal for the caller.

Table 13. Return codes for ptrace (continued)

Return_code	Explanation
EINVAL	The request was not accepted, for one of the following reasons: • The length is larger than the maximum defined length. The maximum defined length is defined in the BPXYPTRC macro. • The length of the area that is to contain the results of a PT_LDINFO, PT_EXPLAIN, or PT_THREAD_INFO request (the return information buffer) is too small to contain all the required information. For PT_LDINFO and PT_EXPLAIN, increase the length up to the maximum defined length and reissue the request. For PT_THREAD_INFO, the required buffer length is returned. Reissue the request, using this returned buffer length. See Table 16 on page 545 for more information (the required length is returned to the Destination Address). • For the PT_CAPTURE request, the input address that is to be captured is not on a page boundary (4K). • For the PT_UNCAPTURE request, the input captured buffer address is not an address that was previously returned from a successful PT_CAPTURE request. • For the PT_BLOCKEDREQ request, some of the requests might not have completed successfully. The Reason_code is set to JRPtSomeBlkedFailed. Check the PtBRStatus field of the PtBRInfo block for each blocked request to determine which have failed. The following reason codes can accompany the return code: JRPtLDBufferTooSmall, JRBuffTooSmall, JRNotPage, JRPtBufNotFound, JRPtInvLength.

Table 13. Return codes for ptrace (continued)

Return_code	Explanation
EIO	The request was not accepted for one of the following reasons: • The caller is not running with PSW key 8. • An incorrect Request was specified. • For a PT_TRACE_ME or PT_ATTACH request, the target process is already being debugged. For a PT_REATTACH or PT_REATTACH2 request, the target process is not already being debugged. • For a PT_DETACH, PT_CONTINUE or PT_THREAD_SIGNAL request, the signal number that was supplied in the Data parameter is not a valid signal number. • An address in the target process is not valid. • A register number for a PT_READ_GPR, PT_WRITE_GPR, PT_READ_FPR, PT_WRITE_FPR, PT_READ_VR, or PT_WRITE_VR request is not defined. The register numbers are defined in the BPXYPTRC macro. • An attempt was made to store into a control register using the PT_WRITE_GPR request. • An attempt was made to store into the left half of the PSW using the PT_WRITE_GPR request. • The user area offset that was supplied with the PT_READ_U request is incorrect. • For the PT_TRACE_ME request, the parent of the debugged process (that is, the debugger) has ended. • For the PT_REATTACH or PT_REATTACH2 request, the original debugger has ended. • For the PT_THREAD_WRITE_FOCUS, PT_THREAD_HOLD, PT_THREAD_MODIFY and PT_THREAD_SIGNAL requests, the thread ID that was supplied is not valid. • For the PT_EXPLAIN request, an extended ptrace event is not in progress. • For the PT_EVENTS request, an attempt was made to add more extended events than the maximum number of events that was specified on the PT_EVENTS request. • The request is not supported while it is stopped for a local fork child, or for an extended event. • For the PT_CAPTURE request, the target process is running in a TSO address space. The following reason codes can accompany the return code: JRPtAttemptedCRStore, JRPtAttemptedPSW0Store, JRPtDbdParentTerm, JRPtDbrPidNotFound, JRPtDbrZombie, JRPtInvCallingMode, JRPtInvDbdAddress, JRPtInvFPRNumber, JRPtInvGPRNumber, JRPtInvNumberThreads, JRPtInvPtraceState, JRPtInvRequest, JRPtInvSignalNumber, JRPtInvUAreaOffset, JRPtOldDbrPidNotFound, JRPtThreadTerm, JRPtLightWeightTHID, JRPtThreadNotFound, JRPtTSO, JRPtRequestDenied, JRPtAsyncThread, JRPtNotXtdEvent, JRPtTooManyEvents, JRPTInvVRNumber.
EMVSSAF2ERR	For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the caller does not have the appropriate privileges to debug the target process. For information about appropriate privileges, see "Authorization" on page 8.
ENOMEM	There is not enough storage available to satisfy a PT_CAPTURE request.

Table 13. Return codes for ptrace (continued)

Return_code	Explanation
EPERM	Permission to issue the request is denied for one of the following reasons: • For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target process is restricted from being debugged. **Note:** For PT_REATTACH or PT_REATTACH2, it is more likely that EIO will be returned, because the target process is not already being debugged. However, in the unlikely event that a restricted process successfully issues a PT_TRACE_ME request, a PT_REATTACH or PT_REATTACH2 could return EPERM. If either of the following is true, the target process is restricted: – The target process is a system address space. For more information about system address spaces, see "MVS-related information" on page 564. – The target process is the INIT process, indicated by a process ID (PID) value of 1. • For the PT_READ_xxx, PT_WRITE_xxx, PT_CONTINUE (to continue at another address), PT_REGSET and PT_LDINFO requests, the target process is currently running in supervisor state. The following reason codes can accompany the return code: JRPtRestrictedProcess, JRPtEdIsAuthorized.
ESRCH	The request was not accepted, for one of the following reasons: • For all requests other than PT_TRACE_ME, PT_ATTACH, PT_REATTACH, PT_REATTACH2, PT_EXTENDED_EVENT, and PT_RECOVER, the target process is not being debugged. • For all requests other than PT_TRACE_ME, PT_ATTACH, PT_REATTACH, PT_REATTACH2, PT_EXTENDED_EVENT, and PT_RECOVER, the target process is not stopped for a ptrace service event. • For all requests other than PT_TRACE_ME, PT_EXTENDED_EVENT, and PT_RECOVER, the target process ID is incorrect. • For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target debugged process is the same as the debugger process. • For the PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests, the target debugged process is the parent of the debugger process. The following reason codes can accompany the return code: JRPtDbdEqualsDbr, JRPtDbdPidNotFound, JRPtProcessNotPtraced, JRPtProcessNotStopped, JRPtDbrParentEqualsDbd.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ptrace service stores the reason code. The ptrace service always returns Reason_code, even if Return_value is not -1. A

Reason_code of 0 is returned for successful completion. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF ptrace Authority Check service return and reason code values, see the following table:

Table 14. RACF ptrace authority check service return and reason code values

RACF return code	RACF reason code	Explanation
8	4	The caller is not authorized to attach to the target process
8	12	Internal error during RACF processing

Constant options for the ptrace request parameter

Table 15 shows the constant options that you can select for the Request parameter. See "BPXYPTRC — Map parameters for ptrace" on page 1018 for the constant definitions.

Table 15. Constant options for the ptrace request parameter

Constant	Explanation
PT_ATTACH	Enable a target process to be debugged with the ptrace service.
PT_CAPTURE	Capture one or more pages of storage in the target debugged process into a buffer in the caller's address space.
PT_CONTINUE	Continue running the debugged process.
PT_DETACH	Disable debugging for the target process.
PT_EVENTS	Enable or disable reporting for an extended event.
PT_EXPLAIN	Return additional information about an extended event.
PT_EXTENDED_EVENT	Notify the debugger of an extended event. For more information, see "Handling a program check or abend in a debugged process" on page 555.
PT_KILL	End the debugged process.
PT_LDINFO	Return information about modules that were loaded by the debugged process.
PT_MULTI	Turn multiprocess debugging mode on or off. For information about multiprocess debugging, see "Multiprocess debugging mode" on page 560.
PT_BLOCKREQ	Several Ptrace request types are blocked together into a single Ptrace call.
PT_READ_BLOCK	Read a block of storage.
PT_READ_D	Return a fullword of data from a specified address in the debugged process. This request reads program data.
PT_READ_FPR	Return the value of a floating-point register.

Table 15. Constant options for the ptrace request parameter (continued)

Constant	Explanation
PT_READ_GPR	Return the value of a general-purpose or machine-control register. The value includes the PSW and control registers, as well as general-purpose registers.
PT_READ_GPRH	Read a specific general-purpose high register.
PT_READ_I	Return a fullword of data from a specified address in the debugged process. This request reads program instructions.
PT_READU	Return the value of a fullword of control information from the user area in the debugged process. For more information, see "User area description" on page 561.
PT_READ_VR	Return the 16-byte value of a vector register.
PT_WRITE_VR	Change the 16-byte value of a vector register.
PT_REATTACH	Enable a target process to be debugged with the ptrace service by a new debugger. The relationship between the target process and its original debugger is removed.
PT_REATTACH2	Enable a target process to be debugged with the ptrace service by a new debugger. The relationship between the target process and its original debugger is removed. This request is an extension of the PT_REATTACH request, and must be used by a debugger to deal with the local fork child environment. For more information, see "Attaching to a process for debugging" on page 551.
PT_RECOVER	Notify the debugger of a program check interrupt or abnormal end. For more information, see "Handling a program check or abend in a debugged process" on page 555.
PT_REGHSET	Read all of the general-purpose high registers.
PT_REGSET	Return the values of all general-purpose registers.
PT_THREAD_HOLD	Hold or unhold a thread in the debugged process.
PT_THREAD_INFO	Return kernel information on all threads in the debugged process.
PT_THREAD_MODIFY	Modify a thread's kernel information.
PT_THREAD_READ_FOCUS	Return the current focus thread ID.
PT_THREAD_SIGNAL	Queue a signal to a thread in the debugged process.
PT_THREAD_WRITE_FOCUS	Change the current focus thread ID.
PT_TRACE_ME	Enable the calling process to be debugged with the ptrace service.

Table 15. Constant options for the ptrace request parameter (continued)

Constant	Explanation
PT_UNCAPTURE	Free one or all buffers that contain captured storage from previous PT_CAPTURE requests.
PT_WRITE_BLOCK	Change the contents of a block of storage.
PT_WRITE_D	Change a fullword of data at a specified address in the debugged process. This request changes program data.
PT_WRITE_FPR	Change the value of a floating-point register.
PT_WRITE_GPR	Change the value of a general-purpose or machine-control register. The value includes the PSW and control registers, as well as general-purpose registers.
PT_WRITE_GPRH	Write to a specific general-purpose high register.
PT_WRITE_I	Change a fullword of data at a specified address in the debugged process. This request changes program instructions.

Parameter attributes for request options

Table 16 shows the ptrace service options for the Request parameter. For each option, the meanings of the Address, Data, and Buffer parameters are shown. Explanations of the terms in the table follow the table:

Table 16. Parameter attributes for request options

Request options	Address	Data	Buffer
PT_ATTACH	0	0	0
PT_CAPTURE	Capture Address	Capture Length	0
PT_CONTINUE	1 = Continue from where process stopped Not 1 = Continue Address	0 = No signal Not 0 = Signal Number	0
PT_DETACH	0	0 = No signal Not 0 = Signal Number	0
PT_EVENTS	Extended Event Id	0 = Disable reporting this event Not 0 = Enable reporting this event	Maximum Events
PT_EXPLAIN	Buffer Address (destination)	Length	0

Table 16. Parameter attributes for request options (continued)

Request options	Address	Data	Buffer
PT_EXTENDED_EVENT	GIParm Address	Extended Event ID	Destination Address (4 bytes)
PT_KILL	0	0	0
PT_LDINFO	Buffer Address (Destination)	Length	0
PT_MULTI	0	0 = Reset multi-process mode Not 0 = Set multiprocess mode	0
PT_BLOCKREQ	Buffer Address (source / destination)	Length	Buffer Address (destination)
PT_READ_BLOCK	Debugged Address	Length	Buffer Address
PT_READ_D	Debugged Address	0	0
PT_READ_FPR	Destination Address	Register Number	0
PT_READ_GPR	Register Number	0	0
PT_READ_GPRH	Register Number	0	0
PT_READ_I	Debugged Address	0	0
PT_READ_U	Target Offset	0	0
PT_READ_VR	Destination address	Register number	0
PT_REATTACH	0	0	0
PT_REATTACH2	0	0	Destination Address
PT_RECOVER	PCParm Address	0	0
PT_REGHSET	Destination Address	0	0
PT_REGSET	Destination Address	0	0
PT_THREAD_HOLD	Thread ID Address	0 = Unhold thread Not 0 = Hold thread	0
PT_THREAD_INFO	Buffer Address	Length	Destination Address
PT_THREAD_MODIFY	Thread ID Address	0	Source Address
PT_THREAD_READ_FOCUS	Thread ID Address	0	0
PT_THREAD_SIGNAL	Thread ID Address	Signal Number	0
PT_THREAD_WRITE_FOCUS	Thread ID Address	0	0
PT_TRACE_ME	0	0	0
PT_UNCAPTURE	0 = Free all buffers Not 0 = Capture Buffer	0	0
PT_WRITE_BLOCK	Debugged Address	Length	Buffer Address
PT_WRITE_D	Debugged Address	Integer Value	0
PT_WRITE_FPR	Source Address	Register Number	0
PT_WRITE_GPR	Register Number	Register Value	0
PT_WRITE_GPRH	Register Number	Register Value	0

Table 16. Parameter attributes for request options (continued)

Request options	Address	Data	Buffer
PT_WRITE_I	Debugged Address	Integer Value	0
PT_WRITE_VR	Source address	Register number	0

Buffer Address
The name of a fullword that contains an address in the caller's process where either:

- The results of the request are to be placed
- The source information for the request is to be obtained

The size of the buffer is specified with the Length parameter.

Capture Address
The name of a fullword that contains an address in the target process that is to be captured into a buffer in the caller's address space. This address must be on a page boundary (4K).

Capture Buffer
The name of a fullword that contains an address in the caller's process that represents a captured storage buffer. This address must have been previously returned to the caller on a PT_CAPTURE request.

Capture Length
The name of a fullword that contains the length of the storage that is to be captured. There is no need to round this length up to the size of a page.

Continue Address
The name of a fullword that contains an address in the target process from which the debugged program is to continue running. The address must include the addressing mode (AMODE) as the high-order bit. A high-order bit of 0 indicates a 24-bit AMODE; a high-order bit of 1 indicates a 31-bit AMODE.

The PT_CONTINUE request can indicate a value of 1 instead of an address that indicates where continuation should begin. This value, which is defined in the BPXYPTRC macro, indicates that the program should continue from where it stopped.

Debugged Address
The name of a fullword that contains an address in the target process.

Destination Address
The name of a fullword that contains an address in the caller's process at which the results of the request are to be placed. The size of the destination area is defined by the request type.

Extended Event ID
The name of a fullword that contains an extended event ID.

Integer Value
The name of a fullword that contains the value that is to be placed at the Debugged Address location.

GIParm Address
The name of a fullword that contains the address of the generic interface parameters. For more information, see “Handling extended events in a debugged process” on page 556.

Length
The name of a fullword that contains the length that is associated with the

Buffer Address. The maximum length value is defined in the BPXYPTRC macro, except for the PT_THREAD_INFO request.

Maximum Events
The name of a fullword that contains the maximum number of extended events that will be added using the PT_EVENTS request. This is required only for the first issuance of PT_EVENTS, but it can be specified on all issuances.

PCParm Address
The name of a fullword that contains the address of the program check parameters. For more information, see "Handling a program check or abend in a debugged process" on page 555.

Register Number
The name of a fullword that contains a defined register number. The register numbers are defined in the BPXYPTRC macro.

Register Value
The name of a fullword that contains the register value that is to be placed in the Register Number in the target process.

Signal Number
The name of a fullword that contains the signal number that is to be sent to the target debugged process or thread. The signal numbers are defined in the BPXYSIGH macro.

Source Address
The name of a fullword that contains an address in the caller's process where the source information for the request is to be obtained. The size of the source area is defined by the request type.

Target Offset
The name of a fullword that contains an offset into the user area in the target process. The user area contains control information. For a description of the user area, see "User area description" on page 561.

Thread ID Address
The name of a fullword that contains an address in the caller's process where either:
- The target thread ID is to be placed
- The target thread ID is to be obtained

The length of the thread ID is 8 bytes.

Return values and return codes for request options

Table 17 shows the ptrace service requests. For each request, the value that is returned in the Return_value parameter is shown. Possible values returned in the Return_code parameter are also shown.

Table 17. Return values and return codes for request options

Request	Return_value	Return_code
(General)	0	EFAULT, EIO, EMVSERR
PT_ATTACH	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH

Table 17. Return values and return codes for request options (continued)

Request	Return_value	Return_code
PT_BLOCKREQ	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH
PT_CAPTURE	Capture buffer address	EINVAL, EIO, ENOMEM
PT_CONTINUE	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_DETACH	0	EAGAIN, ECHILD, EINTR, EIO, ESRCH
PT_EVENTS	0	ESRCH
PT_EXPLAIN	0	EFAULT, EIO, ESRCH
PT_EXTENDED_EVENT	0	EFAULT
PT_KILL	0	EAGAIN, ECHILD, EINTR, ESRCH
PT_LDINFO	0	EAGAIN, ECHILD, EFAULT, EINTR, EINVAL, EPERM, ESRCH
PT_MULTI	0	ESRCH
PT_READ_BLOCK	Value of Data parameter	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH
PT_READ_D	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_READ_FPR	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
PT_READ_GPR	Register contents	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_READ_GPRH	Register contents	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_READ_I	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_READ_VR	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
PT_READ_U	Fullword value	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_REATTACH	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH

Table 17. Return values and return codes for request options (continued)

Request	Return_value	Return_code
PT_REATTACH2	0	EAGAIN, ECHILD, EIO, EMVSSAF2ERR, EPERM, ESRCH
PT_RECOVER	0	EFAULT
PT_REGHSET	0	EAGAIN, ECHILD, EFAULT, EINTR, EPERM, ESRCH
PT_REGSET	0	EAGAIN, ECHILD, EFAULT, EINTR, EPERM, ESRCH
PT_THREAD_HOLD	0	EFAULT, EIO, ESRCH
PT_THREAD_INFO	0	EFAULT, EINVAL, ESRCH
PT_THREAD_MODIFY	0	EFAULT, EINVAL, EIO, ESRCH
PT_THREAD_READ_FOCUS	0	EFAULT, ESRCH
PT_THREAD_SIGNAL	0	EFAULT, EIO, ESRCH
PT_THREAD_WRITE_FOCUS	0	EFAULT, EIO, ESRCH
PT_TRACE_ME	0	EAGAIN, EIO
PT_UNCAPTURE	0	EINVAL
PT_WRITE_BLOCK	Value of Data parameter	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EINVAL, EPERM, ESRCH
PT_WRITE_D	0	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_WRITE_FPR	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH
PT_WRITE_GPR	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_WRITE_GPRH	Value of Data parameter	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_WRITE_I	0	EAGAIN, ECHILD, EINTR, EIO, EPERM, ESRCH
PT_WRITE_VR	0	EAGAIN, ECHILD, EFAULT, EINTR, EIO, EPERM, ESRCH

Starting a process in debugging mode

Typically, a debugger program starts a process to be debugged by calling the fork service to create a child copy of the debugger program. The child then calls the

ptrace service with a PT_TRACE_ME request. This puts the child process into debugging mode. Next, the child calls the exec service to run the program to be debugged.

The PT_TRACE_ME request, along with PT_EXTENDED_EVENT and PT_RECOVER, is issued from the process to be debugged. All other ptrace service requests are issued from the debugger. It is also assumed that the parent of the process issuing a PT_TRACE_ME request is the debugger.

Attaching to a process for debugging

The ptrace service also provides a means for a debugger program to debug an already running, possibly unrelated, process. To do this, the debugger calls ptrace with a PT_ATTACH service request. There are certain restrictions on which processes can be attached (see "Characteristics and restrictions" on page 562). The caller must have the appropriate privileges (see "Authorization" on page 8) to attach to a running process.

The PT_REATTACH request performs a similar function, but is intended to be used in a situation where another debugger process is already attached to the target process. The PT_REATTACH request causes the relationship between the other debugger and the target process to be severed. The caller of PT_REATTACH becomes the new debugger associated with the target process. The PT_REATTACH2 request is identical to PT_REATTACH except in one respect. PT_REATTACH2 provides the address of an area in which return information concerning the reattach can be placed. If the PT_REATTACH2 request is issued against the child process that was created with an attach_exec or attach_execmvs service (a local fork child), the return information is nonzero, indicating to the debugger that alternate reattach processing is required. Otherwise, the return information is zero, telling the debugger that reattach processing should be the same as for PT_REATTACH.

This is required because of the restricted nature of the local fork child environment. Normally, after a reattach to a fork child, the debugger can continue issuing ptrace requests, because the child is a copy of the parent and is already stopped for a ptrace event. However, a local fork child is not a copy of the parent, and in fact most ptrace requests fail if issued to a local fork child, because the target program that is specified on the attach_exec or attach_execmvs service has not yet been loaded for execution. The alternate processing required by the debugger for this environment is to issue a PT_CONTINUE request, followed by a wait(). This causes the local fork child to continue until the next ptrace event, which will be an exec event for the attach_exec service. At this point, the target program is loaded, and the debugger can continue issuing ptrace requests in this valid environment.

Note: ptrace has no way to know whether the target of the PT_REATTACH2 is the local fork parent or the local fork child. It is the debugger's responsibility to issue the request for the local fork child only. The result of issuing this request for the local fork parent is that control of the parent is lost; it will continue running until the next ptrace event.

Here is an example of using the PT_REATTACH or PT_REATTACH2 request: Debugger 1 is currently debugging program A in multiprocess mode. For information about multiprocess debugging, see "Multiprocess debugging mode" on page 560. Program A uses the fork service to create a child process, which becomes program B. Debugger 1 is informed of the fork from both the parent (program A)

and child (program B) processes. Debugger 1 uses the fork service to create a new debugger, which becomes debugger 2. Debugger 2 then uses the PT_REATTACH request to associate itself with program B. At this point, debugger 1 is debugging program A, and debugger 2 is debugging program B.

The PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests cause a SIGTRAP signal to be sent to the target process. This causes a ptrace service signal event to occur if no other event occurs naturally.

Receiving notification of events in a debugged process

When a process has been placed into debugging mode by the PT_TRACE_ME, PT_ATTACH, PT_REATTACH, or PT_REATTACH2 request, certain events in the debugged process cause the process to be placed into a stopped state and the debugger to be notified. The debugger must wait for these events by using the wait service. The following are the events of interest:

- A signal is received. The Status_field parameter on the wait service issued by the debugger contains the signal number.
- An exec service is issued. The Status_field parameter on the wait service issued by the debugger either contains the SIGTRAP signal number, if multiprocess debugging is not in effect; or indicates that the process stopped for an exec (WastStopFlagExec), if multiprocess debugging is in effect. For information about multiprocess debugging, see “Multiprocess debugging mode” on page 560. Also see “BPXYWAST — Map the wait status word” on page 1069 for a description of the Wast values.
- A fork service is issued and multiprocess debugging mode is in effect. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped for a fork (WastStopFlagFork).
- An attach_exec or attach_execmvs service call is issued and multiprocess debugging mode is in effect. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped for a local fork (WastStopFlagLocalFork).
- A spawn service call is issued and multiprocess debugging mode is in effect. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped for a fork (WastStopFlagFork) for the spawn parent, and for a local fork (WastStopFlagLocalFork) for the spawn child.
- An SVC 144 instruction is run. SVC 144 is used as a breakpoint by debugger programs. The debugger uses the ptrace service to store the SVC 144 instructions into the program at the appropriate breakpoints. The Status_field parameter on the wait service issued by the debugger contains the SIGTRAP signal number.
- A program check or abnormal end is encountered. The debugger is notified only if the program check or abnormal end causes the ptrace service to be called with a PT_RECOVER request. This is normally true for programs that detect the error, and that can provide the proper interface to the PT_RECOVER request. An ESPIE routine is an example of this. The Status_field parameter on the wait service issued by the debugger contains the appropriate signal number. For more information, see “Handling a program check or abend in a debugged process” on page 555.
- An extended event occurs and a generic debugger interface module issues the ptrace PT_EXTENDED_EVENT request. Extended events are enabled by using the PT_EVENTS service request. Only those events thus enabled cause a ptrace extended event to occur. “Handling extended events in a debugged process” on page 556 provides more information about the generic debugger interface. The debugger must use the PT_EXPLAIN request to obtain additional information

about the extended event. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped for an extended event (WastStopFlagExtended).

- A loadhfs service is issued. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped because of a file system module load (WastStopFlagLoad).
- A deletehfs service is issued. The Status_field parameter on the wait service issued by the debugger indicates that the process stopped because of a file system module delete (WastStopFlagDelete).

Table 18 summarizes the events and the corresponding status reported to the debugger from the wait call:

Table 18. Corresponding ptrace event and status that is reported to the debugger

ptrace event	Debugger wait service Status_field parameter
Signal received	Signal number
exec service issued	SIGTRAP signal number or WastStopFlagExec
fork service issued	WastStopFlagFork
attach_exec or attach_execmvs service issued	WastStopFlagLocalFork
spawn service issued	WastStopFlagFork (parent), WastStopFlagLocalFork (child)
SVC 144 instruction performed	SIGTRAP signal number
Program check or abend encountered	SIGILL, SIGSEGV, SIGFPE, or SIGABND signal number
Extended event encountered	WastStopFlagExtended

When a process has multiple threads, any thread that encounters one of the ptrace service events causes the process to enter a stopped state. This is accomplished by synchronously suspending all other threads in the process. The thread on which the event occurred is known as the *focus thread*. Because delays could occur between the time the focus thread encounters the ptrace event and the time all the nonfocus threads are suspended, one or more of these other threads could encounter the same or other ptrace events. For instance, several threads could reach a breakpoint in a routine that is common to them all. This creates a situation in which the focus thread is "in control" of ptrace processing, but other ptrace service events are pending. See “Working with threads in a debugged process” on page 554 for more information about handling threads in a debugged process.

While the debugged process is stopped for one of the foregoing events, the debugger can issue ptrace service requests to examine or modify registers, storage, and so on. Most ptrace service requests are issued while the debugged process is stopped for the ptrace service event. Examples are: PT_LDINFO and PT_READ_U. An event ends when a PT_CONTINUE, PT_DETACH, or PT_KILL request is issued. One exception to this is a PT_CONTINUE request with a signal that stops the debugged process (for instance, SIGSTOP). In this case the original event does not end until a PT_CONTINUE is issued with either no signal or a SIGCONT. This is because of the ambiguous nature of a "continue and stop" request. The debugged process does not actually continue running until it is taken out of the stopped

state, either explicitly by a PT_CONTINUE with SIGCONT request, or implicitly by a PT_CONTINUE with no signal.

Working with threads in a debugged process

Several ptrace service requests can assist debuggers in handling multiple threads. The PT_THREAD_INFO service request returns a list of threads and kernel information about each thread, such as its state (active, dead, and so on) and kernel attributes. The PT_THREAD_READ_FOCUS service request returns the current focus thread ID. These ptrace service requests allow the debugger to gather various thread-related information whenever the debugger is awoken for a ptrace service event.

Certain debugger objectives require exact control over which thread or threads are running at any given time. For example, if the debugger wants to single-step the focus thread, the focus thread must be the only thread in the target process that is running. If this is not so, unpredictable results could occur. The PT_THREAD_HOLD service request allows the debugger to selectively place any threads that are not in a dead state into a held state. When a thread is held, it does not run until it is released. The debugger could therefore use this request to hold all but the focus thread, and then single-step the focus thread by inserting breakpoints after each program statement. The PT_THREAD_HOLD service request can also be used to release threads.

The debugger might also want to work with threads other than the current focus thread. An example might be if the current focus thread manipulates data in shared storage that is then acted upon by a different thread. In order to work with this other thread, the debugger must release the thread, and then shift focus to it by using the PT_THREAD_WRITE_FOCUS service request. This request causes the specified thread to become the new focus thread if it is in an active, non-asynchronous state. Other ptrace service requests that read or write storage, registers, and other data always act against the current focus thread, so this is the means by which the debugger specifies which thread is the target of these other requests.

Signals can be sent to the debugged process using the PT_CONTINUE and PT_DETACH requests, as explained “Resuming or detaching from a debugged process.” on page 559. These signals are always directed at the current focus thread. The PT_THREAD_SIGNAL service request provides a means to send signals to individual threads that are not in a dead state. The signal that is specified on this request is generated for the target thread, but it is not delivered until the process is continued. A debugger can use this request to requeue signals that were pending on a thread when a ptrace event occurred. Normally, all signals (except those specified on PT_CONTINUE or PT_DETACH) that are pending on the focus thread are discarded when the process is continued. Using the signal information that is returned on the PT_THREAD_INFO request, the debugger could regenerate all pending signals by using the PT_THREAD_SIGNAL request. This is useful when changing thread focus; otherwise, any signals that were pending on nonfocus threads when the original ptrace event occurred might be lost.

Note: To ensure that pending signal information for the focus thread is not lost (because the signals were discarded), the PT_THREAD_INFO request should be the first ptrace request that is issued when the debugger gets notified of a ptrace service event.

Finally, the PT_THREAD_MODIFY service request allows the debugger to modify individual thread kernel information for a thread that is in a dead state. The input to this request is a single thread info array entry, as returned by the PT_THREAD_INFO request, modified as necessary with the required changes. The following information is allowed to be changed:

- Thread exit status for threads in a dead state (PtPtExitStatus)

Determining modules loaded in a debugged process

If the debugger needs to determine the names and entry points of modules that are loaded into the debugged process, it uses the ptrace PT_LDINFO service request. A structure is returned to the debugger that contains information about loaded modules, including the name of the directory that contains the load module for each module loaded from the file system. (The directory name is not returned for modules loaded from MVS data sets.) One use for this information is to read the load module library file to obtain symbolic debugging information. The returned structure is defined in the BPXYPTRC macro. For more information about PT_LDINFO, see "MVS-related information" on page 564.

Handling a program check or abend in a debugged process

When program checks or abnormal ends occur in a debugged process and are captured by the program's recovery routine (such as an ESPIE or ESTAE exit), the PT_RECOVER request can be issued. This request allows the ptrace service to stop the process and notify the debugger that a program check or abnormal end has just occurred. The caller does not need to determine if the process is being debugged; it can issue the PT_RECOVER request unconditionally. If the process is not being debugged, the returned information indicates to the caller that it can continue as it normally would. The returned information contains PtPICFlags, which are all zeros on return if the process is not being debugged.

However, if the process is being debugged, the ptrace service stops the process by sending an appropriate signal to the focus thread. The debugger (not the application) sets this signal with the Status_field parameter on the wait call. These are the possible signals that are used for program checks:

SIGILL

Unpermitted operation, defined as one of the following:

- Operation exception
- Privileged operation exception
- Execute exception
- Specification exception

SIGSEGV

Addressing error, defined as one of the following:

- Protection exception
- Addressing exception

SIGFPE

Arithmetic error, defined as one of the following:

- Data exception
- Fixed-point overflow exception
- Fixed-point divide exception
- Decimal overflow exception
- Decimal divide exception
- Exponent overflow exception
- Exponent underflow exception

- Significance exception
- Floating-point divide exception

The interface to the PT_RECOVER request is the program check parameters structure, which is pointed to from the Address parameter. This structure contains pointers to environment information from the time of the program check or abnormal end. It also contains flags and information that is returned from the ptrace service. The program check parameters structure is defined in the BPXYPTRC macro. The ptrace service distinguishes between program checks and other abnormal ends. The presence of an abend code causes the interrupt to be interpreted as a SIGABND signal. The presence of an interrupt code without an abend code causes it to be interpreted as SIGILL, SIGSEGV, or SIGFPE, as appropriate. The following shows the information that must be present in the program check parameters structure on input to PT_RECOVER:

1. In all cases the following must be set:
   - PtPICRegisters = address of registers at time of error (0 - 15)
   - PtPICPSW = address of PSW at time of error
   - PtPICFlags = 0 (except PtPICILCExists if PtPICILC is set)
2. For program checks, the following must be set:
   - PtPICIntCode = program interrupt code
   - PtPICAbendCode = 0
3. For program checks, the following are optional:
   - PtPICILC = instruction length code
   - PtPICILCExists = flag set to 1 to indicate PtPICILC is valid
4. For non-program check abnormal ends, the following must be set:
   - PtPICAbendCode = abend code
   - PtPICAbendReason = reason code
   - PtPICILCExists = flag set to 0 to indicate PtPICILC is not used
5. For any program check that is not specified in the foregoing list, SIGFPE is used. For abnormal ends (abends) SIGABND is used.

   The environment information (registers and PSW) can be modified by the debugger by using appropriate ptrace service requests while the debugged process is stopped for the ptrace service event. When the program is continued with the PT_CONTINUE or PT_DETACH request, you need to ensure that any modifications are reflected in the operating environment when the program resumes control.

   When the registers or the PSW are changed by the debugger, the appropriate flag in the program check parameters, as defined in the BPXYPTRC macro, is set to indicate this. Conversely, if the registers or PSW are not changed, the flag is not set. Thus the caller can test these flags to determine if changes were made to the registers or the PSW, and therefore need to be reflected in the program environment before the program resumes running. See “MVS-related information” on page 564.

   Note: The PT_RECOVER request, along with PT_TRACE_ME and PT_EXTENDED_EVENT, is issued from the process that is to be debugged. All other ptrace service requests are issued from the debugger.

Handling extended events in a debugged process

Language Environment supports a generic debugger interface for the high-level languages it supports, such as C. This interface requires that a module named CEEEVDBG be available for Language Environment to load and call when certain events occur in a high-level language program that has had this interface enabled

via a TEST run time option. The input to CEEEVDBG is a parameter list that contains an event code and information that is associated with that code.

To allow events that are invoked in this manner to be used by a debugger using ptrace, z/OS UNIX ships a sample CEEEVDBG module. This module "glues" the Language Environment Interactive Debug Event Handler interface to the debugger, using the ptrace PT_EXTENDED_EVENT request to create an extended event. However, any product can choose to use the PT_EXTENDED_EVENT request to create extended events in this manner; CEEEVDBG is just an example of the proper usage. The relationship between the PT_EXTENDED_EVENT, PT_EVENTS and PT_EXPLAIN requests can best be illustrated with a usage scenario:

1. The Language Environment Interactive Debug Event Handler interface is enabled for the program to be debugged. Refer to *z/OS Language Environment Debugging Guide* for the steps that are required to accomplish this, as this is outside the scope of ptrace.
2. The sample CEEEVDBG module is installed so that Language Environment can load it. More information is provided in "MVS-related information" on page 564.
3. The debugger issues one or more PT_EVENTS requests to establish the set of extended events for which it has an interest. This should normally be done just after the target program has been placed into debugging mode, during debugger initialization regarding the debugged program. There are, however, no restrictions on modifying the list of extended events any time the debugged program is stopped for an event.
4. The program is allowed to run. When Language Environment encounters certain events (for example, a mutex initialization, lock, or unlock), it invokes CEEEVDBG with the appropriate event code.
5. CEEEVDBG collects certain information about the event and issues the PT_EXTENDED_EVENT request to invoke ptrace. The information that is collected consists of the event code and registers 1, 12 and 13 at input to CEEEVDBG. Register 1 contains the address of the parameter list that contains the event code and associated information. Registers 12 and 13 contain the addresses of Language Environment control blocks that the debugger can use to gather additional information. The extended event information structure is defined in the BPXYPTRC macro.
6. The PT_EXTENDED_EVENT request filters the input event code with the set of events established with the PT_EVENTS requests. If the input event code is found in the list, an Extended Event is initiated. This causes the debugged program to stop and the debugger to be notified. The corresponding wait() status reported to the debugger is WastStopFlagExtended.
7. The debugger reacts to the unique wait() status by issuing the PT_EXPLAIN request. This request returns the information collected by the PT_EXTENDED_EVENT request to the debugger.
8. Because the information is in the form of addresses, the debugger must issue PT_READ_D or PT_READ_BLOCK requests to obtain the associated extended event information.

Manipulating data in a debugged process

You can use the ptrace service to look at the following types of data in the debugged process; some might be altered.

1. **General or machine control registers.** This includes general-purpose registers (GPRs), floating-point registers (FPRs), vector registers (VR), control registers (CRs), and the program status word (PSW). Control registers can only be

looked at, never modified. Control registers contain system information, and their content is not necessarily related to the debugged process. The value of some of the control registers might change from one call of PTRACE to the next, even when the debugged process is stopped across both calls. The entire PSW can be looked at, but only the rightmost 4 bytes (the instruction counter and addressing mode) can be changed. The PT_READ_GPR and PT_WRITE_GPR requests are used for all registers except the FPRs, and the interface supports 4 bytes only. As a result, the PSW must be accessed with two ptrace service requests, each specifying the register number for the appropriate half of the PSW.

The PT_READ_VR and PT_WRITE_VR requests are used for the vector registers, and this interface supports 16 bytes. All the register numbers are defined in the BPXYPTRC macro.

Restriction: Only the second fullword of the PSW can be written into.

The PT_READ_FPR and PT_WRITE_FPR requests are used for the FPRs, and this interface supports 8 bytes. In addition to reading and writing the floating point registers, you can also read and write the floating point control register. All the register numbers are defined in the BPXYPTRC macro.

Two special cases exist. One is the PT_REGSET request, which returns all the general-purpose registers. The second is the PT_CONTINUE request, which can indicate that the program should continue at a specified address. In other words, that the instruction counter should be modified.

2. **User program storage.** This takes two forms. The first is for fullword requests, which look at or modify a fullword of storage only. The PT_READ_I, PT_READ_D, PT_READ_U, PT_WRITE_I, and PT_WRITE_D requests are used to accomplish this. For MVS considerations, see "MVS-related information" on page 564. The user area request (PT_READ_U) operates on the user area. For more information, see "User area description" on page 561.

   The second form is for blocks of storage, up to a defined maximum length. For this, the PT_READ_BLOCK and PT_WRITE_BLOCK requests are used. The maximum defined length is defined in the BPXYPTRC macro.

3. **Blocking requests.** Most of the requests described here can be blocked into a single ptrace call by using the PT_BLOCKREQ request. This saves system resources when a large amount of information must be read or written. The PT_BLOCKREQ request can be used, for example, to read or write all the GPRs, all the FPRs, and several areas of user program storage on a single request. The PtBRInfo structure, defined in macro BPXYPTRC, defines the mechanism for blocking several requests into a single request.

Setting a breakpoint in a debugged process

You can use the PT_WRITE_I (or PT_WRITE_D or PT_WRITE_BLOCK) request to store SVC 144 instructions into a debugged program. The SVC 144 instruction causes an SVC 144 event to be recognized by the debugger. See "MVS-related information" on page 564 for MVS considerations regarding the use of SVC 144.

Note: It is the responsibility of the caller to save and restore the actual program instructions that are overlaid by inserted breakpoint SVCs. You can use ptrace services to accomplish this, but no implicit understanding or management of program instructions is done by the ptrace service.

Capturing storage in a debugged process.

When you use the standard ptrace requests previously discussed, you pay a performance penalty when you perform certain operations. For example, stepping over a breakpoint instruction while leaving the breakpoint in the program requires several ptrace requests, as in the following scenario:

- PT_WRITE_I to restore the original instruction over the SVC 144
- PT_READ_I to get the fullword following the restored instruction
- PT_WRITE_I to insert a temporary SVC 144 after the restored instruction
- PT_WRITE_GPR to back up the PSW to point to the restored instruction
- PT_CONTINUE to execute the restored instruction and hit the temporary breakpoint
- PT_WRITE_I to restore the temporarily overlaid instruction
- PT_WRITE_I to reinsert the SVC 144 at its original location
- PT_WRITE_GPR to back up the PSW to point to the restored temporary instruction
- PT_CONTINUE to resume running until the next event

Each of these ptrace requests consumes system resources and requires some amount of time to complete. The cumulative effect might be performance that is slower than expected.

The PT_CAPTURE request allows you to capture one or more virtual pages of storage in the debugged process into a buffer in your address space. After capturing storage in this manner, you have shared write access to the storage, and can access it directly by accessing the returned buffer. This allows you to bypass those ptrace requests that would normally be used to read or write storage in the debugged process. One use for the PT_CAPTURE request could be to capture the entire debugged program load module. Then, using the same example of stepping over a breakpoint instruction, you could eliminate all but the PT_WRITE_GPR and PT_CONTINUE requests by directly placing SVC 144 instructions and restoring program instructions in the captured buffer. Any storage that is accessible by the debugged program can be captured in this manner.

Storage that is captured using the PT_CAPTURE request is always on a 4K page boundary, and the minimum amount of storage captured is one 4K page. You are responsible for determining the correct offset of the desired storage in the captured buffer. For example, if the address you want to capture is 3A094BE8, the PT_CAPTURE request captures the entire page starting at 3A094000. If the service returns a capture buffer address of 35081000, the start of the desired storage in this buffer is 35081BE8.

The PT_UNCAPTURE request is used to free a specific buffer or all captured buffers. Freeing the buffer by using this request severs the capture relationship between the captured storage and the local buffer. To free a specific buffer, pass the buffer address on the ptrace request. To free all buffers, pass a 0 buffer address.

Resuming or detaching from a debugged process.

To cause a stopped, debugged process to resume running, you use the ptrace PT_CONTINUE service request. The request specifies whether running is to continue from where it was stopped, or at another instruction counter address. It also specifies whether the process is to continue as though no signal, or a specified signal, had just been received.

These two functions of the PT_CONTINUE request can be used to accomplish several debugging objectives. For instance, if the debugged program was stopped by a particular signal (for instance, SIGINT), the debugger can indicate that the program can continue normally, and can continue as though a SIGINT had just arrived. In effect, this allows the program to continue as though it had not been interrupted by the ptrace service. The debugger could also choose to ignore the signal that stopped the process (again assume a SIGINT), by specifying PT_CONTINUE without a signal. This allows the program to resume running, but the original SIGINT is discarded before it is delivered to the debugged program.

When a debugger finishes debugging a program, it uses the PT_DETACH request to take the process out of debugging mode and allow it to continue. A signal can be supplied on this request, as it is with the PT_CONTINUE request.

When a process is continued using these ptrace service requests, all signals that are pending on the focus thread, and all signals that are pending on the process (other than the ones supplied on the PT_CONTINUE or PT_DETACH service request, or SIGKILL), are discarded.

When a debugged process is stopped because of a signal, or is waiting for a signal to arrive when the PT_CONTINUE or PT_DETACH request is issued, there are special considerations:

- If no signal is supplied on the ptrace service request, the process continues running immediately. If it was in a stopped state, it behaves as if a SIGCONT had just arrived. If it was waiting for a signal, it behaves as if a signal had just arrived.
- If a signal is supplied on the ptrace service request, that signal takes whatever action it normally would with respect to the state of the debugged process.

For example, if the process was in a stopped state, and a SIGCONT is supplied on the request, the process is taken out of the stopped state. However, if it was also waiting for the arrival of a signal, it still waits after the ptrace service request. Likewise, a signal whose action is to wake up processes that are waiting for a signal does so if the debugged process was waiting for a signal. If the process was in a stopped state, however, it remains in a stopped state after the ptrace service request has been processed. For more information about signal processing, see *z/OS UNIX System Services Messages and Codes*.

Ending a debugged process

To end a stopped, debugged process, you can use the ptrace PT_KILL service request. This causes the process to end as though it had received a SIGKILL signal. You can also use the PT_CONTINUE request to continue with a signal whose action is to end the process, although this has the effect of ending the process with the specified signal instead of with a SIGKILL.

Multiprocess debugging mode

Multiprocess debugging mode allows a debugger to control more than one process. The debugger uses the ptrace PT_MULTI service request to turn multiprocess mode on or off for a target process.

When multiprocess mode is in effect, the behavior of the exec, fork, attach_exec, attach_execmvs, and spawn services is modified. For the exec service, the only

change is that the Status_field parameter on the wait service issued by the debugger indicates that the process stopped for the exec service, instead of that it was stopped by the SIGTRAP signal.

For the fork service, the Status_field parameter on the debugger wait service indicates that the process stopped for the fork call. In addition, the fork service causes both the parent and the new child process to stop, and the debugger gets status for both processes with the wait service. The debugger should issue the wait service until it receives status for both the parent and child processes. This is different from multiprocess mode's not being in effect; in this case neither the parent nor child process stops because of the fork service, and the debugger is not made aware of the fork event at all.

For the attach_exec and attach_execmvs services, the Status_field parameter on the debugger wait service indicates that the process stopped for a local fork. In addition, these services cause both the parent and the new child process to stop, and the debugger gets status for both processes with the wait service. The debugger should issue the wait service until it receives status for both the parent and child processes. This is different from multiprocess mode's not being in effect; in this case, neither the parent nor child process stops because of these services, and the debugger is not made aware of the local fork event at all. After the notification of the local fork event, the attach_exec service loads the executable program into storage and causes the Status_field parameter on the wait service issued by the debugger to indicate that the process stopped for the exec service, instead of that it was stopped by the SIGTRAP signal.

For the spawn service, the effects are a combination of those described for fork and attach_exec. The parent presents status like that for a fork call (the debugger wait Status_field indicates that the process stopped for a fork). The child presents status like that for attach_exec (the debugger wait Status_field indicates that the process stopped for a local fork).

User area description

The PT_READ_U request is used with the user area for a target process. The user area is a collection of control information. It is not necessarily a contiguous storage area, and it is not readily accessible by an end user except via the PT_READ_U request.

To access the user area, an offset, as opposed to an absolute address, must be supplied on the PT_READ_U request. Each unit of control information is a fullword, and the offsets represent each multiple of 4 bytes. The offsets begin with 1 and progress by 1. The following shows the offsets and the associated control information that are defined in the BPXYPTRC macro (see “BPXYPTRC — Map parameters for ptrace” on page 1018):

Constant	Control information
PtUArea#MinSig–PtUArea#MaxSig	Signal catcher information for signal numbers 1 - 64 (the rest of the range is reserved). Not all potential signal numbers are valid; the valid signal numbers are defined in the BPXYSIGH macro. Signal catcher information is one of the following (the constants for signal default and ignore actions are defined in the BPXYSIGH macro): • SIG_DFL#: Take default action for this signal • SIG_IGN#: Ignore this signal • Address: Address of the signal catcher function
PtUArea#IntCode	Program interrupt code, in the following format: • Bytes 0 and 1: unused • Bytes 2 and 3: program interrupt code in hexadecimal
PtUArea#AbendCC	Abend completion code, in the following format: • Byte 0: flags • Bytes 1–3: system completion code (first 12 bits) and user completion code (second 12 bits)
PtUArea#AbendRC	Abend reason code
PtUArea#SigCode	Signal code, in the following format: • Bytes 0 and 1: unused • Bytes 2 and 3: signal code in hex
PtUArea#ILC	Instruction length code, in the following format: • Bytes 0–2: unused • Byte 3: instruction length code

The PT_READ_U request can therefore be used to obtain additional information about signals; or when a debugger is notified that a debugged process stopped with a SIGILL, SIGSEGV, SIGFPE, or SIGABND signal.

Related services

- “attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program” on page 50
- “attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program” on page 59
- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185
- “loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name” on page 333
- “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 780
- “wait (BPX1WAT, BPX4WAT) — Wait for a child process to end” on page 882

Characteristics and restrictions

The following restrictions apply to the use of the ptrace service:

1. The ptrace service is supported from programs that are running in PSW key 8 mode only. Calls to the ptrace service that are made from debugger programs (authorized or problem state) with other than key 8 are rejected with an error code.

2. A process that is being debugged must not be running if any of the following environmental conditions are true:
   - It is running in access register (AR) mode.
   - It is running in supervisor PSW state.
   - It is running with a PSW key not equal to 8.
   - It is running with APF authorization, and the debugger process does not have read permission to the BPX.DEBUG resource in the FACILITY class.
   - It is running with the security product function called Program Access to Data Support (PADS) activated.

   A process that is running with any of these conditions ends abnormally if it attempts to use the ptrace service to notify the debugger of a ptrace service event.
3. A SIGKILL signal that is sent to a process that is being debugged by the ptrace service cannot be trapped. When a SIGKILL signal ends a process, the ptrace service is not given a chance to intervene.

   Note: SIGKILL is delivered to the target process according to normal signal delivery rules. If the target process is stopped, but is not waiting for signals (for example, if it is stopped for a ptrace service event), the SIGKILL remains pending until the process resumes (using the same example, when a PT_CONTINUE, PT_DETACH, or PT_KILL ends the ptrace service event).
4. The PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests cannot be issued with a target process that is a system address space (see "MVS-related information" on page 564). If this attempt is made, the EPERM error is returned in the Return_code parameter.
5. The PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests cannot be issued with a target process that is the INIT process (with a process identifier equal to 1). If this attempt is made, the EPERM error is returned in the Return_code parameter.
6. The PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests cannot be issued with a target process that is the parent of the calling process. If this attempt is made, the ESRCH error is returned in the Return_code parameter.
7. The PT_ATTACH, PT_REATTACH, and PT_REATTACH2 requests cannot be issued with a target process that uses the setuid service to set the uid to 0, unless the process is also running with superuser equal to daemon authority (in other words, is running without an active BPX.DAEMON resource profile in the FACILITY class). If this attempt is made, the EPERM error is returned in the Return_code parameter.
8. The debugger cannot use multiple threads within a single process to debug multiple target processes. If multiprocess debugging is desired, either a single thread debugger process must be associated with all debugged processes, or the debugger must use multiple processes, where the association with debugged processes is on a one-to-one basis.
9. The debugger should not have a signal catcher for the SIGCHLD signal. The ptrace service uses the SIGCHLD signal for internal communication, and the use of a catcher by the debugger would interfere with this communication. The most visible result of using a SIGCHLD catcher would be EINTR errors returned for most ptrace service requests, although other unpredictable results could also occur.

10. To ensure that pending signal information for the focus thread is not lost (because the signals were discarded), the PT_THREAD_INFO request should be the first ptrace request that is issued when the debugger gets notified of a ptrace service event.
11. The following requests are not supported while a debugged process is stopped for a local fork child event:
    - PT_READ_I
    - PT_READ_D
    - PT_READ_BLOCK
    - PT_READ_GPR
    - PT_READ_GPRH
    - PT_READ_FPR
    - PT_WRITE_I
    - PT_WRITE_D
    - PT_WRITE_BLOCK
    - PT_WRITE_GPR
    - PT_WRITE_GPRH
    - PT_WRITE_FPR
    - PT_REGHSET
    - PT_REGSET
    - PT_CONTINUE to continue at a specified address
    - PT_READ_VR
    - PT_WRITE_VR
12. The following requests are not supported while a debugged process is stopped for an extended event:
    - PT_READ_GPR
    - PT_READ_GPRH
    - PT_READ_FPR
    - PT_WRITE_GPR
    - PT_WRITE_GPRH
    - PT_WRITE_FPR
    - PT_REGHSET
    - PT_REGSET
    - PT_CONTINUE to continue at a specified address
    - PT_READ_VR
    - PT_WRITE_VR
13. If the debugger is running in a multi-thread process, then the SIGCHLD signal must be blocked on all threads except the one issuing the BPX1PTR call.

Examples

For an example using this callable service, see “BPX1PTR (ptrace) example” on page 1174.

MVS-related information

1. As a result of the PT_LDINFO request, the ptrace service invokes the Contents Supervisor CSVINFO service. CSVINFO returns information about load modules in the debugged process based on CSV control blocks. This information is then returned to the caller of the PT_LDINFO request. CSVINFO uses the MVS macros ATTACH, LINK or XCTL; or the exec or loadhfs service to return information about all modules brought into storage by any task in the process.
2. PT_READ_GPR requests that read the machine control registers (CRs) can return CR information that is not consistent with the user program that is being

debugged. This is because the ptrace service reads the actual hardware registers that probably have changed because of internal PC invocations.

3. No distinction is made between the instruction area (_I) or data area (_D) for the PT_READ_I, PT_READ_D, PT_WRITE_I, and PT_WRITE_D requests. These are all treated as user storage requests.
4. A debugger cannot set breakpoints in programs that are loaded into read-only storage (for example subpool 252 or LPA). Users of debugger programs that use ptrace must be aware of the storage location of their programs, and, if necessary, take appropriate steps to ensure that the programs are loaded into read/write storage (for example, subpool 251).
5. The PT_RECOVER request can be issued by ESPIE and ESTAE routines that capture program checks in user programs. The main requirement is that any registers or PSW values that are changed by the debugger after it recognizes the program check event be restored before the user program resumes running. Also, if a signal is sent to the debugged program by the user recovery routine, it must be sent outside of the user recovery routine (ESPIE or ESTAE). This ensures that signal delivery operates in the correct environment.
6. SVC 144 instructions can be inserted only into storage key 8 user programs. You cannot use SVC 144 instructions to do breakpoint debugging of system (key 0) routines.

   The SVC 144 routine has the following characteristics:

   - SVC 144 is a type-3 SVC.
   - The user program registers and PSW that are saved by the SVC 144 routine are changed if requested by PT_WRITE_GPR requests.
   - Any modification that is made to register 14 with a PT_WRITE_GPR request is lost. This is because the SVC 144 routine uses register 14 to exit.
   - If the process under which the SVC 144 routine runs is not in ptrace mode (started with a PT_TRACE_ME, PT_ATTACH, PT_REATTACH or PT_REATTACH2 request), the routine abends the caller.
   - If the SVC 144 routine is called while the process is in access register mode, supervisor state, or any key other than 8, the routine abends the caller. In addition, APF-authorized invocation is not allowed unless the debugger has read permission to the BPX.DEBUG resource in the FACILITY class.
7. MVS system address spaces cannot be debugged with the ptrace service. A system address space is identified by one of the following:
   - A command scheduling control block (CSCB) does not exist. The master address space is an example of an address space with no CSCB.
   - The CSCB identifies the address space as a system address space.
8. The sample CEEEVDBG module must be installed as follows:
   - The sample CEEEVDBG module is in the form of source code that is written in basic assembler language. This module must be assembled with the following Language Environment macros made available to the assembler: CEECAA, CEEDSA, CEEENTRY, CEEPPA.
   - The object deck must be link-edited with the object deck for the ptrace system call stub.
   - The load module must be placed into a load library that is accessible by Language Environment.

querydub (BPX1QDB, BPX4QDB) — Obtain the dub status of the current task

Function

The querydub callable service obtains the *dub* status information for the current task. The status information indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QDB):	31-bit
AMODE (BPX4QDB):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QDB,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QDB with the same parameters.

Parameters

Return_value

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the querydub service returns -1 if the request is unsuccessful. Otherwise it contains the returned status, which can have one of the following values:

Value	Description
QDB_DUBBED_FIRST	The task has already been dubbed. This task and this RB caused the dub.
QDB_DUBBED	The task has already been dubbed. Another task or another RB caused the dub.
QDB_DUB_MAY_FAIL	The task has not been dubbed; an attempt to dub the task may fail. The most likely reason for failure may be a missing or incomplete user security profile; or the lack of an OMVS segment.

Value	Description
QDB_DUB_OKAY	The task has not been dubbed; an attempt to dub the task will probably succeed. The service has determined that an OMVS segment exists for the task. However, it has not checked for other potential errors. It is possible that the task may not have the proper UID and GID set up in the security profile, causing a subsequent dub failure.
QDB_DUB_AS_PROCESS	The task has not been dubbed, but its address space has. An attempt to dub the task will cause the task to be dubbed as another process within the address space.
QDB_DUB_AS_THREAD	The task has not been dubbed, but its address space has. An attempt to dub the task will cause the task to be dubbed as a thread within the process (address space).

These constant values are defined in "BPXYCONS — Constants used by services" on page 952.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the querydub service stores the return code. The querydub service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the querydub service stores the reason code. The querydub service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

None.

Characteristics and restrictions

There are no restrictions on the use of the querydub service.

Examples

For an example using this callable service, see "BPX1QDB (querydub) example" on page 1176.

queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered

Function

The queue_interrupt callable service returns to the kernel the last interrupt that was delivered to the signal interface routine (SIR). The interrupt can be a signal, a cancelation request, or a quiesce request.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPXB1SPB):	31-bit
AMODE (BPX4SPB):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPB,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPB with the same parameters.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which the queue_interrupt service returns `0` if it has permission to return the specified interrupt for delivery at the next kernel call. If no interrupt is returned, -1 is returned.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the queue_interrupt service stores the return code. The queue_interrupt service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The queue_interrupt service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of Signal in the PPSD at the time this service was invoked was an unsupported signal. Either there was a storage overlay in the PPSD, or no signal was ever delivered to this task.
EPERM	The caller does not have permission to return the interrupt now. All signals must be blocked, and the task must invoke mvssigsetup before the queue_interrupt service is invoked. The following reason codes can accompany the return code: JRSignalsNotBlocked and JRNotSigsetup.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the queue_interrupt service stores the reason code. The queue_interrupt service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The data that is mapped by the BPXYPPSD control block is used by the queue_interrupt service, and therefore should not be modified by the invoker, because this may result in an EINVAL return code.
2. The queue_interrupt returns the interrupt to the kernel and restores the signal blocking mask to its preinterrupt state. The signal is acted on at the end of the next service.
3. When the PPSDJUMPBACK flag is set on in the BPXYPPSD (see "BPXYPPSD — Map signal delivery data" on page 1014) and the queue_interrupt call is valid, control is not returned to the instruction after the queue_interrupt invocation. Instead, it is returned to the point of the signal interruption that was just queued back to the kernel. General and access registers are restored to the values saved in the PPSD at the time of the interrupt.
4. When the PPSDREDRIVE flag is set on in the BPXYPPSD, the kernel is responsible for rescheduling the queued signal to interrupt the current thread at a later time. The signal interface routine (SIR) is no longer responsible for issuing another syscall to cause delivery of the signal. In fact, nonblocking syscalls (syscalls that do not return EINTR) do not cause delivery of pending deliverable signals when a redrive signal is in progress. Delivery of the signal only occurs when the redrive time limit expires. This time limit is maintained by the kernel and cannot be specified by the user.
5. When the PPSDMASKONLY flag is set on, the kernel does not requeue the signal; it only resets the signal mask to the value in PPSDCURRENTMASK.

Related services

- "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421
- "pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process" on page 515
- "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495

Characteristics and restrictions

The intended use of the queue_interrupt service is from the signal interface routine that is specified on "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421. Although the queue_interrupt service can be used anywhere, all signals must be blocked, and the task must set up signals by invoking the mvssigsetup service before calling queue_interrupt. See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1SPB (queue_interrupt) example" on page 1193.

quiesce (BPX1QSE, BPX4QSE) — Quiesce a file system

Function

The quiesce callable service quiesces a file system, making the files in it unavailable for use. After the file system is quiesced, the system can back up the data in it.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1QSE):	31-bit
AMODE (BPX4QSE):	64-bit
ASC mode:	Primary mode
Serialization:	Enabled for interrupts
Locks:	No locks held
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1QSE,(File_system_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4QSE with the same parameters.

Parameters

File_system_name
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
44 bytes

The name of a 44-character field that contains the file system name. The name must be left-justified and padded with blanks.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the quiesce service returns one of two values:

- `0`, if the request is successful and the file system was mounted at the time of the request.
- `-1`, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the quiesce service stores the return code. The quiesce service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The quiesce service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The resource was temporarily unavailable. The following reason code can accompany the return code: JRLockErr.
EBUSY	The file system that was specified is being unmounted or has already been quiesced; or there are no more locks available. The following reason codes can accompany the return code: JROutOfLocks, JRQuiesced, and JRUnmountInProgress.
EINVAL	The file system that was specified cannot be quiesced. The following reason code can accompany the return code: JRInvalidParms.
ENODEV	The file system that was specified is not mounted. The following reason code can accompany the return code: JrFileSysNotThere.
EPERM	The user cannot request this service because it lacks the required permission. The following reason code can accompany the return code: JRUserNotPrivileged.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the quiesce service stores the reason code. The quiesce service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. After a quiesce service request, the file system is unavailable for use until a subsequent unquiesce service request is received.
2. Users accessing files in a quiesced file system are suspended until an unquiesce request for the file system is processed.

Related services

- "unquiesce (BPX1UQS, BPX4UQS) — Unquiesce a file system" on page 877

Characteristics and restrictions

To quiesce a file system, the requester must be a superuser or, at least, have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class. This is the same authority that is required for mounting or unmounting a file system.

When a system joins the sysplex and processes mounts during initialization, any file system mounted in the sysplex that is in a quiesced state will not be mounted on that system at that time. When the quiesced file system is unquiesced, that file system will be mounted on any systems in the sysplex that do not have it already mounted.

Examples

For an example using this callable service, see "BPX1QSE (quiesce) example" on page 1177.

read (BPX1RED, BPX4RED) — Read from a file or socket

Function

The read callable service reads the number of bytes that you specify from a file or socket into a buffer that you provide.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
	SRB - AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1RED):	31-bit task or srb mode
AMODE (BPX4RED):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RED,(File_descriptor,
              Buffer_address,
              Buffer_ALET,
              Read_count,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RED with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of an open file or socket.

Buffer_address
Parameter supplied and returned

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer into which data is to be read.

Buffer_ALET
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ALET for the Buffer_address that identifies the address space or data space where the buffer resides.

You should specify a Buffer_ALET of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Read_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of bytes that you want to read from the file. This number must be less than or equal to the length of the buffer that you provide for data to be read into.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read service returns the number of bytes that were actually read (this may be 0) if the request is successful, or -1 if it is not successful.

For more information about the return value, refer to usage note 5 on page 575.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read service stores the return code. The read service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The read service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The file was opened with the nonblock option, and data is not available to be read.
EBADF	The File_descriptor parameter does not contain the descriptor of an open file; or the file is not opened for read. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINTR	The service was interrupted by a signal before it could read any data.
EINVAL	The Read_Count parameter contains a value that is less than zero; or the socket is marked shutdown for read. The following reason codes can accompany the return code: JRSocketClosed, JRSocketCallParmError.
EIO	The process is in a background process group, and is attempting to read from its controlling terminal. Either the process is ignoring or blocking the SIGTTIN signal, or the process group is orphaned.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	The Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EWOULDBLOCK	• The socket is marked nonblocking, and no data is waiting to be received. The following reason code can accompany the return code: JRWouldBlock. • The socket is marked blocking, and the call has blocked, without receiving any data, for the time period specified in the SO_RCVTIMEO option . The following reason code can accompany the return code: JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read service stores the reason code. The read service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. **Read_count**: The value of Read_count is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.
3. **Access time**: A successful read updates the access time of the file read.
4. **Origin of bytes read**: If the file that is specified by File_descriptor is a regular file, or any other type of file where a seek operation is possible, bytes are read from the file offset that is associated with the file descriptor. A successful read increments the file offset by the number of bytes that are read. For files for which no seek operation is possible, there is no file offset associated with the file descriptor. Reading begins at the current position in the file.
5. **Number of bytes read**: When a read request completes, the Return_value field shows the number of bytes that were actually read—a number less than or equal to the number that was specified as Read_count. The following are some reasons why the number of bytes read might be less than the number of bytes requested:
   - Fewer than the requested number of bytes remained in the file; the end of file was reached before Read_count bytes were read.
   - The service was interrupted by a signal after some, but not all, of the requested bytes were read. (If no bytes were read, the return value is set to —1 and an error is reported.)
   - The file is a pipe, FIFO, or special file, and fewer bytes than Read_count specified were available for reading.

     There are several reasons why a read request might complete successfully with no bytes read. That is, with Return_value set to `0`. For example, zero bytes are read in these cases:
     - The service specified a Read_count of zero.
     - The starting position for the read was at or beyond the end of the file.
     - The file that is being read is a FIFO file or a pipe, and no process has the pipe open for writing.
     - The file that is being read is a slave pseudoterminal, and a zero-length canonical line was written to the master.
     - A directory is being read and the Physical File System does not support simple reads from directories. Opendir() and readdir() should be used.
6. **Nonblocking**: If a process has a pipe open for reading with nonblocking specified, a request to read from the file ends with a return value of —1 and a "Resource temporarily unavailable" return code. But if nonblocking is not specified, the read request is blocked (does not return) until some data is written, or until the pipe is closed by all other processes that have the pipe open for writing.

   Master and slave pseudoterminals also operate this way, except that how they act depends on how they were opened. If the master or the slave is opened

blocking, the reads are blocked if there is no data. If they are opened nonblocking, EAGAIN is returned if there is no data.

7. **SIGTTIN processing**: The read service causes signal SIGTTIN to be sent under the following conditions:
   - The process is attempting to read from its controlling terminal, and
   - The process is running in a background process group, and
   - The SIGTTIN signal is not blocked or ignored, and
   - The process group of the process is not orphaned.

   If these conditions are met, SIGTTIN is sent. If SIGTTIN has a handler, the handler gets control, and the read ends with the return code set to EINTR. If SIGTTIN is set to default, the process stops in the read and continues when the process is moved to the foreground.

Related services

- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "lseek (BPX1LSK, BPX4LSK) — Change a file's offset" on page 345
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481

Note: The read service is not related to the **read** shell command.

Characteristics and restrictions

If the file was opened by an authorized program, all subsequent reads and writes against the file must be issued from an authorized state.

The read (BPX1RED, BPX4RED) and write (BPX1WRT, BPX4WRT) callable services do not support simultaneous reading or writing of the same shared open file by different threads when one or both of the following are true:

1. Automatic conversion is enabled using Enhanced ASCII (ON) and different character set IDs (CCSIDs) are used.
2. Automatic conversion is enabled using Unicode Services (ALL) and different CCSIDs are used, or mixing read and write operations of multibyte characters are performed which result is storing of partial characters.

The first restriction is not applicable if each thread coordinates its reads and writes so that simultaneous I/O does not occur. Both restrictions are not applicable if each thread opens the file independently.

Reads or writes that cause a conversion of greater than 2 G result in an EINVAL error with reason JrUniOpTooBig.

Refer to the BPX1FCT (fcntl) operations for instances of how a lseek operation in a conversion environment can affect read and write operations

Examples

For an example using this callable service, see "BPX1RED (read) example" on page 1181.

readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory

Function

The readdir callable service reads multiple name entries from a directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RDD):	31-bit
AMODE (BPX4RDD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RDD,(Directory_file_descriptor,
              Buffer_address,
              Buffer_ALET,
              Buffer_length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RDD with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

Directory_file_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened (see "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452).

Buffer_address
Parameter supplied and returned

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer in which readdir is to write the directory entries. This address must be supplied to the readdir call. The directory entries are mapped by the BPXYDIRE macro; see "BPXYDIRE — Map directory entries for readdir" on page 965.

Buffer_ALET
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ALET for the Buffer_address that identifies the address space or data space where the buffer resides.

You should specify a Buffer_ALET of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer_address.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir service returns the number of directory entries that have been read into the buffer, or -1 if it is unsuccessful. A value of `0` in Return_value indicates the end of the directory.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir service stores the return code. The readdir service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The readdir service can return one of the following values in the Return_code parameter:

Return code	Explanation
EBADF	The Directory_file_descriptor argument does not refer to an open directory.
EINVAL	The buffer is too small to contain any entries.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir service stores the reason code. The readdir service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This interface differs from the POSIX C high-level-language interface in that it returns more than one directory entry; it also returns the entries in the caller's buffer.
2. The buffer contains a variable number of variable-length directory entries. Only full entries are placed in the buffer, up to the buffer size specified, and the number of entries is returned.
3. Each directory entry that is returned has the following format (as shown on "BPXYDIRE — Map directory entries for readdir" on page 965):
   - 2-byte Entry_length. The total entry length, including itself.
   - 2-byte Name_length. The length of the following Member_name subfield.
   - Member_name. A character field of length Name_length. This name is not terminated by a null character.
   - File system specific data. If Name_length + 4 = Entry_length, this subfield is not present.

   The entries are packed together, and the length fields are not aligned on any particular boundary.
4. The buffer that is returned by one call to the readdir service must be used again on the next call to the readdir service, to continue reading entries from where you left off. The buffer must not be altered between calls, unless the directory has been rewound.
5. If the contents of the directory have changed (files have been added or removed) since a previous call to the readdir service, a call should be made to the rewinddir service so that the updated contents of the directory can be read.
6. The end of the directory is indicated in one of two ways:
   - A Return_value of `0` entries is returned.
   - Some physical file systems may return a null name entry as the last entry in the caller's buffer. A null name entry has an Entry_length of `4` and a Name_length of `0`.

   The caller of the readdir service should check for both conditions.
7. HFS returns names in sorted order. Other z/OS UNIX file systems, such as zFS and TFS, follow the UNIX standard and do not return names in sorted order.

Related services

- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613

Characteristics and restrictions

There are no restrictions on the use of the readdir service.

Examples

For an example using this callable service, see "BPX1RDD (readdir) example" on page 1179.

readdir2 (BPX1RD2, BPX4RD2) — Read an entry from a directory

Function

The readdir2 callable service reads multiple name entries from a directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RD2):	31-bit
AMODE (BPX4RD2):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RD2,(Directory_file_descriptor,
              UIO,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RD2 with the same parameters. Some of the addresses in the UIO structure are doublewords.

Parameters

Directory_file_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened (see "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452).

UIO
Supplied and returned parameter

Type: Structure

Length:
Fuio#Len (from the BPXYFUIO macro)

The name of an area that contains the user input and output block. This area is mapped by the BPXYFUIO macro (see "BPXYFUIO — Map file system user I/O block" on page 967).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir2 service returns the number of directory entries that have been read into the buffer that is pointed to by the UIO, or -1 if the request is unsuccessful. A value of 0 in Return_value indicates the end of the directory.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir2 service stores the return code. The readdir2 service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The readdir2 service can return one of the following values in the Return_code parameter:

Return code	Explanation
EACCES	The FuioChkAcc bit was set to request that an access check be performed, but the calling process does not have permission to read the specified directory.
EBADF	The Directory_file_descriptor argument does not refer to an open directory.
EINVAL	There was a parameter error; for example, a supplied area was too small. The following reason codes can accompany the return code: JRInvalidFuio,JrBytes2RWZero, JRRddPlusNoCursorSupp

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readdir2 service stores the reason code. The readdir2 service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This interface differs from the POSIX C high-level-language interface in that it returns more than one directory entry, and it also returns the entries in the caller's buffer.
2. The buffer contains a variable number of variable-length directory entries. Only full entries are placed in the buffer, up to the buffer size specified, and the number of entries is returned.

3. Each directory entry that is returned has the following format, which is mapped by BPXYDIRE (see "BPXYDIRE — Map directory entries for readdir" on page 965):
   - 2-byte Entry_length. The total entry length, including itself.
   - 2-byte Name_length. The length of the following Member_name subfield.
   - Member_name. A character field of length Name_length. This name is not terminated by a null charcter.
   - File system specific data. If Name_length + 4 = Entry_length, this subfield is not present.

   The entries are packed together, and the length fields are not aligned on any particular boundary.
4. The end of the directory is indicated when a Return_value of `0` entries is returned.

   In addition, some physical file systems may return a null name entry as the last entry in the caller's buffer. A null name entry has an Entry_length of `4` and a Name_length of `0`. The caller of the readdir2 service should check for both conditions.
5. Two protocols are supported for reading through large directories with successive calls:
   - **Cursor protocol.** The cursor, or offset, that is returned in the UIO by the readdir2 service contains file-system-specific information that locates the next directory entry. The cursor and buffer must be preserved by the caller from one readdir2 call to the next, and reading proceeds based on the cursor. The buffer must not be altered between calls, unless the directory has been rewound.
   - **Index protocol.** The index that is set in the UIO by the caller determines which entry to start reading from. To read through the directory, the caller increments the index by the number of entries that were returned on the previous call.

     Because this index represents the number of entries into the directory, the caller should be aware that if entries are being added or deleted in the directory while the call is being done, duplicate or missing entries could result.

     The cursor protocol is preferred for better performance.
6. The cursor information that is returned from a call to readdir2() can be used on successive calls to readdir().
7. If the contents of the directory have changed (files have been added or removed) since a previous call to the readdir2 service, a call should be made to the rewinddir service so that the updated contents of the directory can be read.
8. The following UIO fields should be set to specify the details of the read directory request:

   FuioID
   Contains Fuio#ID (from the BPXYFUIO macro).

   FuioLen
   Contains the length of the UIO structure.

   FuioChkAcc
   Requests that the PFS perform required access checking before performing the requested readdir2 operation.

FuioBufferAddr
Contains the address of a buffer where the directory entries are to be returned.

FuioBufferAlet
Contains the ALET of the buffer where the directory entries are to be returned.

FuioIBytesRW
Specifies the maximum number of bytes that can be written to the output buffer.

FuioRDIndex
Specifies the first directory entry that is to be returned when the index protocol is used. The directory can be thought of as a 1–based array, and the index specifies which entry in the directory to begin reading from. When the FuioRDIndex is set to any nonzero value it will override any value in the FuioCursor field. To begin reading at the first directory entry, set the FuioRDIndex to 1.

FuioCursor
When the cursor protocol is used, this specifies a value, returned on the previous readdir2 call, that indicates the next entry to be read; or 0 on the first call. The FuioRDIndex must be set to 0 when the cursor protocol is being used. To begin reading at the first directory entry, both the FuioRDIndex and the FuioCursor should be set to 0.

FuioRddPlus
Indicates that the request is for the ReaddirPlus function. The attributes for each entry should be included in the output. If FuioRddPlus is specified then the Index protocol is used.

9. Some addresses in the UIO structure are doublewords, and some are not. If the buffer address is a 64-bit address, the caller must set the FUIOADDR64 flag in BPXYFUIO, and the FUIOBUFF64VADDR must contain the 64-bit virtual buffer address. When FUIOADDR64 is not set, the FUIOBUFFERADDR must contain the 31-bit virtual buffer address.
10. The following UIO fields are returned by the readdir2 service:

FuioPSWKey
This field is set to the caller's key.

FuioCursor
This field is set to the current cursor position after the readdir2 has occurred.

FuioAsid
This field is set to the caller's ASID.

FuioCVerRet
This field indicates that the Cookie Verifier (FuioCVer) is being returned.

FuioCVer
When FuioCVerRet is on, this field is set to the Cookie Verifier for the directory that is being read. When a directory is being read with multiple reads, you can use the FuioCVer that is returned to compare each Cookie Verifier with the previous one. If the directory has been modified between reads, you can reject the request, because the results will not be valid.

11. The buffer contents that are returned by the readdir2 service are mapped by the BPXYDIRE macro (see "BPXYDIRE — Map directory entries for readdir" on page 965).

Related services

- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577
- "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613

Characteristics and restrictions

There are no restrictions on the use of the readdir2 service.

Examples

For an example using this callable service, see "BPX1RD2 (readdir2) example" on page 1180.

read_extlink (BPX1RDX, BPX4RDX) — Read an external symbolic link

Function

The read_extlink callable service reads the contents of an external symbolic link into a buffer that you provide. The external symbolic link contains the external name that was specified when the symbolic link was defined (see "extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link" on page 153).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RDX):	31-bit
AMODE (BPX4RDX):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RDX,(Link_name_length,
              Link_name,
              Buffer_length,
              Buffer_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RDX with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

Link_name_length,
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Link_name.

Link_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Link_name_length parameter

The name of a field that contains the link name of the external symbolic link that is to be read. The length of this field is specified in Link_name_length.

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer_address.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer that is supplied to the read_extlink service, into which the value of the external symbolic link is to be written. The value of the external symbolic link is actually the external name that was specified when the symbolic link was created. The buffer must reside in the process's address space.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read_extlink service returns a count of the number of characters placed in the buffer, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read_extlink service stores the return code. The read_extlink service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The read_extlink service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Search permission is denied for a component of the path prefix.
EINVAL	The file that is named by Link_name is not a symbolic link; or there was a problem with the supplied buffer. The following reason codes can accompany the return code: JRFileNotSymLink, and JRRdlBuffLenInvalid.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
ENAMETOOLONG	Link_name is longer than 1023 characters; or some component of the link name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file with the name specified by Link_name was found. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the path prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the read_extlink service stores the reason code. The read_extlink service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes.*

Usage notes

1. If the buffer that is supplied to the read_extlink service is too small to contain the value of the external symbolic link, the value is truncated to the length of the buffer (Buffer_length). If the value that is returned is the length of the buffer, you can use the lstat service (see "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349) to determine the actual length of the external symbolic link.
2. If the Buffer_length is `0`, the value that is returned is the number of bytes in the external symbolic link. The buffer remains unchanged.
3. It is recommended that this function, rather than the readlink function (see "readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link" on page 587), be used for reading an external link with a symbolic link ending its pathname.

Related services

- "extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link" on page 153

- "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349
- "readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link"
- "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the read_extlink service.

Examples

For an example using this callable service, see "BPX1RDX (read extlink) example" on page 1180.

readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link

Function

The readlink callable service reads the contents of a symbolic link into a buffer that you provide. The symbolic link contains the pathname that was specified when the symbolic link was defined (see "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812).

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RDL):	31-bit
AMODE (BPX4RDL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RDL,(Link_name_length,
              Link_name,
              Buffer_length,
              Buffer_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RDL with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

Link_name_length,
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Link_name.

Link_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Link_name_length parameter

The name of a field that contains the link name of the symbolic link that is to be read. The length of this field is specified in Link_name_length.

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length, in bytes, of the buffer that is pointed to by Buffer_address.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the buffer that is supplied to readlink, into which the value of the symbolic link is to be written. The value of the symbolic link is actually the pathname that was specified when the symbolic link was created. The buffer must reside in the process's address space.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readlink service returns a count of the number of characters placed in the buffer, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readlink service stores the return code. The readlink service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The readlink service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Search permission is denied for a component of the path prefix.
EINVAL	The file named by Link_name is not a symbolic link; or there was a problem with the supplied buffer. The following reason codes can accompany the return code: JRFileNotSymLink, JRRdlBuffLenInvalid.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
ENAMETOOLONG	Link_name is longer than 1023 characters; or some component of the link name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file with the name specified by Link_name was found. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the path prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readlink service stores the reason code. The readlink service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the buffer that is supplied to the readlink service is too small to contain the value of the symbolic link, the value is truncated to the length of the buffer (Buffer_length). If the value that is returned is the length of the buffer, you can use the lstat service (see "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349) to determine the actual length of the symbolic link.
2. If the Buffer_length is `0`, the value that is returned is the number of bytes in the symbolic link. The buffer remains unchanged.

Related services

- "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349
- "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the readlink service.

Examples

For an example using this callable service, see "BPX1RDL (readlink) example" on page 1179.

readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers

Function

The readv callable service reads data and stores it in a set of buffers.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task SRB - AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1RDV):	31-bit task or SRB mode
AMODE (BPX4RDV):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RDV,(File_descriptor,
              Iov_count,
              Iov_struct,
              Iov_alet,
              Iov_buffer_alet,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RDV with the same parameters. All addresses in parameter structures are doublewords.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for which the readv request is to be done.

Iov_count
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the number of buffers that are pointed to by Iov_struct. The total number of buffers may not exceed IOV_MAX (defined in "BPXYIOV — Map the I/O vector structure" on page 986).

Iov_struct
Supplied parameter

Type: Structure

Length:
Iov_count times length(iov)

The name of a field that contains 31(64)-bit pointers to buffers in which data is to be stored, and their lengths. In 64-bit mode, Iov_struct contains doubleword pointer and length subfields. See "BPXYIOV — Map the I/O vector structure" on page 986 for more information.

Iov_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Iov_struct.

Iov_buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for buffers that are pointed to by Iov_struct.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readv service returns one of the following:

- The number of bytes that were read into the buffers, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readv service stores the return code. The readv service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The readv service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The file was opened with the nonblock option, and data is not available to be read.
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINTR	A signal interrupted the readv function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	The socket is marked shutdown for read; or an incorrect length was specified in the iov. The following reason codes can accompany the return code: JRSocketClosed, JRSocketCallParmError.
EIO	The process is in a background process group, and is attempting to read from its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking **SIGTTOU** signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
EWOULDBLOCK	• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available. • The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. The following reason code can accompany the return code: JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the readv service stores the reason code. The readv service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

- See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
- **Socket files**: When this callable service is used for datagram sockets, the readv service returns the entire datagram that was sent, providing that the datagram

fits into the specified buffers. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of readv may be needed to return all the data.

- **Bytes read**: The number of bytes that were requested for reading is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.
- **Access time**: A successful read updates the access time of the file read.
- **Origin of bytes read**: If the file that was specified by File_descriptor is a regular file, or any other type of file where a seek operation is possible, bytes are read from the file offset that is associated with the file descriptor. A successful read increments the file offset by the number of bytes that are read.

  For files where no seek operation is possible, there is no file offset associated with the file descriptor. Reading begins at the current position in the file.
- **Number of bytes read**: When a read request completes, the Return_value field shows the number of bytes that were actually read — a number less than or equal to the number of bytes that were requested. Following are some reasons why the number of bytes that are read might be less than the number of bytes that were requested:
  - Fewer than the requested number of bytes remained in the file; the end of file was reached before all requested bytes were read.
  - The service was interrupted by a signal after some, but not all, of the requested bytes were read. (If no bytes were read, the return value is set to -1 and an error is reported.)
  - The file is a pipe, FIFO, or special file, and fewer bytes than requested were available for reading.

  There are several reasons why a read request might complete successfully with no bytes read — that is, with Return_value set to 0. For example, zero bytes are read in these cases:
  - The service specified that zero bytes were to be read.
  - The starting position for the read was at or beyond the end of the file.
  - The file that is being read is a FIFO file or a pipe, and no process has the pipe open for writing.
  - The file that is being read is a slave pseudoterminal, and a zero-length canonical line was written to the master.
- **Nonblocking**: If a process has a pipe open for reading with nonblocking specified, a request to read from the file ends with a return value of —1 and a "Resource temporarily unavailable" return code. But if nonblocking is not specified, the read request is blocked (does not return) until some data is written, or until the pipe is closed by all other processes that have the pipe open for writing.

  Master and slave pseudoterminals also operate this way, except that how they act depends on how they were opened. If the master or the slave is opened blocking, the reads are blocked if there is no data. If they are opened nonblocking, EAGAIN is returned if there is no data.
- **SIGTTIN processing**: The readv service causes signal **SIGTTIN** to be sent under the following conditions:
  - The process is attempting to read from its controlling terminal, and
  - The process is running in a background process group, and
  - The **SIGTTIN** signal is not blocked or ignored, and
  - The process group of the process is not orphaned.

If these conditions are met, **SIGTTIN** is sent. If **SIGTTIN** has a handler, the handler gets control and the read ends with a return code of EINTR. If **SIGTTIN** is set to default, the process stops in the read and continues when the process is moved to the foreground.

Related services

- "writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers" on page 933

Characteristics and restrictions

There are no restrictions on the use of the readv service.

Examples

For an example using this callable service, see "BPX1RDV (readv) example" on page 1179.

realpath (BPX1RPH, BPX4RPH) — Resolve a pathname

Function

The realpath service derives, from the pathname that is pointed to by Pathname, an absolute pathname that names the same file, whose resolution does not involve dot (.), dot-dot (..), or symbolic links.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RPH):	31-bit
AMODE (BPX4RPH):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RPH,(Pathname_length,
              Pathname,
              Resolved_name_length,
              Resolved_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RPH with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname that is to be resolved.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname that is to be resolved. The length of this field is specified in Pathname_length.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Resolved_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the buffer to which the realpath service returns an absolute pathname without dot (.), dot-dot (..), or symbolic links. Resolved_name_length must be large enough to accommodate the actual length of an absolute pathname, plus one (for the terminating null). A length of zero has special meaning; see the usage notes.

Resolved_name
Parameter supplied and returned

Type: Character string

Character set:
No restriction

Length:
Specified by the Resolved_name_length parameter

The name of the buffer that is to hold the absolute pathname that is to be generated for the input Pathname. The length of this field is specified in Resolved_name_length.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the realpath service returns the length of the pathname that is in the buffer, if the request is successful; or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the realpath service stores the return code. The realpath service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The realpath service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search one of the components of Pathname.
EINVAL	There was a parameter error; for example, Resolved_name_length is not valid. The following reason codes can accompany the return code: JRBuffLenInvalid and JRBadAddress.
EIO	An input/output error occurred.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or a component of Pathname is longer than 255 characters. Name truncation is not supported.
ENOENT	No file named Pathname was found; or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of Pathname is not a directory.
ENOMEM	Insufficient storage space is available. The following reason code can accompany the return code: JRNoStorage.
ERANGE	The specified Resolved_name_length is less than the length of the pathname that was generated for the input Pathname. The specified Resolved_name_length is zero, and the length of the pathname that was generated for the input Pathname is larger than PATH_MAX bytes. The following reason code can accompany the return code: JrBuffTooSmall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the realpath service stores the reason code. The realpath service returns a Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. PATH_MAX plus 1 for the terminating null is a reasonable value for Resolved_name_length and for the size of Resolved_name.
2. If a Resolved_name_length value of zero is passed to this service, the generated pathname is stored, up to a maximum of PATH_MAX bytes, in the buffer that is pointed to by Resolved_name. Resolved_name is assumed to be of sufficient size to contain the pathname that is derived by the realpath service. If the generated pathname is larger than PATH_MAX, the return value is -1 and Return_code is ERANGE.

Related services

- "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215
- "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464
- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819

Characteristics and restrictions

There are no restrictions on the use of the realpath service.

Examples

For an example using this callable service, see "BPX1RPH (realpath) example" on page 1183.

recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer

Function

The recv callable service receives data on a socket and stores it in a buffer. If no messages are available at the socket, the service either waits for a message to arrive, or fails with EWOULDBLOCK — depending on whether the socket has been defined as blocking or nonblocking, and whether the SO_RCVTIMEO socket option is in effect.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE: (BPX1RCV)	31-bit
AMODE: (BPX4RCV)	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RCV,(Socket_descriptor,
              Buffer_length,
              Buffer,
              Buffer_alet,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RCV with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the receive is to be done.

Buffer_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Buffer.

Buffer
Supplied parameter

Type: Character

Length:
Length specified by Buffer_length.

The name of a field into which the data is received.

Buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer_alet of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains information about how the data is to be received. See "BPXYMSGF — Map the message flags" on page 997 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recv service returns one of the following:

- The number of bytes received into the buffer, if the request is successful. A value of `0` indicates that the connection is closed.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recv service stores the return code. The recv service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The recv service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINTR	A signal interrupted the **recv()** function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	The socket is marked shutdown for read. The following reason code can accompany the return code: JRSocketClosed.
EIO	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EWOULDBLOCK	• The socket is marked nonblocking and no data is waiting to be received, or the SO_RCVTIMEO timeout value was reached before data was available. • The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recv service stores the reason code. The recv service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The recv callable service applies only to connected sockets. It can be used with datagram or stream sockets. For datagram sockets, the recv service returns the entire datagram that was sent, providing that the datagram fits into the specified buffers. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of the recv service may be needed to return all the data.
2. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

"send (BPX1SND, BPX4SND) — Send data on a socket" on page 640

Characteristics and restrictions

There are no restrictions on the use of the recv service.

Examples

For an example using this callable service, see "BPX1RCV (recv) example" on page 1178.

recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer

Function

The recvfrom callable service receives data on a socket and stores it in a buffer. It can be used by an application program to receive data from sockets. When no data is available at the socket, the service either waits for data to arrive, or returns an EWOULDBLOCK — depending on whether the socket is defined as blocking or nonblocking, and whether the SO_RCVTIMEO socket option is in effect.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1RFM):	31-bit task or SRB mode
AMODE (BPX4RFM):	64-bit task mode only
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RFM,(Socket_descriptor,
              Buffer_length,
              Buffer,
              Buffer_alet,
              Flags,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RFM with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the recvfrom is to be done.

Buffer_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Buffer.

Buffer
Supplied parameter

Type: Character

Length:
Length specified by Buffer_length

The name of a field into which the data is to be received.

Buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer_alet of `0` for the normal case of a buffer in the user's address space

(current primary address space). If a value other than 0 is specified for the Buffer_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains information about how the data is to be received. See "BPXYMSGF — Map the message flags" on page 997 for more information about the format of this field.

Sockaddr_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that, on input, contains the length of the Sockaddr buffer. On return, this field specifies the size required to represent the address of the connecting socket. If this value is larger than the size supplied on input, the information contained in Sockaddr is truncated to the length supplied on input. The value in this field should be less than 4096 bytes (4KB) in length, and should represent the maximum possible length of the Sockaddr on output.

Sockaddr
Supplied and returned parameter

Type: Structure

Length:
Length specified by Sockaddr_length

The name of a buffer area that, on return, contains the socket address of the sender of the data. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvfrom service returns one of the following:

- The number of bytes received into the buffer, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvfrom service stores the return code. The recvfrom service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The recvfrom service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINTR	A signal interrupted the recvfrom function before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	The socket is marked shutdown for read. The following reason codes can accompany the return code: JRSocketCallParmError, JRSocketClosed.
EIO	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EWOULDBLOCK	• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available. • The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvfrom service stores the reason code. The recvfrom service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The recvfrom callable service can be used with datagram or stream sockets. For datagram sockets, it returns the entire datagram that was sent, providing that the datagram fits into the specified buffer. The excess is discarded. For stream sockets, data is not discarded. Multiple invocations of recvfrom may be needed to return all the data.
2. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

"sendto (BPX1STO, BPX4STO) — Send data on a socket" on page 652

Characteristics and restrictions

There are no restrictions on the use of the recvfrom service.

Examples

For an example using this callable service, see "BPX1RFM (recvfrom) example" on page 1181.

recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers

Function

The recvmsg callable service receives messages on a socket and stores them in a set of buffers. The socket can be either connected or unconnected. If no messages are available at the socket, the service either waits for a message to arrive, or returns an EWOULDBLOCK — depending on whether the socket is defined as blocking or nonblocking, and whether the SO_RCVTIMEO socket option is in effect.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX2RMS):	31-bit
AMODE (BPX4RMS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX2RMS,(Socket_descriptor,
              Message_hdr,
              Flags,
              Iov_alet,
              Iov_buffer_alet,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RMS with the same parameters. All addresses in the Message_hdr structure are doublewords.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the recvmsg is to be done.

Message_hdr
Supplied parameter

Type: Structure

Length:
The length of BPXYMSGH

The name of a field that contains the message header, which describes how the message is to be received. See "BPXYMSGH — Map the message header" on page 999 for more information about the format of this field. In 64-bit mode, Message_hdr contains doubleword pointer subfields, and points to an Iov_struct structure that contains doubleword pointer and length subfields.

Flags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains information about how the data is to be received. See "BPXYMSGF — Map the message flags" on page 997 for more information about the format of this field.

Iov_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for the IOV structure that is specified in Message_hdr.

Iov_buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for the buffers that are pointed to by the IOV structure that is specified in Message_hdr.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvmsg service returns one of the following:

- The number of bytes that were read into the buffers, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvmsg service stores the return code. The recvmsg service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The recvmsg service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINTR	A signal interrupted the recvmsg service before any data was available. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	The socket is marked shutdown for read; or incorrect data was received as a parameter. The following reason codes can accompany the return code: JRInvalidMsgh, JRSocketClosed, JRSocketCallParmError.
EIO	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EWOULDBLOCK	• The socket is marked nonblocking and no data is waiting to be read, or the SO_RCVTIMEO timeout value was reached before data was available. • The socket is marked blocking, and the call has blocked for that time period which was specified in the SO_RCVTIMEO option without receiving any data. The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the recvmsg service stores the reason code. The recvmsg service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. The BPX2RMS/BPX4RMS call supersedes the BPX1RMS call, which is still available for migration purposes only.
3. If the sendmsg security label is not equivalent to the recvmsg security label when access rights are passed on the sendmsg, the new descriptors are not created.
4. The number of buffers that are pointed to by the IOV structure in Message_hdr may not exceed IOV_MAX (defined in "BPXYIOV — Map the I/O vector structure" on page 986).
5. For the IP_PKTINFO ancillary data item, the program has to call setsockopt() or BPXIOPT(Sock#OptOptSetSockOpt) with the IP_RECVPKTINFO option to have the TCPIP stack pass the client's return information in the IN_PKTINFO structure as an ancillary data item on the recvmsg() or BPX2RMS call. That IP_PKTINFO data is used unchanged on the subsequent sendmsg() or BPX2SMS calls to have the reply flow out the same interface through which the request arrived.

Related services

"sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket" on page 648

Characteristics and restrictions

There are no restrictions on the use of the recvmsg service.

Examples

For an example using this callable service, see "BPX2RMS (recvmsg) example" on page 1182.

rename (BPX1REN, BPX4REN) — Rename a file or directory

Function

The rename callable service changes the name of a file or a directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1REN):	31-bit
AMODE (BPX4REN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1REN,(Old_name_length,
              Old_name,
              New_name_length,
              New_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4REN with the same parameters.

Parameters

Old_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file or directory that is to be renamed.

Old_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Old_name_length parameter

The name of a field, of length Old_name_length, that contains the name of the existing file or directory.

New_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname that is to be given to the existing file or directory.

New_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the New_name_length parameter

The name of a field, of length New_name_length, that contains the new pathname of the file or directory.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rename service returns `0` if the request is successful, or -1 if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rename service stores the return code. The rename service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The rename service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: • The process did not have search permission on some component of the old or new pathname; or did not have write permission on the parent directory of the file or directory that is to be renamed. • The S_ISVTX flag is set for the directory that contains Old_name. The caller is neither the owner of Old_name nor the owner of the parent directory, nor does the caller have appropriate privileges (see "Authorization" on page 8). • New_name refers to an existing file. The S_ISVTX flag is set for the directory containing New_name, and the caller is neither the owner of New_name nor the owner of the parent directory, nor does the caller have appropriate privileges.
EAGAIN	One of the files or directories was temporarily unavailable. The following reason code can accompany the return code: JRInvalidVnode.
EBUSY	Old_name and New_name specify directories but one of them cannot be renamed because it is in use as a root or a mount point, or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation. The following reason code can accompany the return code: JRIsFSRoot.
EINVAL	This error is returned for one of the following reasons: • Old_name is part of the pathname prefix of New_name. • Old_name refers to either `.` (dot) or `..` (dot-dot). • New_name refers to either `.` (dot) or `..` (dot-dot). The following reason codes can accompany the return code: JRDotOrDotDot and JROldPartOfNew.
EISDIR	New_name identifies a directory, but Old_name is not a directory. The following reason code can accompany the return code: JRNewIsDir.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Old_name or New_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Old_name or New_name.

Return_code	Explanation
ENAMETOOLONG	Old_name or New_name is longer than 1023 bytes; or a component of one of those names is longer than 255 bytes. Name truncation is not supported.
ENOENT	No file or directory name Old_name was found; or either Old_name or New_name was not specified. The following reason code can accompany the return code: JROldNoExist.
ENOSPC	The directory that is intended to contain New_name cannot be extended.
ENOTDIR	A component of either pathname prefix is not a directory; or Old_name is a directory and New_name is a file that is not a directory. The following reason code can accompany the return code: JRNewNotDir.
ENOTEMPTY	New_name specifies a directory, but the directory is not empty. It contains files or subdirectories.
EROFS	Performing the requested service would make it necessary to write on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.
EXDEV	Old_name and New_name identify files or directories on different file systems. Renaming across file systems is not supported. The following reason code can accompany the return code: JRDiffFileSets.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rename service stores the reason code. The rename service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The rename service changes the name of a file or directory from Old_name to New_name. When the renaming request completes successfully, the change and modification times for the parent directories of Old_name and New_name are updated.

   For renaming to succeed, the calling process needs write permission for the directory that contains Old_name and the directory that contains New_name. If Old_name and New_name are the names of directories, the caller does not need write permission for the directories themselves.
2. If the S_ISVTX flag is set for the directory that contains Old_name, one of the following conditions must be true, or the request will fail with EACCES:
   - The caller is the owner of the file named Old_name
   - The caller is the owner of the parent directory that contains Old_name
   - The caller has appropriate privileges (see "Authorization" on page 8)

   If the S_ISVTX flag is set for the directory that contains New_name, where New_name refers to an existing file, one of the following conditions must be true, or the request will fail with EACCES:
   - The caller is the owner of the file named New_name
   - The caller is the owner of the parent directory containing New_name
   - The caller has appropriate privileges

3. **Renaming files**:
   - If Old_name and New_name are links that refer to the same file, the rename service simply returns successfully.
   - If Old_name is the name of a file, New_name must also name a file, not a directory. If New_name is an existing file, it is unlinked, and then the file that is specified as Old_name is given New_name. The pathname New_name always stays in existence. At the beginning of the operation, New_name refers to its original file, and at the end, it refers to the file that used to be Old_name.
   - The rename will fail with EBUSY if New_name refers to an existing file that is currently open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to "open (BPX1OPN, BPX4OPN) — Open a file" on page 447 for details about the NFS share reservations.
4. **Renaming directories**:
   - If Old_name is the name of a directory, New_name must also name a directory, not a file. If New_name is an existing directory, it must be empty, containing no files or subdirectories. If it is empty, it is removed, as described in "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615.
   - New_name cannot be a directory under Old_name; that is, the old directory cannot be part of the pathname prefix of the new one.

Related services

- "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327
- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the rename service.

Examples

For an example using this callable service, see "BPX1REN (rename) example" on page 1181.

resource (BPX1RMG, BPX4RMG) — Measure resources

Function

The resource callable service gets system-wide resource measurement data from the kernel address space.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RMG):	31-bit
AMODE (BPX41RMG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RMG,(Data_area_length,
              Data_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RMG with the same parameters.

Parameters

Data_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Data_area, which the resource service is to fill with resource measurement information.

Data_area
Supplied parameter

Type: Structure

Length:
Specified by the Data_area_length parameter

The name of a field of length Data_area_length, which the resource service is to fill with resource measurement information. This field is mapped by the macro BPXYRMON. For the structure of Data_area, see "BPXYRMON — Map resource monitor data" on page 1034.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the resource service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the resource service stores the return code. The resource service returns Return_code only if Return_value is -1. See *z/OS*

UNIX System Services Messages and Codes for a complete list of possible return code values. The resource service can return the following value in the Return_code parameter:

Return_code	Explanation
EINVAL	Incorrect argument.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the resource service stores the reason code. The resource service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Some values that are returned by the resource service are continually wrapping counters. At the first call to the resource service, these values should be stored. At subsequent calls, the growth in these values should be calculated by the caller. The following list describes the normal use of wrapping counters that are returned by the resource service:
   a. A first call to the resource service returns the current value. (For example, X'FFFFFFD0' is returned for a system call count.)
   b. After some time interval expires, a second call to the resource service returns the new value. (For example, X'00000028' is returned for a system call count.)
   c. At this point, the increase in the counter can be calculated by the calling application. (In this case, we can calculate that X'58', or 88, system calls have been processed between the first resource service request and the second.)

Characteristics and restrictions

There are no restrictions on the use of the resource service.

Examples

For an example using this callable service, see “BPX1RMG (resource) example” on page 1182.

rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning

Function

The rewinddir callable service "rewinds," or resets, to the beginning of, an open directory. The next call to the readdir service reads the first entry in the directory.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RWD):	31-bit
AMODE (BPX4RWD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RWD (Directory_file_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RWD with the same parameters.

Parameters

Directory_file_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the directory file descriptor that was returned when the directory was opened (see "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rewinddir service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rewinddir service stores the return code. The rewinddir service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The rewinddir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The Directory_file_descriptor parameter does not represent an open directory. The following reason code can accompany the return code: JRRwdFileNotDir.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rewinddir service stores the reason code. The rewinddir service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

If the contents of the directory that you specify have changed since the directory was opened, a call to the rewinddir service resets the pointer into the directory to the beginning. A subsequent call to the readdir service reads from the start of the directory and obtains the new contents.

Related services

- "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105
- "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452
- "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577

Characteristics and restrictions

There are no restrictions on the use of the rewinddir service.

Examples

For an example using this callable service, see "BPX1RWD (rewinddir) example" on page 1184.

rmdir (BPX1RMD, BPX4RMD) — Remove a directory

Function

The rmdir callable service removes a directory. The directory must be empty.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1RMD):	31-bit
AMODE (BPX4RMD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1RMD,(Directory_name_length,
              Directory_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4RMD with the same parameters.

Parameters

Directory_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Directory_name.

Directory_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Directory_name_length parameter

The name of a field that contains the pathname of the directory to be removed. The length of this field is specified in Directory_name_length.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rmdir service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rmdir service stores the return code. The rmdir service returns Return_code only if Return_value is -1. See *z/OS UNIX*

System Services Messages and Codes for a complete list of possible return code values. The rmdir service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: • The process did not have search permission for some component of Directory_name, or did not have write permission for the directory that contains the directory that is to be removed. • The S_ISVTX flag is set for the parent directory of the directory that is to be removed, and the caller is not the owner of that directory or the owner of the parent directory, nor does the caller have appropriate privileges (see "Authorization" on page 8).
EBUSY	The directory cannot be removed, because it is being used by a process. The following reason code can accompany the return code: JRRootNode.
EINVAL	The argument that was supplied was incorrect. Examples of incorrect arguments are dot and dot-dot. The following reason code can accompany the return code: JRDotOrDotDot.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Directory_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Directory_name.
ENAMETOOLONG	The name of the directory is longer than 1023 characters; or some component of the pathname is longer than 255 characters. This could be as a result of encountering a symbolic link during resolution of Directory_name, where the substituted string is longer than 1023 characters.
ENOENT	The directory that was specified by Directory_name was not found; or no Directory_name parameter was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of Directory_name is not a directory. The following reason code can accompany the return code: JRPathNotDir.
ENOTEMPTY	The directory contains files or subdirectories.
EROFS	The directory that is to be removed is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the rmdir service stores the reason code. The rmdir service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The directory must be empty.
2. If the directory is successfully removed, the change and modification times for the parent directory are updated.

3. If the link count of the directory becomes zero and no process has the directory open, the directory itself is deleted. The space that is occupied by the directory is freed for new use, and the contents of the file are lost.
4. If any process has the directory open when the last link is removed, the directory itself is not removed until the last process closes the directory. New files cannot be created under a directory after the last link is removed, even if the directory is still open.
5. If the S_ISVTX flag is set for the parent directory of the directory that is to be removed, one of the following conditions must be true, or the request will fail with EACCES:
   - The caller is the owner of the directory to be removed
   - The caller is the owner of the parent directory
   - The caller has appropriate privileges (see "Authorization" on page 8)

Related services

- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the rmdir service.

Examples

For an example using this callable service, see "BPX1RMD (rmdir) example" on page 1182.

select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues

Function

The select/selectex callable service checks the I/O status of multiple open file descriptors and message queues. The file descriptors can be for character special files, pipes, sockets, or files.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SEL):	31-bit
AMODE (BPX4SEL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SEL,(Number_msgsfds,
              Read_list_length,
              Read_list,
              Write_list_length,
              Write_list,
              Exception_list_length,
              Exception_list,
              Timeout_pointer,
              Ecb_pointer,
              User_option_field,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SEL with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Number_msgsfds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword of which the first halfword (the high-order 16 bits) contains the number of message queues and the second halfword (the low-order 16 bits) contains the number of file descriptors.

The number of file descriptors should be the highest file descriptor that is being checked for status, plus 1.

For example, if you are interested in the I/O status of file descriptors 5 and 9, the second halfword of Number_msgsfds would be 10. Ten is the number of file descriptors that are contained in each of the bit sets (fd 0 through 9 equals 10 fds), and 10 is the highest file descriptor that is being checked, plus 1 (9 plus 1 equals 10). If you want to check file descriptors for status along with message queues, the highest file descriptor you can specify is 2047.

The number of message queues indicates the number of elements (queue IDs) in each of the arrays contained in Read_list, Write_list, and Exception_list. For example, if you specify a value of 10 in the first halfword of Number-msgsfds, it is expected that arrays of 10 elements each are given in Read_list, Write_list, and Exception_list. If you specify a value of 0, it is assumed that no arrays are given and that no message queues are to be checked. The maximum number of message queues that you can specify is 32 767.

Note: In order to select on descriptor numbers higher than 65 534, the descriptor limit of the process must be at least 65 536, and one of the bit lists that is passed must be at least 8192 bytes long. When both of these facts are true, and the fullword value is between 65 536 and the system descriptor maximum, the entire fullword parameter will be assumed to represent the number of file descriptors, and no message queues will be processed.

Read_list_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length, in bytes, of the Read_list. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the Read_list is not checked by the select service.

Read_list
Supplied and returned parameter

Type: Structure

Length:
Length specified by Read_list_length

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. Note that the bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for reading. The format of the bits can be specified with the User_option field. On return, the bits that are set indicate the descriptors that are ready for reading.

If Read_list contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and are ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

Write_list_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length, in bytes, of the Write_list. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the Write_list is not checked by the select service.

Write_list
Supplied and returned parameter

Type: Structure

Length:
Length specified by Write_list_length

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. Note that the bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for writing. The format of the bits can

be specified with the User_option field. On return, the bits that are set indicate the descriptors that are ready for writing.

If Write_list contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and are ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

Exception_list_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length in bytes of the Exception_list. The length is actually the sum of the length (rounded up to a multiple of 4 bytes) of the bit set specifying file descriptors and the length of the array of message queue identifiers. When both file descriptors and message queues are specified, this field should contain a value greater than 256 bytes. If 0 is specified, the Exception_list is not checked by select.

Exception_list
Supplied and returned parameter

Type: Structure

Length:
Length specified by Exception_list_length

The name of a structure that contains the bit set for the specified file descriptors and/or the array of message queue identifiers. Note that the bit set must be padded with extra bytes, if necessary, to round up its length to the next multiple of 4 bytes. The bits in the bit set should be turned on for the corresponding descriptors to be checked for exceptions. The format of the bits can be specified with the User_option field. On return, the bits that are set indicate the descriptors that have had exceptions.

If Exception_list contains both a bit set and an array of message queue identifiers, the bit set must be 256 bytes in length. If only file descriptors are to be checked, the bit set can have any valid size.

Each element of the array of message queue identifiers is 4 bytes in length. Elements with a value of -1 are acceptable and will be ignored. On return, the array is altered such that message queue identifiers that do not meet the criterion are replaced with a value of -1.

Timeout_pointer
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains a pointer to a timeout value that controls how the file descriptors are checked:

1. **Wait indefinitely**:

If the timeout_pointer is zero, the select system call waits (indefinitely) until one of the selected descriptors is ready.

2. **Wait for a specified period of time**:

   If the timeout_pointer is nonzero, it points to a timeout value mapped by the BPXYSELT macro, which contains the number of microseconds and/or seconds to wait for one of the conditions to occur before returning to the caller. The maximum time that can be specified is 31 days. See "BPXYSELT — Map the timeout value for the select syscall" on page 1037 for more information.

   - Microseconds can be a value in the range from 0 to 1 000 000. (1 000 000 microseconds equal 1 second).
   - Seconds can be a value in the range from 0 to 2 678 400. (2 678 400 seconds equal 31 days).

   Note: Microseconds and seconds are added together to determine the timeout value. If the timeout value is more than 0 and less than 300 microseconds, the value is rounded up to 300 microseconds.

3. **No Waiting**:

   If the timeout value is 0, select returns immediately after checking the selected descriptors; no waiting is done.

Ecb_pointer
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

This can be any of the following values:

1. The name of a fullword (doubleword) field that contains a pointer to a user event control block. To specify this usage of Ecb_pointer, set the high-order bit in Ecb_pointer to B'0'.

   If a doubleword is used for Ecb_pointer, the high half must be set to zero (ECBs must be below the bar). In this case, the high-order bit that indicates Ecb_pointer usage is the high bit in the lower half of the doubleword.

2. The name of a fullword (doubleword) field that contains a pointer to a list of ECBs. To specify this usage of Ecb_pointer, set the high-order bit in Ecb_pointer B'1'.

   The list can contain the pointers for up to 1013 ECBs. The high-order bit of the last pointer in the list must be set to B'1'. If the input Ecb_pointer is a doubleword, the high half must be zero, and the bit that is checked is the high-order bit of the lower half of the doubleword. If the high-order bit is a 1, the lower half of the doubleword points to a list of Ecb_pointers. All Ecb_pointers in the list must be 31-bit pointers.

3. The name of a fullword (doubleword) field that contains 0. This indicates that no ECBs are specified.

User_option_field
Supplied and returned parameter

Type: Integer

Length:
Fullword

A dual-purpose field that is used as input to specify the format of the read, write, and exception bit lists, and as output to contain the first selected file descriptor that was not supported by the select service.

On input, specify one of the following (the values are defined in "BPXYSEL — Map the select options" on page 1036):

- SEL#BITSBACKWARD – Bit-backward order by word:

  Bits are read from right to left within each word, with the low-order bit on the right and the high-order bit on the left. For example:

```
 Word 1              Word 2                  Word 3
------------------  ----------------------  ----------------------
31 30 29...3 2 1 0   63 62 61...35 34 33 32   95 94 93...67 66 65 64
------------------  ----------------------  ----------------------
```

  Note: In this example, file descriptor 0 is represented by the last bit on the right in Word 1.

- SEL#BITSFORWARD – Bit-forward order by word:

  Bits are read from left to right within each word, with the low-order bit on the left and the high-order bit on the right. For example:

```
 Word 1              Word 2                  Word 3
------------------  ----------------------  ----------------------
0 1 2 3...29 30 31   32 33 34 35...61 62 63   64 65 66.67...93 94 95
------------------  ----------------------  ----------------------
```

  Note: In this example, file descriptor 0 is represented by the first bit on the left in Word 1.

On output, the select service returns one of the following:

- -1, if all the selected file descriptors supported the select callable service.
- The first selected file descriptor that did not support the select callable service.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the select service returns one of the following:

- The number of read, write, and exceptional conditions that were found among the given message queues; and the number of read, write, and exceptional conditions that were found among the specified file descriptors. These two values are returned, respectively, in the first halfword and the second halfword of Return_value. Should the return value for message queues exceed 32 767, only 32 767 is reported. This is to ensure that Return_value does not appear to be negative. Should the return value for file descriptors be greater than 65 535, only 65 535 is reported.
- 0, if the timeout value expired before any of the conditions were met.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the select service stores the return code. The select service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The select service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINTR	The select service request was interrupted by a signal for the caller.
EINVAL	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRNoLists, JRListTooShort, JRMsOutOfRange, JRInvUserOp, JRSecOutOfRange, JRNoFds, JRTooManyMsgQIds,JRTooManyFds, JRListLenBad.
EIO	One of the descriptors in the select mask has become inoperative, and it is being included repeatedly in a select, even though other operations against this descriptor have been failing with EIO. A socket descriptor can become inoperative, for example, if TCP/IP is shut down. When a descriptor fails, a failure from select does not tell you which descriptor has failed. The select call usually succeeds, and the descriptors are reported to you as being ready for whatever events were specified on the select call. When the descriptor is subsequently used on a receive or other operation, you will receive the EIO failure and can then react to the problem with that individual descriptor. In general, you would close() the descriptor and remove it from the next select mask. If the individual descriptor's failing return code is ignored, however, and an inoperative descriptor is repeatedly selected on and used (even though each time it is used the call fails with EIO), the select call itself will eventually fail with EIO.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the select service stores the reason code. The select service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The bit set for the read_list, write_list, and exception_list is a string of bits such that if X is an element of the set, the bit that represents X is set to 1. For example, if descriptor 1 is to be checked, bit 1 should be turned on in the bit set. Here is how that byte would look:
   - Bit-forward order: B'01000000'.
   - Bit-backward order: B'00000010'.
2. When a positive value is specified for the number of file descriptors:
   - At least one bit set (read, write, or exception) must be specified, and its length must be large enough (rounded up to the next multiple of 4) to contain the bit that represents the largest descriptor you specified.

- If more than one bit set is specified, each bit set must be the same length.

  For example, if you want to check the read status for file descriptor 59 and the write status for file descriptor 6:

  Number of fds = 60

  Read_list_length = 8
  Read_list = the bit representing fd 59 is set on (see User_option_field to determine which bit that would be)

  Write_list_length = 8
  Write_list = the bit representing fd 6 is set on (see User_option_field to determine which bit that would be)

  Exception_list_length = 0

3. When both the first and second halfwords of Number_msgsfds contain a positive value, the Read_list, Write_list, and Exception_list must each contain both a bit set and an array of message queue identifiers, unless a value of 0 is specified for its length.

   When the fullword value is between 65 536 and the system descriptor maximum, one of the lists that is passed is at least 8 192 bytes long, and the descriptor limit of the process is at least 65 536, the fullword value is considered as the number of descriptors in the lists, and no message queues will be processed.

   The following example illustrates what you must do:

   Suppose you want to check the read status for file descriptors 3 and 5 and the write status for message queues whose identifiers are 7 and 8.

   Number of fds = 6 (the largest fd plus 1)
   Number of message queues = 2

   Read_list_length = 264 (256 byte bit set length + 8 byte array length) Read_list = the 256-byte bit set with appropriate bits set on for fds 3 and 5, followed by a two-element array that contains the value of -1 in both elements.

   Write_list_length = 264 (same length as for read) Write_list = the 256-byte bit set with all its bits set off followed by the two-element array that contains the numbers 7 and 8.

   Exception_list_length = 0

4. You can use the select service as a timer-only function by specifying zero for the Read_list_length, Write_list_length, and Exception_list_length, and by specifying timeout_pointer and timeout_value. If you specify zero for timeout_pointer, the select service blocks forever. If you specify zero for timeout_value, no blocking is done, and the select service returns immediately to the caller.
5. You can also specify an Ecb_pointer with the timer only function.
6. Regular files are always ready for reading and writing.
7. When the storage key of the first (or only) ECB matches the caller's PSW key, the kernel performs the wait in the caller's PSW key; otherwise, the kernel performs the wait in the TCB key (TCBPFK). However, if the caller is running in key 0, then the kernel performs the wait in key 0, regardless of the storage key.

Characteristics and restrictions

There are no restrictions on the use of the select service.

Examples

For an example using this callable service, see "BPX1SEL (select) example" on page 1186.

semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations

Function

The semctl service provides semaphore control operations. These functions include reading and changing the values of semaphores and removing a set of semaphores from the system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SCT):	31-bit
AMODE (BPX4SCT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SCT,(Semaphore_ID,
              Semaphore_Number
              Command,
              SValue  |  Argument_address ( Buffer | Array),
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SCT with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Semaphore_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the semaphore identifier.

Semaphore_Number
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the semaphore number. Semaphore_Number ranges from 0 to Number_of_Semaphores - 1. Use with Sem_GETVAL, Sem_SETVAL, Sem_GETNCNT and Sem_GETZCNT. This argument is ignored for all other commands.

Command
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates the semaphore command that is to be executed. For the structure that contains these constants, see "BPXYSEM — Map interprocess communication semaphores" on page 1037 and "BPXYIPCP — Map interprocess communication permissions" on page 987. The values for Command are:

Sem_GETVAL
Returns the value of semval for the requested Semaphore_Number, if the current process has read permission.

Sem_SETVAL
Sets the semval for the requested Semaphore_Number to the contents of SValue, if the current process has alter permission. When this Command is successfully executed, the semadj values that correspond to this semaphore for all processes are cleared.

Sem_GETPID
Returns the ID of the most recent process to update the semaphore, if the current process has read permission.

Sem_GETNCNT
Returns the number of threads waiting on the semaphore to become greater than the current value, if the current process has read permission. See "semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations" on page 636.

Sem_GETZCNT
Returns the number of threads waiting on the semaphore to become zero, if the current process has read permission. See "semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations" on page 636.

Sem_GETALL
Stores all semaphore semvals into the array of halfwords that is pointed to by the Argument_address parameter, if the current process has read permission. It is the caller's responsibility to ensure that the storage that is allocated for the array is large enough to hold all semaphore elements. The number of semaphore values stored is SEM_NSEMS, which may be obtained using the Ipc_STAT command.

Sem_SETALL
Sets semvals according to the array that is pointed to by the Argument_address parameter, if the current process has alter permission. Each value must be zero or positive. When this Command

is successfully executed, the semadj values that correspond to each specified semaphore in all processes are cleared. It is the caller's responsibility to ensure that the storage that is allocated for the array is large enough for all semaphore elements. The number of semaphore values read is SEM_NSEMS, which may be obtained using the Ipc_STAT command.

If IPC_BINSEM is specified on the semget call, this option should not be used when there is a possibility that other threads could be performing semaphore operations on this semaphore, as there may be no serialization while the semaphore values are being updated.

Ipc_STAT
Obtains status information about the semaphore that is identified by the Semaphore_ID parameter, if the current process has read permission. This information is stored in the buffer that is pointed to by the Argument_address parameter.

Ipc_SET
Sets the value of the IPC_UID, IPC_GID and IPC_MODE from the SEMID_DS data structure that is associated with Semaphore_ID into the SEMID_DS structure that is pointed to by Argument_address. Any value for IPC_UID and IPC_GID may be specified. Only the mode bits that are documented for semget argument Semaphore_Flags may be set. This Command can only be executed by a process that has an effective user ID equal either to that of a process with appropriate privileges (see "Authorization" on page 8) or to the value of IPC_CUID or IPC_UID in the SEMID_DS data structure that is associated with Semaphore_ID. This information is taken from the buffer that is pointed to by the Argument_address parameter. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037, SEMID_DS DSECT.

Ipc_RMID
Removes the semaphore identifier that is specified by Semaphore_ID from the system and destroys the set of semaphores and the SEMID_DS data structure that are associated with it. This Command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges (see "Authorization" on page 8) or to the value of IPC_CUID or IPC_UID in the SEMID_DS data structure that is associated with Semaphore_ID.

SValue
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the value to be set for the semaphore that is identified by the Semaphore_Number.

Argument_address (Buffer | Array)
Supplied parameter

Type: Address

Length:
Fullword

The name of a field that contains the address of the Buffer, Array or a null.

Table 19. Calling parameters and commands

Number	Command	Buffer | Array	Return Value
Sem No.	GETVAL	NA	SemVal, -1
Sem No.	SETVAL	SValue	0, -1
Sem No.	GETPID	NA	Pid, -1
Sem No.	GETNCNT	NA	Count, -1
Sem No.	GETZCNT	NA	Count, -1
NA	GETALL	Array, output	0, -1
NA	SETALL	Array, input	0, -1
NA	STAT	Buffer, output	0, -1
NA	SET	Buffer, input	0, -1
NA	RMID	NA	0, -1

Buffer
Supplied and returned parameter

Type: Structure

Length:
Length of SEMID_DS.

The name of a fullword (doubleword) field that contains the address of a data area that is mapped by SEMID_DS. This field is used for stat and set.

Array
Supplied and returned parameter

Type: Structure

Length:
GETALL - An array of 2-byte integers for each semaphore in the set equal to (SEM_NSEMS * 2).

SETALL - A 2-byte integer for each semaphore in the set equal to (SEM_NSEMS * 2).

SETVAL - A 4-byte integer for the specified semaphore. The valid range is 0 through the system limit.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semctl service returns -1, if not successful, or the following when successful:

GETVAL
The value of semval is returned

GETPID
The value of sempid is returned

GETNCNT
The value of semncnt is returned

GETZCNT
The value of semzcnt is returned

All others
A value of zero is returned

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semctl service stores the return code. The semctl service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The semctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Operation permission (read or alter) is denied to the calling process. The following reason code can accompany the return code: JRIpcDenied.
EFAULT	The Buffer or ARRAY parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code: JRBadAddress.
EINVAL	One of the following errors occurred: • Semaphore_ID is not a valid semaphore identifier. • Semaphore_Number is less than zero or greater than or equal to the number of semaphores in this set. • The Command parameter is not a valid command. • The mode bits were not valid (ipc_SET). The following reason codes can accompany the return code: JRIpcBadFlags, JRIpcBadID, JRSema4BadSemN and JRBadEntryCode.
EPERM	The Command was IPC_RMID or IPC_SET, and the effective user ID of the caller is not that of a process with appropriate privileges (see "Authorization" on page 8) and is not the value of IPC_CUID or IPC_UID in the SEMID_DS data structure that is associated with Semaphore_ID. The following reason code can accompany the return code: JRIpcDenied.
ERANGE	The SETVAL or SETALL value exceeds the system-imposed maximum that is defined by SEM#MAX_VAL in BPXYSEM. The following reason code can accompany the return code: JRSema4BadValue.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semctl service stores the reason code. The semctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Each semaphore in the semaphore set is represented by a data structure that is defined as follows:

   semval
   Unsigned halfword semaphore value

   sempid
   Process ID of the last operation

   semncnt
   Unsigned halfword number of processes waiting for semval to become greater than the current value

   semzcnt
   Unsigned halfword number of processes waiting for semval to become zero
2. The Semaphore_ID was obtained from semget (BPX1SGT, BPX4SGT).
3. A semadj variable is maintained by the process for all of its threads. This adjustment value allows the kernel to restore semaphore values if a process terminates before it can issue a semop. It is the application's responsibility to maintain semadj values for process termination.
4. Ipc_SET can change permissions, and may affect a thread's ability to use the semaphore functions.
5. When a semaphore ID is removed (Ipc_RMID) from the system, all waiting threads regain control with RV=-1, RC=EIDRM, and RC=JRIpcRemoved.
6. The remove is complete by the time control is returned to the caller.

Related services

- “mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources” on page 418
- “semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores”
- “semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations” on page 636

Characteristics and restrictions

The invoker is restricted by ownership and read and read-write permissions that are defined by semget and semctl Ipc_SET.

Examples

For an example using this callable service, see “BPX1SCT (semctl) example” on page 1184.

semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores

Function

The semget function creates a new semaphore set or finds an existing semaphore set. The semaphore set ID is returned.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1SGT):	31-bit
AMODE (BPX4SGT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SGT,(Key,
              Number_of_Semaphores,
              Semaphore_Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SGT with the same parameters.

Parameters

Key
Supplied parameter

Type: Integer

Length:
Fullword

Identification for this semaphore set. This is either a user-defined value that serves as a lookup value to determine if the semaphore set already exists, or the reserved value Ipc_PRIVATE. (See "BPXYIPCP — Map interprocess communication permissions" on page 987. Ipc_PRIVATE is sometimes used when a process does not want to share a semaphore set, or when it wants to privately control access to it by other processes.)

Number_of_Semaphores
Supplied parameter

Type: Integer

Length:
Fullword

The number of semaphores that are to be allocated to this set. This value may be zero if the application knows that the semaphore set should already be created for the specified key parameter. A zero value is not allowed with Ipc_CREAT or Ipc_PRIVATE. The maximum for this variable is controlled by the installation. For an existing semaphore identifier, this variable must not be greater than the number of semaphores in that set.

Semaphore_Flags
Supplied parameter

Type: Structure

Length:
Fullword

Valid values for this field include any combination of the following (additional bits cause an EINVAL):

Ipc_CREAT
Creates a message queue if the key specified does not already have an associated ID. Ipc_CREAT is ignored when Ipc_PRIVATE is specified.

Ipc_EXCL
Causes the semget function to fail if the key specified has an associated ID. Ipc_EXCL is ignored when Ipc_CREAT is not specified, or when Ipc_PRIVATE is specified.

Ipc_BINSEM
Binary semaphore. The semaphore must behave in a binary manner: the number of semaphore operations must be 1, and the semop must be either 1 with a semval of 1, or -1 with a semval of 0 or 1. Specifying the SEM_UNDO flag in the SEM_FLGS field of BPXYSEM on a semop() request against a binary semaphore allows the semaphore to be released when a process exits without releasing it. The use of this flag improves performance if the PLO instruction is available on the hardware.

Ipc_SHORTHOLD
Indicates that the application will hold the resource that is being serialized for extremely short intervals of time. When the IPC_BINSEM flag is also specified, the default first-in-first-out ordering of semaphore obtain requesters is bypassed, allowing short duration requesters to cut to the front of the wait chain.

S_IRUSR
Permits the process that owns the semaphore set to read it.

S_IWUSR
Permits the process that owns the semaphore set to alter it.

S_IRGRP
Permits the group that is associated with the semaphore set to read it.

S_IWGRP
Permits the group that is associated with the semaphore set to alter it.

S_IROTH
Permits others to read the semaphore set.

S_IWOTH
Permits others to alter the semaphore set.

The values that begin with the "Ipc_" prefix are defined in BPXYIPCP and are mapped onto S_TYPE, which is in BPXYMODE. (See “BPXYIPCP — Map interprocess communication permissions” on page 987 and “BPXYMODE — Map the mode constants of the file services” on page 996.)

The values that begin with the "S_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

This operand is ignored if the semaphore set is already defined to the system.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semget service returns the semaphore identifier or, if unsuccessful, -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semget service stores the return code. The semget service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The semget service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	A semaphore identifier exists for the Key parameter, but access permission, as specified by the low-order 9 bits of the Semaphore_Flags parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRIpcDenied.
EEXIST	A semaphore identifier exists for the Key parameter, and both Ipc_CREAT and Ipc_EXCL are specified. The following reason code can accompany the return code: JRIpcExists.
EINVAL	Number_of_Semaphores is not valid when: • The semaphore identifier exists for the Key parameter and Number_of_Semaphores exceeds the number of semaphores previously defined. • Number_of_Semaphores is zero. • Number_of_Semaphores exceeds the system limit. This system limit is set with the IPCSEMNSEMS parameter in a BPXPRM parmlib member. You can use the **ipcs -x** shell command to view this value. The Semaphore_Flags parameter includes bits that are not supported by this function. The following reason codes can accompany the return code: JRSema4BadNSems, JRSema4ZeroNSems, JRSema4BigNSems, and JRIpcBadFlags.
ENOENT	A semaphore identifier does not exist for the Key parameter and Ipc_CREAT was not set. The following reason code can accompany the return code: JRIpcNoExists.
ENOSPC	A semaphore identifier is to be created, but the system-imposed limit on the maximum number of allocated semaphore identifiers system-wide would be exceeded. This system limit is set with the IPCSEMNIDS parameter in the BPXPRM parmlib member. You can use *ipcs -x shell* command to the view this value. You can use the **ipcrm** shell command to remove unused semaphore identifiers. The following reason code can accompany the return code: JRIpcMaxIDs.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semget service stores the reason code. The semget service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Each semaphore in the semaphore set is represented by a data structure that is defined as follows:

   semval
   Unsigned halfword semaphore value

   sempid
   Process ID of last operation

   semncnt
   Unsigned halfword number of processes waiting for semval to become greater than current value

   semzcnt
   Unsigned halfword number of processes waiting for semval to become zero
2. When a semaphore set is created, the value of **semval** for all semaphores is set to zero.
3. As long as the semaphore ID is known and access is permitted, any thread can invoke semctl or semop without invoking semget.
4. This function returns the semaphore identifier that is associated with the Key parameter.
5. When it is successful, this function creates a data structure that is defined by SEMID_DS and an array that contains the number of semaphores specified, if one of the following is true:
   - The Key parameter is equal to Ipc_PRIVATE.
   - The Key parameter does not already have a semaphore identifier associated with it, and Ipc_CREAT is set.

   For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037.
6. Upon creation, the data structure that is associated with the new semaphore identifier is initialized as follows:
   - Ipc_CUID and Ipc_UID are set to the effective user ID of the calling process.
   - Ipc_CGID and Ipc_GID are set to the effective group ID of the calling process.
   - The low-order 9 bits of Ipc_MODE are equal to the low-order 9 bits of the Semaphore_Flags parameter.
   - SEM_NSEMS is set equal to the value of the Number_of_Semaphores parameter.
   - SEM_OTIME is set to 0 and SEM_CTIME is set to the current time.
7. If the Key parameter is not Ipc_PRIVATE, Ipc_EXCL is not set, and a semaphore identifier already exists for the specified Key parameter, the value of the Number_of_Semaphores parameter that is specified may not exceed the Number_of_Semaphores specified on the semget that created the semaphore set.
8. The semaphore set is removed from the system as soon as BPX1SCT/BPX4SCT (semctl RMID) is processed.

9. Users of Ipc_PRIVATE semaphore sets are responsible for removing them when they are no longer needed. Failure to do so ties up resources.
10. Semaphores created with the Ipc_BINSEM attribute show this bit, and may also show the Ipc_PLOinUse bit, in the S_MODE byte that is returned with the w_getipc request.

Related services

- "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890
- "semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations" on page 626
- "semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations"

Characteristics and restrictions

- There is a maximum number of semaphore sets and semaphores that are allowed in the system.
- The invoker is restricted by ownership, read, and read-write permissions that are defined by semget and semctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1SGT (semget) example" on page 1188.

semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations

Function

The semop service performs a group of semaphore operations atomically.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SOP):	31-bit
AMODE (BPX4SOP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SOP,(Semaphore_ID,
              Semaphore_Operations,
              Number_of_Semaphore_Operations,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SOP with the same parameters. The Semaphore_Operations parameter is a doubleword.

Parameters

Semaphore_ID
Supplied parameter

Type: Integer

Length:
Halfword

Specifies the semaphore identifier.

Semaphore_Operations
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

A fullword (doubleword) that points to an array of data structures mapped by SEM_BUF_ELE in "BPXYSEM — Map interprocess communication semaphores" on page 1037. The SEM_OP operations modify the semval for a specific semaphore in the semaphore set specified by SEM_NUM. All updates to the semaphores' semval are made atomically when this callable service returns successfully. Partial updates to semval are not performed. Each SEM_BUF_ELE element contains the following:

- SEM_NUM is a halfword semaphore number in the Semaphore_ID set. References to semval, sempid, semncnt, semzcnt are to this element in the semaphore set. SEM_NUM ranges from 0 to Number_of_Semaphore_Operations - 1.
- SEM_OP is a signed halfword with three different operations, described as follows:
  - SEM_OP < 0, evaluate semval + SEM_OP (remember that SEM_OP is negative). If the operation yields a negative number, the operation either returns to the caller (EAGAIN) or suspends execution of the calling thread until the operation yields a non-negative number. Semncnt is incremented for each thread that is waiting, and decremented when waiting is complete. When waiting is complete, semval = semval + SEM_OP.
  - SEM_OP > 0, set semval = semval + SEM_OP.
  - SEM_OP = 0, test semval. If not zero, the operation either returns to the caller (EAGAIN) or suspends execution of the calling thread until semval=0. Semzcnt is incremented for each thread that is waiting, and decremented when waiting is complete.
- SEM_FLGS – contains the Ipc_NOWAIT and Sem_UNDO bits. Ipc_NOWAIT causes SEM_OP=0 and SEM_OP<0 to return immediately with a return code of EAGAIN if the condition cannot be met. Otherwise, processing is suspended. Sem_UNDO instructs the process to maintain an adjustment value for SEM_OP ^= 0. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037.

Number_of_Semaphore_Operations
Supplied parameter

Type: Integer

Length:
Fullword

Contains the number of operations in Semaphore_Operations. A value of zero up to the maximum allowed by the system may be specified.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semop service returns 0 (all SEM_OP operations were performed) or -1 (none of the SEM_OP operations were performed).

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semop service stores the return code. The semop service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The semop service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Permission is denied. The following reason code can accompany the return code: JRIpcDenied.
EAGAIN	The operation would result in suspension of the calling process, but NOWAIT (see SEM_FLGS) was specified. The following reason code can accompany the return code: JRIpcRetry.
EDEADLK	The combination of operations can never be satisfied. This condition is detected by analysis of the operations that were requested and the system maximums, and does not include interactions with other threads. For example, an operation could add 1 to a semaphore, and a later operation in the same SEM_BUF could test it for zero. The following reason code can accompany the return code: JRDeadlock.
EFAULT	The Semaphore_Operations parameter specified an address that caused the service to program check. The following reason code can accompany the return code: JRBadAddress.
EFBIG	SEM_NUM is less than zero or is greater than or equal to the number of semaphores in the set specified by the Number_of_Semaphores parameter of the semget() call. The following reason code can accompany the return code: JRSema4BadSemN.
EIDRM	Semaphore_ID was removed from the system while the caller was waiting. The following reason code can accompany the return code: JRIpcRemoved.
EINTR	semop() was interrupted by a signal. The following reason code can accompany the return code: JRIpcSignaled.
EINVAL	The Semaphore_ID does not represent a semaphore set. The following reason code can accompany the return code: JRIpcBadID.

Return_code	Explanation
ENOSPC	The space that is allotted for all semaphore data would be exceeded by the addition of the UNDO structure for this request. The following reason code can accompany the return code: JRSemStorageLimit.
ERANGE	An operation would cause sem_val or sem_adj to overflow the system-imposed limit. These system limits are defined in BPXYSEM fields SEM#MAX_VAL and SEM#MAX_ADJ. The following reason codes can accompany the return code: JRSema4BadValue and JRSema4BadAdj.
E2BIG	Number_of_Semaphore_Operations exceeds the maximum allowed by the system. This system limit is set with the IPCSEMNOPS parameter in a BPXPRMxx parmlib member. You can use the *ipcs -x shell* command to view this value. The following reason code can accompany the return code: JRSema4BadNOps.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the semop service stores the reason code. The semop service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Each semaphore in the semaphore set is represented by an anonymous data structure, which is defined as follows:

   semval
   Unsigned halfword semaphore value

   sempid
   Process ID of last operation

   semncnt
   Unsigned halfword number of processes waiting for semval to become greater than current value

   semzcnt
   Unsigned halfword number of processes waiting for semval to become zero
2. A nonzero SEM_OP value requires write permission (else EACCES).
3. A zero SEM_OP value requires read permission (else EACCES).
4. Upon successful completion, sempid equals the process ID of the calling process.
5. Wait queue service is unpredictable.
6. Waiting is done on a thread basis. Multiple threads (even within a single process) could be waiting on the same semaphore.
7. Adjustments are maintained on a process basis, and can be changed by threads outside or within the process.
8. Within an array of semaphore operations, either all operations or none of the operations are performed.

9. Incorrect usage of semaphores may cause the application to become deadlocked and wait forever. Designing the semaphore hierarchy so that the semaphores are obtained in a specific order will avoid deadlocks.
10. If the Number_of_Semaphore_Operations is zero, the callable service returns successfully with no semaphore operation being performed.

Related services

- "mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources" on page 418
- "semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations" on page 626
- "semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores" on page 631

Characteristics and restrictions

The invoker is restricted by ownership, read, and read-write permissions that are defined by semget and semctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1SOP (semop) example" on page 1193.

send (BPX1SND, BPX4SND) — Send data on a socket

Function

The send callable service sends data on a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1SND):	31-bit task or SRB mode
AMODE (BPX4SND):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SND,(Socket_descriptor,
              Buffer_length,
              Buffer,
              Buffer_alet,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SND with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the send is to be done.

Buffer_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Buffer.

Buffer
Supplied parameter

Type: Character

Length:
Length specified by Buffer_length

The name of a field that contains the data that is to be transmitted.

Buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Buffer.

You should specify a Buffer_alet of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a field that contains information about how the data is to be sent. See "BPXYMSGF — Map the message flags" on page 997 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send service returns one of the following:

- The number of bytes sent from the buffer, if the request is successful. A value of `0` indicates that the connection is closed.
- -1, if the request is not successful.

Return_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send service stores the return code. The send service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The send service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	Socket_descriptor does not refer to a valid descriptor. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	Connection reset by peer.
EINTR	A signal interrupted the send before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
EIO	There has been a network or transport failure. The following reason code can accompany the return code: JRPrevSockError.
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	The socket is not connected. The following reason code can accompany the return code: JRSocketNotCon.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPIPE	An attempt was made to send to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a **SIGPIPE** signal.
EWOULDBLOCK	• The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available. • The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option. The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send service stores the reason code. The send service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The socket must be connected.
2. If there is not enough room to write the data to the output buffer, the service either blocks waiting for room, or returns an EWOULDBLOCK (depending on whether the socket is marked as blocking or nonblocking, and whether SO_SNDTIMEO timeout value was reached before space became available).
3. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer" on page 597

Characteristics and restrictions

There are no restrictions on the use of the send service.

Examples

For an example using this callable service, see "BPX1SND (send) example" on page 1192.

send_file (BPX1SF, BPX4SF) — Send a file on a socket

Function

The send_file callable service sends a file, with optional header and trailer data, as a byte stream on a socket connection. The service also provides options to close the socket connection after the data has been sent, and to prepare the socket for reuse after it has been closed.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SF):	31-bit
AMODE (BPX4SF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1SF,(Sfpl_length,
              Sfpl,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SF with the same parameters. All addresses in the Sfpl structure are doublewords.

Parameters

Sfpl_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Sfpl structure that is being passed in the Sfpl parameter. To determine the value of Sfpl_length, use the BPXYSFPL macro("BPXYSFPL — Map the send_file parameter list" on page 1038).

Sfpl
Supplied and returned parameter

Type: Structure

Length:
Specified by the Sfpl_length parameter

The name of the Sfpl structure that is to be used to control this I/O operation. See the usage notes for details on setting the fields of this structure.

The Sfpl is mapped by the BPXYSFPL macro ("BPXYSFPL — Map the send_file parameter list" on page 1038).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send_file service returns the following:

- 0, if the request is successful.
- -1, if the request is not successful.
- 1, if the request was interrupted by a signal, or if a nonblocking descriptor would have blocked while sending the data. The Sfpl structure is updated by the system to account for the data that was sent. You can continue the operation from the point at which it was interrupted by reissuing the send_file request with the same Sfpl structure.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send_file service stores the return code. The send_file service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The send_file service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	A descriptor is marked nonblocking, and no data could be sent without blocking.
EBADF	A descriptor that was not valid was supplied; the file was not open for reading; or the socket was not open for writing. Consult Reason_code to determine the exact reason the error occurred. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen, JRRFileWrOnly, JRWFileRdOnly.
ECONNRESET	The connection was reset by a peer.Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSockNotCon.
ECONNABORTED	A connection has been dropped.
EFAULT	An address that was passed could not be referenced in the key of the caller.
EIO	An I/O error occurred.
ENOBUFS	The service was unable to obtain a buffer. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JROutofSocketCells.
ENOMEM	The service was unable to obtain memory to complete the operation.
EINTR	A signal interrupted the send_file service before any data was written. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	Data that was not valid was sent to the request. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSocketCallParmError.
ENOTCONN	The socket was not connected. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSocketNotCon.
EPIPE	An attempt was made to send a message to a socket that is shut down or closed. This error also generates a SIGPIPE signal. Consult Reason_code to determine the exact reason the error occurred. The following reason code can accompany the return code: JRSocketClosed.
EWOULDBLOCK	A descriptor is marked nonblocking and no data could be sent, or the SO_SNDTIMEO timeout value was reached before space became available.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the send_file service stores the reason code. The send_file service returns Reason_code only if Return_value is -1.

Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. **Sfpl Structure (send_file parameter list)**

   The send_file operation is controlled by the values that are set into this structure. Refer to the BPXYSFPL macro for the exact field names ("BPXYSFPL — Map the send_file parameter list" on page 1038).

   Note:

   a. The entire Sfpl structure should be zeroed out before its first use, to ensure that undefined options, reserved space, and fields that might be used are initialized properly.

   b. All doubleword fields are treated as signed 63–bit arithmetic values. For operations that are known to be under 4 gigabytes in size, you can refer to the lower words of these fields (named in the BPXYSFPL macro) as unsigned 32–bit arithmetic values.

   Field Description

   Socket_desc
   The descriptor on which to send the data.

   File_desc
   The descriptor from which to read the data that is sent.

   File_offset
   A doubleword field that contains the byte offset in the file from which to start sending.

   File_bytes
   A doubleword field that contains the number of bytes to be sent, starting from the File_offset.

   If this field is -1, the entire file, from File_offset, is sent. The system updates the field with the number of file bytes that were sent (File_size-File_offset).

   If this field is 0, no file data is sent, and File_desc is ignored.

   If File_desc is not a regular file, it may be necessary to supply a specific value for File_bytes, unless a normal "end of file" indication is expected from File_desc during this operation, or you simply want the operation to run forever, transferring bytes as they arrive.

   File_size
   A doubleword field that is updated by the system after the operation with the file's size.

   Header_len
   The length of the header data.

   Header_ptr
   The address of the header data that is to be sent in front of the file data.

   Header_alet
   The ALET of the header data.

   Trailer_len
   The length of the trailer data.

Trailer_ptr
The address of the trailer data that is to be sent after the file data.

Trailer_alet
The ALET of the trailer data.

Bytes_sent
A doubleword field that is filled in by the system with the total number of bytes that are sent on this call. If the file must be sent with multiple calls because of signal interruptions, this field contains the value for the last call only; it is not a running total.

Options
A field that contains the following bit flags, which have the specified meaning when the bit is on:

- SF_CLOSE — Close the Socket_desc after the data has been sent. If the operation completes successfully and Socket_desc is closed, the system updates Socket_desc in the Sfpl with -1.
- SF_REUSE — Prepare the Socket_desc to be reused after the data has been successfully sent.

  This option is intended for sockets, and for the subsequent use of the descriptor on an accept_and_recv() call. To reuse the socket descriptor, the Socket_desc value, as updated by the system in the Sfpl after the call to send_file(), is specified as the Accepted_socket parameter on the accept_and_recv() call.

  Between the send_file() and the accept_and_recv() calls, a reused socket may only be used on accept_and_recv() or close(). The socket descriptor should be closed if it is not to be used again.

  If reuse is not supported, the system closes Socket_desc, and replaces its value in the Sfpl with -1. This ensures that the output value of Socket_desc is always appropriate as an input value for the Accepted_socket parameter of an accept_and_recv() call.

2. The send_file() function attempts to send the header data, followed by the file data from File_desc, followed by the trailer data, over socket_desc.
3. As data is sent, the system updates the Sfpl structure to account for the data that has been sent. This facilitates continuation after a signal interruption, but it also means that the Sfpl must be almost completely reset to start another new operation.
4. If File_offset>File_size, or File_bytes>(File_size-File_offset), the operation fails with an EINVAL error.
5. The SF_CLOSE and SF_REUSE flags are only effective when the operation completes successfully.
6. The send_file service is not strictly limited to sending a file on a socket. Any two stream-oriented descriptors may be used, although some of the parameters may have to be interpreted differently. When File_desc is a pipe, for example, the File_size and Offset parameters are meaningless.
7. The file cursor for the File_desc that is specified is updated with the results of the send_file operation. This does not affect other send_file() calls, but it does affect later read() and write() operations that use this File_desc.
8. **Application Usage**

   The send_file service is designed to work with the accept_and_recv service to provide an efficient file transfer capability for a connection-oriented server with short connection times and high connection rates.

These functions are designed for a server process/thread model that is different from the traditional one in which a parent thread accepts connections in a loop and spins off child processes or threads to issue the receive and do work. In this new server model, the parent is eliminated. Multiple worker processes or threads are initially created, and each worker process or thread independently executes the accept_and_recv() and send_file() functions in a loop.

The performance benefits of accept_and_recv() and send_file() over the separate operations that they combine include fewer buffer copies, recycled sockets, and optimal thread scheduling.

Socket descriptors can be recycled in the following way:

a. On the first call to accept_and_recv(), the application sets the Accepted_socket parameter to -1. This causes the system to assign a new descriptor to the accepted socket.
b. On the following call to send_file(), the application requests SF_REUSE. The socket session is closed, but the socket descriptor remains available for reuse on the next accept_and_recv().
c. All later calls to accept_and_recv() specify as their Accepted_socket the Socket_desc value that is left over from the previous call to send_file().

   In cases in which the socket does not support reuse, the system sets Socket_desc to -1 after the send_file(), so that the value is suitable for the next accept_and_recv() call.

Related services

- "accept_and_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data" on page 18
- "send (BPX1SND, BPX4SND) — Send data on a socket" on page 640
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1SF (send_file) example" on page 1187.

sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket

Function

The sendmsg callable service sends messages on a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX2SMS):	31-bit
AMODE (BPX4SMS):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX2SMS,(Socket_descriptor,
              Message_hdr,
              Flags,
              Iov_alet,
              Iov_buffer_alet,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SMS with the same parameters. All addresses in the Message_hdr structure are doublewords.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the sendmsg is to be done.

Message_hdr
Supplied parameter

Type: Structure

Length:
Length of BPXYMSGH

The name of a field that contains the message header, which describes how the message is to be sent. In 64-bit mode, Message_hdr contains doubleword pointer subfields, and points to an Iov_struct structure that contains doubleword pointer and length subfields. See "BPXYMSGH — Map the message header" on page 999 for more information about the format of this field.

Flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a field that contains information about how the data is to be sent. See "BPXYMSGF — Map the message flags" on page 997 for more information about the format of this field.

Iov_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for the IOV structure that is specified in Message_hdr.

Iov_buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for the buffers that are pointed to by the IOV structure that is specified in Message_hdr.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendmsg service returns one of the following:

- The number of bytes sent from the buffers, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendmsg service stores the return code. The sendmsg service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The sendmsg service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	For AF_UNIX sockets, search permission is denied for a component of the path prefix or write access to the named socket is denied.
EAFNOSUPPORT	The address family that was specified in the message header is not the same as the address family that owns the socket.
EBADF	A file descriptor that was not valid was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	Connection reset by peer. The following reason code can accompany the return code: JRSockNotCon.
EINTR	A signal interrupted the sendmsg service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	Data that was not valid was sent to the request.The following reason codes can accompany the return code: JRInvalidMsgh, JRSocketCallParmError, and JRSockNoName.

Return_code	Explanation
EIO	There has been a network or transport failure. The following reason codes can accompany the return code: JRInetRecycled, JRPrevSockError.
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPIPE	An attempt was made to send a message to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a **SIGPIPE** signal.
EWOULDBLOCK	• The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available. • The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option. The following reason codes can accompany the return code: JRWouldBlock and JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendmsg service stores the reason code. The sendmsg service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The socket can be either connected or unconnected.
2. If there is not enough room to write the data to an output buffer, the service either blocks waiting for an output buffer to become available, or returns an EWOULDBLOCK (depending on whether the socket is marked as blocking or nonblocking, and whether SO_SNDTIMEO timeout value was reached before space became available).
3. The BPX2SMS/BPX4SMS call supersedes the BPX1SMS call, which is still available for migration purposes only.
4. If the recvmsg security label is not equivalent to the sendmsg security label when access rights are passed on the sendmsg, the new descriptors are not created.
5. The number of buffers that are pointed to by the IOV structure in Message_hdr may not exceed IOV_MAX (defined in "BPXYIOV — Map the I/O vector structure" on page 986).

6. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

"recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers" on page 604

Characteristics and restrictions

There are no restrictions on the use of the sendmsg service.

Examples

For an example using this callable service, see "BPX2SMS (sendmsg) example" on page 1191.

sendto (BPX1STO, BPX4STO) — Send data on a socket

Function

The sendto callable service sends data on a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1STO):	31-bit task or SRB mode
AMODE (BPX4STO):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STO,(Socket_descriptor,
              Buffer_length,
              Buffer,
              Buffer_alet,
              Flags,
              Sockaddr_length,
              Sockaddr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STO with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the sendto is to be done.

Buffer_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Buffer.

Buffer
Supplied parameter

Type: Character

Length:
Length specified by Buffer_length

The name of a field from which the data is to be sent.

Buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Buffer. You should specify a Buffer_alet of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_alet, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a field that contains information about how the data is to be sent. See “BPXYMSGF — Map the message flags” on page 997 for more information about the format of this field.

Sockaddr_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the sockaddr could be on output.

Sockaddr
Supplied and returned parameter

Type: Structure

Length:
Length specified by Sockaddr_length

The name of a socket address structure to which the data is to be sent. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the format of this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendto service returns one of the following:

- The number of bytes that were sent on the socket, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendto service stores the return code. The sendto service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The sendto service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	For AF_UNIX sockets, search permission is denied for a component of the path prefix or write access to the named socket is denied.
EAFNOSUPPORT	The address family that was specified in the sockaddr is not the same address family as the socket.
EBADF	A file descriptor that was not valid was specified.The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.
EINTR	A signal interrupted the sendto service before any data was written. The following reason code can accompany the return code: JRSocketRdwrSignal.
EINVAL	An input parameter was incorrect. The following reason codes can accompany the return code: JRSocketCallParmError, JRSockNoName.
EIO	There has been a network or transport failure.The following reason codes can accompany the return code: JRInetRecycled, JrPrevSockError.

Return_code	Explanation
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPIPE	An attempt was made to send to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a **SIGPIPE** signal.
EPROTOTYPE	The address specifies a socket that is not the correct type for this request.
EWOULDBLOCK	• The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available. • The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option. The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sendto service stores the reason code. The sendto service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A datagram socket can be unconnected.
2. If the sending socket has no space to hold the message that is to be transmitted, the sendto service either blocks waiting for an output buffer to become available, or returns an EWOULDBLOCK (depending on whether the socket is marked as blocking or nonblocking, and whether SO_SNDTIMEO timeout value was reached before space became available).
3. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

- "recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer" on page 600

Characteristics and restrictions

There are no restrictions on the use of the sendto service.

Examples

See "BPX1STO (sendto) example" on page 1200 for an example using this callable service.

server_init (BPX1SIN, BPX4SIN) — Server initialization

Function

The server_init callable service allows a server address space to connect to Work Load Manager (WLM) for the purpose of queueing and servicing work requests.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1SIN):	31-bit task or SRB mode
AMODE (BPX4SIN):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SIN,(ManagerType,
              SubSystemType,
              SubSystemName,
              ApplEnv,
              ParallelEu,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SIN with the same parameters.

Parameters

ManagerType
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one or more of the following values that indicate the type of WLM manager the caller is requesting to become. The following are the supported values:

SRV_WORKMGR
WLM work management services are to be made available to the calling address space. This value can be combined with the SRV_QUEUEMGR and SRV_SERVERMGR values.

SRV_QUEUEMGR
WLM queue management services are to be made available to the calling address space. This value can be combined with the SRV_WORKMGR and SRV_SERVERMGR values.

SRV_SERVERMGR
WLM server management services that are associated with a queue manager are to be made available to the calling address space. This value can be combined with the SRV_QUEUEMGR and SRV_WORKMGR values.

These constants are defined in the BPXYCONS macro; see "BPXYCONS — Constants used by services" on page 952.

SubSystemType
Supplied parameter

Type: Character string

Length:
4 bytes

The name of a 4-byte field that contains the generic subsystem type (such as CICS®, IMS™, and WEB). When SRV_WORKMGR is specified for the ManagerType parameter, this is the primary category under which WLM classification rules are grouped. This parameter must be padded with blanks if the name is less than 4 bytes. When SRV_QUEUEMGR is specified for the ManagerType parameter, the combination of the SubSystemType and SubSystemName parameter values must be unique to a single MVS system.

SubSystemName
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte field that contains the subsystem name to be used for classifying work requests when SRV_WORKMGR is specified for the ManagerType parameter. This parameter must be padded with blanks if the name is less than 8 bytes. When SRV_SERVERMGR is specified for the ManagerType parameter, the subsystem name must match the subsystem name that is specified on the corresponding call to server_init for a work manager (ManagerType = SRV_WORKMGR).

ApplEnv
Supplied parameter

Type: Character string

Length:
32 bytes

The name of a 32-byte area that contains the name of the application environment under which work requests are served. The character string must be padded with blanks if the name is less than 32 characters. This parameter is only valid when SRV_SERVERMGR is specified for the ManagerType parameter. It is ignored for all other ManagerType values.

ParallelEu
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the maximum number of tasks within the address space that will be created to process concurrent work requests. This parameter is only valid when SRV_SERVERMGR is specified for the ManagerType parameter. It is ignored for all other ManagerType values.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_init service returns `0` if the request is successful, or -1 if it is not successful.

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_init service stores the return code. The server_init service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The server_init service can return one of the following values in the Return_code parameter:

Return code	Explanation
EFAULT	An argument of this service contained an address that was not accessible to the caller.
EINVAL	The ManagerType parameter contains a value that is not correct.
EMVSWLMERROR	A WLM service failed. Consult Reason_code to determine the WLM service that failed and the reason the error occurred. See *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for a list of WLM services (IWM*) error reason codes.
EPERM	The calling thread's address space is not permitted to the BPX.WLMSERVER profile in the FACILITY class. The caller's address space must be permitted to the BPX.WLMSERVER profile in the FACILITY class. If the BPX.WLMSERVER resource profile is not defined, the calling process is not defined as a superuser (UID=0).
EMVSSAF2ERR	An error occurred in the security product. Consult Reason_code to determine the exact reason the error occurred.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_init service stores the reason code. The server_init service returns Reason_code only if Return_value is -1.

Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. A successful call to server_init causes the calling address space to be connected to WLM for the WLM management services requested. Additionally, for a successful server manager connection call (SRV_SERVER_MGR ManagerType), the calling process is made a child of and placed in the session and process group of the corresponding work manager. The corresponding work manager is the process that called server_init for the ManagerType combination SRV_WORK_MGR+SRV_QUEUE_MGR with the same SubSystemType and SubSystemName values that were specified by the server manager process.

   This parent child relationship facilitates the use of signals between the server manager and the work manager to communicate with each other. The server manager, for example, after calling this service can issue the getppid service call to obtain the work server's process id, and then send signals to the work server when necessary.

   Because the server manager processes are child processes of the work manager/queue manager process, the work manager/queue manager process needs to ensure that terminated server manager processes get cleaned up. This requires the parent to either prevent the children processes from becoming zombie processes by using the sigaction service for the SIGCHLD signal, or clean up any terminated child processes by using the wait service.
3. This service should be used by a server that is designed to function in one of the following two ways:

   The server is divided into multiple address spaces, with a work and queue manager (MANAGER_TYPE=SRV_WORK_MGR+SRV_QUEUE_MGR) address space obtaining work requests from an external source and then queueing the work requests to one or more server manager (MANAGER_TYPE=SRV_SERVER_MGR) address spaces to process the work requests.

   A single server address space functions as the work and queue manager and as the server manager (MANAGER_TYPE=SRV_WORK_MGR+SRV_QUEUE_MGR+SRV_SERVER_MGR), with one or more threads obtaining work from an external source and then queueing the work requests to one or more server threads that process the work.

   The first method is the recommended approach to using this service, since it takes best advantage of WLM's system workload balancing capabilities by allowing WLM to create and manage the server address spaces against all other work in the system.
4. The server_init service is a privileged service that requires the caller to be authorized in one of the following ways:
   - Have read access to the BPX.WLMSERVER resource profile in the FACILITY class
   - Have a UID of 0 when the BPX.WLMSERVER profile is not defined

Related services

- "server_pwu (BPX1SPW, BPX4SPW) — Server process work unit" on page 660

Examples

For an example using this callable service, see "BPX1SIN (server_init) example" on page 1189.

server_pwu (BPX1SPW, BPX4SPW) — Server process work unit

Function

The server_pwu callable service provides a general purpose interface for managing and processing work using the Work Load Manager (WLM). It lets a program put work requests onto the WLM work queues, obtain work from the WLM work queues, transfer work to other WLM work servers, end units of work, delete WLM enclaves, and refresh WLM work servers.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SPW):	31-bit
AMODE (BPX4SPW):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPW,(FcnCode,
              TransClass,
              ApplEnv,
              ClassifyAreaLen,
              ClassifyAreaPtr,
              ApplDataLen,
              ApplDataPtr,
              FdStrucPtr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPW with the same parameters.

Parameters

FcnCode

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one or more of the following values indicating the function that is requested. The following are the supported values:

SRV_PUT_NEWWRK
A new work request is to be put onto the WLM work queue for the application environment that is identified by the ApplEnv parameter, as part of a newly created WLM enclave. This value cannot be combined with any other FcnCode value.

SRV_PUT_SUBWRK
A new work request is to be put onto the WLM work queue for the application environment that is identified by the ApplEnv parameter, as part of the existing WLM enclave that is associated with the calling thread. This value can be combined only with the SRV_END_WRK FcnCode value.

SRV_TRANSFER_WRK
The work request that is associated with the WLM enclave of the calling thread is to be transferred to the work queue of the target application environment that is identified by the ApplEnv parameter. As part of the transfer, the calling thread is disassociated from its WLM Enclave. This value cannot be combined with any other FcnCode value.

SRV_GET_WRK
A new work request is to be obtained from the WLM work queue for the calling application environment server. The SRV_GET_WRK FcnCode also results in the association of the calling thread with the WLM enlcave that was created when the obtained work request was put onto a WLM work queue. If the calling thread is already associated with a WLM enclave, an implicit SRV_END_WRK is performed. This value can only be combined with the SRV_END_WRK and SRV_DEL_ENC FcnCode values.

SRV_REFRESH_WRK
The servers that are associated with the application environments that are managed by the calling work and queue manager are to be refreshed. This causes all servers to complete existing work requests and then terminate. New servers are then started to process new work.

SRV_END_WRK
The calling thread is to be disassociated from its WLM enclave. This value can only be combined with the SRV_GET_WRK, SRV_PUT_SUBWRK and SRV_DEL_ENC FcnCode values.

SRV_DEL_ENC
The WLM enclave that is associated with the calling thread is to be deleted. This value can only be combined with the SRV_GET_WRK and SRV_END_WRK FcnCode values. This value should not be used to delete an enclave before ending the work units in the enclave, to prevent erroneous workload management results

SRV_DISCONNECT
The calling server's connection to WLM is to be severed. Once a server is disconnected from WLM, it can no longer use this service to process more requests for the application environment for which it had been connected to WLM by a call to the server_init function. If a SRV_DISCONNECT is performed by a work and queue manager, all related server managers implicitly lose their connections to WLM. This means that the related server managers also lose their ability to process more requests via this service.

SRV_DISCONNECT_COND
The calling server's connection to WLM is to be severed only if the caller has no more WLM enclaves that it is still managing. A work and queue manager is still managing an enclave if it has yet to be serviced by a server manager. Once a server is disconnected from WLM, it can no longer use this service to process more requests for the application environment for which it had been connected to WLM by a call to the server_init function. If a SRV_DISCONNECT is performed by a work and queue manager, all related server managers implicitly lose their connection to WLM as well. This means that the related server managers also lose their ability to process more requests via this service.

These constants are defined in the BPXYCONS macro ("BPXYCONS — Constants used by services" on page 952).

TransClass
Supplied parameter

Type: Character string

Length:
8 bytes

The name of an 8-byte area that contains the name of the transaction class that is to be associated with the work request. This parameter is only valid when the SRV_PUT_NEWWRK FcnCode parameter value is specified. It is ignored for the other FcnCode parameter values, and ignored if a classification area is supplied. This parameter must be padded with blanks if the name contains fewer than 8 bytes.

ApplEnv
Supplied parameter

Type: Character string

Length:
32 bytes

The name of a 32-byte area that contains the name of the application environment under which work requests are served. The character string must be padded with blanks, if the name contains fewer than 32 characters. This parameter is valid only when one of the SRV_PUT FcnCode parameter values is specified, or if the SRV_TRANSFER_WRK function code parameter value is specified and is ignored otherwise.

ClassifyAreaLen
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the classification information area, as specified by the ClassifyAreaPtr parameter. This parameter is valid only with the SRV_PUT_NEWWRK FcnCode parameter value, and is ignored for the other FcnCode parameter values.

ClassifyAreaPtr
Supplied parameter

Type: Address

Length:
Fullword

The name of a fullword that contains the address of the classification information for the work request in the form of the parameter list for the IWMCLSFY macro. This parameter is intended for use with the SRV_PUT_NEWWRK FcnCode parameter value only. The length of this area is supplied by the ClassifyAreaLen parameter. This parameter is ignored if the ClassifyAreaLen parameter contains a zero value. Some of the classification data that is pointed to by the IWMCLSFY parameter list is truncated if it exceeds the maximum supported length, as follows:

ACCTINFO
143 bytes maximum length

SUBSYSPM
255 bytes maximum length

SOURCELU
17 bytes maximum length

COLLECTION
18 bytes maximum length

CORRELATION
12 bytes maximum length

ApplDataLen
Supplied or returned parameter

Type: Integer

Length:
Fullword

When one of the SRV_PUT or SRV_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the length of the application data that is specified by the ApplDataPtr parameter.

When the SRV_GET_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server_pwu service is to return the length of the application data that is returned in the ApplDataPtr parameter.

This parameter is intended for use when one of the SRV_PUT, SRV_TRANSFER, or SRV_GET FcnCode parameter values is specified. The maximum length that is supported for the application data is 10 megabytes.

ApplDataPtr
Supplied or returned parameter

Type: Address

Length:
Fullword

When one of the SRV_PUT or SRV_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the address of the application data area that is to be associated with the work request. This application data allows the caller to uniquely identify the specific work that the caller is requesting.

When the SRV_GET_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server_pwu service is to return the address of the application data that is associated with the obtained work request. The returned data area is an identical copy of the data area that was supplied on the corresponding server_pwu call to put the work request on a WLM work queue.

This parameter is intended for use when one of the SRV_PUT, SRV_TRANSFER, or SRV_GET FcnCode parameter values is specified.

FdStrucPtr
Supplied or returned parameter

Type: Address

Length:
Fullword

When one of the SRV_PUT or SRV_TRANSFER FcnCode parameter values is specified, this is a supplied parameter that is the name of a fullword that contains the address of the file descriptor list structure, as mapped by the BPXYSFDL mapping macro. The file descriptors that are specified in the list are to be propagated to the process that calls the server_pwu service to obtain the work request that is created by the call to this service. If the SFDLCLOSE flag is turned on in the SFDLFLAGS field of the supplied structure, all file descriptors in the list are closed in the calling process. If a null address is specified, no file descriptors are propagated.

When the SRV_GET_WRK FcnCode value is specified, this is a returned parameter that is the name of a fullword in which the server_pwu service is to return the address of the file descriptor list structure that is associated with the obtained work request. The returned file descriptor list structure contains a count of entries and a list of file descriptors that represent the list of file descriptors that have been remapped in the calling process for the obtained work request. The remapped file descriptor values correspond to the file descriptor values that were supplied on the server_pwu call to put the work request on a WLM work queue. A file descriptor list is only returned for a SRV_GET_WRK call if the list that was supplied on the corresponding SRV_PUT_WRK, SRV_PUT_SUBWRK or SRV_TRANSFER_WRK call had a file descriptor count of greater than zero.

This parameter is intended for use when one of the SRV_PUT, SRV_TRANSFER, or SRV_GET_WRK FcnCode parameter values is specified. The maximum number of file descriptors that are supported in the file descriptor list is 64.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_pwu service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_pwu service stores the return code. The server_pwu service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The server_pwu service can return one of the following values in the Return_code parameter:

Return code	Explanation
EAGAIN	The requested service could not be performed at the current time. The following reason code can accompany this return code: JRENCLAVESEXIST
EINVAL	The FcnCode parameter contains a value that is not correct.
EMVSWLMERROR	A WLM service failed. Consult Reason_code to determine the WLM service that failed and the reason the error occurred. See *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for a list of WLM services (IWM*) error reason codes.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the server_pwu service stores the reason code. The server_pwu service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Usage of the various server_pwu functions requires that the calling address space has successfully issued a call to the server_init service.
2. For the SRV_PUT_NEWWRK and SRV_REFRESH_WRK functions to run successfully, the caller must have successfully issued a call to the server_init service for one of the following ManagerType parameter combinations:
   - SRV_WORK_MGR + SRV_QUEUE_MGR
   - SRV_WORK_MGR + SRV_QUEUE_MGR + SRV_SERVER_MGR
3. For the SRV_PUT_SUBWRK and SRV_TRANSFER_WRK functions to run successfully, the caller must have successfully issued a call to the server_init service for one of the following ManagerType parameter combinations:
   - SRV_WORK_MGR + SRV_QUEUE_MGR + SRV_SERVER_MGR
   - SRV_SERVER_MGR
4. For the SRV_GET_WRK, SRV_END_WRK and SRV_DEL_ENC functions to run successfully, the caller must have successfully issued a call to the server_init service for one of the following ManagerType parameter combinations:
   - SRV_WORK_MGR + SRV_QUEUE_MGR + SRV_SERVER_MGR
   - SRV_SERVER_MGR
5. A successful call to server_pwu for the SRV_PUT_NEWWRK FcnCode not only creates a work request that is placed onto a WLM work queue, but it also creates a new WLM enclave for that work to run in when the work request is obtained. The newly created WLM enclave is classified based on the classification information that is supplied in the input classify area, or based on the input transaction class. Unlike SRV_PUT_NEWWRK, the

SRV_PUT_SUBWRK and SRV_TRANSFER_WRK FcnCodes queue work requests that eventually are associated with the WLM enclave of the calling thread when the work request is obtained.

6. A successful call to server_pwu for the SRV_GET_WRK FcnCode not only results in the caller's obtaining a work request from a WLM work queue that is associated with the caller's application environment, but it also results in the associating of the calling thread with the WLM enclave that is associated with the obtained work request. When the calling thread goes through task termination, or when its process is terminated, the work request is ended and the associated WLM enclave is deleted if it is owned by the terminating task or process. The SRV_GET_WRK caller owns the enclave if the work was queued using the SRV_PUT_NEWWRK or SRV_TRANSFER_WRK functions. If the caller is a thread created using pthread_create (pthread), the thread task owns the enclave. If the caller is not a pthread, the process owns the enclave.

   When the FdStrucPtr parameter is used to propagate file descriptors, the caller must ensure that all of the file descriptors in the list are valid open file descriptors in the caller's process and are not being closed during the processing of this service. If this is not the case, this service cannot guarantee the proper propagation of the specified file descriptors.
7. The following demonstrates some sample usage scenarios for the FdStrucPtr parameter:
   - The queue manager process puts work on a work queue for a single–threaded server manager with no open file descriptors. The queued work has a supplied file descriptor structure with 3 file descriptors specified:

     ```
     Fds Supplied on SRV_PUT_NEWWRK      Fds returned on SRV_GET_WRK
     ------------------------------      ---------------------------
     0, 1, 2                             0, 1, 2
     ```

   - The queue manager process puts work on a work queue for a multithreaded server manager with open file descriptors. The queued work has a supplied file descriptor structure with 3 file descriptors specified:

     ```
     Fds Supplied on SRV_PUT_NEWWRK      Fds returned on SRV_GET_WRK
     ------------------------------      ---------------------------
     0, 1, 2                             12, 9, 14
     ```

Related services

"server_init (BPX1SIN, BPX4SIN) — Server initialization" on page 656

Examples

For an example using this callable service, see "BPX1SPW (server_pwu) example" on page 1195.

set_dub_default (BPX1SDD, BPX4SDD) — Set the dub default service

Function

The set_dub_default service allows the calling address space to change the current default dub setting for tasks within the address space.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1SDD):	31-bit
AMODE (BPX4SDD):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SDD,(Dub_setting,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SDD with the same parameters.

Parameters

Dub_setting

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the dub setting value.

Specifies the name of a fullword field that must contain one of the following dub setting values:

DUBPROCESS
Dub the subtasks of the caller as new processes when each issues its first z/OS UNIX service call.

DUBTHREAD
Dub the subtasks of the caller as threads in the caller's process when each issues its first z/OS UNIX service call.

DUBTASKACEE
Dub each subtask of the caller with its own z/OS UNIX security environment, if the subtask has a task-level security environment (ACEE) associated with it.

DUBNOSIGNALS
Dub the caller as a process to which signals will not be delivered.

DUBPROCESSDEFER
Dub each subtask of the caller as a new process when it issues its first z/OS UNIX service call. The address space is not dubbed when this call is issued. The first dub of the address space occurs when the next z/OS UNIX service call is issued (by this task or by another task in the address space).

DUBJOBPERM
Make the entire job permanent. All processes dubbed in this job are to be considered permanent processes. A permanent process is one that is

not taken down during a z/OS UNIX shutdown; all z/OS UNIX callable services that are called from these processes during a shutdown and restart window will return in failure.

DUBABENDCALLS
All z/OS UNIX callable services that are called from a process registered as permanent during a shutdown and restart window will end abnormally. This option is only relevant when it is accompanied by the DUBJOBPERM option.

DUBNOJSTUNDUB
Do not undub the entire jobstep when the last dubbed task (other than the jobstep task) undubs.

DUBUNIQUEACEE
Indicates to the kernal that this address space does not share ACEEs between tasks for the life of the process.

DUBFAILNOTREADY
Any syscall attempting to dub the caller as a process during a period when the z/OS UNIX kernel is shutdown will result in a failing return code, EMVSINITIAL, and reason code JrKernelReady. If the syscall does not have a return code parameter, it will receive an EC6 retryable abend with reason code JNotUp.

See "BPXYCONS — Constants used by services" on page 952 for the dub setting values.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_dub_default service returns:

- 0 if this is a dubbed thread and the call is successful
- 1 if this is a dubbed process and the call is successful
- -1 if the call is unsuccessful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_dub_default service stores the return code. The set_dub_default service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The set_dub_default service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One of the parameters contains an unsupported or incorrect value. The following reason code can accompany the return code: JRDubSetting.

Return_code	Explanation
EPERM	The calling process does not have the appropriate privilege to perform the requested operation. The following reason code can accompany the return code: JROK.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_dub_default service stores the reason code. The set_dub_default service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The default dub setting for a process that has not called this service is DUBTHREAD.
2. This service can be called to override a previous call to the service.
3. When a task that is not already dubbed issues its first z/OS UNIX service call, its TCB tree is searched to determine the default dub setting to use. The search starts at the caller's mother task and continues up the TCB tree until an ancestor task is found that is already dubbed. If the search finds a dubbed task, the default dub setting from that task is used. If a dubbed task is not found, the task is dubbed as a new process. A dubbed task is a task that has one or more of the following attributes:
   - It has issued a z/OS UNIX service call.
   - It was created as a result of a fork service call.
   - It was created as a result of an exec or execmvs service call.
   - It was created as a result of an attach_exec or attach_execmvs service call.
   - It was created as a result of a pthread_create service call.
4. DUBNOSIGNALS is mutually exclusive with DUBPROCESSDEFER. Specifying both options yields unpredictable results.
5. If DUBNOSIGNALS is used in a POSIX(ON) environment, the behavior of the process is undefined.
6. DUBPROCESSDEFER should only be used from the job step task. It is not honored when issued from other tasks in the address space.
7. After the job step task issues a set_dub_default call with option DUBPROCESSDEFER:
   - If it is the first task in the address space to issue a z/OS UNIX syscall and be dubbed, the job step task becomes a process in the address space.
   - If another task in the address space has already issued a z/OS UNIX syscall and been dubbed, the job step task becomes a thread in that process.
8. The DUBJOBPERM, DUBABENDCALLS, and DUBNOJSTUNDUB options should be used from the jobstep task prior to the call to any other z/OS UNIX callable service that could dub the address space, or they may not have their intended effect.
9. The DUBUNIQUEACEE option indicates whether an application is sharing ACEEs in a MultiProcess MultiUser (MpMu) address space. When specified, this option indicates that each process in the address space with a task level

ACEE has a unique ACEE. That is, the ACEEs are not shared between tasks. When specified, it allows z/OS UNIX processing to do a clean-up of certain control blocks during the redubbing of a task in the address space.

This option is honored only when it is specified on a BPX1SDD call prior to an address space dub. When issued from any task after the address space has already been dubbed, this option is ignored.

Characteristics and restrictions

1. When you set the DUBTASKACEE option, each task is dubbed as a separate process and uses the task-level ACEE that was set up by the user. In this environment, there are numerous restrictions on which other services can be used. This environment is supported primarily to allow a server to access HFS files and socket services. You cannot use z/OS UNIX security functions, such as setuid. Threads that are created with pthread_create do not inherit the identity of the parent. Fork and spawn do not work correctly.
2. Users of the DUBJOBPERM and DUBABENDCALLS options must meet the following requirements:
   - The calling address space must be a system started task address space.
   - The caller must be running authorized (APF-authorized, system key 0–7, or supervisor state).

   If these requirements are not met, the service will fail with return code EPERM.

Examples

For an example using this callable service, see "BPX1SDD (setdubdefault) example" on page 1185.

setegid (BPX1SEG, BPX4SEG) — Set the effective group ID

Function

The setegid callable service sets the effective group ID (GID) of a process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SEG):	31-bit
AMODE (BPX4SEG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SEG,(Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SEG with the same parameters.

Parameters

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the group ID that the calling process wishes to assume.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setegid service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setegid service stores the return code. The setegid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setegid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Group_ID that was specified is incorrect.
EPERM	The process does not have the appropriate privileges (see "Authorization" on page 8) to set the group ID.
EMVSSAF2ERR	The SAF call IRRSEG00 incurred an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setegid service stores the reason code. The setegid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETEGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	GID is not defined to RACF
8	8	User not authorized to change GID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

Usage notes

1. If Group_ID is equal to the real group ID or saved set group ID of the process, the effective group ID is set to Group_ID.
2. If Group_ID is not the same as the real group ID, and the calling process has the appropriate privileges (see "Authorization" on page 8), the effective group ID is set to Group_ID.
3. The setegid service does not change any supplementary group IDs of the calling process.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217
- "getgid (BPX1GID, BPX4GID) — Get the real group ID" on page 220
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

- If the setegid service is issued from multiple tasks within one address space, use synchronization to ensure that the setegid services are not performed concurrently. The execution of setegid service calls concurrently within one address space can yield unpredictable results.
- If the setegid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

Examples

For an example using this callable service, see "BPX1SEG (setegid) example" on page 1185.

seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID

Function

The seteuid callable service sets the effective user ID (UID) of a process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SEU):	31-bit
AMODE (BPX4SEU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SEU,(User_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SEU with the same parameters.

Parameters

User_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the user ID that the process is to assume.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the seteuid service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

Return_code	Explanation
EINVAL	The User_ID that was specified is incorrect.
EPERM	The process does not have the appropriate privileges to set the user ID. Refer to "Authorization" on page 8 for information on appropriate privileges.
EMVSSAF2ERR	The SAF call IRRSEU00 incurred an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the seteuid service stores the reason code. The seteuid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETEUID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	UID is not defined to RACF
8	8	User not authorized to change UID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

Usage notes

1. If User_ID is the same as the real or saved set user ID of the process, or if the user has the appropriate privilege, the seteuid service sets the effective user ID to be the same as User_ID. Refer to "Authorization" on page 8 for information on appropriate privileges.
2. For information about changing MVS identities, and other restrictions, see the usage notes for "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710.
3. To attach the security environment of the caller of the setuid service to the security environment of the target UID (which then creates a nested ACEE for the target), use the _BPXK_DAEMON_ATTACH environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the APPDATA text of 'RACF-DELEGATED' in the RACF profile protecting the resource.) For more information about nested ACEEs and delegated resources, see *z/OS Security Server RACF Security Administrator's Guide*.

Related services

- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

See the list of restrictions in "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710.

Examples

For an example using this callable service, see "BPX1SEU (seteuid) example" on page 1186.

setgid (BPX1SGI, BPX4SGI) — Set the group ID

Function

The setgid callable service sets the real, effective, and saved set group IDs (GIDs) for the calling process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SGI):	31-bit
AMODE (BPX4SGI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SGI,(Group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SGI with the same parameters.

Parameters

Group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the group ID that the calling process is to assume.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgid service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgid service stores the return code. The setgid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setgid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Group_ID that was specified is incorrect.
EPERM	The process does not have the appropriate privileges to set the group ID. Refer to "Authorization" on page 8 for information about appropriate privileges.
EMVSSAF2ERR	The SAF call IRRSSG00 incurred an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgid service stores the reason code. The setgid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	GID is not defined to RACF
8	8	User not authorized to change GID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

Usage notes

1. If Group_ID is equal to the real group ID or saved set group ID of the process, the effective group ID is set to Group_ID.
2. If Group_ID is not the same as the real group ID, and the calling process has the appropriate privileges, the real, saved set, and effective group IDs are set to Group_ID. Refer to "Authorization" on page 8 for information about appropriate privileges.
3. The setgid service does not change any supplementary group IDs of the calling process.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217
- "getgid (BPX1GID, BPX4GID) — Get the real group ID" on page 220
- "setegid (BPX1SEG, BPX4SEG) — Set the effective group ID" on page 670
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710

Characteristics and restrictions

- The calling process must be privileged in order to change the real group ID—that is, to specify a group ID that is different from the process's real group ID. Refer to "Authorization" on page 8 for information about appropriate privileges.

- If the setgid service is issued from multiple tasks within one address space, use synchronization to ensure that the setgid services are not performed concurrently. The execution of setgid services concurrently within one address space can yield unpredictable results.
- If the setgid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

Examples

For an example using this callable service, see "BPX1SGI (setgid) example" on page 1187.

setgrent (BPX1SGE, BPX4SGE) — Reset the group database

Function

The setgrent callable service resets the group database for subsequent searching by the getgrent service. The next getgrent service that is used after setgrent starts searching from the beginning of the group database.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SGE):	31-bit
AMODE (BPX4SGE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SGE,(Return_value)
```

AMODE 64 callers use BPX4SGE.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgrent service returns `0`.

Usage notes

The setgrent service is intended to be used to interrupt a sequential search of the group database from the calling task. The getgrent service performs the sequential

search. When the setgrent service is called, it resets the search point for the current task in the group database to the beginning. The next getgrent service that is called from this task after this point starts searching the group database from the beginning.

Related services

- "getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database" on page 221

Characteristics and restrictions

There are no restrictions on the use of the setgrent service.

Examples

For an example using this callable service, see "BPX1SGE (setgrent) example" on page 1187.

setgroups (BPX1SGR, BPX4SGR) — Set the supplementary group IDs list

Function

The setgroups callable service replaces the existing supplementary group IDs (GIDs) list for the calling process with the list that is specified by the caller.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SGR):	31-bit
AMODE (BPX4SGR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SGR,(SGid_list_count,
              SGid_list,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SGR with the same parameters. SGid_list is a 64-bit pointer field.

Parameters

SGid_list_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies the number of fullword entries in list that is pointed to by the SGid_list parameter. The value must be in the range of zero to NGroups_Max, inclusive.

Specifying `0` causes all existing supplementary group IDs for the calling process to be deleted. After the setgroups service completes, the calling process does not have any supplementary group IDs.

SGid_list
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains a pointer to an array of group IDs (GIDs). The setgroups service uses this list to establish the list of supplementary group IDs. The number of entries in the list is defined by the SGid_list_count parameter.

If the SGid_list_count specified is `0`, the SGid_list is ignored and does not need to contain a valid address.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgroups service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgroups service stores the return code. The setgroups service returns Return_code only if Return_value is `-1`. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The setgroups service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The SGid_list and SGid_list_count specify an array that is partially or completely outside the addressable storage range.
EINVAL	The SGid_list_count parameter was less than `0` or greater than NGroups_Max.
EMVSSAF2ERR	System authorization facility (SAF) had an error.
EPERM	The caller is not authorized; only authorized users are allowed to alter the supplementary group IDs list.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setgroups service stores the reason code. The setgroups service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

To determine the value of NGroups_Max, see "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819.

Related services

- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674
- "getgroups (BPX1GGR, BPX4GGR) — Get a list of supplementary group IDs" on page 229
- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819

Characteristics and restrictions

- To set the supplementary group IDs, the requester must be a superuser. If a non-superuser caller requests the setgroups service, the service returns an EPERM Return_code.
- To successfully complete the setgroups service, the caller's process must be the only process in the address space. If multiple processes are present (through attach_exec or attach_execMVS), the function does not complete successfully.

Examples

For an example using this callable service, see "BPX1SGR (setgroups) example" on page 1188.

setitimer (BPX1STR, BPX4STR) — Set the value of the iterval timer

Function

The setitimer callable service sets the timer value and optionally returns a pointer to a structure that contains the previous timer value. This function also generates a signal that is to be delivered when the interval timer expires.

Requirements

Operation	Environment
Authorization:	Problem Program or Supervisor State, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STR):	31-bit
AMODE (BPX4STR):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts

Operation	Environment
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STR (Interval_Type,
              Interval_Value_Adr,
              Old_Interval_Value_Adr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STR with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Interval_Type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a numeric value that identifies the interval timer (these values apply to both Interval_Value_Adr and Old_Interval_Value_Adr). This parameter can have the following values:

ITIMER_REAL
Real time (the default if VIRTUAL and PROF are not specified)

ITIMER_VIRTUAL
Virtual time (CPU time minus system time)

ITIMER_PROF
CPU time

ITIMER_MICRO
The initial and reload times are in microseconds (the default if NANO is not specified)

ITIMER_NANO
The initial and reload times are in nanoseconds

The ITIMER_ constants are defined in the BPXYITIM macro.

Interval_Value_Adr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

A fullword (doubleword) field containing the address of a structure that is defined by the BPXYITIM macro. This structure contains the initial interval and reload values in seconds and either microseconds or nanoseconds.

Old_Interval_Value_Adr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

A fullword (doubleword) field containing the address of a structure that is defined by the BPXYITIM macro. This structure contains the time remaining and reload values in seconds and either microseconds or nanoseconds. This address may be zero if the current values are of no interest to the user.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setitimer service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setitimer service stores the return code. The setitimer service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The setitimer service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One of the following occurred: • The value that was specified for *Interval_Type* is incorrect (JrStrIntervalTypeInvalid). • The value that was specified in the structure that is pointed to by *Interval_Value_Adr* is incorrect (JrNanoSecondsTooBig, JrMSecondsTooBig, JRNegativeValueInvalid).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setitimer service stores the reason code. The setitimer service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value.

Usage notes

1. The time between signals is maintained as a priority over the number of signals in an extended period of time. A delay in processing could result in a late signal; the signal generated by the reload value maintains the requested interval. As with other signals, multiples are lost. For example, for a 1 second timer, delays might result in 3597 signals per hour, with no two timers closer

than 1 second. If the application requires exactly 3600 signals per hour, a reload value of zero should be used, and a new setitimer should be calculated and issued by the signal handler.

2. The duration between signals is always greater than, or equal to, the reload value that is specified.
3. Intervals vary, depending on when MVS gives the task its time slices.
4. The setting of the first two words of the Interval_Value disables the timer, regardless of the reload value (third and fourth words).
5. Any setitimer() cancels the previous timer of the same type (that is, REAL, VIRTUAL, or PROF).
6. Real interval timers and alarms are treated as mutually exclusive. If an ITIMER_REAL interval timer is issued while an alarm is set, the ITIMER_REAL interval timer overlays the alarm, and vice versa.
7. The setitimer environment is propagated on the exec() and not propagated on fork().
8. Below are the interval timers and the corresponding signals that are to be generated when the timer expires:
   - ITIMER_REAL, which decrements in real time. A SIGALRM signal is delivered when this timer expires.
   - ITIMER_VIRTUAL, which decrements in task virtual time. It runs only when the task is executing outside the kernel. A SIGVTALRM signal is delivered when it expires. Task virtual is a best estimate, and loses significance when it is run in a multiprocess environment.
   - ITIMER_PROF, which decrements in task time. It runs when the task is running on behalf of the process. A SIGVPROF signal is delivered when it expires.
9. The setitimer(), alarm(), and sleep() services use the MVS STIMERM macro. If the task invokes the STIMERM macro and a combination of these services, the limit of concurrent STIMERM SET requests for a task can be exceeded, which results in an abnormal end.
10. ITIMER_REAL interval timers are supported in both multiprocess and multithreaded environments.
11. You can issue Setitimer() for ITIMER_PROF or ITIMER_VIRTUAL in a multithreaded or multiprocess environment. However, for ITIMER_VIRTUAL in a multithread environment, the results may be unpredictable.
12. If two interval timers of the same type expire before a signal is delivered, only one signal is generated.
13. The reload time is set before the signal interface routine is given control.

MVS-related information

- “getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer” on page 242
- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29

Characteristics and restrictions

None (other than those indicated in the usage notes).

Examples

For an example using this callable service, see “BPX1STR (setitimer) example” on page 1200.

setpeer (BPX1SPR, BPX4SPR) — Preset the peer address associated with a socket

Function

The setpeer callable service presets the peer address that is associated with a socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1SPR):	31-bit task or SRB mode
AMODE (BPX4SPR):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPR,(Socket_descriptor,
              Sockaddr_length,
              Sockaddr,
              Option,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPR with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the setpeer is to be done.

Sockaddr_length
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of Sockaddr.

Sockaddr
Supplied and returned parameter

Type: Structure

Length:
Length specified by Sockaddr_length

The name of a socket address structure that contains the peer address. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the format of this field.

Option
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that indicates the conditions of the setpeer request. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about this field.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpeer service returns one of the following:

- `0` if the request is successful.
- `-1` if the request is not successful.

Return_code
Returned parameter

Type: Integer

Character set:
N/A

Length:
Fullword

The name of a fullword in which the setpeer service stores the return code. The setpeer service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setpeer service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The socket descriptor is incorrect. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpeer service stores the reason code. The setpeer service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The setpeer service is not supported by AF_UNIX, AF_INET, or AF_INET6.
2. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Characteristics and restrictions

There are no restrictions on the use of the setpeer service.

Examples

For an example using this callable service, see "BPX1SPR (setpeer) example" on page 1195.

setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control

Function

The setpgid callable service places a process in a process group. You identify the group by specifying a process group ID. You can assign a process to a different group, or you can start a new group with that process as its leader.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SPG):	31-bit
AMODE (BPX4SPG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPG,(Process_ID,
              Process_group_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPG with the same parameters.

Parameters

Process_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ID of the process that is to be placed in the process group. If the ID is specified as 0, the system uses the process ID of the calling process.

Process_group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ID of the process group where Process_ID is assigned. If the ID is specified as 0, the system uses the process group ID that is indicated by the Process_ID parameter.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpgid service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpgid service stores the return code. The setpgid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setpgid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The value of Process_ID matches the process ID of a child of the calling process, but the child has successfully invoked one of the exec functions. Access to the target process was denied. The following reason code can accompany the return code: JRExecAfterFork.
EINVAL	The Process_group_ID parameter is less than zero or has some other unsupported value. The following reason codes can accompany the return code: JRNoSuchPid and JRPgidDifferentSession.
EPERM	The calling process cannot change the process group ID of the specified process. The following reason codes can accompany the return code: JRPidEQSessLeader, JRPidDifferentSession, and JRPgidDifferentSession.

Return_code	Explanation
ESRCH	The specified Process_ID is not that of the calling process or any of its children. The following reason codes can accompany the return code: JRNotDescendant and JRNoSuchPid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpgid service stores the reason code. The setpgid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The process group ID that is to be assigned to the group must be within the calling process's session.
2. The subject process (the process identified by the Process_ID parameter) must be a child of the process that issues the service and must be in the same session, but it cannot be the session leader. It can be the caller.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID" on page 252
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702
- "tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID" on page 849

Characteristics and restrictions

See the conditions described under Return_code.

Examples

For an example using this callable service, see "BPX1SPG (setpgid) example" on page 1194.

setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process

Function

The setpriority callable service sets the scheduling priority of a process, process group, or user.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1SPY):	31-bit
AMODE (BPX4SPY):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPY,(Which,
              Who,
              Priority,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPY with the same parameters.

Parameters

Which
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates how the Who parameter is to be interpreted. This parameter can have one of the following values:

PRIO_PROCESS
The Who parameter is to be interpreted as a process ID.

PRIO_PGRP
The Who parameter is to be interpreted as a process group ID.

PRIO_USER
The Who parameter is to be interpreted as a user ID.

The PRIO_ constants are defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Who
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the exact process ID, process group ID, or user ID whose priority is to be set. The Which parameter indicates how this parameter is to be interpreted. A value of zero for this parameter specifies the current process, process group, or user ID.

Priority
Supplied parameter

Type: Signed Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the priority value to which the specific process or group of processes is to be set. This value can be an integer in the range of -20 to 19.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpriority service returns a value of zero if successful and -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpriority service stores the return code. The setpriority service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The setpriority service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The priority is being changed to a lower value, and the current process does not have the appropriate privilege (see "Authorization" on page 8) to do so.
EINVAL	The value of the Which parameter was not recognized; or the value of the Who parameter is not a valid process ID, process group ID or user ID.
EMVSSAF2ERR	A security product internal error occurred. Consult the Reason_code parameter for the exact reason for the error.
ENOSYS	The system does not support this function. Your installation chose not to enable this function.
EPERM	A process was located, but neither the real nor the effective user ID of the calling process matches the effective user ID of the process whose priority is being changed.
ESRCH	No process could be located using the Which and Who parameter values that were specified.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpriority service stores the reason code. The setpriority service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the supplied Who and Which values specify more than one process, each of the specified processes has its priority value set to the supplied value. If at least one of the specified processes has its priority value successfully changed, the setpriority service returns successfully.
2. The priority value of a process is an integer that can be in the range of -20 to 19. If the supplied priority value is outside this range, the process's priority is set to the corresponding limit value. The default priority value for all processes is 0.
3. An increase in a process's priority value results in a lower CPU priority for the process. A decrease in a process's priority value results in a higher CPU priority for the process.
4. If the supplied priority value would result in a lower priority value for the specified process or processes, the caller must have appropriate privileges. Refer to "Authorization" on page 8 for information about appropriate privileges.
5. The setting of a process's priority value has a corresponding effect on its nice value because they both represent the process's relative CPU priority. For example, using the setpriority service to set the priority value of a process to its maximum value (19) has the effect of increasing its nice value to its maximum value (2*NICE_ZERO)-1, and this is reflected on the nice, getpriority, and setpriority services. The NICE_ZERO constant is defined in BPXYCONS.
6. If the ENOSYS return code is received, your installation does not support this service. Contact your system administrator if you require activation of this service.
7. If the supplied Who and Which values specify a process in a multiple process address space, each of the processes in the address space have their priority values set to the supplied value.
8. To do the initial system setup for using this service, see Enabling nice(), setpriority(), and chpriority() support in *z/OS UNIX System Services Planning*.

Related services

- "nice (BPX1NIC, BPX4NIC) — Change the nice value of a process" on page 432
- "getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process" on page 255

Characteristics and restrictions

There are no restrictions on the use of the setpriority service.

Examples

For an example using this callable service, see "BPX1SPY (setpriority) example" on page 1196.

setpwent (BPX1SPE, BPX4SPE) — Reset the user database

Function

The setpwent callable service resets the user database for subsequent searching by the getpwent service. The next getpwent service that is used after setpwent starts searching from the beginning of the user database.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SPE):	31-bit
AMODE (BPX4SPE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPE,(Return_value)
```

AMODE 64 callers use BPX4SPE.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setpwent service returns `0`.

Usage notes

The setpwent service is intended to be used to interrupt a sequential search of the user database from the calling task. The getpwent service performs the sequential search. When the setpwent service is called, it resets the search point for the current task in the user database to the beginning. The next getpwent service that is called from this task after this point starts searching the user database from the beginning.

Related services

- "getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database" on page 258

Characteristics and restrictions

There are no restrictions on the use of the setpwent service.

Examples

For an example using this callable service, see "BPX1SPE (setpwent) example" on page 1193.

setregid (BPX1SRG, BPX4SRG) — Set the real and/or effective GIDs

Function

The setregid callable service sets the real or effective GIDs for the calling process to the values that are specified by the input real and effective GID values. If a specified value is equal to -1, the corresponding real or effective GID of the calling process is left unchanged.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SRG):	31-bit
AMODE (BPX4SRG):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SRG,(RGID,
              EGID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SRG with the same parameters.

Parameters

RGID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the real GID to be set for the calling process. If RGID is -1, the real GID for the calling process is left unchanged.

EGID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the effective GID to be set for the calling process. If EGID is -1, the effective GID for the calling process is left unchanged.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setregid service returns -1 if it is not successful. If it is successful, the setregid service returns 0.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setregid service stores the return code. The setregid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setregid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of RGID or EGID is not valid.
EPERM	The process does not have appropriate privileges to set the real GID or the effective GID. Refer to "Authorization" on page 8 for information on appropriate privileges.
EMVSSAF2ERR	The SAF call IRRSSG00 incurred an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setregid service stores the reason code. The setregid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETGID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing

Usage notes

1. A process with appropriate privileges (see "Authorization" on page 8) can set the real and effective GID to any valid GID value. An unprivileged process can

only set the effective GID if the EGID argument is equal to the real, effective, or saved GID of the process. An unprivileged process can only set the real GID if the RGID argument is equal to the real, effective, or saved GID of the process.

2. The setregid does not change any supplementary GIDs of the calling process.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710
- "setreuid (BPX1SRU, BPX4SRU) —Set the real and/or effective UIDs"

Characteristics and restrictions

- If the setregid service is issued from multiple tasks within one address space, use synchronization to ensure that the setregid services are not performed concurrently. The execution of setregid requests concurrently within one address space can yield unpredictable results.
- If the setregid service is issued from an address space with multiple processes, the result of the service call affects all processes in the address space.

Examples

For an example using this callable service, see "BPX1SRG (setregid) example" on page 1196.

setreuid (BPX1SRU, BPX4SRU) —Set the real and/or effective UIDs

Function

The setreuid callable service sets the real and/or effective UIDs for the calling process to the values that are specified by the input real and effective UID values. If a specified value is equal to -1, the corresponding real or effective UID of the calling process is left unchanged.

Requirements

Operation	**Environment**
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SRU):	31-bit
AMODE (BPX4SRU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SRU,(RUID,
              EUID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SRU with the same parameters.

Parameters

RUID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the real UID to be set in the calling process. If RUID is -1, the real UID for the calling process is left unchanged.

EUID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the effective UID to be set in the calling process. If EUID is -1, the effective UID for the calling process is left unchanged.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setreuid service returns -1 if it is not successful. If it is successful, the setreuid service returns 0.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setreuid service stores the return code. The setreuid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setreuid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of RUID or EUID is not valid.

Return_code	Explanation
EPERM	The process does not have appropriate privileges to set the real UID or the effective UID. Refer to "Authorization" on page 8 for information on appropriate privileges.
EMVSSAF2ERR	The SAF call IRRSSU00 incurred an error.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setreuid service stores the reason code. The setreuid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETUID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	User is not privileged
8	12	Internal error during RACF processing

Usage notes

1. A process with appropriate privileges (see "Authorization" on page 8) can set the real and effective UID to any valid UID value. An unprivileged process can set the effective UID only if the EUID argument is equal to the real, effective, or saved UID of the process. An unprivileged process can set the real UID only if the RUID argument is equal to the real, effective, or saved UID of the process.
2. The setreuid service is allowed in a TSO address space so long as the caller does not attempt to change the MVS identity. MVS identity changes are triggered by changing the effective UID. The real UID can always be changed if the invoker has appropriate privileges.
3. For information about changing MVS identities, and other restrictions, see the UsageNotes for "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710.
4. To attach the security environment of the caller of the setuid service to the security environment of the target UID (which then creates a nested ACEE for the target), use the _BPXK_DAEMON_ATTACH environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the APPDATA text of 'RACF-DELEGATED' in the RACF profile protecting the resource.) For more information about nested ACEEs and delegated resources, see *z/OS Security Server RACF Security Administrator's Guide*.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710
- "setregid (BPX1SRG, BPX4SRG) — Set the real and/or effective GIDs" on page 693

Characteristics and restrictions

See the list of restrictions in "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710.

See also "Usage notes" on page 697.

Examples

For an example using this callable service, see "BPX1SRU (setreuid) example" on page 1197.

setrlimit (BPX1SRL, BPX4SRL) — Set resource limits

Function

The setrlimit callable service sets resource limits for the calling process. A resource limit is a pair of values; one specifies the current (soft) limit and the other the maximum (hard) limit.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SRL):	31-bit
AMODE (BPX4SRL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SRL,(Resource,
              Rlimit,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SRL with the same parameters.

Parameters

Resource

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the resource for which to set the hard and soft limits. This parameter can specify one of the resources in the following table:

Table 20. Resources that can be limited by setrlimit

Resource	Description	Action when soft limit is exceeded
RLIMIT_MEMLIMIT	Maximum amount of usable storage above the bar (in 1-MB segments) that can be allocated at one time.	Any attempt to allocate or extend the usable amount of virtual storage above the 2-MB addressing range fails.
RLIMIT_CORE	Maximum core file size (in bytes) created by a process. A value of 0 prevents core file creation.	Core file writing terminates at this size.
RLIMIT_CPU	Maximum amount of CPU time (in seconds) used by a process.	SIGXCPU is sent to the process, and the process is granted a small extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a SIGKILL.
RLIMIT_FSIZE	Maximum file size (in bytes) created by a process. A value of 0 prevents the creation of new files and the expansion of existing files.	SIGXFSZ is sent to the process. If the process is blocking, catching, or ignoring SIGXFSZ, continued attempts to increase the size of a file beyond the limit fail with a return code of EFBIG.
RLIMIT_NOFILE	Maximum number of open file descriptors for a process. This number is one greater than the maximum value that may be assigned to a newly-created descriptor.	Functions that create new file descriptors after the limit is reached fail with a return code of EMFILE.
RLIMIT_AS	Maximum address space size (in bytes) for a process.	The mmap and shmat callable services fail with a return code of ENOMEM. User getmain and storage obtain requests fail (for example, runtime library stack and heap expansion fails).

The RLIMIT_ constants are defined in the BPXYCONS macro. See “BPXYCONS — Constants used by services” on page 952.

Rlimit

Supplied parameter

Type: Structure

Length:
The length of the rlimit structure

The name of an Rlimit structure that contains the values for the hard (maximum) and soft (current) limits for the resource that is identified by the resource parameter. Macro BPXYRLIM defines the Rlimit structure. (See "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033.) Each limit value contains two fullwords. For all resources except RLIMIT_FSIZE, the upper fullword for each limit value is ignored.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setrlimit service returns a value of zero if it is successful, and -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setrlimit service stores the return code. The setrlimit service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The setrlimit service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The operation is not permitted for one of the following reasons: • The resource that was specified is not valid. • The soft limit that is to be set exceeds the hard limit to set. • The soft limit that is to be set is below the current usage. • The hard limit that is to be set exceeds a system–defined limit. • One of the file size limits that was specified is a negative value. The following reason codes can accompany the return code: JrInvalidResource, JrSoftExceedsHard, JrSoftBelowUsage, JrFdOpenAboveLimit, JrOpenFileLimitMax, or JrNegFileSizeLimit.
EMVSSAF2ERR	A Security product internal error has occurred. Consult the Reason_code parameter for the exact reason for the error.
EPERM	An attempt was made to raise a hard (maximum) limit, but the calling process did not have superuser authority. The following reason code can accompany the return code: JrRaiseHardLimit.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setrlimit service stores the reason code. The setrlimit service returns Reason_code only if Return_value is -1.

Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The soft limit can be modified to any value that is less than or equal to the hard limit. For the RLIMIT_CPU, RLIMIT_NOFILE, and RLIMIT_AS resources, if setrlimit is called with a soft limit that is lower than the current usage, setrlimit fails with an EINVAL return code.

   An exception to this rule occurs when the process is running in an address space that contains multiple processes. When you change the RLIMIT_CPU, you can set a new soft limit that is greater than the time limit of the current process, yet greater than the time consumed by the address space. This allows the setrlimit call to succeed, and a SIGXCPU signal is generated. The alternative is not to run multiple processes in the same address space.
2. The hard limit may be lowered to any value that is greater than or equal to the soft limit.
3. The hard limit can only be raised by a process that has superuser authority.
4. Both the soft limit and the hard limit can be changed with a single call to the setrlimit service.
5. If the setrlimit service is called with a soft limit that is greater than the hard limit, setrlimit returns an EINVAL return code.
6. The resource limit values are propagated across the exec, fork, and spawn services. An exception exists for the exec and spawn services. If a daemon process invokes the exec service and it has previously invoked setuid, or invokes the spawn service specifying that a user ID change is to occur, then the limit values are set based on the values found in the OMVS segment of the target user ID. If the target user's OMVS segment does not specify limit values, then the limit values are set based on the values found in the BPXPRM*xx* parmlib member.
7. For processes in a multiprocess address space, the RLIMIT_CPU and RLIMIT_AS limits are shared with all the processes within the address space. For RLIMIT_CPU, when the soft limit is exceeded, action is taken on the first process within the address space. If the action is termination, all the processes within the address space are terminated.
8. In addition to the RLIMIT_CORE limit values, CORE dump defaults are set by SYSMDUMP defaults. See *z/OS MVS Initialization and Tuning Reference* for information on setting up SYSMDUMP defaults via the IEADMR00 parmlib member.
9. Core dumps are taken in 4160-byte increments. Therefore, RLIMIT_CORE values affect the size of core dumps in 4160-byte increments. For example, if the RLIMIT_CORE soft limit value is 0, no core dumps are taken. If the RLIMIT_CORE soft limit value is 8000, the maximum size of a core dump is 8000 * 4160 bytes.
10. Limits may have an infinite value of RLIM_INFINITY. MEMLIMIT cannot exceed 16383G, which is defined RLIM_MEGINFINITY.
11. The hard limit for RLIMIT_NOFILE cannot exceed the system-defined limit of 524287. A value of RLIM_INFINITY for RLIMIT_NOFILE indicates that the current system maximum value should be set.
12. The soft limit for RLIMIT_NOFILE must be set higher than the value of the highest open file descriptor. Attempting to lower the soft limit to a value that is less than or equal to the highest open file descriptor results in an EINVAL return code.

13. Setting a limit of 0 for RLIMIT_FSIZE prevents the creation of new files and the expansion of existing files.
14. When RLIM_INFINITY (X'7FFFFFFF') is passed on a setrlimit request, no limit is enforced by setrlimit. As a result, the maximum allowable limit is set, regardless of the resource. The new service RLIM_MEMLIMIT treats RLIM_INFINITY as a request for 21474836471 1- megabyte pages.
15. When the MEMLIMIT is set by z/OS UNIX, the highest value that is supported is 16383 petabytes, or X'FFFC00000000000000'.
16. Processes can use this service to control CPU resource consumption. For more information, see *z/OS MVS Initialization and Tuning Reference*.

Related services

- "getrlimit (BPX1GRL, BPX4GRL) — Get resource limits" on page 266
- "getrusage (BPX1GRU, BPX4GRU) — Get resource usage" on page 268

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1SRL (setrlimit) example" on page 1197.

setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID

Function

The setsid callable service creates a new session, with the calling process as its session leader. The caller becomes the group leader of a new process group.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SSI):	31-bit
AMODE (BPX4SSI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SSI,(Session_ID,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SSI with the same parameters.

Parameters

Session_ID
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword to which, if successful, the setsid service returns the session or process group ID of the new group. The new session or group process ID is the same as the process ID of the caller.

If not successful in creating a new session, the setsid service returns -1 as the Session_ID value.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setsid service stores the return code. The setsid service returns Return_code only if Session_ID is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setsid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EPERM	The caller is already a process group leader, or the caller's process ID matches the process group ID of some other process. The following reason code can accompany the return code: JRCallerIsPgLeader.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setsid service stores the reason code. The setsid service returns Reason_code only if Session_ID is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The calling process does not have a controlling terminal.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304

- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746

Characteristics and restrictions

The calling process must not already be a process group leader.

Examples

For an example using this callable service, see "BPX1SSI (setsid) example" on page 1198.

setsockopt or getsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket

See "getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket" on page 275.

set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads

Function

The set_thread_limits callable service changes the calling process's limits for pthread_created threads. These limits are the maximum number of MVS tasks used for pthread_created threads, and the maximum number of pthread_created threads. The thread limit includes running, queued, and undetached exited threads.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STL):	31-bit
AMODE (BPX4STL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STL,(Action,
              MaxThreadTasks,
              MaxThreads,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STL with the same parameters.

Parameters

Action
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a numeric value that identifies the process's pthread_created thread limits that are to be set. The following constants, which are defined in BPXYCONS, define the actions that are to be taken (see "BPXYCONS — Constants used by services" on page 952):

STL_MAX_TASKS
Replace the MaxThreadTasks limit for the caller's process with the value that is specified in MaxThreadTasks only.

STL_MAX_THREADS
Replace the MaxThreads limit for pthread_created threads in the caller's process with the fullword value that is specified in MaxThreads only.

STL_SET_BOTH
Replace both the MaxThreadTasks and MaxThreads limits for the caller's process with the fullword values that are specified in MaxThreadTasks and MaxThreads, respectively.

MaxThreadTasks
Supplied parameter

Type: Integer

Length:
Fullword

When the Action that is specified is STL_MAX_TASKS or STL_SET_BOTH, this is the name of a fullword that contains the new MaxThreadTasks value for the caller's process.

MaxThreads
Supplied parameter

Type: Integer

Length:
Fullword

When the Action that is specified is STL_MAX_THREADS or STL_SET_BOTH, this is the name of a fullword that contains the new MaxThreads value for the caller's process.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_thread_limits service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_thread_limits service stores the return code. The set_thread_limits service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The set_thread_limits service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value that was specified for Action, MaxThreadTasks, or MaxThreads is incorrect. The following reason codes can accompany the return code: JRSTLActionInvalid, JRSTLTasksInvalid or JRSTLThreadsInvalid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_thread_limits service stores the reason code. The set_thread_limits service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the set_thread_limits service returns with an unsuccessful return value (-1), the original MaxThreadTasks and MaxThreads values for the caller's process remain unchanged.
2. If any caller, authorized or nonauthorized, attempts to set a limit outside the allowable ranges (see Table 21), the set_thread_limits service returns with a return code of EINVAL and a reason code of JRSTLTasksInvalid or JRSTLThreadsInvalid.

Table 21. Allowable thread limits for calling processes

	MaxThreadTasks		MaxThreads	
	Min	Max	Min	Max
Authorized	1	32768	0	100000
Unauthorized	1	Parmlib	0	Parmlib
Note: Parmlib represents the values that are specified at z/OS UNIX startup by the BPXPRMxx parmlib member.				

3. To determine the allowable ranges for pthread_created thread limits for non-authorized callers, see “sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options” on page 819.
4. For information on setting initial thread limits and performance considerations, see the following publications:
   - MAXTHREADS in *z/OS UNIX System Services Planning*
   - *z/OS MVS Initialization and Tuning Reference*

5. If the MaxThreadTasks limit is decreased below the number of tasks that are currently in use, pthread_exit_and_get requests fail until the number of tasks in use is less than or equal to the new limit.
6. Setting the MaxThreads limit to zero inhibits the creation of pthread_created threads.
7. Setting MaxThreads to be less than or equal to MaxThreadTasks prevents the queueing of pthread_create requests, and limits the number of MVS tasks that are attached for pthread_created threads to the MaxThreads value.
8. If the MaxThreadTasks limit of a process is set below the number of MVS tasks that are already in use for pthread_created threads, the reduction of MVS tasks is completed as running threads terminate. The reduction of tasks is not synchronously carried out when the set_thread_limits service is invoked.
9. For POSIX compliance, the MaxThreads limit for a process must be 64 or greater.

Related services

- "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497
- "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819
- "pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread" on page 505

Characteristics and restrictions

There are no restrictions on the use of the set_thread_limits service.

Examples

For an example using this callable service, see "BPX1STL (set_thread_limits) example" on page 1199.

set_timer_event (BPX1STE, BPX4STE) — Set DIE-mode timer event

Function

The set_timer_event callable service sets a DIE-mode timer event that posts an ECB when it expires. The ECB is located in the BPXYTHLI data area.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STE):	31-bit
AMODE (BPX4STE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STE,(Seconds,
              Nanoseconds,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STE with the same parameters.

Parameters

Seconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that represents the maximum number of seconds that the calling program is willing to wait for one of the specified events to occur.

Note:

1. Seconds can be any value greater than or equal to 0, and less than or equal to 4 294 967 295. The value specified for Seconds is an unsigned integer.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

Nanoseconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer that represents the number of nanoseconds to be added to the value that is specified by Seconds.

Note:

1. Nanoseconds can be any value greater than or equal to 0, and less than 1 000 000 000.
2. The Seconds and Nanoseconds values are combined to determine the timeout value.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service returns 0 if a CW_CONDVAR event occurred, or -1 if it has not.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_timer_event service stores the return code. The set_timer_event service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The set_timer_event service can return one of the following values in the Return_code parameter:

Return_Code	Explanation
EINVAL	One or more of the parameters that were passed to the service are in error. The following reason codes unique to the set_timer_event can accompany the return code: JRNanoSecondsTooBig, JRBadPET, JrReleasedPET.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the set_timer_event service stores the reason code. The set_timer_event service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. Once the time has expired, the kernel posts the ECB that is located at ThliTimerEcb, mapped by BPXYTHLI. The kernel clears this ECB before the timer is set.
2. The timer is canceled on the next syscall, or if the thread is terminated.
3. If the timer is set to a small enough value, the ECB that is defined at location ThliTimerEcb may already have been posted before control is returned to the caller.
4. If a valid unauthorized PET is stored in ThliPET before the BPX1STE/BPX4STE call, the kernel will RELEASE (IEAVRLS) the PE associated with ThliPET instead of posting ThliTimerEcb. The invoker of the PAUSE (IEAVPSE) will receive a release code of Thli#PauseTimeout.

Related services

- "cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event" on page 114

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1STE (set_timer_event) example" on page 1199.

setuid (BPX1SUI, BPX4SUI) — Set user IDs

Function

The setuid callable service sets the real, effective, and saved set user IDs for the current process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SUI):	31-bit
AMODE (BPX4SUI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SUI,(User_ID,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SUI with the same parameters.

Parameters

User_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the user ID the process is to assume.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setuid service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setuid service stores the return code. The setuid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The setuid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The user ID that was specified was incorrect.
EMVSSAF2ERR	The SAF call IRRSSU00 incurred an error.
EPERM	The process does not have the appropriate privileges to set the user ID. Refer to "Authorization" on page 8 for information on appropriate privileges.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the setuid service stores the reason code. The setuid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*. The reason code for EMVSSAF2ERR contains the RACF return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the RACF SETUID service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	UID is not defined to RACF
8	8	User not authorized to change UID
8	12	Internal error during RACF processing
8	16	Unable to establish recovery

For a more detailed description of the RACF CKPRIV service return and reason code values, see the following table:

RACF return code	RACF reason code	Explanation
8	4	User is not privileged
8	12	Internal error during RACF processing

Usage notes

1. If User_ID is the same as the real UID of the process or the saved set UID, the setuid service sets the effective UID to be the same as User_ID.
2. If User_ID is not the same as the real UID of the process, and the calling process has appropriate privileges, the real, effective, and saved set UIDs are set to User_ID. Refer to "Authorization" on page 8 for information on appropriate privileges.
3. In z/OS UNIX, you change the identity of a process by changing the real and effective UIDs and the supplementary groups. In order to change the identity of the process on MVS, you have to also change the MVS security environment. The setuid function calls SAF services to change the MVS identity (user ID) of the address space to the user ID that is associated with

the target UID only if the caller is a daemon, or if the target user ID has been properly authenticated. If the caller is a daemon, the following conditions must be met:

- The caller must be a superuser (UID=0).
- If the BPX.DAEMON profile is defined in the FACILITY class, the caller must be permitted to this profile.
- The calling program must be loaded from a controlled library, as defined by the RACF support for program access to data sets (PADS). (For more information, see the steps for setting up enhanced program security in *z/OS UNIX System Services Planning*.)

If the caller is not a daemon, the target user ID must have been authenticated in one of the following ways:

a. Successful completion of the password service, where the user ID specified is associated with the target UID of the setuid service.
b. If the caller of the setuid service has read access to the BPX.SRV.*userid* profile in the SURROGATE class, where *userid* is the user ID that is associated with the target UID, permission is granted to perform the specified operation. See Defining servers to process users without passwords in *z/OS UNIX System Services Planning* for more information about setting up surrogate profiles.

4. When the MVS identity is to be changed, the target MVS user ID is determined as follows:
   - If an MVS user ID is already known (saved) by the kernel from a previous call to the getpwnam or the password service calls, and the UID created for this user ID matches the UID that is specified on the setuid call, this saved user ID is used.
   - For nonzero target UIDs, if there is no known user ID, or if the UID for the known user ID does not match the UID that is requested on the setuid call, the setuid service queries the security database to retrieve the user ID. The retrieved user ID is then used.
   - If the target UID is 0 and a user ID is not known, the setuid service sets the MVS user ID to BPXROOT, or to a user ID that is specified as a parmlib option during installation. BPXROOT is set up during system initialization as a superuser with a UID of 0. The BPXROOT user ID is not defined to the BPX.DAEMON profile in the FACILITY class. This special processing is necessary to prevent a superuser from gaining daemon authority.
   - When a change is being made from a nonzero UID to a zero UID, the MVS user ID is not changed. When you use the **su** shell command to become a superuser, your shell retains your original MVS user ID.
5. The MVS identity is not changed on a successful call to setuid in the following situations:
   - When a change is being made from a nonzero UID to a zero UID. When you use the **su** shell command to become a superuser, your shell retains your original MVS user ID.
   - When it is running in a setuid program, and a setuid is done back to the original real UID.
6. You should be careful when you are constructing the MVS identity associated with a setuid program. These programs effectively allow a subsequently spawned child non-setuid program to set its effective UID and associated MVS identity to the UID and MVS identity of the setuid of the program.
7. The setuid service is not supported from an address space that is running multiple processes, because it would cause all processes in the address space

to have their security environments changed unexpectedly. The call to the setuid service in this environment fails with an EMVSERR return code and a JRMultiProc reason code.

8. The setuid service is not supported from a TSO address space. The call to the setuid service in this environment fails with an EMVSERR return code and a JRTso reason code.
9. The setuid service is not supported from a task that is currently running with a previously obtained task-level security environment. The call to the setuid service in this environment fails with an EMVSERR return code and a JRTaskAcee reason code.
10. The setuid service is not supported in the following situation: The BPX.DAEMON profile is defined in the FACILITY class and the caller is attempting to change its security environment by changing its MVS user identity, but a load was issued from an uncontrolled data set in the caller's address space. This address space could be corrupted; for this reason, daemon activity is not allowed. The call to the setuid service in this environment fails with an EMVSERR return code and a JREnvDirty reason code.
11. The setuid service is not supported when running from within a setuid program, because in most cases the MVS identity will not change in this environment.
12. To attach the security environment of the caller of the setuid service to the security environment of the target UID (which then creates a nested ACEE for the target), use the _BPXK_DAEMON_ATTACH environment variable. The new client can then access RACF delegated resources for which the daemon, but not necessarily the client, has access. (The delegated resources are designated by the APPDATA text of 'RACF-DELEGATED' in the RACF profile protecting the resource.) For more information about nested ACEEs and delegated resources, see *z/OS Security Server RACF Security Administrator's Guide*.
13. The _BPXK_SUID_FORK environment variable specifies whether the setuid indicator is propagated to child address spaces created by the fork service. For more information, see Commonly used environment variables in *z/OS UNIX System Services Planning*.
14. When the effective UID is changed, the MVS identity is also changed. The original MVS identity (ACEE) is saved so that the address space can be restored to its original security environment when the process terminates. During process termination, no other tasks in the address space should make any security calls since they cannot depend on the address space security environment and unpredictable results may occur.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219
- "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282
- "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672
- "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674

Characteristics and restrictions

If the setuid service is used within a multi-threaded process, use synchronization to ensure that the setuid service is not performed concurrently with other z/OS UNIX services. Unserialized use can yield unpredictable results.

See also "Usage notes" on page 711.

Examples

For an example using this callable service, see "BPX1SUI (setuid) example" on page 1201.

shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment

Function

The shmat service attaches the shared memory segment that is associated with a shared memory identifier.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state; PSW key 2, 8, or 9
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MAT):	31-bit
AMODE (BPX4MAT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MAT,(Shared_Memory_ID,
              Shared_Memory_Address,
              Shared_Memory_Flag,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers need an additional parameter:

```
CALL BPX4MAT,(Shared_Memory_ID,
              Shared_Memory_Address,
              Shared_Memory_Flag,
              Attached_Address
              Return_value,
              Return_code,
              Reason_code)
```

The Shared_Memory_Address parameter is a doubleword.

Parameters

Shared_Memory_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the shared memory identifier that is returned by the shmget service.

Shared_Memory_Address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

For BPX1MAT, the name of a field that contains a 31-bit address of where in the caller's address space storage is to be obtained and the segment is to be attached, or a 31-bit zero.

For BPX4MAT, the name of a field that contains a 64-bit address of where in the caller's address space storage is to be obtained and the segment is to be attached, or a 64-bit zero.

- If Shared_Memory_Address is a null pointer, the segment is attached at the first available address selected by the system that is on a page boundary; or on a megabyte boundary, if the shared memory segment is defined as an IPC_MEGA segment.
- If Shared_Memory_Address is not a null pointer and Shm_RND is specified, the segment's storage address is truncated to a page boundary (last 12 bits zero); or to a megabyte boundary (last 20 bits zero), if the shared memory segment is defined as an IPC_MEGA segment.
- If Shared_Memory_Address is not a null pointer and Shm_RND is not specified, the segment is attached at the address that is specified. If the shared memory segment is defined as an IPC_MEGA segment, the specified address must be a megabyte multiple, or the request is failed with an EINVAL.

For BPX4MAT, the address must always be the same, and it must be above the bar.

If the shared memory segment is defined as an IPC_MEGA segment, the value that is specified in Shared_Memory_Address must be either zero or equal to or greater than 16 megabytes; otherwise, the request fails with an EINVAL.

If the segment being attached is above the 2G bar (that is, it is a 64-bit address), the Shared_Memory_Address must either be zero or the same address that was returned on the shmgt() call. If the segment being attached is below the bar (that is, it was created with option IPC_BELOWBAR or IPC_MEGA on the BPX4MGT call), then the Shared_Memory_Address follows the same rules as for BPX1MAT callers.

Shared_Memory_Flag
Supplied parameter

Type: Integer

Length:
Fullword

Shm_RDONLY identifies the segment that is to be attached for read only; otherwise, the segment is attached for read and write. Shm_RDONLY has no effect for attaches to shared memory segments that are created with the IPC_MEGA option. Whether the segment is attached read only or read and write depends on how it is currently accessed by other attaches, as all users have the same access to shared memory that is created with the IPC_MEGA option.

Shm_RND causes the Shared_Memory_Address to be truncated to a page boundary (last 12 bits zero), or to a megabyte boundary (last 20 bits zero) if the shared memory segment is defined as an IPC_MEGA segment.

Attached_Address
Returned parameter (BPX4MAT only)

Type: Integer

Length:
Doubleword

The name of a doubleword in which the shmat service returns the shared memory segment address (the address that is to be passed to the detach) when Return_value is zero.

Return_value
Returned parameter

Type: Address

Length:
Fullword

The name of a fullword in which the shmat service returns the shared memory segment address (the address that is to be passed to the detach), or -1, if the operation is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmat service stores the return code. The shmat service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The shmat service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: • Operation permission is denied to the caller. The combination of Shared_Memory_Flag and permissions denies the requester access. The following reason code can accompany the return code: JRIpcDenied. • The caller's PSW key does not match the key of the shared memory segment (except that callers running in PSW key 8 can attach to a key 9 shared memory segment). The following reason code can accompany the return code: JrKeyMismatch.

Return_code	Explanation
EINVAL	Shared_Memory_ID is not a valid shared memory identifier, for one of the following reasons: • Shared_Memory_Address is not zero, it is not on a page boundary, and SHM_RND was not specified. • Shared_Memory_Address is not zero, it is not on a megabyte boundary, and SHM_RND was not specified. • The storage at Shared_Memory_Address could not be obtained in the user's address space. • The caller is not running with a PSW key of 2, 8, or 9. The following reason codes can accompany the return code: JRIpcBadID, JRBadAddress, JRNoUserStorage, JRStorNotAvail, or JrUnsupportedKey.
EMFILE	The number of shared memory segments attached to the caller's process exceeds the system-imposed maximum. This system limit is set with the IPCSHMNSEGS parameter in a BPXPRMxx parmlib member. You can use the **ipcs -x** shell command to view this value. The following reason code can accompany the return code: JRShmMaxAttach.
ENOMEM	The available system storage is not large enough to accommodate the shared memory segment. The following reason codes can accompany the return code: JRNoUserStorage, JRSMNoStorage, JRIarvserv or JRShrStgShortage.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmat service stores the reason code. The shmat service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. If an attempt is made to access memory outside the shared memory segment, normal address space storage is accessed.
2. It is the application's responsibility to determine the length of the shared memory segment that is attached.
3. If an attempt is made to update a shared memory segment that is attached with Shm_RDONLY access, a program check occurs.
4. Because of the nature of the mapping of shared memory segments to different addresses within the multiple processes it is attached to, relative addresses should be used as pointers within the shared memory segment.
5. The storage is allocated in subpool 129, which is associated with the job step task. This allows a thread to attach a shared memory segment and exit, allowing other threads in the process to access the storage.
6. See “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 738, usage note 10 on page 742, for an explanation of the storage key chosen for a shared memory segment.
7. Above the bar, shared memory cannot be used in subspace mode.

Related services

- "shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations"
- "shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment" on page 722
- "shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment" on page 738

Characteristics and restrictions

- The invoker is restricted by ownership and read and write permissions defined by shmget and shmctl IPC_SET.
- Restricted to callers running in PSW key 2, 8, or 9. Authorized users can exploit the IARVSERV macro directly to create shared memory in system keys.

Examples

For an example using this callable service, see "BPX1MAT (shmat) example" on page 1160.

shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations

Function

The shmctl service provides a variety of shared memory control operations as specified by the Command parameter. These functions include reading and changing shared memory variables with the shmid_ds data structure, and removing a shared memory segment from the system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state; PSW key 2, 8, or 9
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MCT):	31-bit
AMODE (BPX4MCT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MCT,(Shared_Memory_ID,
              Command,
              Buffer_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MCT with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

Shared_Memory_ID
Supplied parameter

Type: Integer

Length:
Fullword

Specifies the shared memory identifier.

Command
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword field that indicates the shared memory command that is to be executed. For the structure that contains these constants, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039 and "BPXYIPCP — Map interprocess communication permissions" on page 987. The values for Command are:

Ipc_STAT
This command obtains status information about the shared memory that is identified by the Shared_Memory_ID parameter, if the current process has read permission. This information is stored in the area that is pointed to by the Buffer_address parameter and mapped by area MSQID_DS data structure. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039, SHMID_DS DSECT.

Ipc_SET
This command sets the value of the IPC_UID, IPC_GID and IPC_MODE from the SHMID_DS data structure that is associated with Shared_Memory_ID into the SHMID_DS structure that is pointed to by the Buffer_address parameter. Any value for IPC_UID and IPC_GID may be specified. Only mode bits that are defined by semget under Semaphore_Flag argument may be specified in the IPC_MODE field. This command can only be executed by a process with an effective user ID equal to either that of a process with appropriate privileges (see "Authorization" on page 8), or to the value of IPC_CUID or IPC_UID in the SHMID_DS data structure that is associated with Shared_Memory_ID. This information is taken from the buffer pointed to by the Buffer_address parameter. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039, SHMID_DS DSECT.

For shared memory segments that were not created with the Ipc_MEGA option, the permissions that are in effect (IPC_MODE) when a process attaches a segment remain, even though these permissions may change. For shared memory segments that were created with the Ipc_MEGA option, the permissions that are set by this request take effect immediately. All processes that are currently attached to the shared memory segment are able to read only or read and write to it based on the permissions that are specified in the IPC_MODE.

The effect of the new mode on access is determined by the three parts of the mode field: the owner permissions, the group permissions, and other permissions. If all three read and all three write permissions in the new mode are off, the access for all attached processes is changed to read. If any of the three read permission bits is on without the corresponding write permission bit on, the access for all attached processes is changed to read. Otherwise, the access for all attached processes is changed to write.

Ipc_RMID
This command removes the shared memory identifier that is specified by Shared_Memory_ID from the system, and removes the shared memory segment and SHMID_DS data structure that are associated with it. This command can only be executed by a process with an effective user ID equal to either that of a process with appropriate privileges (see "Authorization" on page 8), or to the value of IPC_CUID or IPC_UID in the SHMID_DS data structure that is associated with Shared_Memory_ID.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the buffer that is mapped by SHMID_DS. The shmctl service assumes that the size of this buffer is at least as large as SHMID_DS.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmctl service stores the return value, or a -1 if the operation is unsuccessful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmctl service stores the return code. The shmctl service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The shmctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The command specified was Ipc_STAT, and the calling process does not have read permission. The following reason code can accompany the return code: JRIpcDenied.
EFAULT	The Buffer_Address parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code: JRBadAddress.

Return_code	Explanation
EINVAL	This error code may be returned for the following reasons: • Shared_Memory_ID is not a valid shared memory identifier. • The Command parameter is not a valid command. • The mode bits were not valid (SET). The following reason codes can accompany the return code: JRIpcBadFlags, JRIpcBadID and JRBadEntryCode.
EPERM	Command=IPC_RMID or IPC_SET, and the effective user ID of the caller is not that of a process with appropriate privileges (see "Authorization" on page 8), and is not the value of IPC_CUID or IPC_UID in the SHMID_DS data structure that is associated with Shared_Memory_ID. The following reason code can accompany the return code: JRIpcDenied.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmctl service stores the reason code. The shmctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

1. The remove operation is asynchronous to the return from the system call after the last attachment is broken.
2. When a RMID is processed, no further attaches are allowed.
3. Ipc_SET can change permissions, and may affect a thread's ability to use the shared memory functions.
4. If an RMID was processed before a fork service, the child is not attached to the memory segment.
5. Above the bar, shared memory cannot be used in subspace mode.

Related services

- "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890
- "shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment" on page 714
- "shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment" on page 722
- "shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment" on page 738

Characteristics and restrictions

The caller of the shmctl service is restricted by ownership and read and read-write permissions that are defined by shmget and shmctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1MCT (shmctl) example" on page 1160.

shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment

Function

The shmdt service detaches a shared memory segment.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state; PSW key 2, 8, or 9
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MDT):	31-bit
AMODE (BPX4MDT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MDT,(Shared_Memory_Address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MDT with the same parameters. The Shared_Memory_Address parameter is a doubleword.

Parameters

Shared_Memory_Address
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the starting address of a shared memory segment. This is the Return_value from shmat (BPX1MAT,BPX4MAT). The address returned is 31 bits for AMODE 31 callers and 64 bits for AMODE 64 callers.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmdt service returns `0` if the request was successful, or `-1` if the operation was unsuccessful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmdt service stores the return code. The shmdt service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The shmdt service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	Shared_Memory_Address is not the data segment start address of a shared memory segment attached to the caller's process. The following reason code can accompany the return code: JRBadAddress.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmdt service stores the reason code. The shmdt service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Related services

- "shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment" on page 714
- "shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations" on page 718
- "shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment" on page 738

Usage notes

- When a shared memory segment is detached via shmctl(), the process loses access to the associated shared memory. If this is done before a process has finished using all condition variables and mutexes in the detached shared memory segment, unpredictable results will occur in the application, leading to possible hangs and loss of resources. Because a process is implicitly detached from its shared memory attachments when it terminates, most applications should avoid doing any explicit shmdt() calls before terminating.
- Above the bar, shared memory cannot be used in subspace mode.
- For a segment of type IPC_SHAREAS, a detach call will not cause the cleanup of the segment's storage within the user address space unless no other processes in the address space are attached to the segment.

Characteristics and restrictions

The caller of the shmdt service is restricted by ownership and read and read-write permissions that are defined by shmget and shmctl Ipc_SET.

Examples

For an example using this callable service, see "BPX1MDT (shmdt) example" on page 1161.

shmem_lock (BPX1SLK, BPX4SLK) — Shared memory lock service

Function

The shmem_lock callable service provides a general-purpose interface for managing and operating locks in shared memory. It allows an application to serialize resources that must be shared across multiple address spaces.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SLK):	31-bit
AMODE (BPX4SLK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1SLK,(LockFcnCode,
               LockReqType,
               LockType,
               LockAddr,
               LockAttrAddr,
               LockTokenAddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SLK with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

LockFcnCode

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value indicating the function requested. The following are the supported values:

SLK_INIT
A new shared memory lock is to be created and initialized.

SLK_DESTROY
A shared memory lock is to be destroyed and its resources cleaned up.

SLK_OBTAIN
A shared memory lock is to be obtained unconditionally.

SLK_OBTAIN_COND
A shared memory lock is to be obtained on the condition that is not already obtained. If the requested lock is not available immediately, the request will fail (EBUSY) without blocking.

SLK_RELEASE
A shared memory lock is to be released.

These constants are defined in the BPXYCONS macro ("BPXYCONS — Constants used by services" on page 952).

LockReqType
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one or more of the following values indicating the lock request type. This parameter is valid only for the SLK_INIT function, and is ignored for all other functions. The following are the supported values:

SLK_NORMAL
A new shared memory lock is to be created with no deadlock detection. This value is mutually exclusive with the SLK_ERRORCHECK value.

SLK_ERRORCHECK
A new shared memory lock is to be created with deadlock detection. This value is mutually exclusive with the SLK_NORMAL value.

SLK_RECURSIVE
A new shared memory lock is to be created with a recursive locking capability. This allows the same lock to be obtained multiple times by the same caller, without requiring intervening releases and without causing deadlock. To take advantage of this capability, the lock must be obtained with the same lock type on each obtain call.

These constants are defined in the BPXYCONS macro ("BPXYCONS — Constants used by services" on page 952).

LockType
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains one or more of the following values indicating the lock type. This parameter is valid only for the SLK_INIT and SLK_OBTAIN functions; it is ignored for all other functions. For the SLK_INIT function, the request type values can be combined to create a multiple-type lock (that is, a lock that can be obtained either shared or exclusively). For the SLK_OBTAIN function, only one of the values can be specified on a given call. The following are the supported values:

SLK_SHARED
A shared memory lock is to be created or obtained with the shared attribute. A lock that is obtained with the shared attribute can be obtained concurrently by other callers requesting a shared lock obtain. A lock initialized with this value is, by default, defined as a recursive lock.

SLK_EXCLUSIVE
A shared memory lock is to be created or obtained with the exclusive attribute. A lock that is obtained with the exclusive attribute cannot be obtained concurrently by other callers.

These constants are defined in the BPXYCONS macro; see"BPXYCONS — Constants used by services" on page 952.

LockAddr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the user lockword in shared memory.

LockAttrAddr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the lock attribute area. The LockAttrAddr parameter is for use with the SLK_INIT function only. It is intended to allow for potential extensions to the shared memory locks. Because these extensions are not currently supported, the caller of the shmem_lock service should specify a null pointer for the lock attribute area address.

LockTokenAddr
Supplied and returned parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a fullword that the service uses either to return a lock token, or as the input lock token. When it is specified with the SLK_INIT function, the LockTokenAddr parameter is used as the address of an output area in which to return the lock token of the newly created lock. For all other functions, this parameter contains the address of the lock token that represents the lock to be operated upon.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmem_lock service returns 0, if the request is successful; or -1, if it is not successful. For all successful SLK_INIT and SLK_DESTROY function requests, the shmem_lock service returns 0. For successful SLK_OBTAIN, SLK_OBTAIN_COND, and SLK_RELEASE function requests, the shmem_lock service returns a count of the number of times the calling thread has had the requested lock held.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmem_lock service stores the return code. The shmem_lock service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The shmem_lock service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The requested service could not be performed at the current time because of a lack of available system resources. The following reason codes can accompany this return code: JRTLOCKMAXCNTTHD, JRLOCKMAXCNTSYS, JRLOCKMAXCNTRECURSE.
ENOMEN	The requested service could not be performed at the current time because of a lack of available system storage.
EINTR	A signal interrupted the callable service.
EINVAL	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRLOCKFCNCODE, JRLOCKREQTYPE, JRLOCKTYPE, JRLOCKADDR, JRLOCKTOKEN.
EFAULT	One of the parameters contains an address that is not accessible by the caller. The following reason code can accompany the return code: JRLOCKTOKENADDR.
EBUSY	The specified function cannot be performed because a required resource is already in use. The following reason codes can accompany the return code: JRLOCKINUSE, JRLOCKEDALREADY.
EPERM	The caller is not permitted to perform the specified operation. The following reason codes can accompany the return code: JRLOCKNOTOWNER, JRLOCKSHMACC.
EDEADLK	The caller already owns the lock that is requested.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmem_lock service stores the reason code. The shmem_lock service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. **Lock initialization**

   In order for a lock initialization call to complete successfully, the specified lock address must be in a memory-mapped area or shared storage segment that is read-write accessible by the calling process. To most efficiently manipulate locks in shared storage, it is recommended that the lock be in a shared memory segment. The shmem_lock service is optimized to handle locks that reside in a shared memory segment, rather than a memory-mapped area.

   A successful lock initialization call causes a lock token representing the newly created lock to be returned in the lock token output area that is supplied via the LockTokenAddr parameter. Subsequent calls to the shmem_lock service to manipulate the newly created lock must specify the returned lock token in order to identify the lock that is to be manipulated. A lock initialization call can fail if system resources other than storage are not available to initialize the lock (EAGAIN), or if not enough system storage is available (ENOMEM).

2. **Lock destroy**

   A destroy of a lock causes the system resources for that lock to be cleaned up, if the lock is not in use. If the lock is in use, the destroy request fails (EBUSY). A lock could be in use if a thread has it in a locked state, or if it is being referenced by another thread on a pthread_cond_timedwait or pthread_cond_wait. Once a lock is destroyed, any further operations against that lock fail (EINVAL).

3. **Lock obtain**

   A successful call to the shmem_lock service to obtain a shared memory lock results in a GRS latch obtain against a latch in the 'SYS.BPX.AP00.GXSLT.SHMLOCKS.LSN' latch set. If an application is experiencing serialization problems with a shared memory lock, GRS contention analysis tools such as D GRS,C and IPCS ANALYZE can be used to determine the cause of the problem. The lower halfword of the lock token that is returned by the shmem_lock service indicates the latch number of the corresponding latch within the 'SYS.BPX.AP00.GXSLT.SHMLOCKS.LSN' LATCH set.

   If an exclusive obtain of a lock that is defined as both exclusive and shared is attempted by a thread that already has that lock obtained exclusively, deadlock results. Additionally, if an exclusive or shared obtain of a shared and exclusive lock is attempted by a thread that already has that lock obtained exclusively, deadlock results. To prevent exclusive obtain starvation for a lock that is defined as shared and exclusive, a new shared lock obtain blocks if there are any exclusive obtain callers waiting. A lock that is initialized with the recursive attribute can be obtained multiple times by the same thread, up to a limit of 32 768 iterations. A single thread can hold up to a limit of 128 different shared memory locks concurrently.

4. **Lock release**

   A lock release call against a lock that is not in a locked state or that is not owned by the calling thread results in an error (EPERM). A lock with the recursive attribute that has been obtained *n* times by a given thread must be released *n* times by that same thread in order for the lock to be completely released.

5. **System cleanup**

   During task termination processing of a thread that ends while it is holding a shared memory lock, the lock is released by the system. If a jobstep ends

abnormally (for example, if it is canceled), or if an address space is terminated at end of memory, all shared memory locks that are held by that job or address space are released.

Related services

None.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1SLK (shmem_lock) example" on page 1190.

shmem_mutex_condvar (BPX1SMC, BPX4SMC) — Shared mutex and condition variable service

Function

The shmem_mutex_condvar callable service provides a general-purpose interface for managing and operating mutexes and condition variables in shared memory. An application can:

- Create and initialize a shared memory mutex or condition variable
- Destroy a shared memory mutex or condition variable and clean up its resources
- Post the oldest waiter for a specified mutex or condition variable
- Post all of the waiters for a specified mutex or condition variable
- Wait for a specified condition variable
- Setup to wait for a mutex
- Cancel setup to wait for a specified mutex
- Wait for a specified condition variable and post any waiters for the specified mutex

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SMC):	31-bit
AMODE (BPX4SMC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1SMC,(FcnCode,
               FcnFlags,
               ShrObj1Addr,
               ShrObj2Addr,
               EcbAddr,
               TimeStrucAddr,
               UserDataAddr,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SMC with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

FcnCode
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value indicating the function requested. The following values are supported:

SMC_INIT
Create and initialize a new shared memory mutex or condition variable.

SMC_DESTROY
Destroy a shared memory mutex or condition variable and clean up its resources.

SMC_POST
Post the oldest waiter for the specified mutex or condition variable.

SMC_POSTALL
Post all of the waiters for the specified mutex or condition variable.

SMC_WAIT
Wait for the specified condition variable.

SMC_SETUPTOWAIT
Setup to wait for a mutex. This function is not supported for condition variables.

SMC_CANCELSETUPTOWAIT
Cancel set up to wait for the specified mutex or condition variable.

SMC_WAIT+SMC_POST
Wait for the specified condition variable and post any waiters for the specified mutex.

These constants are defined in the BPXYCONS macro ("BPXYCONS — Constants used by services" on page 952).

FcnFlags
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the function flags for the requested function. The possible values for FcnFlags are:

SMC_Mutex
The input shared memory object represents a mutex. This setting must be specified on all function calls that involve a mutex object.

SMC_Condvar
The input shared memory object represents a condition variable. This setting must be specified on all calls that involve a condition variable. SMC_Condvar or SMC_Mutex must be specified on all calls to BPX1SMC/BPX4SMC.

SMC_TimedWait
Wait for a specified time interval for the specified condition variable. This option is relevant only for a condition variable. When SMC_TimedWait is specified, the TimeStrucAddr parameter points to the time structure that indicates the amount of time to wait.

SMC_OutsideWait
The caller will wait outside of the BPX1SMC/BPX4SMC function for the specified mutex or condition variable. This setting is relevant only for the SMC_SetupToWait function. SMC_OutsideWait must be specified when SMC_Mutex is specified.

ShrObj1Addr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the shared condition variable or mutex control structure (SMCV or SMMX) that is involved in the specified operation. If SMC_MUTEX is specified in the FcnFlags, this parameter must point to a valid shared mutex control structure (SMMX). If SMC_CONDVAR is specified in the FcnFlags, this parameter must point to a valid shared condition variable control structure (SMCV).

ShrObj2Addr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the mutex control structure (SMMX) that is associated with the supplied shared condition variable structure (SMCV). When SMC_CONDVAR is specified in the FcnFlags for a SMC_Wait or SMC_SetupToWait function request, this parameter must contain a pointer to a valid SMMX. For all other calls to this service, this parameter must be specified, but its value will not be validity checked.

EcbAddr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the ECB that the caller will wait on for the specified mutex or condition variable. For the SMC_SetupToWait function for a mutex object, this parameter must contain a valid ECB address. For all other calls to this service, this parameter must be specified, but its value will not be validity checked. For both BPX1SMC and BPX4SMC callers, the ECB must be in below-the-bar storage.

TimeStrucAddr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of the time structure (SMCT) that indicates the amount of time the caller will wait for a specified condition variable. When the SMC_CondVar and SMC_TimedWait function flags are specified with the SMC_SetupToWait or SMC_Wait function, this parameter must point to a valid SMCT structure. For all other calls to this service, this parameter must be specified, but its value will not be validity checked.

UserDataAddr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) pointer field that contains the address of user data supplied by the caller for problem determination support. This parameter must be specified, but its value will not be validity checked.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service returns `0` if the request is successful, or -1 if it is not successful. For a successful initialization call (SMC_INIT), the input shared memory object is filled in as appropriate for an SMMX (mutex) or SMCV (condition variable).

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service stores the return code. The BPX1SMC/BPX4SMC service returns Return_code only if Return_value is -1. The BPX1SMC/BPX4SMC service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	The requested service could not be performed at the current time because of a lack of available system resources. The following reason codes can accompany this return code: JRSMCMaxCntSys, JRSMCMaxCntSeg.
ETIMEDOUT	The requested service reached the specified time-out interval.
EINTR	A signal interrupted the callable service. The following reason code can accompany this return code: JRSignalArrived.
EINVAL	One of the parameters contains a value that is not correct. The following reason codes can accompany this return code: JRSMCFcnCode, JRSMCFcnFlags, JRSMCWrongMutex, JRSMCNotMutex, JRSMCNotCondVar, JRSMCAlreadySetUp, JRSMCNotShared, JRSMCUnusable, JRSMCMutexSetUp, JRSMCMemoryMap, JRSMCNotOwner, JRSMCDisabled.
EFAULT	One of the parameters contains an address that is not accessible by the caller. The following reason codes can accompany this return code: JRSMCObjAddr, JRSMCEcbAddr, JRSMCTimesTrAddr.
EBUSY	The initialization or destroy function cannot be performed because the specified object is already in use. The following reason codes can accompany this return code: JRSMCWaiters, JRSMCMutexLocked, JRSMCCondWaiters, JRSMCAlreadyInit.
EPERM	The caller does not have the appropriate privilege to perform the operation. The following reason code can accompany this return code: JRSMCShMAcc.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPX1SMC/BPX4SMC service stores the reason code. The BPX1SMC/BPX4SMC service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. **The initialization function (SMC_INIT)**

   You initialize a mutex or condition variable by calling the initialization function (SMC_INIT), supplying the shared memory address of the mutex or condition variable. A valid shared memory address is one that is within a shared memory segment that is read/write accessible to the calling process. If a call to initialize a mutex or condition variable supplies an object that is not in shared memory, the call fails with Return_code EINVAL and Reason_code JRSMCNotShared. In this case, the Language Environment pthread_cond_init() and pthread_mutex_init() functions treat the object as a non-shared object, because it is not accessible to any other process.

   A mutex or condition variable is represented by two shared memory serialization structures that are maintained between Language Environment and the kernel: the shared memory mutex block (SMMX) and the shared memory condvar block (SMCV). The C/Language Environment pthread_mutex_init() and pthread_cond_init() functions indicate on the initialization call to BPX1SMC/BPX4SMC the type of call that is being requested by setting the FcnFlags value to SMC_Mutex or SMC_Condvar. The BPX1SMC/BPX4SMC initialization function initially fills in an SMMX or

SMCV structure with information that is pertinent to the mutex or condition variable, and returns the associated kernel-shared memory serialization token in that structure. This kernel token is used by the kernel as the anchor to the kernel control structures for the object. The address of the SMMX and/or SMCV must be supplied on all subsequent calls to BPX1SMC/BPX4SMC for the waiting, posting and destroying of the shared memory mutex or condition variable that is associated with the SMMX or SMCV structure.

In order to prevent overuse of kernel resources, z/OS UNIX imposes limits on the number of shared mutex and condition variables that can be initialized for any one shared memory segment and on any given system. For applications running on behalf of unauthorized users the limits are 65.535 per shared memory segment and 131,072 for the system. For applications running on behalf of authorized users the limits are 4,194,304 (4M) per shared memory segment and 134,217,728 (128M) for the system. An application is authorized if it runs with UID 0 or if it runs with an effective UID that has READ access to the SUPERUSER.SHMMCV.LIMITS resource in the UNIXPRIV class. Each shared memory mutex or condition variable requires roughly 80 bytes of storage in the OMVS address space, so the use of large numbers of mutexes and condition variables causes a significant increase in the amount of storage used in the OMVS address space. Refer to the discussion of the SUPERUSER.SHMMCV.LIMITS resource in *z/OS UNIX System Services Planning* for additional considerations.

2. **The post function (SMC_POST)**

   The post function is used during a pthread_cond_signal(), or when a mutex is unlocked (pthread_mutex_unlock(), pthread_cond_wait(), or pthread_cond_timed_wait()) and there are waiters for a condition variable or mutex. The post function wakes up the oldest waiter. The post call that is done from pthread_mutex_unlock is made after the lock that is associated with the mutex (SMMX) is released. If a post is done against a mutex or condition variable that has no waiters, the call will succeed, but a subsequent wait call will still block. In other words, pre-signalling of a condition variable cannot be done.

3. **The postall function (SMC_POSTALL)**

   The postall function is used during a pthread_cond_broadcast(); it causes all waiters for a condition variable to be awoken. When all waiters have woken up, they contend for the mutex that is associated with the specified condition variable.

4. **The setup to wait function (SMC_SETUPTOWAIT)**

   The setup to wait function is used during a pthread_mutex_lock() function when waiting is necessary. Whenever this function needs to block, it must be called prior to waiting outside the kernel. The FcnFlags input parameter indicates an outside wait (SMC_OutsideWait), and the EcbAddr parameter must point to the ECB that will be waited upon. If the SMC_Timed_Wait flag is on, the TimeStrucAddr parameter must point to a valid SMCT time structure that describes the amount of time required for the timed wait. The setup to wait function is supported only for mutex objects.

5. **The wait function (SMC_WAIT)**

   The wait function can be used by itself to wait for a condition variable, or it can be combined with the post function as a way to wait for a condition variable, at the same time posting any waiters for a mutex. The wait function cannot be used for mutexes, because mutex waits are not signal enabled.

6. **The wait and post function (SMC_WAIT+SMC_POST)**

The combined wait and post function is intended for use on a pthread_cond_timedwait() and pthread_cond_wait() as a way to minimize system overhead. When SMC_WAIT+SMC_POST is used, the ShrObj1Addr parameter must point to a valid SMCV, and the ShrObj2Addr parameter must point to a valid SMMX. The wait function internally performs a setup to wait for the condition variable, and then performs the post of the mutex. If a thread does a setup to wait function call followed by a wait function call for the same object, the wait function will fail.

As part of the post operation, the SMMX lock is updated with CDS to indicate that the caller no longer owns the mutex. After attempting the post of the mutex object, the condition variable is waited upon. After waking up from the condition variable wait, the mutex is reobtained by BPX1SMC/BPX4SMC. Because the mutex may not be available immediately, the service may have to wait for the mutex. The first time a condition variable is waited upon with an associated mutex, the condition variable is tied to the specified mutex for the life of the condition variable and mutex. No other mutex can be associated with the specified condition variable, and no other condition variable can be associated with that mutex, until the condition variable or mutex is destroyed.

7. **The cancel setup to wait function (SMC_CANCELSETUPTOWAIT)**

   The cancel setup to wait function must be called any time a setup to wait is done and a wait for the resource is not performed. If a cancel setup to wait fails, the caller may have already been posted for the associated resource. This function would probably only be used if it were detected that a resource (such as a mutex) had become available without requiring a wait.

8. For an asynchronous signal delivered to the thread, or for thread cancelation processing of a thread that is blocked on a condition variable when the cancelability enable state of the thread is set to PTHREAD_CANCEL_DEFERRED, the BPX1SMC/BPX4SMC service unblocks the thread and returns to Language Environment with an EINTR return code. Language Environment handles this in the same way that it handles an EINTR returned from BPX1CWA. The EINTR return code is not surfaced to the C application.

9. The user data address that is supplied on the call to BPX1SMC/BPX4SMC is used by Language Environment to supply the stack address for the calling thread. This data is recorded for all waiters for a mutex or condition variable, and is displayed for each requestor of a mutex or condition variable on the D OMVS,SER report. When Language Environment successfully obtains a mutex on behalf of a caller, the SMMXOwnerData field is filled in with this information for the same purpose.

10. If BPX1SMC/BPX4SMC is asynchronously interrupted by an abnormal condition (such as a X'22' abend) during critical condition variable and/or mutex processing, the condition variable and/or mutex are invalidated and made unusable. Only a destroy of the condition variable and/or mutex can be performed on the object after the interrupt.

11. **Shared memory data area structures: the SMMX and the SMCV**

    The SMMX (BPXYSMMX) and the SMCV (BPXYSMCV) data area structures represent two new C data types that are supported by the C RTL/Language Environment: the new larger pthread_mutex_t data type for shared-memory-resident mutexes, and the new larger pthread_cond_t data type for shared-memory-resident condition variables, respectively. These data types must be defined to be on a doubleword boundary that is enforced by the definitions of the data type.

    The SMMX and the SMCV are architected for use between Language Environment and the kernel. So that Language Environment can distinguish

between the various types of mutexes and condition variables it supports, the first 8 bytes of the SMMX and SMCV must have specific bits on.

- **SMMX**

  To represent the new larger type of shared mutex, the SMMX must have bits 0, 31, 62, and 63 on in the first 8 bytes of the structure. The characters SMMX (X'E2D4D4D7'') are chosen for the ID field (the first 4 bytes) so that they meet this requirement for bits 0 and 31, at the same time providing a representative eye-catcher for the structure. The second 4 bytes of the structure are a flag word (SmmxFlags), in which the low-order two flag bits must always be initialized to being on, and must never be moved from their bit positions.

  Language Environment must serialize the use of a mutex that is represented by this structure, using CDS on the lock doubleword. The first half of the lock is an ID that uniquely identifies the owner. Language Environment must use the first four bytes of the thread ID of the calling thread as the lock owner ID, because this is guaranteed to be unique for the life of the owning thread, and is useful in providing problem determination support. The second half of the lock is a 3-byte count field of waiters for the mutex and a status byte. If the mutex is not in use and is not destroyed, the lock doubleword is all zeros. If the mutex is in use with no waiters, the lock owner ID field is nonzero, and the waiter count field is zero. If the mutex is in use with waiters, both fields are nonzero. The kernel uses the information in the SMMX structure to provide cleanup and problem determination support for shared memory mutexes.

  As an example, on a pthread_mutex_lock operation, Language Environment would attempt to set the first word of the lock to the first half (first four bytes) of the caller's thread ID, and set the waiter count field in the lock doubleword to its current value. If it can do this atomically with CDS, the mutex will be obtained without contention, and no call to BPX1SMC/BPX4SMC will be necessary. This allows a minimum of calls to the kernel.

  The pthread_mutex_init() and pthread_mutex_destroy operations against a shared memory mutex are serialized by the BPX1SMC/BPX4SMC service and the use of the lock status byte. The lock status byte is updated atomically with CDS by BPX1SMC/BPX4SMC on a destroy operation, to indicate that the mutex has been destroyed. If the mutex is not in use, the CDS will succeed, and any further operations against the mutex will fail. BPX1SMC/BPX4SMC will also ensure that the same shared memory area is not initialized more than once.

- **SMCV**

  The SMCV data area is created for shared memory condition variables. To represent the new larger type of shared mutex, the SMCV must have bits 0, 31, 62 and 63 on in the first 8 bytes of the structure. The characters SMCV (X'E2D4C3E5') are chosen as the ID (the first 4 bytes), so they meet the requirement for bits 0 and 31, and at the same time provide a representative eye-catcher for the structure. The second 4 bytes of the structure are a flag word, in which the low-order two flag bits in the word must always be initialized to being on, and must never be moved from their bit positions.

  Serialization over this structure is provided mainly by the BPX1SMC/BPX4SMC service. This service must be called on all pthread_cond functions for a shared condition variable, unless there are no waiters for the condition variable on a pthread_cond_signal. Language Environment increments the waiter count on a pthread_cond_wait (or timed wait) before doing the call to BPX1SMC/BPX4SMC to wait and post, and

decrements the count after waking up and receiving control back from the BPX1SMC/BPX4SMC service. The wait count can be incremented only if the SmcvUnusable flag is off in the lockword flag field. If this flag is on, the condition variable has been destroyed and is no longer usable. On a pthread_cond_signal, if the wait count is found to be zero, no call to the kernel is necessary.

12. **CondTimedWait structure (SMCT)**

    The CondTimedWait structure is used for timed waits against shared condition variables, and contains the amount of time to wait in seconds and nanoseconds. The SMCT structure maps directly to the timespec structure that is currently supplied on the C pthread_cond_timedwait() function.

13. **Shared memory remove**

    When a shared memory segment is removed with shmctl(), all condition variables and mutexes in the removed shared memory segment must be destroyed if the shared memory segment is to be cleaned up on the remove operation. This may involve waking up any waiters that are still using the mutex or condition variable that is being cleaned up. The actual cleanup for a shared memory segment may be delayed until the last attachor detaches. If the shared memory segment is not actually cleaned up on the remove operation, the shared condition variables and mutexes are still usable by the processes that are still attached to the shared memory segment.

14. **Shared memory detach**

    When a shared memory segment is detached with shmctl(), the process loses access to the associated shared memory. If this is done before a process has stopped using all condition variables and mutexes in the detached shared memory segment, unpredictable results will occur in the application, possible leading to hangs and loss of resources. Because a process is implicitly detached from its shared memory attachments when it terminates, it is recommended for most applications that they avoid doing any explicit shmdt() calls before terminating.

15. **Process/thread termination**

    When a single thread ends, or when an entire process (possibly, many threads) ends while it is an owner of or a waiter for a shared memory mutex or condition variable, there may be waiters for the mutex or associated condition variable that will get hung up if the application does not perform the necessary cleanup. To prevent hangs, kernel cleanup processing for all ending threads and processes is enhanced to perform cleanup for shared memory condition variables and mutexes, if necessary. If a thread ends (normally or abnormally) while it is a waiter for a shared mutex or condition variable, it is removed from the waiter list. If the thread ends abnormally, further cleanup is performed to ensure that the thread is not the owner of a shared mutex. This extra cleanup is not performed by the kernel for a normal termination, because Language Environment thread cleanup unlocks any mutexes that are held by a normally terminated thread.

    In the case of an abnormally terminating thread, Language Environment thread cleanup may also unlock any held mutexes, if it gets a chance to run. If Language Environment cleanup does not unlock the held mutexes, kernel abnormal task termination will detect that the ending thread still owns a mutex. All of the waiters for a held mutex and the associated condition variable will be awoken. The mutex or condvar operation will then either abnormally terminate (X'EC6-xxxx8040' abend) or return an EINVAL, with the mutex and associated condition variable marked unusable. The object is marked unusable because the state of the data that the mutex or condition variable is serializing is uncertain, because of the abnormal ending of the

thread that owned the resource. If BPX1SMC/BPX4SMC is abended while it is waiting for the unusable mutex or condvar, the EC6 abend is percolated to the caller of BPX1SMC/BPX4SMC. An unusable mutex or condition variable can only be destroyed.

16. No stub is provided in SYS1.CSSLIB to make the call successful. Instead, the method described in Appendix A must be used to invoke the service using the USS CSR table.

Related services

- "shmem_lock (BPX1SLK, BPX4SLK) — Shared memory lock service" on page 724

Characteristics and restrictions

None.

shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment

Function

The shmget function returns a shared memory ID that it either created or was allowed to access.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state; PSW key 2, 8, or 9
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1MGT):	31-bit
AMODE (BPX4MGT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1MGT,(Key,
              Shared_Memory_Size,
              Shared_Memory_Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4MGT with the same parameters. The Shared_Memory_Size parameter is a doubleword.

Parameters

Key

Supplied parameter

Type: Integer

Length:
Fullword

Identification for this shared memory segment. This is either a user defined value that serves as a lookup value to determine if the shared memory segment already exists, or the reserved value IPC_PRIVATE. (See "BPXYIPCP — Map interprocess communication permissions" on page 987. IPC_PRIVATE is sometimes used when a process does not want to share a memory segment because it wants to privately control access to it by other processes.)

Shared_Memory_Size
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

A fullword (doubleword) field that contains the number of bytes of shared memory that are required. If IPC_MEGA is specified, the value must be a multiple of megabytes, or the request is failed with EINVAL. If IPC_GIGA is specified, the value must be a multiple of gigabytes, or the request is failed with EINVAL. If the caller is running in AMODE 64, the requested number of bytes will be rounded up to the nearest megabyte multiple.

Shared_Memory_Flags
Supplied parameter

Type: Integer

Length:
Fullword

Valid values for this field include any combination of the following (additional bits cause an EINVAL):

IPC_CREAT
Creates a shared memory segment if the specified key does not already have an associated ID. IPC_CREAT is ignored when IPC_PRIVATE is specified.

IPC_EXCL
Causes the shmget function to fail if the key specified has an associated ID. IPC_EXCL is ignored when IPC_CREAT is not specified or when IPC_PRIVATE is specified.

IPC_MEGA
Allocates shared storage in megabyte multiples. Use IPC_MEGA to decrease ESQA storage utilization in support of shared memory segments.

IPC_GIGA
Allocates shared storage in gigabyte multiples. Use IPC_GIGA to decrease real storage utilization when running in AMODE 64.

IPC_BELOWBAR
For AMODE 64 callers, IPC_BELOWBAR forces the memory object to be allocated from below the 2-gigabyte address range. This allows 64-bit applications to share objects with 31-bit applications.

IPC_SHAREAS
Enables sharing of the same storage area from multiple processes in the same address space.

- For AMODE 31 callers, this flag is only supported when the IPC_MEGA flag is also specified; otherwise, this flag is ignored.
- For AMODE 64 callers, this flag is supported for all shared memory segments that are obtained above the 2G bar.

S_IRUSR
Permits the process that owns the memory segment to read it.

S_IWUSR
Permits the process that owns the memory segment to alter it.

S_IRGRP
Permits the group that is associated with the memory segment to read it.

S_IWGRP
Permits the group that is associated with the memory segment to alter it.

S_IROTH
Permits others to read the memory segment.

S_IWOTH
Permits others to alter the memory segment.

The values that begin with an "IPC_" prefix are defined in BPXYIPCP. They are mapped onto S_TYPE, which is in BPXYMODE. See “BPXYIPCP — Map interprocess communication permissions” on page 987 and “BPXYMODE — Map the mode constants of the file services” on page 996.

The values that begin with an "S_I" prefix are defined in BPXYMODE, and are a subset of the access permissions that apply to files.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmget service returns the shared memory identifier or, if it is unsuccessful, -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmget service stores the return code. The shmget service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The shmget service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One of the following conditions occurred: • The shared memory identifier does not exist for the Key parameter, and either the Shared_Memory-Size parameter is zero or it is greater than the system-imposed maximum. This system-imposed maximum is set with the IPCSHMMPAGES parameter in a BPXPRMxx parmlib member. You can use the **ipcs -x** shell command to view this value. • The shared memory identifier exists for the Key parameter, but the size of the segment that is associated with it is less than the Shared_Memory_Size parameter, and the Shared_Memory_Size parameter is not equal to 0. • The Shared_Memory_Flags include bits that are not supported by this function. The following reason codes can accompany the return code: JRShmBadSize and JRIpcBadFlags.
EACCES	One of the following conditions occurred: • A shared memory identifier exists for the Key parameter, but the operation permission, as specified by the low-order 9–bits of the Shared_Memory_Flags parameter, is not granted (the "S_" items). The following reason code can accompany the return code: JRIpcDenied. • The caller is running in PSW key 2 but has a TCB key other than 2. The following reason code can accompany the return code: JrKeyMismatch
EEXIST	A shared memory identifier exists for the Key parameter, and both IPC_CREAT and IPC_EXCL were specified. The following reason code can accompany the return code: JRIpcExists.
ENOENT	A shared memory identifier does not exist for the Key parameter, and IPC_CREAT was not specified. The following reason code can accompany the return code: JRIpcNoExists.
ENOMEM	A shared memory identifier and associated shared memory segment are to be created, but the amount of system storage would exceed the system-imposed limit. The system limit is set with the IPCSHMSPAGES parameter in a BPXPRMxx parmlib member. You can use the **ipcs -y** shell command to view this value, which is displayed under SPAGES. The following reason code can accompany the return code: JRShmMaxSpages.
ENOSPC	A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allocated shared memory identifiers system-wide would be exceeded. This system limit is set with the IPCSHMNIDS parameter in a BPXPRMxx parmlib member. You can use the **ipcs -x** shell command to view this value. You can use the **ipcrm** shell command to remove unused shared memory identifiers. The following reason code can accompany the return code: JRIpcMaxIDs.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shmget service stores the reason code. The shmget service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. When a shared memory segment has been created, subsequent shmget calls to find the existing shared memory segment must request a size that is less than or equal to the value that was specified when the shared memory segment was created.
2. As long as a task knows the shared memory segment ID, it may issue a shmat, shmctl or shmdt (shmget is not needed).
3. The shmget function returns the shared memory segment identifier that is associated with the Key parameter.
4. This function creates a data structure defined by SHMID_DS if one of the following is true:
   - The Key parameter is equal to IPC_PRIVATE.
   - The Key parameter does not already have a shared memory segment identifier associated with it, and IPC_CREAT is set.

   For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039.
5. Upon creation, the data structure that is associated with the new shared memory segment identifier is initialized as follows:
   - IPC_CUID and IPC_UID are set to the effective user ID of the calling process.
   - IPC_CGID and IPC_GID are set to the effective group ID of the calling process.
   - The low-order 9-bits of IPC_MODE are equal to the low-order 9-bits of the Shared_Memory_Flags parameter.
   - SHM_OTIME is set to 0 and SHM_CTIME is set to the current time.
   - The storage is initialized to nulls when the segment is created.
   - The storage is allocated in key 8.
6. The shared memory segment is removed from the system when shmctl RMID is processed, and when all users have detached (with the shmdt service) or terminated.
7. The first shmget request to define a shared memory segment determines whether the segment has the IPC_MEGA/IPC_GIGA attribute (on the IPC_MEGA/IPC_GIGA option of the Shared_Memory_Flags parameter). Subsequent shmget requests, which use existing shared memory segments, have no effect on the IPC_MEGA attribute that is defined by that segment. In other words, the IPC_MEGA/IPC_GIGA option takes effect only for the first shmget request, and is ignored for all subsequent requests.
8. Shared memory segments created with the IPC_MEGA/IPC_GIGA attribute show this bit in the S_MODE byte that is returned with the w_getipc request.
9. Above the bar, shared memory cannot be used in subspace mode.
10. The user address space storage for a shared memory segment is normally obtained in storage key 8, except under the following special circumstances:
    - The caller of BPX1MGT that initially creates a shared memory segment is running in PSW key 2 or 9 *and either* of the following is true:

- The caller is running in AMODE 64 and the shared memory segment is neither of type IPC_MEGA nor type IPC_BELOWBAR.
- The caller is running in AMODE 31 and the shared memory segment is of type IPC_MEGA.

Under these circumstances, the user address space storage is obtained in the key of the caller (either key 2 or key 9). Any subsequent use of the segment from any address space will cause the user address space storage to be obtained in the same storage key in which the segment was initially created. This is true regardless of the PSW key in which the caller is running at the time of any subsequent attach request.

Related services

- "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890
- "shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment" on page 714
- "shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations" on page 718
- "shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment" on page 722

Characteristics and restrictions

- There is a maximum number of shared memory segments allowed in the system.
- There is a system-imposed limit on the maximum segment size that is defined in the BPXPRMxx parmlib member.
- The caller of the shmget service is restricted by ownership and read and read-write permissions that are defined by shmget and shmctl IPC_SET.

Examples

For an example using this callable service, see "BPX1MGT (shmget) example" on page 1161.

shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection

Function

The shutdown callable service shuts down all or part of a duplex socket connection.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1SHT):	31-bit task or SRB mode
AMODE (BPX4SHT):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SHT,(Socket_descriptor,
              How,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SHT with the same parameters.

Parameters

Socket_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the shutdown is to be done.

How
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the condition of the shutdown:

- 0 ends communication from Socket (Read).
- 1 ends communication to Socket (Write).
- 2 ends communication both to and from Socket.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shutdown service returns one of the following:

- 0 if the request is successful.
- -1 if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shutdown service stores the return code. The shutdown service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The shutdown service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was supplied. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
EINVAL	The How parameter is incorrect. It is not 0, 1, or 2. The following reason code can accompany the return code: JRBadEntryCode.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the shutdown service stores the reason code. The shutdown service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. A shutdown for read means that future write operations from the other end of this socket are rejected. Any data that was already written before the shutdown occurred are available for the application that issued the shutdown to read. The data is read until a read is done that returns zero bytes, indicating that there is no more data for that socket.
3. A shutdown for write means that any future writes by the application that issued the shutdown request are rejected.
4. Regardless of the How option specified, reads are not rejected.

Characteristics and restrictions

There are no restrictions on the use of the shutdown service.

Examples

For an example using this callable service, see "BPX1SHT (shutdown) example" on page 1189.

sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action

Function

The sigaction callable service examines, changes, or both examines and changes the action that is associated with a specific signal for all threads in the process.

Note: The signal handlers, a set of additional signals to be masked, and flags that are specified by the sigaction service are shared by all threads within a process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SIA):	31-bit
AMODE (BPX4SIA):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SIA,(Signal,
              New_sa_handler_address,
              New_sa_mask,
              New_sa_flags,
              Old_sa_handler_address,
              Old_sa_mask,
              Old_sa_flags,
              User_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SIA with the same parameters. All parameter addresses and addresses in parameter structures are doublewords. The User_data parameter is a doubleword field.

Parameters

Signal
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of the signal to examine, set, or both set and examine the action for.

New_sa_handler_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains either zero or the address of a fullword that contains the new signal action.

- If it contains zero, no new action is set for this signal.
- If it is not zero, set the signal action using the options that are described in this topic and in the BPXYSIGH macro. See "BPXYSIGH — Signal constants" on page 1039.

Constant	Description
SIG_DFL#	Take the default action for this signal.
SIG_IGN#	Ignore this signal.
Address	Address of the signal catcher function.

New_sa_mask
Supplied parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte area that contains a 64-bit mask of signals that are to be blocked during execution of the signal-catching function. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. Bits that are set to 1 represent signals that are blocked.

You must always provide this field, even though New_sa_mask is not used when the New_sa_handler_address parameter contains a `0`.

New_sa_flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of the fullword that contains the value of the signal action flags.

You must always provide this field, even though New_sa_flags is not used when the New_sa_handler_address parameter contains a `0`.

New_sa_flags can be set to the following constants defined in the BPXYSIGH macro:

Constant	Description
SA_FLAGS_DFT#	None of the following functions.
SA_NOCLDSTOP#	Do not generate **SIGCHLD** signals to the calling process when its children stop or are continued. (This is used only when Signal is set to **SIGCHLD**).
SA_OLD_STYLE#	The PPSDOLDSTYLE flag is set. This is provided for the C compiler runtime library to implement old-style signal callable service functions. The C compiler runtime library's signal interface routine is responsible for checking PPSDOLDSTYLE and setting sigaction to default action where needed.
SA_ONSTACK#	The PPSDONSTACK flag is set. This is provided for the caller to implement alternate stack signal delivery processing.

Constant	Description
SA_RESETHAND#	The PPSDRESETHAND flag is set. This is provided for the caller to reset the signal action to SIG_DFL# on entry to the signal catcher.
SA_RESTART#	The PPSDRESTART flag is set. This is provided for the caller to implement restart for functions that normally would receive an EINTR if a signal is delivered.
SA_SIGINFO#	The PPSDSIGINF flag is set. This is provided for the caller to provide additional information to the signal catcher, namely additional signal information and user context information.
SA_NOCLDWAIT#	Do not create zombie processes when children of the calling process exit (used only when Signal is set to **SIGCHLD**).
SA_NODEFER	The PPSDNODEFER flag is set. This is provided for the caller to not automatically block the Signal when the signal catcher is invoked.
SA_IGNORE	The value of the new_sa_handler is saved and returned on subsequent calls, but the signal is ignored.

Old_sa_handler_address
Parameter supplied and returned

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains either zero or the address of a fullword in which the system returns the old (current) signal action. If Old_sa_handler_address is specified as `0`, the old signal action, Old_sa_mask, and Old_sa_flags, are not returned.

Old_sa_mask
Returned parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte area where the old (current) value of the 64-bit mask of signals blocked during execution of the signal-catching function is returned. Bits that are set to `1` represent signals that are blocked.

You must provide this parameter, although Old_sa_mask is not returned when Old_sa_handler_address contains `0`.

Old_sa_flags
Returned parameter

Type: Structure

Length:
Fullword

The name of the fullword in which the old (current) signal action flags are returned.

You must always provide this field, even though Old_sa_flags is not returned when Old_sa_handler_address contains `0`.

User_data
Supplied parameter

Type: Character

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains 4 bytes of user-supplied data that is passed to the signal interface routine when the signal is delivered.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigaction service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigaction service stores the return code. The sigaction service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The sigaction service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The specified signal value is incorrect or is an unsupported signal number; an attempt was made to catch a signal that cannot be caught; or an attempt was made to ignore a signal that cannot be ignored. The following reason codes can accompany the return code: JRInvalidSignal and JRInvalidSigact.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigaction service stores the reason code. The sigaction service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If New_sa_handler_address value is set to the action SIG_DFL for a signal that cannot be caught or ignored, the sigaction request is ignored and Return_value is set to `0`.
2. Setting a signal action to ignore for a signal that is pending causes the pending signal to be discarded.

3. Setting signal action SIG_IGN or catch for signals **SIGSTOP**, **SIGTHSTOP**, **SIGTHCONT**, or **SIGKILL** is not allowed.
4. Setting signal action SIG_IGN for **SIGIO** is not allowed.
5. The SA_NOCLDWAIT flag should not be used with the waitpid (BPX1WAT, BPX4WAT) WNOHANG flag, or with the **SIGSTOP** or **SIGCONT** signals. Because the SA_NOCLDWAIT flag indicates that child processes of the calling process do not become zombies, these child processes do not report their status to the calling process when they end. Thus, if the calling process subsequently issues a waitpid call, it suspends until all of its child processes terminate, and then receives an ECHILD error return. This is expected behavior when SA_NOCLDWAIT is set. However, using **SIGSTOP** or **SIGCONT** signals with a child process could cause stop status to be reported to the calling process if it issues a subsequent waitpid call. Likewise, the use of the WNOHANG flag on a subsequent waitpid would result in an immediate return, instead of the process suspending until all child processes terminate. For these reasons, care should be taken not to mix these incompatible functions.
6. Setting signal action SIG_IGN for **SIGCHLD** has the same effect as setting the SA_NOCLDWAIT flag.
7. The SA_NOCLDSTOP and SA_NOCLDWAIT flags, despite having similar names, result in different types of actions. SA_NOCLDSTOP results in **SIGCHLD** signals not being sent when child processes stop or are continued. Setting SA_NOCLDWAIT results in child processes not becoming zombies or reporting their exit status, but **SIGCHLD** signals are still sent when child processes end.
8. The user data is delivered on a per-signal basis for the specific signal that is specified on this invocation. This field must be respecified if user data is desired for the next signal. This user data is set even if the action is SIG_DFT or SIG_IGN.
9. Although the user can be in supervisor state and any PSW key when this service is used, the kernel does not deliver signals to the signal interface routine until the task is running with a PSW key equal to the PSW key when the first callable service was entered and the process was created. See “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421.
10. The sigaction caller's thread must be registered for signals. You can register the thread by calling mvssigsetup, or, after signals are set up, by creating the thread with pthread_create. If the thread is not registered for signals, the sigaction service fails with a return code of EINVAL and a reason code of JRNotSigSetup. See “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421.
11. Constants that are used for this callable service are defined in the BPXYSIGH macro. See “BPXYSIGH — Signal constants” on page 1039.

Related services

- “exec (BPX1EXC, BPX4EXC) — Run a program” on page 132
- “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304
- “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421
- “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask” on page 757
- “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763

Characteristics and restrictions

In a multithreaded process, the new signal action that is set by the sigaction service changes the signal action for all threads in the process.

Examples

For an example using this callable service, see "BPX1SIA (sigaction) example" on page 1189.

__sigactionset (BPX1SA2, BPX4SA2) — Examine or change a set of signal actions

Function

The __sigactionset callable service examines, changes, or both examines and changes the actions that are associated with a set of signals.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SA2):	31-bit
AMODE (BPX4SA2):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1SA2,(New_count,
               New_structure,
               Old_count,
               Old_structure,
               SsetOption_flags,
               Return_value,
               Return_code,
               Reason_code)
```

AMODE 64 callers use BPX4SA2 with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

New_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of array elements in New_structure. New_count must be in the range from 0 to 64. If New_structure is not provided, specify a count of 0.

New_structure
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address that points to the beginning of the new structure. The new structure contains the layout of the desired parameters for sigaction: New_sa_handler_address, New_sa_flags, New_sa_mask, UserData, and ConsolMask. ConsolMask is a bit mask that defines all the signals that are to have the same action. New_sa_handler_address, New_sa_flags, New_sa_mask, and UserData are mapped by the BPXYSSET macro. See "BPXYSSET — Map the sigaction set" on page 1055.

Old_count
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of array elements that are allowed within Old_structure on input. On output, Old_count contains the number of array elements that have been stored. If Old_count is too small to hold the number of array elements that are needed, return code ENOMEM is returned. When ENOMEM is returned, Old_count contains the number of array elements that are required to contain the current signal action state.

Old_count must be in the range from 0 to 64.

If Old_structure is not provided, specify a count of 0.

You may not pass a constant in Old_count. If a constant is passed, an EFAULT is generated when an attempt is made to store back the result on exit.

Old_structure
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address that points to the beginning of the old structure. On output from the call to __sigactionset, Old_structure contains the number of signal actions specified in Old_count.

SsetOption_flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of the area in which the option flags are set. A leftmost bit (Sset_IgInvalid) set to 1 indicates signals that are not valid; signals that are not valid are to be ignored. Possible SsetOption_Flags: Sset_IgInvalid = X'80000000', which indicates that invalid signals and sigactions are to be ignored.

In the following example, Sset_IgInvalid is set to 1 and New_count is passed in as 3. New_structure has been given an address that points to the storage area that contains the five fields shown: ConsolidatedMask, New_sa_flags, New_sa_mask, and New_user_data.

NewStruct

NewCount	ConsolidatedMask (64 Bits)	New SaFlags (4 Bytes)	New SaHandler (4 Bytes)	New SaMask (64 Bits)	New UserData (4 Bytes)
1 →	X'0FFFFFFFFFFFFFFF'	0	0	0	0
2 →	X'3000000000000000	0	1	0	0
3 →	X'C000000000000000'	0	0480E000	'01..10'B	0

Note:

1. New_count can range from 1 to the maximum number of signals.
2. The signal handlers (a set of additional signals to be masked), option flags, and user data that is specified by the __sigactionset service, are shared by all threads within a process.
3. In the example shown:
   - The first set defines the action for signals 5–64 to their default state. Because some of these signals are unsupported, the setting of SsetOption_flags (Sset_IgInvalid) tells __sigactionset to ignore unsupported signals.
   - The second set tells __sigactionset to ignore signals 3 (SIGABRT#) and 4 (SIGILL#).
   - The third set defines a signal catcher with a mask for signals 1 (SIGHUP#) and 2 (SIGINT#).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __sigactionset service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __sigactionset service stores the return code. The __sigactionset service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __sigactionset service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The specified signal is incorrect or is an unsupported signal number; an attempt was made to catch or ignore a signal that cannot be caught or ignored; or the specified signal value was not within the range from 0 to 64. The following reason codes can accompany the return code: JRInvalidSignal, JRInvalidSigact, and JRInvalidRange. The following are examples of incorrect scenarios: • Sset_IgInvalid is set to 0, and any bit position ranging from 33 to 64 is on. JRInvalidSignal is returned. • Sset_IgInvalid is set to 0, and signal 7 (SIGSTOP#), signal 34 (SIGTHSTOP#), signal 35 (SIGTHCONT#), or signal 9 (SIGKILL#) is on. JRInvalidSigact is returned.
ENOMEM	There is not enough memory available to hold the number of array elements required to contain the current signal action state. The following reason codes can accompany the return code: JRSsetTooSmall.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __sigactionset service stores the reason code. The __sigactionset service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. For full details about the sigaction() parameters, see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746.
2. In a multithreaded process, the new signal action that is set by the __sigactionset service changes the signal action for all threads in the process.
3. If multiple masks have a bit set on for the same signal, the one that is set is the last one.
4. If the caller of __sigactionset does not specify Sset_IgInvalid within SsetOption_flags, a return code of EINVAL is returned for all signals and sigactions that are not valid. You can bypass this error by setting Sset_IgInvalid to 1.
5. If New_count is zero (indicating a query of old signal actions), no changes are made to the signal actions.
6. If Old_count is zero, the __signactionset service does not return anything in Old_structure.

Related services

- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered" on page 763

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1SA2 (__sigactionset) example" on page 1184.

sigpending (BPX1SIP, BPX4SIP) — Examine pending signals

Function

The sigpending service returns the union of the set of signals that are pending on the thread and the set of signals that are pending on the process.

Pending signals at the process level are moved to the thread that called the sigpending service.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SIP):	31-bit
AMODE (BPX4SIP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SIP,(Signal_pending_mask,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SIP with the same parameters.

Parameters

Signal_pending_mask
Returned parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte area to which the sigpending service returns a 64-bit signal pending mask. Bits that are set on represent signals that are pending and blocked. Each bit that is set to on represents a signal that is currently pending at the process level or the thread-level and is blocked by the current thread's signal mask. The leftmost bit represents signal 1, and the rightmost bit represents signal 64.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigpending service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigpending service stores the return code. The sigpending service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigpending service stores the reason code. The sigpending service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

For an example using this callable service, see "BPX1SIP (sigpending) example" on page 1190.

sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask

Function

The sigprocmask callable service examines, changes, or both examines and changes the calling thread's signal mask.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SPM):	31-bit
AMODE (BPX4SPM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPM,(How,
              New_signal_mask,
              Old_signal_mask,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPM with the same parameters.

Parameters

How

Supplied parameter

Type: Structure

Length:

Fullword

The name of a fullword that contains a numeric value that identifies the action that is to be taken on the thread's signal mask. The following constants, which are defined in BPXYSIGH, define the actions to be taken: See “BPXYSIGH — Signal constants” on page 1039.

Constant	Description
SIG_BLOCK#	Add the signals in New_signal_mask to those to be blocked for this thread.
SIG_UNBLOCK#	Delete the signals in New_signal_mask from those blocked for this thread.
SIG_SETMASK#	Replace the thread's signal mask with New_signal_mask.

New_signal_mask
Supplied parameter

Type: Address

Length:
Fullword

The name of a fullword that contains either 0 or the address of an 8-byte area that contains the 64-bit signal mask. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. The New_signal_mask parameter is applied to the thread's signal mask as specified by the How parameter. Mask bits that are set on represent signals that are blocked. If zero, the signal mask is not changed, and the How parameter is ignored.

Old_signal_mask
Parameter supplied and returned

Type: Address

Length:
Fullword

The name of a fullword that contains either 0 or the address of an 8-byte signal-mask return area. The service stores in this area the signal mask that was in effect, showing the signals that were blocked. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. Mask bits set on represent signals that are blocked. A zero indicates that no signal mask was returned.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigprocmask service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigprocmask service stores the return code. The sigprocmask service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The sigprocmask service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The specified address for New_signal_mask or Old_signal_mask was incorrect.
EINVAL	The value of the How parameter is not one of the allowable values.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigprocmask service stores the reason code. The sigprocmask service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The sigprocmask service examines, changes, or both examines and changes the signal mask for the calling thread. This mask is called the thread's signal mask. If there are any pending unblocked signals, either at the process level or at the current thread's level, after changing the signal mask, at least one of the signals is delivered to the thread before the sigprocmask service returns.
2. In a multithreaded process, the sigprocmask service is used to control to which thread in the process a signal that is generated by the kill service is delivered. For example, if two threads in a process have **SIGUSR1** signals blocked and one thread does not, the **SIGUSR1** signal that is generated by the kill service from another process is delivered to the thread that does not have the signal blocked.
3. You cannot block the **SIGKILL**, **SIGSTOP**, **SIGTHSTOP**, and **SIGTHCONT** signals. If you call the sigprocmask service with a request that would block those signals, that part of your request is ignored and no error is indicated.
4. A request to block signals that are not supported is accepted, and a return value of zero is returned.
5. All pending unblocked signals are moved from the process level to the current thread.
6. See Appendix I, “Optimizing performance using process- and thread-level information,” on page 1329.

Related services

- “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304
- “mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals” on page 421
- “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action” on page 746
- “sigpending (BPX1SIP, BPX4SIP) — Examine pending signals” on page 755
- “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763

Characteristics and restrictions

See Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1313.

Examples

For an example using this callable service, see “BPX1SPM (sigprocmask) example” on page 1194.

sigqueue (BPX1SGQ, BPX4SGQ) — Queue a signal to a process

Function

The sigqueue callable service queues a signal to a process, a process group, or all processes in the system to which the caller has permission to queue a signal.

CAUTION:
Note that when a caller with appropriate privileges (see "Authorization" on page 8) specifies a process ID equal to -1, the signal will normally be queued to all processes in the system, excluding the INIT process (process ID 1). If the signal action is to terminate the process, all processes will terminate. This may not be the desired action, considering that some processes may be necessary for the continued operation of the system.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SGQ):	31-bit
AMODE (BPX4SGQ):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SGQ,(Process_ID,
              Signal,
              Signal_Value,
              Signal_Options,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SGQ with the same parameters. Signal_Value is a doubleword field.

Parameters

Process_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword whose value specifies the process or processes to which the caller wants to queue a signal:

- If Process_ID is greater than 0, it is assumed to be a process ID. The signal is queued to the process with that process ID.

- If Process_ID is equal to 0, the signal is queued to all processes that have a process group ID equal to that of the caller, and for which the caller has permission to queue the signal.
- If Process_ID is -1, the signal is queued to all processes for which the caller has permission to queue the signal.
- If Process_ID is less than -1, its absolute value is assumed to be a process group ID. The signal is queued to all processes that have a process group ID equal to this absolute value, and for which the caller has permission to queue a signal.

Note the restrictions in "Characteristics and restrictions" on page 763.

Signal
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the signal number that is to be queued to the processes indicated by the Process_ID parameter. The signal number must be defined in the BPXYSIGH macro, or it must be 0. See "BPXYSIGH — Signal constants" on page 1039.

If the signal is 0, error checking takes place, but no signal is queued. Use a signal value of 0 to verify that the Process_ID parameter is correct before actually queuing a signal. This method does not verify permission to queue the signal to the specified Process_ID.

Signal_Value
Supplied parameter

Type: Integer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains data to be delivered with the signal.

Signal_Options
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the binary flags that describe how the signal is to be handled by the system and the user-supplied signal interface routine (SIR). This byte of user information is passed to the SIR in a data structure mapped by the PBXYPPSD macro. See "BPXYPPSD — Map signal delivery data" on page 1014.

Signal options are mapped as follows:

First 2 bytes
User-defined bytes to be delivered with the signal to the SIR in the signal information control block. These bytes are mapped by PPSDKILDATA.

Last 2 bytes
Flag bits, mapped by PPSDKILOPTS, that are defined as follows:

- First bit - signal to bypass Ptrace processing
- Second bit - reserved
- Third bit - the signal code specified in the first 2 bytes is set by the application
- Remaining bits - reserved

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigqueue service returns 0 if it has permission to queue the specified signal to any of the processes specified by the Process_ID parameter. A return value of 0 means that a signal was queued (or could have been queued, if the signal value was 0) to at least one of the specified processes.

If a signal is not queued, the return value is -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigqueue service stores the return code. The sigqueue service stores a return code only if the return value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The sigqueue service may return one of the following values in the Return_code parameter:

Return code	Explanation
EAGAIN	The caller has reached the maximum number of queued signals (MAXQUEUEDSIGS) allowed in a process.
EINVAL	The value specified in the Signal parameter is incorrect, or not the number of a supported signal.
EMVSSAF2ERR	The SAF ck_process_owner (IRRSKO00) callable service returned with an unexpected error.
EPERM	The caller does not have permission to queue the signal to any process specified in the Process_ID parameter.
ESRCH	No processes or process groups corresponding to the value specified in the Process_ID parameter were found.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigqueue service stores the reason code. The sigqueue service stores a reason code only when the return value is -1. The reason code further qualifies the return code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

In the case of EMVSSAF2ERR, the reason code contains the security product return and reason codes, respectively, in the two low-order bytes. For a more detailed description of the security product Check Privilege service return and reason code values, see the following table:

Security product return code	Security product reason code	Explanation
8	4	The caller is not the owner of the target process.
8	12	There was an internal error during security product processing.

Related services

- "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702

Characteristics and restrictions

1. A caller can queue a signal if the real or effective user ID of the caller is the same as the real or saved set user ID of the intended recipient. A caller can also queue signals if it has appropriate privileges (see "Authorization" on page 8).
2. Regardless of its user ID, a caller can always queue a SIGCONT signal to a process that is a member of the same session.
3. A caller can queue a signal to itself. If the signal is not blocked, at least one pending unblocked signal is delivered to the caller before the service returns control. Provided that no other unblocked signals are pending, the signal that is delivered is the signal that is queued. See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313 for more information.
4. Note the caution at the beginning of this callable service description.

Examples

For an example using this callable service, see "BPX1SGQ (sigqueue) example" on page 1187.

sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered

Function

The sigsuspend callable service replaces a thread's current signal mask with a new signal mask. It then suspends the caller's thread until delivery of a signal whose action is either to process a signal-catching service or to end the thread.

Requirements

Operation	Environment
Authorization:	Problem Program or Supervisor State, PSW key when the process was created (not PSW key 0)

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SSU):	31-bit
AMODE (BPX4SSU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SSU,(Signal_mask,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SSU with the same parameter.

Parameters

Signal_mask
Supplied parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte area that contains a 64-bit signal mask that is set before waiting for a signal, and during the execution of any signal catcher. The leftmost bit represent signals 1 and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are blocked.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigsuspend service returns a -1 if it returns to its caller.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigsuspend service stores the return code. The sigsuspend service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The sigsuspend service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINTR	A signal was received and handled successfully.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigsuspend service stores the reason code. The sigsuspend service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The caller's thread starts running again when it receives one of the signals that are not blocked by the mask that is set by this call, or when a system failure occurs that sets Return_code to some value other than EINTR.
2. The signal mask represents a set of signals that are to be blocked. Blocked signals do not "wake up" the suspended service. The signals **SIGSTOP**, **SIGTHSTOP**, **SIGTHCONT**, and **SIGKILL** cannot be blocked or ignored; they are delivered to the program no matter what the signal mask specifies.
3. If the signal action is to end the thread, the sigsuspend service does not return.
4. If the signal interruption is to give control to the signal interface routine (SIR), which is defined by the mvssigsetup service, the SIR is given control with the following PPSD fields:

   PPSDSAMASK
   Set to the New_sa_mask value, which is set by the sigaction service, for the signal number that caused the interruption.

   PPSDCURRENTMASK
   The signal mask value that existed before the sigsuspend service was called.

   To be XPG4 compliant, this is the signal mask that is installed when a signal catcher performs a normal return.

   PPSDCATCHERMASK
   The signal mask that is specified on the sigsuspend service.

   To be XPG4 compliant, the signal mask that is installed before calling a signal catcher is calculated by taking the union of PPSCATCHERTMASK, PPSDSAMASK, and the signal that caused the interrupt.
5. The signal interface routine (SIR) that is defined by the mvssigsetup service is given control only when the PSW key of the sigsuspend caller is equal to the signal catcher key of the process. The signal catcher key is set to the PSW key of the caller of the first z/OS UNIX callable service that created the process.
6. If the caller has a PSW key that is different from the signal catcher key, or has a PSW key of zero, the sigsuspend service returns with a return code of EMVSERR and reason code of JRPSWKeyNotValid.
7. All pending unblocked signals are moved from the process level to the current thread.

Related services

- "pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal" on page 468
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746
- "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757

Characteristics and restrictions

See Appendix G, "The relationship of z/OS UNIX signals to callable services," on page 1313.

Examples

See "BPX1SSU (sigsuspend) example" on page 1198 for an example using this callable service.

sigtimedwait (BPX1STW, BPX4STW) — Wait for a signal with a specified timeout

Function

The sigtimedwait callable service suspends the invoking thread until either the specified timeout expires, or a signal specified in the signal set becomes pending, at either the process or the invoking thread. If a signal that is specified in the signal set is sent to the invoker of sigtimedwait, the value of that signal is returned to the invoker and the sigtimedwait service ends.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, PSW key when the process was created (not PSW key 0)
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STW):	31-bit
AMODE (BPX4STW):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STW,(Signal_mask,
              Siginfo_ptr
              Siginfo_len,
              Seconds,
              Nanoseconds,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STW with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Signal_mask
Supplied parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte field area that contains a 64-bit signal mask that contains the set of signals that this task is to wait on. The leftmost bit represents signal 1, and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are waited on.

Siginfo_ptr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of user-supplied storage that is mapped by the BPXYSINF macro (see "BPXYSINF — Map SIGINFO_T structure" on page 1042). If this address is nonzero, the sigtimedwait service uses this area to place additional signal information when a signal number is returned in Return_value. If this address is zero, or if an error is returned, no additional information is returned in this area.

Siginfo_len
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the user-supplied storage that is mapped by the BPXYSINF macro. If the address of Siginfo_ptr is zero, this parameter is ignored.

Seconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer representing the maximum number of seconds that the calling program is willing to wait for one of the specified signals to become pending.

Note:

1. Seconds can be any value greater than or equal to `0` and less than or equal to 4 294 967 295. The value specified for Seconds must be an unsigned integer.
2. The values in the Seconds and Nanoseconds parameters are combined to determine the timeout value. A combined value of zero indicates that the

sigtimedwait service does not wait at all. A value of SIG#NO_TIMEOUT (see "BPXYSIGH — Signal constants" on page 1039) indicates that no timeout value is set.

Nanoseconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an unsigned integer representing the number of nanoseconds to be added to the value specified by the Seconds parameter. Nanoseconds can be any value greater than or equal to `0` and less than or equal to `1 000 000 000`.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigtimedwait service returns the signal if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigtimedwait service stores the return code. The sigtimedwait service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The sigtimedwait service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	One or more of the parameters that were passed to this service are in error. The following reason codes unique to the sigtimedwait service can accompany the return code: JRNanoSecondsTooBig, JRInvalidSignal.
EAGAIN	The service timed out before any of the specified signals became pending on the invoking thread.
EINTR	The service received a signal that was not specified in the input signal mask.

Reason_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigtimedwait service stores the reason code. The sigtimedwait service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The sigtimedwait service behaves in the same way as the sigwait service, except that with the sigtimedwait service you can specify a timeout value.
2. If a nonzero address is specified for the siginfo_ptr parameter, the sigtimedwait service also returns si_signo, si_code, and si_value, as mapped by BPXYSINF.
3. A timeout value of zero (Seconds + Nanoseconds) means that the sigtimedwait service does not wait at all. It checks for pending signals, and if no signal is found, it returns with an error of EAGAIN. If a signal is found, the service returns with the signal number of the pending signal.
4. A passed timeout value of SIG#NO_TIMEOUT (see "BPXYSIGH — Signal constants" on page 1039) indicates that no timeout value is set. The sigtimedwait service waits until a signal becomes pending.

Related services

- "sigwait (BPX1SWT, BPX4SWT) — Wait for a signal"
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757

Characteristics and restrictions

There are no restrictions on the use of the sigtimedwait service.

Examples

For an example using this callable service, see "BPX1STW (sigtimedwait) example" on page 1201.

sigwait (BPX1SWT, BPX4SWT) — Wait for a signal

Function

The sigwait callable service waits for an asynchronous signal. If a signal that is specified in the signal set is sent to the invoker of sigwait, the value of that signal is returned to the invoker and the sigwait service ends.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SWT):	31-bit
AMODE (BPX4SWT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SWT,(Signal_mask,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SWT with the same parameters.

Parameters

Signal_mask
Supplied parameter

Type: Structure

Length:
8 bytes

The name of an 8-byte field area that contains a 64-bit signal mask that contains the set of signals that this task is to wait on. The leftmost bit represents signal 1, and the rightmost bit represents signal 64. Bits that are set to 1 represent signals that are waited on.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigwait service returns the signal if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigwait service stores the return code. The sigwait service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The sigwait service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The Signal_mask argument contained a signal that represents an incorrect signal number. The following reason code can accompany this return code: JRInvalidSignal.

Reason_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sigwait service stores the reason code. The sigwait service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If any signals that are specified in Signal_mask are pending upon invocation of the sigwait service, one of those signals has its value returned to the invoker, and that signal is cleared from the set of pending signals.
2. If there are no pending signals that were specified in Signal_mask, the sigwait service waits until a signal that is specified in Signal_mask is generated. A signal mask of zero causes the caller to wait until the task or process is terminated.
3. If sigwait is invoked for a **SIGKILL**, **SIGSTOP**, or **SIGTHSTOP** signal, and a **SIGKILL**, **SIGSTOP**, or **SIGTHSTOP** signal arrives, the value of the signal is not returned to the invoker. Rather, the **SIGKILL**, **SIGSTOP**, or **SIGTHSTOP** action occurs.
4. The current sigaction ("sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746) that is associated with a signal that is returned is not performed. This action also remains unchanged by the use of the sigwait service.
5. If there are multiple threads in a process that have issued a sigwait for the same signal, exactly one of these threads returns from sigwait with the signal number if the signal was directed at the process.

Related services

- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757

Characteristics and restrictions

There are no restrictions on the use of the sigwait service.

Examples

For an example using this callable service, see "BPX1SWT (sigwait) example" on page 1201.

sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time

Function

The sleep callable service suspends running of the calling thread (process) until the number of seconds specified by the parameter Seconds has elapsed, or until a signal is delivered to the calling thread to invoke a signal-catching function or end the thread.

Requirements

Operation	**Environment**
Authorization:	Problem Program or Supervisor State, PSW key when the process was created (not PSW key 0)

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SLP):	31-bit
AMODE (BPX4SLP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SLP,(Seconds,
              Return_value)
```

AMODE 64 callers use BPX4SLP with the same parameters.

Parameters

Seconds
Supplied parameter

Type: Integer

Length:
Fullword

The name of an unsigned fullword that contains the number of seconds for the calling thread to sleep. Because of processor delays, the calling thread can sleep slightly longer than this specified time.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of an unsigned fullword in which the sleep service returns the "remaining sleep time" value: the difference between Seconds and the number of seconds that elapsed before the thread was awakened. The return value is rounded to the nearest second. (If the thread was awakened by the ending of the elapsed time specified by Seconds, the return value is `0`.) When a signal arrives and the remaining time left in the sleep is less than a half second, a value of `0` is returned.

Usage notes

1. The suspension can actually be longer than the requested time, because of the scheduling of other activity by the system.
2. The sleep service suspends the thread that is running for a specified number of seconds, or until a signal is delivered to the calling thread that invokes a signal-catching function or ends the thread. An unblocked signal that is received during this time prematurely "wakes up" the thread. The appropriate

signal-handling function is invoked to handle the signal. When that signal-handling function returns, sleep returns immediately, even if there is "sleep time" remaining.

3. The sleep service returns a zero if it slept for the number of seconds that were specified. If the time that was specified by the Seconds parameter has not elapsed when the sleep service is interrupted because of the delivery of a signal, the sleep service returns the unslept amount of time (the requested time minus the time actually slept when the signal was delivered) in seconds. Any time that is consumed by signal-catching functions is not reflected in the value that is returned by the sleep service.
4. The following usage notes are for a **SIGALRM** signal that is generated by the alarm, interval timer, or kill calls during the execution of the sleep call:
   - If the calling thread has **SIGALRM** blocked before it calls the sleep service, the sleep service does not return when **SIGALRM** is generated, and the **SIGALRM** signal is left pending when sleep returns.
   - If the calling process has **SIGALRM** ignored when the **SIGALRM** signal is generated, the sleep service does not return and the **SIGALRM** signal is ignored.
   - If the calling process has **SIGALRM** set to a signal-catching function, that function interrupts the sleep service and receives control. The sleep service returns any unslept amount of time, as it does for any other type of signal.
5. If a signal-catching function interrupts the sleep service and examines or changes the time a **SIGALRM** is scheduled to be generated, the action that is associated with the **SIGALRM** signal is the same as it is when the signal-catching function interrupts any other function.
6. If a signal-catching function interrupts the sleep service, restores a previously saved environment, and does not return, the action that is associated with the **SIGALRM** signal that was saved prior to the sleep service is the same as it is when the signal-catching function interrupts any other function.
7. When the sleep service returns, any previous alarm time that has not elapsed is restored before any signal-catcher gets control. Signal catchers can change this alarm setting. See “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29.
8. An EC6 abend is generated when the caller's PSW key or RB state prevents signals from being delivered.

Related services

- “alarm (BPX1ALR, BPX4ALR) — Set an alarm” on page 29
- “sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action” on page 746
- “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask” on page 757
- “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763

Characteristics and restrictions

See Appendix G, “The relationship of z/OS UNIX signals to callable services,” on page 1313.

Examples

For an example using this callable service, see “BPX1SLP (sleep) example” on page 1190.

MVS-related information

Both the alarm service and the sleep service use the MVS STIMERM macro. It is possible that two STIMERM SET requests can be set by the alarm and sleep services. If the task invokes both the STIMERM SET macro and the sleep service, the limit of concurrent STIMERM SET requests for a task can be exceeded, which results in an abnormal end.

smf_record (BPX1SMF, BPX4SMF) — Write an SMF record

Function

The smf_record callable service writes an SMF record to the SMF data set. The caller must be permitted to the BPX.SMF resource profile in the FACILITY class or must be APF-authorized.

The service can also be used to determine if a particular type or subtype of SMF record is being recorded. This avoids the overhead of data collection if the SMF record is not going to be recorded.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SMF):	31-bit
AMODE (BPX4SMF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SMF,(Smf_record_type,
              Smf_record_subtype,
              Smf_record_length,
              Smf_record_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SMF with the same parameters. The Smf_record_address parameter is a doubleword.

Parameters

Smf_record_type
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the SMF record type. See *z/OS MVS System Management Facilities (SMF)* for information on SMF record type and SMF record layout.

Smf_record_subtype
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the SMF record subtype. See *z/OS MVS System Management Facilities (SMF)* for information about SMF record type and SMF record layout.

Smf_record_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the SMF record length.

Smf_record_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the starting address of the SMF record to be written, or zero. If it contains zero, SMF is tested to determine if a particular record type or subtype is being recorded.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the smf_record service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the smf_record service stores the return code. The smf_record service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The smf_record service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	The value that was specified for an operand was incorrect. The following reason code can accompany the return code: JRSMFBadRecordLength.
ENOMEM	Not enough storage is available. The following reason code can accompany the return code: JRNoStorage.
EPERM	The calling process is not permitted to the BPX.SMF resource in the FACILITY class and the calling processes is not APF authorized. The following reason code can accompany the return code: JRSMFNotAuthorized.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the smf_record service stores the reason code. The smf_record service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

- To determine if a particular type or subtype is being recorded, specify the record address as zero.
  - If the return value is zero, the type or subtype is being recorded.
  - If the return value is -1 and the return code is EMVSERR with a reason code of JRSMFNotAccepting, SMF is not recording this type or subtype.
- When writing the SMF record, user exit IEFU84 is called if the exit is enabled. If the caller wants user exit IEFU83 to be called instead, set ThliEP_FunctionCode to ThliEP_SMFFlagSet and set the ThliEP_SMFIEFU83 bit to on. These fields are mapped in the THLI control block as described in "BPXYTHLI — Thread-level information" on page 1060. z/OS UNIX callable services cannot be issued by these user exits because they are running under this syscall and nested syscalls are not allowed.
- The THLIEP_FUNCTIONCODE and the THLIEP_MRSMFFLAGS fields may be cleared by any syscall and therefore values in these fields should not be expected to be retained across syscalls.

Related services

There are no related services.

Characteristics and restrictions

The caller must be permitted to the BPX.SMF resource profile in the FACILITY class.

Examples

For an example using this callable service, see "BPX1SMF (smf_record) example" on page 1191.

MVS-related information

1. See *z/OS MVS System Management Facilities (SMF)* for information on SMF record types and SMF record layout.
2. See Setting up the BPX.* FACILITY class profiles in *z/OS UNIX System Services Planning* for a description of the BPX.SMF FACILITY class profile and how it is created.

---

socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets

Function

The socket or socketpair callable service creates a socket or a pair of sockets for communication.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE (BPX1SOC):	31-bit task or SRB mode
AMODE (BPX4SOC):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SOC,(Domain,
              Type,
              Protocol,
              Dimension,
              Socket_vector,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SOC with the same parameters.

Parameters

Domain

Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the address domain requested. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the values that are defined for this field.

Type
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the type of socket that is to be created. Some of the socket types are:

Sock#_Stream
Provides sequenced, two-way byte streams that are reliable and connection-oriented. They support out-of-band data.

Sock#_Dgram
Provides datagrams, which are connectionless messages of a fixed maximum length whose reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

Sock#_Raw
Supports AF_INET and AF_INET6. You must be a superuser to use this type.

See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the values that are defined for this field.

Protocol
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the protocol requested. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the values that are defined for this field.

Dimension
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the number of sockets to be returned. The value of this field determines whether the service performed is socket or socketpair. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the values that are defined for this field.

Socket_vector
Supplied parameter

Type: Integer

Length:
Doubleword

The name of a doubleword field into which a socket descriptor or pair of socket descriptors is to be stored.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the socket or socketpair service returns one of the following:

- `0`, if the request is successful.
- `-1` if the request is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the socket or socketpair service stores the return code. The socket or socketpair service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The socket or socketpair service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Permission is denied.
EAFNOSUPPORT	The address family that was specified with the Domain parameter is not supported.
EAGAIN	The resource is temporarily unavailable. The following reason code can accompany the return code: JRTcpNotActive.
EINVAL	Dimension is not a valid value. Only 1 or 2 can be specified for this parameter. The following reason code can accompany the return code: JRInvalidParms.
EIO	There has been a network or transport failure. The following reason code can accompany the return code: JRPFSdead.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
EPROTONOSUPPORT	The Protocol parameter is incorrect. It is not 0. The following reason code can accompany the return code: JRSocketProtocolInvalid.
EPROTOTYPE	The socket type is not supported by the protocol.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the socket or socketpair service stores the reason code. The socket or socketpair service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

2. The socket callable service is invoked by specifying 1 (Sock#dim_socket) for the Dimension parameter.
3. The socketpair callable service is invoked by specifying 2 (Sock#dim_socketpair) for the dimension parameter. Socketpair returns 2 socket descriptors in the socket_vector parameter. The sockets are unnamed and connected.
4. These usage notes apply for IPv6 sockets:
   - An AF_INET6 socket may be opened only if there is at least one IPv6-capable stack active at the time of the call.
   - If a process has stack affinity under CINET, and that single stack is not IPv6-capable, the call will fail.
   - When an IPv6 socket is created through CINET, CINET creates AF_INET6 subsockets to IPv6 stacks and IPv4 sockets to IPv4 stacks.
5. Creating a socket with stack affinity:

   You can use SOCK#DIM_SOCKETWAFFINITY and SOCK#DIM_SOCKETPAIRWAFFINITY to create a socket or a socket pair with affinity to one specific stack under Common INET. The stack name is passed in the Socket_vector parameter, and this field is overlaid with the output socket descriptor or descriptors when the call is successful. The name is in upper case, left-justified, and padded with blanks. This is the same name that was specified on the SUBFILESYSTYPE NAME() statement that defined this stack in BPXPRMxx.

   If Common INET is not installed, the stack name is ignored. If Common INET is installed and the stack name does not match any stack configured under CINET, the call fails with a return code of EIBMBADTCPNAME.

   This affinity overrides any process-level stack affinity for this one socket only.

   Note to PFS implementers: This option is not available for the Master Socket opened as part of stack initialization.
6. UNIX domain sockets (AF_UNIX) do not support being called in SRB-mode.

Related services

None.

Characteristics and restrictions

There are no restrictions on the use of socket or socketpair.

Examples

For an example using this callable service, see "BPX1SOC (socket or socketpair) example" on page 1192.

spawn (BPX1SPN, BPX4SPN) — Spawn a process

Function

The spawn callable service combines the semantics of the fork and exec callable services to create a child process to run a specified z/OS UNIX executable file.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key, any TCB key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SPN):	31-bit
AMODE (BPX4SPN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SPN,(Pathname_length,
              Pathname,
              Argument_count,
              Argument_length_list,
              Argument_list,
              Environment_count,
              Environment_data_length,
              Environment_data_list,
              Filedesc_count,
              Filedesc_list,
              Inherit_area_len,
              Inherit_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SPN with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the path name of the file. The length of the path name can be up to 1023 bytes.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the fully qualified path name of the file that is to be run. Each component of the path name (directory name, subdirectory

name, or file name) can be up to 255 characters long. The complete path name can be up to 1023 characters long, and does not require an ending NUL character.

Path names can begin with or without a slash.

- If the path name begins with a slash, it is an absolute path name; the slash refers to the root directory, and the search for the file starts at the root directory.
- If the path name does not begin with a slash, it is a relative path name; the search for the file starts at the working directory.

Argument_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for the Argument_length_list and the Argument_list. If the program needs no arguments, specify `0`.

Argument_length_list
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each pointer in the list is the 31(64)-bit address of a fullword that gives the length of an argument that is to be passed to the specified program. If the program needs no arguments, define Argument_length_list as the name of a fullword (doubleword) that contains `0`.

If the target executable file arguments require null terminators, the arguments that are supplied to this service must include the null terminator as part of the data string and the length.

Argument_list
Supplied parameter

Type: Structure

Length:
Variable specified by Argument_length_list

The name of a list of 31(64)-bit pointers. Each pointer in the list is the 31(64)-bit address of a character string that is an argument to be passed to the specified program. Each argument is of the length that is specified by the corresponding element in the Argument_length_list. If the program needs no arguments, define Argument_list as the name of a fullword (doubleword) that contains `0`.

Environment_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of pointers in the lists for Environment_data_length and Environment_data_list. If the program needs no environment data, specify 0.

Environment_data_length
Supplied parameter

Type: Structure

Length:
Variable

The name of a list of 31(64)-bit pointers. Each 31(64)-bit pointer in the list is the 31(64)-bit address of a fullword that gives the length of an environment variable that is to be passed to the specified program. If the program does not use environment variables, specify 0.

Environment_data_list
Supplied parameter

Type: Structure

Length:
Variable, specified by Environment_data_length

The name of a list of 31(64)-bit pointers. Each 31(64)-bit pointer in the list is the 31(64)-bit address of a character string that is an environment variable to be passed to the specified program. Each environment variable is of the length that is specified by the corresponding element in Environment_data_length. If the program does not use environment variables, specify 0.

If the target executable file is an IBM Language Environment-enabled program, the environment variables that are supplied to this service must include the null terminator as part of the data string and length. Trailing blanks can cause the environment variable to be processed incorrectly. Each environment variable is searched for a null character; if a null character is found, the environment variable is truncated at that point.

Filedesc_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of file descriptors the child process is to inherit. It may take values from 0 to OPEN_MAX. If the value is 0, all file descriptors from the parent are inherited without remapping by the child, and the filedesc_list is ignored.

Filedesc_list
Supplied parameter

Type: Structure

Length:
Variable

The name of an array of fullword file descriptor remap values that indicate how the child's file descriptors are to be remapped from the caller's (parent's) file descriptors. Except for those file descriptors that are designated by SPAWN_FDCLOSED in the supplied array, each of the child's file descriptors in the range zero to Filedesc_count-1 inherits file descriptor remap values

filedesc_list(1) to filedesc_list(filedesc_count) from the supplied file descriptor array. The constant SPAWN_FDCLOSED is defined in the BPXYCONS macro.

As an example, assume that the caller supplies an array of 3 entries with the values 7, 5, and 4 respectively. This causes the child's file descriptor 0 to be remapped to the parent's file descriptor 7, the child's file descriptor 1 to be remapped to the parent's file descriptor 5, and the child's file descriptor 2 to be remapped to the parent's file descriptor 4.

Inherit_area_len
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the length of the inheritance structure that is to follow. If this parameter contains a value of zero, the Inherit_area parameter is ignored.

Inherit_area
Supplied parameter

Type: Structure

Length:
Specified by **Inherit_area_len**.

The name of a data area that contains the inheritance structure for the child process. See the BPXYINHE mapping for the details of the inheritance structure (“BPXYINHE — Spawn Inheritance Structure” on page 970).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the spawn service returns the process ID of the newly created child process, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the spawn service stores the return code. The spawn service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes*bbbbb for a complete list of possible return code values. The spawn service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The caller does not have appropriate permissions to run the specified file. It may lack permission to search a directory that is named in the Pathname parameter; it may lack execute permission for the file to be run; or the file to be run is not a regular file and the system cannot run files of its type.

Return_code	Explanation
EAGAIN	The resources that are required to let another process be created are not available now; or you have already reached the maximum number of processes or UIDs that you are allowed to create. This error is also generated if _BPX_USERID was specified, and the specified user name was not defined to SAF with an OMVS segment. The following reason codes can accompany the return code: JROK, JRMaxUIDs or JRWlmWonErr.
EBADF	An entry in the filedesc_list is not a valid file descriptor; or the controlling terminal file descriptor that was specified in the inheritance structure is not valid.
EINVAL	One or more of the following conditions were detected: • The user name that was specified on the _BPX_USERID environment variable has an incorrect length. • An attribute that was specified in the inheritance structure (BPXYINHE) is not valid or contains an unsupported value. • The version number that was specified for the inheritance structure (BPXYINHE) is not valid. See "BPXYINHE — Spawn Inheritance Structure" on page 970 for supported version numbers. • The inheritance structure length that was specified by the Inherit_area_len parameter or within the inheritance structure does not contain a length that is appropriate for the BPXYINHE version. See "BPXYINHE — Spawn Inheritance Structure" on page 970 for supported lengths. • The process group ID that was specified in the inheritance structure is less than zero or has some other unsupported value. The following reason codes can accompany the return code: JROK, JRUserNameLenError, JRJsRacXtr, JRInheUserid, JRInheRegion, JRInheCPUTime, JRInheAccountData, JRInheCWD, JRInheEye, JRInheSetPgrp, JRInheVersion, JRInheLength, and JrInheMemLimit.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Filename argument. This error is issued if more than 24 symbolic links are detected in the resolution of Filename.
EMVSERR	If EMVSERR is accompanied by reason code JrLocalSpawnNotAllowed, it means that the parameters specify that a local process must be created. It also means that one or more of the current address space's attributes is to be changed. A spawn request is not allowed to change certain attributes of the current address space. A spawn request is required to create a local process when either the environment variable _BPX_SHAREAS is set to MUST (_BPX_SHAREAS=MUST) or when the inheritance structure in the parameter list specifies InheMustBeLocal. When this reason code is given, the spawn request requires a local process to be created and one or both of the following conditions is also present: • The environment variables specified on the call include a value for _BPX_USERID that does not match the current user ID. • The inheritance structure passed on the call specifies one or more of the following flags: InheSetRegionSz, InheSetMemLimit, InheSetTimeLimit, InheSetAcctData, InheSetJobname.

Return_code	Explanation
EMVSSAF2ERR	The executable file is a set-user-ID or set-group-ID file and the file owner's UID or GID is not defined to the Security Access Facility (SAF).
ENAMETOOLONG	File_name is longer than 1023 characters, or some component of the file name is longer than 255 characters. Name truncation is not supported.
ENOENT	No file name was specified, or one or more of the components of the specified Filename were not found.
ENOEXEC	The specified file has execute permission, but it is not in the proper format to be a process image file. Reason_code contains the loader reason code for the error.
ENOMEM	The new process requires more memory than is permitted by the hardware or the operating system.
ENOTDIR	A directory component of Filename is not a directory.
ENOTTY	The tcsetpgrp failed for the specified controlling terminal file descriptor in the inheritance structure. The failure occurred because the calling process does not have a controlling terminal, or the specified file descriptor is not associated with the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
EPERM	The spawn failed for one of the following reasons: • The spawned process is not a process group leader. • The _BPX_USERID environment variable was specified, and the invoker does not have appropriate privileges (see "Authorization" on page 8) to change the MVS identity. • The invoker does not have the appropriate privileges to change one or more of the attributes specified in the inheritance structure (BPXYINHE). The following reason codes can accompany the return code: JROK, JRNoChangeIdentity, JRInheUserid, JRInheRegion, JRInheCPUTime, and JRInheCWD.
ESRCH	The specified process group ID in the inheritance structure is not that of a process group in the calling process's session.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the spawn service stores the reason code. The spawn service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For most of the reason codes, see *z/OS UNIX System Services Messages and Codes*. For the ENOEXEC Return_code, Reason_code contains the loader reason code for the error:

Reason code	Explanation
X'xxxx0C27'	The target z/OS UNIX file is not in the correct format to be an executable file.
X'xxxx0C31'	The target z/OS UNIX file is built at a level that is higher than that supported by the running system.

Usage notes

1. Aspects of spawn processing are controlled by environment variables. The environment variables that affect spawn processing are the ones that are passed into the spawn syscall, and not the environment variables of the calling process. The environment variables of the calling process do not affect spawn processing, unless they are the same as those that are passed in the Environment_data_list parameter.
2. The new process (called the *child process*) inherits the following attributes from the process that calls spawn (called the *parent process*):
   - Session membership
   - Real user ID
   - Real group ID
   - Supplementary group IDs
   - Priority
   - The region size of the parent is inherited by the child, unless the INHESETREGIONSZ flag in the inheritance structure is set on to indicate that the value specified in INHEREGIONSZ is to be used to determine the child's region size. For more information, see What are soft limits? in *z/OS UNIX System Services Planning* and Inheriting soft limits in *z/OS UNIX System Services Planning*.
   - The MEMLIMIT of the parent is inherited by the child, unless the INHESETMEMLIMIT flag in the inheritance structure is set on to indicate that the value specified in INHEMEMLIMIT is to be used to determine the child's MEMLIMIT.
   - The time limit of the parent is inherited by the child, unless the INHESETTIMELIMIT flag in the inheritance structure is set on to indicate that the value specified in INHETIMELIMIT is to be used to determine the child's time limit.
   - The accounting data of the parent is inherited by the child, unless the INHESETACCTDATA flag in the inheritance structure is set on to indicate that the data pointed to by INHEACCTDATAPTR is to be used to determine the child's accounting data.
   - The current working directory (CWD) of the parent is inherited by the child, unless the INHESETCWD flag in the inheritance structure is set on to indicate that the value pointed to by INHECWDPTR is to be used to determine the child's initial current working directory.
   - The root directory of the parent is inherited by the child.
   - The file creation mask of the parent is inherited by the child, unless the INHESETUMASK flag in the inheritance structure is set on to indicate that the value specified in INHEUMASK is to be used to determine the child's file creation mask.
   - The process group ID of the parent is inherited by the child, unless the INHESETGROUP flag in the inheritance structure is set on to indicate that the value specified in INHEPGROUP is to be used to determine the child's process group. If the value in INHEPGROUP is set to INHE#NEWPGROUP, the child is placed into a new process group with a process group ID set to the child's process ID. Otherwise, the child is placed into the process group that is represented by the value that is specified in INHEPGROUP.
   - Signals that are set to be ignored in the parent are set to be ignored in the child, unless the INHESETSIGDEF flag is on and the INHESIGDEF field specifies an overriding value in the supplied inheritance structure.

- The signal mask is inherited from the parent, unless the INHESETSIGMASK flag is set on in the inheritance structure and the INHESIGMASK field specifies an overriding value in the supplied inheritance structure.
- The user syscall trace setting is propagated to the child process.
- Security information from the parent's address space is propagated to the child's address space, and the child has a security environment equivalent to that of the parent, unless:
  - The _BPX_USERID environment variable specifies otherwise.
  - The INHESETUSERID flag in the inheritance structure is set on, and INHEUSERID contains a user ID.
- The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the current task are propagated to the child's address space, unless the STEPLIB environment variable specifies otherwise. This causes the child's address space to have the same MVS program search order as the calling parent task.
- The jobname of the parent is propagated to the child and appended with a numeric value in the range of 1–9, if the jobname is 7 characters or less. If the jobname is 8 characters, the jobname is propagated as is. When a jobname is appended with a numeric value, the count wraps back to 1 when it exceeds 9.
- If the calling parent task is in a WLM enclave, the child is joined to the same WLM enclave. This allows WLM to manage the parent and child as one "business unit of work" entity for system accounting and management purposes.

3. The new child process has the following differences from the parent process:
   - The child process has a unique process ID (PID) that does not match any active process group ID.
   - The child has a different parent process ID (namely, the process ID of the process that called spawn).
   - If the filedesc_count parameter specifies a 0 value, the child has its own copy of the parent's file descriptors, except for those files that are marked FCTLCLOEXEC or FCTLCLOFORK and for directories that were opened on a call to the opendir service. The files marked FCTLCLOEXEC or FCTLCLOFORK and open directories are not inherited by the child. If the filedesc_count parameter specifies a value greater than 0, the parent's file descriptors are remapped for the child as specified in the filedesc_list array. Those file descriptors from filedesc_count through OPENMAX in the parent are closed in the child, as are any elements in the filedesc_list array that are designated SPAWN_FDCLOSED. See the description of the BPXYCONS macro for the definition of the SPAWN_FDCLOSED constant (“BPXYCONS — Constants used by services” on page 952). The FCTLCLOFORK flag and FCTLCLOEXEC flags have no effect on inheritance when the filedesc_list is used to map the child's file descriptors.
   - The FCTLCLOEXEC and FCTLCLOFORK flags are not inherited from the parent file descriptors to the child's.
   - If the INHESETTCPGRP flag is set in the inheritance structure, INHECTLTTYFD must be set to the file descriptor that is associated with the controlling terminal for this session. The foreground process group for this session is set to the PGID of this child process, thus placing the child process in the foreground process group. (This is done by issuing a tcsetpgrp() syscall as a part of spawn processing.)

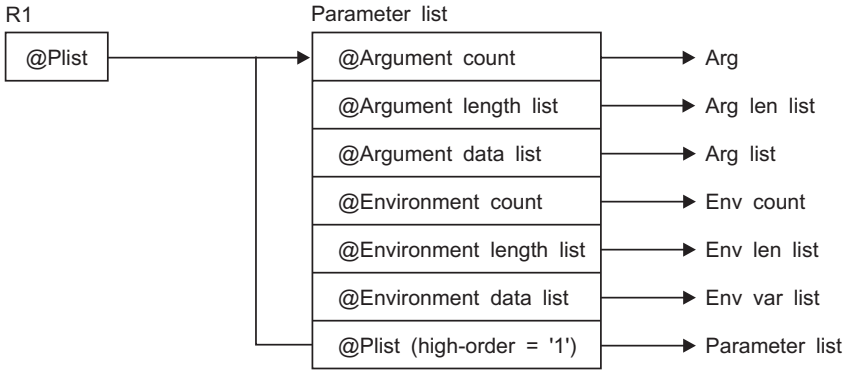
- If INHESETTCPGRP is not set, the foreground process group of the session remains unchanged.
- If the INHESETCWD flag is set on in the inheritance structure, the child's initial working directory is set to the directory path described by INHECWDPTR (pointer) and INHECWDLEN (length), provided that the caller has appropriate privileges.
- If the INHESETUMASK flag is set on in the inheritance structure, the child's file mode creation mask (umask) is set to the value in INHEUMASK, provided that the caller has appropriate privileges.
- If the INHESETREGIONSZ flag is set on in the inheritance structure, the child's region size is set to the value in INHEREGIONSZ, provided that the caller has appropriate privileges.
- If the INHESETTIMELIMIT flag is set on in the inheritance structure, the child's CPU time limit is set to the value in INHETIMELIMIT, provided that the caller has appropriate privileges.
- If the INHESETACCTDATA flag is set on in the inheritance structure, the child's accounting data is set to the value specified by INHEACCTDATAPTR (pointer) and INHEACCTDATALEN (length).
- If the INHESETMEMLIMIT flag is set on in the inheritance structure, the child's MEMLIMIT is set to the value indicated in INHEMEMLIMIT, provided that either the caller has appropriate privileges or INHEMEMLIMIT is less than the current hard limit of the target address space.
- The process and system utilization times for the child are set to zero.
- Any file locks that were previously set by the parent are not inherited by the child.
- The child process has no alarms or interval timers set. (This is similar to the results of a call to the alarm service with Wait_time specified as zero.)
- The child has no pending signals.
- The child gets a new process image to run the executable file that is not a copy of the parent's.
- Signals that are set to be caught are reset to their default action.
- Memory mappings that are established by the parent with the shmat or mmap services are not inherited by the child.
- The semaphore adjustment values, semadj, are set to zero.
- If the SSTFNOSUID bit is set for the file system that contains the new process image file, the effective user ID, effective group ID, saved set-user-ID and saved set-group_ID are unchanged in the new process image. Otherwise, if the `setuid` bit of the new process image file is set, the effective user ID of the new process image is set to the owner id of the new process image file. Similarly, if the `setgid` bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image will remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved `set-user-ID` and the saved `set-group-ID`) for use by the setuid and setgid functions.

4. The executable file that is to be run receives control with the following attributes:
   - When run in its own address space:
     – Problem program state

- TCB and PSW key 8
- AMODE=31(64), taken from the executable
- Primary ASC mode

• When run in the same address space as the caller of spawn:
- Problem program state
- TCB and PSW key equal to the TCB key of the caller of BPX1SPN/BPX4SPN
- AMODE=31(64), taken from the executable
- Primary ASC mode

5. The information that the service passes to the executable file to be run is a parameter list, which is pointed to by register 1. In 31–bit mode, the parameter lists contains a 4–byte addresses; in 64–bit mode the parameter list contains a 64–byte addresses. In 31–bit mode, in the last parameter address the high-order bit is 1. The parameter list consists of the following parameter addresses.



For AMODE 31 callers, the high-order bit in the last parameter address is 1. For AMODE 64 callers, the high-order bit is part of the 64-bit address. There are always *n* parameters, passed with no end-of-parameter-list indicator.

The last parameter that spawn passes to the executable file identifies the caller of the file as the exec or spawn service.

6. To support the creation and propagation of a STEPLIB environment to the new process image, the spawn service allows for the specification of a STEPLIB environment variable. The following are the accepted values for the STEPLIB environment variable, and the actions taken for each value:
   a. STEPLIB=NONE. No STEPLIB DD is to be created for the new process image.
   b. STEPLIB=CURRENT. The TASKLIB, STEPLIB or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the spawn service are propagated to the new process image, if they are found to be cataloged. Uncataloged data sets are not propagated to the new process image.
   c. STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image.

      The actual name of the DD is not STEPLIB, but a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order that is specified. The specified data sets must follow standard MVS data set naming conventions. A data set found to be in violation of this standard is ignored. Also, a data set that follows this standard is still ignored if:

      • The caller does not have the proper security access to a data set, or

- A data set is uncataloged or is not in load library format.

Because a data set in error is ignored, the executable file can run without the proper STEPLIB environment. If a data set is in error because of improper security access, a X'913' abnormal end is generated. The dump for this abnormal end can be suppressed by the installation.

If the STEPLIB environment variable is not specified, the default behavior of the spawn service is the same as if STEPLIB=CURRENT were specified.

If the program that is to be invoked is a set-user-ID or set-group-ID file, and the user-ID or group-ID of the file is different from that of the current process image, the data sets that are to be built into the STEPLIB environment for the new process image must be found in the system sanction list for set-user-id and set-group-id programs. Only those data sets that are found in the sanction list are built into the STEPLIB environment for the new process image. See Using sanction lists in *z/OS UNIX System Services Planning* for detailed information about the sanction list. See Tuning performance in *z/OS UNIX System Services Planning* for information about STEPLIB performance considerations.

7. For performance reasons, the spawn service is allowed to create a new image under a specific user ID that is different from that of the invoker. When an invoker with appropriate privileges specifies a user name on the _BPX_USERID environment variable or in the inheritance structure (INHEUSERID), the resulting image runs under the associated MVS user identity. This service allows spawn() to replace the sequence of functions fork(), setgid(), initgroups(), setuid(), and exec(). The following rules apply to spawn()s with user name changes (using either the _BPX_USERID environment variable or the inheritance structure):
   a. _BPX_USERID must be a valid 1-8 character XPG4-compliant name with a defined OMVS segment. An incorrect username length results in a failure of the spawn() request with an EINVAL and JRUserNameLenError. An undefined username results in an ERRNOJR of JRJsrRacXtr; an incompletely defined OE username results in an ERRNO of EMVSSAF2ERR and ERRNOJR of the propagated SAF return code and reason code.
   b. If the creation of the new address space with the new user identity puts the system over the limit of MAXUIDs, spawn() fails with an ERRNO of EAGAIN and an ERRNOJR of JRMaxUIDs.
   c. Authorization to change the username is the same as for the setuid() function. If the caller is not authorized, spawn() fails with an ERRNO of EPERM and an ERRNOJR of JRNoChangeIdentity (_BPX_USERID) or an ERRNOJR of JRInheUserid (inheritance structure).
   d. If _BPX_SHAREAS is active and an identity change is called for, _BPX_SHAREAS is ignored and a new address space is created for the new image.
8. To allow the caller to control whether the spawned child process runs in a separate address space from the parent address space or in the same address space, the spawn service allows for the specification of the _BPX_SHAREAS environment variable. The following are the accepted values for the _BPX_SHAREAS environment variable, and the actions taken for each value:
   a. _BPX_SHAREAS=YES - Indicates that the child process that is to be created is to run in the same address space as the parent. In the following circumstances, the _BPX_SHAREAS=YES value cannot be honored, and the child process is created in its own address space:

- If the program to be run is a set-user-ID or set-group-ID program that would cause the effective user-ID or group-ID of the child process to be different from that of the parent process.
- If the program to be run is an APF-authorized z/OS UNIX or MVS program and the caller is not running APF-authorized.
- If the program to be run is an unauthorized z/OS UNIX or MVS program and the caller is running APF-authorized.
- If the specified file name represents an external link or a sticky bit file. If, however, the program that is to be run is a shell script and _BPX_SPAWN_SCRIPT=YES is set, the process runs in the same address space.
- If the parent's address space lacks the necessary resources to create another process within the address space.

Note that only one local spawned process per TSO address space is supported at a given time. This is done to reduce conflict among multiple shells running in the same address space.

b. _BPX_SHAREAS=MUST - Indicates that the child process that is to be created must run in the same address space as the parent, or the spawn request will fail. In the following circumstances, the _BPX_SHAREAS=MUST value cannot be honored, and the spawn invocation fails:
   - If the program to be run is a set-user-ID or set-group-ID program that would cause the effective user ID or group ID of the child process to be different from that of the parent process.
   - If the program to be run is an APF-authorized z/OS UNIX or MVS program and the caller is not running APF-authorized.
   - If the program to be run is an unauthorized z/OS UNIX or MVS program and the caller is running APF-authorized.
   - If the parent's address space lacks the necessary resources to create another process within the address space.
   - If the list of environment variables specifies a value for _BPX_USERID that is different from the current user ID.
   - If the inheritance structure sets any of the following flag bits: InheSetRegionSz, InheSetMemLimit, InheSetTimeLimit, InheSetAcctData, InheSetJobname.

c. _BPX_SHAREAS=REUSE - This is equivalent to specifying _BPX_SHAREAS=YES.

d. _BPX_SHAREAS=NO - Indicates that the child process that is to be created is to run in a separate address space from the parent's address space. This is the default behavior for the spawn service if the _BPX_SHAREAS environment variable is not specified, or if it contains an unsupported value.

9. When the parent and child processes are sharing the same address space, special consideration must be given to the resources that are shared in this environment. These considerations include, but are not limited to, the following ones:
   - The parent and child share the same data set allocations, and must coordinate usage of these allocations. Programs that have special ddname allocation requirements should not be run in this shared environment.
   - The parent and child share the same private area storage; they should be careful not to overreach their own storage bounds, and together they must not exceed the region size of the address space.

- A prior loaded copy of a z/OS UNIX program is reused under the same circumstances that apply to the reuse of a prior loaded MVS unauthorized program from an unauthorized library by the MVS XCTL service, with the following exceptions:
  - If the calling process is in Ptrace debug mode, a prior loaded copy is not reused.
  - If the calling process is not in Ptrace debug mode, but the only prior loaded usable copy of the z/OS UNIX program found is in storage that is modifiable by the caller, the prior copy is not reused.
- If the specified file name represents an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is used only if the name is eight characters or less; otherwise the caller receives an error from the spawn service. For a sticky bit program, the file name is used if it is eight characters or less. Otherwise, the program is loaded from the z/OS UNIX file system.
- The program that is being spawned should have the APF extended attribute turned on and should be linked AC=1. DLLs that are loaded by APF-authorized applications should have the APF extended attribute set on and should be linked AC=0.

10. To allow the caller to control whether the spawn service is to treat the specified file as a shell script if it is found not to be in the correct format to be a process image file, the spawn service allows the specification of the _BPX_SPAWN_SCRIPT environment variable. The following are the accepted values for the _BPX_SPAWN_SCRIPT environment variable, and the actions that are taken for each value:
    - _BPX_SPAWN_SCRIPT=YES - Indicates that the specified file is to be treated as a shell script if the following are true:
      - The file is not in the correct format to be a process image file, and does not contain the "magic number" (#!) in the first line.
      - The file is not a REXX exec.

      In this case, the spawn service behaves as follows:
      - The z/OS shell executable file that is current in the caller's environment is executed to run the specified file as a shell script. The path name for the shell executable file is determined by extracting the path name from the SHELL environment variable, if the SHELL variable is present in the environment data list that is supplied to spawn. If it is not present, the default path name of `/bin/sh` is used as the shell executable path name.
      - The argument data list and argument length list that are passed to the shell executable file are to contain the following argument data and corresponding argument data lengths:
        - The shell path name, terminated by a null character (X'00')
        - The string '-S', terminated by a null character (X'00')
        - The string '—', terminated by a null character (X'00')

        The fourth through the last arguments in the list are to contain the list of arguments specified by the caller of the spawn service.
    - _BPX_SPAWN_SCRIPT=NO - Indicates that the specified file is NOT to be treated as a shell script if it is found not to be in the correct format to be a process image file, not to contain the file magic number (#!), and not to be a REXX exec. In this case, the spawn service fails and returns the return code ENOEXEC to the caller of spawn. This is the default behavior for the spawn service if the _BPX_SPAWN_SCRIPT environment variable is not specified, or if it contains an unsupported value.

11. If the specified z/OS UNIX file is not in the correct format to be an executable, but contains the magic number (#!) in the first line, the program that is specified in the magic number header is executed. The expected format of the magic number header is as follows:

    ```
    #! Path String
    ```

    `#!` is the file magic number. It identifies the first line of the file as a special header that contains the name of the program to be run and any argument data to be supplied to it.

    The *Path* parameter specifies the path name of the file that is to be run. It is separated by blank or tab characters from the `#!` characters, or can immediately follow the characters.

    The *String* parameter is an optional character string that can be used to pass options to a target command interpreter (shell) that is to run the script. It must be separated from the *Path* parameter by tab or blank characters, and cannot itself contain tab or blank characters.

    The argument data list and argument length list that are passed to the magic number file are to contain the following argument data and corresponding argument data lengths:

    - The magic number path name, ended by a null character (X'00')
    - The string, if one is supplied, ended by a null character (X'00')

    The remaining arguments in the list are to contain the list of arguments specified by the caller of the spawn service.

    If the path name that is specified in the magic number header cannot be executed for some reason, the spawn request fails with return code ENOEXEC, regardless of the error. ENOEXEC is returned for compatibility purposes, so that existing scripts can continue to run successfully when invoked from an application such as a command interpreter (shell). The reason code indicates the exact reason the magic number file could not be executed.
12. If the target executable program is enabled by Language Environment, the environment variables that are supplied to the service must include the null terminator as part of the string and length.
13. If the _BPX_PTRACE_ATTACH environment variable is set to YES, the target executable program is loaded into user-modifiable storage to allow subsequent debugging. Any additional programs that are loaded into storage during the execution of the target program are also loaded into user-modifiable storage, with the exception of modules that are loaded from the LPA.
14. The _BPXK_MDUMP environment variable can be used to specify where a SYSMDUMP is to be written. The following are the allowable values:

    Value **Description**

    OFF The dump is to be written to the current directory. This is the default. This dump is only written if the user allocates a SYSMDUMP data set for the TSO/E session. The system creates a file named coredump.*pid* in the user's working directory (where *pid* is the process ID for the process being dumped) and writes the core dump (SYSMDUMP) in hexadecimal format.

    MVS data set name
    The dump is to be written to an MVS data set. The data set name must be fully qualified and can be up to 44 characters. It can be specified in uppercase, lowercase, or both; it is folded to uppercase.

    z/OS UNIX file name
    The dump is to be written to a z/OS UNIX file. The file name can be

up to 1024 characters and must begin with a slash. The slash refers to the root directory, in which the file is created.

15. The _BPXK_JOBLOG environment variable can be used to specify that WTO messages are to be written to an open z/OS UNIX job log file. The following are the allowable values:

    Value **Description**

    nn Job log messages are to be written to open file descriptor *nn*.

    STDERR
    Job log messages are to be written to the standard error file descriptor, 2.

    NONE
    Job log messages are not to be written. This is the default.

    The file that is used to capture messages can be changed at any time by calling the oe_env_np service and specifying _BPXK_JOBLOG with a different file descriptor.

    Message capturing is turned off if the specified file descriptor is marked for close on a fork or exec.

    Message capturing is process-related. All threads under a given process share the same job log file. Message capturing may be initiated by any thread under that process.

    Multiple processes in a single address space can each have different files active as the JOBLOG file; some or all of them can share the same file; and some processes can have message capturing active while others do not.

    When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. Failure to do this causes unpredictable results.

    Only files that can be represented by file descriptors may be used as job log files; MVS data sets are not supported.

    Message capturing is propagated on a fork() or spawn(). If a file descriptor is specified, the physical file must be the same in order for message capturing to continue in the forked or spawned process. If STDERR is specified, the file descriptor may be remapped to a different physical file.

    Message capturing may be overridden on exec() or spawn() by specifying the _BPXK_JOBLOG environment variable as a parameter on the exec() or spawn().

    Message capturing only works in forked (BPXAS) address spaces.

    This is not true joblog support: messages that would normally go to the JESYSMSG data set are captured, but messages that go to JESMSGLG are not.

16. When the INHESETUSERID or INHESETACCTDATA flags are set on in the inheritance structure, the corresponding environment variables for user name (_BPX_USERID) or accounting data (_BPX_ACCT_DATA) are ignored.

17. Depending on the attributes and values that are specified in the inheritance structure (BPXYINHE), the caller is required to have different levels of authorization. The following table defines the specific authorization that is required for each attribute. Inheritance structure attributes that are not listed in the table do not require authorization.

BPXYINHE field	Authority required
INHEUSERID	The caller must have daemon authority. See Giving daemon authority to vendor-written programs in *z/OS UNIX System Services Planning* for information about setting up daemon authority for a user.
INHEREGIONSZ	When the new region size is smaller than the caller's current hard limit for RLIMIT_AS, no authorization is required. To exceed the current hard limit, the caller must have superuser authority (UID=0), or the spawn function will fail.
INHETIMELIMIT	When the new CPU time limit is less than the caller's current hard limit for RLIMIT_CPU, no authorization is required. To exceed the current hard limit, the caller must have superuser authority (UID=0), or the spawn function will fail.
INHEUMASK	The caller must have superuser authority to set the child's file mode creation mask, or the spawn function will fail.
INHECWD	The caller must have superuser authority to set the child's current working directory, or the spawn function will fail.

18. If the INHESETDEBUGENV flag in the inheritance structure is set on, the target program is under the control of the debugger process.
19. If the INHEMUSTBELOCAL flag in the inheritance structure is set on, the program must run in the same address space as the caller, or the spawn invocation will fail. (This flag causes the same behavior as _BPX_SHAREAS=MUST.) See Usage Note 8b for conditions that will cause the spawn invocation to fail when the INHEMUSTBELOCAL flag is set on.
20. If the BPXK_SIGDANGER environment variable is set to YES, the process will receive a **SIGDANGER** signal rather than a SIGTERM signal when an OMVS shutdown is initiated. This may be advantageous for an application that uses the SIGTERM signal for other purposes.
21. The _BPX_UNLIMITED_OUTPUT environment variable can be used to specify that default installation limits for sysout output are to be overridden. This environment variable is only processed when it is specified for a non-local spawn. These are the supported values:
    - YES - The sysout job limits (BYTES, CARDS, LINES and PAGES) will be set to the maximum values and an action of WARNING. This effectively allows unlimited output for the job that is associated with the newly spawned child.
    - NO - The installation defaults will be used for job output limits. This is the default behavior. Any other value will also result in the installation defaults being used.

    For non-local spawn to process the _BPX_UNLIMITED_OUTPUT environment variable, the caller must have appropriate privileges:
    a. Be a superuser (UID=0) or
    b. Be permitted to the BPX.UNLIMITED.OUTPUT resource profile in the FACILITY class with at least READ access.
22. When the spawn service is called from a __login process in a multiprocess, multiuser environment (environment variable _BPX_SHAREAS=YES is in effect), the newly created child process will have the MVS identity of the address space and the POSIX permissions of its parent process. It is expected that the application program will immediately call the __login service to establish a proper process-level identity in the child process.
23. The _BPXK_INITTAB_RESPAWN environment variable specifies whether a process is to be dynamically started with the respawn attribute. The following are the allowable values:

Value	Description
YES	Specifies that a process is to be started with the respawn attribute. Setting this to YES after the process has started does not affect the setting of the respawn attribute.
NO	Disables the respawn capability of the process. The process will not be respawned when it ends.

24. When the executable file to be run is a REXX exec, the first argument should be the path name of the REXX exec. Subsequent arguments for the exec can follow this argument.
25. The _BPXK_INITTAB_RESPAWN attribute is only allowed for non-local spawns. The _BPXK_INITTAB_RESPAWN setting is ignored if the process is spawned in the same address space. Setting _BPX_SHAREAS=YES, _BPX_SHAREAS=MUST, or INHEMUSTBELOCAL causes the _BPXK_INITTAB_RESPAWN option to be ignored on the spawn call.
26. z/OS UNIX sets a default message class of "A" for all forked or spawned processes. Unlike JES, z/OS UNIX does not have a method for accepting a user-supplied default message class, and a default had to be supplied to the converter interpreter. Message class A was chosen as the default for BPXAS initiators. There is currently no way to dynamically change this default value. The MSGCLASS for the joblog (JESMSGLG, JESJCL, JESYSMSG) is set to class A before the fork or spawn that associates the process with the BPXAS initiator is begun.
27. The _BPX_JOBNAME environment variable can be used to change the jobname of the new process image. The jobname change is allowed only if the invoker has appropriate privileges (see "Authorization" on page 8) . If these conditions are not met, the environment variable is ignored. Accepted values are strings of 1–8 alphanumeric characters. Incorrect specifications are ignored.
    - A job name can be specified in the INHE structure. If both the INHE JOBNAME and environment variable _BPX_JOBNAME are specified, the value in the INHE structure is used.
    - If _BPX_JOBNAME is specified and _BPX_SHAREAS is set to YES, then _BPX_JOBNAME is ignored.
28. If the caller specifies _BPXK_DISABLE_SHLIB=YES then future loadhfs() and loadhfs_extended() system calls will ignore the st_sharelib attribute and load the program into private storage. If the caller specifies NO (the default) then normal system shared library processing takes place. For more information, see Commonly used environment variables in *z/OS UNIX System Services Planning*
29. If the specified file name resolves to an external link or a sticky bit file, the program is loaded from the caller's MVS load library search order. For an external link, the external name is used only if the name is eight characters or less; otherwise the caller receives an error from the spawn service. For a sticky bit program, the file name is used if it is eight characters or less. Otherwise, the program is loaded from the z/OS UNIX file system. If the specified sticky bit file or link resolves to a MVS program link-edited AC=1 located in an APF-authorized library, the attributes of the sticky bit file or external link must be set up properly to allow this type of invocation. For a sticky bit file, it must be installed with an owning UID of 0 or with the APF extended attribute. The owning UID of 0 requirement would also apply to a symbolic link that resolves to the sticky bit file. For an external link, it must be installed with an owning UID of 0. A sticky bit a file with the APF extended attribute is not allowed if found in a file system that is mounted as NOSETUID. If the specified file name represents a symbolic link to a sticky bit file that has the

set-user-id attribute, the symbolic link must have an owning uid of 0 or an owning uid equal to that of the sticky bit file. If the sticky bit file has the set-group-id attribute, the symbolic link must have an owning uid of 0 or an owning GID equal to that of the sticky bit file. A file or link that is found in a file system mounted as NOSECURITY is not considered trusted for this type of invocation, regardless of its attributes. Failure to follow this set up will cause the child process created to run the MVS program to end abnormally with a EC6-xxxxC04A abend when the MVS program is invoked via the spawn service.

Related services

- "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29
- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755
- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805
- "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710
- "tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID" on page 849
- "umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask" on page 866

Characteristics and restrictions

1. Executing a program from z/OS UNIX causes the program environment to become uncontrolled, unless the program is identified as program controlled. (That is, unless the ST_PROGCTL attribute is ON for the z/OS UNIX program file). Running a z/OS UNIX program with the ST_PROGCTL attribute set to OFF prevents future invocations of authorized programs like program access to data sets (PADS) programs. These are programs givenspecial authorization by the installation and by the installed security product (such as RACF) to read or write to protected data sets. In addition, PADS programs should not attempt to load programs from z/OS UNIX with the ST_PROGCTL attribute OFF, because these programs are considered uncontrolled and could have been modified by users that do not have the same level of authorization as the PADS program.
2. The _BPXK_TIMEOUT environment variable is processed for this callable service. It allows applications to specify whether a process waiting on terminal input in the shell should be timed out. For more information about the _BPXK_TIMEOUT environment variable, see *z/OS UNIX System Services Planning*.

Examples

For an example using this callable service, see "BPX1SPN (spawn) example" on page 1194.

srx_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket

Function

The srx_np callable service sends or receives data on a socket using CSM (Communications Storage Manager) buffers.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task SRB - AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1SRX):	31-bit task or SRB mode
AMODE (BPX4SRX):	64-bit SRB mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SRX,(Socket_descriptor,
              Direction,
              Msghdrx_length,
              Msghdrx,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SRX with the same parameters. All addresses in the Msghrdx structure are 31-bit pointers.

Parameters

`Socket_descriptor`
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the socket file descriptor for which the srx_np service is requested.

`Direction`
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the operation requested:

- MSGX_SEND or 0, for a send operation
- MSGX_RECV or 1, for a receive operation

Msghdrx_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the length of the Msghdrx parameter.

Before the data structures are built for the first time, you can use a value of 0 in this field to determine whether the operation is supported on a given socket. If the operation is supported, a Return_value of 0 is returned. If the operation is not supported, a Return_value of -1 with a Return_code of ENOSYS is returned.

Msghdrx
Supplied and returned parameter

Type: Structure

Length:
Length of MSGX from BPXYMSGX

The name of the MSGX structure that contains the information for this operation. See the usage notes and the BPXYMSGX macro ("BPXYMSGX — Map the message header" on page 999) for more information about the MSGX structure.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the srx_np service returns one of the following:

- The number of bytes that were sent or received from the buffers, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the srx_np service stores the return code. The srx_np service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The srx_np service can return one of the following values in the Return_code parameter:

Return_code	Explanation
ENOSYS	This function is not supported on the specified socket.
EAFNOSUPPORT	The address family that was specified in the message header is not the same as the address family that owns the socket.

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	The connection was reset by a peer. The following reason code can accompany the return code: JRSockNotCon.
EINTR	A signal interrupted the srx_np service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	An input parameter was incorrect. The following reason codes can accompany the return code: JRInvalidMsgh, JRSocketCallParmError, and JRSockNoName.
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
EFAULT	An address that was passed pointed to storage that could not be accessed.
ENOTCONN	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPIPE	An attempt was made to send a message to a socket that is shut down or closed. This error also generates a SIGPIPE signal. The following reason code can accompany the return code: JRSocketClosed.
EWOULDBLOCK	For a receive operation, the socket is marked nonblocking and there is no data available to be read, or the SO_RCVTIMEO timeout value was reached before data was available. The following reason codes can accompany the return code: JRWouldBlock, JrTimeOut.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the srx_np service stores the reason code. The srx_np service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The Communications Storage Manager (CSM) provides a facility that allows programs to avoid data moves on a communications session by transferring buffer ownership instead of copying the buffer contents.

   The srx_np service provides a way to send these buffers on a socket session. It is assumed that the application has its own interactions with CSM that allow it to obtain and free these buffers independently from the srx_np service. CSM is restricted to authorized programs, and the buffers are in key 6 storage. The srx_np service, however, may be invoked from problem state or authorized programs. All parameters are in local application storage and the caller's key.

   For more information about CSM, see *z/OS Communications Server: CSM Guide*.
2. The general flow for using this service is as follows:

When **sending:**

a. The application obtains CSM buffers and fills them with the data to be sent. The collection of one or more CSM data buffers is described with an IOVX array that may be built in application storage or in another CSM buffer. This whole structure and operational characteristics are specified in a Msghdrx structure, which is passed to the srx_np callable service.

b. The request is passed on to the transport, such as TCP/IP, for the specified socket. If the socket transport does not support CSM buffers, the call fails with ENOSYS. A specific socket can be tested for support before the buffers are built, by specifying `0` for Msghdrx_length. If CSM buffers are not supported, the data will have to be sent with standard services, such as send (BPX1SND, BPX4SND) or writev (BPX1WRV, BPX4WRV). CSM buffers could be used on the standard services, but they would be treated as application buffers, and the application would retain ownership and be responsible for freeing the buffers.

c. The socket transport transfers ownership of the CSM data buffers, not the IOVX, to itself and passes them along to the communications adapter. Ownership of the IOVX buffer, if it is a CSM buffer, remains with the application.

d. The communications adapter transfers buffer ownership to itself and transmits the data.

e. When the I/O is complete, the adapter issues CSM deallocates for the buffers.

f. CSM puts the buffers back into its global free pool.

g. When control returns to the application after the srx_np call, it no longer owns the buffers and must not reference them again.

When **receiving:**

a. Inbound data is received into CSM buffers obtained by the communications adapter.

b. These buffers are passed up to the socket transport, who assumes ownership.

c. The application calls srx_np to receive. A Msghdrx structure is passed that may contain some control information, but that does not specify any buffers or an IOVX array.

d. The socket transport builds an IOVX array to describe the inbound data buffers that have been accumulated. This array is itself in a CSM buffer.

   If data has not arrived yet, the request is suspended or failed with EWOULDBLOCK, as for any other socket receive type of operation.

   When data is to be returned to the application, the transport assigns ownership of the CSM buffers to the application, and the application's Msghdrx structure is filled in with a description of the IOVX array buffer.

e. When control returns to the application after the srx_np call, it has ownership of the CSM buffers and may process the data that has been received.

f. When the application has finished with the buffers and the IOVX array, it issues CSM deallocates for them.

g. CSM puts the buffers back into its global free pool.

3. For a receive operation, Msghdrx contains the following fields:

Field	Description
MsgxNamePtr	A pointer to a sockaddr buffer in which the system returns the source address of the data that is received. This field is optional. If it is not used, MsgxNamePtr or MsgxnameLen should be zero.
MsgxNameLen	The length of the sockaddr buffer that is pointed to by MsgxNamePtr.
MsgxIovx	An IVTBUFL structure in which the system describes the CSM buffer containing the IOVX array being returned for this request. This CSM buffer is obtained by the system and freed by the calling application. The IOVX array contains IVTBUFL structures, each of which describes a CSM data buffer that contains the data received by this request. The CSM data buffers that are used by this service are obtained by the system and freed by the caller.
MsgxMsgFlags	MSG_* flags for this operation. Refer to "BPXYMSGF — Map the message flags" on page 997.
MsgxFlags	Control flags: • MSGX_CECSA, indicating that the CSM buffers should be obtained from ECSA • MSGX_CDSPACE, indicating that the CSM buffers should be obtained from one of the CSM data spaces If neither flag is specified, the application can handle CSM buffers in either ECSA or a data space.
MsgxDataLen	The maximum or minimum amount of data that is to be received: • When MSG_WAITALL is off, MsgxDataLen specifies the maximum amount of data that the caller wants to receive. • When MSG_WAITALL is on, MsgxDataLen specifies the minimum amount of data that the caller wants to receive. You can use this value to control the amount of data that is received, in the same way that you use the Buffer_length parameter of the recv service. If this field is `0`, the receive operation completes as soon as the first block of data is available, and whatever data is available is returned. If the receive operation cannot be completed immediately, the application blocks or receives an EWOULDBLOCK error, depending on its blocking state.
MsgxTcb	The TCB address of a task with which the CSM storage is to be associated. By default the storage is associated with the calling task. This field is optional, and should be `0` if not specified.

4. For a send operation, Msghdrx contains the following fields:

Field	Description
MsgxNamePtr	A pointer to a sockaddr buffer that contains the destination address for the send operation. This field is optional. If it is not used, MsgxNamePtr or MsgxNameLen should be 0.
MsgxNameLen	The length of the sockaddr buffer that is pointed to by MsgxNamePtr.
MsgxIovx	An IVTBUFL structure that describes the buffer containing the IOVX array. This buffer may be a CSM buffer, or it may be in the caller's storage. Ownership of a CSM buffer used for the IOVX array remains with the application. The IOVX array contains IVTBUFL structures, each of which describes a CSM data buffer that contains the data to be sent. The CSM data buffers that are used by this service are obtained by the caller and freed by the system.
MsgxMsgFlags	MSG_* flags for this operation. Refer to "BPXYMSGF — Map the message flags" on page 997.
MsgxIVTBUFLOffset	The returned offset of the IOVX array entry for the first CSM data buffer that the application still owns. After a successful send, this value is equal to the length of the IOVX array. If this value is zero, no buffers were taken.
MsgxErrIovx	The offset of the IOVX array entry that is in error. This field and MsgxErrData are returned only when there is an error that is specifically related to one of the IOVX entries or their associated buffers. Refer to the Return_code and Reason_code for details on the error.
MsgxErrData	The amount of data that has been sent successfully from the buffer that is indicated by MsgxErrIovx.

MsgxErrIovx and MsgxErrData should only be examined when the request completes with a Return_value of -1, or when the amount of data sent is less than the amount of data that was requested to be sent.

5. A C header, BPXYSRXH, is available which contains a C structure for the Msghdrx and a prototype for srx_np. With this header and the IVTBUFL C header, you can send and receive CSM buffers from C programs.

   Note, however, that this program would simply be making C calls to the srx_np callable service, and not making normal C functional references. In particular, the return value and errno value would be returned in explicit calling parameters, rather than in the standard C method.
6. The socket may be connected or unconnected.
7. Consult the documentation for the TCP/IP stack that is being used for support for this function.
8. Some **common INET considerations:**

   When the socket is associated to a specific transport, the requests are accepted or rejected based on that transport's support for CSM buffers. A socket becomes associated with a specific transport by being a connected stream socket, bound to a specific IP address, or through setibmopt(IBM_IMAGE) or ioctl(SIOCSETRTTD).

   When the application's socket is associated with more than one transport, every associated transport must support CSM buffers for a receive operation to be

accepted. For a send operation, the transport chosen by the system for the destination IP address must support CSM buffers.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.

Related services

"recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers" on page 604

Characteristics and restrictions

There are no restrictions on the use of the srx_np service.

Examples

For an example using this callable service, see "BPX1SRX (srx_np) example" on page 1197.

stat (BPX1STA, BPX4STA) — Get status information about a file by pathname

Function

The stat callable service obtains status information about a specified file. You specify the file by its pathname.

If the pathname that is specified refers to a symbolic link, the symbolic link name is resolved to a file, and the status information for that file is returned. To obtain status information about a symbolic link, rather than a file it refers to, use "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349.

For the corresponding service using a file descriptor, see "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STA):	31-bit
AMODE (BPX4STA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STA,(Pathname_length,
              Pathname,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STA with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Pathname.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of an area, of length Pathname_length, that contains the pathname of the file for which you want to obtain status. The Pathname can be a pathname to a file, a link named by a pathname to a file (as created by "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327), or a symbolic link named by a pathname to a file (as created by "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812).

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area to which the service returns Status_area. To determine the value of Status_area_length, use the BPXYSTAT macro (see "BPXYSTAT — Map the response structure for stat" on page 1057).

Status_area
Parameter supplied and returned

Type: Structure

Length:
Specified by the Status_area_length parameter

The name of an area, of length Status_area_length, to which the service returns the status information for the file. The Status_area is mapped by the BPXYSTAT macro (see "BPXYSTAT — Map the response structure for stat" on page 1057).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the stat service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the stat service stores the return code. The stat service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The stat service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The process does not have permission to search some component of the Pathname prefix.
EINVAL	Parameter error—for example, a zero-length buffer. The following reason code can accompany the return code: JRBuffTooSmall.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or some component of the pathname is longer than 255 characters. This could be as a result of encountering a symbolic link during resolution of Pathname, if the substituted string is longer than 1023 characters.
ENOENT	No file named Pathname was found, or Pathname was not specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	A component of the Pathname prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the stat service stores the reason code. The stat service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. All time fields in the Status_area are in POSIX format.
2. The File Mode field in the Status_area is mapped by the BPXYMODE macro (see "BPXYMODE — Map the mode constants of the file services" on page 996). For information on the values for file type, see "BPXYFTYP — File type definitions" on page 967.
3. If no security label (SECLABEL) exists for the file, the security label field in the Status_area contains binary zeros.

Related services

- "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90
- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191
- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196
- "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327
- "mkdir (BPX1MKD, BPX4MKD) — Make a directory" on page 361
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572
- "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872
- "utime (BPX1UTI, BPX4UTI) — Set file access and modification times" on page 879
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

To obtain information about a file, you need not have permissions for the file itself; however, you must have search permission for all the directory components of Pathname.

Examples

For an example using this callable service, see "BPX1STA (stat) example" on page 1198.

statvfs (BPX1STV, BPX4STV) — Get the file system status

Function

The statvfs callable service obtains status information about a file system that is specified by a file pathname from the desired file system.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1STV):	31-bit
AMODE (BPX4STV):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STV,(Pathname_length,
              Pathname,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STV with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname.

Pathname
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
Pathname_length

The name of a field, of length Pathname_length, that specifies a file pathname in the file system about which status is desired.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the area to which the service returns status information.

Status_area
Parameter supplied and returned

Type: Structure

Length:
Specified by the Status_area_length parameter

The name of an area of length Status_area_length to which the service returns the status information for the file system. The BPXYSSTF macro maps this area. For information on this macro, see "BPXYSSTF — Map response structure for file system status" on page 1055.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the statvfs service returns the length of the status written to the Status_area if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the statvfs service stores the return code. The statvfs service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The statvfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search some component of the Pathname prefix.
EAGAIN	Information is temporarily unavailable. This can occur if the mount process for the file system is not complete.
EINVAL	Parameter error; for example, Status_area_length is too small. The following reason code can accompany the return code: JRBuffTooSmall.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters.
ENOENT	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.

Return_code	Explanation
ENOTDIR	Some component of the Pathname prefix is not a directory.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the statvfs service stores the reason code. The statvfs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Provided that the passed Status_area_length is not less than or equal to zero, it is not considered an error if the Status_area_length is not sufficient to hold all the requested information. (That is, future expansion is allowed for.) As much information as will fit is written to Status_area, and this amount is returned.
2. The amount of valid data that is returned in the Status_area is indicated by the Return_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

Related services

- "fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status" on page 199
- "w_statvfs (BPX1STF, BPX4STF) — Get the file system status" on page 926

Characteristics and restrictions

There are no restrictions on the use of the statvfs service.

Examples

For an example using this callable service, see "BPX1STV (statvfs) example" on page 1200.

sw_sigdlv (BPX1DSD, BPX4DSD) — Switch the setting for signal delivery

Function

The sw_sigdlv callable service enables or disables signal delivery for the current process.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1DSD):	31-bit
AMODE (BPX4DSD):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1DSD,(signal_ind)
```

AMODE 64 callers use BPX4DSD with the same parameter.

Parameters

signal_ind
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing a numeric value that indicates whether signal delivery should be enabled or disabled. The signal_ind constants are defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Description
SW_SIGDLV_ENABLE#	Enable signal delivery
SW_SIGDLV_DISABLE#	Disable signal delivery

Usage notes

There are no returns from the sw_sigdlv callable service. The task is abended if an error occurs.

Related services

- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup" on page 430
- "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757

Characteristics and restrictions

There are no restrictions on the use of the sw_sigdlv service.

symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name

Function

The symlink callable service creates a symbolic link to a path name. A file of type "symbolic link" is created.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SYM):	31-bit
AMODE (BPX4SYM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SYM,(Pathname_length,
              Pathname,
              Link_name_length,
              Link_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SYM with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Pathname.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field, of length Pathname_length, that contains the pathname for which you are creating a symbolic link.

path names can begin with or without a slash:

- A path name that begins with a slash is an absolute path name. The slash refers to the root directory, and the search for the file starts at the root directory.
- A path name that does not begin with a slash is a relative path name, and the search for the file starts at the parent directory of the symbolic link file.

Link_name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Link_name. The name can be up to 1023 bytes long; each component of the name (between delimiters) can be up to 255 bytes long.

Link_name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Link_name_length parameter

The name of a field that contains the symbolic link that is being created.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the symlink service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the symlink service stores the return code. The symlink service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The symlink service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The requested operation requires writing in a directory with a mode that denies write permission.
EEXIST	The link name already exists. The following reason code can accompany the return code: JRSymFileAlreadyExists.
EFBIG	A request to create a symbolic link is prohibited because the file size limit for the process is set to 0.

Return_code	Explanation
EINVAL	This error code may be returned for any of the following reasons: • A component of the path prefix of Pathname or the entire path name exceeds the maximum allowed. • The value of Pathname_length is less than or equal to zero. • A null character appears in Pathname. • Link_name has a slash as its last component, which indicates that the preceding component is a directory. A symbolic link cannot be a directory. The following reason code can accompany the return code: JRCompNotDir, JRInvalidSymLinkCom, JRInvalidSymLinkLen, and JRNullInPath.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Link_name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Link_name.
ENAMETOOLONG	Pathname or Link_name is longer than 1023 characters, or some component of that name is longer than 255 characters. Name truncation is not supported.
ENOSPC	The directory in which the entry for the symbolic link is being placed cannot be extended; not enough space remains in the file system.
ENOTDIR	A component of the path prefix of Link_name is not a directory.
EROFS	The requested operation requires writing in a directory on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFS.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the symlink service stores the reason code. The symlink service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The symlink service creates a symbolic link (Link_name) with the file that you specify by Pathname.
2. Like a hard link (described in "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327), a symbolic link allows a file to have more than one name. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link, however, provides no such assurance; in fact, the file identified by Pathname need not exist when the symbolic link is created. In addition, a symbolic link can cross file system boundaries.
3. When a component of a path name refers to a symbolic link rather than to a directory, the path name that is contained in the symbolic link is resolved. If the path name in the symbolic link begins with / (slash), the symbolic link path name is resolved relative to the process root directory. If the path name in

the symbolic link does not begin with /, the symbolic link path name is resolved relative to the directory that contains the symbolic link.

4. If the symbolic link is not the last component of the original path name, remaining components of the original path name are resolved from there.

   When a symbolic link is the last component of a path name, it may or may not be resolved. Resolution depends on the function that is using the path name. For example, a rename request does not have a symbolic link resolved when it appears as the final component of the new or old path name. However, an open request does have a symbolic link resolved when it appears as the last component.

   When a slash is the last component of a path name, and it is preceded by a symbolic link, the symbolic link is always resolved.

5. Because the mode of a symbolic link cannot be changed, its mode is ignored during the lookup process. Any files and directories to which a symbolic link refers are checked for access permission.

6. Sysplex members participating in a shared file system can access (read/write) file system data on other systems in the sysplex. For example, if SY1 and SY2 are two systems in a sysplex, a user on SY1 can access SY2's **/etc** directory.

   The shared file system capability requires that **/etc**, **/dev**, **/var**, and **/tmp** be converted into symbolic links. If the content of the symbolic link begins with $VERSION, $SYSNAME, $SYSSYMR, or $SYSSYMA, the symbolic link will resolve in a specific manner:

   - If the content of the symbolic link begins with $SYSNAME and the BPXPRMxx parameter SYSPLEX is specified YES, $SYSNAME is replaced with a slash followed by the system name (/SY1). If SYSPLEX(NO) is specified, $SYSNAME is replaced with /SYSTEM. For example, if you have specified SYSPLEX(YES) and the symbolic link for **/etc** has the contents **$SYSNAME/etc**, this will resolve to **/SY1/etc** on a system whose name is SY1, and will resolve to **/SY2/etc** on a system whose name is SY2.
   - If the content of the symbolic link begins with $VERSION, $VERSION will resolve to the value specified on the VERSION parameter in BPXPRMxx. Thus, if VERSION in parmlib is set to REL9, resolution of a symbolic link with $VERSION causes $VERSION to be replaced with /REL9. For example, the symbolic link for **/bin**, which has the contents **$VERSION/bin**, will resolve to **/REL9/bin** on a system whose $VERSION value is set to REL9.
   - When a component of the path name is a symlink that begins with $SYSSYMR or $SYSSYMA, any MVS static symbols in the template are replaced with the resolved substitution text. In the following examples, **&SYSR1** is an MVS static symbol:
     - If the content of the symbolic link begins with $SYSSYMR, $SYSSYMR/ results in a relative path name; that is, the lookup proceeds from its current position in the path name. For example, if the symlink is **/x/y/sym1** and the symlink contains **$SYSSYMR/&SYSR1./resdir**, a path name lookup on **/x/y/sym1** from SY1 will resolve the symlink to **OSV315/resdir**. Because it is a relative path name (the identifier was $SYSSYMR/), the resulting path name will be **/x/y/OSV315/resdir**.
     - If the content of the symbolic link begins with $SYSSYMA, $SYSSYMA/ results in an absolute path name; that is, the lookup starts over at the root. For example, if the symlink is **/x/y/sym1** and the symlink contains **$SYSSYMA/&SYSR1./resdir**, a path name lookup on **/x/y/sym1** from SY1 will resolve the symlink to **/OSV315/resdir**. Because it is an absolute path name (the identifier was $SYSSYMA/), the resulting path name will be **/OSV315/resdir**.

Only the occurrence of $SYSSYMR/ or $SYSSYMA/ at the start will be recognized as an identifier for which the remaining text requires substitution. Any other identifiers after the beginning will remain as is in the resolved linkname. There must be some text following a $SYSSYMR/ or $SYSSYMA/ for it to be recognized as a valid identifier with text containing symbols to be resolved.

If the content of the symbolic link begins with $SYSSECA/ or $SYSSECR/, the user's current security label is substituted into the path name. The symbol $SYSSECA indicates that the user's current security label should be substituted into the path name as an absolute directory name. Pathname resolution continues at the ROOT with a directory name of the user's current security label. The symbol $SYSSECR indicates that the user's current security label should be substituted into the path name as a relative directory. The path name resolution continues in the directory in which the symbolic link is encountered, with a directory name of the user's current security label. For more information about using security labels, see *z/OS Planning for Multilevel Security and the Common Criteria*.

For more information about file system sharing in a sysplex, see Shared file systems in a sysplex in *z/OS UNIX System Services Planning*. The SYSPLEX(YES|NO) and VERSION('nnnn') BPXPRMxx parameters are described in *z/OS MVS Initialization and Tuning Reference*.

7. Certain directories like **/etc, /dev, /tmp,** and **/var** are converted to symbolic links. Some shell commands have minor technical differences when referring to symbolic links than for regular files or directories. For example, **ls** does not follow symbolic links by default. **/etc** is a symbolic link, so `ls /etc` will display only the symbolic link name, in this case **/etc**.

   In order to follow symbolic links, you must specify `ls -L` or provide a trailing slash. For example, `ls -L /etc` and `ls /etc/` both display the files in the directory that the **/etc** symbolic link points to.

   Other shell commands that have differences due to symbolic links are **du, find, pax, rm** and **tar**.

8. By default, the owning GID of the symbolic link is set to that of the parent directory. However, if the FILE.GROUPOWNER.SETGID profile exists in the UNIXPRIV class, the owning GID is determined by the set-gid bit of the parent directory, as follows:
   - If the parent's set-gid bit is on, the owning GID is set to that of the parent directory.
   - If the parent's set-gid bit is off, the owning GID is set to the effective GID of the process.

Related services

- “chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory” on page 93
- “mkdir (BPX1MKD, BPX4MKD) — Make a directory” on page 361
- “mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file” on page 364
- “lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name” on page 349
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 447
- “readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link” on page 587
- “rename (BPX1REN, BPX4REN) — Rename a file or directory” on page 607

- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615
- "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872

Characteristics and restrictions

There are no restrictions on the use of the symlink service.

Examples

For an example using this callable service, see "BPX1SYM (symlink) example" on page 1202.

sync (BPX1SYN, BPX4SYN) — Schedule file system updates

Function

The sync callable service causes all information in memory that updates file systems to be scheduled for writing.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SYN):	31-bit
AMODE (BPX4SYN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SYN,(Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SYN.

Parameters

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sync service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sync service stores the return code. The sync service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sync service stores the reason code. The sync service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The actual writing of data to all file systems is scheduled, but is not necessarily completed, upon return from the sync() service.

Characteristics and restrictions

There are no restrictions on the use of the sync service.

Examples

For an example using this callable service, see "BPX1SYN (sync) example" on page 1202.

sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options

Function

The sysconf callable service gets the value of a configurable system variable.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1SYC):	31-bit
AMODE (BPX4SYC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1SYC,(Sysconf_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4SYC with the same parameters.

Parameters

Sysconf_name
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that specifies the configurable system variable that is to be retrieved. Each configurable system variable is mapped to a specific value that is defined in the BPXYCONS macro. See "BPXYCONS — Constants used by services" on page 952.

Constant	Configurable system variable returned
SC_ARG_MAX	For ARG_MAX
SC_CHILD_MAX	For CHILD_MAX
SC_CLK_TCK	For CLK_TCK
SC_JOB_CONTROL	For _POSIX_JOB_CONTROL
SC_NGROUPS_MAX	For NGROUPS_MAX
SC_OPEN_MAX	For OPEN_MAX
SC_SAVED_IDS	For _POSIX_SAVED_IDS
SC_MMAP_MEM_MAX_NP	For MMAP_MEM_MAX_NP
SC_TTY_GROUP	For TTY GROUP
SC_THREADS_MAX_NP	For _THREADS_MAX_NP
SC_THREAD_TASKS_MAX_NP	For _THREAD_TASKS_MAX_NP
SC_TZNAME_MAX	For TZNAME_MAX
SC_VERSION	For _POSIX_VERSION
SC_2_CHAR_TERM	For CHAR_TERM
SC_PAGESIZE	For PAGESIZE
SC_PAGE_SIZE	For PAGE_SIZE

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sysconf service returns the actual value of the configurable system variable if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sysconf service stores the return code. The sysconf service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The sysconf service can return the following value in the Return_code parameter:

Return_code	Explanation
EINVAL	The value of the Sysconf_name argument is not valid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the sysconf service stores the reason code. The sysconf service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. SC_MAX_THREADS_NP, SC_CHILD_MAX, SC_OPEN_MAX, SC_MMAP_MEM_MAX, and SC_MAX_THREAD_TASKS_NP return the limits that are defined for the caller's process, not the system-wide limits.
2. SC_PAGE_SIZE and SC_PAGESIZE return the page size based on the AMODE of the caller. AMODE 31 callers get a page size of 4 K, and AMODE 64 callers get a page size of 1M.

Related services

- "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464
- "set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads" on page 704

Examples

For an example using this callable service, see "BPX1SYC (sysconf) example" on page 1202.

takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program

Function

The takesocket callable service acquires a specified socket from the program that is identified in the Clientid parameter.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TAK):	31-bit

Operation	Environment
AMODE (BPX4TAK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TAK,(Clientid,
              Socket_Id,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TAK with the same parameters.

Parameters

Clientid
Supplied parameter

Type: Structure

Length:
Length of BPXYCID

The name of a structure that contains Clientid information that identifies the (server) program from which the socket is to be taken. This information is typically obtained with the getclientid (BPX1GCL, BPX4GCL) service, issued by the server and passed to the slave. See "BPXYCID — Map the returning structure for getclientid()" on page 951 for more information about the format of this field. Clientid input may be as follows:

CIdDomain
Domain of the socket that is to be taken. See "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 for more information about the values that are defined for this field.

CIdName
One of the following:

- The server program's address space name
- A fullword of binary zeros followed by the server program's process id.

CIdTask
The server program's subtask identifier (supplied only if the address space name was supplied in the CIdName field).

CIdReserved
Binary zeros.

Socket_Id
Supplied parameter

Type: Integer

Length:
Fullword

An identifier for the socket that is being taken. This is supplied by the server program, and is either the socket descriptor obtained from an accept, or the socket token returned on a givesocket (BPX1GIV, BPX4GIV) service if givesocket was invoked with CIdType=CId#Close.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the takesocket service returns one of the following:

- -1 if the request is not successful.
- If not -1, the return value is the new socket descriptor.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the takesocket service stores the return code. The takesocket service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The takesocket service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The Socket_Id does not specify a valid socket that is owned by the other application; or the socket has already been taken.
EACCES	The other application did not give the socket to your application.
EFAULT	Using the Clientid parameter as specified would result in an attempt to access storage that is outside the caller's address space.
EINVAL	The Clientid parameter does not specify a valid client identifier: either the client's process cannot be found, or the client's process was found, but it has no outstanding givesockets.
EMFILE	The socket descriptor table is already full.
EPERM	The givesocket security label does not match the takesocket security label (JrUserNotAuthorized).

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the takesocket service stores the reason code. The takesocket service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The takesocket callable service used to be an MVS TCP/IP API, and was added to the z/OS UNIX callable services to allow migration of applications to a single library.
2. The Clientid output of getclientid (BPX1GCL, BPX4GCL) that is issued by the server program and passed to the slave is intended to be used as the input Clientid of the takesocket service. This identifies the program from which the socket is to be taken. By using a FunctionCode of 2 on the getclientid service to obtain Clientid information that is to be used as the Clientid input of the takesocket service, the best performance of the takesocket service is achieved.

Related services

- "getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier" on page 213
- "givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program" on page 285

Characteristics and restrictions

There are no restrictions on the use of the takesocket service.

tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted

Function

The tcdrain callable service waits until all output sent to a file descriptor has actually been sent to the terminal device.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TDR):	31-bit
AMODE (BPX4TDR):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TDR,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TDR with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor that represents the output device.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcdrain service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcdrain service stores the return code. The tcdrain service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The tcdrain service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor does not describe a valid open file.
EINTR	A signal interrupted the service before all output had been sent.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENOTTY	File_descriptor is not associated with a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcdrain service stores the reason code. The tcdrain service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. For slave pseudoterminals, data is considered written when the master side has read it.

2. The following table defines the processing of the **SIGTTOU** signal when tcdrain is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal"
- "tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal" on page 829
- "tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal" on page 840

Characteristics and restrictions

There are no restrictions on the use of the tcdrain service.

Examples

For an example using this callable service, see "BPX1TDR (tcdrain) example" on page 1203.

tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal

Function

The tcflow callable service suspends or resumes data flow on a terminal.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TFW):	31-bit
AMODE (BPX4TFW):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TFW,(File_descriptor,
              Action,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TFW with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal device.

Action
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains an indicator of the action that is to be taken. The possible constants are mapped in the BPXYTIOS macro (see "BPXYTIOS — Map the termios structure" on page 1065).

Constant	Description
TCIOFF	Send a STOP character to the terminal to stop the terminal from sending any further input.
TCION	Send a START character to the terminal to start the terminal sending input.
TCOOFF	Suspend output to the terminal.
TCOON	Resume output to the terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflow service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflow service stores the return code. The tcflow service returns Return_code only if Return_value is -1. See *z/OS UNIX*

System Services Messages and Codes for a complete list of possible return code values. The tcflow service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor does not describe a valid open file.
EINTR	A signal interrupted the call.
EINVAL	The Action parameter does not contain one of the expected values.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENOTTY	File_descriptor is not associated with a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflow service stores the reason code. The tcflow service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The following table defines the processing of the **SIGTTOU** signal when the tcflow service is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted" on page 824
- "tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal" on page 829
- "tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal" on page 840

Characteristics and restrictions

There are no restrictions on the use of the tcflow service.

Examples

For an example using this callable service, see "BPX1TFW (tcflow) example" on page 1204.

tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal

Function

The tcflush callable service flushes all data that is sent to a device. Depending on the value of the Queue_selector parameter, any data written, but not sent, or any data received, but not read, is discarded.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TFH):	31-bit
AMODE (BPX4TFH):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TFH,(File_descriptor,
              Queue_selector,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TFH with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal.

Queue_selector
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that specifies the queues that are to be flushed. The constants are mapped in the BPXYTIOS macro; see “BPXYTIOS — Map the termios structure” on page 1065.

Constant	Description
TCIFLUSH	Flush data received but not read
TCOFLUSH	Flush data written but not sent

Constant	Description
TCIOFLUSH	Flush both data received but not read and data written but not sent

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflush service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflush service stores the return code. The tcflush service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The tcflush service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is not a valid open file descriptor.
EINTR	A signal interrupted the call.
EINVAL	The Queue_selector specified was incorrect.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENOTTY	The file that is associated with the file descriptor is not a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcflush service stores the reason code. The tcflush service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The following table defines the processing of the **SIGTTOU** signal when tcflush is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted" on page 824
- "tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal" on page 826
- "tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal" on page 840

Characteristics and restrictions

There are no restrictions on the use of the tcflush service.

Examples

For an example using this callable service, see "BPX1TFH (tcflush) example" on page 1204.

tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal

Function

The tcgetattr callable service gets control information for a terminal and stores it in the specified Termios_structure.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, state any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TGA):	31-bit
AMODE (BPX4TGA):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TGA,(File_descriptor,
              Termios_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGA with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal for which you want attributes.

Termios_structure
Returned parameter

Type: Structure

Length:
Specified by BPXYTIOS#LENGTH in the BPXYTIOS macro

The name of an area into which the function is to return the terminal information. Termios_structure is mapped by the BPXYTIOS macro. This structure contains the control modes, input modes, output modes, local modes, and special control characters as defined by the POSIX standard (see "BPXYTIOS — Map the termios structure" on page 1065).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetattr service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetattr service stores the return code. The tcgetattr service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcgetattr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is not a valid open file descriptor.
ENOTTY	The file that is associated with the file descriptor is not a terminal; the process does not have a controlling terminal; or the file is not the controlling terminal for the process.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetattr service stores the reason code. The tcgetattr service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The BPXYTIOS macro should be used to map the termios structure and define the equates for bits and values. Note the following about BPXYTIOS:
   - BPXYTIOS generates standard POSIX-defined names, except that all names are uppercase. In addition, all names can have a user-specified prefix.
   - When testing or setting bits in flag fields, you should use an offset name to define which byte in the flag field contains the bit. For instance: TM C_CFLAG+HUPCL_O,HUPCL.
   - CS5 through CS8 values can be contained in CSIZE. CSIZE is essentially a 2-bit integer that can contain decimal values `0` through `3`, as defined by CS5 through CS8.
   - BPXYTIOS can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
   - The C_CC field is an array of 1-byte fields, indexed by the various special character equates. These equates can be used as offsets into C_CC, or can be put into a register to be used with indexing instructions. For instance:

```
MVC   C_CC+VSUSP,NEWVAL    To set a new value
LA    R10,VSUSP            To set an register to use as an index
                              in a later IC or STC instructions
```

2. You can run the tcgetattr service in either a foreground or a background process. However, if the process is in the background, a foreground process can later change the attributes that you obtained.

Related services

"tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842

Characteristics and restrictions

There are no restrictions on the use of the tcgetattr service.

Examples

For an example using this callable service, see "BPX1TGA (tcgetattr) example" on page 1204.

tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names

Function

The tcgetcp callable service gets the terminal session code page names and Code Page Change Notification (CPCN) capability.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key

Operation	Environment
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TGC):	31-bit
AMODE (BPX4TGC):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TGC,(File_descriptor,
              Termcp_length,
              Termcp_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGC with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal for which you want to get the code page names and data conversion environment.

Termcp_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the length of the Termcp_structure. The Termcp_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See "BPXYTCCP — Map the terminal control code page structure" on page 1058.

Termcp_structure
Returned parameter

Type: Structure

Length:
Specified by Termcp_length.

The name of an area where the tcgetcp service returns the Termcp_structure. The Termcp_structure is mapped by the BPXYTCCP macro. See "BPXYTCCP — Map the terminal control code page structure" on page 1058.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetcp service returns one of the following:

- 1, if the terminal device supports a capability of *forward code page names only*
- 2, if the terminal device supports a capability of *forward code page names and tables*
- -1, if the request is not successful

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetcp service stores the return code. The tcgetcp service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The tcgetcp service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is an incorrect open file descriptor.
EINVAL	One of the parameters contains a value that is not correct. Consult Reason_Code returned to determine the exact reason the error occurred.
ENODEV	One of the following error conditions exists: • The terminal device driver does not support CPCN functions. • CPCN functions have not been enabled. For a pseudoterminal device file, issue the tcsetcp (BPX1TSC, BPX4TSC) callable service against the master pty first, to enable CPCN support.
ENOTTY	The file that is associated with the file descriptor is not a terminal device.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetcp service stores the reason code. The tcgetcp service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. For terminal devices that support *forward code page names only* CPCN capability, use the tcsetcp (BPX1TSC, BPX4TSC) callable service to change the terminal session data conversion environment.

   The pseudoterminal device driver supports this CPCN capability.

2. For terminal devices that support *forward code page names and tables* CPCN capability, use the tcsettables (BPX1TST, BPX4TST) callable service to change the terminal session code conversion environment.

   The OCS remote-tty device driver supports this CPCN capability.
3. In the returned Termcp_structure, if the TCCPBINARY flag is set, the code page names should not be used. BINARY indicates that the data conversion point is to perform no data conversion for the terminal session.
4. For pseudoterminal support, the tcsetcp (BPX1TSC, BPX4TSC) callable service must be against the pty master terminal device for CPCN functions to be enabled.
5. In the returned Termcp_structure, if the TCCPFASTP flag is set, the data conversion that is specified by the source and target code page names can be performed locally to the data conversion application. This is valid any time that a table-driven conversion can be performed. For example, the data conversion point (application) could use the z/OS UNIX *iconv()* service to build local data conversion tables and perform all data conversion using the local tables, instead of using *iconv()* all in subsequent conversions. This provides for better-performing data conversion.
6. The BPXYTCCP macro should be used to map the Termcp_structure and define the equates for the flag byte values. Note the following about BPXYTCCP:
   - BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME should be terminated by a NUL (X'00') character.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME are case sensitive.

Related services

- "tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names" on page 845
- "tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables" on page 852

Characteristics and restrictions

The tcgetcp service is supported by the pseudoterminal and (OCS) remote terminal device drivers.

Examples

For an example using this callable service, see "BPX1TGC (tcgetcp) example" on page 1204.

tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID

Function

The tcgetpgrp callable service gets the process group ID of the foreground process group that is associated with a terminal, which is identified by its file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TGP):	31-bit
AMODE (BPX4TGP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TGP,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGP with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetpgrp service returns the process group ID of the foreground process group that is associated with the terminal, if the request is successful; or -1, if it is not successful. If there is no foreground process group, a positive value, not equal to any existing process group, is returned.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetpgrp service stores the return code. The tcgetpgrp service returns Return_code only if Return_value is -1. For a

complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcgetpgrp service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor parameter does not specify a valid open file descriptor.
ENOTTY	The file descriptor is not associated with a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetpgrp service stores the reason code. The tcgetpgrp service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702
- "tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID" on page 849

Characteristics and restrictions

There are no restrictions on the use of the tcgetpgrp service.

Examples

For an example using this callable service, see "BPX1TGP (tcgetpgrp) example" on page 1205.

tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal

Function

The tcgetsid callable service obtains the process group ID of the session leader that is associated with the terminal that is specified by the file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TGS):	31-bit
AMODE (BPX4TGS):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TGS,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TGS with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetsid service returns the process group ID associated with the terminal if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetsid service stores the return code. The tcgetsid service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The tcgetsid service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The File_descriptor parameter is not associated with a controlling terminal.
EBADF	The File_descriptor parameter does not specify a valid file descriptor.
ENOTTY	The file descriptor is not associated with a terminal device.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcgetsid service stores the reason code. The tcgetsid service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Characteristics and restrictions

There are no restrictions on the use of the tcgetsid service.

tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal

Function

The tcsendbreak callable service sends a BREAK signal to a terminal that uses asynchronous serial data transmission.

If the target terminal is an OCS-attached serial terminal, the BREAK signal is sent to the terminal. If the target terminal is a pseudoterminal (pty), control returns without any significant action.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TSB):	31-bit
AMODE (BPX4TSB):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TSB,(File_descriptor,
              Duration,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSB with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal device to which the break is to be sent.

Duration
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the duration of the **BREAK** transmission. If the target terminal is a pseudoterminal, the Duration parameter has no effect.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsendbreak service returns `0` if the request is successful, or `-1` if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsendbreak service stores the return code. The tcsendbreak service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcsendbreak service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is not a valid open file descriptor.
EINTR	The tcsendbreak (BPX1TSB, BPX4TSB) service was called from a background job, and the **SIGTTOU** signal either had default action or a signal handler. The function was not performed.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENOTTY	File_descriptor is not associated with a terminal.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsendbreak service stores the reason code. The tcsendbreak service returns Reason_code only if Return_value is -1.

Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The following table defines the processing of the SIGTTOU signal when tcsendbreak is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted" on page 824
- "tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal" on page 826
- "tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal" on page 829

Characteristics and restrictions

There are no restrictions on the use of the tcsendbreak service.

Examples

For an example using this callable service, see "BPX1TSB (tcsendbreak) example" on page 1206.

tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal

Function

The tcgetattr callable service sets control information for a terminal from the specified Termios_structure.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TSA):	31-bit
AMODE (BPX4TSA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TSA,(File_descriptor,
              Actions,
              Termios_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSA with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal for which attributes are to be set.

Actions
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains indicators that control the setting of the attributes. The following possible values are defined in the BPXYTIOS macro; see "BPXYTIOS — Map the termios structure" on page 1065.

Constant	Description
TCSANOW	Change the terminal attributes immediately.
TCSADRAIN	Change the terminal attributes when all output to the terminal has been sent.
TCSAFLUSH	Change the terminal attributes when all output to the terminal has been sent; and all input that has been received, but not read, is to be discarded.

Termios_structure
Supplied parameter

Type: Structure

Length:
Specified by BPXYTIOS#LENGTH in the BPXYTIOS macro

The name of an area that contains the attributes that are to be set. Termios_structure is mapped by the BPXYTIOS macro. This structure contains the control modes, input modes, output modes, local modes, and special control characters. For the layout of the Termios_structure, see "BPXYTIOS — Map the termios structure" on page 1065.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetattr service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetattr service stores the return code. The tcsetattr service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcsetattr service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is an incorrect open file descriptor.
EINTR	A signal interrupted the call.
EINVAL	An action or value that was specified was incorrect.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENOTTY	The file that is associated with the file descriptor is not a terminal.
EPERM	A change was made that is not permitted from a slave pty. See "Characteristics and restrictions" on page 845.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetattr service stores the reason code. The tcsetattr service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A program should always issue the tcsetattr callable service using a termios structure that was returned from a previous call to tcgetattr (BPX1TGA, BPX4TGA) (see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831), with appropriate changes to the various fields.
2. The BPXYTIOS macro should be used to map the termios structure and define the equates for bits and values. Note the following about BPXYTIOS:
   - BPXYTIOS generates standard POSIX-defined names, except that all names are uppercase. In addition, all names can have a user-specified prefix.
   - When testing or setting bits in flag fields, you should use an offset name to define which byte in the flag field contains the bit. For instance: TM C_CFLAG+HUPCL_O,HUPCL.

- CS5 through CS8 values can be contained in CSIZE. CSIZE is essentially a 2-bit integer that can contain decimal values 0 through 3, as defined by CS5 through CS8.
- BPXYTIOS can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
- The C_CC field is an array of 1-byte fields, indexed by the various special character equates. These equates can be used as offsets into C_CC, or can be put into a register to be used with indexing instructions. For instance:

```
MVC   C_CC+VSUSP,NEWVAL    To set a new value
LA    R10,VSUSP            To set a register to use as an index
                             in a later IC or STC instruction
```

3. The following table defines the processing of the SIGTTOU signal when the tcsetattr (BPX1TSA, BPX4TSA) service is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The SIGTTOU signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The SIGTTOU signal is not sent. The function continues normally.

Related services

- "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831

Characteristics and restrictions

- The slave pty cannot set the PACKET, PKTXTND, or PKT3270 bits.
- Neither the slave nor the master pty can set the PTU3270 bit if PKT3270 is not on.
- The master pty cannot set the PKT3270 bit unless PKRXTND is also on.

Examples

For an example using this callable service, see "BPX1TSA (tcsetattr) example" on page 1206.

tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names

Function

The tcsetcp callable service sets the terminal session code page names to the specified values.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TSC):	31-bit
AMODE (BPX4TSC):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TSC,(File_descriptor,
              Termcp_length,
              Termcp_structure,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSC with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal for which the code page names are to be set.

Termcp_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the length of the Termcp_structure. The Termcp_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See “BPXYTCCP — Map the terminal control code page structure” on page 1058.

Termcp_structure
Supplied parameter

Type: Structure

Length:
Specified by Termcp_length

The name of an area that contains the code page information to be set. This structure contains the source (ASCII) code page name, target (EBCDIC) code page name, and control flags. The Termcp_structure is mapped by the BPXYTCCP macro (see “BPXYTCCP — Map the terminal control code page structure” on page 1058).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetcp service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetcp service stores the return code. The tcsetcp service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The tcsetcp service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is an incorrect open file descriptor.
EINTR	A signal interrupted the call.
EINVAL	One of the parameters contains a value that is not correct. Consult Reason_Code returned to determine the exact reason the error occurred.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENODEV	One of the following error conditions exists: • CPCN functions have not been enabled. tcsetcp must be issued against the master pty before any CPCN function can be issued against the slave pty. • The terminal device driver does not support the *forward code page names only* CPCN capability.
ENOTTY	The file that is associated with the file descriptor is not a terminal device.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetcp service stores the reason code. The tcsetcp service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Attention: Use this service carefully. By changing the code pages for the data conversion, you may cause unpredictable behavior in the terminal session if the actual data used for the session is not encoded to the specified source (ASCII) and target (EBCDIC) code pages.

1. Use the tcsetcp callable service to send new code page names to the terminal session data conversion point to change the data conversion environment.

The tcsetcp callable service is used with terminal devices that support the *forward code page names only* CPCN capability. Use the tcgetcp (BPX1TGC, BPX4TGC) callable service to determine the terminal device CPCN capability.

2. The BPXYTCCP macro should be used to map the Termcp_structure and define the equates for the flag byte values. Note the following about BPXYTCCP:
   - BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME must be terminated by a NUL (X'00') character.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME are case sensitive.
3. The tcsetcp callable service is supported by the pseudoterminal (pty) device driver. For terminal sessions that use pty support, the data conversion point is the application that uses the master pty. An example data conversion point is the **rlogin** server. Here, **rlogin** uses CPCN functions to change the ASCII source or EBCDIC target code pages to use in its data conversion for the terminal session.

   During its processing of the tcsetcp service, the pty device driver applies the new code page names once the pty outbound data queue is drained. When this occurs, the pty input data queue is also flushed, and a TIOCXPKT_CHCP packet exception event is generated (if extended packet mode is enabled) to notify the master pty application that the code page names have been changed. The master pty application can then use the tcgetcp (BPX1TGC, BPX4TGC) callable service to retrieve the new code page names and establish the new data conversion environment.

   The tcsetcp service is supported by both the master and slave pty device drivers. However, CPCN functions must first be enabled by the application that uses the master pty; enabling CPCN functions is performed by the system during the initial tcsetcp invocation against the master pty device. When the tcsetcp invocation is performed against the master pty it may be subsequently issued against the slave pty.
4. The data conversion for a terminal session is performed on a session (terminal file) basis. If you change the data conversion characteristics for one file descriptor, the new data conversion applies to all open file descriptors that are associated with this terminal file.
5. Use the tcsetcp callable service to notify the data conversion point to stop data conversion. This is done by setting the TCCPBINARY flag. If this flag is set, the source and target code page names (TCCPSRCNAME and TCCPTRGNAME, respectively) are not changed from their current values.

   Attention: Use this option carefully. When the data conversion is disabled, the z/OS shell cannot be used until the data conversion is reenabled, using valid code pages for the terminal session.
6. Use the TCCPFASTP flag to indicate to the data conversion point (such as **rlogin**) that the data conversion that is specified by the source and target code page names can be performed locally to the application. This is valid any time that a table-driven conversion can be performed. For example, the data conversion point (application) could use the **iconv()** command to build the local data conversion tables and perform all data conversion using the local tables, instead of using **iconv()** in subsequent conversions. This provides for better-performing data conversion.
7. The following table defines the processing of the **SIGTTOU** signal when the tcsetcp service is called from a background process group against its controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names" on page 833
- "tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables" on page 852

Characteristics and restrictions

The tcsetcp service is supported by the pseudoterminal device driver.

Examples

For an example using this callable service, see "BPX1TSC (tcsetcp) example" on page 1207.

tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID

Function

The tcsetpgrp callable service moves the requested process group into the foreground, replacing the current foreground process group. The current foreground process group then becomes a background process group.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TSP):	31-bit
AMODE (BPX4TSP):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TSP,(File_descriptor,
              Process_group_id,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TSP with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal device.

Process_group_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the process group ID that is to be associated with the controlling terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetpgrp service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetpgrp service stores the return code. The tcsetpgrp service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcsetpgrp service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is not a valid open file descriptor.
EINTR	A signal interrupted the function.
EINVAL	Process_group_ID is not a process group ID that is supported by this implementation.
ENOTTY	The calling process does not have a controlling terminal; File_descriptor is not associated with the controlling terminal; or the controlling terminal is no longer associated with the session of the calling process.
EPERM	Process_group_ID does not match the process group ID of any process in the same session as the calling process.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsetpgrp service stores the reason code. The tcsetpgrp service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The tcsetpgrp service moves the requested process group into the foreground, replacing the current foreground process group. The current foreground process group then becomes a background group. This terminal must be the controlling terminal of the calling process, and it must be currently associated with the session of the calling process. Process_group_ID must represent a process group in the same session as the calling process.
2. After the foreground process group is set, reads by the process group that was formerly in the foreground fail or cause the process group to stop from a **SIGTTIN** signal. Writes can also cause the process to stop (from a **SIGTTOU** signal), or they can succeed, depending upon the current setting of TOSTOP (from tcsetattr) and the signal options for **SIGTTOU**.
3. The system issues a **SIGTTOU** signal when tcsetpgrp() is issued from a background process, unless SIGTTOU is being ignored or blocked. If the signal is set to default processing (SIG_DFL), the process group is stopped. If there is a handler, the handler gets control and errno=EINTR is returned.
4. The File_descriptor parameter that is specified can be any of the descriptors that represent the controlling terminal (such as standard input [**stdin**], standard output [**stdout**], and standard error [**stderr**]). The service affects future access from any file descriptor in use for the terminal.

   Note: You must consider redirection when choosing the file descriptor to specify.
5. The following table defines the processing of the **SIGTTOU** signal when the tcsetpgrp service is called from a background process against a controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.
Ignored or blocked	The **SIGTTOU** signal is not sent. The function continues normally.

Related services

- "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686
- "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702
- "tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID" on page 836

Characteristics and restrictions

There are no restrictions on the use of the tcsetpgrp service.

Examples

For an example using this callable service, see "BPX1TSP (tcsetpgrp) example" on page 1207.

tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables

Function

The tcsettables callable service sets the terminal session code page names and conversion tables to the specified values.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TST):	31-bit
AMODE (BPX4TST):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TST,(File_descriptor,
              Termcp_length,
              Termcp_structure,
              Srctable,
              Trgtable,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TST with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the terminal for which the code page names and data conversion tables are to be set.

Termcp_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword that contains the length of the Termcp_structure. The Termcp_structure is mapped by BPXYTCCP, and has a length of TCCP#LENGTH. See "BPXYTCCP — Map the terminal control code page structure" on page 1058.

Termcp_structure
Supplied parameter

Type: Structure

Length:
Specified by Termcp_length.

The name of an area that contains the code page information that is to be set. This structure contains the source (ASCII) code page name, target (EBCDIC) code page name, and control flags. The Termcp_structure is mapped by the BPXYTCCP macro (see "BPXYTCCP — Map the terminal control code page structure" on page 1058).

Srctable
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
256 bytes

The name of a field that contains a 256-byte data conversion table for the source-to-target (ASCII to EBCDIC) data conversion. The byte offset into this table corresponds to the character code from the source (ASCII) code page. The data value at each offset is the *converted* target (EBCDIC) character code.

Trgtable
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
256 bytes

The name of a field that contains a 256-byte data conversion table for the target-to-source (EBCDIC to ASCII) data conversion. The byte offset into this table corresponds to the character code from the target (EBCDIC) code page. The data value at each offset is the *converted* source (ASCII) character code.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsettables service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsettables service stores the return code. The tcsettables service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The tcsettables service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	File_descriptor is an incorrect open file descriptor.
EINTR	A signal interrupted the call.
EINVAL	One of the following error conditions exists: • The value of *Termcp_length* was not valid. • An incorrect combination of multi-byte code page names was specified in the Termcp_structure. One of the following applies: – The source code page that was specified in TCCPSRCNAME specified a supported ASCI multi-byte code page, and the TCCPTRGNAME did not specify a supported EBCDIC multi-byte code page. – The target code page that was specified in TCCPTRGNAME specified a supported EBCDIC multi-byte code page, and the TCCPSRCNAME did not specify a supported ASCII multi-byte code page.
EIO	The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU.
ENODEV	The terminal device driver does not support the *forward code page names and tables* CPCN capability.
ENOTTY	The file that is associated with the file descriptor is not a terminal device.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the tcsettables service stores the reason code. The tcsettables service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Attention: Use this service carefully. By changing the code pages for the data conversion, you may cause unpredictable behavior in the terminal session if the actual data that is used for the session is not encoded to the specified source (ASCII) and target (EBCDIC) code pages.

1. Use the tcsettables callable service to send new code page names and data conversion tables to the terminal session data conversion point to change the data conversion environment.

   The tcsettables callable service is used with terminal devices that support the *forward code page names and tables* CPCN capability. Use the tcgetcp (BPX1TGC, BPX4TGC) callable service to determine the terminal device CPCN capability.
2. The BPXYTCCP macro should be used to map the Termcp_structure and define the equates for the flag byte values. Note the following about BPXYTCCP:
   - BPXYTCCP can be used to define either a DSECT or an inline structure. This is determined by the DSECT= keyword.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME must be terminated by a NUL (X'00') character.
   - The code page names that are contained in TCCPSRCNAME and TCCPTRGNAME are case sensitive.
3. The OCS remote-tty (rty) device driver supports this function. For OCS terminal sessions, the data conversion is performed by OCS outboard on the AIX® server system. Use the tcsettables service to specify new code pages and conversion tables that are to be used in the data conversion.

   During its processing of the tcsettables service, the OCS rty device driver applies the new code page names when the outbound data queue is drained. When this occurs, the rty input data queue is also flushed, and the new conversion environment takes effect.

   The *Srctable* and *Trgtable* parameters are used as follows:
   - If the code page names that are specified in the Termcp_structure are for supported double-byte data conversion the *SrcTable* and *TrgTable* arguments are not used. The following double-byte translation is supported for OCS sessions:

Source (ASCII) code page	Target (EBCDIC) code page
IBM-eucJP	IBM-939
IBM-932	IBM-939

   - If TCCPSRCNAME specifies **ISO8859-1** and TCCPTRGNAME specifies **IBM-1047**, OCS uses its own data conversion tables and the *Srctable* and *Trgtable* parameters are not used.
   - Otherwise the conversion tables in *Srctable* and *Trgtable* are used.
4. The data conversion for a terminal session is performed on a session (terminal file) basis. If you change the data conversion characteristics for one file descriptor, the new data conversion applies to all open file descriptors that are associated with this terminal file.
5. Use the tcsettables callable service to notify the data conversion point to stop data conversion. This is done by setting the TCCPBINARY flag. If this flag is set, the source and target code page names (TCCPSRCNAME and TCCPTRGNAME, respectively) are not changed; the Srctable and Trgtable parameters are not used.

   Note: Use this option carefully. When the data conversion is disabled, the z/OS shell cannot be used until the data conversion is reenabled, using valid code pages for the terminal session.
6. The TCCPFASTP flag is not used by the OCS rty device driver. The value of this flag has no effect and is ignored.

7. The following table defines the processing of the **SIGTTOU** signal when the tcsettables service is called from a background process group against its controlling terminal:

SIGTTOU processing	Expected behavior
Default or signal handler	The **SIGTTOU** signal is generated. The function is not performed. Return_value is set to -1, and Return_code is set to EINTR.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1TST (tcsettables) example" on page 1207.

times (BPX1TIM, BPX4TIM) — Get process and child process times

Function

The times callable service gathers information about processor time used by the current process or related processes.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TIM):	31-bit
AMODE (BPX4TIM):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TIM,(Time_data,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TIM with the same parameters.

Parameters

Time_data
Returned parameter

Type: Structure

Length:
16 bytes

The name of a data area where the times service returns information about processor time used. This field is mapped by the BPXYTIMS macro. For the structure of the data area, see "BPXYTIMS — Map the response structure for times" on page 1064.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the service places its return value. The value that is returned is the number of clock ticks (hundredths of a second) that have elapsed since the current address space was last dubbed a process. If this value cannot be determined, the service returns -1.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the times service stores the return code. The times service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The times service can return one of the following values in the Return_code parameter:

Return_code	Explanation
ERANGE	An overflow occurred while time values were being computed.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the times service stores the reason code. The times service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

Processor times for a child process that has ended are not added to the TIMSCUTIME and TIMSCSTIME of the parent process until the parent issues a wait or waitpid for that child process. See "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882 for more information about this subject.

Related services

- "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132
- "execmvs (BPX1EXM, BPX4EXM) — Run an MVS program" on page 144

- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources" on page 418
- "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882

Characteristics and restrictions

There are no restrictions on the use of the times service.

Examples

For an example using this callable service, see "BPX1TIM (times) example" on page 1205.

MVS-related information

The TIMSSTIME value that is returned by the times service is a portion of the total TCB time of the calling process—the portion that was spent processing z/OS UNIX services in the kernel address space. This TCB time is accumulated from the most recent time the MVS address space was dubbed a process (made eligible to issue z/OS UNIX callable services).

The TIMSUTIME value consists of the total processor time that has been accumulated by the calling address space in the current job-step. This includes all job step TCB and SRB time that was accumulated before the address space became a process, all SRB time that was accumulated after the address space became a process, and all TCB time that was accumulated after the address space became a process, except for the TCB time that was accumulated while the process was running in the kernel. The value of TIMSUTIME can be calculated as follows:

```
TIMSUTIME = <job-step SRB time> + <job-step TCB time> - TIMSSTIME
```

Note:

1. An MVS address space can be dubbed a process, undubbed (no longer a process), and then dubbed a process again in the same job step. The TIMSSTIME value for the address space in this case reflects only the kernel TCB time since the address space was last dubbed. The TIMSUTIME value, however, reflects TCB and SRB time for the entire life of the job step.
2. The exec service (BPX1EXC, BPX4EXC) and the execmvs service (BPX1EXM, BPX4EXM) cause new substeps in the current address space. Address-space-level processor time counters (in the address space control block) are reset. As long as the address space remains a process, the values from previous substeps are retained, and are included in values that are returned by the times service. However, if the mvsprocclp service (BPX1MPC, BPX4MPC) is invoked to undub the process after the exec or execmvs service has been issued, subsequent invocations of the times service return processor times starting at the beginning of the new substep.
3. The times service reports an approximation of the usage by the system, and is not a completely accurate representation of the time used on behalf of the system and by the user. The function guarantees that the user time reported is ever increasing; it does not do the same for the system time.

truncate (BPX1TRU, BPX4TRU) — Change the size of a file

Function

The truncate service changes the size of a file. The file is identified by a path name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TRU):	31-bit
AMODE (BPX4TRU):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TRU,(Pathname_length,
              Pathname,
              File_length,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TRU with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the pathname of the file whose size is to be changed.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the pathname of the file. This field has the length that is specified in Pathname_length.

Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

File_length
Supplied parameter

Type: Integer

Length:
Doubleword

The name of a doubleword that contains the number of bytes that are to be contained in the file after the size is changed.

This field is a doubleword to accommodate large files. For normal processing with a singleword value, the second word should be zero. The truncate service accepts only positive values.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the truncate service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the truncate service stores the return code. The truncate service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The truncate service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	The calling process does not have permission to search some component of the Pathname prefix; or write permission is denied on the file.
EBUSY	The file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
EFBIG	The File_length parameter is greater than the maximum file size limit for the process. The following reason code can accompany the return code: JRWriteBeyondLimit.
EINVAL	The file is not a regular file; or the File_length that is specified is either negative or greater than the maximum file size. The following reason codes can accompany the return code: JRTrNegOffset, JRTrNotRegFile.
EISDIR	The file is a directory.

Return_code	Explanation
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters, or a component of the pathname is longer than 255 characters.
ENOENT	No file named Pathname was found, or no pathname was specified. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of the Pathname prefix is not a directory.
EROFS	The specified file is on a read-only file system. The following reason code can accompany the return code: JRTrMountedRO.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the truncate service stores the reason code. The truncate service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The truncate service changes the file size to File_length bytes, beginning from the first byte of the file. If the file was originally larger than File_length bytes, the data from File_length to the original end of the file is removed. If the file was originally shorter than File_length, bytes between the old and new lengths are read as zeros.
2. If File_length is greater than the soft file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.
3. Full blocks are returned to the file system, so that they can be used again.
4. A file may not be truncated if it is currently open by a remote NFS client with a share reservation that prevents the file from being opened for writing. Refer to "open (BPX1OPN, BPX4OPN) — Open a file" on page 447 for details about the NFS share reservations.

Related services

- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203

Characteristics and restrictions

The file that is specified must be a regular file to which the calling process has write access.

Examples

For an example using this callable service, see "BPX1TRU (truncate) example" on page 1206.

ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal

Function

The ttyname callable service obtains the pathname of the terminal that is associated with the file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1TYN):	31-bit
AMODE (BPX4TYN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1TYN,(File_descriptor,
              Terminal_name_length,
              Terminal_name)
```

AMODE 64 callers use BPX4TYN with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor.

Terminal_name_length
Parameter supplied and returned

Type: Integer

Length:
Fullword

The name of a fullword that contains the size, in bytes, of the buffer that is referred to by Terminal_name. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the terminal_name could be on output.

Terminal_name
Returned parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Terminal_name_length parameter

The name of an area to which ttyname returns either the pathname of the terminal, terminated by a X'00', or a single byte of X'00' (null string), if the file descriptor is not valid or does not represent a terminal.

The length of Terminal_name should be 1024 bytes long (PATH_MAX+1), unless the pathname is known to be shorter.

Usage notes

1. This service does not return -1 to indicate a failure. If the file descriptor is incorrect, it returns a null string.
2. If Terminal_name is an area smaller than the actual pathname of the terminal, the name is truncated.

Related services

- "ttyname (BPX2TYN, BPX4TYN) (X/Open version) — Get the name of a terminal"
- "isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal" on page 301
- "isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal" on page 303

Characteristics and restrictions

There are no restrictions on the use of the ttyname service.

Examples

For an example using this callable service, see "BPX1TYN (ttyname) example" on page 1208.

ttyname (BPX2TYN, BPX4TYN) (X/Open version) — Get the name of a terminal

Function

The ttyname callable service obtains the pathname of the terminal that is associated with the file descriptor.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX2TYN):	31-bit
AMODE (BPX4TYN):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked

Operation	Environment
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX2TYN,(File_descriptor,
              Terminal_name_length,
              Terminal_name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4TYN with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor.

Terminal_name_length
Parameter supplied and returned

Type: Integer

Length:
Fullword

The name of a fullword that contains the size, in bytes, of the buffer that is referred to by Terminal_name. The size of this field should be less than 4096 bytes (4KB) in length. The size of the buffer that is specified should be the maximum length that the terminal_name could be on output.

Terminal_name
Returned parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Terminal_name_length parameter

The name of an area to which ttyname returns either the pathname of the terminal, terminated by a X'00', or a single byte of X'00' (null string), if the file descriptor is not valid or does not represent a terminal.

The length of Terminal_name should be 1024 bytes long (PATH_MAX+1), unless the pathname is known to be shorter.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ttyname service returns `0` if the request is successful, or -1 if it is not successful.

`Return_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ttyname service stores the return code. The ttyname service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The ttyname service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The File_descriptor argument is not a valid open file descriptor.
ENOTTY	The File_descriptor argument is not associated with a terminal.

`Reason_code`
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the ttyname service stores the reason code. The ttyname service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This version of ttyname is XPG4 compliant.
2. If Terminal_name is an area smaller than the actual pathname of the terminal, the name is truncated.

Related services

"isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal" on page 301

Characteristics and restrictions

There are no restrictions on the use of the ttyname service.

Examples

For an example using this callable service, see "BPX2TYN (ttyname) example" on page 1208.

umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask

Function

The umask callable service changes the file mode creation mask of a process. The file mode creation mask is used by the security package to turn off permission bits in the mode parameter that is specified. Bit positions that are set in the file mode creation mask are cleared in the mode of the created file.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UMK):	31-bit
AMODE (BPX4UMK):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UMK,(File_mode_creation_mask,
              Return_value)
```

AMODE 64 callers use BPX4UMK with the same parameters.

Parameters

File_mode_creation_mask
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword that contains the file mode creation mask. This mask turns off permission bits in a file's mode. File_mode_creation_mask is mapped by the BPXYMODE macro (see "BPXYMODE — Map the mode constants of the file services" on page 996).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the umask service returns the previous value of the file mode creation mask. This fullword has the same mapping as the File_mode_creation_mask parameter.

Usage notes

1. The umask service changes the process's file creation mask. This mask controls file permission bits that are set whenever the process creates a file. File permission bits that are turned on in the file creation mask are turned off in the file permission bits of files that are created by the process. For example, if a call to the open (BPX1OPN, BPX4OPN) service specifies a "mode" argument with file permission bits, the process's file creation mask affects that argument: Bits that are on in the mask are turned off in the "mode" argument, and therefore in the mode of the created file.
2. Only the file permission bits of the new mask are used. For example, the type of file field in File_Mode cannot be masked.

Related services

- “mkdir (BPX1MKD, BPX4MKD) — Make a directory” on page 361
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 447

Characteristics and restrictions

There are no restrictions on the use of the umask service.

Examples

For an example using this callable service, see “BPX1UMK (umask) example” on page 1208.

umount (BPX1UMT, BPX4UMT) — Remove a virtual file system

Function

The umount callable service unmounts or remounts a virtual file system. That is, it removes a virtual file system from the file tree, or it remounts a virtual file system to the file tree.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UMT):	31-bit
AMODE (BPX4UMT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UMT,(File_system_name,
              Flags,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UMT with the same parameters.

Parameters

File_system_name
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
44 bytes

The name of a 44-character field that contains the file system that is to be unmounted. The file system name must be left-justified and padded with blanks.

Flags
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword binary field that contains the unmount options.

This field is mapped by the BPXYMTM macro. See "BPXYMTM — Map the modes for mount and unmount" on page 1000 for the contents of the macro.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the umount service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the umount service stores the return code. The umount service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The umount service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBUSY	Honoring the request would require unmounting a file system that is still in use. The following reason codes can accompany the return code: JRFileSysWasReset, JRFsForceUmount, JRFsMustReset, JRFsParentFs, JRFsUnmountInProgress, JRIsFsRoot, and JRQuiesced.
EINTR	The call was interrupted by a signal. The following reason code can accompany the return code: JRSigDuringWait.

Return_code	Explanation
EINVAL	An incorrect parameter was specified. The file system name is not the name of a file system; an incorrect combination of flags was specified; a umount drain or remount request was specified in a sysplex; or an umount force was specified before an immediate umount was attempted. The following reason codes can accompany the return code: JRFileSysNotThere, JRInvalidParms, JRMustUmountImmed, JRQuiescing, JRNotSupInSysplex, JrRemntMode.
EPERM	The calling process is not a superuser. The following reason code can accompany the return code: JRUserNotPrivileged.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the umount service stores the reason code. The umount service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. A file system that has file systems mounted on it can be remounted but cannot be unmounted. Before the file system can be can be unmounted, all children file systems must be unmounted first.
2. A reset request can stop only an umount service drain request. It has no effect if it is issued when there is no umount request outstanding. Currently, umount service drain requests are not supported in a sysplex environment. If such a request is issued in a sysplex, the following behavior is exhibited:
   - If there is no activity in the file system, the drain request performs the unmount, but it behaves like a umount normal. (Where a normal request specifies that if no user is accessing any of the files in the specified file system, the system processes the umount request. Otherwise, the system rejects the umount request. This is the default.)
   - If there is activity in the file system, the drain request returns a Return_value of -1, with Return_code EINVAL and Reason_code JRNotSupInSysplex.
3. A umount service request with no other options specified succeeds only if the unmount can be processed immediately. Otherwise, an EBUSY is returned.
4. MTMREMOUNT is specified to change the mount mode between read-only and read/write. If neither MTMRO nor MTMRDWR is specified, the mode is set to the opposite of its current state. If a mode is specified, it must be the opposite of the current state.
5. MTMSAMEMODE is specified to remount the file system without changing the mount mode. This function can be used to attempt to regain use of a file system that has had I/O errors. If MTMREAD or MTMRDWR is also specified, the mode specified must be the current mode.
6. Before a file system is remounted (using any method other than MTMSAMEMODE), any open FIFO files must be closed, or the remount attempt will be rejected with EINVAL, JrFIFIinFileSys.
7. If the file system that is to be unmounted is the root file system, the IMMED option must be specified.

Related services

"mount (BPX1MNT) — Make a file system available" on page 377

Characteristics and restrictions

In order to unmount a file system, the requester must be an authorized program, or must be running for a user with appropriate privileges (see "Authorization" on page 8).

Examples

See "BPX1UMT (umount) example" on page 1209 for an example using this callable service.

uname (BPX1UNA, BPX4UNA) — Obtain the name of the current operating system

Function

The uname callable service obtains information about the z/OS UNIX system the caller is running on.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UNA):	31-bit
AMODE (BPX4UNA):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UNA,(Data_area_length,
              Data_area_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UNA with the same parameters. The Data_area_address parameter is a doubleword.

Parameters

Data_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the data area that is pointed to by Data_area_address. The area must be at least the length of UTSN#LENGTH. For a mapping of this data area, refer to "BPXYUTSN — Map the response structure for uname" on page 1068.

Data_area_address
Returned parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of the area in which the system information is to be returned. For a mapping of this data area, refer to "BPXYUTSN — Map the response structure for uname" on page 1068.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the uname service returns a nonnegative value if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the uname service stores the return code. The uname service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The uname service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	The pointer to the UTSN from the invoker is bad. The following reason code can accompany the return code: JRBadAddress.
EINVAL	The passed length of the invoker UTSN is not valid. The following reason code can accompany the return code: JROK.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the uname service stores the reason code. The uname service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Characteristics and restrictions

There are no restrictions on the use of the uname service.

Examples

For an example using this callable service, see "BPX1UNA (uname) example" on page 1209.

unlink (BPX1UNL, BPX4UNL) — Remove a directory entry

Function

The unlink service removes a directory entry. A directory entry can be identified by a pathname to a file, a link name to a file, or a symbolic link.

If a link to a file is removed, and the link count becomes zero, and no other process has the file open, the file itself is deleted.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UNL):	31-bit
AMODE (BPX4UNL):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UNL,(Name_length,
              Name,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UNL with the same parameters.

Parameters

Name_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of Name.

Name
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Name_length parameter

The name of a field, of length Name_length, that contains the name of the directory entry that is to be removed. Name can be a pathname to a file, a link name to a file, or a symbolic link name. The pathname was specified when the file was created (see "open (BPX1OPN, BPX4OPN) — Open a file" on page 447). The link name was specified when a link to the file was created (see "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327), or when the symbolic link was created (see "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812).

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlink service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlink service stores the return code. The unlink service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The unlink service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following conditions occurred: • The calling process does not have permission to search some component of Pathname, or does not have write permission for the directory that contains the link that is to be removed. • The S_ISVTX flag is set for the parent directory of the file that is to be removed, and the caller is not the owner of the file or of the parent directory; nor does the caller have appropriate privileges (see "Authorization" on page 8).
EBUSY	The file cannot be unlinked because it is being used by the system or the file is open by a remote NFS client with a share reservation that conflicts with the requested operation.
EINVAL	The Name parameter is incorrect. It contains a null character.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Name argument. This error is issued if more than 24 symbolic links are detected in the resolution of Name.
ENAMETOOLONG	Name is longer than 1023 characters, or some component of the pathname is longer than 255 characters. Name truncation is not supported.

Return_code	Explanation
ENOENT	Name was not found, or no name was specified. The following reason code can accompany the return code: JRUnlNoEnt.
ENOTDIR	Some component of the pathname prefix is not a directory.
EPERM	Name refers to a directory. Directories cannot be removed using unlink. The following reason code can accompany the return code: JRUnlDir.
EROFS	The link that is to be removed is on a read-only file system. The following reason code can accompany the return code: JRUnlMountRO.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlink service stores the reason code. The unlink service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. If the name that is specified refers to a symbolic link, the symbolic link file that is named by Name is deleted.
2. If the unlink service request is successful and the link count becomes zero, the file is deleted. The contents of the file are discarded, and the space it occupied is freed for reuse. However, if another process (or more than one) has the file open or in use when the last link is removed, the file is not removed until the last process closes it.
3. When the unlink service is successful in removing the directory entry and decrementing the link count, whether or not the link count becomes zero, it returns control to the caller with Return_value set to `0`. It updates the change and modification times for the parent directory and the change time for the file itself (unless the file is deleted).
4. Directories cannot be removed using the unlink service. To remove a directory, refer to “rmdir (BPX1RMD, BPX4RMD) — Remove a directory” on page 615.
5. If the S_ISVTX flag is set for the parent directory of the file that is to be unlinked, one of the following conditions must be true, or the request will fail with EACCES:
   - The caller is the owner of the file to be unlinked
   - The caller is the owner of the parent directory
   - The caller has appropriate privileges (see “Authorization” on page 8)
6. A file may not be unlinked if it is currently open by a remote NFS client with a share reservation that would prevent the file from being opened for writing. Refer to “open (BPX1OPN, BPX4OPN) — Open a file” on page 447 for details about the NFS Share reservations.

Related services

- “close (BPX1CLO, BPX4CLO) — Close a file” on page 103
- “link (BPX1LNK, BPX4LNK) — Create a link to a file” on page 327
- “open (BPX1OPN, BPX4OPN) — Open a file” on page 447

- "rename (BPX1REN, BPX4REN) — Rename a file or directory" on page 607
- "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615

Characteristics and restrictions

There are no restrictions on the use of the unlink service.

Examples

For an example using this callable service, see "BPX1UNL (unlink) example" on page 1209.

unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal master/slave pair

Function

The unlockpt callable service unlocks the slave pseudoterminal device that is associated with the master to which the file descriptor refers.

Note: Because access to pseudoterminals is granted by changing ownership during the first slave open, and that caller must have the same UID as the master opener, neither the grantpt nor the unlockpt services are functionally required. They are provided in order to be compatible with XPG4, and for ported programs that may use them.

Requirements

Operation	**Environment**
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UPT):	31-bit
AMODE (BPX4UPT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UPT,(File_descriptor,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UPT with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for the terminal.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlockpt service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlockpt service stores the return code. The unlockpt service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The unlockpt service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Either a grantpt has not yet been issued, or an unlockpt has already been issued. An unlockpt must be issued after a grantpt, and can only be issued once.
EBADF	The File_descriptor parameter does not specify a file descriptor that is open for writing.
EINVAL	The file descriptor is not associated with a master pseudoterminal device.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unlockpt service stores the reason code. The unlockpt service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Characteristics and restrictions

There are no restrictions on the use of the unlockpt service.

Examples

For an example using this callable service, see "BPX1UPT (unlockpt) example" on page 1210.

unquiesce (BPX1UQS, BPX4UQS) — Unquiesce a file system

Function

The unquiesce callable service unquiesces a file system, making the files in it available for use again. The backup of the data in the file system is complete.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UQS):	31-bit
AMODE (BPX4UQS):	64-bit
ASC mode:	Primary mode
Serialization:	Enabled for interrupts
Locks:	No locks held
Control parameters:	All parameters addressable in Primary

Format

```
CALL BPX1UQS,(File_system_name,
              Unquiesce_parms,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UQS with the same parameters.

Parameters

File_system_name
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
44 bytes

The name of a 44-character field that contains the file system name. The name should be left-justified in the field and padded with blanks.

Unquiesce_Parms
Supplied parameter

Type: Structure

Length:
Fullword

The name of a fullword binary field that contains the unquiesce service options. This field is mapped by the BPXYMTM macro. Refer to "BPXYMTM — Map the modes for mount and unmount" on page 1000 for the unquiesce service options that are available.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unquiesce service returns 0 if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unquiesce service stores the return code. The unquiesce service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The unquiesce service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBUSY	The file system that was specified was not quiesced by this user, and force was not specified in Unquiesce_parms. The following reason code can accompany the return code: JRInvalidRequester.
EINVAL	An incorrect parameter was specified. Verify that only the force bit in Unquiesce_ parms was specified, that File_system_name is correct, and that File_system_name is for a quiesced file system. The following reason codes can accompany the return code: JRFileSysNotThere, JRInvalidParms, and JRNotQuiesced.
EPERM	The user cannot request this service, because it lacks the permission required to do so. The following reason code can accompany the return code: JRUserNotPrivileged.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the unquiesce service stores the reason code. The unquiesce service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

The unquiesce service makes a file system available for use again following a previous quiesce request.

Related services

"quiesce (BPX1QSE, BPX4QSE) — Quiesce a file system" on page 570

Characteristics and restrictions

1. In order to unquiesce a file system, the requester must be a superuser. This is the same authority that is required to mount or quiesce a file system.
2. In a sysplex, an unquiesce will result in the file system being mounted on any system that did not have the file system mounted at that time. This situation could occur if a system joined the sysplex during the period of time that the file system was in a quiesced state.

Examples

For an example using this callable service, see "BPX1UQS (unquiesce) example" on page 1210.

utime (BPX1UTI, BPX4UTI) — Set file access and modification times

Function

The utime callable service sets the access and modification times of a file.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1UTI):	31-bit
AMODE (BPX4UTI):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1UTI,(Pathname_length,
              Pathname,
              Newtimes,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4UTI with the same parameters. The Newtimes parameter is two doublewords.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the fully qualified name (pathname) of the file. The pathname can be up to 1023 bytes long.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field of length Pathname_length that contains the pathname of the file.

Pathnames can begin with or without a slash.

- A pathname that begins with a slash is an *absolute* pathname. The slash refers to the root directory, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a *relative* pathname. The search for the file starts at the working directory.

Newtimes
Supplied parameter

Type: Structure

Length:
Doubleword (two doublewords)

The name of a doubleword or two doublewords that contain the access and modification times for the file. The first fullword (doubleword) contains the new access time, and the second fullword (doubleword) contains the new modification time. These times can be retrieved with "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805 or "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196.

- Times are specified as the number of seconds that have elapsed between 00:00 a.m. on January 1, 1970, and the desired time. The times must be specified as nonnegative values other than -1 (see this topic for the special case of -1). AMODE 64 callers must specify each time as a doubleword. Times beyond the year 2038 require more than a fullword.
- In order to request that the current time be used for both access and modification times, specify X'FFFFFFFF' or X'FFFFFFFFFFFFFFFF' (-1) in either or both words (doublewords) of this field. The current time in the file's status is also updated.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the utime service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the utime service stores the return code. The utime service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The utime service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	One of the following is true: • The process does not have search permission for some component of the Pathname prefix. • Newtimes equals the current time; the effective ID does not match the file's owner; the process does not have write permission for the file; and the process does not have appropriate privileges (see "Authorization" on page 8).
EINVAL	The argument that was supplied is incorrect. The following reason code can accompany the return code: JRNegativeValueInvalid.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	The length of the pathname is greater than 1023, or some component of the fully qualified name is longer than 255 bytes. This could be as a result of encountering a symbolic link during resolution of Pathname, where the substituted string is longer than 1023 characters.
ENOENT	No file named Pathname was found; or Pathname was blank. The following reason code can accompany the return code: JRFileNotThere.
ENOTDIR	Some component of the pathname prefix is not a directory.
EPERM	The Newtimes value did not specify the current time; the effective user ID of the calling process does not match the owner of the file; and the calling process does not have appropriate privileges (see "Authorization" on page 8).
EROFS	Pathname is on a read-only file system. The following reason code can accompany the return code: JRReadOnlyFs.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the utime service stores the reason code. The utime service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Related services

- "fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor" on page 196
- "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805

Characteristics and restrictions

There are no restrictions on the use of the utime service.

Examples

For an example using this callable service, see "BPX1UTI (utime) example" on page 1210.

wait (BPX1WAT, BPX4WAT) — Wait for a child process to end

Function

The wait callable service obtains the status of a child process that has ended or stopped. You can use the wait service to obtain the status of a process that is being debugged with the ptrace facilities. The term *child* refers to children that are created by the fork service, as well as processes that are attached by ptrace.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1WAT):	31-bit
AMODE (BPX4WAT):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1WAT,(Process_ID,
              Options,
              Status_field_address,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4WAT with the same parameters. The Status_field_address parameter is a doubleword.

Parameters

Process_ID
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the event that the caller is waiting upon:

- A value greater than zero is assumed to be a process ID. The caller waits for the child or debugged process with that specific process ID to end or to stop.
- A value of zero specifies that the caller is waiting for any children or debugged processes with a process group ID equal to the caller's to end or to stop.
- A value of -1 specifies that the caller is waiting for any of its children or debugged processes to end or to stop.
- If the value is negative and less than -1, its absolute value is assumed to be a process group ID. The caller waits for any children or debugged processes with that process group ID to end or to stop.

Options
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the wait options for this invocation of the wait service. The wait options that are specified affect the actions that are taken by the wait service, as described in this topic. These options can be specified separately or in combination. A zero value for the wait options implies that the wait service performs its default processing; that is, it waits for a child process to end. The following flags defined in the BPXYCONS macro are the allowable wait options (see "BPXYCONS — Constants used by services" on page 952).

Constant	Description
WNOHANG	The wait service does not suspend execution of the calling process if status is not immediately available for one of the child processes that is specified by Process_ID.
WUNTRACED	The wait service also returns the status of any child processes that are specified by Process_ID that are stopped, and whose status has not yet been reported since they stopped. If this option is not specified, the wait service returns only the status of processes that end.
WCONTINUED	The wait service returns the status for any continued child process that is specified by Process_ID whose status has not yet been reported since it continued from a job control stop.

Status_field_address
Returned parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a fullword in which this service is to place the status value for the child process that ended or stopped. The status value can be analyzed with the status value map BPXYWAST. For a description of this mapping, see "BPXYWAST — Map the wait status word" on page 1069. The status value is returned only if status is available for a child or debugged process, and the address specified in this field is not zero.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the wait service returns the process ID of the child that the status information applied to, if the request is successful, or -1 if it is not successful, or 0 if WNOHANG was specified and there is at least one process whose status information is not available.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the wait service stores the return code. The wait service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The wait service can return one of the following values in the Return_code parameter:

Return_code	Explanation
ECHILD	The caller has no appropriate child process; that is, no child process whose status has not already been obtained through earlier calls to wait meets the criteria for waiting.
EFAULT	One of the parameters that was specified contained the address of a storage area that is not accessible to the caller. The following reason code unique to this service can accompany this return code: JRBadExitStatusAddr.
EINTR	The calling process received a signal before the completion of an event that would cause the wait service to return. The service was interrupted by a signal. In this case, the value that is contained in Status_field_address is undefined.
EINVAL	The value of the option is not valid.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the wait service stores the reason code. The wait service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. The wait service suspends execution of the calling thread until one of the requested child or debugged processes ends, or until it obtains information about the process that ended. If a child or debugged process has already ended, but its status has not been reported when wait is called, the routine immediately returns with that status information to the caller.

If the WUNTRACED option is specified, the foregoing also applies for stopped children and stopped debugged processes.

2. The wait service always returns status for stopped *debugged* processes, even if WUNTRACED is not specified.

   If status is available for one or more processes, the order in which the status is reported is unspecified.

3. If the wait service is invoked simultaneously from multiple threads within the same process, the following behavior should be noted:
   - When multiple threads issue a fork call followed by a call to the wait service to wait for any child process to end, the status that is received by each thread may not be the status of the child that was created by that thread. If a thread wishes to receive the status of the child that it created, the thread should specify the returned child Process Id when it calls the wait service to wait for the child process to end.
   - If the wait service is called from multiple threads requesting status for the same process, the thread that receives the status is not specified when the process ends or stops. The thread that does not receive the status is returned to with a return value of -1 and a return code of ECHILD.

   Note: A debugged process is one that is being monitored for debugging purposes with the ptrace service.

Related services

- "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150
- "fork (BPX1FRK, BPX4FRK) — Create a new process" on page 185
- "pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal" on page 468
- "ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging" on page 537

Characteristics and restrictions

There are no restrictions on the use of the wait service.

Examples

For an example using this callable service, see "BPX1WAT (wait) example" on page 1210.

wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children

Function

The wait-extension callable service allows the calling process to obtain status information for its child processes.

Requirements

Operation	Environment
Authorization:	Supervisor or problem state, any PSW key
Dispatchable unit mode:	Task

Operation	Environment
Cross memory mode:	PASN = HASN
AMODE (BPX1WTE):	31-bit
AMODE (BPX4WTE):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1WTE(Function_code,
             Idtype,
             Id,
             Stat_loc_ptr,
             Options,
             Info_area_ptr,
             Return_value,
             Return_code,
             Reason_code)
```

AMODE 64 callers use BPX4WTE with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

Function_code
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the function to perform. If the value is #wait3, the wait3() function is performed. If the value is #waitid, the waitid() function is performed. The constants #wait3 and #waitid are defined in macro BPXYCONS.

Idtype
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates what type of children to wait for. This parameter is valid only when Function_code is #waitid. It can be one of the following values:

Table 22. Idtypes

Idtype	Description
P_PID	waitid() waits for the child with a process ID that is equal to the value that is specified in the 'id' parameter.
P_PGID	waitid() waits for the child with a process group ID that is equal to the value that is specified in the 'id' parameter.

Table 22. Idtypes (continued)

Idtype	Description
P_ALL	waitid() waits for any children. The 'id' parameter is ignored.

The P_ constants are defined in the BPXYCONS macro.

Id Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that contains the process id or process group id of the children to wait for. This value is valid only when the function_code is #waitid. Together with Idtype, Id is used to determine which children are to be waited for.

Stat_loc_ptr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the address of a fullword in which this service is to place the status value for the child process whose status is available.

This parameter is valid only when the function_code is #wait3.

If the wait-extension service returns because the status of a child process is available, and if Stat_loc_Ptr is not a null pointer, information is stored in the location that is pointed to by Stat_loc_ptr. If this field is null, no information is returned. This area is mapped by BPXYWAST.

Options
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the wait options for this invocation of the wait-extension service.

If the function_code is #wait3, the possible values are as the same as the 'options' parameter for the wait (BPX1WAT, BPX4WAT) service.

If the function_code is #waitid, this parameter is used to specify which state changes to wait for. It is formed by ORing together one or more of the following flags:

Table 23. Options

Option	Description
WEXITED	Wait for child processes that have exited.
WSTOPPED	Return status for any child that has stopped upon receipt of a signal.

Table 23. Options (continued)

Option	Description
WCONTINUED	Return status for any child that has stopped and has been continued.
WNOHANG	Return immediately if there are no children to wait for.
WNOWAIT	Keep the process whose status is returned in the info_area_ptr parameter in a waitable state. This does not affect the state of the process; the process may be waited for again after this call completes.

These constants are defined in BPXYCONS.

Info_area_ptr
Supplied parameter

Type: Pointer

Length:
Fullword (doubleword)

If Function_code is #wait3, Info_area_ptr is the name of a fullword (doubleword) that contains the address of an rusage structure. If this field is null, no information is returned. The rusage structure is defined in macro BPXYRLIM.

If Function_code is #waitid, Info_area_ptr is the name of a fullword (doubleword) that contains the address of a siginfo_t structure. If the function returns because a child process was found that satisfied the conditions that were indicated by the arguments Idtype and Options, the structure that is pointed to by info_area_ptr is filled in by the system with the status of the process. If this field is null, no information is returned. The siginfo_t structure type is defined in macro BPXYSINF.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the BPXWTE service returns -1 if it is not successful.

If it is successful and the Function_code is #waitid, the wait-extension service returns a value of zero.

If it is successful and the Function_code is #wait3, the wait-extension service returns the process id of the child status is being reported for. If WNOHANG was specified and status is not available for any children specified by the Id, the wait-extension service returns a value of zero.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the wait-extension service stores the return code. The wait-extension service returns Return_code only if Return_value is

-1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The wait-extension service can return one of the following values in the Return_code parameter:

Return_code	Explanation
ECHILD	The calling process has no existing unwaited-for child processes.
EFAULT	The address of a returned parameter is incorrect. The following reason codes can accompany the return code: JrBadExitStatusAddr, JrBadSiginfoAddr, or JrBadRusageAddr.
EINTR	The function was interrupted because the calling process received a signal.
EINVAL	An incorrect Option, Idtype, or Function_code was specified. The following reason codes can accompany the return code: JrBadOptions, JrBadIdType, or JrBadEntryCode.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the wait-extension service stores the reason code. The wait-extension service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. When the siginfo_t structure is returned, the following applies:
   - si_signo is always set to SIGCHLD.
   - si_errno is always set to 0.
   - si_code is set to CLD_EXITED, CLD_KILLED, CLD_DUMPED, CLD_TRAPPED, CLD_STOPPED, or CLD_CONTINUED. The CLD_ constants are defined in macro BPXYSIGH.
   - si_pid is set to the process ID of the child status is being returned for.
   - si_uid is set to the user ID of the child status is being returned for.
   - si_addr is set to the faulting instruction if the child process terminated because of a SIGILL, SIGFPE, or SIGSEGV signal; otherwise, si_addr is set to 0.
   - si_status is set to the child's exit status. The exit status is mapped by macro BPXYWAST.
   - si_band is always set to 0.
2. If the Options field is 0, the wait-extension service waits for processes that have exited.

Characteristics and restrictions

None.

Examples

For an example using this callable service, see "BPX1WTE (wait extension) example" on page 1212.

w_getipc (BPX1GET, BPX4GET) — Query interprocess communications

Function

The w_getipc service queries shared memory, messages, semaphores, and map service objects for the next or specified member to which the invoker has read access.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GET):	31-bit
AMODE (BPX4GET):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GET,(Ipc_Token | Ipc_Member_ID,
              Buffer_Address,
              Buffer_Length,
              Command,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GET with the same parameters. Buffer_address is a doubleword.

Parameters

Ipc_Token
Supplied parameter

Type: Integer

Length:
Word

Specifies a token that corresponds to a message queue, shared memory segment, or semaphore member ID. Zero represents the first member ID. The token to be used in the next invocation is passed back in Return_value. Ipc_Token is ignored when Ipc_OVER is specified.

Ipc_Member_ID
Supplied parameter

Type: Integer

Length:
Word

Specifies a message queue ID, semaphore ID, or shared member ID.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

Address of the buffer structure defined by IPCQ. For the structure describing this buffer, see "BPXYIPCQ — Map w_getipc structure" on page 987.

Buffer_Length
Supplied parameter

Type: Address

Length:
Fullword

Length of the structure defined by IPCQ. Set to IPCQ#LENGTH. Field IPCQLENGTH will differ from IPCQ#LENGTH when the system call is at a different level than the included IPCQ. An error will be returned if this length is less than 4. The buffer will be filled to the lesser of IPCQ#LENGTH or the value specified here.

Command
Supplied parameter

Type: Integer

Length:
Fullword

Command	Description
Ipcq#ALL	Retrieve next shared memory, message and semaphore member.
Ipcq#MSG	Retrieve next message member.
Ipcq#SEM	Retrieve next semaphore member.
Ipcq#SHM	Retrieve next shared memory member.
Ipcq#OVER	Overview of system variables. Ignores the value of the first operand (Ipc_Token).
Ipcq#MAP	Retrieve mapped memory objects.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getipc service returns the next Ipc_Token (a negative number), `0`, or `-1` (error). If Ipc_Token is specified, `0` indicates end of file. If Ipc_Member_ID is specified, `0` indicates success.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getipc service stores the return code. The w_getipc service returns Return_code only if Return_value is -1. See *z/OS*

UNIX System Services Messages and Codes for a complete list of possible return code values. The w_getipc service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EACCES	Operation permission (read) is denied to the calling process for the Ipc_Member_ID specified. The following reason code can accompany the return code: JRIpcDenied.
EINVAL	The Ipc_Member_ID is not valid for the command specified: • The Command parameter is not a valid command. • The buffer pointer was zero or the buffer length was less than 4. The following reason codes can accompany the return code: JRBuffTooSmall, JRIpcBadID, or JRBadEntryCode.
EFAULT	An input parameter specified an address that caused the callable service to program check. The following reason code can accompany the return code: JRBadAddress.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getipc service stores the reason code. The w_getipc service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. See *z/OS UNIX System Services Messages and Codes* for the reason codes.

Usage notes

1. With Ipc_Token, return_values should be tested for 0 (end of file) or -1 (error). Other values are negative and will be the next Ipc_Token.
2. With Ipc_Member_ID, return_values should be tested for -1 (error).
3. A member's accessibility can change if the permissions are changed.
4. A given Ipc_Token may not always retrieve the same member.
5. If a specific member is desired and has been found using Ipc_Token, subsequent requests may place it at that token or later (never earlier).
6. The Ipc_BINSEM, Ipc_MEGA, Ipc_RcvTypePID, Ipc_SndTypePID, Ipc_PLO1, and Ipc_PLO2 bits in the S_MODE field in IpcqIpcp show the values that were requested on the original get request.
7. The Ipc_PLOinUse bit in the S_MODE field in IpcqIpcp shows actual usage of the PLO (Perform Lock Operation) instruction for serialization.
8. When the message queue serialization uses latches, all activity is stopped for the duration of the w_getipc request, and the values that are returned show a snapshot in time.
9. When the message queue serialization uses the PLO instruction (see Ipc_PLOinUse), msgsnd and msgrcv activity continues during the w_getipc request. This can cause misleading results. For example, while the w_getipc service is counting messages on the queue, elements can be added and removed, causing a number that is too high or too low. In the same way, with the msgrcv and msgsnd waiters, a waiter's PID could appear twice in the list.

10. When the Ipcq#MAP command is specified, the w_getipc service returns information about a map service object each time it is called. It also returns a token, which the caller can use on the next call to provide information for the system to find the next map service object.

    With Ipcq#MAP, the w_getipc service may, and likely will, return information about a particular map service object multiple times – once for each process that is using it. If the caller wishes to provide summary information, it is the caller's responsibility to associate the responses for the same map service object with each other. This can be done using the creator PID, because any one process can create only one object, and descendants of that process cannot create a map service object at all. Map service objects are inherited from the parent process.

    The following information is returned for Ipcq#MAP:

    - The creating process's PID. You can use this information to tie together the data returned from other w_getipc calls.
    - The PID of a process that is using this object.
    - The UID of a process that is using this object.
    - The GID of a process that is using this object.
    - The shutdown indicator.
    - The size of the blocks.
    - The number of blocks in the map area.
    - The number of blocks in use.
    - The number of blocks mapped by this process.

    Note: Some of these fields, which are always the same for all processes that are using a particular map service object at any one moment, may differ from one call to another. This is because they may have changed since the w_getipc call for an earlier process. The shutdown indicator, the number of blocks in the map area, and the number of blocks in use, in particular, may show this behavior.

Related services

- “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 738
- “msgget (BPX1QGT, BPX4QGT) — Create or find a message queue” on page 391
- “semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores” on page 631

Characteristics and restrictions

There are no restrictions on the use of the w_getipc service.

Examples

For an example using this callable service, see “BPX1GET (w_getipc) example” on page 1143.

w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems

Function

The w_getmntent callable service gets information about mounted file systems.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1GMN):	31-bit
AMODE (BPX4GMN):	64-bit
ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1GMN,(Buffer_length,
              Buffer,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4GMN with the same parameters.

Parameters

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the size of the specified buffer.

Buffer
Parameter supplied and returned

Type: Structure

Length:
Specified by the Buffer_length parameter

The name of the buffer where the information about the mount entries is stored. The area consists of a header followed by a series of entries describing the file systems, all of which are mapped by BPXYMNTE. For information about the content of this area, see "BPXYMNTE — Map response and element structure of w_getmntent" on page 993.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getmntent service returns the number of mount entries that were written to the buffer, or -1 if unsuccessful. A 0 indicates that no more mount entries were found.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getmntent service stores the return code. The w_getmntent service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The w_getmntent service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	Parameter error; for example, the buffer is too short to hold one entry, or the mount header portion of the buffer was not cleared before the first call. The following reason codes can accompany the return code: JRBuffTooSmall, JRInvalidCursor, JRInvalidParms, and JRFilesysNotThere.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getmntent service stores the reason code. The w_getmntent service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Except for the situation that is described in Usage Note 5 or in Usage Note 7, before a program calls w_getmntent for the first time, the header part of the buffer must be cleared to zeros. For information about the format and length of this header, refer to "BPXYMNTE — Map response and element structure of w_getmntent" on page 993.
2. If more than one call is made to w_getmntent, use the same buffer on each call, because part of the information that is returned in the buffer tells the file system where to continue retrieving its information.
3. The w_getmntent call normally returns information about as many file systems as are mounted, or as many as fit in the passed buffer. The number of entries that are contained in the buffer is returned. The caller must have a buffer large enough to receive information about at least a single mount entry with each call. If a zero-length buffer is passed, no information is returned, but the return value contains the total number of mounted file systems. This value could then be used to get enough storage to retrieve information about all these file systems in one additional call.

If no parameter was specified when the file system was mounted, MNTENTPARMLEN and MNTENTPARMOFFSET are each zero. If a parameter was specified, its address is the sum of the address of MNTE and the contents of MNTENTPARMOFFSET.

If an entry together with its mount parameter does not fit in the buffer, the entry is returned without the mount parameter. In this case, MNTENTPARMLEN contains the length of the mount parameter, and MNTENTPARMOFFSET is zero. To ensure that at least one entry, including the mount parameter, is returned, you should allocate space for at least two entries.

A device number of x40000000 can be provided to indicate the returned path name should be up to 64 bytes of the mount point path name at the time of the mount.

4. You could also retrieve all mount entries by setting up a loop that continues to call w_getmntent until a return value of either `-1` (in an error) or `0` (no more entries found) is returned.
5. Information about a specific file system can be obtained if the device number of that file system is known. In this case, the device number can be filled into the header of the buffer along with the eye catcher for the buffer, and the w_getmntent call returns a single entry with information about that file system. Or a device number value of x40000000 can be used to indicate that the returned path name should be up to 64 bytes of the mount point path name at the time of the mount.
6. If the caller of w_getmntent lacks search authorization to one or more of the directories in the mount point, or if the file system is being mounted asynchronously, MNTENTMOUNTPOINT is returned empty. That is, MNTENTPATHLEN is zero and MNTENTMOUNTPOINT contains a null character as the first character. The lack of search authorization might be caused by permission bits, ACLs and the FSACESS class checks for each file system that is traversed.
7. If the caller of w_getmntent is requesting the additional information that is available in the expanded MNTE data structure, MNT2, the caller must construct the buffer according to the following rules:
   a. The buffer must be an appropriate size to hold the additional data that will be returned with the MNT2 version of the control block.
   b. The eye-catcher in the MNTE header must be filled in with the MNT2 value.
   c. The bodylength field, also in the header, must be set to the length of the MNTE2 body.
8. If an entry together with its system list does not fit in the buffer, the entry is returned without the system list. In this case, the MNTENTSYSLISTOFFSET is zero, and MNTENTSYSLISTLENGTH contains the length of the system list.
9. When an aggregate name is present for a file system, it is included in the output if there is room for it. The offset field is set to the offset of the name from the beginning of this mount entry, and the length field is set to the length of the name. If the offset is zero and the length is nonzero, this indicates that there is an aggregate name, but there was not enough space left in the output buffer to hold it. In this case, the length field tells the program how much more space is needed.

   Aggregate names are present for zFS file systems. They may be up to 44 characters long, and are returned in a string that is terminated by a null character. The returned length does not include the null terminator byte.

10. A value returned in the MNTENTROSECLABEL (read-only security label) indicates that the file system is protected with that security label. The absence of a value in this field indicates only that a read-only security label is not in effect for that file system, and does not mean that the file system contents are not protected with security labels.
11. Be aware that the size of the mount table could change (for instance, due to automount activity) in the interval between successive w_getmntent calls.

Related services

- "mount (BPX1MNT) — Make a file system available" on page 377
- "umount (BPX1UMT, BPX4UMT) — Remove a virtual file system" on page 867

Characteristics and restrictions

There are no restrictions on the use of the w_getmntent service.

Examples

For an example using this callable service, see "BPX1GMN (w_getmntent) example" on page 1147.

w_getpsent (BPX1GPS) — Get process data

Function

The w_getpsent callable service provides data describing the status of a process. This data includes, but is not limited to, running time, user IDs (UIDs), groups IDs (GIDs), and invocation parameters. Data is returned for the processes that the caller can access.

Note: There is no 64-bit version of the w_getpsent callable service. To get equivalent function, use "__getthent (BPX1GTH, BPX4GTH) — Get thread data" on page 278 in 64-bit mode.

Requirements

Operation	Environment
Authorization:	Problem program or supervisor state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	No latches should be held
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL  BPX1GPS,(Process_token,
               Buffer_length,
               Buffer_address,
               Return_value,
               Return_code,
               Reason_code)
```

Parameters

Process_token
Returned parameter

Type: Integer

Length:
Fullword

The name of the fullword containing the process token that identifies the relative position of a process in the system. Zero represents the first process in the system.

Buffer_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of the fullword containing the value PGPS#LENGTH.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword

The name of the fullword containing the address of the buffer. For the mapping of these options, see "BPXYPGPS — Map the response structure for w_getpsent" on page 1007. Several fields in this buffer should be initialized:

```
PGPSCONTTYBLEN    Length of PGPSCONTTYBUF
PGPSCONTTYPTR     Address of PGPSCONTTYBUF(Len¬=0)
PGPSPATHBLEN      Length of PGPSPATHBUF
PGPSPATHPTR       Address of PGPSPATHBUF (Len¬=0)
PGPSCMDBLEN       Length of PGPSCMDBUF
PGPSCMDPTR        Address of PGPSCMDBUF (Len¬=0)
```

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getpsent service returns the process token or `0` if the request is successful, or `-1` if it is not successful.

Value	Explanation
Process Token	The process token of the next logical process in the system.
`0`	End of file. There are no active processes at or following the requested process which the user is allowed access.
`-1`	Error. See Return_code for an explanation.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getpsent service stores the return code. The w_getpsent service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The w_getpsent service can return one of the following values in the Return_code parameter:

Value	Explanation
EFAULT	An input parameter contained the address of storage where the invoker is not authorized.
EINVAL	The process_token is not in the valid range.

Reason_code

Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_getpsent service stores the reason code. The w_getpsent service returns Reason_code only if Return_value is -1 Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. Only those processes are returned for which RACF allows the user access based on its EUid, RUid, or SUid.
2. The normal user starts with Process_token at zero, and continues calling BPX1GPS with the process token returned in Return_value until the value of `0`, end of file, is reached.
3. PGPSSTARTTIME is in seconds since the Epoch (00:00:00 on 1 January 1970).
4. PGPSUSERTIME and PGPSSYSTIME are task-elapsed times in 1/100ths of seconds.
5. The CONTTY, PATH, and CMD input fields are initialized by the BPXYPGPS macro when it is expanded in the program CSECT for a non-reentrant program.
6. If Buffer_length does not match that used by the callable service, the task sets PGPSLENERR on. This can reflect a change in BPXYPGPS caused by the addition of functions in later releases. This could be intentional: data is returned up to the length specified in Buffer_length. If the length specified is less than the offset of PGPSCONTTYBLEN, BPX1GPS treats the request as if the three BLEN fields were zero.
7. PGPSSYSTIME reports the system CPU time consumed for the address space that the process is running in. When only one process is running in the address space, this time represents the accumulated system CPU time for that process. However, when more than one process is running in an address space, the information that is returned is actually the accumulated system CPU time consumed by all of the work running in the address space.

Characteristics and restrictions

None.

Examples

The following example starts with the first process (relative process zero) and reports the status for all processes for which the invoker is allowed access (by the security access facility).

The program is reentrant and should be link-edited with RENT in the IEWL PARM.

```
BOOKSAM4 CSECT ,                    Reentrant linkage
BOOKSAM4 AMODE 31
BOOKSAM4 RMODE ANY
         USING *,R15                Program addressability
@BEGIN0  B     @BEGIN1              Branch around program header
         DROP  R15
         DC    C'Sequential w_getpsent'
         DS    0H
@BEGIN1  STM   R14,12,12(13)        Save caller's registers
         LR    R2,13                Hold address of caller's area
         LR    R3,R1                Hold parameter register
         LR    12,R15               R12 program base register
         USING @BEGIN0,12           Program addressability
         L     R0,@SIZEDAT          Size this program's dynamic area
         GETMAIN RU,LV=(0)          Getmain dynamic storage
         LR    13,R1                R13 -> this program's dynamic/save
         USING @DYNAM,13            Dynamic addressability
         ST    R2,@BACK             Save caller's save area pointer
         ST    13,8(,R2)            Give caller out save area
         LR    R1,R3                Restore parameter register
@BEGIN2  EQU   *  *  *  *  *  *  *   End of the entry linkage code
         SPACE ,
         MVC   WTOHEAD,WTOCONS      Initialize WTO line
         MVI   DOT,C'.'
* If BPX1GPS has been link-edited with this program, the V-CON will be
* resolved; if not, BPX1GPS must be loaded.  In either case, the address
* of the module is stored.
         ICM   R0,B'1111',GPSVCON   BPX1GPS address if link edited
         BNZ   STGPSEP              Branch to store GPS entry point
         LOAD  EP=BPX1GPS           Load w_getpsent stub
STGPSEP  ST    R0,GPSENTRY          Store BPX1GPS entry point
* Initialize the variables and enter the loop.
         XC    PROCTOKEN,PROCTOKEN  Start with 1st process
         MVC   PGPSCONTTYBLEN,=A(L'PGPSCONTTYBUF)  Controlling TTY
         LA    R2,PGPSCONTTYBUF
         ST    R2,PGPSCONTTYPTR
         MVC   PGPSPATHBLEN,=A(L'PGPSPATHBUF)      Path name
         LA    R2,PGPSPATHBUF
         ST    R2,PGPSPATHPTR
         MVC   PGPSCMDBLEN,=A(L'PGPSCMDBUF)        Command
         LA    R2,PGPSCMDBUF
         ST    R2,PGPSCMDPTR
         LA    R2,PGPS              Address of PGPS buffer
         ST    R2,PGPSA
         SPACE ,
GETPS    L     R15,GPSENTRY         Address of BPX1GPS load module
         CALL  (15),                Get process data                 +
               (PROCTOKEN,          Relative process token           +
               PGPSL,               Length of buffer                 +
               PGPSA,               Buffer, mapped by BPXYPGPS       +
               RETVAL,              Return value (next, eof or error) +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      --------------------------------
         SPACE ,  *  *  *  *  *  *  Test for end of file
         ICM   R2,B'1111',RETVAL    Load return value, set CCode
         BZ    RETURN               0 is end of file
```

```
         BL    RETURNRC              -1 is error
         ST    R2,PROCTOKEN          Store the next process token
         SPACE ,  *  *  *  *  *  *   Initialize WTO area & message
         MVI   XPID,C' '             Blank variable portion of line
         MVC   XPID+1(WTO#BLANK-1),XPID
* Convert the process ID to printable hex.
         L     R8,PGPSPID            R8 = process ID
         LA    R9,XPID               To be placed at message start
         LA    R15,8                 8 nibbles to convert (4 bytes)
         LA    R10,9                 For 0-9 / A-F compare
NIBBLE   LR    R11,R8                Target bits in 0-3     XYYYYYYZ
         SRL   R11,28                Bits 0-3 to 28-31      0000000X
         SLL   R8,4                  Drop bits 0-3 off end  YYYYYYZ0
         CLR   R11,R10               Are 4 bits 0-9 or A-F
         BC    B'0010',AF            Branch if A-F
         LA    R11,57(,R11)          Add for 0-9 (57+183=240 or F0)
AF       LA    R11,183(,R11)         Add for 0-F (183+10=193 or C1)
         STC   R11,0(,R9)            Store to results location
         LA    R9,1(,R9)             Increment R9 to next location
         BCT   R15,NIBBLE            Decrement half byte counter, loop
* Go after the state of the process
         MVI   THREAD,C'1'                Assume single task thread
         TM    PGPSSTATUS1,PGPSMULTHREAD  if multithread process
         BZ    NOTMULT
         MVI   THREAD,C'M'
NOTMULT  TM    PGPSSTATUS1,PGPSPTHREAD    if pthread_create task(s)
         BZ    NOTIPT
         MVI   THREAD,C'H'
NOTIPT   MVC   STATE,PGPSSTATUS3          Z, W, X, S, C, F, K, R
         TM    PGPSSTATUS0,PGPSSWAP       if swapped out
         BZ    NOTSWAP
         MVC   SWAPA,=CL4'SWAP'
NOTSWAP  TM    PGPSSTATUS1,PGPSSTOPPED    if stopped
         BZ    NOTSTOP
         MVC   STOPA,=CL4'STOP'
NOTSTOP  TM    PGPSSTATUS1,PGPSTRACE      if ptrace
         BZ    NOTTRAC
         MVC   TRACA,=CL4'TRAC'
NOTTRAC  EQU   *
         SPACE ,  *  *  *  *  *  *   Display message to operator
         WTO   MF=(E,WTOAREA)        Write to Operator
         SPACE ,  *  *  *  *  *  *   Loop back
         B     GETPS                 for the next Process data
         SPACE ,
* * * * *.* * *.* * * * * * * * * * *.* * * * * * * * * * * * * * * * *
RETURN   XR    R15,R15               Zero return code
RETURNRC L     R0,@SIZEDAT           Size this program's dynamic area
         LR    R1,13                 R1 -> this program's dynamic arae
         L     13,@BACK              R2 -> caller's save area
         DROP  13
         FREEMAIN RU,LV=(0),A=(1)
         L     R14,12(,13)           Restore caller's R14
         LM    R0,12,20(13)          Restore caller's R0-R12
         BSM   0,R14                 Branch back to caller
@SIZEDAT DC    A(@ENDYN-@DYNAM)      Size of dynamic storage
         SPACE ,  * * * * * * * * * *.* Program constants * * * * * * *
PGPSL    DC    A(PGPS#LENGTH)        Length of process data buffer
         WXTRN BPX1GPS               Weak to allow link edit or not
GPSVCON  DC    V(BPX1GPS)            Get Process data module
WTOCONS  DS    0CL8                  Constant value for WTOHEAD
         DC    AL2(WTO#LENGTH)         Length of area
         DC    AL2(0)                  WTO flags
         DC    CL4'PID='               Process ID =
         SPACE ,  *  *  *  *  *  *   Dynamic storage variables
@DYNAM   DSECT ,
@SAVE00  DS    0D                    Standard save area - 72 Bytes
         DS    A
```

```
@BACK    DS    A                    Backwards savearea pointer
@FORWARD DS    A                    Forwards savearea pointer
         DS    15A                  Regs 14,15,0-12
         SPACE ,
WTOAREA  DS    0F                   WTO message
WTOHEAD  DS    CL8                  Mapped by WTOCONS
XPID     DS    CL8                  Hex of process ID
         DS    CL1
THREAD   DS    CL1                  1, M or H
         DS    CL1
STATE    DS    CL1                  Z, W, X, S, C, F, K, R
         DS    CL1
SWAPA    DS    CL4                  SWAP or blank
         DS    CL1
STOPA    DS    CL4                  STOP or blank
         DS    CL1
TRACA    DS    CL4                  TRAC or blank
WTO#BLANK EQU   *-XPID               Length to blank
DOT      DS    CL1
WTO#LENGTH EQU *-WTOAREA             Length of WTO area
         SPACE ,
GPSENTRY DS    A                    Address of BPX1GPS
PROCTOKEN DS   F                    Relative process token
PLIST    DS    6A                   Calling parameter list
RETVAL   DS    F                    Return value - next PROCTOKEN
RETCODE  DS    F                    Return code
RSNCODE  DS    F                    Reason code
         SPACE ,
PGPSA    DC    A(PGPS)              ->Process data buffer
         BPXYPGPS DSECT=NO,         Place in current dsect           +
               VARLEN=(0,0,0)       ConTty=0,Path=0,Cmd=0
@ENDYN   EQU   *                    End of dynamic storage
         SPACE 3  * * * * * * * * * *.* Register equates  * * * * * * *
R0       EQU   0
R1       EQU   1                    Parameter list pointer
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
*                12                 Program base register
*                13                 Savearea & dynamic storage base
R14      EQU   14                   Return address
R15      EQU   15                   Branch location
         SPACE ,
         END
```

w_ioctl (BPX1IOC, BPX4IOC) — Control I/O

Function

The w_ioctl callable service conveys a command to a device. The specific actions that are specified by the w_ioctl callable service vary by device and physical file system, and are defined by the device driver or physical file system.

The SIOCGPARTNERINFO ioctl provides an interface for an application to retrieve information about its partner, including connection routing information, the user ID of the partner, or both. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.

The SIOCSPARTNERINFO ioctl provides an interface for an application to set up the environment that is required to retrieve the user ID of its partner using the SIOCGPARTNERINFO ioctl. Issuing the SIOCSPARTNERINFO ioctl prior to the SIOCGPARTNERINFO ioctl can provide better performance, potentially eliminating wait time when issuing the SIOCGPARTNERINFO ioctl. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB (AF_INET/AF_INET6 socket support only)
Cross memory mode:	PASN = HASN
AMODE (BPX1IOC):	31-bit task or SRB mode
AMODE (BPX4IOC):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1IOC,(File_descriptor,
              Command,
              Argument_length,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4IOC with the same parameters.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of an open file or a socket descriptor.

Command
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ioctl command that is to be passed to the device driver or physical file system.

See "BPXYIOCC — Ioctl command definitions" on page 971 for a complete list of the commands that are supported.

Argument_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the length of the argument. The length of the argument is specified as an integer value in the range 0–51 200.

Argument
Parameter supplied and returned

Type: Defined by the device driver or physical file system

Character set:
No restriction

Length:
Specified by the Argument_length parameter

Specifies the name of a buffer, of length Argument_Length, containing the argument to be passed to the device driver or physical file system.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_ioctl service returns one of the following values:

Return_value	Explanation
0	Request was successful. For the **getfacl** command, return_value contains the FACL length if the request is successful.
-1	Request was not successful.
1	The SIOCSECENVR ioctl with the SIOC#GETENVR argument was issued and the buffer size specified with the SECO_BUFFERLEN argument was zero or was not large enough to contain the security object. (See usage note 18 on page 915.)

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_ioctl service stores the return code. The w_ioctl service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The w_ioctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	The *fildes* parameter is not a valid file or socket descriptor. The following reason code can accompany the return code: JrFileNotOpen.

Return_code	Explanation
EFAULT	The address is incorrect. The following reason codes can accompany the return code: JrReadUserStorageFailed, JrWriteUserStorageFailed.
EINVAL	One of the following occurred: • The w_ioctl service specified an incorrect length for the argument. The correct argument length range is 0–51 200. • An invalid command was encountered. The following reason codes can accompany the return code: JRInvIoctlCmd, JrNotSupportedForFileType, JrFileNotOpen, JrBadSubField.
EIO	One of the following occurred: • The process group of the process that is issuing the function is an orphaned, background process group, and the process that is issuing the function is not ignoring or blocking SIGTTOU. • There has been a network or transport failure. The following reason codes can accompany the return code: JRSingleTDRegd, JRPrevSockError.
EMVSPARM	Incorrect parameters were passed to the service. The following reason codes can accompany the return code: JRNoStorage and JRInvParmLength.
ENOBUFS	Insufficient buffer space available. The following reason code can accompany the return code: JrNoArea.
ENODEV	The device is incorrect. The function is not supported by the device driver. The following reason code can accompany the return code: JRFuncNotSupported.
ENOTTY	The w_ioctl service specified an incorrect file descriptor. The file type was not character special. The following reason code can accompany the return code: JRNotSupportedForFileType.
EALREADY	An attempt was made to unregister a file that is not registered.
E2BIG	The argument_length passed on a SetfACL or GetfACL request was not large enough to contain even the minimum amount of data. The size specified must be large enough to hold a RACL_Edit, followed by an FACL and as many FACL_Entry(s) as needed.
EIBMBADTCPNAME	The command passed was IOCC#DIRIOCTL, and the stack name was not found attached to this socket. The specific error is determined by the reason code that accompanies this return code: **JrNoCINET** Common INET is not configured, or this is not a socket and the name did not match the PFS name. This error may not be critical to the application, because the imbedded ioctl can be sent directly to the one and only stack or PFS as a regular ioctl. **JrCINETBadName** CINET is configured, and this name does not match any stack. **JrCINETNotAttached** CINET is configured and this name matches a stack, but that stack is not attached to this socket.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_ioctl service stores the reason code. The w_ioctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. z/OS UNIX domain sockets support the following commands:
   - FIONBIO
   - FIONREAD
   - FIONWRITE
   - SECIGET
   - SIOCATMARK
   - SIOCSECENVR
3. Inet sockets pass the ioctl command to TCP/IP. Refer to *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference* for the commands that are supported.
4. Pseudoterminals (ptys) and remote terminals (rtys) support TIOCGWINSZ and TIOCSWINSZ to get and set window size. Ptys also support TIOCNOTIFY begin (IOCC#PWBEGIN) and TIOCNOTIFY end (IOCC#PWEND) secure data mode.

   TIOCGWINSZ and TIOCSWINSZ retrieve and store the winsize structure (BPXYWNSZ). TIOCNOTIFY sets the TIOCXPKT_PWBEGIN and TIOCXPKT_PWEND bits on master read() when in extended packet mode.
5. The pipe file system does not support ioctl.
6. The IOCC#UPDTOFTE command updates a 100-byte state area that is associated with an Open File Table Entry (OFTE). OFTEs are created by the socket, open, and pipe functions, and are shared by child processes.

   This function is intended for use by runtime libraries.

   The Argument buffer contains an UPDTOFTE subcommand and the offset, length, and value of the data to be updated. Refer to the BPXYIOCC macro for the mapping of this structure.

   Data written to or read from the state area is addressed by offset and length within the state area. The state area is initialized to all zeros when it is allocated.

   Three subcommands are available:

   - IocUo#Write

     The specified data value is written to the specified offset in the state area. This subcommand also initially allocates the area and must be the first UPDTOFTE subcommand issued.
   - IocUo#Read

     The data at the specified offset in the state area is returned.
   - IocUo#CS

     This is used for a "compare and swap" type write to the state area. The specified old_value is compared to what is currently in the state area at the

old_offset. If they match, the new_value is written to the new_offset. If they do not match, the current data at the old_offset in the state area is returned in the old_value along with a Return_Value of 1. The old data and the new data do not have to be at the same offset within the state area.

All of the subcommand operations are atomic with respect to other tasks attempting to access the same OFTE state area.

7. The Iocc#RegFileInt command registers interest in a file and allows the program to be notified when a change to that file occurs.

   The program creates one or more IPC message queues and specifies a Queue Id on each registration, along with a message type and a user token that identifies the file to the program. These are specified in the Rfis structure in the BPXYRFIS macro. See "BPXYRFIS — Map the register file interest structures" on page 1032. A Registered File Token, RfTok, is returned from the registration; this can be used later to unregister the file.

   You can register files by descriptor with the w_ioctl service, or by path name with the w_pioctl service.

   When a change occurs to a registered file, a message is sent on the registered IPC Message Queue. The message content is described by the Rfim structure in the BPXYRFIS macro, and contains:

   - The message type specified on registration
   - The user token specified on registration
   - The type of change that occurred

   The types of file changes that generate a message are:

   - File write, including truncate and open(O_TRUNC)
   - Any attribute change, such as a chmod or chown request
   - Renaming, removal, or unlinking of any of the file's names
   - Attempts to unmount the containing file system

   Because a registered file is implicitly unregistered when a message is sent, only one message is sent for any given registration.

   A file can be explicitly unregistered with the w_ioctl or the w_pioctl service. An Rfis structure is passed on these calls that contains the RfTok that was returned when the file was registered. The file descriptor or path name that is used on the call is not important, but it must be valid. If the registered file is no longer open, and its file descriptor is therefore not readily available, you can use the w_pioctl service with a path name of "/".

   If you try to unregister a file that has already been implicitly or explicitly unregistered, the call fails with EALREADY. If you receive this return code, there may be a message waiting for you on the queue, so you should coordinate the freeing of any file-related control blocks that might be referenced when that message is read.

   All file registrations are removed if the registering process terminates or issues an exec-type call and no messages are sent.

   To receive a change message, the queue must be writable by anyone who might change the files, so we recommend that you create the queue with permission bits of 622.

   The queue must be large enough to accommodate the expected number of unprocessed messages, and the messages must be processed fast enough so that the system limit on total outstanding messages is not exceeded. Messages that cannot be queued immediately are discarded, but the fact that messages were lost is remembered. This information is communicated to the application in one of two ways: (1) the Rfim_LostMsgs flag is set on subsequent change

messages sent to this process until a message is successfully queued; or (2) the Rfis_LostMsgs flag is returned on the next successful registration or unregistration.

When an application is informed that messages have been lost, it should do the following:

- Unregister all registered files, ignoring any EALREADY return codes
- Drain the message queue, ignoring any change messages received
- Start over

Program errors can also prevent messages from being delivered; for example, if a bad queue id is specified on registration. When a message cannot be delivered, a Ctrace entry is written for component SYSOMVS of type FILE. The trace entry contains the character string "RFIPCERR", the returned failure codes from the msgsnd service, the queue id used, and the message that was being sent. You can use this information during program development to diagnose simple bugs.

A registered file does not have to be open to be, or to remain, registered.

A file can be registered multiple times, and by different processes. Each registration causes a separate message when the file is changed.

Any file type can be registered, but some change events only apply to regular files. In particular, writes to a directory (that is, file creation and deletion) do not generate a change message for a registered directory.

No special authority is required to register a file. Any file that the caller has open or is allowed to make stat() calls to can be registered.

Registration and file change notification are intended for use by programs that would otherwise issue periodic stat() or fstat() calls to monitor a file's time stamps in order to detect changes to the file.

8. For file systems that support access control lists (ACLs), you can use the following commands:

   GetfACL

   Retrieves information from an access control list. The Argument parameter specifies the user buffer containing the following input:

   - A structure of type RACL_EDIT, defined in IRRPCOMP, followed by
   - A structure of type FACL, defined in IRRPFACL.

   z/OS Security Server RACF Data Areas describes these structures.

   Upon successful return, the buffer holds the requested ACLs. Therefore, the size of the buffer passed to BPX1IOC (specified by Argument_length) must be big enough to hold the returned ACLs. If it is not big enough, another call will be needed. The maximum number of ACL entries is 1024.

   Set the RACL_EDIT and FACL fields as follows:

Field name	Value
RACL_EDIT_OPTYPE	0
RACL_EDIT_ACLTYPE	The type of ACL being requested (RACL_ACCESS, for instance). You must issue separate calls for access and default ACLs.
FACL_ID	FACL
FACL_LEN	Size of FACL (FACL_LENGTH)
FACL_LEN_ENTRY	FACL_ENTRY_LENGTH

Field name	Value
FACL_VERS	Version number (for example, X'01')

Upon successful return, FACL_NUM_ENTRY (offset X'12') contains the number of ACL entries that the file has of the specified type (access or default). It is up to the caller to determine whether the buffer is big enough to hold that many entries. The calculation for the amount of space needed is:

```
(length of RACL_EDIT structure) + (FACL_NUM_ENTRY × FACL_LEN_ENTRY)
```

The entries start at FACL_ENTRIES and are mapped by FACL_ENTRY.

SetfACL

Sets information into an access control list. There are four types of operations you can perform on access or default ACLs:

a. Delete a whole ACL
b. Add a whole ACL
c. Add or change individual ACL entries
d. Delete individual ACL entries

The contents and length of the user buffer passed in the Argument parameter depend on the type of operation, as follows:

a. Deleting a whole ACL: Only a RACL_EDIT structure needs to be passed in the buffer and the buffer only needs to be as big as that structure. Set the RACL_EDIT fields as follows:

   The table shows the field name and value needed when deleting a whole ACL.

Field name	Value
RACL_EDIT_OPTYPE	RACL_DELETE
RACL_EDIT_ACLTYPE	RACL_ACCESS, RACL_FILEMOD, or RACL_DIRMOD

   If the ACL is not found, the request is ignored.

b. Adding a whole ACL: A RACL_EDIT structure, FACL structure, and all FACL_ENTRY blocks to be added must be passed in the Argument buffer. Argument_length must indicate the size of the entire buffer. Set the RACL_EDIT, FACL, and FACL_ENTRY fields as follows:

Field name	Value
RACL_EDIT_OPTYPE	RACL_ADD
RACL_EDIT_ACLTYPE	RACL_ACCESS, RACL_FILEMOD, or RACL_DIRMOD
FACL_ID	FACL
FACL_LEN	FACL_LENGTH + (number of FACL_ENTRYs × FACL_ENTRY_LENGTH) **Note:** Do not include the length of RACL_EDIT.
FACL_LEN_ENTRY	FACL_ENTRY_LENGTH
FACL_VERS	Version number (for example, X'01')
FACL_NUM_ENTRY	Number of FACL_ENTRY blocks in the buffer
Set the following fields for each FACL_ENTRY, as appropriate:	
FACL_READ	1 (to give permission)
FACL_WRITE	1 (to give permission)

Field name	Value
FACL_EXECUTE	1 (to give permission)
FACL_ENTRY_TYPE	X'01' for user or X'02' for group
FACL_ENTRY_ID	UID or GID (in decimal), based on FACL_ENTRY_TYPE

c. Adding or changing individual ACL entries: A RACL_EDIT structure, FACL structure, and all FACL_ENTRY blocks to be added or modified must be passed in the Argument buffer. Argument_length must indicate the size of the entire buffer. Set the RACL_EDIT, FACL, and FACL_ENTRY fields as follows:

Field name	Value
RACL_EDIT_OPTYPE	RACL_MODIFY
RACL_EDIT_ACLTYPE	RACL_ACCESS, RACL_FILEMOD, or RACL_DIRMOD
FACL_ID	FACL
FACL_LEN	FACL_LENGTH + (number of FACL_ENTRYs × FACL_ENTRY_LENGTH) **Note:** Do not include the length of RACL_EDIT.
FACL_LEN_ENTRY	FACL_ENTRY_LENGTH
FACL_VERS	Version number (for example, X'01')
FACL_NUM_ENTRY	Number of FACL_ENTRY blocks in the buffer
Set the following fields for each FACL_ENTRY, as appropriate:	
FACL_READ	1 (to give permission)
FACL_WRITE	1 (to give permission)
FACL_EXECUTE	1 (to give permission)
FACL_ENTRY_TYPE	X'01' for user or X'02' for group
FACL_ENTRY_ID	UID or GID (in decimal), based on FACL_ENTRY_TYPE

If the entry is not found in the existing ACL, it is added as a new entry. If the entry is found for the given user or group, it is modified with the specified permissions.

d. Deleting individual ACL entries: A RACL_EDIT structure, FACL structure, and all FACL_EDIT_ENTRY blocks to be deleted must be passed in the Argument buffer. Argument_length must indicate the size of the entire buffer. Set the RACL_EDIT, FACL, and FACL_EDIT_ENTRY fields as follows:

Field name	Value
RACL_EDIT_OPTYPE	RACL_MODIFY
RACL_EDIT_ACLTYPE	RACL_ACCESS, RACL_FILEMOD, or RACL_DIRMOD
FACL_ID	FACL
FACL_LEN	FACL_LENGTH + (number of FACL_EDIT_ENTRYs × FACL_ENTRY_LENGTH) **Note:** Do not include the length of RACL_EDIT.
FACL_LEN_ENTRY	FACL_ENTRY_LENGTH
FACL_VERS	Version number (for example, X'01')
FACL_NUM_ENTRY	Number of FACL_EDIT_ENTRY blocks in the buffer
Set the following fields for each FACL_EDIT_ENTRY to be deleted:	
FACL_DEL_ENTRY	1
FACL_EDIT_TYPE	X'01' for user or X'02' for group

If the entry is not found in the existing ACL, it is ignored. If the entry is found for the given user or group, it is deleted. You can

have entries to be deleted along with entries to be added in the same buffer.

The following FACL_ENTRY fields are useful for debugging:

FACL_RACF_RETURN_CODE
The return code from RACF, documented in *z/OS Security Server RACF Callable Services* (see the return and reason codes for the **makeFSP** service).

FACL_RACF_REASON_CODE
The reason code from RACF, documented in *z/OS Security Server RACF Callable Services* (see the return and reason codes for the **makeFSP** service).

FACL_ERROFF
If the problem is with an entry, this field indicates the offset into the Argument buffer where the problem occurred.

Also, refer to the usage notes for the **R_setfacl** service in *z/OS Security Server RACF Callable Services*.

9. The IOCC#DIRIOCTL (Directed Ioctl) command sends an imbedded ioctl command and argument to a specified stack. The input argument for this command is the IocDirIoctl structure, from the BPXYIOCC macro ("BPXYIOCC — Ioctl command definitions" on page 971), with the following fields:

   Field Description

   IocDirName
   The name of the stack

   IocDirCmd
   The ioctl command to be sent to IocDirName

   IocDirArgLen
   The length of IocDirArg, which follows

   IocDirArg
   The ioctl argument to be sent to IocDirName

   The imbedded ioctl is passed to the specified stack, if that stack is attached to this socket, without any examination or processing by the system. Any errors that are returned are usually returned by the stack. Directed Ioctl is not strictly restricted to socket stacks. The name should match the PFS name for the descriptor that is used.

   If the imbedded ioctl generates output in its argument buffer, the output is returned in the IocDirArg buffer.

   A unique error can be returned by z/OS UNIX System Services for this ioctl command, EIBMBADTCPNAME, when the stack name is not found attached to this socket.

10. The IOCC#GETSTACKS (Get TCPIP Stack Names) command returns the names of all the transport stacks that are attached to a socket, and information related to those stacks. The output argument for this command is the IocStackInfo structure, from the BPXYIOCC macro ("BPXYIOCC — Ioctl command definitions" on page 971), with the following fields:

    Field Description

    IocStackEntries
    The number of IocStackName array entries that were returned. With CINET, one or more entries may be returned, depending on how

many stacks have been configured under CINET, how many are or have been active, and any stack affinity that may have been established for the socket or process.

IocStackName
The name of the stack.

IocStackCINET
Indicates that this is a CINET socket. When this bit is on, the IocStack_IPv6_Interfaces and IocStack_IPv4_Interfaces flags indicate whether the specified stack has configured interfaces of each type. Without CINET, use the SIOCGIFVERSION ioctl to obtain this information directly from the Inet stack. See SIOCGIFVERSION (determine if an IPv4 or IPv6 interface has been configured on a TCP/IP stack) in *z/OS UNIX System Services File System Interface Reference*for information about the SIOCGIFVERSION ioctl command.

IocStack_IPv6_Support
Indicates that this stack supports IPv6 protocols and sockets created with AF_INET6. CINET supports IPv6 sockets over stacks that do not themselves support IPv6, as long as IPv4-mapped addresses can be used.

IocStackTdIndex
The CINET TdIndex for this stack. This value is used in the upper halfword of Interface Indices when CINET is configured.

IocStack_Active
Indicates that this stack is active. When used with this ioctl command, this bit is usually on, because inactive stacks are not usually attached to a socket, unless the stack has recently terminated.

This ioctl is not strictly restricted to socket stacks; however, with any other type of Physical File System, all of the socket-related flags would be off.

Tip: You can use the PC#TdNames pfsctl command function of the pfsctl (BPX1PCT, BPX4PCT) service to obtain a complete list of all the stack names, active or inactive, that are configured under CINET.

11. The IOCC#GRTRSELECT (Get CINET PreRouter Selections) command returns the CINET stack that would be chosen for each of a list of destination IP addresses. This ioctl is passed an array of IP addresses, and returns for each address the CINET stack that would be chosen for that destination. This is the stack over which a connect() or sendto(), for instance, would be routed if that address were specified on the call at this time. If CINET is not configured, the socket's one and only stack is returned. The input and output argument for this command is the IocRtrSelect structure, from the BPXYIOCC macro (“BPXYIOCC — Ioctl command definitions” on page 971), with the following fields in each array entry:

Field Description

IocRtrIpAddr
Specifies the IP address to test. This is an IPv6 address or an IPv6-mapped IPv4 address.

IocRtrStack
Returns the name of the stack that would be chosen.

IocRtrErrTest
When equal to B'0', this indicates that there was an error with this one

IP address. The following two fields are also returned: IocRtrErrno, which contains the failing return code (errno), and IocRtrRsn, which contains the failing reason code.

12. The SIOCGIFNAMEINDEX (Get Interface Name/Index Table) command returns the Interface Name/Index table for every stack that is attached to a socket. The output argument for this command is the If_NameIndex structure, from the BPXYIOCC macro ("BPXYIOCC — Ioctl command definitions" on page 971), with the following fields:

    Field Description

    If_NITotalIF
    Contains the total number of interfaces that have indices assigned on the stacks that are attached to this socket.

    If_NIEntries
    Contains the number of interfaces that have been returned. When the total is greater than the number of entries returned, the supplied buffer was not large enough to hold all of the required information. In that case, If_NITotalIF can be used to calculate the amount of space needed and the call can be repeated. When all the interfaces can be returned, the two values are equal.

    If_NITable
    Contains an array of If_NameIndexEntry structures.

    Each interface is described by an If_NameIndexEntry structure consisting of:

    Structure
    Description

    If_NIIndex
    Contains the Interface Index, as described in this topic.

    If_NIName
    Contains the Interface Name, as a 1- to 16-byte character string, left-justified, and padded with blanks. When there is more than one stack, these names may not be unique, because the names are defined to each stack individually with their own configuration procedures.

    If_NINameTerm
    A null character supplied to terminate the name string for the convenience of C routines.

    Tip: To query for the total number of interfaces, you can specify an argument length of 8, just large enough for the first two fields, and the total will be returned in If_NITotalIF, with an If_NIEntries value of 0.

    This output is similar to the output of the if_nameindex() C/C++ function. For a CINET socket with more than one stack attached, the tables from each stack are concatenated into one output table. For a CINET socket, in general, a Transport Driver Index, TdIndex, value will be inserted into the Interface Indices to uniquely identify the interfaces. For example, with two stacks (1) TCPA, with interfaces IFA1 and IFA2, whose interface indices are 1 and 2, respectively, within TCPA, and (2) TCPB, with interfaces IFB1 and IFB2, whose interface indices are 1 and 3, respectively, within TCPB, the output of this ioctl would be something like:

    ```
    ('00010001'x, IFA1),('00010002'x, IFA2),('00020001'x, IFB1),('00020003'x, IFB2)
    ```

The first halfword of the index value indicates which stack under CINET the interface belongs to. The second halfword contains that stack's interface index for this interface.

Without CINET, if TCPA was configured as the only stack, and it was IPv6-enabled, the output of this ioctl would be:

```
('00000001'x, IFA1),('00000002'x, IFA2)
```

Interface indices are used in various places in IPv6, such as for the scope_id of the IPv6 sockaddr structure and within the in6_pktinfo structure. In a CINET configuration, the first halfword of an interface index is used to route a call to the corresponding numbered stack. The upper halfword is cleared before the data is passed to the stack, so that one could use interface indices of the form X'000N0000' as a way to route a call to stack number N without actually specifying an interface index to that stack. The specified stack must be attached to the current socket. The stacks under CINET are numbered in the order of the SUBFILESYSTYPE statements in the BPXPRMxx parmlib member that defined the configuration. These values can be determined from the IocStackTdIndex field of the Iocc#GetStacks ioctl, or from the order of the names returned by the PC#TdNames pfsctl.

Refer to the C/C++ functions if_nameindex(), if_nametoindex(), and if_indextoname() for more information about interface names and indices. (See *z/OS XL C/C++ Runtime Library Reference*.)

13. The SIOCGSOCKPOEATTRS and SIOCGFDPOEATTRS commands return port of entry information for multilevel security. SIOCGSOCKPOEATTRS returns port of entry attributes for a socket resource. SIOCGFDPOEATTRS returns port of entry attributes for a non-socket resource. The port of entry information that can be returned by these commands is defined in the IocPoeAttr block in "BPXYIOCC — Ioctl command definitions" on page 971.
14. The Iocc#DevConsole command allows a program with appropriate privileges to suppress the message number and user ID that are normally prefixed to message BPXF024I when text that is written to /dev/console is sent to the system console. The Argument_length must be 4 and an Argument value of Iocc#DevConSuppress (1) enables suppression of the header on future writes. An Argument value of Iocc#DevConUnSupprs (0) cancels the suppression, so future writes will contain the header.
15. The SECIGET_T ioctl command returns both process-level and, if available, task-level security information of the peer for an AF_UNIX stream-connected socket. The task-level security information is from the task that issued the connect or accept call. The security information is returned in a BPXYSECT structure. The security information is not available until **accept()** completes. The availability of the peer's task-level security data is determined by the task-level userID length field. If the length is zero, the peer does not have task-level security data.
16. The SIOCTIEDESTHRD ioctl command with the SIOC#TIESD argument ties or associates a descriptor with the thread of the SIOCTIEDESTHRD caller. If that task terminates before the descriptor is closed or untied from the task, then the termination processing for the file system thread will close the descriptor.

    The SIOCTIEDESTHRD ioctl command with the SIOC#UNTIESD argument unties a previously tied descriptor from a thread. SIOCTIEDESTHRD can be used on heavy-weight and medium-weight threads.
17. The FIONWRITE ioctl command returns the number of bytes that can be written to the connected peer AF_UNIX stream socket before the socket blocks or returns an EWOULDBLOCK return code. Note that the number of bytes returned by FIONWRITE is not guaranteed unless there is serialization among the calling applications.

18. The SIOCSECENVR ioctl command sets or gets the security environment of a client that is connecting to an AF_UNIX stream socket server. Arguments for the SIOCSECENVR ioctl are mapped by the BPXYSECO structure (see "BPXYSECO — Map the input/output of BPX1IOC for the SIOCSECENVR request" on page 1035). A server must have appropriate privileges to issue this ioctl.
    - The SIOCSECENVR ioctl with the SIOC#SETENVR argument is for use by an AF_UNIX stream socket server to mark the server socket as one that requires the full security environment of a connecting client to be available before a connect() will successfully complete. The connect service obtains the security environment of the connector and anchors it off of the connector's socket for use by the server. If the security environment cannot be obtained during connect processing, the connect() will fail. This ioctl is meaningful only for sockets that will become server sockets; it has no effect for all other sockets.
    - The SIOCSECENVR ioctl with the SIOC#GETENVR argument is for use by an AF_UNIX stream socket server to copy the previously set security environment from the connector's address space to the server's address space so that it can be used as input on calls to the security product. This ioctl is only meaningful for server sockets that previously issued the SIOCSECENVR ioctl with the SIOC#SETENVR argument.

      Servers must issue the SIOCSECENVR ioctl with the SIOC#GETENVR argument in a timely fashion. It should be issued immediately following the **accept()** call. If any **read()** calls are issued before the SIOC#GETENVR request, then the server will no longer be able to use a SIOC#GETENVR request to obtain the client's security environment.

      Servers may specify the buffer in which to hold the client's security environment in the BPXYSECO structure. If the specified buffer is not large enough to contain the security environment or if SECO_BUFFERLEN is zero, the service will obtain a buffer of the correct size in the server's address space and return the security environment in that buffer. Information about the buffer and the security environment will be returned in the BPXYSECO structure and the return value will be set to 1. The server must free this buffer when it no longer needs it.

      The security environment returned by a SIOC#GETENVR request can be specified as input to the RACROUTE interface using the ENVRIN keyword or to the initACEE callable service using the ENVR_in parameter.
19. The SIOCGIFCONF6 (Get IPv6 Interface Configuration) command gets the name, address, and other information about the configured IPv6 network interfaces. This is similar to the SIOCGIFCONF command for IPv4.

    A Net_IfConf6Header structure is passed as the argument of the ioctl. This structure specifies the buffer where the configuration information is to be written and is returned with the number of entries and entry length of the Net_IfConf6Entry structures that were written to the output buffer. These structures are defined in the BPXYIOC6 macro.

    If the specified buffer address and buffer length are both zero, a Query function is performed and the header is returned with the total number of entries that would be output and the length of each individual entry for the specified version. If the specified version is zero or not supported, it is replaced with the maximum supported version and the entry length returned corresponds to that version.

    If a call to get information fails with either return code ERANGE or with both return code EINVAL and the Nif6h_Version field having been changed, the

call was converted into a Query function and the header has been filled. In this case, the content of the output buffer is unpredictable.

If Common INET (CINET) is configured and multiple TCP/IP stacks are attached to the socket, the output from each stack that is enabled for IPv6 will be concatenated in the output buffer and the header will contain the total number of entries returned from all the stacks. The version returned with the Query function will be the highest version supported by all the stacks.

This ioctl can be issued on an AF_INET or AF_INET6 socket.

20. The Iocc#GetPathName and Iocc#GetPathNameRel **(get pathname** and **get relative pathname**) commands return the absolute or relative path name, respectively, of the file referred to by *File_descriptor*. The output path name is placed in the Argument buffer and is ll terminated by a null character. The length of the output path name is determined by scanning for the trailing null byte. The Argument buffer provided must be large enough to contain the output name and the trailing null byte or the call will fail with RC=ERANGE.
21. The SIOCGPARTNERINFO ioctl provides an interface for an application to retrieve information about its partner, including connection routing information, the user ID of the partner, or both. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.
22. The SIOCSPARTNERINFO ioctl provides an interface for an application to set up the environment that is required to retrieve the user ID of its partner using the SIOCGPARTNERINFO ioctl. Issuing the SIOCSPARTNERINFO ioctl prior to the SIOCGPARTNERINFO ioctl can provide better performance, potentially eliminating wait time when issuing the SIOCGPARTNERINFO ioctl. Refer to *z/OS Communications Server: IP Programmer's Guide and Reference* for details.

Characteristics and restrictions

The argument is limited to 51 200 bytes.

Examples

For an example using this callable service, see "BPX1IOC (w_ioctl) example" on page 1154.

__wlm (BPX1WLM, BPX4WLM) — WLM interface service

Function

The __wlm callable service invokes a wide variety of Workload Manager (WLM) functions. You can also use it to invoke Enterprise Workload Manager™ (eWLM) ARM (Application Response Measurement) functions.

For information about the ARM functions, see *IBM Tivoli® eWorkload Management Version 1*.

Requirements

Operation	**Environment**
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1WLM):	31-bit
AMODE (BPX4WLM):	64-bit
ASC mode:	Primary mode

Operation	Environment
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1WLM,(FunctionCode,
              ParmListPtr,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4WLM with the same parameters. All parameter addresses and addresses in parameter structures are doublewords.

Parameters

FunctionCode

Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains a value that indicates the type of WLM or eWLM function that the caller is requesting. The following are the supported values:

Value	Description
WLM_QUERY_METRICS	Query WLM System Information
WLM_QUERY_SCHEDENV	Query WLM Scheduling Environment
WLM_CHECK_SCHEDENV	Check WLM Scheduling Environment
WLM_DISCONNECT	Disconnect from WLM
WLM_DELETE_WORKUNIT	Delete a WLM Work Unit
WLM_JOIN_WORKUNIT	Join a WLM Work Unit
WLM_LEAVE_WORKUNIT	Leave a WLM Work Unit
WLM_CONNECT_WORKMGR	Connect to WLM as a work manager
WLM_CONNECT_SERVERMGR	Connect to WLM as a server manager
WLM_CREATE_WORKUNIT	Create a WLM work unit (this function creates an independent WLM enclave)
WLM_CONTINUE_WORKUNIT	Continue WLM work unit (this function creates a dependent WLM enclave)
WLM_EXTRACT_WORKUNIT	Extract the WLM work unit token (this function returns the WLM enclave token)
WLM_EXPORT_WORKUNIT	Export a WLM work unit
WLM_UNDOEXPORT_WORKUNIT	Undo a prior export request for a WLM work unit
WLM_IMPORT_WORKUNIT	Import a WLM work unit

Value	Description
WLM_UNDOIMPORT_WORKUNIT	Undo a prior import request for a WLM work unit
WLM_QUERY_ENCLAVECLASS	Query enclave class information for a WLM work unit
WLM_CONNECT_EXPORTIMPORT	Connect a subsystem to WLM to export and import work units, but not to create them
ARM_BIND_THREAD	Indicates that the calling thread is performing on behalf of an ARM transaction
ARM_BLOCK_TRANSACTION	Indicates that a started transaction is blocked waiting for an external transaction or some other event to complete
ARM_DESTROY_APPLICATION	Indicates that the registration data about an application is no longer needed
ARM_DISCARD_TRANSACTION	Signals that a started ARM transaction should be ignored
ARM_GENERATE_CORRELATOR	Generates an ARM correlator for use with ARM_REPORT_TRANSACTION
ARM_GET_ARRIVAL_TIME	Stores a 64-bit integer representing the current time
ARM_REGISTER_APPLICATION	Informs ARM of metadata about the application
ARM_REGISTER_METRIC	Informs ARM of metadata about each metric the application provides
ARM_REGISTER_TRANSACTION	Informs ARM of metadata about the transaction measured by the application
ARM_REPORT_TRANSACTION	Reports statistics about a transaction that has already completed
ARM_START_APPLICATION	Indicates that an instance of an application has started running and is prepared to make ARM calls
ARM_START_TRANSACTION	Indicates that a transaction is beginning execution
ARM_STOP_APPLICATION	Indicates that the application instance is finished making ARM calls
ARM_STOP_TRANSACTION	Signals the end of a transaction
ARM_UNBIND_THREAD	Indicates that the calling thread is no longer performing on behalf of an ARM transaction
ARM_UNBLOCK_TRANSACTION	Indicates that a transaction is no longer waiting for a downstream transaction to complete
ARM_UPDATE_TRANSACTION	Signals that a transaction is still processing
EWLM_CLASSIFY_CORRELATOR	Creates an eWLM specific ARM correlator for classification purpose

These constants are defined in the BPXYWLM macro; see “BPXYWLM — WLM constants and parameter list DSECTs” on page 1069.

For detailed information about the ARM function codes, see *IBM Tivoli eWorkload Management Version 1*.

ParmListPtr
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) field that contains the address of the parameter list for the WLM function that is to be performed. See "BPXYWLM — WLM constants and parameter list DSECTs" on page 1069 for the mapping of the parameter lists for the various WLM functions.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __wlm service returns the return value for the WLM function that was requested.

For the following set of WLM functions, the service returns `0` if the request is successful, or `-1` if it is not successful:

- WLM_CHECK_SCHEDENV
- WLM_DISCONNECT
- WLM_DELETE_WORKUNIT
- WLM_JOIN_WORKUNIT
- WLM_LEAVE_WORKUNIT
- WLM_CREATE_WORKUNIT
- WLM_CONTINUE_WORKUNIT
- WLM_QUERY_METRICS
- WLM_QUERY_SCHEDENV
- WLM_EXTRACT_WORKUNIT
- WLM_EXPORT_WORKUNIT
- WLM_UNDOEXPORT_WORKUNIT
- WLM_IMPORT_WORKUNIT
- WLM_UNDOIMPORT_WORKUNIT
- WLM_QUERY_ENCLAVECLASS
- ARM_BIND_THREAD
- ARM_BLOCK_TRANSACTION
- ARM_DESTROY_APPLICATION
- ARM_DISCARD_TRANSACTION
- ARM_GENERATE_CORRELATOR
- ARM_GET_ARRIVAL_TIME
- ARM_REGISTER_APPLICATION
- ARM_REGISTER_METRIC
- ARM_REGISTER_TRANSACTION
- ARM_REPORT_TRANSACTION

- ARM_START_APPLICATION
- ARM_START_TRANSACTION
- ARM_STOP_APPLICATION
- ARM_STOP_TRANSACTION
- ARM_UNBIND_THREAD
- ARM_UNBLOCK_TRANSACTION
- ARM_UPDATE_TRANSACTION
- EWLM_CLASSIFY_CORRELATOR

If the WLM_QUERY_METRICS, WLM_QUERY_SCHEDENV, or WLM_QUERY_ENCLAVECLASS function fails with an error that indicates that the supplied buffer was too small, the supplied length field in the input parameter list is updated to contain the length that is required for the function to succeed.

For the following set of WLM functions, the service returns a WLM connect token if the request is successful, or -1 if it is not successful:

- WLM_CONNECT_WORKMGR
- WLM_CONNECT_SERVERMGR
- WLM_CONNECT_EXPORTIMPORT

Return_Code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __wlm service stores the return code. The __wlm service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The __wlm service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EFAULT	An argument of this service contained an address that was not accessible to the caller.
EINVAL	The FunctionCode parameter contains a value that is not correct; or the function parameter list data is not correct.
EMVSWLMERROR	A WLM service failed. Consult Reason_code to determine the WLM service that failed and the reason for the error. See *z/OS MVS System Messages, Vol 9 (IGF-IWM)* for a list of WLM services (IWM*) error reason codes.
EMVSARMERROR	An ARM error occurred. Consult Reason_code to determine the reason for the error. The ARM reason codes are documented in the **_Elmarm4.h** header file.
EPERM	The calling thread's address space is not permitted to the BPX.WLMSERVER FACILITY class profile. The caller's address space must be permitted to the BPX.WLMSERVER FACILITY class profile. If BPX.WLMSERVER is not defined, the calling process is not defined as a superuser (UID=0).
EMVSSAF2ERR	An error occurred in the security product.

Return_code	Explanation
ESRCH	A WLM_EXTRACT_WORKUNIT request was issued, but the WLM enclave token was not returned. Consult Reason_code to determine the exact reason it was not returned. Most likely, the unit of work is not in an enclave.
EMVSERR	Recovery processing was entered for a reason other than EFAULT.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the __wlm service stores the reason code. The __wlm service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the WLM reason codes, see *z/OS UNIX System Services Messages and Codes*.

ARM reason codes are documented in the **_Elmarm4.h** header file.

Usage notes

1. The WLM_CONNECT_WORKMGR and WLM_CONNECT_EXPORTIMPORT functions both enable use of the export and import functions, but only the former enables use of the create function.
2. For a WLM_CREATE_WORKUNIT function invocation, some of the classification data that is pointed to by the supplied IWMCLSFY parameter list is truncated if it exceeds the maximum supported length, as follows:

Data	Maximum length
ACCTINFO	143 bytes
SUBSYSPM	255 bytes
SOURCELU	17 bytes
COLLECTION	18 bytes
CORRELATION	12 bytes

Related services

None.

Characteristics and restrictions

1. Certain __wlm functions require that the caller have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined. The following table shows the authorization required for each __wlm function:

Table 24. Authorization requirements for __wlm functions

Function	Authorization
WLM_QUERY_METRICS WLM_CONNECT_SERVERMGR WLM_EXTRACT WORKUNIT	No authorization required

Table 24. Authorization requirements for __wlm functions (continued)

Function	Authorization
WLM_CONNECT_EXPORTIMPORT	If the caller has not already made a WLM_CONNECT_EXPORTIMPORT call, read access to the BPX.WLMSERVER FACILITY or UID 0 are not required
WLM_DELETE_WORKUNIT	If the caller has made a WLM_CONNECT_EXPORTIMPORT call, special authorization is not required.
WLM_QUERY_SCHEDENV WLM_CHECK_SCHEDENV WLM_DISCONNECT WLM_JOIN_WORKUNIT WLM_LEAVE_WORKUNIT WLM_CONNECT_WORKMGR WLM_CREATE WORKUNIT WLM_IMPORT_WORKUNIT WLM_QUERY_ENCLAVECLASS WLM_UNDOIMPORT_WORKUNIT	Read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
WLM_CONTINUE WORKUNIT	A process can have one dependent enclave active at a time without authorization. If a process needs to have more than one dependent enclave active at the same time, it must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
WLM_EXPORT_WORKUNIT	A process can export the enclave it created using WLM_CONTINUE_WORKUNIT without authorization. To export some other enclave, the process must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.
WLM_UNDOEXPORT_WORKUNIT	A process can undo its prior WLM_EXPORT_WORKUNIT request without authorization. To export some other enclave, the process must have read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.

2. All ARM services, with the exception of ARM_GET_ARRIVAL_TIME, require read access to the BPX.WLMSERVER FACILITY class profile, or a UID of 0 if the BPX.WLMSERVER FACILITY class profile is not defined.

Examples

For an example using this callable service, see "BPX1WLM (__WLM) example" on page 1211.

w_pioctl (BPX1PIO, BPX4PIO) — Path name I/O control

Function

The w_pioctl callable service conveys a command to the physical file system that owns the specified file.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE (BPX1PIO):	31-bit
AMODE (BPX4PIO):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1PIO,(Pathname_length,
              Pathname,
              Command,
              Argument_length,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4PIO with the same parameters.

Parameters

Pathname_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the length of the Pathname of the file.

Pathname
Supplied parameter

Type: Character string

Character set:
No restriction

Length:
Specified by the Pathname_length parameter

The name of a field that contains the name of the file to be acted upon.

Command
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the command to be passed to the Physical File System.

Argument_length
Parameter supplied and returned

Type: Integer

Length:
Fullword

The name of a fullword containing the length of the argument. The length of the argument is specified as an integer value in the range 0–51 200.

Argument
Parameter supplied and returned

Type: Defined by the Physical File System

Character set:
No restriction

Length:
Specified by the Argument_length parameter

Specifies the name of a buffer that contains the argument to be passed to the Physical File System.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_pioctl service returns `0` if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_pioctl service stores the return code. The w_pioctl service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The w_pioctl service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EINVAL	An incorrect Command or Argument_length was specified; or the function was directed against a character special file. The following reason codes can accompany the return code: JRInvIoctlCmd, JRIOBufLengthInvalid and JRNotSupportedForFiletype.
EMVSPARM	Incorrect parameters were passed to the service. The following reason codes can accompany the return code: JRNoStorage and JRInvParmLength.
ENODEV	The device is incorrect. The function is not supported for this file. The following reason code can accompany the return code: JRFuncNotSupported.
EACCES	The calling process does not have search permission for some component of the Pathname prefix; or does not have permission to perform the requested function against the specified file.
ENOENT	No file named Pathname was found; or no pathname was specified. The following reason code can accompany the return code: JrFileNotThere.
ELOOP	A loop exists in symbolic links that were encountered during resolution of the Pathname argument. This error is issued if more than 24 symbolic links are detected in the resolution of Pathname.
ENAMETOOLONG	Pathname is longer than 1023 characters; or some component of the pathname is longer than 255 characters. Name truncation is not supported.
ENOTDIR	A component of the Pathname prefix is not a directory.
EALREADY	An attempt was made to unregister a file that is not registered.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_pioctl service stores the reason code. The w_pioctl service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. This form of ioctl may not be used with character special files. Refer to w_ioctl (BPX1IOC, BPX4IOC) for these files.
2. One of the uses of this function is to edit the access control lists of DFS remote files, and to register interest in files by pathname.
3. This function can also be used to set or get the access options for z/OS UNIX files and directories. For more information, see the usage notes for "w_ioctl (BPX1IOC, BPX4IOC) — Control I/O" on page 902 for descriptions of the SetfACL and GetfACL commands.

Characteristics and restrictions

The argument is limited to 51 200 bytes.

w_statvfs (BPX1STF, BPX4STF) — Get the file system status

Function

The w_statvfs callable service obtains status information about a specified file system. You specify the file system by its file system name.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE BPX1STF):	31-bit
AMODE BPX4STF):	64-bit
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1STF,(File_system_name,
              Status_area_length,
              Status_area,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4STF with the same parameters.

Parameters

File_system_name
Supplied parameter

Type: Character string

Character set:
Printable characters

Length:
44 bytes

The name of 44-character field that identifies the file system whose status is to be returned. The name must be left-justified and padded on the right with blanks.

This is the file system name as specified on the mount.

Status_area_length
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword containing the length of the area to which the service returns status information.

Status_area
Parameter supplied and returned

Type: Structure

Length:
Specified by the Status_area_length parameter

The name of an area of length Status_area_length to which the service returns the status information for the file system. The BPXYSSTF macro maps this area. For information on this macro, see "BPXYSSTF — Map response structure for file system status" on page 1055.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_statvfs service returns the length of the status written to the Status_area if the request is successful, or -1 if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_statvfs service stores the return code. The w_statvfs service returns Return_code only if Return_value is -1. For a complete list of possible return code values, see *z/OS UNIX System Services Messages and Codes*. The w_statvfs service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	Information is temporarily unavailable. This can occur because the mount process for the file system is incomplete.
EINVAL	Parameter error; for example, File_system_name was not found. The following reason code can accompany the return code: JRFileSysNotThere.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the w_statvfs service stores the reason code. The w_statvfs service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. It is not considered an error if the passed Status_area_length is not sufficient to hold all the returned information. (That is, future expansion is allowed for.) As much information as will fit is written to Status_area, and this amount is returned.
2. If a buffer of length of zero is passed to this service, no data is returned and the return value is zero. You can check for the existence of a file system by passing such a length.
3. The amount of valid data returned in the Status_area is indicated by the Return_value. This allows for differences in the release levels of z/OS UNIX and the physical file systems.

Related services

- "fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status" on page 199
- "statvfs (BPX1STV, BPX4STV) — Get the file system status" on page 809

Characteristics and restrictions

There are no restrictions on the use of the w_statvfs service.

Examples

For an example using this callable service, see "BPX1STF (w_statvfs) example" on page 1199.

write (BPX1WRT, BPX4WRT) — Write to a file or a socket

Function

The write callable service writes data from a buffer to an open file or socket.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
	SRB - AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1WRT):	31-bit task or SRB mode
AMODE (BPX4WRT):	64-bit task or SRB mode
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1WRT,(File_descriptor,
              Buffer_address,
              Buffer_ALET,
              Write_count,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4WRT with the same parameters. The Buffer_address parameter is a doubleword.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor of the file or socket to write to.

Buffer_address
Supplied parameter

Type: Address

Length:
Fullword (doubleword)

The name of a fullword (doubleword) that contains the starting address of the data that is to be written.

Buffer_ALET
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the ALET for Buffer_address, which identifies the address space or data space the buffer resides in.

You should specify a Buffer_ALET of `0` for the normal case of a buffer in the user's address space (current primary address space). If a value other than `0` is specified for the Buffer_ALET, the value must represent a valid entry in the dispatchable unit access list (DUAL).

Write_count
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the number of bytes that are to be written.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the write service returns the number of actual bytes that were written, if the request is successful, or -1, if it is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the write service stores the return code. The write service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The write service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EAGAIN	Blocking is not in effect for the specified file, and output cannot be written immediately.
EBADF	The File_descriptor parameter does not contain the descriptor of an open file; or that file is not opened for write services. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.
EFBIG	Writing to the specified file would exceed either the file size limit for the process or the maximum file size that is supported by the physical file system.
EINTR	The service was interrupted by a signal before it could write any data. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	The Write_Count parameter contains a value that is less than zero.
EIO	The process is in a background process group and is attempting to write to its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
EPIPE	The request is for a write to a pipe that is not open for reading by any other process; or an attempt was made to write to a socket that is shut down or closed. This error also generates a SIGPIPE signal. The following reason code can accompany the return code: JRSocketClosed.

Return_code	Explanation
EWOULDBLOCK	• The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available. • The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option. The following reason codes can accompany the return code: JRWouldBlock, JRTimeout.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the write service stores the reason code. The write service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. **Write_Count**: The value of Write_count is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.

   The value of Write_count is checked against the file size limit for the process. If no data can be written without exceeding this limit, an error of EFBIG is returned and the SIGXFSZ signal is generated for the process. If at least one byte can be written before exceeding the file size limit, the write is considered successful.
3. **File offset**: If File_descriptor specifies a regular file or any other type of file on which you can seek, the write service begins writing at the file offset that is associated with that file descriptor. A successful write operation increments the file offset by the number of bytes that are written. If the incremented file offset is greater than the previous length of the file, the file is extended; the length of the file is set to the new file offset.

   If the file descriptor refers to a file on which you cannot seek, the service begins writing at the current position. No file offset is associated with such a file.

   If the file was opened with the "append" option, the write routine sets the file offset to the end of the file before it writes output.
4. **Number of bytes written**: Ordinarily, the number of bytes written to the output file is the number you specify in the Write_count parameter. (This number can be zero. If you ask to write zero bytes, the service simply returns a return value of zero without attempting any other action.)

   If the write count that you specify is greater than the remaining space on the output device, or greater than the file size limit for the process, fewer bytes than you requested are written. When at least 1 byte is written, the write is considered successful. If you are not using a pseudoterminal, an attempt to append to the same file causes an error. An error of ENOSPC is returned when there is no remaining space on the output device. An error of EFBIG is returned

when the file size limit for the physical file system is exceeded. An error of EFBIG is also returned if the file size limit for the process is exceeded, at which time the write service also generates a SIGXFSZ signal for the process. With a pseudoterminal, if there is not enough room in the buffer for the whole write, the number of bytes that fit are written, and the number of bytes written is returned. However, on the next write (assuming the buffer is still full), there is a block or EAGAIN is returned, depending on whether the file was opened blocking or nonblocking.

Similarly, fewer bytes are written if the service is interrupted by a signal after some, but not all, of the specified number of bytes are written. The return value shows the number of bytes that are written. But if no bytes were written before the routine was interrupted, the return value is -1**,** and an EINTR error is reported.

5. The write service causes signal SIGTTOU to be sent if all the following conditions are met:
   - The process is attempting to write to its controlling terminal.
   - TOSTOP is set as a terminal attribute (see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831 or "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842).
   - The process is running in a background process group.
   - The SIGTTOU signal is not blocked or ignored.
   - The process is not an orphan.

Related services

- "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174
- "lseek (BPX1LSK, BPX4LSK) — Change a file's offset" on page 345
- "open (BPX1OPN, BPX4OPN) — Open a file" on page 447
- "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481
- "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572

Note: The write service is not related to the **write** shell command.

Characteristics and restrictions

If the file was opened by an authorized program, all subsequent reads and writes against the file must be issued from an authorized state.

The read (BPX1RED, BPX4RED) and write (BPX1WRT, BPX4WRT) callable services do not support simultaneous reading or writing of the same shared open file by different threads when one or both of the following are true:

1. Automatic conversion is enabled using Enhanced ASCII (ON) and different character set IDs (CCSIDs) are used.
2. Automatic conversion is enabled using Unicode Services (ALL) and different CCSIDs are used, or mixing read and write operations of multibyte characters are performed which result is storing of partial characters.

The first restriction is not applicable if each thread coordinates its reads and writes so that simultaneous I/O does not occur. Both restrictions are not applicable if each thread opens the file independently.

Reads or writes that cause a conversion of greater than 2 G result in an EINVAL error with reason JrUniOpTooBig.

Refer to “lseek (BPX1LSK, BPX4LSK) — Change a file's offset” on page 345 for instances of how a lseek operation in a conversion environment can affect read and write operations

Examples

For an example using this callable service, see “BPX1WRT (write) example” on page 1211.

writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers

Function

The writev callable service writes data from a set of buffers.

Requirements

Operation	Environment
Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task SRB - AF_INET/AF_INET6 socket support only
Cross memory mode:	PASN = HASN
AMODE (BPX1WRV):	31-bit task or SRB mode
AMODE (BPX4WRV):	64-bit task mode only
ASC mode:	Primary mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL BPX1WRV,(File_descriptor,
              Iov_count,
              Iov_struct,
              Iov_alet,
              Iov_buffer_alet,
              Return_value,
              Return_code,
              Reason_code)
```

AMODE 64 callers use BPX4WRV with the same parameters. All addresses in the Iov_struct structure are doublewords.

Parameters

File_descriptor
Supplied parameter

Type: Integer

Length:
Fullword

The name of a fullword that contains the file descriptor for which the writev is to be done.

Iov_count
Supplied and returned parameter

Type: Integer

Length:
Fullword

The name of a field that contains the number of buffers that are pointed to by Iov_struct. The total number of buffers may not exceed IOV_MAX (defined in "BPXYIOV — Map the I/O vector structure" on page 986).

Iov_struct
Supplied parameter

Type: Character

Length:
iov_count × length(iov)

The name of a field that contains 31(64)-bit pointers to buffers from which data is to be retrieved for the purpose of writing to the file or socket. In 64-bit mode, Iov_struct contains doubleword pointer and length subfields. See "BPXYIOV — Map the I/O vector structure" on page 986 for more information about the format of this field.

Iov_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for Iov_struct.

Iov_buffer_alet
Supplied parameter

Type: Integer

Length:
Fullword

The name of a field that contains the ALET for buffers that are pointed to by Iov_struct.

Return_value
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the writev service returns one of the following:

- The number of bytes that were written from the buffers, if the request is successful.
- -1, if the request is not successful.

Return_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the writev service stores the return code. The writev service returns Return_code only if Return_value is -1. See *z/OS UNIX System Services Messages and Codes* for a complete list of possible return code values. The writev service can return one of the following values in the Return_code parameter:

Return_code	Explanation
EBADF	An incorrect file descriptor was specified. The following reason codes can accompany the return code: JRFileDesNotInUse, JRFileNotOpen.
ECONNRESET	Connection reset by peer. The following reason code can accompany the return code: JRSocketNotCon.
EFBIG	Writing to the specified file would exceed either the file size limit for the process, or the maximum file size supported by the physical file system.
EINTR	A signal interrupted the writev service before any data was written. The following reason code can accompany the return code: JRSockRdwrSignal.
EINVAL	An incorrect value was specified on one of the input parameters. The following reason code can accompany the return code: JRSocketCallParmError.
EIO	The process is in a background process group and is attempting to write to its controlling terminal. However, TOSTOP is set, the process is neither ignoring nor blocking **SIGTTOU** signals, and the process group of the process is orphaned. This can happen, for example, if a background job tries to write to the terminal after the user has logged off.
EMSGSIZE	The message is too large to be sent all at once, as the socket requires. The following reason code can accompany the return code: JRSockBufMax.
ENOBUFS	A buffer could not be obtained. The following reason code can accompany the return code: JROutofSocketCells.
ENOTCONN	The socket was not connected. The following reason code can accompany the return code: JRSocketNotCon.
ENOTSOCK	Socket_descriptor does not refer to a valid socket descriptor. The following reason code can accompany the return code: JRMustBeSocket.
EPIPE	An attempt was made to write to a socket that is shut down or closed. The following reason code can accompany the return code: JRSocketClosed. This error also generates a **SIGPIPE** signal.
EPROTOTYPE	An incorrect socket type was supplied. The following reason code can accompany the return code: JRIncorrectSocketType.
EWOULDBLOCK	• The socket is marked nonblocking and no space is available for data to be written, or the SO_SNDTIMEO timeout value was reached before space became available. • The socket is marked blocking. The call is blocked, without sending any data, for that time period which was specified in the SO_SNDTIMEO option. The following reason codes can accompany the return code: JRTimeout, JRWouldBlock.

Reason_code
Returned parameter

Type: Integer

Length:
Fullword

The name of a fullword in which the writev service stores the reason code. The writev service returns Reason_code only if Return_value is -1. Reason_code further qualifies the Return_code value. For the reason codes, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

1. See Appendix J, "Callable services available to SRB mode routines," on page 1333 for more information about programming considerations for SRB mode.
2. This callable service works with any open file descriptor, including files and sockets.
3. **Number of bytes written**: If the number of bytes to be written is greater than the remaining space on the output device, or greater than the file size limit for the process, not all of the data can be written. When at least 1 byte is written, the writev is considered successful. The return value shows the number of bytes that were written. An attempt to writev again to the same file causes an EFBIG error, and if the process file size limit has been exceeded, the writev service generates a SIGXFSZ signal for the process.
4. **Bytes written**: The number of bytes that are requested for writing is not checked against any system limit. A limit can be imposed by a high-level-language POSIX implementation.

   The number of bytes that are requested for writing is checked against the file size limit for the process. If no data can be written without exceeding this limit, an error of EFBIG is returned and the SIGXFSZ signal is generated for the process. If at least one byte can be written before exceeding the file size limit, the write is considered successful.
5. **File offset**: If File_descriptor specifies a regular file or any other type of file on which you can seek, the write service begins writing at the file offset that is associated with that file descriptor. A successful write operation increments the file offset by the number of bytes that are written. If the incremented file offset is greater than the previous length of the file, the file is extended; the length of the file is set to the new file offset.

   If the file descriptor refers to a file on which you cannot seek, the service begins writing at the current position. No file offset is associated with such a file.

   If the file was opened with the "append" option, the write routine sets the file offset to the end of the file before writing output.
6. **Number of bytes written**: Ordinarily, the number of bytes that are written to the output file is the number requested for writing. (This number can be zero. If you ask to write zero bytes, the service simply returns a return value of zero without attempting any other action.)

   If the write count that you specify is greater than the remaining space on the output device, or greater than the file size limit for the process, fewer bytes than you requested are written. When at least 1 byte is written, the write is considered successful. If you are not using a pseudoterminal, an attempt to append to the same file causes an error. An error of ENOSPC is returned when there is no remaining space on the output device. An error of EFBIG is returned when the file size limit for the physical file system is exceeded. An error of EFBIG is also returned if the file size limit for the process is exceeded, at which time the write service also generates a SIGXFSZ signal for the process. With a

pseudoterminal, if there is not enough room in the buffer for the whole write, the number of bytes that fit are written, and the number of bytes that were written is returned. However, on the next write (assuming the buffer is still full) there is a block or EAGAIN is returned, depending on whether the file was opened blocking or nonblocking.

Similarly, fewer bytes are written if the service is interrupted by a signal after some, but not all, of the specified number of bytes are written. The return value shows the number of bytes that were written. But if no bytes were written before the routine was interrupted, the return value is -1 and an EINTR error is reported.

7. The writev service causes signal **SIGTTOU** to be sent if all the following conditions are met:
   - The process is attempting to write to its controlling terminal.
   - TOSTOP is set as a terminal attribute (see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831 or "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842).
   - The process is running in a background process group.
   - The **SIGTTOU** signal is not blocked or ignored.
   - The process is not an orphan.

Related services

- "readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers" on page 590
- "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928

Characteristics and restrictions

There are no restrictions on the use of the writev service.

Examples

For an example using this callable service, see "BPX1WRV (writev) example" on page 1212.

Appendix A. System control offsets to callable services

An alternative to loading or link-editing the service stub is to include in the code the system control offset to the callable service. For example, use decimal 52 for the offset of access (BPX1ACC).

When using the offsets, set the registers up as follows:

Register 1
To contain the address of your parameter list. Set bit `0` of the last address in the list on.

Register 14
To contain the return address in the invoking module.

Register 15
To contain the address of the callable service code.

Example

The following is an example of code that specifies the offset. The example assumes that register 1 is set up with the address of the parameter list. Replace *offset* with the appropriate value from the following offset table.

```
L     15,16           CVT - common vector table
L     15,544(15)      CSRTABLE
L     15,24(15)       CSR slot
L     15,offset(15)   Address of the service
BALR  14,15           Branch and link
```

List of offsets

Table 25. System control offsets to callable services

Service	Offset	Function
BPX1ACC	52	access
BPX1ACK	972	auth_check_rsrc_np
BPX1ACP	508	accept
BPX1AIO	988	asyncio
BPX1ALR	224	alarm
BPX1ANR	1060	accept_and_recv
BPX1ASP	1088	aio_suspend
BPX1ATM	668	attach_execmvs
BPX1ATX	664	attach_exec
BPX1BND	512	bind
BPX1BAS	592	bind with source address selection
BPX1CCA	480	cond_cancel
BPX1CCS	1012	console_np
BPX1CHA	84	chaudit
BPX1CHD	56	chdir
BPX1CHM	60	chmod
BPX1CHO	64	chown
BPX1CHP	764	chpriority
BPX1CHR	500	chattr
BPX1CID	968	convert_id_np
BPX1CLD	68	closedir

Table 25. System control offsets to callable services (continued)

Service	Offset	Function
BPX1CLO	72	close
BPX1CON	516	connect
BPX1CPL	1132	__cpl
BPX1CPO	484	cond_post
BPX1CRT	872	chroot
BPX1CSE	488	cond_setup
BPX1CTW	492	cond_timed_wait
BPX1CWA	496	cond_wait
BPX1DEL	888	deletehfs
BPX1DSD	1124	sw_signaldelv
BPX1ENV	960	oe_env_np
BPX1EXC	228	exec
BPX1EXI	232	_exit
BPX1EXM	236	execmvs
BPX1EXT	200	extlink_np
BPX1FAI	1168	FreeAddrInfo
BPX1FCA	140	fchaudit
BPX1FCD	852	fchdir
BPX1FCM	88	fchmod
BPX1FCO	92	fchown
BPX1FCR	504	fchattr
BPX1FCT	96	fcntl
BPX1FPC	100	fpathconf
BPX1FRK	240	fork
BPX1FST	104	fstat
BPX1FSY	108	fsync
BPX1FTR	112	ftruncate
BPX1FTV	848	FstatVfs
BPX1GAI	1164	GetAddrInfo
BPX1GCL	1024	getclientid
BPX1GCW	116	getcwd
BPX1GEG	244	getegid
BPX1GEP	860	getpgid
BPX1GES	864	getsid
BPX1GET	736	w_getipc
BPX1GEU	248	geteuid
BPX1GGE	772	getgrent
BPX1GGI	252	getgrgid
BPX1GGN	256	getgrnam
BPX1GGR	260	getgroups
BPX1GHA	1160	gethostbyaddr
BPX1GHN	1156	gethostbyname
BPX1GID	264	getgid
BPX1GIV	1028	givesocket
BPX1GLG	268	getlogin
BPX1GMN	76	w_getmntent
BPX1GNI	1172	GetNameInfo
BPX1GNM	524	getpeername
BPX1GPE	776	getpwent
BPX1GPG	272	getpgrp
BPX1GPI	276	getpid
BPX1GPN	280	getpwnam
BPX1GPP	284	getppid
BPX1GPS	428	w_getpsent

Table 25. System control offsets to callable services (continued)

Service	Offset	Function
BPX1GPT	916	grantpt
BPX1GPU	288	getpwuid
BPX1GPY	744	getpriority
BPX1GRL	820	getrlimit
BPX1GRU	824	getrusage
BPX1GTH	1056	__getthent
BPX1GTR	752	getitimer
BPX1GUG	292	getugrps
BPX1GUI	296	getuid
BPX1GWD	936	getwd
BPX1HST	520	gethostid
BPX1IOC	120	w_ioctl
BPX1IPT	396	MvsIptAffinity
BPX1ITY	12	isatty
BPX1KIL	308	kill
BPX1LCO	832	lchown
BPX1LCR	1180	lchattr
BPX1LNK	124	link
BPX1LOD	880	loadhfs
BPX1LSK	128	lseek
BPX1LSN	532	listen
BPX1LST	132	lstat
BPX1MAT	720	shmat
BPX1MCT	724	shmctl
BPX1MDT	728	shmdt
BPX1MGT	732	shmget
BPX1MKD	136	mkdir
BPX1MKN	144	mknod
BPX1MMI	1136	__map_init
BPX1MMP	796	mmap
BPX1MMS	1140	__map_service
BPX1MNT	148	mount
BPX1MP	688	MVSpause
BPX1MPC	408	mvsprocclp
BPX1MPI	680	MVSpauseInit
BPX1MPR	800	mprotect
BPX1MSD	336	mvsunsigsetup
BPX1MSS	312	mvssigsetup
BPX1MSY	804	msync
BPX1MUN	808	munmap
BPX1NIC	748	nice
BPX1OPD	152	opendir
BPX1OPN	156	open
BPX1OPT	528	getsockopt
BPX1OSE	1100	__osenv
BPX1PAF	1072	__pid_affinity
BPX1PAS	316	pause
BPX1PCF	160	pathconf
BPX1PCT	768	pfsctl
BPX1PIO	984	w_pioctl
BPX1PIP	164	pipe
BPX1POE	1176	__poe
BPX1POL	932	poll
BPX1PQG	1152	Pthread_quiesce_and_get_np

Table 25. System control offsets to callable services (continued)

Service	Offset	Function
BPX1PSI	460	pthread_setintr
BPX1PST	472	Pthread_setintrtype
BPX1PTB	448	pthread_cancel
BPX1PTC	432	pthread_create
BPX1PTD	444	pthread_detach
BPX1PTI	476	Pthread_testintr
BPX1PTJ	440	pthread_join
BPX1PTK	464	pthread_kill
BPX1PTQ	412	pthread_quiesc
BPX1PTR	320	ptrace
BPX1PTS	452	pthread_self
BPX1PTT	1016	pthread_tag_np
BPX1PTX	436	pthread_xandg
BPX1PWD	788	password
BPX1QCT	692	msgctl
BPX1QDB	948	querydub
BPX1QGT	696	msgget
BPX1QRC	700	msgrcv
BPX1QSE	388	quiesce
BPX1QSN	704	msgsnd
BPX1RCV	540	recv
BPX1RDD	168	readdir
BPX1RDL	172	readlink
BPX1RDV	536	readv
BPX1RDX	940	read_extlink
BPX1RD2	856	readdir2
BPX1RED	176	read
BPX1REN	180	rename
BPX1RFM	544	recvfrom
BPX1RMD	188	rmdir
BPX1RMG	8	resource
BPX1RMS	548	recvmsg
BPX1RPH	884	realpath
BPX1RW	1108	Pread
BPX1RWD	184	rewinddir
BPX1SA2	1084	__Sigactionset
BPX1SCT	708	semctl
BPX1SDD	300	setdubdefault
BPX1SEC	1044	__security
BPX1SEG	424	setegid
BPX1SEL	552	select
BPX1SEU	420	seteuid
BPX1SF	1064	send_file
BPX1SGE	780	setgrent
BPX1SGI	328	setgid
BPX1SGQ	1104	sigqueue
BPX1SGR	792	setgroups
BPX1SGT	712	semget
BPX1SHT	572	shutdown
BPX1SIA	324	sigaction
BPX1SIN	1004	server_init
BPX1SIP	340	sigpending
BPX1SLK	1068	__shm_lock
BPX1SLP	344	sleep

Table 25. System control offsets to callable services (continued)

Service	Offset	Function
BPX1SMC	1112	__smc
BPX1SMF	1036	__smf_record
BPX1SMS	560	sendmsg
BPX1SND	556	send
BPX1SOC	576	socket_pair
BPX1SOP	716	semop
BPX1SPB	416	sigputback
BPX1SPE	784	setpwent
BPX1SPG	348	setpgid
BPX1SPM	352	sigprocmask
BPX1SPN	760	spawn
BPX1SPR	568	setpeer
BPX1SPW	1008	server_pwu
BPX1SPY	740	setpriority
BPX1SRG	896	setregid
BPX1SRL	816	setrlimit
BPX1SRU	892	setreuid
BPX1SRX	1080	srx_np
BPX1SSI	356	setsid
BPX1SSU	360	sigsuspend
BPX1STA	192	stat
BPX1STE	1076	Set_Timer_Event
BPX1STF	80	w_statfs
BPX1STL	684	Set_limits
BPX1STO	564	sendto
BPX1STQ	1144	server_thread_query
BPX1STR	756	setitimer
BPX1STV	844	StatVfs
BPX1STW	1096	sigtimedwait
BPX1SUI	364	setuid
BPX1SWT	468	sigwait
BPX1SYC	368	sysconf
BPX1SYM	196	symlink
BPX1SYN	868	sync
BPX1TAF	1148	MvsThreadAffinity
BPX1TAK	1032	takesocket
BPX1TDR	24	tcdrain
BPX1TFH	20	tcflush
BPX1TFW	28	tcflow
BPX1TGA	32	tcgetattr
BPX1TGC	900	tcgetcp
BPX1TGP	36	tcgetpgrp
BPX1TGS	912	tcgetsid
BPX1TIM	372	times
BPX1TLS	964	pthread_security_np
BPX1TRU	828	truncate
BPX1TSA	40	tcsetattr
BPX1TSB	44	tcsendbreak
BPX1TSC	904	tcsetcp
BPX1TSP	48	tcsetpgrp
BPX1TST	908	tcsettables
BPX1TYN	16	ttyname
BPX1UMK	204	umask
BPX1UMT	208	umount

Table 25. System control offsets to callable services (continued)

Service	Offset	Function
BPX1UNA	376	uname
BPX1UNL	212	unlink
BPX1UPT	920	unlockpt
BPX1UQS	392	unquiesce
BPX1UTI	216	utime
BPX1VAC	944	v_access
BPX1VCL	1188	v_close
BPX1VCR	620	v_create
BPX1VEX	876	v_export
BPX1VGA	632	v_getattr
BPX1VGT	596	v_get
BPX1VLK	604	v_lookup
BPX1VLN	640	v_link
BPX1VLO	660	v_lockctl
BPX1VMD	624	v_mkdir
BPX1VOP	1184	v_open
BPX1VPC	1040	v_pathconf
BPX1VRA	616	v_readlink
BPX1VRD	612	v_readdir
BPX1VRE	644	v_rmdir
BPX1VRG	584	v_reg
BPX1VRL	600	v_rel
BPX1VRM	648	v_remove
BPX1VRN	652	v_rename
BPX1VRP	588	v_rpn
BPX1VRW	608	v_rdwr
BPX1VSA	636	v_settatr
BPX1VSF	656	v_fstatfs
BPX1VSY	628	v_symlink
BPX1WAT	380	wait
BPX1WLM	1048	__wlm
BPX1WRT	220	write
BPX1WRV	580	writev
BPX1WTE	840	waitid/wait3
BPX2ITY	928	isatty2
BPX2MNT	1128	__mount
BPX2OPN	1052	openstat
BPX2RMS	976	recvmsg2
BPX2SMS	980	sendmsg2
BPX2TYN	924	ttyname2

Appendix B. Mapping macros—AMODE 31

Mapping macros map the parameter options in many callable services. The fields with the comment "Reserved for IBM Use" are not programming interfaces. A complete list of the options for each macro is listed in the macro in “Macros mapping parameter options.”

Most of the mapping macros can be expanded with or without a *DSECT* statement. The invocation operand *DSECT=YES* (default) can be used with either reentrant or nonreentrant programs with the appropriate rules governing the storage backed by the *USING* statement.

Many of the mapping macros exploit the fact that *DC* expands as a *DS* in a *DSECT* and as a *DC* with its initialized value in a *CSECT*. When these fields are expanded as or within DSECTs, the program is responsible for initializing the necessary fields.

Macros mapping parameter options

Specifying DSECT=YES (the default for all macros) creates a DSECT. Addressability requires a USING and a register pointing to storage.

Specifying DSECT=NO (exceptions are listed when this is not allowed) allocates space in the current DSECT or CSECT. In reentrant programs, programmers can place these macros in the DSECT with DSECT=NO, and addressability is accomplished without the individual USING required by DSECT=YES. Nonreentrant programs can place their macros in the program's CSECT and addressability is obtained through the program base register(s).

Specifying LIST=YES (the default for most macros) causes the expansion of the macro to appear in the listing. You can override this by using PRINT OFF.

Specifying LIST=NO removes the macro expansion from the listing.

Additional keywords VARLEN and PREFIX are described in the individual sections where they apply.

BPXYACC — Map flag values for access

```
          BPXYACC    ,
** BPXYACC:  Access intent flags
**  Used by: ACC
ACC                 DSECT ,
ACCRSRV             DS    CL2    Reserved
ACCFLAGS            DS    XL1    Flags
ACCEFFID            EQU   X'04'   check effective ids
ACCDEVNO            EQU   X'02'   return devno if exists
ACCWAIT             EQU   X'01'   Wait for Async Mount
ACCINTENTFLAGS      DS    XL1    Access Intent Flags
*                   EQU   X'F0'  Reserved
ACC_F_OK            EQU   X'08'  Check for file existence
ACC_R_OK            EQU   X'04'  Check for read access to file
```

```
ACC_W_OK             EQU   X'02'  Check for write access to file
ACC_X_OK             EQU   X'01'  Check for execute access to file
ACC#LENGTH           EQU   *-ACC  Length of this structure
** BPXYACC End
```

BPXYAIO — Map asyncio parameter list

AMODE 64 callers use "BPXYAIO — Map asyncio parameter list" on page 1085.

```
           BPXYAIO    ,
* ------------------------------ 31-Bit Version
* ------------------------------
** BPXYAIO:  Asyncio parameter block
**  Used by: AIO
AIO                  DSECT ,
AIOFD                DS    F      File Descriptor
AIOBUFFDW            DS    0CL8   Eight byte addresses
AIOBUFFALET          DS    F      Alet for AioBuffPtr
&AIOBUFFPTR31        DS    F      Buffer Pointer
AIOBUFFSIZE          DS    F      Buffer Length or Iov count
AIOOFFSETDW          DS    0CL8   Offset in File
AIOOFFSETH           DS    F      Offset in File highword
AIOOFFSET            DS    F      Offset in File lowword
AIOMSGEVENT          DS    0C     Message Event overlays SigEv
&AIOSIGEVENT31       DS    CL20   POSIX Signals
AIOREQPRIO           DS    F      REQUEST PRIORITY
AIOLIOOPCODE         DS    F      LIO_LISTIO() OP
*
                     ORG   AIOLIOOPCODE
AIOCMD               DS    F      Command Code
AIONOTIFYTYPE        DS    H      Notification Type
AIOCFLAGS            DS    XL1    Control Flags
AIOOK2COMPIMD        EQU   X'80'  Ok to complete immediately
AIOCALLB4            EQU   X'40'  Call exit before redrive
AIOSYNC              EQU   X'10'  Do synchronously
AIOEXITMODETCB       EQU   X'08'  0=SRB, 1=TCB
AIOCANCELNOWAIT      EQU   X'04'  Nowait option on cancel
AIOCANCELNONOTIFY    EQU   X'02'  NoNotify option on cancel
AIOTCBAFFINITY       EQU   X'01'  TCB Affinity I/O
AIOCFLAGS2           DS    XL1    Control Flags2
AIOUSERKEY           EQU   X'F0'  Caller's User's Key bit positions
AIOUSEUSERKEY        EQU   X'08'  Use User's Key for moves
AIOTHLICOMECB        EQU   X'04'  AioEcbPtr points tp ThliComEcb
AIOCOMMBUFF          EQU   X'02'  Common Area Buffer
AIOMSGIOVALET        DS    F      Alet for recvmsg/sendmsg IOV
AIOIOVBUFALET        DS    F      Alet for all IOV buffers
*
AIORV                DS    F      Return value
AIORC                DS    F      Return code
AIORSN               DS    F      Reason code
*
AIOPOSIXFLAGS        DS    XL4    Posix flags
&AIOEXITPTR31        DS    F      Pointer to user exit
AIOEXITDATA          DS    CL8    User Data for exit program
AIOECBPTR            DS    F      ECB address
AIOSOCKADDRLEN       DS    F      Sockaddr length
&AIOSOCKADDRPTR31    DS    F      Sockaddr pointer
AIOTIMEOUT           DS    F      TimeOut Value in Milli-seconds
AIOACEE              DS    F      SRB ACEE for MLS
AIOSICODE            DS    XL2    Signal si_code
AIORES06             DS    CL2    Reserved
AIOLEN               DS    F      (Output,debug) Len of AIO rcvd
AIOENDVER1           DS    0D     End of Original Aiocb
* -------- 64-Bit Extension
AIOLP64              DS    0D
&AIOBUFFPTR64        DS    AD     Buffer Ptr
```

```
&AIOEXITPTR64        DS    AD     Exit Program Address
&AIOSIGEVENT64       DS    CL32   SigEvent Structure
&AIOSOCKADDRPTR64    DS    AD     Sockaddr Ptr
                     DS    CL8
* -------- Version 3 Extension
* ------------------------------ 31-Bit Version
AIORES01             DS    F      RESERVED
AIOLOCSOCKADDRPTR    DS    F      Local Sockaddr Ptr for ANR
AIOLOCSOCKADDRLEN    DS    F      Local Sockaddr Len for ANR
AIOANRSOCKET         DS    F      Accepted Socket for ANR
                     DS    CL48
AIOENDVER3           DS    0D     End of Version 3 extension

AIOEND               DS    0D     End of Aiocb
*
AIO#LENGTH           EQU   *-AIO  Length of this structure
*
** AIO command values
AIO#ACCEPT           EQU   126
AIO#CONNECT          EQU   128
AIO#READ             EQU   43
AIO#WRITE            EQU   54
AIO#READV            EQU   133
AIO#WRITEV           EQU   144
AIO#RECV             EQU   134
AIO#SEND             EQU   138
AIO#RECVFROM         EQU   135
AIO#SENDTO           EQU   140
AIO#RECVMSG          EQU   243
AIO#SENDMSG          EQU   244
AIO#ANR              EQU   264
AIO#BRLOCK           EQU   264
AIO#SELPOLL          EQU   2
AIO#CANCEL           EQU   1
*
** AIO notify type
AIO#POSIX            EQU   0
AIO#MVS              EQU   1
AIO#MSGQ             EQU   2
*
** AIO Message Event Structure
*        For AioNotifyType of AIO#MSGQ the AioMsgEvent
*          structure overlays AioSigEvent (31-bit location).
*        Msgbuf and Msgbuf64 are defined in BPXYMSG.
*        IPC_NOWAIT is defined in BPXYIPCP.
*
AIO_BEFORE_MSGEV     DS    0C             Note current position
                     ORG   AIOMSGEVENT
AIOMSGEV_QID         DS    F              Msg Queue Id
AIOMSGEV_SIZE        DS    H              Length of Msg_mtext
AIOMSGEV_FLAG        DS    H              0 or IPC_NOWAIT
AIOMSGEV_ADDR64      DS    D              Amode(64)-> MsgBuf64
                     ORG   AIOMSGEV_ADDR64
AIOMSGEV_ADDRH       DS    F
AIOMSGEV_ADDR        DS    F              Amode(31)-> MsgBuf
                     ORG   AIO_BEFORE_MSGEV    Return to above
*
AIO#MSGTEXTMAX       EQU  240             Max Msg_MText
*
** AIO Signal Event
SIGEVENT        DSECT ,
SIGEVENT             DS 0F
SIGEV_NOTIFY         DS    F              NOTIFICATION TYPE
SIGEV_SIGNO          DS    F              SIGNAL NUMBER
SIGEV_VALUE          DS    &AIOPTRSIZE    SIG VALUE
                     ORG   SIGEV_VALUE
SIVAL_INT            DS    F
```

```
                    ORG   SIGEV_VALUE
SIVAL_PTR           DS    &AIOPTRSIZE
SIGEV_NOTIFY_FUNCTION   DS &AIOPTRSIZE   NOTIF. FUNCTION
SIGEV_NOTIFY_ATTRIBUTES DS &AIOPTRSIZE   NOTIF. ATTRIBUTES
*
SIGEV#LENGTH        EQU   *-SIGEVENT  Length of this structure
*
*    SIGEV_NOTIFY Values
SIGEV_SIGNAL        EQU   0             GENERATE A SIGNAL
SIGEV_NONE          EQU   1             DON'T GENERATE SIGNAL
SIGEV_THREAD        EQU   2             Call Notif. function
*
** AIOTIMEOUT VALUES
AIO#FOREVER         EQU   0             NO TIMEOUT, JUST WAIT
AIO#NOWAITING       EQU   X'FFFFFFFF'   NO WAITING, JUST CHECK
** AIO CANCEL RETURN VALUES
AIO_CANCELED        EQU   1             ALL CANCELS SUCCESSFUL
AIO_NOTCANCELED     EQU   2             AT LEAST 1 CANCEL FAILED
AIO_ALLDONE         EQU   3             NONE CANCELED, ALL COMP
*
** BPXYAIO End
```

BPXYATT — Map file attributes for chattr and fchattr

```
           BPXYATT    ,
**  BPXYATT:  File attributes for chattr system call
**  Used By:  CHR FCR
ATT                 DSECT ,
ATTBEGIN            DS    0D
*
ATTID               DC    C'ATT '  Eye Catcher
ATTVERSION          DC    AL2(ATT#VER)
*                               Version of this structure
ATTRES01            DS    CL2    Reserved
ATTSETFLAGS         DS    0XL4   Flags - which fields to set
ATTSETFLAGS1        DS    X      Flag byte 1
ATTMODECHG          EQU   X'80'  1 = Change to the mode indicated
ATTOWNERCHG         EQU   X'40'  1 = Change to Owner indicated
ATTSETGEN           EQU   X'20'  1 = Set General attributes
ATTTRUNC            EQU   X'10'  1 = Truncate size
ATTATIMECHG         EQU   X'08'  1 = Change the Atime
ATTATIMETOD         EQU   X'04'  1 = Change to the Current Time
ATTMTIMECHG         EQU   X'02'  1 = Change the Mtime
ATTMTIMETOD         EQU   X'01'  1 = Change to the Current Time
ATTSETFLAGS2        DS    X      Flag byte 2
ATTMAAUDIT          EQU   X'80'  1 = Modify auditor audit info
ATTMUAUDIT          EQU   X'40'  1 = Modify user audit info
ATTCTIMECHG         EQU   X'20'  1 = Change the Ctime
ATTCTIMETOD         EQU   X'10'  1 = Change Ctime to the Current
*                                    Time
ATTREFTIMECHG       EQU   X'08'  1 = Change the RefTime
ATTREFTIMETOD       EQU   X'04'  1 = Change RefTime to Current Time
ATTFILEFMTCHG       EQU   X'02'  1 = Change File Format
ATTRES04            EQU   X'01'  Reserved
ATTSETFLAGS3        DS    X      Reserved
ATTRES05            EQU   X'80'  Reserved
ATTCHARSETIDCHG     EQU   X'40'  1 = Change File tag
ATTLP64TIMES        EQU   X'20'  1 = Use 64-bit times
ATTSECLABELCHG      EQU   X'10'  1 = Set Seclabel
ATTSETFLAGS4        DS    X      Reserved
ATTMODE             DS    F      File Mode, mapped by BPXYMODE
ATTUID              DS    F      User ID of the owner of the file
ATTGID              DS    F      Group ID of the Group of the file
ATTGENMASK          DS    0XL4   Mask to indicate which General
*                                attributes bits to modify
*                                --Must match AttGenValue
```

```
ATTOPAQUEMASK        DS    XL3    Opaque attribute flags - Reserved
*                                  for ADSTAR use
ATTVISIBLEMASK       DS    X      Visible attribute flags
ATTNODELFILESMASK    EQU   X'20'  Files should not be deleted
ATTSHARELIBMASK      EQU   X'10'  Shared Library
ATTNOSHAREASMASK     EQU   X'08'  No shareas flag
ATTAPFAUTHMASK       EQU   X'04'  APF authorized flag
ATTPROGCTLMASK       EQU   X'02'  Program controlled flag
ATTGENVALUE          DS    0XL4   General attribute values
*                                  --Must match AttGenMask
ATTOPAQUE            DS    XL3    Opaque attribute flags - Reserved
*                                  for ADSTAR use
ATTVISIBLE           DS    X      Visible attribute flags
ATTNODELFILES        EQU   X'20'  Files should not be deleted
ATTSHARELIB          EQU   X'10'  Shared Library
ATTNOSHAREAS         EQU   X'08'  No shareas flag
ATTAPFAUTH           EQU   X'04'  APF authorized flag
ATTPROGCTL           EQU   X'02'  Program controlled flag
ATTSIZE              DS    0D     File Size in bytes, for regular
*                                  files.  Unspecified, for others
ATTSIZE_H            DS    F      First word of size
ATTSIZE_L            DS    F      Second word of size
ATTATIME             DS    F      Time of last access
ATTMTIME             DS    F      Time of last data modification
ATTAUDITORAUDIT      DS    F      Area for auditor audit info
ATTUSERAUDIT         DS    F      Area for user audit info
ATTCTIME             DS    F      Time of last file status change
*                                  Time is in seconds since
*                                  00:00:00 GMT, Jan. 1, 1970
ATTREFTIME           DS    F      Reference time
ATTENDVER1           DS    0D     End of Version 1
ATTFILEFMT           DS    XL1    File Format
ATTRES02             DS    XL3    Reserved for future
ATTFILETAG           DS    F      File tag (see BPXYSTAT)
ATTRES03             DS    CL8    Reserved for future
ATTENDVER2           DS    0D     End of Version 2
*
ATTATIME64           DS    D      Access Time
ATTMTIME64           DS    D      Data Mod Time
ATTCTIME64           DS    D      Medadata Change Time
ATTREFTIME64         DS    D      Reference Time
ATTSECLABEL          DS    CL8    Security Label
ATTVER3RES02         DS    CL8    Reserved for R6
ATTENDVER3           DS    0D     End of Version 3
*
*   Constants
*
ATT#VER              EQU   ATT#VER03 Current version
ATT#VER01            EQU   1         Version 1 of this structure
ATT#VER02            EQU   2         Version 2 of this structure
ATT#VER03            EQU   3         Version 3 of this structure
ATT#LENGTH           EQU   *-ATTBEGIN                              X
                                     Length of ATT
ATT#VER1LEN          EQU   ATTENDVER1-ATTBEGIN                     X
                                     Length of Version 1 ATT
ATT#VER2LEN          EQU   ATTENDVER2-ATTBEGIN                     X
                                     Length of Version 2 ATT
ATT#VER3LEN          EQU   ATTENDVER3-ATTBEGIN                     X
                                     Length of Version 3 ATT
** BPXYATT  End
```

BPXYAUDT — Map flag values for chaudit and fchaudit

```
          BPXYAUDT   ,
** BPXYAUDT: External audit flags
**  Used By: CHA, FCA
```

```
AUDT                DSECT ,
AUDTREADACCESS      DS    XL1    Read Access Auditing Flags
AUDTREADFAIL        EQU   X'02'  1 = audit failing read accesses
AUDTREADSUCC        EQU   X'01'  1 = audit successful read accesses
AUDTWRITEACCESS     DS    XL1    Write Access Auditing Flags
AUDTWRITEFAIL       EQU   X'02'  1 = audit failing write accesses
AUDTWRITESUCC       EQU   X'01'  1 = audit successful write accesses
AUDTEXECACCESS      DS    XL1    Execute/Search Auditing Flags
AUDTEXECFAIL        EQU   X'02'  1 = audit failing exec or search
AUDTEXECSUCC        EQU   X'01'  1 = audit successful exec or search
AUDTRSRV            DS    XL1    Flag byte 4 -Reserved
AUDT#LENGTH         EQU   *-AUDT Length of this structure
** BPXYAUDT End
```

BPXYBRLK — Map byte range lock request for fcntl

```
           BPXYBRLK   ,
** BPXYBRLK: External Byte Range Locking interface control block
**  Used By: FCT
BRLK                DSECT ,
L_TYPE              DS    H      Requested lock type:
F_RDLCK             EQU   1       Shared or read lock
F_WRLCK             EQU   2       Exclusive or write lock
F_UNLCK             EQU   3       Unlock
L_WHENCE            DS    H      Flag for starting offset
L_START             DS    0CL8   Relative offset in bytes
L_START_H           DS    F      High word of relative offset
L_START_L           DS    F      Low word of relative offset
L_LEN               DS    0CL8   Size of lock in bytes
L_LEN_H             DS    F      High word of size of lock in bytes
L_LEN_L             DS    F      Low word of size of lock in bytes
L_PID               DS    F      Process ID of process holding lock
BRLK#LENGTH         EQU   *-BRLK Length of this area
** BPXYBRLK End
```

BPXYCCA — Map input/output structure for __console()

AMODE 64 callers use “BPXYCCA — Map input/output structure for __console()” on page 1088.

```
           BPXYCCA    ,
**  BPXYCCA:  Msg Attributes for console_np service
**  Used By:  CCS
CCA                 DSECT ,
CCABEGIN            DS    0D
*
CCAVERSION          DC    AL2(CCA#VER)
*                         Version of this structure
CCARES01            DS    CL2    Reserved
CCAMSGLENGTH        DS    F      Length of msg pointed to by CCAMSGPTR
CCAMSGPTR           DS    A      Pointer to Msg text
CCARES02            DS    CL8    Reserved
CCAENDVER1          DS    0F     End of Version 1
CCASTARTVER2        DS    0F     Start of Version 2
CCARES03            DS    F      Reserved
CCAWTOPARMS         DS    0F     Start of WTO message attributes
CCAROUTCDELIST      DS    A      Pointer to list of message routing   X
                                 codes
CCARES04            DS    F      Reserved
CCADESCLIST         DS    A      Pointer to list of message           X
                                 descriptor codes
CCARES05            DS    F      Reserved
CCAWMCSFLAGS        DS    0F     WTO MCS Flags
CCAMCSFLAGB1        DS    XL1    MCS flags byte 1
```

```
CCAHRDCPY           EQU   X'80'  Send message to hard copy log only
CCAMCSFLAGB2        DS    XL1    MCS flags byte 2
CCAMCSFLAGB3        DS    XL1    MCS flags byte 3
CCAMCSFLAGB4        DS    XL1    MCS flags byte 4
CCAWTOTOKEN         DS    F      Token for message to be issued
CCAMSGIDPTR         DS    A      Pointer to location where message    X
                                 is is stored by BPX1CCS
CCARES06            DS    F      Reserved
CCARES07            DS    F      Reserved
CCADOMPARMS         DS    0F     Delete message parameters
CCADOMTOKEN         DS    F      Token of message(s) to be deleted
CCAMSGIDLIST        DS    A      Pointer to list of message ids to    X
                                 be deleted
CCARES08            DS    F      Reserved
CCAENDVER2          DS    0D     End of version 2
CCASTARTVER3        DS    0CL40  Start of version 3
CCAMODCARTPTR       DS    A      Pointer to 8  byte CART returned for X
                                 MODIFY/STOP command
CCARES09            DS    CL4    Not used for amode 31
CCAMODCONSOLEIDPTR  DS    A      Pointer to 4 byte ConsoleID returned X
                                 for MODIFY/STOP command
CCARES10            DS    CL4    Not used for amode 31
CCAMSGCART          DS    CL8    Supplied - CART to be specified on   X
                                 WTO when message is issued
CCAMSGCONSOLEID     DS    CL4    Supplied - ConsoleID to be specified X
                                 on WTO when message is issued
CCARES11            DS    CL12   Reserved
CCAENDVER3          DS    0D     End of version 3
*
*   Constants
*
CCA#VER             EQU   CCA#VER02 Current version
CCA#VER01           EQU   1        Version 1 of this structure
CCA#VER02           EQU   2        Version 2 of this structure
CCA#VER03           EQU   3        Version 3 of this structure
CCA#LENGTH          EQU   *-CCABEGIN                                   X
                                 Length of CCA
CCA#VER1LEN         EQU   CCAENDVER1-CCABEGIN                          X
                                 Length of Version 1 CCA
CCA#VER2LEN         EQU   CCAENDVER2-CCABEGIN                          X
                                 Length of Version 2 CCA
CCA#VER3LEN         EQU   CCAENDVER3-CCABEGIN                          X
                                 Length of Version 3 CCA
** BPXYCCA  End
```

BPXYCID — Map the returning structure for getclientid()

```
          BPXYCID    ,
*
*********************************************************************
** BPXYCID: z/OS UNIX ClientId Structure                            *
** Used By: Sockets LFS                                            *
*********************************************************************
*
CID                 DSECT ,          ClientId structure
CIDBEGIN            DS    0D
*
CIDDOMAIN           DS    F          Domain
CIDNAME             DS    CL8        Address space name
CIDTASK             DS    CL8        Subtask name
CIDRESERVED         DS    CL20       Reserved
*
CID#LENGTH          EQU   *-CID      Constant - Fixed length of CID
*
                    ORG   CIDNAME
CIDNAMEUPPER        DS    F          Binary zeroes
```

```
CIDPID              DS   F           Process Id
*
                    ORG  CIDRESERVED
CIDTYPE             DS   X           Type of request
CIDSPECIFIC         DS   CL19
*
                    ORG  CIDSPECIFIC
                    DS   CL3
CIDSOCKTOKEN        DS   F           Returned token
                    ORG  ,
*
CID#CLOSE           EQU  1           Close socket
CID#SELECT          EQU  2           Giver will do select
*
*
*
** BPXYCID End
```

BPXYCONS — Constants used by services

BPXYCONS is composed only of EQUates. DSECT= is allowed but ignored.

```
** BPXYCONS: Syscall constants
**  Used By: Many syscalls
DFLT_ARG_MAX         EQU   1048576  Constant for default ARG_MAX  @EGC
DFLT_CHILD_MAX       EQU   6    Constant for default CHILD_MAX
*                             (_POSIX_CHILD_MAX)
DFLT_CLK_TCK         EQU   100  Constant for default CLK_TCK
*                             (100 ticks per second)
DFLT_NGROUPS_MAX     EQU   8191 Constant for default NGROUPS_MAX
*                             (RACF Maximum value)
DFLT_OPEN_MAX        EQU   16   Constant for default OPEN_MAX
*                             (_POSIX_OPEN_MAX)
DFLT_TZNAME_MAX      EQU   9    Constant for default TZNAME_MAX
DFLT_JOB_CONTROL     EQU   1    Constant for default JOB_CONTROL
DFLT_SAVED_IDS       EQU   1    Constant for default SAVED_IDS
DFLT_VERSION         EQU   199009  Constant for default VERSION
DFLT_THREAD_TASKS_MAX_NP  EQU  50  Constant default THREAD_TASKS_MAX_NP
DFLT_USERIDLEN_MAX   EQU   8    Max characters for a userid       @DKA
DFLT_PASSWDLEN_MAX   EQU   8    Max characters for a password     @DKA
DFLT_PASSWDPHRLEN_MAX EQU  100 Max characters for password phrase @EBA
DFLT_2_CHAR_TERM     EQU   1    Constant default SC_2_CHAR_TERM  @P1A
                SPACE ,
* items from sysconf()
SC_ARG_MAX           EQU   1    Constant for querying ARG_MAX
SC_CHILD_MAX         EQU   2    Constant for querying CHILD_MAX
SC_CLK_TCK           EQU   3    Constant for querying CLK_TCK
SC_JOB_CONTROL       EQU   4    Constant for querying JOB_CONTROL
SC_NGROUPS_MAX       EQU   5    Constant for querying NGROUPS_MAX
SC_OPEN_MAX          EQU   6    Constant for querying OPEN_MAX
SC_SAVED_IDS         EQU   7    Constant for querying SAVED_IDS
SC_TZNAME_MAX        EQU   9    Constant for querying TZNAME_MAX
SC_VERSION           EQU   10   Constant for querying VERSION
SC_THREAD_TASKS_MAX_NP EQU 11   Constant to query THREAD_TASKS_MAX_NP
SC_2_CHAR_TERM       EQU   12   Constant for querying VERSION    @P1A
SC_THREADS_MAX_NP    EQU   13   Constant to query THREADS_MAX_NP @D5A
SC_MMAP_MEM_MAX_NP   EQU   14   Constant to query MMAP_MEM_MAX_NP@DAA
SC_TTY_GROUP         EQU   15   Constant to query TTY GROUP      @PEA
SC_PAGESIZE          EQU   16   Constant to query Page Size      @D2A
SC_PAGE_SIZE         EQU   16   Constant to query Page Size      @D2A
                SPACE ,
* wait function code                                           @DCA
#WAIT3               EQU   1    wait3() function code          @DCA
#WAITID              EQU   2    waitid() function code         @DCA
                SPACE ,
* items from waitf()
WNOHANG              EQU   1    Wait, do not suspend execution
```

```
WUNTRACED            EQU   2    Wait, return status of stopped child
WCONTINUED           EQU   4    Wait, return status of continued child
*                                                                @DCA
WEXITED              EQU   8    Wait for process that have exited
*                                                                @DCA
WSTOPPED             EQU  16    Wait, return status of stopped child
*                                                                @DCA
WNOWAIT              EQU  32    Wait, return status of a child without
*                               changing the state.  The child can be
*                               waited for again.                @DCA
                SPACE ,
* waitid() id type options                                       @DCA
P_PID                EQU   0    Wait for the child with a process ID
*                                                                @DCA
P_PGID               EQU   1    Wait for any child with a process
*                               group ID                         @DCA
P_ALL                EQU   2    Wait for any child               @DCA
                SPACE ,
* BPX1PTX Options
PTEXITTHREAD         EQU   0    Pthread exit
PTGETNEWTHREAD       EQU   1    Pthread get new
PTFAILIFLASTTHREAD   EQU   2    Pthread fail if last thread      @D4A
                SPACE ,
QUIESCE_TERM         EQU   1    Quiesce threads type = term      @D3A
QUIESCE_FORCE        EQU   2    Quiesce threads type = force     @D3A
QUIESCE_QUERY        EQU   3    Alias of pthread_query           @P4C
PTHREAD_QUERY        EQU   3    Quiesce threads type = query     @P4A
QUIESCE_FREEZE       EQU   4    Quiesce threads type = freeze    @D6A
QUIESCE_UNFREEZE     EQU   5    Quiesce threads type = unfreeze  @D6A
FREEZE_THIS_THREAD   EQU   6    Quiesce threads type = freezeme  @D6A
* Skip 7 because of collision with BPXZCONS Freeze_Force
FREEZE_EXIT          EQU   8    Quiesce threads type = freeze_exit
QUIESCE_SRB          EQU   9    Quiesce threads type = SRBs      @DGA
* Skip 10 and 11 due to collision with BPXZCONS Freeze/Unfreeze Fast
*                                                                @P6A
                SPACE ,
PTHREAD_INTR_ENABLE#       EQU 0  Cancel request type = enabled
PTHREAD_INTR_DISABLE#      EQU 1  Cancel request type = disabled
PTHREAD_INTR_CONTROLLED#   EQU 0  Cancel request type = controlled
PTHREAD_INTR_ASYNCHRONOUS# EQU 1  Cancel request type = Asynchronous
                SPACE ,
STDIN_FILENO         EQU   0    Standard input value, file descriptor
STDOUT_FILENO        EQU   1    Standard output value, file descriptor
STDERR_FILENO        EQU   2    Standard error value, file descriptor
                SPACE ,
DUBTHREAD            EQU   0    Dub a thread default setting     @L1A
DUBPROCESS           EQU   1    Dub a process default setting    @L1A
DUBTASKACEE          EQU   2    Dub a task ACEE setting          @02A
DUBPROCESSDEFER      EQU   4    Dub a process - but defer dub    @04A
DUBNOSIGNALS         EQU   8    Dub a process - no signals       @DWA
DUBJOBPERM           EQU   16   Dub as a permanent Job           @DYA
DUBNOJSTUNDUB        EQU   32   Dub process such that jobstep does not
*                                                                @DYA
DUBABENDCALLS        EQU   64   Dub process such that system calls
*                               abend during a shutdown/restart window
*                                                                @DYA
DUBUNIQUEACEE        EQU   128                                   @07A
DUBFAILNOTREADY      EQU   256  Fail dub if kernel is not up     @F1A
                SPACE ,
STL_MAX_TASKS        EQU   1    Replace MaxThreadTask only       @D7A
STL_MAX_THREADS      EQU   2    Replace MaxThreads only          @D7A
STL_SET_BOTH         EQU   3    Replace both limits              @D7A
                SPACE ,
NICE_ZERO            EQU   20   Default Process Scheduling Priority
                SPACE ,
PRIO_PROCESS         EQU   1    Looking for a specific process ID
PRIO_PGRP            EQU   2    Looking for processes in a process grp
```

```
PRIO_USER            EQU   3    Looking for processes for a user ID
                 SPACE ,
CPRIO_ABSOLUTE       EQU   1    Priority value is an absolute value
CPRIO_RELATIVE       EQU   2    Priority value is a relative value
                 SPACE ,
*   Define equates for memory map
PROT_READ            EQU   1    Mapped data can be read          @DAA
PROT_WRITE           EQU   2    Mapped data can be written       @DAA
PROT_NONE            EQU   4    Mapped data cannot be accessed   @DAA
PROT_EXEC            EQU   8    Mapped data can be executed (treated
*                               as PROT_READ)                    @DAA
                 SPACE ,
MAP_PRIVATE          EQU   1    Changes to the mapped data are private
MAP_SHARED           EQU   2    Changes to the mapped data are shared
MAP_FIXED            EQU   4    Interpret map address exactly    @DAA
MAP_MEGA             EQU   8    Use megabyte allocations         @D4A
                 SPACE ,
MS_SYNC              EQU   1    Performs synchronous writes      @DAA
MS_ASYNC             EQU   2    Performs asynchronous writes     @DAA
MS_INVALIDATE        EQU   4    Invalidate the cached memory mapped
*                               pages                            @DAA
                 SPACE ,
*   Define equates for spawn                                   @D9A
SPAWN_FDCLOSED       EQU  -1    Do not inherit this file desc  @D9A
                 SPACE ,
RLIMIT_CORE          EQU   4    Limit size of core dump          @DBA
RLIMIT_CPU           EQU   0    Limit CPU time per process       @DBA
RLIMIT_FSIZE         EQU   1    Limit file size                  @DBA
RLIMIT_NOFILE        EQU   6    Limit number of open files       @DBA
RLIMIT_AS            EQU   5    Limit address space size         @DBA
RLIMIT_MEMLIMIT      EQU   7    Limit storage above the bar      @E0A
                 SPACE ,
RLIM_INFINITY        EQU   2147483647 No limit value             @DBA
                 SPACE ,
RUSAGE_SELF          EQU   0    Rusage for current process       @DBA
RUSAGE_CHILDREN      EQU  -1    Rusage for terminated children   @DBA
                 SPACE ,
*   Define equates for querydub output status
QDB_DUBBED_FIRST     EQU   1    Task has already been dubbed.
*                               This task and this RB caused the
*                               dub.                             @DCA
QDB_DUBBED           EQU   2    Task has already been dubbed.
*                               Other task or other RB caused
*                               the dub                          @DCA
QDB_DUB_MAY_FAIL     EQU   4    Task has not been dubbed, but may
*                               fail if attempted.  Most likely
*                               reason for failure will be a missing
*                               or incomplete user security profile,
*                               or OMVS segment not defined      @DCA
QDB_DUB_OKAY         EQU   8    Task has not been dubbed, and should
*                               succeed if attempted             @DCA
QDB_DUB_AS_PROCESS   EQU   16   Task has not been dubbed, but its
*                               address space has. New task will dub
*                               as another process within the address
*                               space                            @DCA
QDB_DUB_AS_THREAD    EQU   32   Task has not been dubbed, but its
*                               address space has. New task will dub as
*                               a thread within the process      @DCA
                 SPACE ,
*   Define equates for oe_env_np syscall function codes
ENQWAIT_PROCESS      EQU   1    Examine/Change ENQ wait interruption
*                               state                            @P6A
FREEZE_EXIT_REG      EQU   2    Register/deregister an exit
*                               for pthread_quiesce(freeze_exit) @P6A
MVS_USERID           EQU   3    Retrieve MVS userid of invoker   @P7A
ENV_TOGGLE_SEC       EQU   4    Toggle btw task/process security @P7A
DFP_CLEANUP_EXIT_REG EQU   5    Register DFP Close cleanup exit  @01A
```

```
BPXK_PARAMETER       EQU   6    Env Vars to Kernel               @P9A
ENV_STOR_SERVICE     EQU   7    Swappable or Non-Swappable
*                               address space                   @DJA
QUICK_FREEZE_EXIT_REG EQU  8    Register/deregister an exit for
*                               fast pthread_quiesce_and_get    @DUA
SHUTDOWN_REG         EQU  9     Register/block for shutdown
*                               processing                      @DXA
WRITE_DOWN           EQU  10    Query/Alter write_down state of an
*                               ACEE (MLS support)              @E5A
PIDXFER_QUERY        EQU  11    Query if process image is a result of
*                               a PIDXFER                       @DXA
QUERY_MODE           EQU  12    Query AMODE/RMODE/AMODE capability of
*                               target proess                   @E1A
MUST_STAY_CLEAN      EQU  13    ENABLE/QUERY Must Stay Clean state of
*                               the invoking process            @E9A
                SPACE ,
*   Define equates for possible options of ENV_STOR_SERVICE
*   For future additions, make equates multiples of 2
*                                                              4@PED
BPX_SWAP         EQU   1    Make the address space swappable    @PEC
BPX_NONSWAP      EQU   2    Make the address space non-swappable
*                                                              @PEC
                SPACE ,
*                                                              @E9A
*   Define equates for possible options of MUST_STAY_CLEAN     @E9A
* @E9A
MSC_QUERY        EQU   0    Query current Must Stay Clean state  @E9A
MSC_ENABLE       EQU   1    Enable Must Stay Clean state         @E9A
*                                                              @E9A
*   Define possible output for MSC_QUERY option                @E9A
* @E9A
MSC_DISABLED     EQU   0     Query result: disabled              @E9A
MSC_ENABLED      EQU   1     Query result: enabled even accoss Job
*                            Step termination                  @E9A
MSC_ENABLED_COND EQU   2     Query result: enabled conditionally,
*                            Job Step termination will disable  @E9A
*
*   Define equates for all possible options for WRITE_DOWN function
*   code of BPX1ENV
*
WD_OUERY         EQU   0     Query write_down                  @E5A
WD_ACTIVATE      EQU   1     Activate write_down               @E5A
WD_INACTIVATE    EQU   2     Inactivate write_down             @E5A
WD_RESET         EQU   3     Reset write_down to default       @E5A
*
WD_SCOPE_AS      EQU   1     Target ACEE is AS                 @E5A
WD_SCOPE_THD     EQU   2     Target ACEE is task               @E5A
*
WD_IS_ACTIVE     EQU   1     Query result: active              @E5A
WD_IS_INACTIVE   EQU   0     Query result: inactive            @E5A
*
*   Define equates for QUERY_MODE return values                @E1A
*                                                              @E1A
BIT24_MODE           EQU   1    24 bit AMODE, RMODE or AMODE cap.@E1A
BIT31_MODE           EQU   2    31 bit AMODE, RMODE or AMODE cap.@E1A
BIT64_MODE           EQU   3    64 bit AMODE, RMODE or AMODE cap.@E1A
AMODE_INITIALIZING   EQU   4    AMODE for process not known yet  @PNA
                SPACE ,
*   Define equates for possible options of ENV_SHUTDOWN_REG
*                                                              @DYA
ENV_REGISTERBLOCK     EQU  1   Register to Block Shutdown       @DYA
ENV_REGISTERPERMP     EQU  2   Register as a Permanent job/proc @DYA
ENV_DEREGISTERBLOCK   EQU  3   Dereg as a blocking job/proc     @DYA
ENV_DEREGISTERPERM    EQU  4   Dereg as a permanent job/proc    @DYA
ENV_REGISTERNOTIFY    EQU  5   Register as a notify job/proc    @DYA
ENV_DEREGISTERNOTIFY  EQU  6   Dereg as a notify job/proc       @DYA
ENV_REGISTERJOB       EQU  1   Register Job                     @DYA
```

```
ENV_REGISTERPROC      EQU  2   Register Process                 @DYA
*                                                              @DYA
                SPACE ,
*   Define equates for PIDXFER_QUERY return values             @DXA
*                                                              @DXA
PIDXFER_YES          EQU   1    Process was PIDXFERed           @DXA
PIDXFER_NO           EQU   2    Process was not PIDXFERed       @DXC
                SPACE ,
*   Define equates for versions of OSMF on BPXESMF syscall
OSMF_VER_HOM1110     EQU   1    Version 1 of OSMF, for HOM1110  @P5A
OSMF_VER_HOM1120     EQU   2    Version 2 of OSMF, for HOM1120  @P5A
OSMF_VER_HOM1130     EQU   3    Version 3 of OSMF, for HOM1130  @P5A
                SPACE ,
*   Define equates for task ecurity syscall function codes
TLS_CREATE_THREAD_SEC# EQU 1    Build Task Security            @P7A
TLS_DELETE_THREAD_SEC# EQU 2    Delete Task Security           @P7A
TLS_TASK_ACEE#         EQU 3    set posix identity from task ACEE@DYA
TLS_TASK_ACEE_USP#     EQU 4    User passed ACEE/USP pair      @E6A
TLS_DAEMON_THREAD_SEC# EQU 5    Build unauthenticated Security @E8A
TLS_IDENTITY_USERID#   EQU 1    User identity: 1-8 char userid @P7A
TLS_IDENTITY_UID#      EQU 2    User identity: 4-byte uid      @P7A
TLS_IDENTITY_CERT#     EQU 4    User identity: CERT structure  @PFA
                SPACE ,
*   Define equates for __Security syscall
*    __Security function code
SECURITY_CREATE#       EQU 1    Create new security environment  @DKA
SECURITY_CERTREG#      EQU 2    Register certificate with caller @DOA
SECURITY_CERTDEREG#    EQU 3    DeReg certificate from caller    @DOA
SECURITY_CERTAUTH#     EQU 4    Authorize certificate from caller@E3A
*   __Security user identity
SECURITY_USERID#       EQU 1    User identity is 1-8 char userid @DKA
SECURITY_CERTIFICATE#  EQU 2    User identity is a certificate   @DKA
                SPACE ,
*   Define equates for convert_id_np (BPX1CID) syscall function codes
CID_GET_UUID#          EQU 1    Retrieve UUID                  @P8A
CID_GET_USERID#        EQU 2    Retrieve userid                @P8A
                SPACE ,
*   Define equates for __pid_affinity (BPX1PAF) syscall function codes
PAF_ADD_PID#           EQU 1    Add PID to affinity list       @DMA
PAF_DELETE_PID#        EQU 2    Delete PID from affinity list  @DMA
                SPACE ,
* Define equates for auth_check_resource_np  syscall access types
ACK_READ#              EQU 1    Test READ access               @P8A
ACK_UPDATE#            EQU 2    Test UPDATE access             @P8A
ACK_CONTROL#           EQU 3    Test CONTROL access            @P8A
ACK_ALTER#             EQU 4    Test ALTER access              @P8A
                SPACE ,
*   The high order two bytes of the reason codes returned by
*   OpenMVS services contains a value that is used to qualify
*   the contents of the low order two bytes. If the contents of
*   the high-order two bytes are within the range of #CMID_LO to
*   #CMID_HI, the error represented by the reason code is defined
*   by OpenMVS. If the contents of the high order two bytes lie
*   outside the range, the error represented by the reason code
*   is not an OpenMVS reason code.
#CMID_LOW            EQU   0000   Low range
#CMID_HI             EQU   8447   High range
*   Define equates for console cmd
CC_MAX_MSG_LENGTH       EQU 17850 Max Wto string length for SUs   @DIC
CC_MAX_MSG_LENGTH_NONSU EQU 17780 Max Wto string length for nonSU @DIC
CC_MODIFY_BUFFER_LENGTH EQU 128  Length of Modify Buffer          @PAC
CONSOLE_MODIFY          EQU   1  Service interrupted by Modify    @PAC
CONSOLE_STOP            EQU   2  Service interrupted by Stop      @PAC
*   Define equates for server_init syscall ManagerType parameter
SRV_WORKMGR             EQU   1  Work Manager services requested  @DGA
SRV_QUEUEMGR            EQU   2  Queue Mgr services requested     @DGA
SRV_SERVERMGR           EQU   4  Server Mgr services requested    @DGA
```

```
SRV_SERVERMGRDYNAMIC    EQU   8  Server Mgr With Dynamic mngt     @DSA
*   Define equates for server_pwu syscall FcnCode parameter
SRV_PUT_NEWWRK          EQU   1  Put new work function requested  @DGA
SRV_PUT_SUBWRK          EQU   2  Put sub work function requested  @DGA
SRV_TRANSFER_WRK        EQU   4  Transfer work function requested @DGA
SRV_GET_WRK             EQU   8  Get work function requested      @DGA
SRV_REFRESH_WRK         EQU  16  Refresh work fcn requested       @DGA
SRV_END_WRK             EQU  32  End work function requested      @DGA
SRV_DEL_ENC             EQU  64  Delete Enclave Fcn requested     @DGA
SRV_DISCONNECT          EQU 128  Disconnect from WLM              @PBA
SRV_DISCONNECT_COND     EQU 256  Disconnect conditional from WLM  @PCA
*                       EQU 512  Reserved for internal use        @PGA
*                                See BPXZCONS                     @PGA
*   Define equates for BPX1SLK syscall LockFcnCode parameter
SLK_OBTAIN              EQU   1  Obtain function request          @DZA
SLK_OBTAIN_COND         EQU   2  Obtain conditional function req  @DZA
SLK_INIT                EQU   4  Initialization function request  @DZA
SLK_DESTROY             EQU   8  Destroy function request         @DZA
SLK_RELEASE             EQU  16  Release function request         @DZA
*   Define equates for BPX1SLK syscall LockReqType parameter
SLK_NORMAL              EQU   1  Normal request type              @DZA
SLK_ERRORCHECK          EQU   2  Errorcheck request type          @DZA
SLK_RECURSIVE           EQU   4  Recursive request type           @DZA
*   Define equates for BPX1SLK syscall LockType parameter
SLK_EXCLUSIVE           EQU   1  Exclusive lock type              @DZA
SLK_SHARED              EQU   2  Shared lock type                 @DZA
*
* Constants for BPX1PCT pfsctl
*
*  Constants for BPXVRCAC - LFS Cache                            @DHA
PC#ADDFILE              EQU X'80000007' Filecache cmd '80000007'x @PDC
PC#DELETEFILE           EQU X'80000008' Filecache cmd '80000008'x @PDC
PC#PURGECACHE           EQU X'8000000A' Filecache cmd '8000000A'x @PDC
PC#REFRESHCACHE         EQU X'80000009' Filecache cmd '80000009'x @PDC
PC#SHUTDOWNFILESYS      EQU X'8000000B' Soft Shutdown '8000000B'x @DVA
PC#PfsRecycle           EQU X'8000000C' Shutdown PFS for recycle
PC#PfsRestart           EQU X'8000000D' Restart the PFS
*
PC#SETIBMASYIO          EQU X'C0000006' SetIbm AsyncIO 'C0000006'x@PDC
PC#SETIBMOPTCMD         EQU X'C0000005' SetIBMOpt TCP  'C0000005'x@PDC
PC#ERRORTEXT            EQU X'C000000B' Get error text 'C000000B'x@DNA
*                                                     -1073741813 @DNA
PC#SYSNAMES             EQU X'C000000E' Get sysnames   'C000000C'x@PxA
PC#TDNAMES              EQU X'C000000F' GET CINET TDNAMES         @03A
PC#HFSSTATS             EQU X'C0000010' GET HFS Stats             @PKA
PC#BRLMSRVR             EQU X'C0000011' GET brlm server name      @PKA
PC#SFSDIAG              EQU X'80000012' Shared-FS Diagnose        @DTA
PC#USERSIGNAL           EQU X'C0000013' Set Lost Locks Signal(@05)@06A
PC#DIRGETHOST           EQU X'C0000014' Directed GetHost          @06A
PC#SHUTTINGDOWNFS       EQU X'C0000015' File system shutdown      @E7A
PC#RECYCLEDONE          EQU X'C0000016' recycle is complete       @EDA
*  PC#GetVnodeToken - Allocate Vde & insert an open
PC#GETVNODETOKEN        EQU X'80000017'                           @F2A
*         inet6_is_srcaddr() function                            @F3A
PC#ISSRCADDR            EQU X'C0000018' inet6_is_srcaddr          @F3A
*  PC#ErrorText Subfunctions   - Request type:
PC#ETDESC               EQU X'0000'     Get description text      @DNA
PC#ETACTION             EQU X'0001'     Get action text           @DNA
PC#ETMODNAME            EQU X'0002'     Get module name           @DNA
*  PC#ErrorText Error code type
PC#ETREASON             EQU X'0000'     Reason code input         @DNA
PC#ETERRNO              EQU X'0001'     Errno code input          @DNA
*  PC#ShuttingDownFS args
PC#TYPEFILESYS          EQU X'00000001' shutdown=filesys          @E7A
PC#TYPEFILEOWNER        EQU X'00000002' shutdown=fileowner        @E7A
PC#TYPEOMVS             EQU X'00000003' omvs shutdown             @E7A
*
```

```
* Constants for BPX1LOD  - HFS load - options                    @EFC
*
LOD_ERROR_ST_EXLINK     EQU X'80000000'  Error if sticky/ext lnk  @DRA
LOD_IGNORE_STICKY       EQU X'40000000'  Skip sticky check        @DRA
*
*
* Constants for BPX1LDX  - extended HFS load - options           @EFC
*                          and directed load returned parameters
*
LOD_DIRECTED            EQU X'20000000'  Directed loadhfs         @ECA
*
DIRECTEDLOADRETURNEDPARMS   DSECT ,                               @ECA
DIRECTEDLOADMODULELENGTH64  DS  FD Directed Load Mod Len AMODE64  @ECA
                            ORG DIRECTEDLOADMODULELENGTH64        @ECA
                            DS  F                                 @ECA
DIRECTEDLOADMODULELENGTH    DS  F  Directed Load Mod Len AMODE31  @ECA
*
DIRECTEDLOADMODULESTART64   DS  AD Directed Load Mod Start AMODE64@ECA
                            ORG DIRECTEDLOADMODULESTART64         @ECA
                            DS  F                                 @ECA
DIRECTEDLOADMODULESTART     DS  A  Directed Load Mod Start AMODE31@ECA
*
DIRECTEDLOADMODULEENTRYPT64 DS  AD Directed Load Mod Entry AMODE64@ECA
                            ORG DIRECTEDLOADMODULEENTRYPT64       @ECA
                            DS  F                                 @ECA
DIRECTEDLOADMODULEENTRYPT   DS  A  Directed Load Mod Entry AMODE31@ECA
*
* Constants for BPX1DSD                                          @DPA
*
SW_SIGDLV_ENABLE#       EQU 1                                     @DPA
SW_SIGDLV_DISABLE#      EQU 2                                     @DPA
*
*   Define equates for BPX1OSE syscall function_code parameter
OSENV_GET               EQU   1  get function                     @DQA
OSENV_SET               EQU   2  set function                     @DQA
OSENV_UNSET             EQU   4  unset function                   @DQA
OSENV_PERSIST           EQU   8  persist function                 @DQA
OSENV_UNPERSIST         EQU   16 unpersist function               @DQA
*   Define equates for BPX1OSE syscall Request_Flags parameter
OSENV_WLM               EQU   1  WLM Enclave membership           @DQA
OSENV_SECURITY          EQU   2  pthread security environment     @DQA
*
*   Define equates for BPX1PQG syscall RequestType parameter
THDQ_FREEZE             EQU   2  Freezes the threads identified in
*                                the THDQ Data List array (BPXYTHDQ)
*                                                                 @DUA
THDQ_UNFREEZE_ALL       EQU   8  Unfreezes all threads that are frozen
*                                in the caller process            @DUA
THDQ_GET_STATE          EQU   1  Retrieves the state data for the
*                                threads identified in the THDQ data
*                                list array or for all threads.
*                                This value can only be specified with
*                                THDQ_FREEZE                      @DUA
*
** BPXYCONS End
```

BPXYCW — Serialization constants used by many services

BPXYCW is composed only of EQUates. DSECT= is allowed but ignored.

```
           BPXYCW   ,
** BPXYCW:    Serialization Constants
CW_INTRPT       EQU   1      Thread interrupted by a signal
*                                              (x'0000 0001')
CW_CONDVAR      EQU   32     Thread notified that some condition
```

```
*                               has been met     (x'0000 0020')
CW_TIMEOUT       EQU   64     Timeout occurred   (x'0000 0040')
*
** BPXYCW   End
```

BPXYDCOR — dbx cordump cache information

BPXYDCOR contains the mapping of dump related information used by **dbx** when a dump is being formatted. AMODE 64 callers use "BPXYDCOR — dbx cordump cache information" on page 1089.

```
            BPXYDCOR   ,
*
* ********************************************************************
* *                                                                  *
* * Level information                                               *
* *                                                                  *
* ********************************************************************
*
*
DCOR_LEVEL1 EQU 65536    65536='00010000'x.
DCOR_LEVEL2 EQU 131072   131072='00020000'x.
*
* ********************************************************************
* *                                                                  *
* * Function codes for BPXGMCDE routine                             *
* *                                                                  *
* ********************************************************************
*
*
DCOR_OPEN# EQU 1
DCOR_CLOSE# EQU 2
DCOR_STATUS# EQU 3
*
*    ********************************************************************
*    *                                                                  *
*    * Open return codes                                               *
*    *                                                                  *
*    ********************************************************************
*
*
DCOR_CDERC_OK EQU 0      The specified function completed successfully
DCOR_CDERC_PARMERR EQU 4 A parmeter error was detected. See return     X
                         value 1 for more detail
DCOR_CDERC_PROCERR EQU 8 A DCORE processing error occurred. See return X
                         value 1 for more detail
DCOR_CDERC_IKJTSOEVERR EQU 12 An error was encountered trying to       X
                         establish a TSO environment with the IKJTSOEV X
                         service. See return values for more           X
                         information
DCOR_CDERC_IKJEFTSRERR EQU 16 An error was encountered trying to run   X
                         the REXX EXEC with the IKJEFTSR service. See  X
                         return values for more information
DCOR_CDERC_ALLOCATEERR EQU 20 An error was encountered trying to       X
                         allocate one of the user specified data sets.
DCOR_CDERC_IRXINITERR EQU 28 An error was encountered trying to        X
                         establish a REXX environment
*
*    ********************************************************************
*    *                                                                  *
*    * Status return codes                                            *
*    *                                                                  *
*    ********************************************************************
*
*
```

```
DCOR_CDERC_STATUS_OPENCOMPLETE EQU 0
DCOR_CDERC_STATUS_OPENCONTINUING EQU 1
DCOR_CDERC_STATUS_OPENTERMINATED EQU 2
DCOR_CDERC_STATUS_INVALIDTOKEN EQU 3
*
*    ****************************************************************
*    *                                                              *
*    * Status Rc values when Status return code is                  *
*    * Dcor_CDErc_Status_OpenContinuing                            *
*    *                                                              *
*    ****************************************************************
*
*
DCOR_STATUS_CONT_STARTTSOENV EQU 0 Starting the TSO environment
DCOR_STATUS_CONT_EXECSTARTED EQU 1 BPXTIPCS started
DCOR_STATUS_CONT_EXECCLIST EQU 2 BPXTIPCS allocating CLIST data set
DCOR_STATUS_CONT_DUMPDDIR EQU 3 BPXTIPCS allocating/creating dump      X
                        directory via BLSCDDIR
DCOR_STATUS_CONT_ALLOCDUMPDS EQU 4 BPXTIPCS allocating the dump data   X
                        set
DCOR_STATUS_CONT_INVOKEIPCS EQU 5 BPXTIPCS invoking IPCS
DCOR_STATUS_CONT_INVOKEVERBX EQU 6 BPXTIPC2 invoking VERBX routine
DCOR_STATUS_CONT_ANALYSISSTART EQU 7 Dump analysis started
DCOR_STATUS_CONT_ANALYSISPROCASIDS EQU 8 Analysis processing Asids
DCOR_STATUS_CONT_EXECEXITING EQU 9 BPXTIPCS exiting
DCOR_STATUS_CONT_RECALL EQU 10 BPXTIPCS recalling data set
*
*    ****************************************************************
*    *                                                              *
*    * R1 values when return code is Dcor_CDErc_ParmErr             *
*    *                                                              *
*    ****************************************************************
*
*
DCOR_R1_PARMERR_DUMPDSNREQ EQU 1 The name of a dump data set is        X
                        required
DCOR_R1_PARMERR_HFSDSNREQ EQU 2 The name of a dump data set in the HFS X
                        could not be found
*
*      ************************************************************
*      *                                                            *
*      * R1 values when return code is Dcor_CDErc_ProcErr           *
*      *                                                            *
*      ************************************************************
*
*
DCOR_R1_PROCERR_SYSTEMERRATC EQU 1 An unexpected system error has      X
                        occured while trying to establish the IPCS    X
                        environment. The R2 value contains an ABEND   X
                        reason code
*
*      ************************************************************
*      *                                                            *
*      * R1 values when return code is Dcor_CDErc_AllocateErr       *
*      *                                                            *
*      ************************************************************
*
*
DCOR_R1_ALLOCATEERR_LOGDSN EQU 1 Error allocating the log data set.    X
                        The R2 field is the return code from          X
                        allocation and the R3 field is the reason     X
                        code.
DCOR_R1_ALLOCATEERR_EXECDSN EQU 2 Error allocating the EXEC data set.  X
                        The R2 field is the return code from          X
                        allocation and the R3 field is the reason     X
                        code.
*
```

```
* ********************************************************************
* *                                                                  *
* * Function codes for BPXGMPTR Ptrace Dump Access Routine           *
* *                                                                  *
* ********************************************************************
*
*
DCOR_ASID_LIST# EQU 1
DCOR_SET_ASID# EQU 2
DCOR_PID_LIST# EQU 3
DCOR_SET_PID# EQU 4
DCOR_LDINFO# EQU 5
DCOR_THREAD_LIST# EQU 6
DCOR_THREAD_CURRENT# EQU 7
DCOR_SET_THREAD# EQU 8
DCOR_PSW# EQU  9
DCOR_GPR_LIST# EQU 10
DCOR_THREAD_STATUS# EQU 11
DCOR_READ_D# EQU 12
DCOR_ERROR_PSW# EQU 13
DCOR_CAPTURE# EQU 14
DCOR_ERROR_GPR_LIST# EQU 15
DCOR_FLT_LIST# EQU 16
DCOR_ERROR_FLT_LIST# EQU 17
DCOR_CONDINFO# EQU 18
DCOR_IPCSCMD# EQU 19
DCOR_PTRRC_OKVALUE EQU 0 The specified function completed successfully
DCOR_PTRRC_ASIDNOTFOUND EQU 1 The requested asid(s) not in dump
DCOR_PTRRC_ASIDNOTSET EQU 2 An ASID or PID has not been established    X
                        for this session
DCOR_PTRRC_REQTYPENOTDEFINED EQU 3 The function type provided on this  X
                        request is not supported by BPXGMPT2
DCOR_PTRRC_REQINVALIDTOKEN EQU 4 The open token provided on this       X
                        request is not not valid
DCOR_PTRRC_REQDCORTERMINATED EQU 5 Dcor dump access services are not   X
                        available
DCOR_PTRRC_THREADNOTFOUND EQU 6 The request thread(s) were not in the  X
                        dump
DCOR_PTRRC_THREADNOTSET EQU 7 The current thread has not been          X
                        established
DCOR_PTRRC_PIDNOTSET EQU 9 The request PID(s) were not in the dump
DCOR_PTRRC_PIDNOTFOUND EQU 10 The current process has not been         X
                        established
DCOR_PTRRC_STORAGENOTINDUMP EQU 11 The requested storage was not       X
                        dumped
DCOR_PTRRC_NASTANDALONEDUMP EQU 12 Not supported in a standalone dump
DCOR_PTRRC_ABENDOCCURRED EQU 13 Not supported in a standalone dump
DCOR_PTRRC_STORAGELENGTHBAD EQU 14 The requested storage length was    X
                        zero
DCOR_PTRRC_SOMESTORAGEINDUMP EQU 15 The number of bytes of storage     X
                        successfully retrieved is returned in the     X
                        reason code field
RSNOKVALUE EQU 0
RSNDCORERROR EQU 1       See Dcor return codes
RSNMVSERROR EQU 2        Ususally an out of storage condition or an    X
                        abend
RSNIPCSERROR EQU 3       When An IPCS error occurs use the DCOR log to X
                        view the messages generated by IPCS (normally X
                        suppress)
RSNCSVERROR EQU 4
RSNCSVMODI12ERR EQU 1
RSNCSVMODI3ERR EQU 2
RSNCSVTOOMANYEXTENTS EQU 3
*
* ********************************************************************
* *   parameter definitions for BPXGMPTR Ptrace Dump Access Routine  *
* *   1. Parm     1   function code                                  *
```

```
* *   2. Parms    2   Token returned from DCOR_OPEN#                *
* *   3. Parms    3-5 Function parameters                           *
* *   3. Parms    6-8 retvalue, retcode, rsncode                    *
* ********************************************************************
*
*
PARMS    DSECT
PARMS_FUNCTYPEPTR DS 1AL4
PARMS_DCOMTOKENPTR DS 1AL4
PARMS_INTERFACE DS 0CL0012
         ORG   PARMS_INTERFACE
PARMS_CAPTURE DS 0CL0012
PARMS_CAPTURE_PSTORADR DS 1AL4
PARMS_CAPTURE_PSTORLEN DS 1AL4
PARMS_CAPTURE_PDATAADR DS 1AL4 Address output buffer
         ORG   PARMS_INTERFACE
PARMS_READD DS 0CL0012
PARMS_READD_PSTORADR DS 1AL4
PARMS_READD_PSTORLEN DS 1AL4
PARMS_READD_PDATAADR DS 1AL4 user provided buffer
         ORG   PARMS_INTERFACE
PARMS_LDINFO DS 0CL0004
PARMS_LDINFO_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADLIST DS 0CL0008
PARMS_THREADLIST_OUTBUFPTR DS 1AL4
PARMS_THREADLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_PIDLIST DS 0CL0008
PARMS_PIDLIST_OUTBUFPTR DS 1AL4
PARMS_PIDLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ASIDLIST DS 0CL0008
PARMS_ASIDLIST_OUTBUFPTR DS 1AL4
PARMS_ASIDLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADCURRENT DS 0CL0004
PARMS_THREADCURRENT_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETASID DS 0CL0004
PARMS_SETASID_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETPID DS 0CL0004
PARMS_SETPID_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETTHREAD DS 0CL0004
PARMS_SETTHREAD_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_PSW DS   0CL0004
PARMS_PSW_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_PSW DS 0CL0004
PARMS_ERROR_PSW_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADSTATUS DS 0CL0008
PARMS_THREADSTATUS_OUTBUFPTR DS 1AL4
PARMS_THREADSTATUS_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_GPRLIST DS 0CL0008
PARMS_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_GPRLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_GPRLIST DS 0CL0008
PARMS_ERROR_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_GPRLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_FLTLIST DS 0CL0008
```

```
PARMS_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_FLTLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_FLTLIST DS 0CL0008
PARMS_ERROR_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_FLTLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_CONDITIONINFO DS 0CL0008
PARMS_CONDITIONINFO_OUTBUFPTR DS 1AL4
PARMS_CONDITIONINFO_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_IPCSCMD DS 0CL0012
PARMS_IPCSCMDTEXT_INBUFPTR DS 1AL4
PARMS_IPCSCMDTEXT_INBUFLEN DS 1AL4
PARMS_IPCSCMDPRNT_LRECL DS 1AL4
PARMS_XRVPTR DS 1AL4     Return Value
PARMS_XRCPTR DS 1AL4     Return Code
PARMS_XRSNPTR DS 1AL4    Reason Code
PARMS_LEN EQU  *-PARMS
PARMSG   DSECT
PARMS_FUNCTYPEPTRG DS 1AD
PARMS_DCOMTOKENPTRG DS 1AD
PARMS_INTERFACEG DS 0CL0024
         ORG   PARMS_INTERFACEG
PARMS_CAPTUREG DS 0CL0024
PARMS_CAPTURE_PSTORADRG DS 1AD
PARMS_CAPTURE_PSTORLENG DS 1AD
PARMS_CAPTURE_PDATAADRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_READDG DS 0CL0024
PARMS_READD_PSTORADRG DS 1AD
PARMS_READD_PSTORLENG DS 1AD
PARMS_READD_PDATAADRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_LDINFOG DS 0CL0008
PARMS_LDINFO_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_THREADLISTG DS 0CL0016
PARMS_THREADLIST_OUTBUFPTRG DS 1AD
PARMS_THREADLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_PIDLISTG DS 0CL0016
PARMS_PIDLIST_OUTBUFPTRG DS 1AD
PARMS_PIDLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ASIDLISTG DS 0CL0016
PARMS_ASIDLIST_OUTBUFPTRG DS 1AD
PARMS_ASIDLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_THREADCURRENTG DS 0CL0008
PARMS_THREADCURRENT_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETASIDG DS 0CL0008
PARMS_SETASID_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETPIDG DS 0CL0008
PARMS_SETPID_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETTHREADG DS 0CL0008
PARMS_SETTHREAD_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_PSWG DS  0CL0008
PARMS_PSW_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_PSWG DS 0CL0008
PARMS_ERROR_PSW_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
```

```
PARMS_THREADSTATUSG DS 0CL0016
PARMS_THREADSTATUS_OUTBUFPTRG DS 1AD
PARMS_THREADSTATUS_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_GPRLISTG DS 0CL0016
PARMS_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_GPRLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_GPRLISTG DS 0CL0016
PARMS_ERROR_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_GPRLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_FLTLISTG DS 0CL0016
PARMS_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_FLTLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_FLTLISTG DS 0CL0016
PARMS_ERROR_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_FLTLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_CONDITIONINFOG DS 0CL0016
PARMS_CONDITIONINFO_OUTBUFPTRG DS 1AD
PARMS_CONDITIONINFO_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_IPCSCMDG DS 0CL0024
PARMS_IPCSCMDTEXT_INBUFPTRG DS 1AD
PARMS_IPCSCMDTEXT_INBUFLENG DS 1AD
PARMS_IPCSCMDPRNT_LRECLG DS 1AD
PARMS_XRVPTRG DS 1AD
PARMS_XRCPTRG DS 1AD
PARMS_XRSNPTRG DS 1AD
PARMSG_LEN EQU *-PARMSG
ASIDLIST_MAP DSECT
ASID_NEXTOFF DS 1FL4     Offset to the next ASID in DcomAsidList
ASID_NUM DS    1FL2
ASID_CPU DS    1FL1      CPUID
ASID_FLAGS DS  0BL1      Status flags
ASID_HOME EQU  X'80'     Current HOMEAsid
ASID_PRIM EQU  X'40'     Current PRIMARY ASID
ASID_SEC EQU   X'20'     Current SECONDARY ASID
         ORG   ASID_FLAGS+X'00000001'
ASID_JOBNAME DS 1CL0009
         DS    1CL0003   Reserved
ASID_PIDCNT DS 1FL4      Number of Pids in this Asid
ASID_ASCB DS   1AL4      Pointer to ASCB
         DS    1CL0004   Reserved
ASID_PIDLISTPTR DS 1AL4  Pointer to the pidlist for This Asid
         DS    1CL0004   Reserved
ASID_MAPEND DS 0C        end of block
ASIDLIST_MAP_LEN EQU *-ASIDLIST_MAP
PIDLIST_MAP DSECT
PID_NEXTOFF DS 1FL4      Offset to the next Pid in DcomPidList
PID_     DS    1FL4      Process id
PID_ASID DS    1FL2      Asid of this Pid
PID_THIDCNT DS 1FL2      Count of thids in this pid
PID_FOCUSTHREAD DS 1CL0008 Ptrace focus thread
PID_ERRORTHREAD DS 1CL0008 Ptrace error thread
PID_LOGINNAME DS 1CL0009 Tso logon
         DS    1CL0003   Reserved
PID_THIDLISTPTR DS 1AL4  list info for each THID
         DS    1CL0004   Reserved
PID_PENDINGSIGMASK DS 1BL8 Signals pending at the process that could   X
                         not be delivered to any thread
PID_BLOCKEDSIGMASK DS 1BL8 Signals blocked on all thread
PID_MAPEND DS  0C        end of block
PIDLIST_MAP_LEN EQU *-PIDLIST_MAP
CONDINFO_MAP DSECT
```

```
COND_CURABENDINFO DS 0CL0016 If current task abended
COND_CURINTCODE DS 1FL2  Interrupt code
COND_CURSIGNUMBER DS 1FL2 Signal number raised
COND_CURABENDCODE DS 0BL4 Abend code
COND_CURABENDFLAGS DS 1BL1 System or user
COND_CURABENDCC DS 1BL3  Abend Number
COND_CURABENDREASON DS 1BL4 Abend Reason
COND_CURILC DS 1FL2      Instruction length
         DS    1CL0002   Reserved
CONDINFO_MAPEND DS 0C    end of block
CONDINFO_MAP_LEN EQU *-CONDINFO_MAP
```

BPXYDIRE — Map directory entries for readdir

DSECT=NO is not allowed; the basing for the PFSOTHER data is not known, as it depends on the length of the name.

```
           BPXYDIRE   ,
** BPXYDIRE: Mapping of directory entry
**  Used By: RDD
* LA     RegOne,buffer            RegOne->BPX1RDD buffer and 1st DIRE
* USING  DIRE,RegOne              Addressability to DIRE
DIRE                DSECT ,
DIRENTINFO          DS    0X     Fixed length information
DIRENTLEN           DS    H      Entry length
DIRENTNAML          DS    H      Name length
DIRENTNAME          DS    0C     Name
* LR     RegTwo,RegOne            RegTwo->DIRE
* LA     RegTwo,4(RegTwo)         RegTwo->start of name
* SLR    RegThree,RegThree        Clear register
* ICM    RegThree,3,DIRENTNAML    Load name length
* ALR    RegTwo,RegThree          RegTwo->end of name+1
* USING  DIRENTPFSDATA,RegTwo     Addressability to DIRENTPFSDATA
DIRENTPFSDATA       DSECT ,      Physical file system-specific data
DIRENTPFSINO        DS    CL4    File Serial Number = st_ino
DIRENTPFSOTHER      DS    0C     Other PFS specific data
                    ORG   DIRENTPFSDATA
DIRENTPLUSATTR      DS    0C     ReaddirPlus Attr
*
* ICM    RegThree,3,DIRENTLEN     Load entry length
* ALR    RegOne,RegThree          RegOne->Next DIRE in buffer
* BCT    Return_Value,Back_to_process_next_DIRE
** BPXYDIRE End
```

BPXYENFO — ENF signal constants

BPXYENFO is composed only of EQUates for listeners of kernel ENF signals. DSECT= is allowed but ignored.

```
           BPXYENFO   ,
** BPXYENFO: OMVS ENF constants
**  Used By: OMVS ENF Listeners and OMVS ENF Signallers
* OMVS ENF QUALifier values
BPXYENFOACT  EQU   X'80000000'    OMVS Active
** BPXYENFO End
```

BPXYERNO — Component return and reason codes

BPXYERNO is composed only of EQUates. DSECT= is allowed but ignored. Because the return codes and reason codes that are in this macro are in *z/OS UNIX System Services Messages and Codes*, the expansion of this macro is suppressed.

```
   BPXYERNO   LIST=NO
PUSH  PRINT  BPXYERNO: z/OS UNIX Component return/reason codes
PRINT OFF
POP   PRINT
```

BPXYFCTL — Command values and flags for fcntl

```
            BPXYFCTL   ,
** BPXYFCTL: File descriptor flags and command values
**  Used By: FCT
FCTL                DSECT ,
*                             External file descriptor flags
FCTLFDFL1           DS    B
FCTLRS01            EQU   X'80'  Reserved-DO NOT USE THIS BIT!
*                             FCTLFDFLAGS must never be < 0
FCTLFDFL2           DS    B      Reserved
FCTLFDFL3           DS    B      Reserved
FCTLFDFL4           DS    B
FCTLCLOFORK         EQU   X'02'  1= close_on_fork
FCTLCLOEXEC         EQU   X'01'  1= close_on_exec
*                             Command value definitions
F_CVT               DSECT ,      F_CONTROL_CVT section
FCVT_CMD            DS    F      Sub-Command
| SETCVTALL          EQU   4      Unicode Services enabled
| SETCVTAUTOALL      EQU   5      Unicode Services enabled if AUTOCVT(ALL)
SETCVTOFF           EQU   0      Set Off
SETCVTON            EQU   1      Set On
SETAUTOCVTON        EQU   2      Set On if AUTOCVT(YES)
QUERYCVT            EQU   3      Query current mode
FCVT_PCCSID         DS    H      Program CCSID
FCVT_FCCSID         DS    H      File CCSID
*                             External file descriptor flags
F_DUPFD             EQU   0      Duplicate file descriptor
F_GETFD             EQU   1      Get file descriptor flags
F_SETFD             EQU   2      Set file descriptor flags
F_GETFL             EQU   3      Get file status flags
F_SETFL             EQU   4      Set file status flags
F_GETLK             EQU   5      Get record locking information
F_SETLK             EQU   6      Set record locking information
F_SETLKW            EQU   7      Set record locking information -
*                                wait if blocked
F_DUPFD2            EQU   8      Duplicate file descriptor, option 2
F_CLOSFD            EQU   9      Close file descriptors
F_GETOWN            EQU   10     Get process id or process group
F_SETOWN            EQU   11     Set process id or process group
F_SETTAG            EQU   12     Set File Tag
F_CONTROL_CVT       EQU   13     Control conversion
FCTL#LENGTH         EQU   *-FCTL Length of this structure
** BPXYFCTL End
```

BPXYFDUM — Logical file system dump parameter list

DSECT=YES is required.

```
            BPXYFDUM   DSECT=YES
** BPXYFDUM: FDUM - LFS dump list passed to PFS initialization
FDUM                DSECT ,
FDUMBEGIN           DS    0D
*
FDUMPHDRINFO        DS    0F
FDUMPENTS           DS    F            NUMBER OF ENTRIES
FDUMPID             DC    C'FDUM'      EYE CATCHER
FDUMPHRES1          DS    CL8          SPACE RESERVED FOR EXPANSION
```

```
*
FDUM#LENH            EQU   *-FDUMBEGIN
*
FDUMPDATA            DSECT ,
                     DS    0F           ONE SET FOR EACH AREA TO DUMP
FDUMPSTOKEN          DS    CL8          STOKEN FOR DUMP
FDUMPRES1            DS    CL8          RESERVED
FDUMPSTART           DS    F            FIRST BYTE TO DUMP
FDUMPEND             DS    F            LAST BYTE TO DUMP
*
FDUM#LENENT          EQU   *-FDUMPDATA
*
* To access the FDUM header (dumpptr must be a copy of pfsi_dumpptr):
* L      RegOne,dumpptr             RegOne->pfsi_dumpents from BPXYPFSI
* USING  FDUM,RegOne                Addressability to FDUM
*
* To access the first FDUMPDATA:
* LR     RegTwo,RegOne              RegTwo->FDUM
* LA     RegTwo,FDUM#LENH(RegTwo)   RegTwo->FDUMPDATA
* USING  FDUMPDATA,RegTwo           Addressability to FDUMPDATA fields
*
* To access the next FDUMPDATA:
* LA     RegTwo,FDUM#LENENT(RegTwo) RegTwo-> next FDUMPDATA
*
** BPXYFDUM End
```

BPXYFTYP — File type definitions

BPXYFTYP is composed only of EQUates. DSECT= is allowed but ignored.

```
           BPXYFTYP   ,
** BPXYFTYP: File type definitions
**  Used By: FST MKD MKN OPN
FT_DIR               EQU   1      Directory File
FT_CHARSPEC          EQU   2      Character Special File
FT_REGFILE           EQU   3      Regular File
FT_FIFO              EQU   4      Named Pipe (FIFO) File
FT_SYMLINK           EQU   5      Symbolic link
*                    EQU   6      Reserved for Block Special
FT_SOCKET            EQU   7      Socket File
*
** File format definitions (for chattr)                      9
FTFFNA               EQU   0      Not specified              7
FTFFBINARY           EQU   1      Binary data
*                                 Text data delimiters:
FTFFNL               EQU   2        New Line
FTFFCR               EQU   3        Carrage Return
FTFFLF               EQU   4        Line Feed
FTFFCRLF             EQU   5        CR & LF
FTFFLFCR             EQU   6        LF & CR
FTFFCRNL             EQU   7        CR & NL
*                             Data consists of Records:
FTFFRECORD           EQU   8        Records
** BPXYFTYP End
```

BPXYFUIO — Map file system user I/O block

BPXYFUIO is used to map the user and file system I/O block.

```
           BPXYFUIO   ,
**  BPXYFUIO: User I/O block
**  Used By:  VRW VRD VRA
FUIO                  DSECT ,
FUIOBEGIN            DS    0D
FUIOHDR              DS    0D
```

```
*
FUIOID               DC    C'FUIO'                                    X
                           EBCDIC ID - FUIO
FUIOLEN              DC    AL4(FUIO#LENGTH)                           X
                           Length of this FUIO
FUIOINFO             DS    0D     Note: The following fields must     X
                                  map to BPXZDDPL
FUIOBUFFERADDR64     DS    0CL8   64 Bit Real Buffer address
FUIOBUFFERADDR64P    DS    0D     64 Bit Real Buffer address
FUIOBUFFERADDR       DS    F      Buffer address for READ or          X
                                  WRITE, etc.  Address of iov   X
                                  for READV and WRITEV
FUIOBUFFALET         DS    F      Alet associated with Buffer
FUIOCURSOR           DS    0F     Current position in the file
FUIOCUR1             DS    F      Word 1 of cursor
FUIOCUR2             DS    F      Word 2 of cursor
FUIOIBYTESRW         DS    F      Num of bytes to read or write       X
                                  (or iovcnt for READV and WRITEV)
FUIOASID             DS    H      Address Space ID
*
*
FUIOFLAGS            DS    XL1    Flags
*
FUIORWIND            EQU   FUIOFLAGS                                  X
                           Indicates if READ or WRITE          X
                           0 - Read, 1 - Write
FUIO#RD              EQU   X'7F'  Read: AND with FUIORWIND
FUIO#WRT             EQU   X'80'  Write: OR with FUIORWIND
*
FUIOPSWKEY           EQU   FUIOFLAGS                                  X
                           Describes bits 1 through 4 of       X
                           byte FUIOFLAGS
FUIOPSWKEYMASK       EQU   X'78'  AND with FUIOPSWKEY to clear        X
                           non-PSWKEY bits in FUIOFLAGS
*
FUIOSYNC             EQU   X'04'  Sync on write requested
FUIOSYNCDONE         EQU   X'02'  Sync on write was done
FUIOCHKACC           EQU   X'01'  Perform access checking
FUIOFLAG2            DS    XL1    More flags
FUIOREALPAGE         EQU   X'80'  Real page address provided
FUIOLIMITEX          EQU   X'40'  File size limit exceeded
FUIOIOVINUIO         EQU   X'20'  uio contains an iov struc
FUIOSHUTD            EQU   X'10'  Do shutdown after send
FUIOADDR64           EQU   X'08'  64 bit addressing
FUIOVSPECIFIC        DS    CL8    Vnop Specific Fields
FUIOFSSIZELIMIT      DS    0CL8   Rlimit support
FUIOFSSIZELIMITHW    DS    F      hiword - filesize limit
FUIONONEWFILES       EQU   X'80'  can't create new files
FUIOFSSIZELIMITLW    DS    F      loword - filesize limit
FUIOREL2SIZE         DS    0F     Fuio before Rel 3 expansion
*
FUIOINTERNAL         DS    0CL16
FUIOCURRBUFFPTR      DS    F      Buffer currently being processed
FUIOCURRBUFFLEN      DS    F      Length of current buffer
FUIOCURRBUFFOFFSET   DS    F      Offset into current buffer
FUIOCURRIOVENTRY     DS    F      Iov entry being processed
*
FUIOIOVRESIDUALCNT   DS    F      Num bytes remaining in iov str
FUIOTOTALBYTESRW     DS    F      Total number of bytes to be moved   X
                           If FuioIovinUio=on,                 X
                           this is the sum of all bytes        X
                           in the iov. Otherwise, this is      X
                           the same as FuioIBytesRW
FUIOBUFF64VADDR      DS    D      64 Bit Virtual Buffer address
FUIOEND              DS    0F     End of FUIO
*-------------------------------------------------------------
*   ReadDir Specific Information
```

```
*-------------------------------------------------------------
FUIOREADDIR          ORG   FUIOVSPECIFIC
FUIORDINDEX          DS    F      Readdir Index
FUIORDDFLAGS         DS    0XL4   Readdir flags
                     DS    XL3
FUIORDDFLAGS4        DS    XL1    Readdir flags:
FUIOCVERRET          EQU   X'02'   Cookie Verifier Returned
FUIORDDPLUS          EQU   X'01'   ReaddirPlus requested
*-------------------------------------------------------------
*   VN_ReadWriteV and VN_SRMsg Specific Information
*-------------------------------------------------------------
FUIOSOCKETALETS      ORG   FUIOVSPECIFIC
FUIOIOVALET          DS    F      SRMsg IOV Alet
FUIOIOVBUFALET       DS    F      All IOV's Buff's Alet
*
*   Readdir and ReaddirPlus Output Cookie Verifier
                     ORG   FUIOINTERNAL
FUIOCVER             DS    CL8    Cookie Verifier
                     ORG
*
*   Constants
*
FUIO#LEN             EQU   FUIOEND-FUIOBEGIN                          X
                                 Length of FUIO
FUIO#LENGTH          EQU   FUIO#LEN Length of FUIO
FUIO#REL2LEN         EQU   FUIOREL2SIZE-FUIOBEGIN                     X
                                 Length of Release 2 FUIO
FUIO#SP              EQU   3        Subpool for the FUIO
** BPXYFUIO End
```

BPXYGIDN — Map data returned for getpwnam and getpwuid

DSECT=NO is not allowed. The storage belongs to the service and a pointer is returned to the invoker.

```
            BPXYGIDN   ,
** BPXYGIDN: getpwnam, getpwuid amd getpwent return structure
**  Used By: GPN GPU GPE
GIDN                 DSECT ,        USING on GPN, GPU, GPE Return_value
GIDN_U_LEN           DS    F        Length of GIDN_U_NAME          1-8
GIDN_U_NAME          DS    0C       User name (trailing blanks)
*  Add GIDN_U_LEN to Index or base to access next field
                     DS    F        Length of user ID               4
GIDN_USERID          DS    F        User ID
                     DS    F        Length of group ID              4
GIDN_GROUPID         DS    F        Group ID
GIDN_D_LEN           DS    F        Length of GIDN_D_NAME       0-1023
GIDN_D_NAME          DS    0C       Initial working directory name
*  Add GIDN_D_LEN to Index or base to access next field
GIDN_P_LEN           DS    F        Length of GIDN_P_NAME       0-1023
GIDN_P_NAME          DS    0C       Initial user program name
GIDN#LENGTH          EQU   *-GIDN   Length less U_LEN, D_LEN and P_LEN
** BPXYGIDN End
```

BPXYGIDS — Map data returned for getgrnam and getgrpid

DSECT=NO is not allowed. The storage belongs to the service and a pointer is returned to the invoker.

```
            BPXYGIDS   ,
** BPXYGIDS: getgrnam, getgrgid and getgrent return structure
**  Used By: GGI GGN GGE
GIDS                 DSECT ,
GIDS_G_LEN           DS    F        Length of GIDS_G_NAME         1-8
```

```
GIDS_G_NAME          DS    0C       Group name (trailing blanks)
* Add GIDS_G_LEN to index or base to access following fields
                     DS    F        Length of group ID, always 4
GIDS_GROUPID         DS    F        Group ID
GIDS_COUNT           DS    F        Count of array elements
* Make a local copy of GIDS_COUNT
* Test: if local copy of GIDS_COUNT zero, quit
GIDS_M_LEN           DS    F        Length of GIDS_M_NAME         1-8
GIDS_M_NAME          DS    0C       Member name (trailing blanks)
* Add GIDS_M_LEN+4 to index or base
* Decrement local copy of GIDS_COUNT, goto test.
GIDS#LENGTH          EQU   *-GIDS   Length less all variable fields
** BPXYGIDS End
```

BPXYINHE — Spawn Inheritance Structure

AMODE 64 callers use "BPXYINHE — Spawn inheritance structure" on page 1095.

```
            BPXYINHE   ,
** BPXYINHE: Inheritance Area
**  Used By: spawn() callable service
INHE                 DSECT ,
INHEBEGIN            DS    0D
*
INHEEYE              DC    C'INHE'  Eye catcher
INHELENGTH           DC    AL2(INHE#LENGTH)                          X
                             Length of this structure
INHEVERSION          DC    AL2(INHE#VER)
INHE#VER             EQU   3     Version of this structure
INHEFLAGS            DS    0BL4  Flags indicating contents of structure
INHEFLAGS0           DS    XL1   1st byte
INHESETPGROUP        EQU   X'80' Set Process Group using INHEPGROUP
INHESETSIGMASK       EQU   X'40' Set Signal Mask using INHESIGMASK
INHESETSIGDEF        EQU   X'20' Set Signal Defaults using INHESIGDEF
INHESETTCPGRP        EQU   X'10' Set TTY Pgrp using INHECTLTTYFD
INHESETCWD           EQU   X'08' Set CWD using INHECWDPTR
INHESETUMASK         EQU   X'04' Set UMASK using INHEUMASK
INHESETUSERID        EQU   X'02' Set Userid using INHEUSERID
INHESETREGIONSZ      EQU   X'01' Set Region using INHEREGIONSZ
INHEFLAGS1           DS    XL1   2nd byte
INHESETTIMELIMIT     EQU   X'80' Set Timelimit with INHETIMELIMIT
INHESETACCTDATA      EQU   X'40' SET ACCTDATA using INHEACCTDATA
INHESETJOBNAME       EQU   X'20' SET JOBNAME using INHEJOBNAME
INHEMUSTBELOCAL      EQU   X'10' Spawn locally or else fail
INHESETDEBUGENV      EQU   X'08' Setup Debug Environment
INHESETMEMLIMIT      EQU   X'04' Set MemLimit with INHEMEMLIMIT
INHEFLAGS2           DS    XL1   3rd byte
INHEFLAGS3           DS    XL1   4th byte
INHEPGROUP           DS    F     Process Group for child
INHE#NEWPGROUP       EQU   0     Put child in a new proc grp of its own
INHESIGMASK          DS    BL8   Signal Mask for child
INHESIGDEF           DS    BL8   Set of default signals for child
INHECTLTTYFD         DS    F     Cntl TTY FD for tcsetgrp() in child
                     DS    0F    31-Bit Addressing Version
INHECWDPTR           DS    F     Pointer to the users CWD
INHECWDLEN           DS    H     Length of the users CWD
INHEACCTDATALEN      DS    H     LENGTH OF THE USERS ACCTDATA
INHEACCTDATAPTR      DS    F     POINTER TO THE USERS ACCTDATA
INHEUMASK            DS    XL4   Users Umask
INHEUSERID           DS    CL8   New A.S. user identity
INHEJOBNAME          DS    CL8   New A.S. jobname
INHEREGIONSZ         DS    F     New A.S. region size
INHETIMELIMIT        DS    F     New A.S. Time limit
```

```
                    DS    CL20  reserved
INHEMEMLIMIT        DS    D     New A.S. Memlimit #bytes
INHE#LENGTH         EQU   *-INHE
** BPXYINHE End
```

BPXYIOCC — Ioctl command definitions

BPXYIOCC is composed only of EQUates. DSECT= is allowed but ignored.

```
         BPXYIOCC   ,
** BPXYIOCC: Ioctl Command Constant Definitions
**  Used By: Ioctl syscalls
*   Ioctl command constants - Range 1-255 reserved for OpenMVS
*  Authorized/Tcpip CMD values
IOCC#TCI            EQU   5000 Cmd for Tcpip Initialization
IOCC#TCC            EQU   5001 Cmd for Complete Tcpip Initialization
IOCC#TCS            EQU   5002 Cmd for Tcpip Path Sever
IOCC#TCR            EQU   5003 Cmd for Tcpip Reply/Post call
IOCC#TCG            EQU   5004 Cmd for Tcpip SiGnal call         @p3a
IOCC#TCCE           EQU   5006 Cmd for Tcpip End Registration    @D5A
SIOCMSDELRT         EQU   5007 Cmd for Delete Route
*                                  (Pre-Router wrap)       @D5A
SIOCMSADDRT         EQU   5008 Cmd for Add Route
*                                  (Pre-Router wrap)       @D5A
SIOCMSSIFADDR       EQU   5009 Cmd for Set Interface address
*                                  (Pre-Router wrap)       @D5A
SIOCMSSIFFLAGS      EQU   5010 Cmd for Set Interface Flags
*                                  (Pre-Router wrap)       @D5A
SIOCMSSIFDSTADDR    EQU   5011 Cmd for Set point-to-point interface
*                                 address (Pre-Router wrap)       @D5A
SIOCMSSIFBRDADDR    EQU   5012 Cmd for Set Broadcast address
*                                  (Pre-Router wrap)       @D5A
SIOCMSSIFNETMASK    EQU   5013 Cmd for Set interface network
*                                 mask for an Internet address
*                                  (Pre-Router wrap)       @D5A
SIOCMSSIFMETRIC     EQU   5014 Cmd for Set Interface routing metric
*                                  (Pre-Router wrap)       @D5A
SIOCMSRBRTTABLE     EQU   5015 Cmd for Set Routing table required
*                                  required request
*                                  (Pre-Router wrap)       @D5A
SIOMSMETRIC1RT      EQU   5016 Cmd for Set metric1
*                                  (Pre-Router wrap)       @D5A
SIOCMSICMPREDIRECT  EQU   5017 Cmd for Propagating ICMP redirects
*                                  (Pre-Router wrap)       @D5A
SIOCSETTKN          EQU   X'8008139A' 5018 Set Tcp/Ip master Tkn @P6A
*
SIOCMSADDRT6        EQU   X'8044F604'     Add IPV6 Route         @DDA
*
SIOCMSDELRT6        EQU   X'8044F605'     Delete IPV6 Route      @DDA
*
SIOCGRT6TABLE       EQU   X'C014F606' Get IPV6 Network Routing
*                                    Table                   @DDA
SIOCGRT6TABLE64     EQU   X'C018F606' Get IPV6 Network Routing
*                                    Table  64-BIT           @PFC
SIOCMSRBRT6TABLE    EQU   X'8000F607' Rebuild IPV6 Route Tables  @DDA
*
SIOCGHOMEIF6        EQU   X'C014F608'  Get IPV6 Home Interface
*                                      Configuration          @PBC
SIOCGHOMEIF664      EQU   X'C018F608'  Get IPV6 Home Interface
*                                      Configuration 64-Bit   @PFC
SIOCMSRBHOMEIF6     EQU   X'8000F609' Rebuild IPV6 Home Interface@PBC
*
SIOCMSCHGRT6METRIC  EQU   X'8044F60A' Change IPV6 route's metric @PDA
*
SIOCMSMODHOMEIF6    EQU   X'8008F60B' Modify IPV6 Home Interface @DHA
*
```

```
SIOCMSADDRT6V2       EQU   X'8058F60C' Add IPV6 route version 2   @PGA
*
SIOCMSDELRT6V2       EQU   X'8058F60D' Del IPV6 route version 2   @PGA
*
SIOCMSCHGRT6METRICV2 EQU   X'8058F60E' Change IPV6 route's metric @PGA
*
* Connection type and security credentials on TCPIP sockets.      @DRA
*   Refer to: Comm Svr: IP Programmer's Guide and References.     @DRA
SIOCGPARTNERINFO     EQU   X'C000F612'  Get Info                  @DRA
SIOCSPARTNERINFO     EQU   X'8004F613'  Set Optimization         @DRA
*  Ioctl Command Constants - terminal control
TIOCGWINSZ           EQU   X'4008A368'  get window size           @D3A
TIOCSWINSZ           EQU   X'8008A367'  set window size           @D3A
TIOCNOTIFY           EQU   X'8001A364'  notify master by packet   @P7A
*  Constants for argument when TIOCNOTIFY is specified           @P7A
IOCC#PWBEGIN         EQU   1            Begin secure data         @P7A
IOCC#PWEND           EQU   2            End   secure data         @P7A
*
* Ioctl command constants - for Router query                     @D5A
SIOCGRTTABLE         EQU   X'C008C980'  Gets Network Routing Tab  @D5A
SIOCGRTTABLE64       EQU   X'C00CC980'  Get NRT for 64-Bit C Pgm  @PCA
*
SIOCSETRTTD          EQU   X'8008C981'  Set Socket to be attached to
*                                         1 TD                   @D5A
*
SIOCMSMODHOMEIF      EQU   X'8008C983'  Modify Home Interface     @DHA
*
SIOCMSMODHOMEIFV2    EQU   X'8016C984'  Modify Home Interface V2  @PIA
*
SIOCMSADDRTV2        EQU   X'8054C985'  Cmd for Add Route V2      @PIA
*
SIOCMSDELRTV2        EQU   X'8054C986'  Cmd for Delete Route V2   @PIA
*
SIOMSMETRIC1RTV2     EQU   X'8054C987'  Cmd for Set Metric1 V2    @PIA
*
FIONBIO              EQU   X'8004A77E'  set/reset nonblock I/O
FIONREAD             EQU   X'4004A77F'  get number of readable bytes
*                                       available
FIONWRITE            EQU   X'4004A78A'  get number of writeable bytes
*                                       available                @DIA
FIOASYNC             EQU   X'8004A77D'  set/clear async I/O       @D5A
FIOSETOWN            EQU   X'8004A77C'  set owner                 @D5A
FIOGETOWN            EQU   X'4004A77B'  get owner                 @D5A
SECIGET              EQU   X'4010E401'  get security information
SECIGET_T            EQU   X'4028E403'  Get peer task security    @DIA
SIOCTIEDESTHRD       EQU   X'8004E404'  Tie descriptor to thread  @DIA
SIOCSECENVR          EQU   X'C012A78B'  SET/GET Client Security   @DLA
*                                         Environment            @DLA
SIOCADDRT            EQU   X'8030A70A'  IBM use only, Add routing
*                                       table entry
SIOCATMARK           EQU   X'4004A707'  Is current location pointing
*                                       to out-of-band data?
SIOCSPGRP            EQU   X'8004A708'  Set process group         @DDA
SIOCGPGRP            EQU   X'4004A709'  Get process group         @DDA
SIOCDELRT            EQU   X'8030A70B'  IBM use only, Delete routing
*                                       table entry
SIOMETRIC1RT         EQU   X'8030A70C'  IBM use only, Set metric1 @D5A
SIOCSIFADDR          EQU   X'8020A70C'  Set Network interface addr@D5A
SIOCGIFADDR          EQU   X'C020A70D'  Get Network interface address
SIOCGIFBRDADDR       EQU   X'C020A712'  Get Network interface
*                                       Broadcast Address
SIOCSIFBRDADDR       EQU   X'8020A713'  Sets Network interface
*                                       Broadcast Address        @D5A
SIOCGIFCONF          EQU   X'C008A714'  Get Network interface Config
SIOCGIFCONF64        EQU   X'C00CA714'    for 64-Bit C Pgms
SIOCGIFCONF6         EQU   X'C018A722'  Get IPv6 Network IfConf  @DJA
*
```

```
SIOCGIFMTU           EQU   X'C020A726'  Get MTU Size             @DQA
*
SIOCGIFDSTADDR       EQU   X'C020A70F'  Get Network interface
*                                      Destination Address
SIOCGIFFLAGS         EQU   X'C020A711'  Get Network interface Flags
SIOCGIFMETRIC        EQU   X'C020A717'  IBM use only, Gets Network
*                                      Interface Routing Metric
SIOCGIFNETMASK       EQU   X'C020A715'  Get Network interface
*                                      Network Mask
SIOCSIFNETMASK       EQU   X'8020A716'  Set Network interface
*                                      Network Mask               @D5A
SIOCSIFDSTADDR       EQU   X'8020A70E'  IBM use only, Sets Network
*                                      Interface Destination Address
SIOCSIFFLAGS         EQU   X'8020A710'  IBM use only, Sets Network
*                                      Interface Flags
SIOCSIFMETRIC        EQU   X'8020A718'  IBM use only, Sets Network
*                                      Interface Routing Metric
SIOCSARP             EQU   X'8024A71E'  IBM use only, Sets ARP
*                                      Entry                      @D5A
SIOCGARP             EQU   X'C024A71F'  IBM use only, Gets ARP
*                                      Entry                      @D5A
SIOCDARP             EQU   X'8024A720'  IBM use only, Deletes ARP
*                                      Entry                      @D5A
SIOCSHIWAT           EQU   X'8004A700'  Set High Water Mark
*                                        (Not Supported)          @D5A
SIOCGHIWAT           EQU   X'4004A701'  Get High Water Mark
*                                        (Not Supported)          @D5A
SIOCSLOWAT           EQU   X'8004A702'  Set Low Water Mark
*                                        (Not Supported)          @D5A
SIOCGLOWAT           EQU   X'4004A703'  Get Low Water Mark
*                                        (Not Supported)          @D5A
FIOFCTLNBIO          EQU   X'0000E402'  change blocking/nonblocking
*               STREAMS
IOCC#ILINK           EQU   X'4004E21A'  I_LINK                   @D9A
*               DFS ACLs
IOCC#EDITACL         EQU   X'2000C100'  Edit ACL                 @P5A
*               RACF ACLs
SETFACL              EQU   X'0000D301'  SET FILE ACL             @DBA
IOCC#SETFACL         EQU   X'0000D301'  SET FILE ACL             @DCA
GETFACL              EQU   X'0000D302'  GET FILE ACL             @DBA
IOCC#GETFACL         EQU   X'0000D302'  GET FILE ACL             @DCA
*
*   Get Port of Entry for Multilevel Security                    @DDA
*   Get Port of Entry Attributes for a Socket Resource           @DDA
SIOCGSOCKPOEATTRS    EQU   X'4000D305'                           @DDA
*   Get Port of Entry Attributes for a non-Socket Resource       @DGA
SIOCGFDPOEATTRS      EQU   X'4000D306'                           @DDA
*   Get Multilevel Security info for a Socket Resource           @DKA
SIOCGSOCKMLSINFO     EQU   X'4000D307'                           @DKA
*  Constants for argument when FIONBIO is specified
IOCC#BLOCK           EQU   X'00000000'  Allow blocking to occur
IOCC#NONBLOCK        EQU   X'00000001'  Do not allow blocking to occur
*  Constants for argument when SIOCTIEDESTHRD is specified       @DIA
SIOC#TIESD           EQU   X'00000001'  Tie descriptor to thread  @DIA
SIOC#UNTIESD         EQU   X'00000000'  UnTie descriptor from thrd@DIA
*  Constants for argument when SIOCSECENVR is specified          @DLA
SIOC#SETENVR         EQU   X'00000001'  SET Security Environment  @DLA
SIOC#GETENVR         EQU   X'00000002'  GET Security Envrionment  @DLA
*
*********************************************************************
*                                                                   *
*                       I P v 6                            @DDA *
*                                                                   *
*********************************************************************
*        IPv6 Ioctls
SIOCGIFVERSION       EQU   X'4000F601'  Get Interface Ver   Out
SIOCGSRCIPADDR       EQU   X'C000F602'  Get Source Addr   InOut
```

```
SIOCGIFNAMEINDEX     EQU   X'4000F603'  Get If Name/Index   Out

*********************************************************************
*                                                                   *
* Get and Set ip_msfilter                                           *
*                                                                   *
********************************************************************* @DMA
* Get and Set ip_msfilter (IPv4 only)                               @DMA
SIOCGIPMSFILTER      EQU   X'C000A724'                              @DMA
SIOCSIPMSFILTER      EQU   X'8000A725'                              @DMA
*********************************************************************
*                                                                   *
* Get and Set group_filter                                          *
*                                                                   *
********************************************************************* @DMA
* Get and Set group_filter (IPv6 or IPv4)                           @DMA
SIOCGMSFILTER        EQU   X'C000F610'                              @DMA
SIOCSMSFILTER        EQU   X'8000F611'                              @DMA

*********************************************************************
* Packet mode or Extended Packet mode data record control data.   @D7C*
*                                                                   *
* Returned on master read when no control information is pending.     *
* In packet mode one byte is returned.  In extended packet mode, four *
* bytes are returned.  Data follows the control data.               *
*********************************************************************
TIOC_DATA            EQU   X'00'        Data packet                @D3A
*********************************************************************
* Packet mode control byte - returned on master read()            @D7C*
*                                                                   *
* A single control byte is returned in packet mode.  In extended      *
* packet mode, four bytes are returned, with the non-extended bits    *
* in the fourth byte.  The equates below can be used against the      *
* fourth byte (with TM, OI and NI) or against all four bytes (with    *
* OC, NC, etc.).                                                    *
*********************************************************************
TIOCPKT_FLUSHREAD    EQU   X'01'        Input was flushed          @D3A
TIOCPKT_FLUSHWRITE   EQU   X'02'        Output was flushed         @D3A
TIOCPKT_STOP         EQU   X'04'        Stop output                @D3A
TIOCPKT_START        EQU   X'08'        Start output               @D3A
TIOCPKT_NOSTOP       EQU   X'10'        STOP/START not standard    @D3A
TIOCPKT_DOSTOP       EQU   X'20'        STOP/START standard        @D3A
*********************************************************************
* Extended Packet mode control byte - returned on master read()   @D7C*
*********************************************************************
TIOCXPKT_PASSTHRU    EQU   X'00000100'  3270 Passthrough mode     @D7C
TIOCXPKT_NOPASSTHRU  EQU   X'00000200'  Not 3270 Passthrough mode @D7C
TIOCXPKT_ECHO        EQU   X'00000400'  ECHO set on               @D7A
TIOCXPKT_NOECHO      EQU   X'00000800'  ECHO set off              @D7A
TIOCXPKT_CHCP        EQU   X'00001000'  Code page change          @D7A
TIOCXPKT_PWBEGIN     EQU   X'00002000'  Begin secure data         @P7A
TIOCXPKT_PWEND       EQU   X'00004000'  End   secure data         @P7A
*********************************************************************
* Get Pathname                                                      @DOA
IOCC#GETPATHNAME     EQU   17           Absolute name              @DOA
IOCC#GETPATHNAMEREL  EQU   19           Relative name              @DOA
*********************************************************************
* UPDTOFTE                                                          @D8C
*********************************************************************
IOCC#UPDTOFTE        EQU   20         UPDATE OFTE CMD              @D8A
*                                                                  @DDA
         AIF   ('&DSECT' EQ 'NO').B411                         @DDA
IOCUOFTE             DSECT ,          ARGUMENT BUFFER             @DDA
         AGO   .C411                                           @DDA
.B411    ANOP  ,                                               @DDA
IOCUOFTE             DS    0F         ARGUMENT BUFFER             @DDA
.C411    ANOP  ,                                               @DDA
```

```
*                                                                       @DDA
IOCUOCMD            DS     F          SUBCMD                            @D8A
IOCUO#READ          EQU    1            READ                            @D8A
IOCUO#WRITE         EQU    2            WRITE                           @D8A
IOCUO#CS            EQU    3            COMPARE & SWAP          @D8A
IOCUOVALUEBUFF      DS     0F         VALUE TO/FROM STATE AREA     @D8A
IOCUOVOFFSET        DS     F             OFFSET  (>=0)           @D8A
IOCUOVLEN           DS     F             LENGTH  (>0)            @D8A
IOCUOVDATA          DS     0C            DATA                            @D8A
*                                                                       @DDA
         AIF   ('&DSECT' EQ 'NO').B412                          @DDA
IOCUOCSBUFF         DSECT ,           COMPARE VALUE FOR CS SUBCMD @DDA
         AGO   .C412                                                    @DDA
.B412    ANOP  ,                                                        @DDA
IOCUOCSBUFF         DS    0F          COMPARE VALUE FOR CS SUBCMD @DDA
.C412    ANOP  ,                                                        @DDA
*                                                                       @DDA
IOCUOCSOFFSET       DS     CL4           OFFSET (BYTE BDY)         @D8A
IOCUOCSLEN          DS     CL4           LENGTH (BYTE BDY)         @D8A
IOCUOCSDATA         DS     0C            DATA                            @D8A
*
IOCC#REGFILEINT     EQU   21          REGISTER FILE INTR              @DAA
IOCC#FASTPATH       EQU   22          Set FastPath Ops                @P9A
*
IOCC#DEVCONSOLE     EQU   23          /dev/console behavior           @DEA
IOCC#DEVCONSUPPRESS EQU   1           /dev/console - set suppress     @DEA
IOCC#DEVCONUNSUPPRS EQU   0           /dev/console -   unsuppress    @DEA
*
IOCC#DEVFD          EQU   27          /dev/fd behavior                @DFA
*                      LFS/Cinet Level Ioctls                         @DDA
IOCC#GETSTACKS      EQU   24          Get Stack Names                 @DDA
IOCC#DIRIOCTL       EQU   25          Directed Ioctl                  @DDA
IOCC#GRTRSELECT     EQU   26          Get PreRtr Select               @DDA
*
*********************************************************************
*                                                                     @DDA*
*   Iocc#GetStacks -                                                    *
*      Get the names of the stacks that are attached to a socket.       *
*                                                                       *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B413                          @DDA
IOCSTACKINFO        DSECT ,                                         @DDA
         AGO   .C413                                                    @DDA
.B413    ANOP  ,                                                        @DDA
IOCSTACKINFO        DS    0F                                        @DDA
.C413    ANOP  ,                                                        @DDA
IOCSTACKINFOHEADER  DS    CL8
                    ORG   IOCSTACKINFOHEADER
IOCSTACKINFOFLAGS   DS    X             Flags
IOCSTACKCINET       EQU   X'80'         Cinet socket
                    DS    CL3
IOCSTACKENTRIES     DS    F             Number of Names returned
                    ORG
IOCSTACKNAMES       DS    CL16          Array of stack names
*********************************************************************
*   Array of IOCSTACKNAMES                                            *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B414                          @DDA
IOCSTACKNAMESD      DSECT ,                                         @DDA
         AGO   .C414                                                    @DDA
.B414    ANOP  ,                                                        @DDA
IOCSTACKNAMESD      DS    0F                                        @DDA
.C414    ANOP  ,                                                        @DDA
IOCSTACKNAME        DS    CL8           Stack name
IOCSTACKTDINDEX     DS    X             Cinet Stack TdIndex
```

```
IOCSTACKFLAGS        DS    X             Flags
IOCSTACK_ACTIVE          EQU  X'80'      Active
IOCSTACK_IPV6_SUPPORT    EQU  X'40'      IPv6 is supported
IOCSTACK_IPV6_INTERFACES EQU  X'20'      IPv6 Home Interfaces
IOCSTACK_IPV4_INTERFACES EQU  X'10'      IPv4 Home Interfaces
                     DS    CL6

*********************************************************************
*                                                                   @DDA*
*   Iocc#DirIoctl  - Directed Ioctl                                     *
*      Passes the imbedded ioctl to the specified stack.                *
*                                                                       *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B415                          @DDA
IOCDIRIOCTL          DSECT ,                                         @DDA
         AGO   .C415                                             @DDA
.B415    ANOP  ,                                                 @DDA
IOCDIRIOCTL          DS    0F                                        @DDA
.C415    ANOP  ,                                                 @DDA
IOCDIRHDR            DS    CL16
                     ORG   IOCDIRHDR
IOCDIRNAME           DS    CL8    Target Stack Name
IOCDIRCMD            DS    XL4    Imbedded ioctl Command
IOCDIRARGLEN         DS    F      Imbedded ioctl Argument Length
                     ORG
IOCDIRARG            DS    C      Imbedded ioctl Argument

*********************************************************************
*                                                                   @DDA*
*   Iocc#GRtrSelect - Get Cinet PreRouter's selected stack for each   *
*          of an array of specified destination IP addresses.         *
*                                                                       *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B416                          @DDA
IOCRTRSELECT         DSECT ,                                         @DDA
         AGO   .C416                                             @DDA
.B416    ANOP  ,                                                 @DDA
IOCRTRSELECT         DS    0F                                        @DDA
.C416    ANOP  ,                                                 @DDA
IOCRTRIPADDR         DS    CL16     Input IP Address
IOCRTRSTACK          DS    CL8      Output Selected Stack Name
                     ORG   IOCRTRSTACK
IOCRTRERRTEST        DS    CL1      Error if = 0
                     DS    CL1
IOCRTRERRNO          DS    XL2      Error RC (Errno)
IOCRTRRSN            DS    XL4      Error Rsn (ErrnoJr)
                     ORG

IOCRTRERROR          EQU   X'00'  IocRtrErrTest value to test for error

*********************************************************************
*                                                                   @DDA*
*   SiocGIfNameIndex - Get Interface Name/Index Table                   *
*                                                                       *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B418                          @DDA
IF_NAMEINDEXENTRY    DSECT ,                                         @DDA
         AGO   .C418                                             @DDA
.B418    ANOP  ,                                                 @DDA
IF_NAMEINDEXENTRY    DS    0F                                        @DDA
.C418    ANOP  ,                                                 @DDA
IF_NIINDEX           DS    F         Interface Index
                     ORG   IF_NIINDEX
IF_NITDINDEX         DS    H         CInet Td Index
```

```
IF_NIIFINDEX         DS    H         Stack Interface Index
                     ORG
IF_NINAME            DS    CL16      Interface Name, blank padded
IF_NIEXT             DS    CL4
                     ORG   IF_NIEXT
IF_NINAMETERM        DS    CL1       Null for C for Name len=16
IF_NIFLAGS           DS    X         Name Index Flags             @PKA
IF_NIOSM             EQU   X'80'     1 = OSM Interface            @PKA
                     DS    CL2       Reserved                     @PKC
                     ORG
IF_NAMEINDEXENTRYL   EQU   *-IF_NAMEINDEXENTRY                    @DDA

         AIF   ('&DSECT' EQ 'NO').B417                          @DDA
IF_NAMEINDEX         DSECT ,                                      @DDA
         AGO   .C417                                            @DDA
.B417    ANOP  ,                                                @DDA
IF_NAMEINDEX         DS    0F                                     @DDA
.C417    ANOP  ,                                                @DDA
IF_NIHEADER          DS    2F
                     ORG   IF_NIHEADER
IF_NITOTALIF         DS    F      Total Active Interfaces on System
IF_NIENTRIES         DS    F      Number of entries returned
                     ORG
IF_NITABLE           DS    CL(IF_NAMEINDEXENTRYL)

*************************************************************************
*                                                                  @DDA *
*   SiocGSockPoeAttrs - Socket Port of Entry Attributes              *
*                                                                    *
*************************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B419                          @DDA
IOCPOEATTR           DSECT ,                                      @DDA
         AGO   .C419                                            @DDA
.B419    ANOP  ,                                                @DDA
IOCPOEATTR           DS    0F                                     @DDA
.C419    ANOP  ,                                                @DDA
IOCPOEPEERIPADDR     DS    CL16          Peer IP Address
                     ORG   IOCPOEPEERIPADDR
IOCPOEPEERIPV6PREFIX DS    CL12
IOCPOEPEERIPV4ADDR   DS    F
                     ORG
IOCPOETERMID         DS    CL8           TERMINAL Profile Name
IOCPOELABEL          DS    CL8           Security Label
IOCPOEPROFILE        DS    CL64          SERVAUTH Resource Name

*************************************************************************
*                                                                  @DKA *
*   SiocGSockMLSINFO  - Socket Multilevel Security Information       *
*     IocPoeProfile returns full resource name                       *
*     IocMlsProfile returns actual profile name in use               *
*                                                                    *
*************************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B420                          @DKA
IOCMLSINFO           DSECT ,                                      @DKA
         AGO   .C420                                            @DKA
.B420    ANOP  ,                                                @DKA
IOCMLSINFO           DS    0F                                     @DKA
.C420    ANOP  ,                                                @DKA
IOCMLSMVSNAME        DS    CL8           MVS System Name          @DKA
IOCMLSSTKNAME        DS    CL8           Stack Job Name           @DKA
IOCMLSNAZNAME        DS    CL8           NetAccess Zone Name      @DKA
IOCMLSUSRNAME        DS    CL8           Caller UserID            @DKA
IOCMLSUSRLBL         DS    CL8           Caller Security Label    @DKA
IOCMLSSTKLBL         DS    CL8           Stack  Security Label    @DKA
IOCMLSNAZLBL         DS    CL8           Zone   Security Label    @DKA
```

```
IOCMLSCONLBL         DS    CL8           Connct Security Label    @DKA
IOCMLSPROFILE        DS    CL64          SERVAUTH Profile Name    @DKA

*--------------------------------------------------------------------*
*     Multicast Source Filter Structures from RFC 3678               *
*                                                                    *
*   *** These require the inclusion of BPXYSOCK ***                  *
*-------------------------------------------------------------------- @DMA
*                                                                      @DMA
         AIF   ('&MCAST' EQ 'NO').NOMCAST                          @DNA
*                                                                      @DMA
* SiocGIPMSFilter  - Get a list of multicast source addresses        @DMA
* SiocSIPMSFilter  - Set a list of multicast source addresses        @DMA
*                                                                      @DMA
         AIF   ('&DSECT' EQ 'NO').B421                             @DMA
IP_MSFILTER DSECT ,                                                  @DMA
         AGO   .C421                                                 @DMA
.B421    ANOP  ,                                                     @DMA
IP_MSFILTER DS 0F                                                    @DMA
.C421    ANOP  ,                                                     @DMA
IMSF_HEADER DS 0C                Header                              @DMA
IMSF_MULTIADDR DS CL4            IP Multicast address of group       @DMA
IMSF_INTERFACE DS CL4            Local IP addr of interface          @DMA
IMSF_FMODE DS CL4                Filter mode                         @DMA
IMSF_NUMSRC DS CL4               Number of sources in src_list       @DMA
IMSF_HEADER_LEN EQU *-IP_MSFILTER                                    @DMA
IMSF_SLIST DS 0CL(L'IMSF_SRCADDR) Start of source list               @DMA
*                                                                    @DMA
         AIF   ('&DSECT' EQ 'NO').B422                             @DMA
IMSF_SRCENTRY DSECT ,                                                @DMA
         AGO   .C422                                                 @DMA
.B422    ANOP  ,                                                     @DMA
IMSF_SRCENTRY DS 0F              Source list entry                   @DMA
.C422    ANOP  ,                                                     @DMA
IMSF_SRCADDR DS CL4              Source IP address                   @DMA
IMSF_SRCENTRY_LEN EQU *-IMSF_SRCENTRY Length                         @PHA
*                                                                    @DMA
*                                                                    @DMA
* SiocGMSFilter  - Get a list of multicast source addresses         @DMA
* SiocSMSFilter  - Set a list of multicast source addresses         @DMA
*                                                                    @DMA
**********************************************************************
*                                                                      *
* GROUP_REQ STRUCTURE                                                  *
*                                                                      *
********************************************************************** @DMA
         AIF   ('&DSECT' EQ 'NO').B423                             @DMA
GROUP_FILTER DSECT ,                                                 @DMA
         AGO   .C423                                                 @DMA
.B423    ANOP  ,                                                     @DMA
GROUP_FILTER DS 0F                                                   @DMA
.C423    ANOP  ,                                                     @DMA
GF_HEADER DS 0C                  Header                              @DMA
GF_INTERFACE DS CL4              Interface index                     @DMA
         DS    CL4               Padding                             @DMA
GF_GROUP DS    CL(L'SOCKADDR_STORAGE) Group address                  @DMA
GF_FMODE DS    CL4               Filter mode                         @PHM
GF_NUMSRC DS   CL4               Number of sources                   @PHM
GF_HEADER_LEN EQU *-GROUP_FILTER                                     @PHM
GF_SLIST DS 0CL(L'GF_SRCENTRY)   Start of source list                @DMA
         ORG   GF_GROUP                                             @DMA
GF_MULTISOCKADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN)                       @DMA
         ORG   GF_GROUP                                             @DMA
GF_MULTISOCKADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN)                      @DMA
*                                                                    @DMA
         AIF   ('&DSECT' EQ 'NO').B424                             @DMA
GF_SRCENTRY DSECT ,                                                  @DMA
```

```
         AGO   .C424                                                  @DMA
.B424    ANOP  ,                                                      @DMA
GF_SRCENTRY DS 0F                Source list entry                    @DMA
.C424    ANOP  ,                                                      @DMA
GF_SRCADDR DS CL(L'SOCKADDR_STORAGE) Source address                   @DMA
GF_SRCENTRY_LEN EQU *-GF_SRCENTRY  Length                             @PHA
         ORG   GF_SRCADDR                                            @DMA
GF_SRCADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN)                              @DMA
         ORG   GF_SRCADDR                                            @DMA
GF_SRCADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN)                             @DMA
*                                                                     @DMA
MCAST_INCLUDE      EQU   0                                            @DNA
MCAST_EXCLUDE      EQU   1                                            @DNA
MCAST_NUMSRC_MAX   EQU  64            Max number of sources for     @PHA
*                                      GF_NUMSRC and IMSF_NUMSRC     @DPA
*                                                                    @DNA
.NOMCAST ANOP  ,              End of Multicast Structures            @DNA
*********************************************************************
* SIOCGRTTABLE - Obtain route information. Returns information for   *
*                IPv4 routes from the TCP/IP stack's main route table.*
*                                                                    *
*                The Route entry structures returned can either be   *
*                Version 1, Version 2, or Version 3 structures.      *
*                  Version 1                                         *
*                    - Field IOCN_IPADDRRTMSGHOMEIF contains an IP   *
*                      address                                       *
*                  Version 2                                         *
*                    - IOCN_RTMSGTYPE structure is the same size     *
*                      as Version 1                                  *
*                    - Field IOCN_IPADDRRTMSGHOMEIF contains an      *
*                      interface index.                              *
*                  Version 3                                         *
*                    - IOCN_RTMSGTYPE structure is larger than the   *
*                      Version 1 or 2 size                           *
*                    - Field IOCN_IPADDRRTMSGHOMEIF contains an      *
*                      interface index                               *
*                    - New MTU field added                           *
*                The RTEV3 macro variable controls whether the       *
*                Version 3 Route entry structure is generated by     *
*                this macro.  By default, the macro generates the    *
*                Version 3 Route entry structure.                    *
*                                                                    *
*                When requesting Version 2 or Version 3 output in a  *
*                CINET environment, invoking applications must either *
*                have stack affinity or use the IOCC#DIRIOCTL to     *
*                invoke this ioctl.  Otherwise, the output from the  *
*                first stack will be in the requested version format *
*                but, the output from subsequent stacks will be in   *
*                Version 1 format.                                   *
*                                                                    *
* Input     - Input to the ioctl is a buffer length and a buffer     *
*             address.  By default, Version 1 Route entry structures *
*             are returned.                                          *
*                                                                    *
*             To obtain Version 2 or 3 Route entry output, an        *
*             IOCN_RTMSGHDRTYPE structure must be setup at the       *
*             beginning of the output buffer, before invoking the    *
*             ioctl.  The following fields must be set:              *
*               - IOCN_RTMSGHDRVER set to Version 2 or 3             *
*               - IOCN_RTMSGHDREYEID set to the correct eyecatcher   *
*                 value.                                             *
*                                                                    *
*                                                                    *
* Output    - Return_value = 0                                       *
*                 - IOCN_RTMSGHDRTYPE structure returned with        *
*                   field IOCN_RTMSGHDRNUMENT set to the number      *
*                   of Route entry structures returned.  In the      *
```

```
*                   output buffer, the header is followed by the     *
*                   Route entry structures.                          *
*                 - The IOCN_RTMSGHDRVER and IOCN_RTMSGHDRSUBVER     *
*                   fields indicate whether Version 1, 2, or 3       *
*                   of the Route entry structures was returned.      *
*                 - IOCN_RTMSGHDRNUMENT - number of either Version   *
*                   1, 2, or 3 Route entry structures returned.      *
*           - Return_value = -1                                      *
*                 - Return_code = EBADMSG                          *
*                   This means that the output buffer is not large   *
*                   enough for one IOCN_RTMSGHDRTYPE structure       *
*                   and one Route entry structure.                   *
*                 - Return_code = ERANGE                           *
*                   This means that the output buffer is not large   *
*                   enough for all the Route entry structures which  *
*                   can be returned. In this case, only the          *
*                   IOCN_RTMSGHDRTYPE structure will be returned.    *
*                   Field IOCN_RTMSGHDRNUMENT in this header will    *
*                   contain the number of Route entry structures     *
*                   which can be returned.  Field IOCN_RTMSGHDRVER   *
*                   will indicate which version of the Route entry   *
*                   structures will be returned.                     *
*                                                                @PIA *
*********************************************************************
*                                                                @PIA
* Header structure - IOCN_RTMSGHDRTYPE                           @PIA
*                                                                @PIA
         AIF   ('&DSECT' EQ 'NO').B425                       @PIA
IOCN_RTMSGHDRTYPE    DSECT ,      Route header                 @PIA
         AGO   .C425                                         @PIA
.B425    ANOP  ,                                             @PIA
IOCN_RTMSGHDRTYPE    DS    0F     Route header                 @PIA
.C425    ANOP  ,                                             @PIA
*
IOCN_RTMSGHDRLEN     DS     H     Length of header             @PIA
IOCN_RTMSGHDRVER     DS     FL1   Version of Route entry structures
*                                 returned                     @PIA
                     DS     CL1   Reserved                     @PIA
IOCN_RTMSGHDREYEID   DS     CL4   Eye catcher 'Rmsg'           @PIA
IOCN_RTMSGHDREYEC    EQU    C'Rmsg'  Eye catcher constant       @PIA
IOCN_RTMSGHDRSUBVER  DS     FL1   Version of Route entry structures
*                                 returned                     @PIA
                     DS     CL3   Reserved                     @PIA
IOCN_RTMSGHDRIOCTLNUM  DS   F     Set to x'10' to indicate the
*                                 SIOCGRTTABLE ioctl           @PIA
IOCN_RTMSGHDRNUMENT  DS     F     Output - number of Route entries
*                                 returned                     @PIA
                     DS     CL52  Reserved                     @PIA
IOCN_RTMSGHDR#LEN    EQU    *-IOCN_RTMSGHDRTYPE  Length of header@PIA
*
* Version constants for fields IOCN_RTMSGHDRVER  and
* IOCN_RTMSGHDRSUBVER                                           @PIA
*
IOCN_RTMSGTYPEV1     EQU    1       Version 1                   @PIA
IOCN_RTMSGTYPEV2     EQU    2       Version 2                   @PIA
IOCN_RTMSGTYPEV3     EQU    3       Version 3                   @PIA
*                                                               @PIA
* Version 1 and Version 2 Route entry structure - IOCN_RTMSGTYPE @PIA
*                                                               @PIA
         AIF   ('&DSECT' EQ 'NO').B426                       @PIA
IOCN_RTMSGTYPE       DSECT ,      Route entry structure        @PIA
         AGO   .C426                                         @PIA
.B426    ANOP  ,                                             @PIA
IOCN_RTMSGTYPE       DS    0F     Route entry structure        @PIA
.C426    ANOP  ,                                             @PIA
*
IOCN_IPADDRRTMSGDEST DS     CL4   Destination IP address      @PIA
```

```
IOCN_IPADDRRTMSGMASK DS     CL4   Subnet mask                    @PIA
IOCN_IPADDRRTMSGGATE DS     CL4   Gateway IP address             @PIA
IOCN_IPADDRRTMSGHOMEIF  DS  F     Version 1 = IP address         @PIA
*                                 Other versions = interface
*                                 index                          @PIA
IOCN_RTMSGMETRICTYPE DS     F     Metric type is always 1, meaning
*                                 metric value is in hop counts  @PIA
IOCN_RTMSGMETRIC     DS     F     Metric value in hop counts     @PIA
                     DS     CL3   Reserved                       @PIA
IOCN_RTATTRRTMSG     DS     XL1   Route flags                    @PIA
IOCN_BRTATTRLOOPBACK EQU    X'80'   1 = Loopback interface       @PIA
IOCN_BRTATTRLOCAL    EQU    X'40'   1 = Local/Home IP address    @PIA
IOCN_BRTATTRDYNBUILT EQU    X'10'   1 = Dynamically built, e.g.
*                                       by ICMP redirect       @PIA
IOCN_BRTATTRHOST     EQU    X'04'   1 = Host route, 0 = Network
*                                   route                      @PIA
IOCN_BRTATTRGATEWAY  EQU    X'02'   1 = Gateway                @PIA
IOCN_BRTATTRRTUP     EQU    X'01'   1 = Route is active, 0 = Route
*                                   is inactive                @PIA
IOCN_RTMSGRTETYPE    DS     XL1   Route type                     @PIA
IOCN_RTOTHER         EQU    1       Other (default, direct)      @PIA
IOCN_RTLOCAL         EQU    2       Static (configured)          @PIA
IOCN_RTICMP          EQU    4       ICMP                         @PIA
IOCN_RTRIP           EQU    8       RIP                          @PIA
IOCN_RTOSPF          EQU    13      OSPF                         @PIA
IOCN_RTREPSTAT       EQU    130     Replaceable static           @PIA
                     DS     CL3   Reserved                       @PIA
IOCN_RTMSGV1#LEN     EQU    *-IOCN_RTMSGTYPE V1 Route entry len  @PIA
IOCN_RTMSGV2#LEN     EQU    *-IOCN_RTMSGTYPE V2 Route entry len  @PIA
        AIF   ('&RTEV3' EQ 'NO').B427                          @PIA
*                                                              @PIA
* Version 3 Route entry structure - additional field           @PIA
*                                                              @PIA
IOCN_RTMSGMTU        DS     H     Route's MTU value            @PIA
                     DS     H     Reserved                     @PIA
                     DS     F     Reserved                     @PIA
                     DS     F     Reserved                     @PIA
                     DS     F     Reserved                     @PIA
IOCN_RTMSGV3#LEN     EQU    *-IOCN_RTMSGTYPE  V3 Route entry len @PIA
.B427    ANOP  ,                                               @PIA
*
********************************************************************
* IOCN_IFREQ                                                  @DQA*
*  Mapping that defines the network interface block that is used    *
*  on ioctls that transfer network interface information            *
*                                                                   *
*  This is equivalent to the C ifreq structure from in.h            *
*  This is the ASM version of the PL/X IOCN_IfType from BPXZIOCN.   *
*                                                                   *
********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B428
IOCN_IFREQ DSECT         Mapping for network interface information
         AGO   .C428
.B428    ANOP  ,
IOCN_IFREQ DS 0F         Mapping for network interface information
.C428    ANOP  ,
IOCN_IFTYPE   DS 0F      Structure Name from BPXZIOCN.
IOCN_IFNAME   DS CL16     Interface name.
IOCN_IFUNION  DS 0CL16    Union of fields:
         ORG   IOCN_IFUNION
IOCN_SADDRIF     DS CL16  The address of the interface
         ORG   IOCN_IFUNION
IOCN_SADDRIFDEST DS CL16  Destination address in a point to point link
         ORG   IOCN_IFUNION
IOCN_SADDRIFBROADCAST DS CL16 Address for Broadcasting
         ORG   IOCN_IFUNION
```

```
IOCN_IFMETRIC    DS F     Interface metric
         ORG   IOCN_IFUNION
IOCN_PIFDATA     DS A     Pointer to an area set by TCPIP
         ORG   IOCN_IFUNION
IOCN_MTUSIZE     DS F     MTU size.  Used with SIOCGIFMTU.
         ORG   IOCN_IFUNION
IOCN_IFATTRIF    DS 0BL2  Flag area.
*
IOCN_IFATTRBYTE1    DS 0CL1
IOCN_BIFATTRSNAP        EQU X'20'
IOCN_BIFATTRTOKBRIDGE   EQU X'10'
IOCN_BIFATTRCHECKSUM    EQU X'04'
IOCN_BIFATTRALLMULTI    EQU X'02'
IOCN_BIFATTRALLPACKSUPT EQU X'01'
         ORG   IOCN_IFATTRBYTE1+1
IOCN_IFATTRBYTE2    DS 0CL1
IOCN_BIFATTRARPNOTSUPT  EQU X'80'
IOCN_BIFATTRRESALLOC    EQU X'40'
IOCN_BIFATTRNOTRAILER   EQU X'20'
IOCN_BIFATTRPTTOPT      EQU X'10'
IOCN_BIFATTRLOOPBACK    EQU X'08'
IOCN_BIFATTRDEBUG       EQU X'04'
IOCN_BIFATTRBROADCAST   EQU X'02'
IOCN_BIFATTRUP          EQU X'01'
*
         ORG   IOCN_IFUNION+16
IOCN_IFEND       DS   0C  End of structure
IOCN_IFREQ_LEN   EQU *-IOCN_IFREQ Length of Structure
IOCN_#IFNAMESIZE EQU 16   Size of the name field
*
** BPXYIOCC End
```

BPXYIOC6 — Map IPV6 prerouter structures

BPXYIOC6 is used by transport providers. DSECT= is allowed but ignored. AMODE 64 callers use "BPXYIOC6 — Map IPV6 prerouter structures" on page 1096.

```
         BPXYIOC6   ,
NETCONFHDR           DSECT ,
* ------------- 32-Bit Version                                        @P2A
NCHEYECATCHER        DS    CL4    Eye catcher
NCHIOCTL             DS    F      Ioctl being processed (RAS)
NCHBUFFERLENGTH      DS    F      Buffer Length
NCHBUFFERPTR         DS    F      Buffer Pointer
NCHNUMENTRYRET       DS    F      Number of HomeIF returned via        *
                                       SIOCGHOMEIF6 or the number of   *
                                       GRT6RtEntry's returned via      *
                                       SIOCGRT6TABLE.
NETCONFHDR#LENGTH    EQU   *-NETCONFHDR   Length of NETCONFHDR
*
*********************************************************************
* HomeIf Structure                                                  *
*********************************************************************
*
HOMEIF               DSECT ,      HomeIf structure
HomeIfAddress        DS    CL16   Home Interface Address
*
HomeIf#LENGTH        EQU   *-HOMEIF   Length of HOMEIF
*
*********************************************************************
* GRT6RtEntry Structure                                             *
*********************************************************************
*
GRT6RTENTRY          DSECT ,      GRT6RtEntry Structure
```

```
*
GRT6DESTINATION      DS    CL16 Destination IP Address
GRT6GATEWAY          DS    CL16 First HOP on the trip if going through *
                              a gateway
GRT6DESTPREFIXLEN    DS    F    Destination's Prefix Length which is a *
                              decimal value that specifies how many *
                              of the leftmost contiguous bits of the*
                              address comprise the prefix
GRT6RTMETRIC         DS    F    Metric - hop count. Currently Tcp/Ip   *
                              returns 1 for indirect routes and 0   *
                              for direct routes. If route is from   *
                              routing daemon, metric is whatever    *
                              routing daemon set it to.
GRT6RTFLAGS          DS    F    IPV6 Route Flags.  Mapped by
*                               IPV6RtFlags structure           @P5C
*
GRT6RTENTRY#LENGTH   EQU   *-GRT6RTENTRY   Length of GRT6RTENTRY
*
*********************************************************************
* RT6Entry Structure                                                *
*********************************************************************
*
RT6ENTRY          DSECT ,         Rt6Entry Structure
*
RT6DESTINATION       DS    CL28   Destination IP address (in an IPV6   *
                                    sockaddr structure)
RT6GATEWAY           DS    CL28   First HOP on the trip if going       *
                                    through a gateway (in an IPV6      *
                                    sockaddr structure)
RT6DESTPREFIXLEN     DS    F      Destination's Prefix Length,         *
                                      which is a decimal value         *
                                      that specifies how many of       *
                                      the leftmost contiguous          *
                                      bits of the address              *
                                      comprise the prefix.
RT6METRIC            DS    F      Metric - hop count                   *
                                     Currently Tcp/IP returns          *
                                      1 for indirect route and         *
                                      0 for direct route.              *
                                     If route is from routing          *
                                     daemon, metric is whatever        *
                                     routing daemon set it to.
RT6FLAGS             DS    F      IPV6 Route Flags.
*
RT6ENTRY#LENGTH      EQU   *-RT6ENTRY   Length of RT6ENTRY
*
*********************************************************************
* GRT6RtEntryV2 Structure                                           *
*********************************************************************
*
GRT6RTENTRYV2        DSECT ,      New Route Entry used with DCR A846 - *
                                  Route Modification
*
GRT6OLDRTENTRY       DS    CL44   Old GRT6 Route Entry
GRT6RTHOMEIFIDX      DS    F      Route's Home Interface Idx
GRT6RTIFINDEX        DS    F      Route's Interface Index        @P5A
GRT6RTMTU            DS    H      Route's MTU Value              @P5A
                     DS    H      Reserved                       @P5C
                     DS    F      Reserved                       @P5C
                     DS    F      Reserved                       @P5C
*
GRT6RTENTRYV2#LENGTH   EQU   *-GRT6RTENTRYV2   Length of GRT6RTENTRYV2
*
*********************************************************************
* RT6EntryV2 Structure                                              *
*********************************************************************
*
```

```
RT6ENTRYV2          DSECT ,      New Route Entry Used with A846       *
                                 MSADDRT6V2/MSDELRT6V2 IOCTLs
*
RT6OLDENTRY         DS    CL68   Old Route Entry used before A846     *
                                 with SIOCMSADDRT6/SIOCMSDELRT6 IOCTL
RT6RTHOMEIFIDX      DS    F      Route's Home Interface Idx
                    DS    F      Reserved
                    DS    F      Reserved
                    DS    F      Reserved
                    DS    F      Reserved
*
RT6ENTRYV2#LENGTH   EQU   *-RT6ENTRYV2    Length of RT6ENTRYV2
*
*********************************************************************
* IPV6RtFlags Structure                                             *
*********************************************************************
*
IPV6RTFLAGS      DSECT ,         IPV6RtFlags Structure
*
IPV6FLGROUTETYPE    DS    XL1   Route Type                    @D1C
IPV6FLGBYTE2        DS    XL1    Reserved
IPV6FLGBYTE3        DS    XL1    Reserved
IPV6FLGBYTE4        DS    XL1    FLAGS:
*                   EQU   X'80'  Reserved
*                   EQU   X'40'  Reserved
IPV6BITV3           EQU   X'20'  1 = Version 3 fields included: @P5A
*                                      - interface index
*                                      - MTU value
IPV6BITLOOPBACK     EQU   X'10'  1 = Loopback Interface
IPV6BITHOME         EQU   X'08'  1 = Home interface
IPV6BITHOST         EQU   X'04'  1 = Host Route. 0 = Network Route
IPV6BITGATE         EQU   X'02'  1 = Gateway
IPV6BITRTUP         EQU   X'01'  1 = Route is active
*
* *-------------------------------------------------------------------*
* * SiocGifConf6 - Get IPv6 Interface Configuration.            @D3A*
* *                                                                  *
* *  Net_IfConf6Header is passed as the argument of the ioctl and   *
* *  is returned with the number of entries and entry length of the *
* *  Net_IfConf6Entry structs that were written to the output buffer.*
* *                                                                  *
* *  If Buflen=0=Buffer a Query function is performed and the       *
* *  header is returned with: (1) the maximum supported version,    *
* *  (2) the total number of entries that would be output and       *
* *  (3) the length of each individual entry.                       *
* *                                                                  *
* *  If a call to get information fails with RC=ERANGE or with      *
* *  (RC=EINVAL & Nif6h_Version is changed) the call is converted   *
* *  into a Query function and the content of the output buffer     *
* *  is unpredictable.                                              *
* *                                                                  *
* *  For information on the data returned in this structure refer   *
* *  to the z/OS Communication Server's IP Configuration Guide and  *
* *  IPv6 Network and Application Design Guide.                     *
* *                                                                  *
* *-------------------------------------------------------------------*
NET_IFCONF6HEADER DSECT   Header                                 @D3A
NIF6H_VERSION DS F       Input for Get IfConf6 Output for Query
NIF6H_ENTRIES DS F       Output: number of entries returned in output  *
                         buffer
NIF6H_ENTRYLEN DS F      Output: length of an entry
NIF6H_BUFLEN DS F        Input: length of buffer
NIF6H_BUFFER64 DS 0CL8   Input: Amode(64) Buffer ptr
NIF6H_BUFFER64H DS F
NIF6H_BUFFER DS A        Input: Amode(31) Buffer ptr to output buffer  *
                         that will be filled with an array of          *
                         Net_IfConf6Entrys.
```

```
NET_IFCONF6HEADER_LEN EQU *-NET_IFCONF6HEADER
*
NET_IFCONF6ENTRY DSECT   Entry                                  @D3A
NIF6E_NAME DS  CL16      x00 interface name (blank padded - no null)
NIF6E_STACKNAME DS CL8   x10 tcpip stack name (blank padded - no null)
NIF6E_ADDR DS  CL28      x18 Sock_Inet6_SockAddr of the interface
NIF6E_ROUTEMETRIC DS F   x34 route metric
NIF6E_PREFIXLEN DS H     x38 routing prefix length
NIF6E_PREFIXORIGIN DS X  x3A prefix origin, see below
NIF6E_STATUS DS X        x3B status, see below
NIF6E_FLAGS DS 0BL4      x3C Flags:
NIF6E_FLAGS1 DS B
NIF6E_FLAGS2 DS B
NIF6E_FLAGS3 DS 0B
NIF6E_VIRTUAL EQU X'40'
NIF6E_MULTIPOINT EQU X'08'
NIF6E_MULTICASTCAPABLE EQU X'04'
         ORG   NIF6E_FLAGS3+1
NIF6E_FLAGS4 DS 0B
NIF6E_POINT2POINT EQU X'10'
NIF6E_LOOPBACK EQU X'08'
NIF6E_ONLINK EQU X'01'
         ORG   NIF6E_FLAGS+4
NIF6E_MTU DS   F         x40 mtu
*
*    ********************************************************************
*    *                                                                  *
*    * Constants for nif6h_version                                 @D3A
*    *                                                                  *
*    ********************************************************************
*
*
NIF6H#VER EQU  1         Current Version
NIF6H#VER1 EQU 1         Initial Version
*
*    ********************************************************************
*    *                                                                  *
*    * Constants for nif6e_prefixorigin                            @D3A
*    *                                                                  *
*    ********************************************************************
*
*
NIF6H#WELLKNOWN EQU 1
NIF6H#MANUAL EQU 2
NIF6H#RTRADV EQU 3
NIF6H#OTHER EQU 8
*
*    ********************************************************************
*    *                                                                  *
*    * Constants for nif6e_status                                  @D3A
*    *                                                                  *
*    ********************************************************************
*
*
NIF6H#PREFERRED EQU 1
NIF6H#DEPRECATED EQU 2
NIF6H#INVALID EQU 3
NIF6H#INACCESSIBLE EQU 4
NIF6H#UNKNOWN EQU 5
NIF6H#TENTATIVE EQU 6
NIF6H#DUPLICATE EQU 7
NET_IFCONF6ENTRY_LEN EQU *-NET_IFCONF6ENTRY
*                                     End SiocGifConf6 -------- @D3A
*
* ********************************************************************
* *                                                                  *
* *   Constants                                                      *
```

```
* *                                                                   *
* ********************************************************************
*
*
IOC6_#HOMEIFPREFIXLEN EQU 128 The prefix length for a home interface   *
                        address returned on the SIOCGHOMEIF6 IOCTL.
IOC6_NCH#EYE EQU C'6NCH' IPV6 Network Configuration Header EyeCatcher.
IOC6_NCH64#EYE EQU C'6N64' IPV6 NetConfHdr EyeCatcher 64-BIT
*
* ********************************************************************
* *                                                                   *
* * Maximum hop count for the Metric fields:                          *
* *    GRT6RtMetric                                                   *
* *    Rt6Metric                                                      *
* *                                                                   *
* ********************************************************************
*
*
IOC6_#MAXHOPMETRIC EQU 16
*
* ********************************************************************
* *                                                                   *
* *   Constants used for size of control areas                        *
* *                                                                   *
* ********************************************************************
*
*
IOC6_#MAXROUTES EQU 600
IOC6_#GRT6ROUTELEN EQU 44
*
*    ******************************************************************
*     Initial buffer size for SIOCGHOMEIF6 and SIOCGRT6TABLE.
*    ******************************************************************
*
*
IOC6_#MAXGRT6LEN EQU 26400
IOC6_#NETCONFHDRLEN EQU 20
IOC6_#GRT6V2ROUTELEN EQU 64
IOC6_#MAXGRT6V2LEN EQU 38400
*
** BPXYIOC6 End
```

BPXYIOV — Map the I/O vector structure

BPXYIOV is used by readv(), writev(), sendmsg() and recvmsg(). AMODE 64 callers use "BPXYIOV — Map the I/O vector structure" on page 1100.

```
            BPXYIOV    ,
** BPXYIOV: Socket I/O Vectors
**  Used By: FCT OPN
IOV                  DSECT ,
IOV_ENTRY            DS    0F     Array Entry
*  ------------------------- 31-bit format
IOV_BASE             DS    A      Address of buffer
IOV_LEN              DS    F      Length of buffer
*
IOV#LENGTH          EQU   *-IOV_ENTRY Length of this structure
IOV_MAX             EQU   120     Maximum number of entries
** BPXYIOV End
```

BPXYIPCP — Map interprocess communication permissions

```
           BPXYIPCP   ,
** BPXYIPCP: Interprocess Communications Permission
**  Used By: MCT, MGT, SCT, SGT, QCT, QGT
IPC_PERM            DSECT ,      Interprocess Communications
IPC_UID             DS    F      Owner's effective user ID
IPC_GID             DS    F      Owner's effective group ID
IPC_CUID            DS    F      Creator's effective user ID
IPC_CGID            DS    F      Creator's effective group ID
IPC_MODE            DS    XL4    Mode, mapped by BPXYMODE
IPC#LENGTH   EQU   *-IPC_PERM  Length of Interprocess Control block
                    SPACE ,
* Key:
IPC_PRIVATE         EQU   0      Private key.
                    SPACE ,
* Mode bits:                    Map over S_TYPE in BPXYMODE
IPC_CREAT           EQU   1      Create entry if key does not exist.
IPC_EXCL            EQU   2      Fail if key exists.
IPC_MEGA            EQU   4      Allocation in meg
IPC_BINSEM          EQU   4      Binary semaphore
IPC_RCVTYPEPID      EQU   4      Msgrcv TYPE=PID
IPC_SNDTYPEPID      EQU   8      Msgsnd TYPE=PID
IPC_PLO1            EQU  16      Use PLO for serialization
IPC_SHORTHOLD       EQU  16      Binary semaphore short        @D5A
IPC_PLO2            EQU  32      Use PLO if practical
IPC_PLOINUSE        EQU   1      PLO is in use (_getipc only)
IPC_GIGA            EQU   8      Allocation in Gig - amode 64  @D6A
IPC_BELOWBAR        EQU  16      Allocate below bar            @D6A
IPC_SHAREAS         EQU  32      Share within Address Space    @D7A
                    SPACE ,
* Flag bits - semop, msgrcv, msgsnd:
IPC_NOWAIT          EQU   1      Error if request must wait.
                    SPACE ,
* Control Command:
IPC_RMID            EQU   1      Remove identifier.
IPC_SET             EQU   2      Set options.
IPC_STAT            EQU   3      Access status.
                    SPACE ,
* CONSTANTS WHICH MAP OVER BYTE S_TYPE, SEE BPXYMODE
** BPXYIPCP End
```

BPXYIPCQ — Map w_getipc structure

AMODE 64 callers use "BPXYIPCQ — Map w_getipc structure" on page 1100.

```
           BPXYIPCQ   ,
*************************************************************************
*                                                                       *
*  BPXYIPCQ: w_getipc interface mapping                                 *
*   Used By: BPXGXGET                                                   *
*                                                                       *
*************************************************************************
IPCQ          DSECT ,      Interprocess Communications - Query
IPCQLENGTH    DS    F      IPCQ#LENGTH used by system call.  If not
*                          equal, check BPXYIPCQ and system levels.
IPCQTYPE      DS    CL4    "IMSG", "ISEM", "ISHM", "OVER", "IMAP"
IPCQOVER      DS    0D     OVERVIEW MAPPING STARTS HERE
*-----------------------------------------------------------------------*
*  For IPCQTYPE = OVER, data starts here and the rest of the fields     *
*  in this section of code are not filled in.                           *
*-----------------------------------------------------------------------*
IPCQMID       DS    FL4    MEMBER ID
IPCQKEY       DS    XL4    KEY
IPCQIPCP      DS    CL20   MAPPED BY BPXYIPCP
```

```
IPCQGTIME     DS    XL4    TIME_T OF LAST ...GET()
IPCQCTIME     DS    XL4    TIME_T OF LAST ...CTL()
IPCQTTIME     DS    XL4    TIME_T CHANGED BY TERMINATION
*-------------------------------------------------------------------*
*  Start of Unique data for IPCQTYPE requested                       *
*-------------------------------------------------------------------*
IPCQREST      DS    0C     IPCQMSG, IPCQSHM, IPCQSEM, MAPPED MEMORY
*********************************************************************
* Message Queue unique data                                         *
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQBYTES     DS    F      # BYTES OF MESSAGES ON QUEUE
IPCQQBYTES    DS    F      MAX # BYTES OF MESSAGES ALLOWED ON QUEUE
IPCQLSPID     DS    F      PID OF LAST MSGSND()
IPCQLRPID     DS    F      PID OF LAST MSGRCV()
IPCQSTIME     DS    F      TIME_T OF LAST MSGSND()
IPCQRTIME     DS    F      TIME_T OF LAST MSGRCV()
IPCQNUM       DS    F      # OF MESSAGES ON QUEUE
IPCQRCNT      DS    F      COUNT OF WAITING MSGRCV
IPCQSCNT      DS    F      COUNT OF WAITING MSGSND
              DS    0CL16  MSGRCV AND MSGSND WAITERS
              DS    0CL8    MSGRCV - WAIT FOR TYPE
IPCQQRPID     DS    F        PROCESS ID
IPCQQRMSGTYPE DS    F        MESSAGE TYPE
              DS    0CL8    MSGSND - WAIT FOR ROOM TO SEND
IPCQQSPID     DS    F        PROCESS ID
IPCQQSMSGLEN  DS    F        MESSAGE LENGTH
              DS    9CL16  MSGSND AND MSGRCV WAITERS
              DS    0CL8   MESSAGES WAITING TO BE RECEIVED
IPCQQMPID     DS    F        PROCESS ID
IPCQQMMSGTYPE DS    F        MESSAGE TYPE
              DS    9CL8   MESSAGES
              DS    F      Reserved
              DS    0D
* The 64 bit time fields will be set for either 31 or 64 bit mode
* Must define storage different, depending on how assembled
* AMODE 31
IPCQSTIME64   DS    2F     TIME64_T OF LAST MSGSND()
IPCQRTIME64   DS    2F     TIME64_T OF LAST MSGRCV()
IPCQQRMSGTYPE64 DS  20F    MSGRCV 64 BIT MSG TYPE
IPCQQMMSGTYPE64 DS  20F    MSG WAITING 64 BIT MSG TYPE
              DS    CL96   Reserved for expansion
*********************************************************************
* Semaphore unique data                                             *
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQLOPID     DS    XL4    PID OF LAST SEMOP
IPCQOTIME     DS    F      TIME_T LAST SEMOP
IPCQADJBADCNT DS    F      TERMINATION BUMPS SEM_VAL LIMITS
IPCQNSEMS     DS    FL2    NUMBER OF SEMAPHORES IN THIS SET
IPCQADJCNT    DS    FL2    NUMBER OF UNDO STRUCTURES
IPCQNCNT      DS    FL2    COUNT OF WAITERS FOR >0
IPCQZCNT      DS    FL2    COUNT OF WAITERS FOR =0
              DS    0CL16  WAITERS AND ADJUSTERS
              DS    0CL8    WAITER
IPCQSWPID     DS    F        PROCESS ID
IPCQSWNUM     DS    H        SEMAPHORE NUMBER
IPCQSWOP      DS    H        SEMAPHORE OPERATION
              DS    0CL8    ADJUSTER
IPCQSAPID     DS    F        PROCESS ID
IPCQSANUM     DS    H        SEMAPHORE NUMBER
IPCQSAADJ     DS    H        SEMAPHORE OPERATION
              DS    9CL16  WAITERS AND ADJUSTERS
              DS    0D
* AMode 31
```

```
IPCQOTIME64   DS    2F     TIME64_T LAST SEMOP
              DS    CL360  Reserved for expansion
*********************************************************************
* Shared Memory unique data                                         *
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQACNT      DS    F      USE COUNT (#SHMAT - #SHMDT)
IPCQSEGSZ     DS    F      MEMORY SEGMENT SIZE
IPCQDTIME     DS    F      TIME_T OF LAST SHMDT()
IPCQATIME     DS    F      TIME_T OF LAST SHMAT()
IPCQLPID      DS    F      PID OF LAST SHMAT() OR SHMDT()
IPCQCPID      DS    XL4    PID OF CREATOR
*-------------------------------------------------------------------*
* 31 bit callers - 10 Element array of segments attached             *
*    Each element is the 4 byte PID followed by the 31 bit address   *
*-------------------------------------------------------------------*
IPCQATPID     DS    F      ATTACHED PROCESS ID
IPCQATADDRESS DS    F      SEGMENT ADDRESS FOR PROCESS
              DS    18F    MORE ATTACHED PROCESS IDS AND
*                          SEGMENT ADDRESS
              DS    20F    Reserved - match 64 bit lengths for array
IPCQDTIME64   DS    2F     TIME_T OF LAST SHMDT()
IPCQATIME64   DS    2F     TIME_T OF LAST SHMAT()
              DS    2F     Reserved - segment size in 64 bit section
              DS    CL344  Reserved
*********************************************************************
* Mapped Memory unique data                                         *
*********************************************************************
              ORG   IPCQREST Mapped Memory unique data
              DS    0F
IPCQMAPCPID   DS    F      CREATOR PROCESS ID
IPCQMAPUPID   DS    F      USER PROCESS ID
IPCQMAPTOKEN  DS    2F     MAP TOKEN
IPCQMAPUID    DS    F      USER'S EFFECTIVE UID
IPCQMAPGID    DS    F      USER'S EFFECTIVE GID
IPCQMAPFLAGS  DS    XL4    FLAGS
* Flags in first byte
IPCQMAPSHUT   EQU   X'80'  SHUTDOWN OF OBJECT
IPCQBLKSZ     DS    F      SIZE OF BLOCKS IN MEGS
IPCQBLKSINUSE DS    F      NUMBER OF BLOCKS IN USE
IPCQBLKSINMAP DS    F      NUMBER OF BLOCKS IN MAP AREA
IPCQBLKSMAPPED DS   F      NUMBER OF BLOCKS MAPPED
*                          BY THIS PROCESS
              DS    CL508  Reserved for expansion
*********************************************************************
* Continuation of Common data                                       *
* This next ORG gets us past the largest unique section of data     *
* We need to preserve the field offsets from prior releases so      *
* needed to add the rest of this common data at the end of the      *
* unique data instead of within the common area defined above.      *
*********************************************************************
              ORG
IPCQGTIME64   DS    2F     TIME64_T OF LAST ...GET()
IPCQCTIME64   DS    2F     TIME64_T OF LAST ...CTL()
IPCQTTIME64   DS    2F     TIME64_T CHANGED BY TERMINATION
IPCQSECLABEL  DS    2F     SECLABEL
*********************************************************************
* Overview - summary data for msgqs, semaphores, shared memory      *
*********************************************************************
              ORG   IPCQOVER Overview
                 DS    0F  MESSAGE QUEUES
IPCQOMSGNIDS     DS    F        Maximum number MSQs allowed
IPCQOMSGHIGHH2O  DS    F        Most MSQs at one time
IPCQOMSGFREE     DS    F        Number MSQs available
IPCQOMSGPRIVATE  DS    F        Number MSQs with Ipc_PRIVATE
IPCQOMSGKEYED    DS    F        Number MSQs with KEYs
```

```
IPCQOMSGREJECTS  DS    F        TIMES MSGGET DENIED
IPCQOMSGQBYTES   DS    F        MAX BYTES PER QUEUE
IPCQOMSGQMNUM    DS    F        MAX NUMBER MESSAGES PER QUEUE
IPCQOMSGNOALC    DS    F        # MSGSNDS THAT RETURNED ENOMEM
                 DS    F
                 DS    0F  SEMAPHORE
IPCQOSEMNIDS     DS    F        Maximum number SEMs allowed
IPCQOSEMHIGHH2O  DS    F        Most SEMs at one time
IPCQOSEMFREE     DS    F        Number SEMs available
IPCQOSEMPRIVATE  DS    F        Number SEMs with Ipc_PRIVATE
IPCQOSEMKEYED    DS    F        Number SEMs with KEYs
IPCQOSEMREJECTS  DS    F        TIMES SEMGET DENIED
IPCQOSEMSNSEMS   DS    F        MAX NUMBER OF SEMAPHORES PER SET
IPCQOSEMSNOPS    DS    F        MAX NUMBER OPERATION IN SEMOP
IPCQOSEMSBYTES   DS    F        STORAGE LIMIT
IPCQOSEMCBYTES   DS    F        STORAGE COUNT
                 DS    F
                 DS    0F  SHARED MEMORY
IPCQOSHMNIDS     DS    F        Maximum number SHMs allowed
IPCQOSHMHIGHH2O  DS    F        Most SHMs at one time
IPCQOSHMFREE     DS    F        Number SHMs available
IPCQOSHMPRIVATE  DS    F        Number SHMs with Ipc_PRIVATE
IPCQOSHMKEYED    DS    F        Number SHMs with KEYs
IPCQOSHMREJECTS  DS    F        TIMES SHMGET DENIED
IPCQOSHMSPAGES   DS    F        MAX # PAGES PER SYSTEM LIMIT
IPCQOSHMMPAGES   DS    F        MAX # PAGES PER SEGMENT LIMIT - ZERO
*                               IF 32 BITS EXCEEDED - USE
*                               IPCQOSHMMPAGES64 FOR GREATER THAN 32
*                               BITS
IPCQOSHMNSEGS    DS    F        MAX # SEGMENTS PER PROCESS LIMIT
IPCQOSHMCPAGES   DS    F        CURRENT # BYTES SYSTEM WIDE
*                               This field does not include pages for
*                               shared memory requests processed with
*                               the ipc_MEGA option
IPCQOSHMBIGGEST  DS    F        LARGEST SEGMENT ALLOCATED - ZERO IF
*                               32 BITS EXCEEDED - USE
*                               IPCQOSHMBIGGEST64 FOR GREATER THAN 32
*                               BITS
                 DS    0D
IPCQOSHMMPAGES64  DS   2F       MAX # PAGES PER SEGMENT LIMIT
IPCQOSHMBIGGEST64 DS   2F       LARGEST SEGMENT ALLOCATED
             ORG   ,
IPCQ#LENGTH   EQU   *-IPCQ Storage needed for w_getipc function
* w-getipc Command:
IPCQ#MSG      EQU   1      Retrieve next message queue
IPCQ#SHM      EQU   2      Retrieve next shared memory segment
IPCQ#SEM      EQU   3      Retrieve next semaphore set
IPCQ#ALL      EQU   4      Retrieve next member, all mechanisms
IPCQ#OVER     EQU   5      Retrieve overview
IPCQ#MAP      EQU   6      Retrieve mapped memory
** BPXYIPCQ End
```

BPXYITIM — Map getitimer, setitimer structure

AMODE 64 callers use “BPXYITIM — Map getitimer, setitimer structure” on page 1103.

```
           BPXYITIM   ,
** BPXYITIM: getitimer and setitimer interval structure
**  Used By: GTR STR
ITIM               DSECT ,
** STRUCTURE OF GETITIMER (PARAMETER 2), SETITIMER (PARAMETERS 2,3)
ITIMIPAIR          DS    0CL8      Initial value or value at cancel
ITIMISECONDS       DS    F           Seconds       0-7FFFFFFF   x
ITIMIMICROSEC      DS    0F          Microseconds  0-000F423F   x
ITIMINANOSEC       DS    F           Nanoseconds   0-369AC9FF   x
```

```
ITIMRPAIR           DS    0CL8      Reload Interval
ITIMRSECONDS        DS    F           Seconds       0-2147483647 d
ITIMRMICROSEC       DS    0F          Microseconds  0-999999     d
ITIMRNANOSEC        DS    F           Nanoseconds   0-999999999  d
ITIMER_REAL         EQU   0         REAL TIME
ITIMER_VIRTUAL      EQU   1         VIRTUAL TIME (CPU - SYSTEM)
ITIMER_PROF         EQU   2         CPU TIME
ITIMER_MICRO        EQU   0         1/1,000,000 of seconds
ITIMER_NANO         EQU   4         1/1,000,000,000 of seconds
ITIM#LENGTH         EQU   16        LENGTH THIS STRUCTURE
** BPXYITIM End
```

BPXYMMG — Map interface for _map_init and _map_service

AMODE 64 callers use "BPXYMMG — Map Interface for _map_init and _map_service" on page 1104.

```
           BPXYMMG    ,
** BPXYMMG: BPX1MMI & BPX1MMS Interface Declares
**  Used By: Callers of the BPX1MMI & BPX1MMS Interface
*
*********************************************************************
*
*   Function Code Constants
*
*********************************************************************
*
MMG_INIT              EQU 1
MMG_SERVICE           EQU 2
*
*********************************************************************
*                                                                   *
*   Parameter list mapping for the BPX1MMI MMG_INIT call            *
*                                                                   *
*********************************************************************
*
_MMG_INIT_PARM   DSECT ,        MMG_INIT Parameter List
_MMG_NUMBLKS     DS    F        Fullword that contains the number of
*                               blocks to be contained in the map
*                               area.
_MMG_MEGSPERBLK  DS    F        Fullword that contains the size in
*                               megabytes of each block in the map
*                               area
_MMG_MAPTOKEN    DS    CL8      Token for map area
_MMG_RES01A      DS    A        Reserved for future use
_MMG_RES01B      DS    A        Reserved for future use
_MMG_AREAADDR    DS    A        Fullword that contains, on input,
*                               the suggested starting address of the
*                               map area or 0.  On output, this field
*                               is set to the actual map starting
*                               address.
_MMG_INIT_PARM_LEN EQU *-_MMG_INIT_PARM
*
*********************************************************************
*
*   Parameter list mapping for the BPX1MMS MMG_SERVICE request
*
*   The parameter list is an array of entries, each entry having the
*   format as mapped by _MMG_SERVICE_BLK.  Each entry is a request for
*   one of the supported request types:  MMG_NEWBLOCK, MMG_CONN,
*   MMG_DISCONN, MMG_CNTL or MMG_FREE.  In addition, an entry can be
*   marked as inactive by setting its value to MMG_NOP, which will
*   cause the entry to be skipped.  The result of a given request will
*   be reflected in the array entry.
*
*   The meaning of array entry fields is dependant on the requested
```

```
*   function.  The following table defines the field meanings for each
*   of the supported functions. A field not used by a service is marked
*   N/A.  Fields so marked are ignored and their value is not
*   important for the specified service.  All reserved fields must be
*   zero.
*
*   Function      Field               Field usage
*   ---------------------------------------------------
*   _newblock
*                 _MMG_SERVICETYPE    MMG_NEWBLOCK
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     MMG_NOCONN may be set to one if
*                                     the new block is to be allocated
*                                     in the backing storage but not
*                                     connected to the map area
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          output
*                 _MMG_BlkAddr        input  - 0 or address where the
*                                              new block is to be
*                                              allocated
*                                     output - An address in the map
*                                              area where the new
*                                              block was allocated
*   _conn
*                 _MMG_SERVICETYPE    MMG_CONN
*                 _MMG_SERVICEIFLAG   All bits should be zero
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          input
*                 _MMG_BlkAddr        input  - 0 or address where the
*                                              block identified by
*                                              token is to be
*                                              allocated
*                                     output - An address in the map
*                                              area where the block
*                                              was allocated
*   _disconn
*                 _MMG_SERVICETYPE    MMG_DISCONN
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     the MMG_FREE bit may be on if
*                                     backing storage is to be
*                                     released for the data
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          N/A
*                 _MMG_BlkAddr        input  - Address of the block
*                                              containing data to
*                                              be disconnected
*   _free
*                 _MMG_SERVICETYPE    MMG_FREE
*                 _MMG_SERVICEIFLAG   All bits should be zero
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          input  - Token of the data
*                                              contained in the
*                                              backing storage which
*                                              is to be release
*                 _MMG_BlkAddr        N/A
*   _cntl
*                 _MMG_SERVICETYPE    MMG_CNTL
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     those that define the access
*                                     state of the data (read or
*                                     read/write flags)
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          N/A
*                 _MMG_BlkAddr        input  - Address of the block
*                                              containing data to be
*                                              affected by the state
*                                              change
```

```
*
*********************************************************************
*
_MMG_SERVICE_PARM DSECT ,       MMG_SERVICE Parameter List
_MMG_SERVICE_ENTRY DS 0H
_MMG_SERVICETYPE DS    FL2      Type of service requested. eg, MMG_CONN
_MMG_SERVICEIFLAG DS   BL1      Flags
                 ORG   _MMG_SERVICEIFLAG
_MMG_READONLY    EQU   X'80'    All pages of each area are to be made
*                               read-only
_MMG_READWRITE   EQU   X'40'    All pages of each area are to be made
*                               read-write
_MMG_FREEBLOCK   EQU   X'20'    The backing storage for the specified
*                               block is to be freed
_MMG_NOCONN      EQU   X'10'    The new block is to be allocated in the
*                               backing storage but not connected to
*                               the map area
                 ORG   _MMG_SERVICEIFLAG+L'_MMG_SERVICEIFLAG
_MMG_SERVICEOFLAG DS   BL1      Flags
                 ORG   _MMG_SERVICEOFLAG
_MMG_REQFAIL     EQU   X'80'    If on, a failure occured on this entry
*                               or this entry was not processed
                 ORG   _MMG_SERVICEOFLAG+L'_MMG_SERVICEOFLAG
_MMG_TOKEN       DS    CL8      Token for a data block
_MMG_RES02B      DS    A        Reserved
_MMG_BLKADDR     DS    A        Fullword that contains the virtual
*                               address of a map area block
_MMG_MAXARRAYCOUNT EQU 1000     Maximum number of requests that can be
*                               in a service request array
_MMG_SERVICE_PARM_LEN EQU *-_MMG_SERVICE_PARM
*
*********************************************************************
*
*   BPX1MMS SERVICE Request Constants (values for field
*   _MMG_SERVICETYPE)
*
*********************************************************************
*
MMG_NOP                EQU 0
MMG_NEWBLOCK           EQU 1
MMG_CONN               EQU 2
MMG_DISCONN            EQU 3
MMG_FREE               EQU 4
MMG_CNTL               EQU 5
*
*********************************************************************
*
** BPXYMMG End
```

BPXYMNTE — Map response and element structure of w_getmntent

DSECT (MNTENTPARMDATA) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. To get the new version of the MNTE, set MNTE2=YES. Users of MNTE2=YES must initialize MNTEHID to 'MNT2' and set MNTEHBLEN to MNTE#LENGTH.

```
           BPXYMNTE   MNTE2=YES
** BPXYMNTE: z/OS UNIX w_getmntent response structure and element
**  Used By: GMN
MNTEH               DSECT ,
MNTEHID             DC    C'MNT2'   Eye catcher
MNTEHSIZE           DC    A(MNTE#LENGTH)   Size of area (MNTEH+MNTE)
MNTEHCUR            DC    XL8'0000000000000000'
*                                Index of next element to return
```

```
*                                  - must be zero (i.e.
*                                    X'0000000000000000'),
*                                    on initial call
*                                  - must be left undisturbed
*                                    for subsequent calls
MNTEHDEVNO           DS     F'0'  Device number - this value is
*                                 specified if information about only
*                                 one file system is requested
MNTEHBLEN            DS     F     Length of mnte body used
MNTEHRES1            DS     BL8   Reserved for future - must be zero
*                                 on entry
MNTEH#LENGTH         EQU   *-MNTEH Length of header structure
*
MNTE                 DSECT ,
MNTENTBODYV1V2       DS    0F     Define V1 and V2 body size
MNTENTFSTYPE         DS    F      File system type
MNTENTFSTYPEMVS      EQU   1      MVS Local File System
MNTENTFSTYPEREMOTE   EQU   2      Remote File System
MNTENTFSTYPEPIPE     EQU   3      Pipe file system
MNTENTFSTYPESOCKET   EQU   4      Socket file system
MNTENTFSTYPEXPFS     EQU   5      Cross System PFS (XPFS)
MNTENTFSTYPECSPS     EQU   6      Char special streams
MNTENTFSTYPENFS      EQU   MNTENTFSTYPEREMOTE
MNTENTFSMODE         DS    0F     File system mount flags
MNTENTFSMODE1        DS    B      File system mount method - byte 1
MNTENTFSMODE2        DS    B      File system mount method - byte 2
MNTENTFSMODE3        DS    B      File system mount method - byte 3
MNTENTFSSYNCHONLY    EQU   X'01'  File system SynchOnly specified
MNTENTFSMODE4        DS    B      File system mount method - byte 4
MNTENTSECACL         EQU   X'80'  Acls supported by sec product
MNTENTFSAUNMOUNT     EQU   X'40'  UnMount during recovery
MNTENTFSCLIENT       EQU   X'20'  File system is a client
MNTENTFSNOAUTOMOVE   EQU   X'10'  Automove allowed
MNTENTFSMODENOSEC    EQU   X'08'  No Security checks enforced
MNTENTFSMODEEXPORT   EQU   X'04'  File system exported by DFS
MNTENTFSMODENOSUID   EQU   X'02'  SetUID not permitted for
*                                 files in this file system
MNTENTFSMODERDONLY   EQU   X'01'  File system mounted read only
MNTENTFSMODERDWR     EQU   X'00'  File system mounted read/write
MNTENTFSDEV          DS    F      st_dev value to be returned by
*                                 the stat system call for all files
*                                 in this file system
MNTENTPARENTDEV      DS    F      st_dev of the parent file system
MNTENTROOTINO        DS    F      ino of the mount point
MNTENTSTATUS         DS    B      Status of the file system
MNTENTFILEACTIVE     EQU   B'00000000' File system is active
MNTENTFILEDEAD       EQU   B'00000001' File system is dead
MNTENTFILERESET      EQU   B'00000010' File system being reset
MNTENTFILEDRAIN      EQU   B'00000100' File system being unmounted with
*                                        drain option
MNTENTFILEFORCE      EQU   B'00001000' File system being unmounted with
*                                        force option
MNTENTFILEIMMED      EQU   B'00010000' File system being unmounted with
*                                        immed option
MNTENTFILENORM       EQU   B'00100000' File system being unmounted with
*                                        normal option
MNTENTIMMEDTRIED     EQU   B'01000000' File system Umount immed failed
MNTENTQUIESCED       EQU   B'10000000' File system is quiesced
MNTENTMNTINPROGRESS  EQU   B'10000001' Mount in progress for
*                                        this file system
MNTENTASYNCHMOUNT    EQU   B'10000010' Asynchronous mount in progress
*                                        for this file system
MNTENTFSDDNAME       DS    CL9    DDNAME specified on mount - null
*                                 terminated
MNTENTFSTNAME        DS    CL9    File system type name -
*                                 from the FILESYSTYPE parmlib
*                                 statement - null terminated
```

```
MNTENTFSNAM44        DS    CL44   File system name - as a 44 byte field
                     ORG   MNTENTFSNAM44
MNTENTFSNAME         DS    CL45   File system name - for PDSE/X, this
*                                 is the name of the PDSE/X containing
*                                 file system, null terminated
MNTENTPATHLEN        DS    F      length of mount point path name
MNTENTMOUNTPOINT     DS    CL1024 Name of directory where the file
*                                 system is mounted - (mount point
*                                 path name - null terminated
MNTENTJOBNAME        DS    CL8    Job name of quiesce requestor
MNTENTPID            DS    F      PID of quiesce requestor
MNTENTPARMOFFSET     DS    F      Offset of MntEntParm from MNTE
*                                 (Zero if none)
MNTENTPARMLEN        DS    H      Length of mount parameter
*                                 (Zero if none)
MNTENTSYSNAME        DS    CL8    Name of system to mount on
MNTENTQSYSNAME       DS    CL8    Name of queisce system name
MNTENTFROMSYS        DS    CL8    Filesystems to be moved from here
MNTENTRES00          DS    2B     Alignment
MNTENTRFLAGS         DS    0F     Request flags
MNTENTRFLAGS1        DS    B      Request flags - byte 1
MNTENTRFLAGS2        DS    B      Request flags - byte 2
MNTENTRFLAGS3        DS    B      Request flags - byte 3
MNTENTRFLAGS4        DS    B      Request flags - byte 4
MNTENTCHANGE         EQU   X'01'  Change f.s. server request
MNTENTNEWAUTO        EQU   X'02'  Change automove setting
MNTENTSTATUS2        DS    0F     Status of filesystem
MNTENTSTATUS2B1      DS    B      Status of filesystem - byte 1
MNTENTSTATUS2B2      DS    B      Status of filesystem - byte 2
MNTENTSTATUS2B3      DS    B      Status of filesystem - byte 3
MNTENTSTATUS2B4      DS    B      Status of filesystem - byte 4
MNTENTUNOWNED        EQU   B'00000001' File system unowned
MNTENTINRECOVERY     EQU   B'00000010' File system in recovery
MNTENTSUPERQUIESCED  EQU   B'00000100' File system super quiesced
MNTENTSUCCESS        DS    F      Successful moves
MNTENTREADCT         DS    F      Number of reads from filesys
MNTENTWRITECT        DS    F      Number of writes done
MNTENTDIRIBC         DS    F      Number of directory I/O blocks
MNTENTREADIBC        DS    F      Number of read I/O blocks
MNTENTWRITEIBC       DS    F      Number of write I/O blocks
MNTENTBYTESREAD      DS    BL8    Number of bytes read
MNTENTBYTESWRITTEN   DS    BL8    Number of bytes written
MNTENTFILETAG        DS    CL4    File tag (see BPXYSTAT)
MNTENTSYSLISTOFFSET  DS    F      Offset of system list
MNTENTSYSLISTLENGTH  DS    H      Length of system list
MNTENTAGGNAMELENGTH  DS    H      Length of Aggregate name
MNTENTAGGNAMEOFFSET  DS    F      Aggregate Name Offset or 0
MNTENTROSECLABEL     DS    CL8    Readonly seclabel
MNTE#LENGTH          EQU   *-MNTE Length of this structure
*
*
MNTENTPARMDATA       DSECT ,      Mount() parameter data dsect
MNTENTPARM           DS    0C     Parameter specified with mount()
*
*
MNTENTSYSLISTINFO    DSECT ,      Deadsys move to syslist dsect
MNTENTSYSLISTNUM     DS    H      Number of entries in the syslist
MNTENTSYSLISTFLAGS   DS    H      Flags
MNTENTSYSLISTINCL    EQU   X'0000'  Include syslist
MNTENTSYSLISTEXCL    EQU   X'0001'  Exclude syslist
MNTENTSYSLIST        DS    32CL8  System names
*
*
MNTENTAGGNAMEDSECT   DSECT ,      At MntEntAggNameOffset if not 0
MNTENTAGGNAME        DS    0C     Aggregate Name, Null terminated
*
* To access MNTEH, MNTE and MNTENTPARM:
```

```
* LA     RegOne,buffer            RegOne->BPX1GMN buffer and MNTEH
* USING  MNTEH,RegOne             Addressability to MNTEH
*
* LR     RegTwo,RegOne            RegTwo->MNTEH
* LA     RegTwo,MNTEH#LENGTH(RegTwo) RegTwo->MNTE
* USING  MNTE,RegTwo              Addressability to MNTENTPARMLEN
*                                 and MNTENTPARMOFFSET
*
* ICM    RegThree,15,MNTENTPARMOFFSET Load offset from start of
*                                 entry  (i.e. start of MNTE)
* BZ     SkipParm                 If zero, skip processing parm
* ALR    RegThree,RegTwo          RegTwo->MNTE,
*                                 RegThree=MNTENTPARMOFFSET
*                                 RegThree->MNTENTPARMDATA (after)
* USING  MNTENTPARMDATA,RegThree  Addressability to MNTENTPARMDATA
*
** BPXYMNTE End
```

BPXYMODE — Map the mode constants of the file services

```
           BPXYMODE   ,
** BPXYMODE: Mode constants specified on system calls
**  Used By: CHM FCM MKD MKN OPN UMK
S_MODE                DSECT ,
                      DS    0F
*
S_TYPE                DS    B      File types, mapped by BPXYFTYP
*                                  Flag bytes
S_MODE3B              DS    0XL3   All flag bytes
S_RES01               DS    0BL.8  Reserved
S_MODE1               DS    B      Flag byte 1 - reserved
*
S_RES02               DS    0BL.4  Reserved
S_MODE2               DS    B      Flag byte 2
*                                  Set ID flags
S_ISUID               EQU   X'08'  Set user ID on execution
S_ISGID               EQU   X'04'  Set group ID on execution
S_ISVTX               EQU   X'02'  Sticky Bit: For executables, look
*                                   first in normal MVS search order
*                                   For directories, deletion rstd
*                                   to owner or superuser.
*                                  Owner flags
S_IRWXU1              EQU   X'01'  All permissions for user - part I
S_IRUSR               EQU   X'01'  Read permission
*
S_MODE3               DS    B      Flag byte 3
*                                  Owner flags - continued
S_IRWXU2              EQU   X'C0'  All permissions for user - Part II
S_IWUSR               EQU   X'80'  Write permission
S_IXUSR               EQU   X'40'  Search (if a directory) or
*                                  execute (otherwise) permission
*                                  Group flags
S_IRWXG               EQU   X'38'  All permissions for group
S_IRGRP               EQU   X'20'  Read permission
S_IWGRP               EQU   X'10'  Write permission
S_IXGRP               EQU   X'08'  Search (if a directory) or
*                                  execute (otherwise) permission
*                                  Other flags
S_IRWXO               EQU   X'07'  All permissions for other
S_IROTH               EQU   X'04'  Read permission
S_IWOTH               EQU   X'02'  Write permission
S_IXOTH               EQU   X'01'  Search (if a directory) or
*                                  execute (otherwise) permission
S_MODE#LENGTH         EQU   *-S_MODE Length this structure
** BPXYMODE End
```

BPXYMSG — Map interprocess communication message queues

DSECT (MSGBUF) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. Default for the message size is 100 bytes. Specify VARLEN= to override this value.

AMODE 64 callers use "BPXYMSG — Map interprocess communication message queues" on page 1106.

```
            BPXYMSG    ,
** BPXYMSG: Interprocess Communication Message Queue Structure
**  Used By: msgctl
MSQID_DS              DSECT ,      message queue structure
MSG_PERM              DS    CL(IPC#LENGTH) Mapped by BPXYIPCP
MSG_QNUM              DS    F      # of messages on queue
MSG_QBYTES            DS    F      max bytes allowed on queue
MSG_LSPID             DS    F      process ID of last msgsnd()
MSG_LRPID             DS    F      process ID of last msgrcv()
MSG_STIME             DS    F      time of last msgsnd()
MSG_RTIME             DS    F      time of last msgrcv()
MSG_CTIME             DS    F      time of last change  get/ctl
MSQ#LENGTH            EQU *-MSQID_DS Length of this DSECT
MSGBUF                DSECT ,      Message buffer - msgsnd, msgrcv
MSG_TYPE              DS    F      Message type
MSG_MTEXT             DS    CL100  Message text
MSGB#LENGTH           EQU *-MSGBUF Length of this DSECT
MSGXBUF               DSECT ,      Message buffer - msgxrcv
MSGX_MTIME            DS    F      time message sent
MSGX_UID              DS    F      sender's effective UID
MSGX_GID              DS    F      sender's effective GID
MSGX_PID              DS    F      sender's PID
MSGX_TYPE             DS    F      Message type
MSGX_MTEXT            DS    CL100  Message text
MSGX#LENGTH           EQU *-MSGXBUF Length of this DSECT
* Flag bits - msgrcv (also IPC_NOWAIT
MSG_NOERROR           EQU   4      No error if big message.
MSG_INFO              EQU   8      Use MSGXBUF not MSGBUF format
** BPXYMSG End
```

BPXYMSGF — Map the message flags

BPXYMSGF is used by send(), recv(), sendto(), recvfrom(), sendmsg() and recvmsg()

```
** BPXYMSGF: Socket MSG_* flags
**  Used By: SND RCV STO RFM SMS RMS SRX AIO
AIF   ('&DSECT' EQ 'NO').B411
MSG_FLAGS             DSECT ,
AGO   .C411
.B411    ANOP  ,
MSG_FLAGS             DS    0F
.C411    ANOP  ,
MSG_FLAGS1            DS    B      I_flags - byte 1
MSGFHIGH              EQU   X'80'  DO NOT USE THIS BIT! *
*                                  MSG_FLAGS must never be < 0
*
MSG_ACK_GEN           EQU   X'40'  Generate a UDP 'ACK packet' *
*                                  automatically to the originator if
*                                  an incoming UDP packet arrives
*                This flag is no longer supported by z/OS TCPIP.
*
MSG_ACK_TIMEOUT       EQU   X'20'  The caller expects an incoming UDP *
                                   packet within the "standard ACK
*                                  time interval".  Return to caller
```

```
*                                with an EINTR return code if no
*                                incoming UDP packet arrives
*                                within this time interval.
*               This flag is no longer supported by z/OS TCPIP.
*
MSG_ACK_EXPECTED     EQU   X'10'  (Used along with MSG_ACK_TIMEOUT)
*                                The incoming packet is expected to
*                                be an ACK. If the ACK arrives,
*                                the caller does not need to be
*                                activated to process it.
*                                Instead, the protocol will just
*                                cancel the timeout and let the
*                                application wait for the real data
*                                to arrive.
*               This flag is no longer supported by z/OS TCPIP.
SPACE ,
MSG_FLAGS2           DS    B
MSG_flags - byte 2
*
SPACE ,
MSG_FLAGS3           DS    B      MSG_flags - byte 3
MSG_EOF              EQU   X'80'  Close socket after the send.
*             On the send function only - Requests that the socket
*             be closed after all data has been transmitted.
*
SPACE ,
MSG_FLAGS4           DS    B      MSG_flags - byte 4

MSG_CONNTERM         EQU   X'80'  Complete when connection ends.
*             Requests that a receive-type function completes only
*             when a TCP socket connection is terminated.
*             - The buffer length specified on the operation must be 0,
*             thus there is no data associated with this receive and
*             the other MSG_FLAGS may not be used.  This operation is
*             only for connection termination notification.
*             - Any other normal outstanding receive-type requests will
*             also be completed at connection termination and these
*             completions may be running in parallel with that of
*             the MSG_CONNTERM request.
*
MSG_WAITALL          EQU   X'40'  Wait until all data returned
*             Requests that a receive-type function block until the
*             full amount of data requested can be returned.
*             The function may return a smaller amount of data if a
*             signal is caught, the connection is terminated, an error
*             is pending or SO_RCVTIMEO is set and the timer expires.
*
MSG_CTRUNC           EQU   X'20'  Control data was truncated.
*             An output flag for recvmsg, returned in the MsgHFlags
*             field of the MSGH structure.
MSG_TRUNC            EQU   X'10'  Normal data was truncated.
*             An output flag for recvmsg, returned in the MsgHFlags
*             field of the MSGH structure.
*
MSG_EOR              EQU   X'08'  Terminate a record.            @D2A
*             Only supported for FRCA enabled TCP sockets.
*
MSG_DONTROUTE        EQU   X'04'  Send without network routing.
*             Bypass normal routing for one send-type function request.
*             Usually used only by diagnostic or routing programs.
*
MSG_PEEK             EQU   X'02'  Peek at incoming data.
*             The data is returned but not consumed, so that a
*             subsequent receive-type function will see the same data.
*
MSG_OOB              EQU   X'01'  Send/Receive out of band data.
*             Sends out-of-band data for send, sendto or sendmsg.
```

```
*             Requests out-of-band data for recv, recvfrom or recvmsg.
*             Also, is an output flag for recvmsg indicating that
*             out-of-band data was received.  This is returned
*             in the MsgHFlags field of the MSGH structure,
MSG#LENGTH          EQU   *-MSG_FLAGS Length of this structure
** BPXYMSGF End
```

BPXYMSGH — Map the message header

BPXYMSGH is used by the sendmsg and recvmsg syscalls. AMODE 64 callers use "BPXYMSGH — Map the message header" on page 1107.

```
            BPXYMSGH   ,
** BPXYMSGH: MSGH system call structure
**  Used By: SendMsg / RecvMsg
MSGH                DSECT ,
MSGHBEGIN           DS    0D
* ------------- 32-Bit Version
MSGHNAMEPTR         DS    A(0)   Pointer to a structure that contains
*                                 the recipient's address.
MSGHNAMELEN         DS    F'0'   Buffer length.
MSGHIOVPTR          DS    A(0)   Pointer to an array of IOVEC buffers.
MSGHIOVNUM          DS    F'0'   Number of elements in IOVEC array.
MSGHCONTROLPTR      DS    0AL4   Pointer to ancillary data buffer
MSGHACCRIGHTSPTR    DS    A(0)   Pointer to access rights buffer.
MSGHCONTROLLEN      DS    0FL4   Length of ancillary data buffer
MSGHACCRIGHTSLEN    DS    F'0'   Access rights buffer length.
MSGHFLAGS           DS    F'0'   Output flags on received message
*
*   Constants
*
MSGH#LENGTH         EQU   *-MSGH   Length of MsgH
*
CMSGPTR             DS    A(0)   CMsg pointer
*
CMSGHDR             DSECT ,
CMSGLEN             DS    F'0'   Length, including header
CMSGLEVEL           DS    F'0'   Level
CMSGTYPE            DS    F'0'   Type
CMSGDATA            DS    0C     Data
*
*   Constants
*
SCM_RIGHTS          EQU   1        Access Rights
SCM_SECINFO         EQU   16386    Security Information
*
** BPXYMSGH End
```

BPXYMSGX — Map the message header

BPXYMSGX is used by the srx_np() syscall. BPXYMSGX uses constants defined by mapping macro IVTBUFL.

```
            IVTBUFL    ,
IVTBUFL  DSECT                      BUFFER DESCRIPTOR
BUFL_VERSION   DS    X              VERSION OF BUFFER DESCRIPTOR
BUFL_VERSIONC  EQU   X'00'          VERSION 0
BUFL_SOURCE    DS    X              BUFFER SOURCE
BUFL_CECSA     EQU   X'80'          INDICATES THAT THE STORAGE
*                                   IS IN CSM ECSA
BUFL_CDSPACE   EQU   X'40'          INDICATES THAT THE STORAGE
*                                   IS IN CSM DATA SPACE
BUFL_UDSPACE   EQU   X'20'          INDICATES THAT THE STORAGE
*                                   IS IN A USER DATA SPACE
```

```
BUFL_USTOR     EQU   X'10'             INDICATES THAT THE STORAGE
*                                      IS A USER'S STORAGE OTHER THAN
*                                      A DATA SPACE
BUFL_TYPE      DS    X                 BUFFER TYPE
BUFL_FIXED     EQU   X'80'             INDICATES THAT THE STORAGE IS
*                                      IN A GUARANTEED TO BE FIXED
*                                      STATE
BUFL_PAGEABLE  EQU   X'40'             INDICATES THAT THE STORAGE IS
*                                      IN A GUARANTEED TO BE PAGEABLE
*                                      STATE
BUFL_PAGEELIG  EQU   X'20'             INDICATES THAT THE STORAGE IS
*                                      ELIGIBLE TO BE PAGEFREED BY
*                                      CSM
               DS    XL1               RESERVED
BUFL_TOKEN     DS    XL12              CSM BUFFER TOKEN
BUFL_ALET      DS    F                 DATA SPACE ALET
BUFL_ADDR      DS    A                 POINTER TO BUFFER
BUFL_SIZE      DS    F                 THE SIZE OF THE ALLOCATED BUFFER
*                                      ON GET_BUFFER REQUESTS, THE DATA
*                                      LENGTH ON COPY_DATA REQUESTS
BUFL_END       DS    0F                END OF IVTBUFL
           BPXYMSGX   ,
** BPXYMSGX: MSGX system call structure
**  Used By: BPX1SRX
MSGX                 DSECT ,
*
MSGXNAMEPTR          DS    A        /* PTR TO SOCKADDR BUFFER       */
MSGXNAMELEN          DS    F        /* LENGTH OF SOCKADDR BUFFER    */
MSGXFLAGS            DS    0BL4     /* SRX CONTROL FLAGS            */
MSGXFLAGS1           DS    BL1
MSGXFLAGS2           DS    BL1
MSGXFLAGS3           DS    BL1
MSGXFLAGS4           DS    BL1      FLAGS ARE IN THE 4TH BYTE
MSGX_CECSA           EQU   X'02'    /*   RECV IN ECSA BUFFERS       */
MSGX_CDSPACE         EQU   X'01'    /*   RECV IN DATA SPACE BUFFERS */
MSGXMSGFLAGS         DS    BL4      /* MSG_* FLAGS, SEE BPXYMSGF    */
MSGXDATALEN          DS    F        /* MAX/MIN DATA TO RECEIVE      */
MSGXTCB              DS    A        /* TCB TO OWN RECEIVE BUFFERS   */
MSGXERRIOVX          DS    F        /* SEND IOVX ELEMENT IN ERROR   */
MSGXERRDATA          DS    F        /* AMOUNT SENT FROM LAST BUFFER */
MSGXIVTBUFLOFFSET    DS    F        /*1ST BUFF APPL STILL OWNS
                     DS    CL4      RESERVED
MSGXCONTROLPTR       DS    A          Ancillary Data buffer
MSGXCONTROLLEN       DS    F          Length of ancillary data
MSGXIOVX             DS    CL(BUFL_END-IVTBUFL) IVTBUFL FOR IOVX ARRAY
MSGXEND              EQU   *
*
MSGX#LEN             EQU   MSGXEND-MSGX
*
*  IOVX - ARRAY OF IVTBUFL BUFFER DESCRIPTIONS
*
IOVX                 DSECT ,        /* DESCRIBED BY MSGXIOVX        */
IOVXBUFL             DS    0CL(BUFL_END-IVTBUFL)       ARRAY ELEMENT
*
*   Constants
*                                    /* BPX1SRX DIRECTION PARAMETER: */
MSGX_SEND            EQU  0          /*   SEND OPERATION             */
MSGX_RECV            EQU  1          /*   RECEIVE OPERATION          */
*
** BPXYMSGX End
```

BPXYMTM — Map the modes for mount and unmount

```
           BPXYMTM    ,
** BPXYMTM:  File system mount/unmount modes
**  Used By: MNT UMT
```

```
MTM                DSECT ,
MTM1               DS    B     Flag byte 1
MTMRO              EQU   X'80' Mount file set read-only
MTMRDWR            EQU   X'40' Mount file set read/write
MTMDDNAME          EQU   X'20' FileSet is a DDName
MTMUMOUNT          EQU   X'10' This is a normal unmount request.
*                              If no one is using any of the files
*                              in the named filesystem, the unmount
*                              will be done.  Otherwise, the request
*                              will be rejected.
MTMIMMED           EQU   X'08' This is an unmount immediate request.
*                              The filesystem will be unmounted
*                              immediately, forcing any users of any
*                              files in the named filesystem to fail.
*                              All data changes that were made up to
*                              the time of the request will be saved.
*                              If there is a problem saving the data,
*                              the unmount request will fail.
MTMFORCE           EQU   X'04' This is an unmount force request.
*                              The filesystem will be unmounted
*                              immediately, forcing any users of any
*                              files in the named filesystem to fail.
*                              All data changes that were made up to
*                              the time of the request will be saved.
*                              If there is a problem saving the data,
*                              the request will continue and data may
*                              be lost.  Since data may be lost,
*                              before a forced request will be
*                              allowed, a previous immediate unmount
*                              request must have been attempted, or
*                              the request will be rejected.
MTMDRAIN           EQU   X'02' This is an unmount drain request.
*                              The requestor is willing to wait for
*                              all uses of this filesystem to be
*                              normally terminated and the
*                              filesystem to be unmounted.
MTMRESET           EQU   X'01' This is a reset unmount request.  This
*                              will allow a previous unmount drain
*                              request to be stopped.
MTM2               DS    B     Flag byte 2
MTM2RES80          EQU   X'80' Must not be used
MTM2RES40          EQU   X'40' Must not be used
MTMTERMUNMOUNT     EQU   X'20' Unmount from PFS term
MTMSAMEMODE        EQU   X'10' Remount in same mode
MTMMNTINCOMP       EQU   X'08' Mount is incomplete
MTMUNQSEFORCE      EQU   X'04' Force this unquiesce request, even
*                              if the requester process is not
*                              the process that made the quiesce
*                              request.
MTM2RES02          EQU   X'02' Must not be used
MTM2RES01          EQU   X'01' Must not be used
MTM3               DS    B     Flag byte 3 - reserved
MTM3RES80          EQU   X'80' Must not be used
MTM3RES40          EQU   X'40' Must not be used
MTM3RES20          EQU   X'20' Must not be used
MTM3RES10          EQU   X'10' Must not be used
MTM3RES08          EQU   X'08' Must not be used
MTMNOSUID          EQU   X'04' Dont allow setuid
MTMSYNCHONLY       EQU   X'02' Mount must be completed
*                              synchronously.  That is, mount()
*                              must not return +1
MTMREMOUNT         EQU   X'01' Change attributes of mounted file
*                              system
MTM4               DS    B     Flag byte 4 - reserved
MTMNOSEC           EQU   X'80' NoSecurity option
MTM4RES40          EQU   X'40' Must not be used
MTM4RES20          EQU   X'20' Must not be used
```

```
MTMAMOVE             EQU   X'10' Automove option
*                                BPX2MNT/BPX4MNT use only
MTMAUNMOUNT          EQU   X'08' UnMount during recovery
*                                BPX2MNT/BPX4MNT use only
MTM4RES08            EQU   X'08' Must not be used
MTM4RES04            EQU   X'04' Must not be used
MTM4RES02            EQU   X'02' Must not be used
MTM#LENGTH           EQU   *-MTM Length of this structure
*
* MTM#MOUNTTEST defines the valid MTM bit settings for the BPX1MNT
* interface
MTM#MOUNTTEST        EQU   X'C0000680'
** BPXYMTM End
```

BPXYOCRT — Map the OE certificate support structure

AMODE 64 callers use "BPXYOCRT — Map the OE certificate support structure" on page 1107.

```
           BPXYOCRT   ,
** BPXYOCRT: OE Certificate support structure
**  Used By: TLS
OCRT       DSECT ,
OCRTTYPE   DS    F      type of certificate attached
OCRTUSERID DS    CL9    MVS userid, null terminated, input/output
           DS    CL3    reserved
OCRTCLEN   DS    F      length of certificate associated with type
OCRTCPTR   DS    A      31-Bit ptr to the actual certifcate
OCRT_LEN   EQU   *-OCRT
OCRT_X509  EQU   1      Certificate type X509
** BPXYOCRT End
```

BPXYOEXT — Map the common external control block

DSECT=NO is not allowed. The storage belongs to z/OS UNIX.

```
** BPXYOEXT: Common External Control Block
*
*     The address of BPXYOEXT control block can be obtained as follows:
*
*        L        14,16(0,0)      GET CVT ADDRESS
*        L        15,140(0,14)    GET ECVT ADDRESS
*        L        14,244(0,15)    GET BPXYOEXT ADDRESS
*
*
*    OEXT user exit support:
*
*      When the kernel detects that the OEXTUSEREXIT address is
*      non-zero, control will be given to this exit on:
*
*         1) Successful completion of the GETPWNAME service and
*            specified name matches invoking userid.
*
*               Parm 1 = 4 byte function code set to OEXT#UEGETPWNAME
*               Parm 2 = 4 byte length of Current Working Directory
*               Parm 3 = N byte Current Working Directory
*
*            Purpose of call is to allow exit to examine/change CWD.
*            Length of CWD must remain the same.
*
*      User Exit will be given control in supervisor state key zero.
*
*      Input:                                  ---------------
*           Register 1 = Parmlist address ---> | Parm 1 addr |
*                                              ---------------
*                                              | Parm 2 addr |
```

```
*                                          ---------------
*                                                .
*                                                .
*                                  Bit 31 on  ---------------
*                                  in last -->| Parm N addr |
*                                  parm       ---------------
*
*
*           Register 13 = Save area address
*           Register 14 = Return address
*
*      Output:
*            Possible modification of CWD, length must not change.
*
*       Exit will be passed a 64 bytes save area in Register 13.  It is
*       the responsibility of the User Exit to save all resisters upon
*       entry and to restore all registers before return.
*OEXT                DSECT ,
OEXT                DS    0D
OEXTID              DS    CL4     Eye catcher
OEXTSP              DS    FL1     Subpool of this structure
OEXTLEN             DS    FL3     Length this structure
OEXTFLGS            DS    0CL4
OEXTFLG1            DS    B
OEXTF1DEFSEG        EQU   X'80'   OMVS default segment exists
*                   EQU   X'40'   Reserved for future use
OEXTSYSPLEXACTV     EQU   X'20'   OMVS is a member of sysplex
OEXTNOACCT          EQU   X'10'   Don't allow user to change acct-info
*
OEXTSETIP@          EQU   X'08'   Deamons should set IP@ of client
*
OEXTGRPIDVALID      EQU   X'04'   If on, OextTtyGrpId is valid
OEXTSERVICESAVAILABLE EQU X'02'   IF ON, z/OS UNIX Services Available
OEXTFLG2            DS    B
OEXTCINETCONF       EQU   X'80'   Cinet is configured
OEXTIPV6            EQU   X'40'   IPv6 Configured & Supported
OEXTIPV6CONF        EQU   X'20'   IPv6 is Configured
OEXTFLG3            DS    B
OEXTFLG4            DS    B
OEXTSTARTM          DS    BL.64   Time z/OS UNIX was started
*                                 (TOD format)
OEXTSMFP            DS    A       Pointer to the SMF Process accounting
*                                 data collection routine.  This offset
*                                 must not change.  It is hardcoded in
*                                 executable macro BPXESMF.
OEXTLVL             DS    0XL4    z/OS UNIX Release level
*                                 indicator
*                                 (Multiple bits may be set, e.g.
*                                 OEXT1120 and OEXT1130 will both
*                                 be set)
OEXTLV0             DS    XL1     Byte 0 of OEXTLVL
*                   EQU   X'80'   Reserved
OEXT1120            EQU   X'40'   HOM1120 functions are present
OEXT1130            EQU   X'20'   HOM1130 functions are present
OEXT1130IPC         EQU   X'10'   HOM1130 functions InterProcess
*                                 Communication (IPC), NFSC, directory
*                                 sticky bit, lchown UID=-1, are
*                                 present.
OEXT1140            EQU   X'08'   HOM1140 functions are present
OEXT1150            EQU   X'04'   HBB6603 functions are present. Future
*                                 z/OS UNIX release indicators will
*                                 not be added since z/OS UNIX is
*                                 now an integral part of the BCP.
OEXTTECFS           EQU   X'02'   The Thli ECB comunication function is
*                                 supported
OEXTTAFS            EQU   X'01'   The MvsThreadAffinity function is
*                                 supported
```

```
OEXTLV1             DS    XL1    Byte 1 of OEXTLVL
OEXTPQG             EQU   X'80'  The pthread_quiesce_and_get_np
*                                function (BPX1PQG) is supported
OEXTMMAPEXTKEY      EQU   X'40'  The mmap extended key function
*                                is supported
                    DS    XL2    Bytes 2-3 of OEXTLVL
OEXTAPPCBPXEXIT     DS    A      Pointer to the z/OS UNIX Exit
*                                for APPC Processing.
OEXTTIXP            DS    A      Pointer to z/OS UNIX timer exit.
*                                Invoked by IEAVLEXT.
OEXTRUNOPTSPTR      DS    A      Pointer to the RUNOPTS() string
*                                specified at IPL time.
OEXTRUNOPTSLEN      DS    FL4    Length of the RUNOPTS() string
*                                specified at IPL time.
OEXTBPXWLMEXIT      DS    A      Pointer to z/OS UNIX Exit for WLM
*                                IWMUWON timeout processing
OEXTUSEREXIT        DS    A      Pointer to z/OS UNiX User Exit
OEXTMSGACBRTN       DS    A      Pointer to JESYSMSG ACB PUT rtn
OEXTPARMLIBPTR      DS    A      Pointer to Parmlib Data Area
OEXTARG_MAX         DS    FL4    Sysconf ARG_MAX value
OEXTCLK_TCK         DS    FL4    Sysconf CLK_TCK value
OEXTTZNAME_MAX      DS    FL4    Sysconf TZNAME_MAX value
OEXTJOB_CONTROL     DS    FL4    Sysconf JOB_CONTROL value
OEXTVERSION         DS    FL4    Sysconf VERSION value
OEXT2_CHAR_TERM     DS    FL4    Sysconf CHAR_TERM value
OEXTTTYGRPID        DS    FL4    sysconf TTY_GROUP value
                    DS    1CL0008   Reserved
                    DS    0D     Ensure end on double word boundary
OEXT#LENGTH         EQU   *-OEXT Length of this structure
OEXT#UEGETPWNAME    EQU   1      Function code indicating user exit
*                                called from getpwname
** BPXYOEXT End
```

BPXYOPNF — Map flag values for open

```
           BPXYOPNF   ,
** BPXYOPNF: File status flags
**  Used By: FCT OPN
O_FLAGS             DSECT ,
O_FLAGS1            DS    B      Open flags - byte 1
OPNFHIGH            EQU   X'80'  DO NOT USE THIS BIT!
*                                O_FLAGS must never be < 0
                    SPACE ,
O_FLAGS2            DS    B      Open flags - byte 2
OPNFEXEC            EQU   X'80'  Execute access requested -
*                                authorization required for use
                    SPACE ,
O_FLAGS3            DS    B      Open flags - byte 3
O_NOLARGEFILE       EQU   X'08'  Large Files not allowed          @D5A
O_LARGEFILE         EQU   X'04'  Ignored                          @D5A
O_ASYNCSIG          EQU   X'02'  An asynchronous signal may occur @D4A
O_SYNC              EQU   X'01'  Force synchronous updates    @D3A
                    SPACE ,
O_FLAGS4            DS    B      Open flags - byte 4
O_CREXCL            EQU   X'C0'  Create file only if non-existent
O_CREAT             EQU   X'80'  Create file
O_EXCL              EQU   X'40'  Exclusive flag
O_NOCTTY            EQU   X'20'  Not a controlling terminal
O_TRUNC             EQU   X'10'  Truncate flag
O_APPEND            EQU   X'08'  Set offset to EOF on write
O_NONBLOCK          EQU   X'04'  Don't block this file
FNDELAY             EQU   X'04'  Don't block this file        @D2A
O_RDWR              EQU   X'03'  Open for Read and Write
O_RDONLY            EQU   X'02'  Open for Read Only
O_WRONLY            EQU   X'01'  Open for Write Only
O_ACCMODE           EQU   X'03'  Mask for file access modes
```

```
O_GETFL             EQU   X'0F'  Mask for file access modes and
*                                file status flags together   @P4A
                    SPACE ,
OPNF#LENGTH         EQU   *-O_FLAGS Length of this structure
** BPXYOPNF End
```

BPXYPCF — Command values for pathconf and pathconf

BPXYPCF is composed only of EQUates. DSECT= is allowed but ignored.

```
           BPXYPCF    ,
** BPXYPCF:  Command values
**  Used By: FPC PCF
PC_CHOWN_RESTRICTED  EQU 1        _POSIX_CHOWN_RESTRICTED option
PC_LINK_MAX          EQU 2        LINK_MAX option
PC_MAX_CANON         EQU 3        _POSIX_MAX_CANON option
PC_MAX_INPUT         EQU 4        _POSIX_MAX_INPUT option
PC_NAME_MAX          EQU 5        NAME_MAX option
PC_NO_TRUNC          EQU 6        _POSIX_NO_TRUNC option
PC_PATH_MAX          EQU 7        PATH_MAX option
PC_PIPE_BUF          EQU 8        PIPE_BUF option
PC_VDISABLE          EQU 9        _POSIX_VDISABLE option
PC_ACL               EQU 10       _PC_ACL option
PC_ACL_ENTRIES_MAX   EQU 11       _PC_ACL_ENTRIES_MAX
PC_CASE              EQU 100      Case Flags
*
PCFGMAX              EQU 11       Max _POSIX_ value
*
*  Pathconf Case Flags - vn_pathconf(PC_CASE) returned value
PCCASEFLAGS          DSECT ,
                     DS    XL3
PCCASEFLAGSBYTE      DS    XL1
PCCASEINSENSITIVE    EQU   X'02'          0=SENSITIVE,1=NOT
PCCASENONPRESERVING  EQU   X'01'          0=PERSERVING,1=NOT
*
*  Pathconf File Group - for v_pathconf(BPX1VPC)
PCFG                 DSECT ,
PCFGLINKMAX          DS    F              LINK_MAX
PCFGNAMEMAX          DS    F              NAME_MAX
PCFGPCFLAGS          DS    XL1            FLAGS:
PCFGNOTRUNC           EQU   X'80'            POSIX_NO_TRUNC
PCFGCHOWNRSTD         EQU   X'40'            CHOWN RESTRICTED
PCFGCASEINSENSITIVE   EQU   X'20'            0=SENSITIVE,1=NOT
PCFGCASENONPRESERVING EQU   X'10'            0=PERSERVING,1=NOT
PCFGSECACL            EQU   X'08'            0=ACLSUPPORT,1=NONE
                     DS    XL3
*
** BPXYPCF End
```

BPXYPEDB — Mapping of process exit data block

```
           BPXYPEDB    ,
PEDB     DSECT           PEDB - Process Exit Data Block
PEDBEYE  DS    1CL0004   Eye catcher - 'PEDB'
PEDBLENGTH DS  1FL2      Length of structure
PEDBVERSION DS 1FL1      Version number
PEDBEXITPOINTID DS 1FL1  Unique value identifying exit point, these    X
                         constants are defined below
PEDBFLAGS DS   0FL4      Flags
PEDBCREATEDVIAFLAGS DS 0CL0001 Bits indicating what the process is     X
                         being created via
PEDBVIAFORK EQU X'80'    On = process is being created via fork()
PEDBVIASPAWN EQU X'40'   On = process is being created via spawn()
PEDBVIAATTEXEC EQU X'20' On = process is being created via             X
```

```
                        attach_exec()
PEDBVIAATTEXECMVS EQU X'10' On = process is being created via          X
                        attach_execmvs()
PEDBVIA1STCALLABLE EQU X'08' On = process is being created via the 1st X
                        callable service from a non-z/OS UNIX address space
         ORG   PEDBCREATEDVIAFLAGS+X'00000001'
PEDBFLAGS2 DS  0CL0001   2nd flag byte
PEDBVIAMEMTERM EQU X'80' On = process is being terminated via memterm
PEDBVIAABTERM EQU X'40'  On = process is being terminated via abterm
         ORG   PEDBFLAGS2+X'00000001'
PEDBFLAGS3 DS  1CL0001   3rd flag byte
PEDBFLAGS4 DS  1CL0001   4th flag byte
PEDBUNIQUEID DS 1BL8     A Unique Id identifying this process's set of X
                        exits. This Id is the same starting at the    X
                        pre-process initialization exit all the way   X
                        to the pre-process term exit. It also happens X
                        to be TOD when the pre-process initial-       X
                        zation exit was called.
*
*      ************************************************************
*      *                                                          *
*      * Information specific to Initiator of the new process     *
*      * (creator) This section is filled out ONLY when the following*
*      * exits hit: BPX_PREPROC_INIT - pre-process initialization  *
*      * BPX_POSPROC_INIT - post process initialization           *
*      * BPX_IMAGE_INIT - process image change This section is NOT *
*      * filled out by the following exits: BPX_PREPROC_TERM -     *
*      * pre-process termination                                  *
*      *                                                          *
*      ************************************************************
*
*
PEDBCREATORINFO DS 0CL0164
PEDBCREATORPROCID DS 1FL4 Process ID initiating New process
PEDBCREATORASID DS 1FL2  ASID of initiating new process
PEDBCREATORUSERIDLEN DS 1FL1 Length of the Userid initiating the new   X
                        process
PEDBCREATORALIASLEN DS 1FL1 Length of the Alias initiating the new     X
                        process
PEDBCREATORPROGNAMELEN DS 1FL2 Length of the Program Name initiating   X
                        new process
         DS    1FL2      Reserved
PEDBCREATORJOBNAME DS 1CL0008 Jobname initiating the new process
PEDBCREATORUSERID DS 1CL0008 Userid initiating the new process
PEDBCREATORALIAS DS 1CL0008 Alias initiating the new process
PEDBCREATORPROGNAME DS 1CL0128 Program Name of the initiating new      X
                        process
*
*      ************************************************************
*      *                                                          *
*      * New Process / Terminating Process Information (child) This  *
*      * section is filled out ONLY when the following exits hit:  *
*      * BPX_POSPROC_INIT - post process initialization           *
*      * BPX_IMAGE_INIT - process image change BPX_PREPROC_TERM -  *
*      * pre-process termination This section is NOT filled out by  *
*      * the following exits: BPX_PREPROC_INIT - pre-process       *
*      * initialization                                           *
*      *                                                          *
*      ************************************************************
*
*
PEDBNEWINFO DS 0CL0164
PEDBTERMINFO DS 0CL0164
PEDBNEWPROCID DS 0FL4    Process ID of New process
PEDBTERMPROCID DS 1FL4   Process ID for the terminating process
PEDBNEWASID DS 0FL2      ASID of new process
PEDBTERMASID DS 1FL2     ASID of the terminating process
```

```
PEDBNEWUSERIDLEN DS 0FL1 Length of the Userid of the new process
PEDBTERMUSERIDLEN DS 1FL1 Length of the Userid of the terminating      X
                         process
PEDBNEWALIASLEN DS 0FL1  Length of the Alias of the new process
PEDBTERMALIASLEN DS 1FL1 Length of the Alias of the terminating        X
                         process
PEDBNEWPROGNAMELEN DS 0FL2 Length of the Program Name of the new       X
                         process
PEDBTERMPROGNAMELEN DS 1FL2 Length of Program Name of the terminating  X
                         process
         DS    1FL2      Reserved
PEDBNEWJOBNAME DS 0CL0008 Jobname of new process
PEDBTERMJOBNAME DS 1CL0008 Jobname of terminating process
PEDBNEWUSERID DS 0CL0008 Userid of the new process
PEDBTERMUSERID DS 1CL0008 Userid of the terminating process
PEDBNEWALIAS DS 0CL0008  Alias of the new process
PEDBTERMALIAS DS 1CL0008 Alias of the terminating process
PEDBNEWPROGNAME DS 0CL0128 Program Name of the new process
PEDBTERMPROGNAME DS 1CL0128 Program Name of the terminating process
*
*      ****************************************************************
*      *                                                              *
*      *                                                              *
*      ****************************************************************
*
*
         DS    1CL0064   Reserved for future use
PEDBVER1LEN DS 0C        End of Version 1
PEDB#ID  EQU   C'PEDB'   Eye catcher
PEDB#VER EQU   1         Current version of this control block
PEDB#VER01 EQU 1         Version 1 of control block
PEDB#LEN01 EQU 412       Version 1 of PEDB control block len
PEDB#LEN EQU   412       Length of PEDB
*
*    Constants to fill in PEDBExitPointId field
*
*
PEDB_BPX_PREPROC_INIT EQU 1 Identifies that this this structure was    X
                         built for the pre-process initiation exit
PEDB_BPX_POSPROC_INIT EQU 2 Identifies that this this structure was    X
                         built for the post process initiation exit
PEDB_BPX_IMAGE_INIT EQU 3 Identifies that this this structure was      X
                         built for the process image change exit
PEDB_BPX_PREPROC_TERM EQU 4 Identifies that this this structure was    X
                         built for the pre-process termination
PEDB_LEN EQU   *-PEDB
```

BPXYPGPS — Map the response structure for w_getpsent

VARLEN accepts three operands. Operands omitted (like the first) default to the maximum needed. Use zero if the associated field is not needed.

VARLEN describes the number of bytes to map the following:

1. Controlling TTY name and its length
2. Pathname and its length
3. Command and its length

```
           BPXYPGPS   VARLEN=(1028,1028,1028)
** BPXYPGPS: w_getpsent return data structure
**  Used By: GPS
PGPS                DSECT ,
PGPSSTATUS0         DS    B         MVS status
PGPSSWAP            EQU   X'80'       Swapped out
*                   EQU   X'7F'       Not Used
```

```
PGPSSTATUS1          DS    B         Process status
PGPSSTOPPED          EQU   X'80'       Stopped process
PGPSTRACE            EQU   X'40'       PTrace active
PGPSMULTHREAD        EQU   X'20'        0=One open task in process
PGPSPTHREAD          EQU   X'10'        0=No pthread task in process
PGPSMULPROCESS       EQU   X'08'        0=One process in addr space
*                    EQU   X'07'       Not Used
PGPSSTATUS2          DS    B         System Call Status
PGPSLENERR           EQU   X'80'       PGPSLENGTH conflict
*                    EQU   X'7F'       Not Used
PGPSSTATUS3          DS    CL1       State of reported task - with
*                          PGPSPTHREAD=0 the most recent created thread
*                          PGPSPTHREAD=1 the initial pthread task (IPT)
PGPSMSGRCV           EQU   C'A'      IPC MSGRCV WAIT
PGPSMSGSND           EQU   C'B'      IPC MSGSND WAIT
PGPSWAITC            EQU   C'C'      COMM KERNELWAIT
PGPSSEMOP            EQU   C'D'      IPC SEMOP WAIT
PGPSFREEZE           EQU   C'E'      QUIESCEFREEZE
PGPSWAITF            EQU   C'F'      F S  KERNEL WAIT
PGPSMVSPAUSE         EQU   C'G'      MVSPAUSE
PGPSZOMBIE2          EQU   C'L'      PROCESS TERMINATED AND STILL
*                                    SESSION OR PROCESS GROUP LEADER
PGPSWAITO            EQU   C'K'      OTHER KERNEL WAIT
PGPSQUIESCET         EQU   C'Q'      QUIESCE TEMRINATION WAIT
PGPSRUN              EQU   C'R'      NOT KERNEL WAIT
PGPSSLEEP            EQU   C'S'      SLEEP() ISSUED
PGPSCHILD            EQU   C'W'      WAITING FOR CHILD
PGPSFORK             EQU   C'X'      FORK NEW PROCESS
PGPSZOMBIE           EQU   C'Z'      PROCESS TERMINATED AND PARENT
*                                    HAS NOT ISSUED WAIT SYSCALL
PGPSPID              DS    F         Process ID
PGPSPPID             DS    F         Parent ID
PGPSSID              DS    F         Session ID (leader)
PGPSPGPID            DS    F         Process Group
PGPSFGPID            DS    F         Foreground Process Group
PGPSEUID             DS    F         Effective User ID
PGPSRUID             DS    F         Real User ID
PGPSSUID             DS    F         Saved Set User ID
PGPSEGID             DS    F         Effective Group ID
PGPSRGID             DS    F         Real Group ID
PGPSSGID             DS    F         Saved Set Group ID
PGPSTSIZE            DS    F         Total size
PGPSSTARTTIME        DS    F         Starting time, GMT since EPOCH
PGPSUSERTIME         DS    F         User CPU time (clock_t)
PGPSSYSTIME          DS    F         System CPU time (clock_t)
PGPSCONTTYBLEN       DC    A(1028)          L'PGPSCONTTYBUF
PGPSCONTTYPTR        DC    A(PGPSCONTTYBUF) ->PGPSCONTTYBUF
PGPSPATHBLEN         DC    A(1028)          L'PGPSPATHBUF
PGPSPATHPTR          DC    A(PGPSPATHBUF)   ->PGPSPATHBUF
PGPSCMDBLEN          DC    A(1028)          L'PGPSCMDBUF
PGPSCMDPTR           DC    A(PGPSCMDBUF)    ->PGPSCMDBUF
PGPSSERVERTYPE       DS    F         Server type (FILE=1, LOCK=2)
PGPSSERVERNAME       DS    CL32      Name supplied on registration
PGPSMAXVNODETOKENS   DS    F         Max number of VNode Toks allowed
PGPSVNODETOKENCOUNT  DS    F         Current number of VNode Tokens
PGPSSERFLAGS         DS    F         Server flags
PGPSSYSCALLCOUNT     DS    F         Count of syscalls this process
PGPSJOBNAME          DS    CL8       AscbJBNI/JBNS  JobName
PGPSWAITTIME         DS    F         Since Kern Wait Started
PGPSASID             DS    FL2       Address space ID
PGPS#LENGTH          EQU   *-PGPS    Length of this structure
* Variable portion - Controlling terminal buffer
*
* Notes on format of controlling terminal string in PGPSCONTTYBUF
* 1. Controlling terminal string returned in PGPSCONTTY is
*    null-terminated.
* 2. The PGPSCONTTYLEN value does NOT include the terminating
```

```
*    null character.
PGPSCONTTYBUF        DS    0CL1028   ConTty Len+Buf
PGPSCONTTYLEN        DS    FL4       Length ConTty returned
PGPSCONTTY           DS    CL1024    ConTty (len+1-th char=null)
*
* Notes on format of path string in PGPSPATHBUF:
* 1. Pathname returned in PGPSPATH is null-terminated.
* 2. The PGPSPATHLEN value does NOT include the terminating null
*    character.
* 3. TSO (non-shell) pathnames may be padded with spaces to eight
*    characters.
PGPSPATHBUF          DS    0CL1028   Pathname Len+Buf
PGPSPATHLEN          DS    FL4       Length Pathname returned
PGPSPATH             DS    CL1024    Pathname (len+1-th char=null)
*
* Notes on format of PGPSCMDBUF:
* 1. PGPSCMD consists of one or more character fields representing
*    the command and its arguments (if any).
* 2. Each character field consists of a four byte length field and
*    a null-terminated character string.
* 3. TSO (non-shell) commands may be padded with spaces to eight
*    characters.
* 4. Unlike PGPSCONTTYLEN and PGPSPATHLEN, each character field
*    length value DOES include the null-terminating character.
* 5. The PGPSCMDLEN value is the sum of all character fields (length
*    fields and character strings).
PGPSCMDBUF           DS    0CL1028   Command Len+Buf
PGPSCMDLEN           DS    FL4       Length Command returned
PGPSCMD              DS    CL1024    Command (array of len, element)
PGPS#STORAGE         EQU   *-PGPS    Length, total area used
** BPXYPGPS End
```

BPXYPGTH — Map the __getthent input/output structure

```
        BPXYPGTH   ,
** BPXYPGTH: __getthent input and output structures
**  Used By: GTH
PGTHA                DSECT ,        I N P U T - - - - - - - - - - -
PGTHACONTINUE        DS    0CL14
PGTHAPID             DS    F         PROCESS ID (IGNORED IF FIRST)
PGTHATHID            DS    CL8       THREAD ID (IGNORED IF FIRST/LAST)
PGTHAACCESSPID       DS    FL1       FIRST, CURRENT, NEXT
PGTH#NEXT            EQU   2         NEXT AFTER SPECIFIED
PGTH#CURRENT         EQU   1         AS SPECIFIED
PGTH#FIRST           EQU   0         FIRST (EQUIV NEXT WITH PID=0)
PGTH#LAST            EQU   3         only with PGTHIACCESSTHID
*
PGTHAACCESSTHID      DS    FL1       FIRST, CURRENT, NEXT, LAST
* ONLY FLAG1 BITS THREAD AND PTAG WILL BE CONSIDERED WHEN
* ACCESSPID=CURRENT AND ACCESSTHID=NEXT
*
* ASID AND LOGINNAME FILTERS APPLY ONLY WHEN ACCESSPID = FIRST, NEXT
PGTHAASID            DS    FL2       FILTER - ASID
* LOGINNAME COMPARISON WILL LOOK FOR UNIX ALIAS. IF PGHTALOGINNAME
* IS NOT AN ALIAS, IT WILL BE SHIFTED TO UPPER CASE AND CHECKED
* AGAINST MVS ID.
PGTHALOGINNAME       DS    CL8       FILTER - USERID ALIAS OR MVS
*
PGTHAFLAG1           DS    FL1       WHAT OUTPUT AREAS TO INCLUDE
PGTHAPROCESS         EQU   X'80'       PGTHC, PROCESS DATA
PGTHACONTTY          EQU   X'40'       PGTHD, CONTTY
PGTHAPATH            EQU   X'20'       PGTHE, PATH
PGTHACOMMAND         EQU   X'10'       PGTHF, CMD & ARGS - UP TO
                                                          1024 BYTES
PGTHAFILEDATA        EQU   X'08'       PGTHG, FILE DATA
PGTHATHREAD          EQU   X'04'       PGTHJ, THREAD DATA
PGTHAPTAG            EQU   X'02'       PGTHK, PTAG (NEEDS PGTHJ)
```

```
PGTHACOMMANDLONG      EQU   X'01'      PGTHF, CMD & ARGS - UP TO
                                                      2048 BYTES
                      DS    FL1
PGTHA#LEN             EQU   *-PGTHA
*
PGTHB                 DSECT ,         O U T P U T - - - - - - - - - - -
PGTHBID               DS    CL4       "gthb"
PGTHBCONTINUE         DS    0CL14     NEXT VALUE FOR PGTHACONTINUE
PGTHBPID              DS    F         PROCESS ID
PGTHBTHID             DS    CL8       THREAD ID
PGTHBACCESSPID        DS    FL1       CURRENT/FIRST/NEXT
PGTHBACCESSTHID       DS    FL1       CURRENT/FIRST/NEXT/LAST
                      DS    FL2
PGTHBLENUSED          DS    F         LENGTH OF OUTPUT BUFFER USED
PGTHBLIMITC           DS    CL1       N, A
PGTHBOFFC             DS    FL3       OFFSET OF PROCESS AREA
PGTHBLIMITD           DS    CL1       N, A, X
PGTHBOFFD             DS    FL3       OFFSET OF CONTTY  AREA
PGTHBLIMITE           DS    CL1       N, A, X
PGTHBOFFE             DS    FL3       OFFSET OF PATH    AREA
PGTHBLIMITF           DS    CL1       N, A, X
PGTHBOFFF             DS    FL3       OFFSET OF COMMAND AREA
PGTHBLIMITG           DS    CL1       N, A, X
PGTHBOFFG             DS    FL3       OFFSET OF FILE DATA AREA
PGTHBLIMITJ           DS    CL1       N, A, V, X
PGTHBOFFJ             DS    FL3       OFFSET OF THREAD AREA
PGTHB#LEN             EQU   *-PGTHB
*
*****************************************************************************
*                                                                           *
* Values for PGTHBLIMITx fields and PgthGLimitH, PgthJLimitJ, and           *
*            PgthJLimitK fields                                             *
*                                                                           *
*         N - associated area was not requested to be filled in             *
*         A - the section was completely filled in                          *
*         S - the output buffer is not big enough for the requested         *
*             data.  The section has been filled in as much                 *
*             as possible.                                                  *
*         V - the section was started but could not be completed            *
*             due to a system error.  Data in this section can              *
*             not be trusted.                                               *
*         X - the requested data was not available                          *
*         0 - processing did not get far enough to fill out this            *
*             section of the buffer.  Most likely, a buffer full            *
*             condition occurred while filling out a previous section       *
*             and the service stops further processing and returns          *
*             EINVAL JrBuffTooSmall to the caller.                          *
*                                                                           *
*****************************************************************************

* VALUES FOR PGTH.LIMIT. FIELDS
PGTH#NOTREQUESTED     EQU   C'N'      Associated PghtA.. bit off
PGTH#OK               EQU   C'A'      All data included
PGTH#STORAGE          EQU   C'S'      output buffer exhausted
* EXHAUSTED STORAGE < 1ST PGTHJ RESULTS IN -1 EINVAL JRBUFFTOOSMALL
PGTH#VAGUE            EQU   C'V'      Changed out from under us
PGTH#NOTCONNECTED     EQU   C'X'      Need data not connected
*
* USING PGTHC,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFC
PGTHC                 DSECT ,         P R O C E S S - - - - - - - - -
PGTHCID               DS    CL4       "gthc"
PGTHCFLAG1            DS    FL1
PGTHCMULPROCESS       EQU   X'80'     MULTIPLE PROCESSES
PGTHCSWAP             EQU   X'40'     TCBOUT
PGTHCTRACE            EQU   X'20'     THREAD IS BEING TRACED
PGTHCSTOPPED          EQU   X'10'     STOPPED
PGTHCINCOMPLETE       EQU   X'08'     NOT ALL BLOCKS PRESENT
```

```
PGTHCZOMBIE           EQU   X'04'    PROCESS IS A ZOMBIE
PGTHCBLOCKING         EQU   X'02'    Shutdown blocking
PGTHCPERM             EQU   X'01'    SHutdown permanent
PGTHCFLAG2            DS    FL1
PGTHCMEMLTYPE         EQU   X'80'    ON - MemLimit is a BinMult
PGTHCRESPAWN          EQU   X'40'    respawnable process
PGTHCUSERTRACEACT     EQU   X'20'    User Syscall Trace Active
PGTHCFLAG3            DS    FL1
PGTHCMEMUTYPE         EQU   X'80'    ON - MemUsage is a BinMult
                      DS    1FL1
PGTHCPID              DS    F        PROCESS ID
PGTHCPPID             DS    F        PARENT ID
PGTHCPGPID            DS    F        PROCESS GROUP
PGTHCSID              DS    F        SESSION ID
PGTHCFGPID            DS    F        FOREGROUND PROCESS GROUP
PGTHCEUID             DS    F        EFFECTIVE USER ID
PGTHCRUID             DS    F        REAL USER ID
PGTHCSUID             DS    F        SAVED SET USER ID
PGTHCEGID             DS    F        EFFECTIVE GROUP ID
PGTHCRGID             DS    F        REAL GROUP ID
PGTHCSGID             DS    F        SAVED SET GROUP ID
PGTHCTSIZE            DS    F        TOTAL SIZE
PGTHCSYSCALLCOUNT     DS    F        COUNT OF SLOW-PATH SYSCALLS
PGTHCUSERTIME         DS    F        TIME SPENT IN USER CODE
PGTHCSYSTIME          DS    F        TIME SPENT IN SYSTEM CODE
PGTHCSTARTTIME        DS    F        TIME PROCESS WAS DUBBED
PGTHCCNTOE            DS    FL2      NO. OE THREADS
PGTHCCNTPTCREATED     DS    FL2      NO. PTHREAD CREATED THREADS
PGTHCCNTTHREADS       DS    FL2      COUNT OF ALL THREADS
PGTHCASID             DS    FL2      ADDRESS SPACE ID
PGTHCJOBNAME          DS    CL8      MVS JOB NAME
PGTHCLOGINNAME        DS    CL8      LOGIN NAME - ALIAS OR MVS
PGTHCMEMLIMIT         DS    1FL4     maximum Memlimit in bytes
        ORG    PGTHCMEMLIMIT
PGTHCMEMLIMITVAL      DS    1FL3     Hex value
PGTHCMEMLMULT         DS    1CL1     multiplier when PGTHCMEMLTYPE
PGTHCMEMUSAGE         DS    1FL4     bytes in use
        ORG    PGTHCMEMUSAGE
PGTHCMEMUSAGEVAL      DS    1FL3     Hex value
PGTHCMEMUMULT         DS    1CL1     multiplier when PGTHCMEMUTYPE
PGTHCX   DS    0C
*
*      ************************************************************
*      *
*      * PGTHCMEMLIMIT constants are used by PGTHCMEMLMULT and
*      * PGTHCMEMUMULT when the TYPE is a binmult.
*      *
*      * When PGTHCMEMLTYPE is on PGTHCMEMLIMIT consists or a
*      * 24bit binary value in the first three bytes followed by
*      * and ebcdic contanst that indicates the demonination.
*      *
*      * When PGTHCMEMLTYPE is off PGTHCMEMLIMIT consists or a
*      * 32bit binary value.
*      *
*      ************************************************************
*
*
PGTH#KILO EQU  C'K'      Kilobytes
PGTH#MEGA EQU  C'M'      Megabytes
PGTH#GIGA EQU  C'G'      Gigabytes
PGTH#TERA EQU  C'T'      Terabytes
PGTH#PETA EQU  C'P'      Petabytes
PGTHC#LEN             EQU   *-PGTHC
*
* USING PGTHD,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFD
PGTHD                 DSECT ,        C O N T T Y - - - - - - - - - -
PGTHDID               DS    CL4      "gthd"
```

```
PGTHDLEN              DS    FL2      Length of ConTty
PGTHDCONTTY           DS    CL1024   1024 = max ConTty
*
* USING PGTHE,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFE
PGTHE                 DSECT ,        P A T H - - - - - - - - - - - -
PGTHEID               DS    CL4      "gthe"
PGTHELEN              DS    FL2      Length of Path
PGTHEPATH             DS    CL1024   1024 = max path
*
* USING PGTHF,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFF
PGTHF                 DSECT ,        C O M M A N D - - - - - - - - -
PGTHFID               DS    CL4      "gthf"
PGTHFLEN              DS    FL2      Length of command and arguments
PGTHFCOMMAND          DS    CL1024   1024 = max command
          ORG  PGTHFCOMMAND
PGTHFCOMMANDL         DS    CL2048    Allow up to 2K for cmd/args
                      SPACE ,
*
* USING PGTHG,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFG
PGTHG                 DSECT ,        F I L E   H E A D E R - - - - -
PGTHGID               DS    CL4      "gthg"
PGTHGLIMITH           DS    CL1      N, A, S, X
PGTHGOFFH             DS    FL3      Offset of PgthH
PGTHGCOUNT            DS    F        Count of PgthH elements
PGTHGMAXVNODETOKENS   DS    F        MAX NUMBER VNODE TOKENS
PGTHGVNODETOKENCOUNT  DS    F        CURRENT NUMBER VNODE TOKENS
PGTHGSERVERFLAGS      DS    F        SABFLAGS
PGTHGSERVERNAME       DS    CL32     SABSERVERNAME SERVER=
PGTHGACTIVEFILES      DS    F        SABVDECOUNT   AF=
PGTHGMAXFILES         DS    F        SABMAXVDES    MF=
PGTHGSERVERTYPE       DS    F        SABSERVERTYPE TYPE=
PGTHG#LEN             EQU   *-PGTHG
*
PGTHGARRAY            DS    0C       first PGTHH
*
* USING PGTHH,Rx where Rx = ADDRESS of PGTHB + PGHTGOFFH
* Increment   Rx by PGTHH#LEN until PGTHGCOUNT exhausted
PGTHH                 DSECT ,        F I L E   D A T A - - - - - - -
PGTHHID               DS    CL2
PGTHH#IDR             EQU   C'rd'    root directory    (first)
PGTHH#IDC             EQU   C'cd'    current directory (second)
PGTHH#IDF             EQU   C'fd'    file directory
PGTHH#IDV             EQU   C'vd'    vnode directory
PGTHHTYPE             DS    BL1      Mapped in BPXYFTYP see FT_DIR +
PGTHHOPEN             DS    BL1      Mapped in BPXYOPNF see O_FLAGS4
PGTHHINODE            DS    F        I-NODE        see stat()
PGTHHDEVNO            DS    F        DEVICE NUMBER see stat()
PGTHH#LEN             EQU   *-PGTHH
*
* USING PGTHJ,Rx where Rx = ADDRESS of PGTHB + PGTHBOFFJ
* Reset Rx to be PGTHB + PGTHJOFFJ for the next thread
PGTHJ                 DSECT ,        T H R E A D - - - - - - - - - -
PGTHJID               DS    CL4      "gthj"
PGTHJLIMITJ           DS    CL1      A, S, X
PGTHJOFFJ             DS    FL3      Offset of next PgthJ
PGTHJLIMITK           DS    CL1      N, A, S, X
PGTHJOFFK             DS    FL3      Offset of PgthK, this thread
PGTHJTHID             DS    CL8      THREAD ID
PGTHJSYSCALL          DS    CL4      SYSCALL (eg. "1FRK" for fork)
PGTHJTCB              DS    A        TCB ADDRESS
PGTHJTTIME            DS    F        TIME RUNNING .001 SECS
PGTHJWTIME            DS    F        OE WAITING TIME .001 SECS
                      DS    F         space
PGTHJSEMNUM           DS    H        SEMAPHORE NUMBER IF STATUS2=D
PGTHJSEMVAL           DS    H        SEMAPHORE VALUE  IF STATUS2=D
PGTHJLATCHWAITPID     DS    F        LATCH PROCESS ID WAITED FOR
PGTHJPENMASK          DS    XL8      SIGNAL PENDING MASK
```

```
PGTHJLOGINNAME        DS    CL8      LOGIN NAME - ALIAS or MVS
PGTHJPREVSC           DS    5CL4     LAST FIVE SYSCALLS
PGTHJSTATUSCHARS      DS    0CL5     STATUS
*
PGTHJSTATUS1          DS    CL1      STATUS 1
PGTHJ#PTHDCREATED     EQU   C'J'       pthread created
*
PGTHJSTATUS2          DS    CL1      STATUS 2
PGTHJ#MSGRCV          EQU   C'A'       msgrcv wait
PGTHJ#MSGSND          EQU   C'B'       msgsnd wait
PGTHJ#WAITC           EQU   C'C'       communication wait
PGTHJ#SEMOP           EQU   C'D'       see PgthJSemVal/SemNum
PGTHJ#WAITF           EQU   C'F'       file system wait
PGTHJ#MVSPAUSE        EQU   C'G'       MVS in pause
PGTHJ#WAITO           EQU   C'K'       other kernel wait
PGTHJ#WAITP           EQU   C'P'       PTwaiting
PGTHJ#RUN             EQU   C'R'       running / non-kernel wait
PGTHJ#SLEEP           EQU   C'S'       sleep
PGTHJ#CHILD           EQU   C'W'       waiting for child
PGTHJ#FORK            EQU   C'X'       fork new process
PGTHJ#MVSWAIT         EQU   C'Y'       MVS wait
*
PGTHJSTATUS3          DS    CL1      STATUS 3
PGTHJ#MEDIUMWGHT      EQU   C'N'       medium weight thread
PGTHJ#ASYNC           EQU   C'O'       asynchronous thread
PGTHJ#IPT             EQU   C'U'       Initial process thread
PGTHJ#ZOMBIE          EQU   C'Z'       Process terminated and parent
*                                           has not completed wait
*
PGTHJSTATUS4          DS    CL1      STATUS 4
PGTHJ#DETACHED        EQU   C'V'       thread is detached
*
PGTHJSTATUS5          DS    CL1      STATUS 5
PGTHJ#FREEZE          EQU   C'E'       quiesce freeze
                      DS    CL3
*
PGTHJ#LEN             EQU   *-PGTHJ
* USING PGTHH,Rx where Rx = ADDRESS of PGTHB + PGTHJOFFK
PGTHK                 DSECT ,        P T A G - - - - - - - - - - - -
PGTHKDATALEN          DS    F        LENGTH TO TRAILING NULL
PGTHKDATA             DS    CL68     SEE pthread_tag_np
PGTHK#LEN             EQU   *-PGTHH
*
** BPXYPGTH End
```

BPXYPOE — Map poe syscall parameters

This structure is passed to the poe syscall.

```
           BPXYPOE   ,
POE      DSECT
POEOPTIONS DS  1FL4      +00 Options for POE
POEENTRYTYPE DS 1FL4     +04 Point Of Entry Type
POEENTRYLEN DS 1FL4      +08 Point Of Entry Length
         DS    1CL0004   +0C Reserved
POEENTRYPTR64 DS 0CL0008 +10 64 Address of Port of Entry
         DS    1CL0004   +10 Padding
POEENTRYPTR DS 1AL4      +14 Address of Point Of Entry
POE#LEN  EQU   24        Length of POE
POE#ENTRYSOCKET EQU 1    Entry is a file descriptor for a socket file
POE#ENTRYSOCKETLEN EQU 4 Length of file descriptor of a socket file
POE#ENTRYFILE EQU 2      Entry is a file descriptor for a non-socket   X
                         file
POE#ENTRYFILELEN EQU 4   Length of file descriptor for a non-socket    X
                         file
*
```

```
*     Options for Poe
*
*
POE#SCOPETHREAD EQU 1    Thread scope
POE#SCOPEPROCESS EQU 2   Process scope
POE_LEN  EQU   *-POE
```

BPXYPOLL — Map poll syscall parameters

This structure is passed to the poll syscall.

```
           BPXYPOLL   ,
** BPXYPOLL: POLLFD structure for poll syscall
**  Used By: POL
POLLFD               DSECT ,
POLLHFD              DS    FL4       File descriptor
                     ORG   POLLHFD
POLLHMQID            DS    FL4       Message queue identifier
POLLEVENTS           DS    0XL2      Events
                     DC    XL1'0'    Reserved
                     DS    XL1       POLLEVENTS+1
POLLEPRI             EQU   X'10'     High-pri data may be recv'd
POLLEWRBAND          EQU   X'08'     Priority data may be written
POLLEWRNORM          EQU   X'04'     Data on band 0 may be written.
POLLEOUT             EQU   X'04'     Same as WrNorm
POLLEIN              EQU   X'03'     Same as RdNorm
POLLERDBAND          EQU   X'02'     Non-0 band data may be read
POLLERDNORM          EQU   X'01'     Data on band 0 may be read.
POLLREVENTS          DS    0XL2      Returned events
                     DS    XL1       Reserved
                     DS    XL1       POLLREVENTS+1
POLLRNVAL            EQU   X'80'     Invalid FD member.(Revent Only)
POLLRHUP             EQU   X'40'     Hangup occurred (Revent Only)
POLLRERR             EQU   X'20'     Error occurred. (Revent Only)
POLLRPRI             EQU   X'10'     High-pri data may be recv'd
POLLRWRBAND          EQU   X'08'     Priority data may be written
POLLRWRNORM          EQU   X'04'     Data on band 0 may be written.
POLLROUT             EQU   X'04'     Same as WrNorm
POLLRIN              EQU   X'03'     Same as RdNorm
POLLRRDBAND          EQU   X'02'     Non-0 band data may be read
POLLRRDNORM          EQU   X'01'     Data on band 0 may be read.
POLLFD#LENGTH        EQU   *-POLLFD
*
#POLLEMASK           EQU   X'001F'
#POLLRDMASK          EQU   X'00130000'  All Read bits
#POLLWRMASK          EQU   X'000C0000'  All Write bits
#POLLPRIMASK         EQU   X'00100000'  The PollPri bit
#POLLINMASK          EQU   X'00030000'  Pollin rdnorm rdband bits
#POLLRNMASK          EQU   X'00010000'  Read Normal
#POLLWNMASK          EQU   X'00040000'  Write Normal
#POLLEVMASK          EQU   X'001F0000'  Events
*
*
** BPXYPOLL End
```

BPXYPPSD — Map signal delivery data

This structure is passed to a signal interface routine (SIR). AMODE 64 callers use "BPXYPPSD — Map signal delivery data" on page 1108.

```
           BPXYPPSD   ,
** BPXYPPSD: Signal Data Area
**  Used By: User written signal interrupt routines
PPSD                 DSECT ,
```

```
PPSDID              DC    C'PPSD'  Eye catcher
PPSD#ID             EQU   C'PPSD'    Control Block Acronym
PPSDSP              DS    FL1      Subpool number of this PPSD
PPSD#SP             EQU   230        Subpool for the PPSD
PPSDLEN             DC    AL3(PPSD#LENGTH)   Length this structure
*
*    ****************************************************************
*     PpsdSIRParms is used to setup up a parameter list to the
*     Signal Interface Routine (SIR).  When the SIR is invoked, the
*     address of PpsdSIRParms field is set in Register 1.  The
*     PpsdAddrPpsd contains the address of the Ppsd.
*    ****************************************************************
*
PPSDSIRPARMS        DS    0A       SIR Parameters
PPSDADDRPPSD        DC    A(PPSD)  Pointer to the top of the Ppsd
PPSDSIRPARMEND      EQU   X'80'    End of Parameters flag set on
PPSDTRMEXITSTATUS   DS    F        4 Byte status passed to PRTRM
PPSDSIGNUM          DS    F        Signal number
PPSDFL              DS    XL2      X'7FFF' reserved
                    ORG   PPSDFL
PPSDFLAGS2A         DS    0B
PPSDQUIESCEFREEZE   EQU   X'80'      Interrupt due to freeze
PPSDSIRCOMPLETE     EQU   X'40'    Sir done with async I/O exits
PPSDPROCDFT         EQU   X'20'    Process default
PPSDSIGQUEUE        EQU   X'10'    NSSGQ queued signal
PPSDREDRIVE         EQU   X'08'    SPB will Resend signal later
PPSDJUMPBACK        EQU   X'04'    SPB return to point of interrupt
PPSDMASKONLY        EQU   X'02'    SPB restore mask only
PPSDSIGTHSTOP       EQU   X'01'    Interrupt due to thread-stop
*                                  signal
                    ORG   PPSDFL+0001
PPSDFLAGS2B         DS    B
PPSDQUIESCEANDGET   EQU   X'80'    Interrupt due to
*                                  pthread_quiesce_and_get_np
PPSDF2_64           EQU   X'40'    Use PSWxxx64 fields
PPSDACTION          DS    B        Action for this signal
*                                     catch
*                                     SIR determines default action
PPSDFLAGS           DS    B        X'00' reserved
PPSDASYNC           EQU   X'80'    Signal delivered Asynchronously
PPSDDUMP            EQU   X'40'    Dump for terminating signals
PPSDPTHREADKILL     EQU   X'20'    Signal sent via BPX1PTK
PPSDTHISTHREADGEN   EQU   X'10'    Sending=Receiving thread
PPSDSIGNAL          EQU   X'08'    Interrupt due to signal
PPSDCANCEL          EQU   X'04'    Interrupt due to cancel
PPSDQUIESCE         EQU   X'02'    Interrupt due to quiesce
PPSDIPT             EQU   X'01'    If ON then this is the IPT
PPSDSAHANDLER       DS    A        Addr of catcher function
PPSDSAMASK          DS    XL8      Signal mask set by BPX1SIA for
*                                  this signal
PPSDSAFLAGS         DS    XL4      X'00FFFFFF' reserved
PPSDNOCLDSTOP       EQU   X'80'      Do not generate SIGCHLD on stops
PPSDOLDSTYLE        EQU   X'40'      Signal defined by signal() funct.
PPSDONSTACK         EQU   X'20'      Deliver on alternate stack
PPSDRESETHAND       EQU   X'10'      Reset action on delivery
PPSDRESTART         EQU   X'08'      Restart interruptable funcs
PPSDSIGINF          EQU   X'04'      Pass sig info to catcher
PPSDNOCLDWAIT       EQU   X'02'      Don't create zombie on exit
PPSDNODEFER         EQU   X'01'      Don't block sig on delivery
PPSDCURRENTMASK     DS    XL8      This is the signal mask to be set
*                                  when the signal catcher returns.
*                                  Signal mask at time of interrupt
*                                  except for sigsuspend case. If
*                                  signal during sigsuspend, then
*                                  this mask is the signal mask prior
*                                  to call to sigsuspend.
PPSDSIR             DS    A        Addr Signal interrupt routine
```

```
PPSDUSERDATA         DS    A        User data speicified on BPX1MSS
PPSDGENREGS          DS    CL64     Users general regs at interrupt
PPSDPSW              DS    XL8      Users PSW at interrupt
PPSDARREGS           DS    16F      Users AR regs at interrupt
PPSDKILDATA          DS    FL2      User specified data on BPX1KIL
PPSDKILOPTS          DS    XL2      X'7FFF' reserved
*                                   User specified options on BPX1KIL
PPSDPTBYPASS         EQU   X'80'      Ptrace Bypass option in effect
PPSDKERNSICODE       EQU   X'40'      PpsdKilData=Kern set SiCode
PPSDAPPLSICODE       EQU   X'20'      PpsdKilData=Appl set SiCode
PPSDCONSCANCEL       EQU   X'10'      Console MODIFY cancel qualifier
*                                     in PpdsKilData
PPSDSUPERKILL        EQU   X'08'      Superkill option on BPX1KIL
PPSDTRACEOVERRIDE    EQU   X'04'      SYSCALL Trace Override Option
PPSDTRACEACTION      EQU   X'02'      SYSCALL Trace Action Setting

PPSDQUIESCEDATA      DS    F        Quiesce_Data specified on BPX1QUT
PPSDLASTPTSIG        DS    F        Last Ptraced Signal
PPSDSIGACTIONDATA    DS    F        User_Data specified on BPX1SIA
PPSDPTXLWAPTR        DS    A        Threads workarea address specified
*                                   on BPX1PTC (pthread_create).  This
*                                   address is zero if the thread was
*                                   not pt_created.
PPSDSENDINGTHREAD    DS    CL8      Sending thread id
PPSDTARGETTHREAD     DS    CL8      Target thread id
PPSDSENDINGPID       DS    F        Sending process id
PPSDSENDINGUID       DS    F        Sending real uid
PPSDSIADDR           DS    A        Address of faulting instruction
*                                   for SIGILL, SIGFPE, SIGSEGV
PPSDSISTATUS         DS    F        Exit status or signal
PPSDSIBAND           DS    F        Band event
PPSDERRNO            DS    F        Error return code
PPSDCATCHERMASK      DS    XL8      Signal Mask to be set before signal
*                                  catcher is called. If signal during
*                                  sigsuspend then this field is same
*                                  as mask specified on sigsuspend. If
*                                  not sigsuspend, then PpsdCatcherMask
*                                  and PpsdCurrentMask are equal.
PPSDRES10            DS    25F       Reserved
PPSDSQV              DS    F         Signal si_value
PPSDREDRIVETIME      DS    F         Time to delay signal 1000 per mic
PPSDG64H             DS    16F       Users G64H at interrupt
PPSDRRTRMMSGTHID     DS    CL8       Sending thread id for MSG
*                                  BPXP010I
PPSDSENDINGJOBNAME   DS    CL8       Jobname of thread sending signal
*
                     DS    4F        Reserved
PPSDRES11            DS    22F       Reserved in 31 bit mode
                     DS    4F        Reserved
                     DS    FL2       Reserved
PPSDAIOCB64          DS    FL2       Amode(64) Exit Flags
PPSDEXCOUNT          DS    FL2       Count of PpsdAiocb's
PPSDEXLASTIX         DS    FL2       Last array index used
PPSDAIOCB            DS    12D       Aiocb Array for Async Exit
PPSDEND              DS    0D        End of PPSD on double word
PPSD#LENGTH          EQU   *-PPSD    Length of this structure
** BPXYPPSD End
```

BPXYPRLI — Process-level information

```
            BPXYPRLI   ,
** BPXYPRLI: Process Level Information
PRLI            DSECT ,
PRLIID          DC    C'PRLI'   EBCDIC ID
PRLISP          DS    FL1       Subpool number of this Prli
PRLILEN         DS    FL3       Length of this Prli
```

```
PRLIPROCESSID   DS    F         Process ID. Used for fast getpid()
PRLICATCHERMASK DS    BL8       Mask of signals that may be caught
PRLIOAPB        DS    A         Oapb Addr of this process
PRLIFLAG        DS    BL1       Flag byte
PRLIF1MED       EQU   B'10000000'  Process is medium weight local
PRLIF1DISSIG    EQU   B'01000000'  Disable signal delivery
PRLISYSCONFOK   EQU   B'00100000'  SC_ fields valid. Note, that        X
                                   this implies the OEXT SC_          X
                                   fields are also valid
PRLIF1TERMT     EQU   B'00010000'  Terminate threads
PRLIMAGICNUMBER DS    CL2       Magic Number Characters
                DS    CL1       Reserved
PRLIL16JRC      DS    F         Return code area for L16J FastCGI
PRLIRUID        DS    A         Real Uid addr
PRLIEUID        DS    A         Effective Uid addr
PRLIRGID        DS    A         Real Gid addr
PRLIEGID        DS    A         Effective gid
PRLIPROCGRPID   DS    A         Process Group ID addr
PRLIPARENTPID   DS    A         Parent Process ID addr
PRLITHREADTASKSMAX DS A         SC_THREAD_TASK_MAX_NP value addr
PRLITHREADSMAX  DS    A         SC_THREADS_MAX_NP value address
PRLICHILDMAX    DS    A         SC_CHILD_MAX value addr
PRLIOPENMAX     DS    A         SC_OPEN_MAX value address
PRLIMMAPMEMMAX  DS    A         SC_MMAP_MEM_MAX value address
PRLIEND         DS    0C        End of Prli
PRLI#ID         EQU   C'PRLI'   Control Block Acronym
PRLI#LEN        EQU   32        Length of Prli
PRLI#SP         EQU   230       Subpool for the Prli
PRLI_LEN        EQU   *-PRLI
** BPXYPRLI End
```

BPXYPTAT — Map attributes for pthread_exit_and_get

VARLEN defines the number of bytes set aside to define the pthread attributes.

```
           BPXYPTAT   VARLEN=1024
** BPXYPTAT: Pthread Attributes
**  Used By:
PTAT                DSECT ,
PTATEYE             DC    C'BPXYPTAT'       Eye Catchter
PTATLENGTH          DC    A(PTAT#LENGTH)    Length of PTAT
PTATSYSOFFSET       DC    A(PTATSYSOFFVAL)  Offset of SYSATTRS
PTATSYSLENGTH       DC    A(PTATSYSLENVAL)  Length of SYSATTRS
PTATUSEROFFSET      DC    A(PTATUSEROFFVAL) Offset of USERATTRS
PTATUSERLENGTH      DC    A(L'PTATUSERATTRS) Length of USERATTRS
PTATSYSOFFVAL       EQU   *-PTAT Offset value of System Attribute Area
PTATSYSATTRS        DS    0F System attributes
PTATDETACHSTATE     DS    F  Detach State of thread to be created:
PTATUNDETACHED      EQU   0
PTATDETACHED        EQU   1
PTATWEIGHT          DS    F  Weight of thread to be created:
PTATHEAVY           EQU   0
PTATMEDIUM          EQU   1
PTATSYNCTYPE        DS    F  Synchronous processing type of thread:
PTATSYNCHRONOUS     EQU   0
PTATASYNCHRONOUS    EQU   1                                 /*
PTATSHSPMASK        DS    0XL16                             /*
                    DS    XL15                              /*
PTATSHSPBYTE16      DS    XL1                               /*
PTATSHSPINUSE       EQU   X'01' 0=system default used
*                           1=use mask
*                           default shared subpools 1, 2, 78
PTATSYSLENVAL       EQU   *-PTATSYSATTRS Length of System Attributes
```

```
PTATUSEROFFVAL       EQU   *-PTAT   Offset of user attribute area
PTATUSERATTRS        DS    CL1024   User attributes area
PTAT#LENGTH          EQU   *-PTAT   Length of this structure
** BPXYPTAT End
```

BPXYPTRC — Map parameters for ptrace

VARLEN defines the number of bytes needed to hold the pathname (the default is the maximum pathname, 1024).

```
           BPXYPTRC
*
* ********************************************************************
* *                                                                  *
* * Ptrace PT_LDINFO return structure.  Note that this maps one      *
* * element, corresponding to one load module.  Each element        *
* * consists of a fixed portion, and a variable portion (the path    *
* * name and member name character strings).  The character strings  *
* * are terminated with a null value (X'00').  Each loader info      *
* * element immediately follows the last null terminator for the     *
* * previous element.  The first full word of each element is an     *
* * offset to the next element.  Thus, the start of the next element *
* * can be specified as follows:                                     *
* *                                                                  *
* *     NextLDInfo = Addr(PtLDInfo)+PtLDInfoNext                     *
* *                                                                  *
* ********************************************************************
*
*
PTLDINFO DSECT
PTLDFIXEDAREA DS 0CL0032
PTLDINFONEXT DS 1FL4     Offset to next element
PTLDINFOFD DS  1FL4      File descriptor for this load module (not     X
                         used)
PTLDTEXTORG DS 1AL4      Program text origin address (i.e. load point  X
                         address)
PTLDTEXTSIZE DS 1FL4     Length of text
PTLDTEXTSUBPOOL DS 1CL0001 Subpool where text is loaded
PTLDTEXTFLAGS DS 0BL1    Text related flags
PTLDTEXTWRITE EQU X'80'  0 = text can be read but not written into 1 = X
                         text can be read and written into
PTLDTEXTMVS EQU X'40'    0 = File system load module 1 = MVS load      X
                         module
PTLDTEXTEXT EQU X'20'    0 = Only 1 text extent 1 = More than one text X
                         extent. First extent is in this element,      X
                         extent 2 - n are in the PtLDInfoExt area
         ORG   PTLDTEXTFLAGS+X'00000001'
PTLDOFFEXT DS  1FL2      Offset from this element to element           X
                         extension. 0 if there is no extension for     X
                         this element
PTLDDATAORG DS 1AL4      Program data origin address (not used)
PTLDDATASIZE DS 1FL4     Length of data (not used)
PTLDDATASUBPOOL DS 1FL1  Subpool where data is loaded (not used)
PTLDDATAFLAGS DS 1BL1    Data related flags (not used)
         DS    1FL2      Reserved
PTLDVARAREA DS 0C
PTLDPATHNAME DS 0C       Fully qualified path name of load module
PTLDMEMBERNAME DS 0C     Member name of load module (not used)
PTLDINFO_LEN EQU *-PTLDINFO
PTLDINFOEXT DSECT
PTLDINFONUMTEXTEXT DS 1FL2 Number of additional text extents in the    X
                         following arrays that are meaningful, up to   X
                         15 in this area, for a total of 16
         DS    1FL2      reserved
PTLDTEXTORGEXT DS 1AL4   Program text origin address (i.e. load point  X
                         address)
```

```
         ORG   PTLDTEXTORGEXT+X'0000003C'
PTLDTEXTSIZEEXT DS 1FL4  Length of text
PTLDNULLTERM EQU 0       Null terminator for character strings
         ORG   PTLDINFOEXT+X'0000007C'
PTLDINFOEXT_LEN EQU *-PTLDINFOEXT
*
* ******************************************************************
* *                                                                *
* * Ptrace thread information return structure.  Note that this    *
* * maps one element, corresponding to one thread.                 *
* *                                                                *
* * Note: the only valid information for a dead thread is:         *
* *                                                                *
* *   PTPTNEXT, PTPTTHID, PTPTSTATEACTIVE=0, PTPTKERNELPTHREAD,    *
* *   PTPTEXITSTATUS                                               *
* ******************************************************************
*
*
PTPTINFO DSECT
PTPTNEXT DS    1AL4      Offset to next element
PTPTTHID DS    1CL0008   Thread ID
PTPTRESERVED DS 1CL0016  Reserved
PTPTSTATE DS   0BL4      Thread state flags
PTPTSTATE1 DS  0BL1      Thread state flag byte
PTPTSTATEACTIVE EQU X'80'
*
*      ****************************************************************
*      *                                                              *
*      * 0 = thread is dead
*      * 1 = thread is active
*      *                                                              *
*      ****************************************************************
*
*
PTPTSTATEASYNC EQU X'40'
*
*      ****************************************************************
*      *                                                              *
*      * 1 = thread is asynchronous
*      * (is also active but not
*      * yet running)
*      *                                                              *
*      ****************************************************************
*
*
PTPTSTATECANCELPEND EQU X'20'
*
*      ****************************************************************
*      *                                                              *
*      * 1 = cancel is pending
*      *                                                              *
*      ****************************************************************
*
*
         ORG   PTPTSTATE1+X'00000001'
PTPTSTATE2 DS  1BL1      Thread state flag byte
PTPTSTATE3 DS  1BL1      Thread state flag byte
PTPTSTATE4 DS  1BL1      Thread state flag byte
PTPTKERNELATTR DS 0BL4   Thread kernel attributes
PTPTKERNEL1 DS 0BL1      Thread kernel attribute byte
PTPTKERNELDETACH EQU X'80'
*
*      ****************************************************************
*      *                                                              *
*      * 0 = thread is not detached
*      * 1 = thread is detached
*      *                                                              *
```

```
*      ***************************************************************
*
*
PTPTKERNELMEDIUM EQU X'40'
*
*      ***************************************************************
*      *                                                             *
*      * 0 = thread is heavyweight
*      * 1 = thread is mediumweight
*      *                                                             *
*      ***************************************************************
*
*
PTPTKERNELASYNC EQU X'20'
*
*      ***************************************************************
*      *                                                             *
*      * 0 = thread is synchronous
*      * 1 = thread is asynchronous
*      *                                                             *
*      ***************************************************************
*
*
PTPTKERNELPTHREAD EQU X'10'
*
*      ***************************************************************
*      *                                                             *
*      * 1 = thread is created via
*      * pthread create
*      *                                                             *
*      ***************************************************************
*
*
         ORG   PTPTKERNEL1+X'00000001'
PTPTKERNEL2 DS 0BL1      Thread kernel attribute byte
PTPTKERNELHOLD EQU X'80'
*
*      ***************************************************************
*      *                                                             *
*      * 1 = thread is held
*      *                                                             *
*      ***************************************************************
*
*
         ORG   PTPTKERNEL2+X'00000001'
PTPTKERNEL3 DS 1BL1      Reserved
PTPTKERNEL4 DS 1BL1      Reserved
PTPTEXITSTATUS DS 1CL0004
*
*      ***************************************************************
*      *                                                             *
*      * Thread exit status if dead
*      * (PtptStateActive = 0)
*      * (Low half if AMODE 64)
*      *                                                             *
*      ***************************************************************
*
*
PTPTPENDINGSIGMASK DS 1BL8
PTPTEXITSTATUSHIGH DS 1CL0004
*
*      ***************************************************************
*      *                                                             *
*      * If AMODE 64, high half of thread exit status
*      * (PtptStateActive = 0)
*      * (Not used for AMODE 31)
*      *                                                             *
```

```
*      ***************************************************************
*
*
         DS    1FL4      Reserved
PTPTINFO_LEN EQU *-PTPTINFO
*
*    ***************************************************************
*    *                                                             *
*    * Mask of pending signals
*    * (bit 0 represents signal 1)
*    * (bit 63 represents signal 64)
*    *                                                             *
*    ***************************************************************
*
* ****************************************************************
* *                                                              *
* * Ptrace thread information extended structure. Note that this     *
* * maps one element, corresponding to one thread. PtpxInfo maps     *
* * exactly to PtPtInfo                                            *
* *                                                              *
* *                                                              *
* *                                                              *
* * Note: the only valid information for a dead thread is:           *
* *                                                              *
* *   PTPXNEXT, PTPXTHID, PTPXSTATEACTIVE=0, PTPXKERNELPTHREAD,      *
* *   PTPXEXITSTATUS                                             *
* ****************************************************************
*
*
PTPHINFO DSECT           PT_THREAD_INFO_EXTENDED header information
PTPHINFOBASE DS 0CL0052
*
*      ***************************************************************
*      *                                                             *
*      * PtphInfoBase contains information about the process and     *
*      * pointers to the next array of thread info
*      *                                                             *
*      ***************************************************************
*
*
PTPHID   DS    1CL0004   Acronym
PTPHNEXT DS    1AL4      Address of the next PtPhInfo
PTPHPTPXOFF DS 1AL4      Offset of first Ptpx in this chunk of storage
PTPHPID  DS    1FL4      Process id of the threads
PTPHPENDINGSIGMASK DS 1BL8 Signals pending at the process
PTPHBLOCKEDSIGMASK DS 1BL8 blocked signals at process
PTPHTHREADNUM DS 1FL4    Total number of threads reported in chain
PTPHTNUM DS    1CL0004   Threads in the Current buffer
PTPHPTPXLEN DS 1FL4      Length of the PtPx in this buffer
         DS    1CL0004   Reserved
         ORG   PTPHINFO+X'00000034'
PTPHINFO_LEN EQU *-PTPHINFO
PTPXINFO DSECT           PT_THREAD_INFO_EXTENDED maps a single entry
PTPXINFOBASE DS 0CL0072
PTPXNEXT DS    1AL4      Offset to next element
PTPXTHID DS    1CL0008   Thread ID
PTPXTCB  DS    1AL4      Tcb address for this process
PTPXOTCB DS    1AL4      Otcb address for this process
PTPXBLOCKEDSIGMASK DS 1BL8 blocked signals
PTPXSTATE DS   0BL4      Thread state flags
PTPXSTATE1 DS  0BL1      Thread state flag byte
PTPXSTATEACTIVE EQU X'80' 0 = thread is dead
*
*      ***************************************************************
*      *                                                             *
*      * 1 = thread is active
*      *                                                             *
```

```
*      ***************************************************************
*
*
PTPXSTATEASYNC EQU X'40' 1 = N/A
*
*      ***************************************************************
*      *                                                             *
*      * (is also active but not
*      * yet running)
*      *                                                             *
*      ***************************************************************
*
*
PTPXSTATECANCELPEND EQU X'20'
*
*      ***************************************************************
*      *                                                             *
*      * 1 = cancel is pending
*      *                                                             *
*      ***************************************************************
*
*
         ORG   PTPXSTATE1+X'00000001'
PTPXSTATE2 DS  1BL1      Thread state flag byte
PTPXSTATE3 DS  1BL1      Thread state flag byte
PTPXSTATE4 DS  1BL1      Thread state flag byte
PTPXKERNELATTR DS 0BL4   Thread kernel attrtes
PTPXKERNEL1 DS 0BL1      Thread kernel attribute byte
PTPXKERNELDETACH EQU X'80'
*
*      ***************************************************************
*      *                                                             *
*      * 0 = thread is not detached
*      * 1 = thread is detached
*      *                                                             *
*      ***************************************************************
*
*
PTPXKERNELMEDIUM EQU X'40'
*
*      ***************************************************************
*      *                                                             *
*      * 0 = thread is heavyweight
*      * 1 = thread is mediumweight
*      *                                                             *
*      ***************************************************************
*
*
PTPXKERNELASYNC EQU X'20'
*
*      ***************************************************************
*      *                                                             *
*      * 0 = thread is synchronous
*      * 1 = thread is asynchronous
*      *                                                             *
*      ***************************************************************
*
*
PTPXKERNELPTHREAD EQU X'10'
*
*      ***************************************************************
*      *                                                             *
*      * 1 = thread is created via
*      * pthread create
*      *                                                             *
*      ***************************************************************
*
```

```
*
         ORG   PTPXKERNEL1+X'00000001'
PTPXKERNEL2 DS 0BL1      Thread kernel attribute byte
PTPXKERNELHOLD EQU X'80'
*
*      ****************************************************************
*      *                                                              *
*      * 1 = thread is held
*      *                                                              *
*      ****************************************************************
*
*
         ORG   PTPXKERNEL2+X'00000001'
PTPXKERNEL3 DS 1BL1      Reserved
PTPXKERNEL4 DS 1BL1      Reserved
PTPXEXITSTATUS DS 1CL0004
*
*      ****************************************************************
*      *                                                              *
*      * Thread exit status if dead
*      * (PTPXStateActive = 0)
*      * (Low half if AMODE 64)
*      *                                                              *
*      ****************************************************************
*
*
PTPXPENDINGSIGMASK DS 1BL8 Mask of pending signals is set) (bit 0      X
                         represents signal 1)
PTPXPID  DS    1FL4      Process id for this Thid
PTPXASID DS    1FL2
PTPXFLAGS DS   0BL2      Thread related flags
PTPXIPT  EQU   X'80'     Ipt Thread
PTPXINCOMPLETE EQU X'40' The reported thread information is incomplete
         ORG   PTPXFLAGS+X'00000002'
PTPXOAPB DS    1AL4      Pointer to the Oapb
PTPXExitStatusHigh DS 1CL0004
*
*      ****************************************************************
*      *                                                              *
*      * If AMODE 64, high half of thread exit status
*      * (PtptStateActive = 0)
*      * (Not used for AMODE 31)
*      *                                                              *
*      ****************************************************************
*
PTPXINFO_LEN EQU *-PTPXINFO
*
* ********************************************************************
* *                                                                  *
* * Ptrace explain information return structure.                     *
* *                                                                  *
* ********************************************************************
*
*
PTEXINFO DSECT
PTEXREG1 DS    1FL4      Register 1 at CEEEVDBG entry
PTEXREG12 DS   1FL4      Register 12 at CEEEVDBG entry
PTEXREG13 DS   1FL4      Register 13 at CEEEVDBG entry
          DS   1FL4
PTEXG64R1  DS  FD        Register 1 at CEEEVDBG entry
PTEXG64R12 DS  FD        Register 12 at CEEEVDBG entry
PTEXG64R13 DS  FD        Register 13 at CEEEVDBG entry
PTEXINFO_LEN EQU *-PTEXINFO
*
* ********************************************************************
* *                                                                  *
* * Ptrace program recovery parameters structure.                    *
```

```
* * (This area is provided by the caller)                          *
* *                                                                *
* ****************************************************************
*
*
PTPICPARMS DSECT
PTPICREGISTERS DS 1AL4   Address of GPRs at time of interrupt
PTPICPSW DS    1AL4      Address of PSW at time of interrupt
PTPICINTCODE DS 1FL2     Program interrupt code
PTPICSIGNUMBER DS 1FL2   Return value indicating signal number that    X
                         should be raised by the caller if the         X
                         PtPICUseSigNum flag is set
PTPICFLAGS DS  0BL4      Flags
PTPICICMODIFIED EQU X'80' 0 = The instruction counter portion of the   X
                         PSW pointed to by the PtPICPSW field has not  X
                         been modified 1 = The instruction counter     X
                         portion of the PSW pointed to by the PtPICPSW X
                         field has been modified - continue execution  X
                         at this modified address
PTPICREGSMODIFIED EQU X'40' 0 = The registers pointed to by the        X
                         PtPICRegisters field have not been modified 1 X
                         = The registers pointed to by the             X
                         PtPICRegisters field have been modified
PTPICUSESIGNUM EQU X'20' 1 = Raise the signal number returned in the   X
                         PtPICSigNumber field upon return
PTPICBYPASSSIG EQU X'10' 1 = Do not raise any signal upon return
PTPICILCEXISTS EQU X'08' 1 = PtPICILC field is present
PTPICHIREGSEXISTS EQU X'04' 1 = The PtPICHiRegisters field is present
PTPICHIREGSMODIFIED EQU X'02' 0 = The registers pointed to by the      X
                         PtPICHiRegisters field have not been modified X
                         1 = The registers pointed to by the           X
                         PtPICHiRegisters field have been modified
PTPICAMODE64   EQU X'01' 1 = use 64 bit addresses for PSW and          X
                         registers
         DS    1BL.024   Reserved
         ORG   PTPICFLAGS+X'00000004'
PTPICABENDCODE DS 0BL4   Abend code or zero
PTPICABENDFLAGS DS 1BL1  Abend code flags
PTPICABENDCC DS 1BL3     System completion code (first 12 bits) and    X
                         user completion code (second 12 bits)
PTPICABENDREASON DS 1FL4 Abend reason code or zero
PTPICILC DS    1FL1      Instruction length code (only present if      X
                         PtPICILCExists flag is set)
PTPICRESERVED DS 1CL0003 Reserved
PTPICHIREGISTERS DS 1AL4 Address of high GPRs at time of interrupt
PTPICREGISTERS64 DS  AD  Address of GPRs at time of interrupt
PTPICPSW64 DS    AD      Address of PSW at time of interrupt
PTPICHIREGISTERS64 DS AD Address of high GPRs at time of interrupt
PTPICRSVD DS   1CL0008   Reserved. This area is provided by the caller X
                         and may not be present in old releases of     X
                         code
*
* ****************************************************************
* *                                                              *
* * Ptrace request parameter definitions.                        *
* *                                                              *
* ****************************************************************
*
*
PT_TRACE_ME EQU 0        Debug this process
PT_READ_I EQU  1         Read a full word
PT_READ_D EQU  2         Read a full word
PT_READ_U EQU  3         Read control info
PT_WRITE_I EQU 4         Write a full word
PT_WRITE_D EQU 5         Write a full word
PT_CONTINUE EQU 7        Continue the process
PT_KILL  EQU   8         Terminate the process
```

```
PT_READ_GPR EQU 11       Read GPR, CR, PSW
PT_READ_FPR EQU 12       Read FPR
PT_WRITE_GPR EQU 14      Write GPR, CR, PSW
PT_WRITE_FPR EQU 15      Write FPR
PT_READ_BLOCK EQU 17     Read storage
PT_WRITE_BLOCK EQU 19    Write storage
PT_READ_GPRH EQU 20      Read GPRH
PT_WRITE_GPRH EQU 21     Write GPRH
PT_REGHSET EQU 22        Read all GPRHs
PT_ATTACH EQU  30        Attach to a process
PT_DETACH EQU  31        Detach from a process
PT_REGSET EQU  32        Read all GPRs
PT_REATTACH EQU 33       Reattach to a process
PT_LDINFO EQU  34        Read loader info
PT_MULTI EQU   35        Multi process mode
PT_BLOCKREQ EQU 40       Block request
PT_THREAD_INFO EQU 60    Read thread info
PT_THREAD_MODIFY EQU 61  ********************************************X
                         ************** Modify thread kernel          X
                         information ********************************X
                         **************************
PT_THREAD_READ_FOCUS EQU 62 ******************************************X
                         ****************** Read current focus thread  X
                         ID ******************************************X
                         ******************
PT_THREAD_WRITE_FOCUS EQU 63 *****************************************X
                         ******************** Modify current focus      X
                         thread ID ***********************************X
                         ************************
PT_THREAD_HOLD EQU 64    ********************************************X
                         *************** Modify thread hold state *****X
                         ********************************************X
                         **********
PT_THREAD_SIGNAL EQU 65  ********************************************X
                         *************** Queue a signal for a thread **X
                         ********************************************X
                         *************
PT_EXPLAIN EQU 66        ********************************************X
                         *************** Return extended event info ***X
                         ********************************************X
                         ************
PT_EVENTS EQU  67        ********************************************X
                         *************** Modify extended events list **X
                         ********************************************X
                         *************
PT_THREAD_INFO_EXTENDED EQU 68 ***************************************X
                         ********************** Read extended thread    X
                         info ****************************************X
                         ********************
PT_REATTACH2 EQU 71      ********************************************X
                         *************** Reattach to a process         X
                         (extended) **********************************X
                         **************************
PT_CAPTURE EQU 72        ********************************************X
                         *************** Capture debugged storage *****X
                         ********************************************X
                         **********
PT_UNCAPTURE EQU 73      ********************************************X
                         *************** Uncapture debugged storage ***X
                         ********************************************X
                         ************
PT_GET_THREAD_TCB EQU 74 ********************************************X
                         *************** Get TCB address for thread ***X
                         ********************************************X
                         *********
PT_GET_ALET       EQU 75 ********************************************X
                         *************** Get Alet of target PID     ***X
```

```
                        ********************************************X
                        *********
PT_SWAPIN         EQU 76 ********************************************X
                        *************** Swapin target PID's A.S.   ***X
                        ********************************************X
                        *********
PT_EXTENDED_EVENT EQU 98 ********************************************X
                        *************** Debug an extended event ******X
                        ********************************************X
                        *********
PT_RECOVER EQU 99       Debug a program check
*
* ***********************************************************************
* *                                                                     *
* * Ptrace register definitions.  The following are defined:           *
* *                                                                     *
* * - General purpose registers                                        *
* * - Floating point registers                                         *
* * - PSW registers                                                    *
* * - Control registers                                                *
* * - General Purpose High Registers
* *                                                                     *
* ***********************************************************************
*
*
PT_GPR0  EQU   0        General purpose register 0
PT_GPR1  EQU   1        General purpose register 1
PT_GPR2  EQU   2        General purpose register 2
PT_GPR3  EQU   3        General purpose register 3
PT_GPR4  EQU   4        General purpose register 4
PT_GPR5  EQU   5        General purpose register 5
PT_GPR6  EQU   6        General purpose register 6
PT_GPR7  EQU   7        General purpose register 7
PT_GPR8  EQU   8        General purpose register 8
PT_GPR9  EQU   9        General purpose register 9
PT_GPR10 EQU   10       General purpose register 10
PT_GPR11 EQU   11       General purpose register 11
PT_GPR12 EQU   12       General purpose register 12
PT_GPR13 EQU   13       General purpose register 13
PT_GPR14 EQU   14       General purpose register 14
PT_GPR15 EQU   15       General purpose register 15
PT_FPR0  EQU   16       Floating point register 0
PT_FPR1  EQU   17       Floating point register 1
PT_FPR2  EQU   18       Floating point register 2
PT_FPR3  EQU   19       Floating point register 3
PT_FPR4  EQU   20       Floating point register 4
PT_FPR5  EQU   21       Floating point register 5
PT_FPR6  EQU   22       Floating point register 6
PT_FPR7  EQU   23       Floating point register 7
PT_FPR8  EQU   24       Floating point register 8
PT_FPR9  EQU   25       Floating point register 9
PT_FPR10 EQU   26       Floating point register 10
PT_FPR11 EQU   27       Floating point register 11
PT_FPR12 EQU   28       Floating point register 12
PT_FPR13 EQU   29       Floating point register 13
PT_FPR14 EQU   30       Floating point register 14
PT_FPR15 EQU   31       Floating point register 15
PT_FPC   EQU   32       Floating point control register
PT_PSW   EQU   40       PSW
PT_PSW0  EQU   40       Left half of the PSW
PT_PSW1  EQU   41       Right half of the PSW
PT_CR0   EQU   42       Control register 0
PT_CR1   EQU   43       Control register 1
PT_CR2   EQU   44       Control register 2
PT_CR3   EQU   45       Control register 3
PT_CR4   EQU   46       Control register 4
PT_CR5   EQU   47       Control register 5
```

```
PT_CR6   EQU   48        Control register 6
PT_CR7   EQU   49        Control register 7
PT_CR8   EQU   50        Control register 8
PT_CR9   EQU   51        Control register 9
PT_CR10  EQU   52        Control register 10
PT_CR11  EQU   53        Control register 11
PT_CR12  EQU   54        Control register 12
PT_CR13  EQU   55        Control register 13
PT_CR14  EQU   56        Control register 14
PT_CR15  EQU   57        Control register 15
PT_GPRH0 EQU   58        GP High register 0
PT_GPRH1 EQU   59        GP High register 1
PT_GPRH2 EQU   60        GP High register 2
PT_GPRH3 EQU   61        GP High register 3
PT_GPRH4 EQU   62        GP High register 4
PT_GPRH5 EQU   63        GP High register 5
PT_GPRH6 EQU   64        GP High register 6
PT_GPRH7 EQU   65        GP High register 7
PT_GPRH8 EQU   66        GP High register 8
PT_GPRH9 EQU   67        GP High register 9
PT_GPRH10 EQU  68        GP High register 10
PT_GPRH11 EQU  69        GP High register 11
PT_GPRH12 EQU  70        GP High register 12
PT_GPRH13 EQU  71        GP High register 13
PT_GPRH14 EQU  72        GP High register 14
PT_GPRH15 EQU  73        GP High register 15
*
* ********************************************************************
* *                                                                  *
* * Ptrace User Area offset definitions.  Offsets for signal catcher *
* * information are defined by the limits below.  Any offset between *
* * the minimum and maximum signal numbers is a request for signal   *
* * catcher information for that signal number (i.e. offset 3 means  *
* * signal catcher information for signal number 3).                 *
* *                                                                  *
* ********************************************************************
*
*
PTUAREA#MINSIG EQU 1     Lowest signal number
PTUAREA#MAXSIG EQU 1024  Highest signal number
PTUAREA#INTCODE EQU 1025 Request for program interrupt code
PTUAREA#ABENDCC EQU 1026 Request for abend completion code
PTUAREA#ABENDRC EQU 1027 Request for abend reason code
PTUAREA#SIGCODE EQU 1028 Request for signal code
PTUAREA#ILC EQU 1029     Request for instruction length code
PTUAREA#PRFLAGS EQU 1030 Request for process flags
*
* ********************************************************************
* *                                                                  *
* * Ptrace miscellaneous definitions.                                *
* *                                                                  *
* ********************************************************************
*
*
PTCONTNORM EQU 1         Continue normally (continue address not       X
                         changed) for a PT_CONTINUE request
PTNOSTICKYPGM EQU 1      Main program of process is not sticky bit     X
                         program. Returned on PTUAREA#PRFLAGS request
PTMAXIMUMLENGTH EQU 64000 Maximum storage length
PTLD#FIXEDLEN EQU 32     Length of PtLDInfo fixed area
PTPT#LENGTH EQU 52       Length of PtptInfo fixed area
PTPH#LENGTH EQU 56       Length of PtPhInfo fixed area on double word  X
                         boundary
PTPX#LENGTH EQU 72       Length of PtpxInfo fixed area on double word  X
                         boundary
PTEX#LENGTH EQU 40       Length of PtExInfo
PTEX31#LENGTH EQU 12     Length of PtExInfo 31 bit
```

```
PTPIC#LENGTH1 EQU 28     Length of PtPicParms if PtPicHiRegsExists =   X
                         OFF
PTPIC#LENGTH2 EQU 32     Length of PtPicParms if PtPicHiRegsExists =   X
                         ON
PTPIC#LENGTH EQU 64      Length of PtPICParms
PTPICPARMS_LEN EQU *-PTPICPARMS
*
* ********************************************************************
* *                                                                  *
* * Ptrace PT_BlockReq structure.  This request allows the user to   *
* * block several different Ptrace requests into a single call to    *
* * Ptrace.  The block request structures mapped below must be       *
* * contained in a single large area.  This area is pointed to by    *
* * the Ptrace Address parameter and its length is contained in the  *
* * Ptrace Data parameter.  The PtBRInfo structure must be at offset *
* * zero into the provided area.                                     *
* *                                                                  *
* * Offsets are used to locate all relevant areas so that the Ptrace *
* * block request input may be relocated.  All offsets are relative  *
* * to the main input, the PtBRInfo area.  A given request block,    *
* * such as the PtBR_GPR structure, may be found using the           *
* * PtBRInfo address + PtBRReqBlkOff(x).                             *
* *                                                                  *
* * Only certain requests may be blocked into a single call to      *
* * Ptrace.  The requests that may be blocked are --                 *
* *   PT_READ_GPR                                                    *
* *   PT_WRITE_GPR                                                   *
* *   PT_READ_FPR                                                    *
* *   PT_WRITE_FPR                                                   *
* *   PT_READ_GPRH                                                   *
* *   PT_WRITE_GPRH                                                  *
* *   PT_READ_U                                                      *
* *   PT_READ_D                                                      *
* *   PT_READ_I                                                      *
* *   PT_WRITE_D                                                     *
* *   PT_WRITE_I                                                     *
* *   PT_READ_BLOCK                                                  *
* *   PT_WRITE_BLOCK.                                                *
* *
* ********************************************************************
*
*
PTBRINFO DSECT
PTBRFIXEDAREA DS 0CL0016
PTBRNUMREQS DS 1FL4      Number of requests in PtBRReqs
         DS    1CL0012   Reserved
PTBRREQS DS    0CL0016   requests
PTBRTYPE DS    1FL4      Type of request. For example, PT_READ_BLOCK.  X
                         This entry is ignored if this field is zero
PTBRSTATUS DS  1FL4      Status from request. Same as reasoncode on    X
                         individual call of same type
PTBRREQBLKOFF DS 1FL4    Offset to request block further defining      X
                         request and whose format is dependant on the  X
                         request type
         DS    1CL0004   reserved
PTBRINFO_LEN EQU *-PTBRINFO
*
* ********************************************************************
* * Structure for PT_Read_GPR and PT_Write_GPR.                      *
* ********************************************************************
*
*
PTBR_GPR DSECT
PTBR_GPR_CNTLGPR DS 0BL2 Only used on write request
PTBR_GPR_CNTLGPR1 DS 0BL1
PTBR_GPR_WGPR0 EQU X'80' Write content of GPR 0
PTBR_GPR_WGPR1 EQU X'40' Write content of GPR 1
```

```
PTBR_GPR_WGPR2 EQU X'20' Write content of GPR 2
PTBR_GPR_WGPR3 EQU X'10' Write content of GPR 3
PTBR_GPR_WGPR4 EQU X'08' Write content of GPR 4
PTBR_GPR_WGPR5 EQU X'04' Write content of GPR 5
PTBR_GPR_WGPR6 EQU X'02' Write content of GPR 6
PTBR_GPR_WGPR7 EQU X'01' Write content of GPR 7
         ORG   PTBR_GPR_CNTLGPR1+X'00000001'
PTBR_GPR_CNTLGPR2 DS 0BL1
PTBR_GPR_WGPR8 EQU X'80' Write content of GPR 8
PTBR_GPR_WGPR9 EQU X'40' Write content of GPR 9
PTBR_GPR_WGPR10 EQU X'20' Write content of GPR 10
PTBR_GPR_WGPR11 EQU X'10' Write content of GPR 11
PTBR_GPR_WGPR12 EQU X'08' Write content of GPR 12
PTBR_GPR_WGPR13 EQU X'04' Write content of GPR 13
PTBR_GPR_WGPR14 EQU X'02' Write content of GPR 14
PTBR_GPR_WGPR15 EQU X'01' Write content of GPR 15
         ORG   PTBR_GPR_CNTLGPR+X'00000002'
PTBR_GPR_CNTLMISC DS 0BL2 Only used on write request
PTBR_GPR_WPSW EQU X'80'  Write content of PSW, word 2
         ORG   PTBR_GPR_CNTLMISC+X'00000002'
         DS    1CL0012   Reserved
PTBR_GPR_GPRS DS 0CL0064 General purpose registers
PTBR_GPR_GPR00 DS 1FL4   GPR 00
PTBR_GPR_GPR01 DS 1FL4   GPR 01
PTBR_GPR_GPR02 DS 1FL4   GPR 02
PTBR_GPR_GPR03 DS 1FL4   GPR 03
PTBR_GPR_GPR04 DS 1FL4   GPR 04
PTBR_GPR_GPR05 DS 1FL4   GPR 05
PTBR_GPR_GPR06 DS 1FL4   GPR 06
PTBR_GPR_GPR07 DS 1FL4   GPR 07
PTBR_GPR_GPR08 DS 1FL4   GPR 08
PTBR_GPR_GPR09 DS 1FL4   GPR 09
PTBR_GPR_GPR10 DS 1FL4   GPR 10
PTBR_GPR_GPR11 DS 1FL4   GPR 11
PTBR_GPR_GPR12 DS 1FL4   GPR 12
PTBR_GPR_GPR13 DS 1FL4   GPR 13
PTBR_GPR_GPR14 DS 1FL4   GPR 14
PTBR_GPR_GPR15 DS 1FL4   GPR 15
PTBR_GPR_CRS DS 0CL0064  Control registers. May be read but will not   X
                         be written
PTBR_GPR_CR00 DS 1FL4    CR 00
PTBR_GPR_CR01 DS 1FL4    CR 01
PTBR_GPR_CR02 DS 1FL4    CR 02
PTBR_GPR_CR03 DS 1FL4    CR 03
PTBR_GPR_CR04 DS 1FL4    CR 04
PTBR_GPR_CR05 DS 1FL4    CR 05
PTBR_GPR_CR06 DS 1FL4    CR 06
PTBR_GPR_CR07 DS 1FL4    CR 07
PTBR_GPR_CR08 DS 1FL4    CR 08
PTBR_GPR_CR09 DS 1FL4    CR 09
PTBR_GPR_CR10 DS 1FL4    CR 10
PTBR_GPR_CR11 DS 1FL4    CR 11
PTBR_GPR_CR12 DS 1FL4    CR 12
PTBR_GPR_CR13 DS 1FL4    CR 13
PTBR_GPR_CR14 DS 1FL4    CR 14
PTBR_GPR_CR15 DS 1FL4    CR 15
PTBR_GPR_PSW DS 0CL0008  PSW. May be read but only the rightmost 4     X
                         bytes (word 2) will be written
PTBR_GPR_PSW_W1 DS 1CL0004 PSW word 1.
PTBR_GPR_PSW_W2 DS 1CL0004 PSW word 2
PTBR_GPR_LEN EQU *-PTBR_GPR
*
* ********************************************************************
* * Structure for PT_Read_FPR and PT_Write_FPR.                      *
* ********************************************************************
*
*
```

```
PTBR_FPR DSECT
PTBR_FPR_CNTLFPR DS 0BL2 Only used on write request
PTBR_FPR_CNTLFPR1 DS 0BL1
PTBR_FPR_WFPR0 EQU X'80' Write content of FPR 0
PTBR_FPR_WFPR1 EQU X'40' Write content of FPR 1
PTBR_FPR_WFPR2 EQU X'20' Write content of FPR 2
PTBR_FPR_WFPR3 EQU X'10' Write content of FPR 3
PTBR_FPR_WFPR4 EQU X'08' Write content of FPR 4
PTBR_FPR_WFPR5 EQU X'04' Write content of FPR 5
PTBR_FPR_WFPR6 EQU X'02' Write content of FPR 6
PTBR_FPR_WFPR7 EQU X'01' Write content of FPR 7
         ORG   PTBR_FPR_CNTLFPR1+X'00000001'
PTBR_FPR_CNTLFPR2 DS 0BL1
PTBR_FPR_WFPR8 EQU X'80' Write content of FPR 8
PTBR_FPR_WFPR9 EQU X'40' Write content of FPR 9
PTBR_FPR_WFPR10 EQU X'20' Write content of FPR 10
PTBR_FPR_WFPR11 EQU X'10' Write content of FPR 11
PTBR_FPR_WFPR12 EQU X'08' Write content of FPR 12
PTBR_FPR_WFPR13 EQU X'04' Write content of FPR 13
PTBR_FPR_WFPR14 EQU X'02' Write content of FPR 14
PTBR_FPR_WFPR15 EQU X'01' Write content of FPR 15
         ORG   PTBR_FPR_CNTLFPR+X'00000002'
PTBR_FPR_CNTLMISC DS 0BL2 Only used on write request
PTBR_FPR_WFPC EQU X'80'  Write content of FPC
         ORG   PTBR_FPR_CNTLMISC+X'00000002'
         DS    1CL0012   Reserved
PTBR_FPR_FPRS DS 0CL0128 Floating point registers
PTBR_FPR_FPR00 DS 1CL0008 FPR 00
PTBR_FPR_FPR01 DS 1CL0008 FPR 01
PTBR_FPR_FPR02 DS 1CL0008 FPR 02
PTBR_FPR_FPR03 DS 1CL0008 FPR 03
PTBR_FPR_FPR04 DS 1CL0008 FPR 04
PTBR_FPR_FPR05 DS 1CL0008 FPR 05
PTBR_FPR_FPR06 DS 1CL0008 FPR 06
PTBR_FPR_FPR07 DS 1CL0008 FPR 07
PTBR_FPR_FPR08 DS 1CL0008 FPR 08
PTBR_FPR_FPR09 DS 1CL0008 FPR 09
PTBR_FPR_FPR10 DS 1CL0008 FPR 10
PTBR_FPR_FPR11 DS 1CL0008 FPR 11
PTBR_FPR_FPR12 DS 1CL0008 FPR 12
PTBR_FPR_FPR13 DS 1CL0008 FPR 13
PTBR_FPR_FPR14 DS 1CL0008 FPR 14
PTBR_FPR_FPR15 DS 1CL0008 FPR 15
PTBR_FPR_FPC DS 1CL0004  Floating Point Control Register
PTBR_FPR_LEN EQU *-PTBR_FPR
*
* ********************************************************************
* * Structure for PT_Read_GPRH PT_Write_GPRH.
* ********************************************************************
*
*
PTBR_GPRH DSECT
PTBR_GPRH_CNTLGPRH DS 0BL2 Only used on write request
PTBR_GPRH_CNTLGPRH1 DS 0BL1
PTBR_GPRH_WGPRH0 EQU X'80' Write content of GPRH 0
PTBR_GPRH_WGPRH1 EQU X'40' Write content of GPRH 1
PTBR_GPRH_WGPRH2 EQU X'20' Write content of GPRH 2
PTBR_GPRH_WGPRH3 EQU X'10' Write content of GPRH 3
PTBR_GPRH_WGPRH4 EQU X'08' Write content of GPRH 4
PTBR_GPRH_WGPRH5 EQU X'04' Write content of GPRH 5
PTBR_GPRH_WGPRH6 EQU X'02' Write content of GPRH 6
PTBR_GPRH_WGPRH7 EQU X'01' Write content of GPRH 7
         ORG   PTBR_GPRH_CNTLGPRH1+X'00000001'
PTBR_GPRH_CNTLGPRH2 DS 0BL1
PTBR_GPRH_WGPRH8 EQU X'80' Write content of GPRH 8
PTBR_GPRH_WGPRH9 EQU X'40' Write content of GPRH 9
PTBR_GPRH_WGPRH10 EQU X'20' Write content of GPRH10
```

```
PTBR_GPRH_WGPRH11 EQU X'10' Write content of GPRH11
PTBR_GPRH_WGPRH12 EQU X'08' Write content of GPRH12
PTBR_GPRH_WGPRH13 EQU X'04' Write content of GPRH13
PTBR_GPRH_WGPRH14 EQU X'02' Write content of GPRH14
PTBR_GPRH_WGPRH15 EQU X'01' Write content of GPRH15
         ORG   PTBR_GPRH_CNTLGPRH+X'00000002'
PTBR_GPRH_CNTLMISC DS 1BL2 Only used on write request
         DS    1CL0012   Reserved
PTBR_GPRH_GPRHS DS 0CL0064 GP High registers.
PTBR_GPRH_GPRH00 DS 1FL4 GPRH 00
PTBR_GPRH_GPRH01 DS 1FL4 GPRH 01
PTBR_GPRH_GPRH02 DS 1FL4 GPRH 02
PTBR_GPRH_GPRH03 DS 1FL4 GPRH 03
PTBR_GPRH_GPRH04 DS 1FL4 GPRH 04
PTBR_GPRH_GPRH05 DS 1FL4 GPRH 05
PTBR_GPRH_GPRH06 DS 1FL4 GPRH 06
PTBR_GPRH_GPRH07 DS 1FL4 GPRH 07
PTBR_GPRH_GPRH08 DS 1FL4 GPRH 08
PTBR_GPRH_GPRH09 DS 1FL4 GPRH 09
PTBR_GPRH_GPRH10 DS 1FL4 GPRH 10
PTBR_GPRH_GPRH11 DS 1FL4 GPRH 11
PTBR_GPRH_GPRH12 DS 1FL4 GPRH 12
PTBR_GPRH_GPRH13 DS 1FL4 GPRH 13
PTBR_GPRH_GPRH14 DS 1FL4 GPRH 14
PTBR_GPRH_GPRH15 DS 1FL4 GPRH 15
         DS    1CL0008   Reserved
PTBR_GPRH_LEN EQU *-PTBR_GPRH
*
* ********************************************************************
* * Structure for PT_Read_Block and PT_Write_Block.                  *
* ********************************************************************
*
*
PTBR_BLOCK DSECT
PTBR_BLOCK_AADDR DS 1AL4 address of area to read
PTBR_BLOCK_ALEN DS 1FL4  length of area to read
         DS    1CL0008   Reserved
PTBR_BLOCK_BUF DS 0C     area to read into or write from. Must be at   X
                         least PtBR_Block_ALen bytes large
PTBR_BLOCK_LEN EQU *-PTBR_BLOCK
PTBR_BLOCK64 DSECT
PTBR_BLOCK_AADDR64 DS AD   address of area to read
PTBR_BLOCK_ALEN64 DS 1FL4  length of area to read
         DS    1CL0004   Reserved
PTBR_BLOCK_BUF64 DS 0C   area to read into or write from. Must be at   X
                         least PtBR_Block_ALen bytes large
PTBR_BLOCK_LEN64 EQU *-PTBR_BLOCK64
*
* ********************************************************************
* * Structure for PT_Read_D and PT_Write_D.                          *
* ********************************************************************
*
*
PTBR_D   DSECT
PTBR_D_WORDPTR DS 1AL4   Address of fullword of data
PTBR_D_WORD DS 1FL4      fullword of data at specified address for a   X
                         read request or the data to be written to the X
                         specified address for a write request
PTBR_D_LEN EQU *-PTBR_D
PTBR_D64 DSECT
PTBR_D_WORDPTR64 DS AD   Address of fullword of data
PTBR_D_WORD64 DS 1FL4    fullword of data at specified address for a   X
                         read request or the data to be written to the X
                         specified address for a write request
PTBR_D_LEN64 EQU *-PTBR_D64
*
* ********************************************************************
```

```
* * Structure for PT_Read_I and PT_Write_I.                          *
* ********************************************************************
*
*
PTBR_I   DSECT
PTBR_I_WORDPTR DS 1AL4   Address of fullword of program data
PTBR_I_WORD DS 1FL4      fullword of program data at specified address X
                         for a read request or the program data to be  X
                         written to the specified address for a write  X
                         request
PTBR_I_LEN EQU *-PTBR_I
PTBR_I64 DSECT
PTBR_I_WORDPTR64 DS AD   Address of fullword of program data
PTBR_I_WORD64 DS 1FL4    fullword of program data at specified address X
                         for a read request or the program data to be  X
                         written to the specified address for a write  X
                         request
PTBR_I_LEN64 EQU *-PTBR_I64
*
* ********************************************************************
* * Structure for PT_Read_U.                                         *
* ********************************************************************
*
*
PTBR_U   DSECT
PTBR_U_NUMOFFSETS DS 1FL4 Number of entries in offset/control word     X
                         array
         DS    1CL0004   Reserved
PTBR_U_OWARRAY DS 0CL0008 Array of offsets and control words
PTBR_U_OFFSET DS 1FL4    Offset of fullword of control information
PTBR_U_WORD DS 1FL4      fullword of control information from user     X
                         area in the debugged process
PTBR_U_LEN EQU *-PTBR_U
```

BPXYPTXL — Map the parameter list for pthread_create

AMODE 64 callers use "BPXYPTXL — Map the parameter pist for pthread_create" on page 1110.

```
           BPXYPTXL   ,
** BPXYPTXL: Pthread Parameter List
**  Used By: PTX
PTXL                DSECT ,     Parameter List returned by BPX1PTX
PTXLWORKAREAPTR     DS    A     Pointer to User Work Area
PTXLATTRIBUTEPTR    DS    A     Pointer to User Attributes
PTXLTHIDPTR         DS    A     Pointer to Thread ID
PTXLSTATUSPTR       DS    A     Pointer to Thread Run Status
PTXL#LENGTH         EQU   *-PTXL
PTXLRS              DSECT ,     Thread Run Status
                    DS    0F
PTXLRSFLAGS         DS    0BL4  Thread Run Status Flags
PTXLRSFLAGS0        DS    B     1st byte
PTXLRSREADY         EQU   X'80'   Thread is ready to run
PTXLRSFLAGS1        DS    B     2nd byte
PTXLRSFLAGS2        DS    B     3rd byte
PTXLRSFLAGS3        DS    B     4th byte
PTXLRS#LENGTH       EQU   *-PTXLRS
** BPXYPTXL End
```

BPXYRFIS — Map the register file interest structures

```
           BPXYRFIS   ,
** BPXYRFIS
*
```

```
*  Register File Interest Structure
*
RFIS                 DSECT ,
RFIS_CMD             DS    H           CMD = REG OR UNREG
RFIS_FLAGS           DS    0H          FLAGS
RFIS_FLAGS1          DS    XL1          FLAGS Byte 1
RFIS_LOSTMSGS        EQU   X'80'         MSGSND HAS FAILED
RFIS_FLAGS2          DS    XL1          FLAGS Byte 2
RFIS_RFTOK           DS    CL8         SYSTEM REGISTERED FILE TOKEN
RFIS_QID             DS    F             IPC MSG QUEUE ID
RFIS_TYPE            DS    F             IPC MSG TYPE
RFIS_UTOK            DS    CL8           IPC MSG USER TOKEN
*
RFIS#LENGTH          EQU   *-RFIS Length of this structure
*
** RFIS_CMD Values
RFIS#REG             EQU   1
RFIS#UNREG           EQU   2
*
*  Registered File invalidate Message
*
RFIM                 DSECT ,
RFIM_TYPE            DS    F        IPC MSG TYPE, FROM RFIS_TYPE
RFIM_TEXT            DS    0CL12    IPC MSG TEXT:
RFIM_UTOK            DS    CL8        USER TOKEN, FROM RFIS_UTOK
RFIM_EVENT           DS    H          CHANGE EVENT
RFIM_FLAGS           DS    0H         FLAGS
RFIM_FLAGS1          DS    XL1         FLAGS Byte 1
RFIM_LOSTMSGS        EQU   X'80'         MSGSND HAS FAILED
RFIM_FLAGS2          DS    XL1         FLAGS Byte 2
*
RFIM#LENGTH          EQU   *-RFIM Length of this structure
*
** RFIM_EVENT Values
RFIM#WRITE           EQU   1         WRITE, TRUNC, OPEN(O_TRUNC)
RFIM#ATTR            EQU   2         ANY ATTR CHANGE, CHMOD, ETC.
RFIM#UNLINK          EQU   3         ANY NAME UNLINKED
RFIM#RENAME          EQU   4         ANY NAME RENAMED
RFIM#UNMOUNT         EQU   5         CONTAINING FILE SYS UNMNTED
*
** BPXYRFIS End
```

BPXYRLIM — Map the rlimit, rusage, and timeval structures

AMODE 64 callers use “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1110.

```
           BPXYRLIM   ,
** BPXYRLIM: Rlimit, Timeval, and Rusage Structures
**  Used By: setrlimit, getrlimit, and getrusage
RLIMIT               DSECT ,      Rlimit structure
RLIM_CUR_DW          DS    0CL8   Current limit (doubleword)
RLIM_CUR_HW          DS    F      Current (soft) limit highword -      X
                                    used only for RLIMIT_FSIZE        X
                                    and RLIMIT_MEMLIMIT, it is        X
                                    ignored for all other resources
RLIM_CUR             DS    0F     Current (soft) limit lowword
RLIM_CUR_LW          DS    F      Current (soft) limit lowword
RLIM_MAX_DW          DS    0CL8   Current limit (doubleword)
RLIM_MAX_HW          DS    F      Current (hard) limit highword -      X
                                    used only for RLIMIT_FSIZE        X
                                    and RLIMIT_MEMLIMIT, it is        X
                                    ignored for all other resources
RLIM_MAX             DS    0F     Maximum (hard) limit lowword
RLIM_MAX_LW          DS    F      Maximum (hard) limit lowword
RLIMIT#LENGTH        EQU *-RLIMIT Length of this DSECT
```

```
TIMEVAL             DSECT ,      Timeval structure
TMVL_SEC            DS    F      Seconds
TMVL_USEC           DS    F      Microseconds
TIMEVAL#LENGTH      EQU *-TIMEVAL Length of this DSECT
RUSAGE              DSECT ,      Rusage structure
RU_UTIME            DS    CL(TIMEVAL#LENGTH) User time used
RU_STIME            DS    CL(TIMEVAL#LENGTH) System time used
RUSAGE#LENGTH       EQU *-RUSAGE Length of this DSECT
** BPXYRLIM End
```

BPXYRMON — Map resource monitor data

```
           BPXYRMON   ,
** BPXYRMON: Resource monitor data mapping
**  Used By: RMG
RMON                DSECT ,
RMONID              DC    C'RMON'  Eye catcher
RMONLENGTH          DC    A(RMON#LENGTH)  Length of this structure
RMONSYSCALLS        DS    F      Total  Syscalls.         This
*                                includes syscalls done internally
*                                by the kernel.  It does not include
*                                all trivial syscalls.
RMONCPUTIME         DS    F      Total CPU time spent in
*                                kernel (Hundredths of a second)
RMONOVERRUN         DS    0CL12
RMONOVRPROC         DS    F      Count of times the maximum number
*                                of processes was exceeded.
RMONOVRUID          DS    F      Count of times the maximum number
*                                of active UIDs was exceeded.
RMONOVRPRUID        DS    F      Count of times the maximum number
*                                of processes per UID was exceeded.
RMONLIMITS          DS    0CL6
RMONMAXPROC         DS    H      Maximum number of processes
RMONMAXUID          DS    H      Maximum number of active UIDs
RMONMAXPRUID        DS    H      Maximum number of processes per UID
RMONCURRENT         DS    0CL6
RMONNUMPROC         DS    H      Current number of processes
RMONNUMUID          DS    H      Current number of active UIDs
                    DS    H      Reserved
RMONOVERRUNIPC      DS    0CL16
RMONOVRIPCMSGNIDS   DS    F      Number of attempts to exceed
*                                maximum number of message queue
*                                IDs
RMONOVRIPCSEMNIDS   DS    F      Number of attempts to exceed
*                                maximum number of semaphore
*                                IDs
RMONOVRIPCSHMNIDS   DS    F      Number of attempts to exceed
*                                maximum number of shared memory
*                                IDs
RMONOVRIPCSHMSPGS   DS    F      Number of attempts to exceed
*                                maximum number of shared memory
*                                pages for all segments
RMONLIMITSIPC       DS    0CL16
RMONMAXIPCMSGNIDS   DS    F      Maximum number of message queue
*                                IDs
RMONMAXIPCSEMNIDS   DS    F      Maximum number of semaphore
*                                IDs
RMONMAXIPCSHMNIDS   DS    F      Maximum number of shared memory
*                                IDs
RMONMAXIPCSHMSPGS   DS    F      Maximum number of shared memory
*                                pages for all segments
RMONCURRENTIPC      DS    0CL16
RMONNUMIPCMSGNIDS   DS    F      Current number of message queue
*                                IDs
RMONNUMIPCSEMNIDS   DS    F      Current number of semaphore
*                                IDs
```

```
RMONNUMIPCSHMNIDS    DS    F      Current number of shared memory
*                                 IDs
RMONNUMIPCSHMSPGS    DS    F      Current number of shared memory
*                                 pages for all segments
RMONOVRMMAPAREA      DS    F      Number of attempts to exceed
*                                 maximum number of mmap storage
*                                 pages
RMONMAXMMAPAREA      DS    F      Maximum number of mmap storage
*                                 pages
RMONNUMMMAPPAGES     DS    F      Current number of mmap storage
*                                 pages (in use)
RMONMAXSHRPAGES      DS    F      Maximum number of shared storage
*                                 pages as specified by BPXPRMXX
*                                 parmlib statement MAXSHAREPAGES
RMONNUMSHRPAGES      DS    F      Current number of shared storage
*                                 pages
RMONOVRSHRPAGES      DS    F      Number of attempts to exceed
*                                 maximum number of shared storage
*                                 pages
RMONMAXSHRLIBRGN     DS    F      Maximum amount of storage available
*                                 for shared library region as
*                                 specified by parmlib statement
*                                 SHRLIBRGNSIZE in megabytes
RMONCURSHRLIBRGN     DS    F      Current amount of shared library
*                                 storage allocated in megabytes
RMONOVRSHRLIBRGN     DS    F      Number of attempts to exceed maximum
*                                 storage amount for shared library
*                                 region
RMONMAXQUEUEDSIGS    DS    F      Maximum amount of queued signals
*                                 allowed per process as specified
*                                 by parmlib statement
*                                 MAXQUEUEDSIGS
RMONOVRQUEUEDSIGS    DS    F      Number of attempts to exceed maximum
*                                 number of queued signals
RMON#LENGTH          EQU   *-RMON Length of RMON
** BPXYRMON End
```

BPXYSECI — Map the output of BPX1IOC for the SECIGET request

```
            BPXYSECI   ,
** BPXYSECI: Socket Peer Security Identifiers
**  Used By: IOC
SECI                 DSECT ,
SECIUSERID           DS    CL8    MVS User ID
SECIEUID             DS    F      Effective UID
SECIEGID             DS    F      Effective GID
SECI#LENGTH          EQU   *-SECI Length of this area
** BPXYSECI End
```

BPXYSECO — Map the input/output of BPX1IOC for the SIOCSECENVR request

```
            BPXYSECO   ,
** BPXYSECO: Security Environment Object
** Used By: IOC
SECO                 DSECT ,
SECO_ARGUMENT        DS  F       Input: SET / GET argument.
SECO_ENVR_OBJECT     DS  0CL14   GET Output: Security ENVR OBJECT:
SECO_OBJECTLEN       DS  F       GET Output: ENVR Object length.
SECO_BUFFERLEN       DS  F       GET Input/Output: Buffer Length.
SECO_BUFFERADDR      DS  A       GET Input/Output: Buffer Address.
```

```
SECO_BUFFERSP        DS  X       GET Input/Output: Buffer SubPool.
SECO_BUFFERKEY       DS  X       GET Input/Output: Buffer Key.
SECO#LENGTH          EQU *-SECO  Length of this area
** BPXYSECO End
```

BPXYSECT — Map the output of BPX1IOC for the SECIGET_T request

```
            BPXYSECT   ,
** BPXYSECT: Socket Peer Security Identifiers
**  Used By: IOC
SECT                  DSECT ,
SECTPUSERID           DS    CL8    Process MVS User ID
SECTPEUID             DS    F      Process Effective UID
SECTPEGID             DS    F      Process Effective GID
SECTPUSERIDLEN        DS    F      Process MVS User ID Length
SECTTUSERID           DS    CL8    Task    MVS User ID
SECTTEUID             DS    F      Task    Effective UID
SECTTEGID             DS    F      Task    Effective GID
SECTTUSERIDLEN        DS    F      Task    MVS User ID Length
SECT#LENGTH           EQU   *-SECT Length of this area
** BPXYSECT End
```

BPXYSEEK — Constants for lseek

BPXYSEEK is composed only of EQUates. DSECT= is allowed but ignored.

```
            BPXYSEEK   ,
** BPXYSEEK: Lseek constant definitions
**  Used By: LSK
SEEK_SET              EQU   0    Set file offset to offset
SEEK_CUR              EQU   1    Set file offset to current + offset
SEEK_END              EQU   2    Set file offset to EOF + offset
** BPXYSEEK End
```

BPXYSEL — Map the select options

BPXYSEL contains the read, write and exception options for the select system call.

```
            BPXYSEL    ,
** BPXYSEL: Select Options
**  Used By: SEL
SEL                   DSECT ,
SELBEGIN              DS    0F
*
SELBITS               DS    0XL4    Flag Bits.8F FF FF FF Reserved
SELPOLLFLAGS          DS    XL2     Select flags / Poll (r)events
*-------------------------------------------------------
* Select flags
*-------------------------------------------------------
SELFLAGS              ORG   SELPOLLFLAGS
                      DS    XL1
*                     EQU   X'80'   Never use this bit
SELREAD               EQU   X'40'   Descriptor ready for read.
SELWRITE              EQU   X'20'   Descriptor ready for write.
SELXCEPT              EQU   X'10'   Descriptor ready for exception.
                      DS    XL1     Available byte
*-------------------------------------------------------
* Poll Events/Returned Events
*-------------------------------------------------------
SELPOLLEVENTS         ORG   SELPOLLFLAGS
                      DS    XL2     Mapped by PollEvents(BPXYPOLL)
SELPOLLREVENTS        ORG   SELPOLLFLAGS
                      DS    XL2     Mapped by PollRevents(BPXYPOLL)
```

```
*
                    DS    XL1     Available byte
                    DS    XL1     Reserved for internal use
*
*   Constants
*
SEL#LENGTH          EQU   *-SEL   Length of SEL
SEL#QUERY           EQU   1       Query function
SEL#CANCEL          EQU   2       Cancel function
SEL#BATSELQ         EQU   3       Batch-Select Query function
SEL#BATSELC         EQU   4       Batch-Select Cancel function
SEL#POLLQUERY       EQU   5       Poll Query function
SEL#BATPOLLQ        EQU   6       Batch-Poll Query function
SEL#BATPOLLC        EQU   7       Batch-Poll Cancel function
SEL#POLLCANCEL      EQU   8       Poll Cancel function
SEL#BITSBACKWARD    EQU   0       Bit Backward Order by word
SEL#BITSFORWARD     EQU   1       Bit Forward Order by word
SEL#TYPES           EQU   3       3 TYPES (Read Write Except)
SEL#RBIT            EQU   64      Read bit position in byte
SEL#WBIT            EQU   32      Write bit position in byte
SEL#XBIT            EQU   16      Xcept bit position in byte
** BPXYSEL End
```

BPXYSELT — Map the timeout value for the select syscall

AMODE 64 callers use "BPXYSELT — Map the timeout value for the select syscall" on page 1111.

```
           BPXYSELT   ,
** BPXYSELT: Select Time Structure
**  Used By: Select Syscall
SELT                DSECT ,
SELTBEGIN           DS    0D
*-------------------------31-bit format
*
TV_SEC              DS    F'0'    Seconds
TV_USEC             DS    F'0'    Microseconds
*   Constants
*
SELT#LENGTH         EQU   *-SELT Length of SELT
** BPXYSELT End
```

BPXYSEM — Map interprocess communication semaphores

DSECTs (SEMID_DS, SEM_ARRAY and SEM_BUF_ELE) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use "BPXYSEM — Map interprocess communication semaphores" on page 1111.

```
           BPXYSEM    ,
** BPXYSEM: Interprocess Communications Permission
**  Used By: XSO, XSC
SEMID_DS            DSECT ,      semctl structure
SEM_PERM            DS    CL(IPC#LENGTH)   Mapped by BPXYIPCP
SEM_NSEMS           DS    H      number of semaphores in set
                    DS    H      spacer
SEM_OTIME           DS    FL4    last semop() time
SEM_CTIME           DS    FL4    last time changed by semctl()
SEM#LENGTH          EQU   *-SEMID_DS   Length of this DSECT
* SETVAL - a one element array for Semaphore_Number
* SETALL, GETALL - an array with Number_of_Semaphore elements
SEM_ARRAY           DSECT ,      SETALL, GETALL, SETVAL
```

```
SEM_ARRAY_VAL        DS    FL2    semaphore value
SEM_BUF_ELE          DSECT ,      sembuf element - semop
SEM_NUM              DS    FL2    semaphore number (0 to n-1)
SEM_OP               DS    FL2    semaphore operation
SEM_FLG              DS    H      operation flags
SEM#BUFLEN           EQU   *-SEM_BUF_ELE
* Flag bits - semop (also IPC_NOWAIT
SEM_UNDO             EQU   2      Set up adjust on exit entry.
* Control Commands - (also IPC_RMID, IPC_SET, IPC_STAT):
SEM_GETVAL           EQU   21     Get the current semaphore value
SEM_SETVAL           EQU   22     Change the semaphore value
SEM_GETPID           EQU   23     Get PID of last process to alter sem
SEM_GETNCNT          EQU   24     Get count of tasks waiting for val>0
SEM_GETZCNT          EQU   25     Get count of tasks waiting for val=0
SEM_GETALL           EQU   26     Get the current semaphore values
SEM_SETALL           EQU   27     Change the semaphore values
* Maximum and minimum values
SEM#MAX_VAL          EQU   32767  Maximum sem_val (min = 0)
SEM#MAX_ADJ          EQU   16383  Maximum sem_adj (min = -MAX)
** BPXYSEM End
```

BPXYSFDL — Map the server file descriptor list structure

The mapping macro only provides enough space for one file descriptor; follow the invocation with up to 63 additional words.

```
           BPXYSFDL   ,
** BPXYSFDL: Dile descriptor List
**  Used By: SPW
SFDL         DSECT ,
SFDLHEADER   DS  0CL8
SFDLCOUNT    DS  F     Number of entries in this file descriptor list
SFDLFLAGS    DS  F     Flags
SFDLCLOSE    EQU X'80' All files to be closed (Bit 0 of SFDLFLAGS)
SFDLDESC     DS  F     First FD(follow by COUNT-1 additional FDs)
SFDLMAXCOUNT EQU 64    Maximum value for SFDLCOUNT
SFDL_LEN     EQU *-SFDL
** BPXYSFDL End
```

BPXYSFPL — Map the send_file parameter list

AMODE 64 callers use “BPXYSFPL — Map the send_file parameter list” on page 1112.

```
           BPXYSFPL   ,
** BPXYSFPL: SFPL system call structure
**  Used By: BPX1SF
SFPL                 DSECT ,
SFSOCKETDES          DS    F         Socket Descriptor
SFHEADERLEN          DS    F         Header Length
SFHEADERVPTR         DS    0F
SFHEADERALET         DS    F         Header Alet
SFHEADERPTR          DS    F         31-bit Header Ptr
SFFILEDES            DS    F         File Descriptor
SFFILEBYTESDW        DS    0F        Bytes to send Double Word (-1=all)
SFFILEBYTESH         DS    F          High Word
SFFILEBYTESL         DS    F          Low  Word
SFFILEOFFSETDW       DS    0F        Offset Double Word
SFFILEOFFSETH        DS    F          High Word
SFFILEOFFSETL        DS    F          Low  Word
SFFILESIZEDW         DS    0F        File Size Double Word
SFFILESIZEH          DS    F          High Word
SFFILESIZEL          DS    F          Low  Word
SFTRAILERLEN         DS    F         Trailer Length
```

```
SFTRAILERVPTR        DS    0F
SFTRAILERALET        DS    F          Trailer Alet
SFTRAILERPTR         DS    F         31-bit Trailer Ptr
SFBYTESSENTDW        DS    0F        Bytes Sent Double Word
SFBYTESSENTH         DS    F          High Word
SFBYTESSENTL         DS    F          Low  Word
SFFLAGS              DS    0XL4      Control Flags
SFPLVERSION          DS    XL1        Version
SFFLAGBYTE2          DS    XL1       Reserved
SFFLAGBYTE3          DS    XL1       Reserved
SFFLAGBYTE4          DS    XL1       Flags
SF_CLOSE             EQU   2         Close Socket Descriptor
SF_REUSE             EQU   1         Reuse Socket Descriptor
SFRESERVE            DS    CL12      Reserved
*
SFPLEND              EQU   *
*
SFPL#LENGTH          EQU   SFPLEND-SFPL
*
*  Constants
*
*
** BPXYSFPL End
```

BPXYSHM—Map interprocess communication shared memory segments

AMODE 64 callers use "BPXYSHM—Map interprocess communication shared memory segments" on page 1113.

```
           BPXYSHM    ,
** BPXYSHM: Interprocess Communications Permission
**  Used By: XMC
SHMID_DS             DSECT ,      SHMID_DS - shmctl structure
SHM_PERM             DS    CL(IPC#LENGTH)   Mapped by BPXYIPC
SHM_SEGSZ            DS    F      size of segment in bytes
SHM_LPID             DS    F      process ID of last operation
SHM_CPID             DS    F      process ID of creator
SHM_NATTCH           DS    F      number of current attaches
SHM_ATIME            DS    F      time of last shmat
SHM_DTIME            DS    F      time of last shmdt
SHM_CTIME            DS    F      time of last change shmget/shmctl
* Mode bits (mapped over S_TYPE in BPXYMODE):
SHM_RDONLY           EQU   1      Attach read-only (else read-write)
SHM_RND              EQU   2      Round attach address to SHMLBA
SHMLBA               EQU   4096   Rounding boundary
SHM#LENGTH           EQU   *-SHMID_DS   Length of this DSECT
** BPXYSHM End
```

BPXYSIGH — Signal constants

BPXYSIGH is composed of only EQUates. DSECT= is allowed but ignored.

```
           BPXYSIGH   ,
** BPXYSIGH: Component signal definition
**  Used By: KIL SIA SPM
*********************************************************************
* Signals with default action ABNORMAL TERMINATION
SIGHUP#    EQU   1   Hangup detected on controlling terminal
SIGINT#    EQU   2   Interactive attention
SIGABRT#   EQU   3   Abnormal termination
SIGILL#    EQU   4   Detection of an incorrect hardware instruction
SIGPOLL#   EQU   5   Pollable event
SIGURG#    EQU   6   High bandwidth data is available at a socket
```

```
SIGFPE#    EQU   8   Erroneous arithmetic operation, such as division
*                    by zero of an operation resulting in overflow
SIGKILL#   EQU   9   Termination (cannot be caught or ignored)
SIGBUS#    EQU  10   Bus error
SIGSEGV#   EQU  11   Detection of an incorrect memory reference
SIGSYS#    EQU  12   Bad System Call
SIGPIPE#   EQU  13   Write on a pipe with no readers
SIGALRM#   EQU  14   Timeout
SIGTERM#   EQU  15   Termination
SIGUSR1#   EQU  16   Reserved as application-defined signal 1
SIGUSR2#   EQU  17   Reserved as application-defined signal 2
SIGABND#   EQU  18   Abend
SIGQUIT#   EQU  24   Interactive termination
SIGTRAP#   EQU  26   Trap used by the ptrace call
SIGXCPU#   EQU  29   CPU time limit exceeded
SIGXFSZ#   EQU  30   File size limit exceeded
SIGVTALRM# EQU  31   Virtual timer expired
SIGPROF#   EQU  32   Profiling timer expired
SIGDANGER# EQU  33   Shutdown Imminent
* Signals with default action of CONTINUE
* Signals with default action IGNORE THE SIGNAL
SIGNULL#   EQU   0   Null - no signal sent
SIGCHLD#   EQU  20   Child process terminated or stopped
SIGIO#     EQU  23   Completion of input or output
SIGIOER#   EQU  27   Input or Output Error
SIGWINCH#  EQU  28   Change size of window
SIGTRACE#  EQU  37   Trace the target process
SIGDUMP#   EQU  39   Take a SYSMDUMP
* Signals with default action STOP
SIGSTOP#   EQU   7   Stop (cannot be caught or ignored)
SIGTTIN#   EQU  21   Read from a control terminal attempted by a
*                    member of a background process group
SIGTTOU#   EQU  22   Write from a control terminal attempted by a
*                    member of a background process group
SIGTSTP#   EQU  25   Interactive stop
SIGTHSTOP# EQU  34   Thread stop (cannot be caught or blocked or
*                    ignored)
* Signals with default action CONTINUE IF IT IS CURRENTLY STOPPED,
*                             OTHERWISE IGNORE THE SIGNAL
SIGCONT#   EQU  19   Continue if stopped
SIGTHCONT# EQU  35   Thread continue (cannot be caught or blocked or
*                    ignored)
*********************************************************************
** Equates   that define sa_handler values on Sigaction()
*********************************************************************
SIG_DFL#   EQU   0   Default signal action
SIG_IGN#   EQU   1   Ignore signal action
*********************************************************************
** Constants that define sa_flags   values on Sigaction()
*********************************************************************
SA_FLAGS_DFT#  EQU X'00000000'  Default sa_flags
SA_NOCLDSTOP#  EQU X'80000000'  No SIGCHLD when children stop
SA_OLD_STYLE#  EQU X'40000000'  Old style signal() function
SA_ONSTACK#    EQU X'20000000'  Deliver on alternate stack
SA_RESETHAND#  EQU X'10000000'  Reset action on delivery
SA_RESTART#    EQU X'08000000'  Restart interruptible funcs
SA_SIGINFO#    EQU X'04000000'  Pass siginfo to catcher
SA_NOCLDWAIT#  EQU X'02000000'  Don't create zombie on exit
SA_NODEFER#    EQU X'01000000'  Don't block signal on delivery
SA_IGNORE#     EQU X'00000001'  Act as though sa_handler contained
*                               SIG_IGN#
*********************************************************************
** Constants that define how parameter on sigprocmask()
*********************************************************************
SIG_BLOCK#      EQU    0  Block signals set on in New_signal_mask
SIG_UNBLOCK#    EQU    1  Unblock signals set on in New_signal_mask
SIG_SETMASK#    EQU    2  Set signal mask to New_signal_mask
```

```
*********************************************************************
** Constants that define the lower two bytes of the Signal_Options  *
** on the BPX1KIL and BPX1PTK syscalls.  If a signal generated with *
** one or more of these flags is handled by the Signal Interface    *
** Routine, the flags will appear in the PpsdKilOpts field upon     *
** delivery of said signal.
** When the lower two bytes contain x'1000' (SIG_CONSCANCEL#) the   *
** upper two bytes will contain the SIGCNCL type qualifier
*********************************************************************
SIG_FLAGS_DFT#     EQU  X'0000' Default options
SIG_PTRACEBYPASS#  EQU  X'8000' Bypass ptrace processing
SIG_KERNSICODE#    EQU  X'4000' z/OS UNIX kernel set si_code
SIG_APPLSICODE#    EQU  X'2000' Application set si_code
SIG_CONSCANCEL#    EQU  X'1000' Console (MODIFY) cancel thread
*********************************************************************
** Constants that define si_codes which are passed in the upper two *
** bytes of the Signal_Options on the BPX1KIL and BPX1PTK syscalls  *
** If a signal generated with a si_code is handled by the Signal    *
** Interface Routine the si_code will appear in the PpsdKilData     *
** field upon delivery of said signal.
*********************************************************************
ILL_ILLOPC#     EQU   11  Illegal opcode
ILL_ILLOPN#     EQU   12  Illegal operand
ILL_ILLADR#     EQU   13  Illegal addressing mode
ILL_ILLTRP#     EQU   14  Illegal trap
ILL_PRVOPC#     EQU   15  Privileged opcode
ILL_PRVREG#     EQU   16  Privileged register
ILL_COPROC#     EQU   17  Coprocessor error
ILL_BADSTK#     EQU   18  Internal stack error
ILL_EXECUTE#    EQU   19  Execute exception
ILL_ILLSPEC#    EQU   20  Specification exception
*********************************************************************
FPE_INTDIV#     EQU   31  Integer divide by zero
FPE_INTOVF#     EQU   32  Integer overflow
FPE_FLTDIV#     EQU   33  Floating point divide by zero
FPE_FLTOVF#     EQU   34  Floating point overflow
FPE_FLTUND#     EQU   35  Floating point underflow
FPE_FLTRES#     EQU   36  Floating point inexact result
FPE_FLTINV#     EQU   37  Invalid floating point operation
FPE_FLTSUB#     EQU   38  Subscript out of range
FPE_FLTSIG#     EQU   39  Floating point significance exception
FPE_DECDATA#    EQU   40  Decimal data exception
FPE_DECDIV#     EQU   41  Decimal divide by zero
FPE_DECOVF#     EQU   42  Decimal overflow
FPE_UNKWN#      EQU   43  Undetermined exception
*********************************************************************
SEGV_MAPERR#    EQU   51  Address not mapped to object
SEGV_ACCERR#    EQU   52  Invalid permissions for mapped object
SEGV_PROTECT#   EQU   53  Invalid key access
SEGV_ADDRESS#   EQU   54  Invalid address
*********************************************************************
BUS_ADRALN#     EQU   71  Invalid address alignment
BUS_ADRERR#     EQU   72  Non-existent physical address
BUS_OBJERR#     EQU   73  Object specific hardware error
*********************************************************************
TRAP_BRKPT#     EQU   91  Process breakpoint
TRAP_TRACE#     EQU   92  Process trace trap
*********************************************************************
CLD_EXITED#     EQU  101  Child has exited
CLD_KILLED#     EQU  102  Child was killed
CLD_DUMPED#     EQU  103  Child was terminated abnormally
CLD_TRAPPED#    EQU  104  Traced child has trapped
CLD_STOPPED#    EQU  105  Child has stopped
CLD_CONTINUED#  EQU  106  Stopped child was continued
*********************************************************************
POLL_IN#        EQU  111  Data input available
POLL_OUT#       EQU  112  Output buffers available
```

```
POLL_MSG#       EQU  113  Input message available
POLL_ERR#       EQU  114  I/O error
POLL_PRI#       EQU  115  High priority input available
POLL_HUP#       EQU  116  Device disconnected
*********************************************************************
ABND_REAL#      EQU  170  Abend Real
*********************************************************************
SI_ASYNCIO#     EQU  175  Completion of an asynchronous I/O
SI_QUEUE#       EQU  176  Signal sent by sigqueue()
*********************************************************************
** Equate for BPX1STW (sigtimedwait) syscall that when specified    *
** for the "Seconds" parameter indicates not to timeout while       *
** waiting for signal(s).                                           *
*********************************************************************
SIG#NO_TIMEOUT  EQU  X'7FFFFFFF'
** BPXYSIGH End
```

BPXYSINF — Map SIGINFO_T structure

DSECT (SIGINFO_T) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use “BPXYSINF — Map SIGINFO_T structure” on page 1113.

```
           BPXYSINF   ,
** BPXYSINF: siginfo_t Structure
**  Used By: waitid
SIGINFO_T            DSECT ,      Siginfo_t structure
SI_SIGNO             DS    F      signal number
SI_ERRNO             DS    F      error number
SI_CODE              DS    F      signal code
SI_PID               DS    F      sending process ID
SI_UID               DS    F      real user ID of sending process
SI_ADDR              DS    A      address of faulting instruction
SI_STATUS            DS    F      exit value or signal
SI_BAND              DS    F      band event for SIGPOLL
SI_VALUE             DS    F      signal value
SIGINFO#LENGTH       EQU *-SIGINFO_T Length of this DSECT
** BPXYSINF End
```

BPXYSMC — Map shared mutex/condvar declares and constants

BPXYSMC maps declares and constants for shared mutex/condvar support

```
           BPXYSMC   ,
*
* ********************************************************************
* *   Define equates for FcnCode parameter                           *
* ********************************************************************
*
*
SMC_WAIT EQU   1         Wait function request
SMC_POST EQU   2         Post function request
SMC_INIT EQU   4         Initialization function request
SMC_DESTROY EQU 8        Destroy function request
SMC_POSTALL EQU 16       Postall function request
SMC_SETUPTOWAIT EQU 32   SetupToWait function request
SMC_CANCELSETUPTOWAIT EQU 64 CancelSetupToWait function request
*
* ********************************************************************
* * Mapping for FcnFlags parameter                                   *
* ********************************************************************
*
*
```

```
SMC_FCNFLAGS DSECT
SMC_FCNFLAGSB1 DS 0BL1
SMC_MUTEX EQU  X'80'
SMC_CONDVAR EQU X'40'
SMC_TIMEDWAIT EQU X'20'
SMC_OUTSIDEWAIT EQU X'10'
         ORG   SMC_FCNFLAGSB1+X'00000001'
SMC_FCNFLAGSB2 DS 1BL1
SMC_FCNFLAGSB3 DS 1BL1
SMC_FCNFLAGSB4 DS 1BL1
SMC_FCNFLAGS_LEN EQU *-SMC_FCNFLAGS
*
* ******************************************************************
* * Mapping for Time Structure pointed to by TimeStrucAddr parameter *
* ******************************************************************
*
*
SMCT     DSECT
SMCTSECS DS    1FL4      The time to wait for the condition variable   X
                         expressed in seconds. Seconds can be any      X
                         value greater or equal to 0 or less than or   X
                         equal to 4,294,967,295.
SMCTNANOSECS DS 1FL4     The time in nanoseconds to be added to        X
                         SmctSecs to wait for condition variable.      X
                         Nanoseconds can be any value greater than or  X
                         equal to 0 and less than 1,000,000,000.
SMCT_LEN EQU   *-SMCT
```

BPXYSOCK — Map SOCKADDR structure and constants

BPXYSOCK maps the SOCKADDR structure for socket, accept, bind, sendto, recvfrom, getsockname, and getpeername.

```
           BPXYSOCK   ,
*
*********************************************************************
** BPXYSOCK: z/OS UNIX Socket Address Structure                      *
** Used By: Sockets PFS                                             *
*********************************************************************
*
SOCKADDR            DSECT ,
         AGO   .C411
.B411    ANOP  ,
SOCKADDR            DS    0F
.C411    ANOP  ,
SOCKBEGIN           DS    0F
*
SOCK_LEN            DS    X          Address Length - Length of      **
                                     either SOCK_SIN (for AF_INET    **
                                     sockets) or of the name supplied**
                                     in SOCK_SUN_NAME (for AF_UNIX   **
                                     sockets)
SOCK_FAMILY         DS    X          Address Family
SOCK_DATA           DS    0C         Protocol specific area
*
SOCK#LEN            EQU   *-SOCKADDR Constant - Fixed length of SOCK
*
*********************************************************************
*                                                                   *
*   AF_Inet Socket Address Structure                                *
*                                                                   *
*********************************************************************
*
                    ORG  SOCK_DATA   Start of AF_Inet unique area
SOCK_SIN            DS   0C
SOCK_SIN_PORT       DS   H           Port number used by the appl
```

```
SOCK_SIN_ADDR        DS   CL4         INET address (netid)
                     DS   CL8         Reserved area not used
*
SOCK_SIN#LEN         EQU  *-SOCK_SIN  Constant - Fixed length of
*                                     AF_Inet unique area
*
*
*********************************************************************
*                                                                   *
*   AF_UNIX Socket Address Structure                                *
*                                                                   *
*********************************************************************
*
                     ORG  SOCK_DATA   Start of AF_Unix unique area
SOCK_SUN             DS   0C
SOCK_SUN_NAME        DS   CL108       Path name of the socket  @P0C
*                                     Length 108 matchs RS/6000@P0A
*
SOCK_SUN#LEN         EQU  *-SOCK_SUN  Constant - Fixed length of
*                                     AF_Unix unique area
*
*********************************************************************
*                                                                   *
*   AF_Inet6 Socket Address Structure                               *
*                                                                   *
*********************************************************************
*
                     ORG  SOCK_DATA   Start of AF_Inet6 area       @PBA
SOCK_SIN6            DS   0C                                       @PBA
SOCK_SIN6_PORT       DS   H           Port number used by the appl @PBA
SOCK_SIN6_FLOWINFO   DS   CL4         FLOW INFORMATION             @PBA
SOCK_SIN6_ADDR       DS   CL16        INET address (netid)         @PBA
SOCK_SIN6_SCOPE_ID   DS   CL4         SCOPE ID                     @PBA
*                                                                  @PBA
SOCK_SIN6#LEN        EQU  *-SOCK_SIN6 Length of AF_INET6 area      @PBA
*
*********************************************************************
*                                                                   *
*   Equates for Address Families                                    *
*                                                                   *
*********************************************************************
*
AF_UNSPEC            EQU  0           Unspecified
AF_UNIX              EQU  1           Unix Domain
AF_INET              EQU  2           Internetwork: UDP TCP
AF_IMPLINK           EQU  3           Arpanet imp addresses
AF_PUP               EQU  4           pup protocols: BSP
AF_CHAOS             EQU  5           mit CHAOS protocols
AF_NS                EQU  6           XEROX NS protocols
AF_NBS               EQU  7           nbs protocols
AF_ECMA              EQU  8           European computer man.
AF_DATAKIT           EQU  9           datakit protocols
AF_CCITT             EQU  10          CCITT protocols: X.25
AF_SNA               EQU  11          IBM SNA
AF_DECNET            EQU  12          DECNet
AF_DLI               EQU  13          Direct data link interface
AF_LAT               EQU  14          LAT
AF_HYLINK            EQU  15          NSC hyperchannel
AF_APPLETALK         EQU  16          Apple Talk
AF_IUCV              EQU  17          IBM IUCV
AF_ESCON             EQU  18          ESCON UDP                    @D4A
AF_INET6             EQU  19          IPv6                         @P8A
AF_ROUTE             EQU  20          Routing Sockets              @P8A
AF_MAX               EQU  21                                       @P8C
*
*********************************************************************
*   Equates for protocol                                            @P2A
```

```
*********************************************************************
*
IPPROTO_IP           EQU   0          DEFAULT PROTOCOL
IPPROTO_TCP          EQU   6          TCP
IPPROTO_UDP          EQU  17          USER DATAGRAM
IPPROTO_IPV6         EQU  41          IPv6                          @PBA
IPPROTO_ICMPV6       EQU  58          IPv6 ICMP                     @D9A
*
IPPROTO_HOPOPTS      EQU   0                                        @DAA
IPPROTO_ROUTING      EQU  43                                        @DAA
IPPROTO_FRAGMENT     EQU  44                                        @DAA
IPPROTO_ESP          EQU  50                                        @DAA
IPPROTO_AH           EQU  51                                        @DAA
IPPROTO_NONE         EQU  59                                        @DAA
IPPROTO_DSTOPTS      EQU  60                                        @DAA
*
*********************************************************************
*                                                                   *
*   Equates for setpeer options                                     *
*                                                                   *
*********************************************************************
*
SOCK#SO_SET          DC   X'00000200'
SOCK#SO_SET_EQU      EQU  X'00000200'                      @04A
SOCK#SO_UNSET        DC   X'00000400'
SOCK#SO_UNSET_EQU    EQU  X'00000400'                      @04A
*
*********************************************************************
*                                                                   *
*   Equates for socket types                                        *
*                                                                   *
*********************************************************************
*
SOCK#_STREAM         EQU  1
SOCK#_DGRAM          EQU  2
SOCK#_RAW            EQU  3
SOCK#_RDM            EQU  4
SOCK#_SEQPACKET      EQU  5
*
*********************************************************************
*                                                                   *
*   Equates for Dimension   (socket/socketpair syscall)             *
*                                                                   *
*********************************************************************
*
SOCK#DIM_SOCKET      EQU  1
SOCK#DIM_SOCKETPAIR  EQU  2
SOCK#DIM_SOCKETWAFFINITY     EQU  3                          @02A
SOCK#DIM_SOCKETPAIRWAFFINITY EQU  4                          @02A
*
*********************************************************************
*                                                                   *
*   Equates for getname option                                      *
*                                                                   *
*********************************************************************
*
SOCK#GNMOPTGETPEERNAME EQU  1
SOCK#GNMOPTGETSOCKNAME EQU  2
*
*********************************************************************
*                                                                   *
*   Equates for sockopt                                             *
*                                                                   *
*********************************************************************
*
SOCK#OPTOPTGETSOCKOPT    EQU  1
SOCK#OPTOPTSETSOCKOPT    EQU  2
```

```
SOCK#OPTOPTSETIBMSOCKOPT EQU  3                                      @D5A
*
*********************************************************************
*                                                                   *
*   Equates for Shutdown options                                    *
*                                                                   *
*********************************************************************
*
SOCK#SHUTDOWNREAD      EQU  0
SOCK#SHUTDOWNWRITE     EQU  1
SOCK#SHUTDOWNBOTH      EQU  2
*
*
*********************************************************************
*                                                                   *
*   Equate for Level Number for socket options                      *
*                                                                   *
*********************************************************************
*
SOCK#SOL_SOCKET      DC   X'0000FFFF'
SOCK#SOL_SOCKET_EQU  EQU  X'0000FFFF'                      @04A
*
*
*********************************************************************
*                                                                   *
*   Equate for InAddrAny for bind requests                          *
*                                                                   *
*********************************************************************
*
INADDR_ANY           DC   X'00000000'
INADDR_ANY_EQU       EQU  X'00000000'                      @04A
*
INADDR_LOOPBACK      DC   X'7F000001'                            @PBA
INADDR_LOOPBACK_EQU  EQU  X'7F000001'                      @04A
IN6ADDR_ANY          DC   X'00000000000000000000000000000000'     @PBA
IN6ADDR_LOOPBACK     DC   X'00000000000000000000000000000001'     @PBA
IN6ADDR_MAPPEDV4     DC   X'00000000000000000000FFFF'             @D9A
IN6ADDR_COMPATV4     DC   X'000000000000000000000000'             @D9A
*
*********************************************************************
*                                                                   *
*   Equates for Socket options                                      *
*                                                                   *
*********************************************************************
*
SOCK#SO_DEBUG        DC   X'00000001'
SOCK#SO_ACCEPTCONN   DC   X'00000002'
SOCK#SO_REUSEADDR    DC   X'00000004'
SOCK#SO_KEEPALIVE    DC   X'00000008'
SOCK#SO_DONTROUTE    DC   X'00000010'
SOCK#SO_BROADCAST    DC   X'00000020'
SOCK#SO_USELOOPBACK  DC   X'00000040'
SOCK#SO_LINGER       DC   X'00000080'
SOCK#SO_OOBINLINE    DC   X'00000100'
SOCK#SO_REUSEPORT   EQU   X'00000200'     To match socket.h       @DHA
SOCK#SO_REUSEPORT2  EQU   X'00000007'     As implemented in TCPIP @DHA
*
SOCK#SO_SNDBUF       DC   X'00001001'
SOCK#SO_RCVBUF       DC   X'00001002'
SOCK#SO_SNDLOWAT     DC   X'00001003'
SOCK#SO_RCVLOWAT     DC   X'00001004'
SOCK#SO_SNDTIMEO     DC   X'00001005'
SOCK#SO_RCVTIMEO     DC   X'00001006'
SOCK#SO_ERROR        DC   X'00001007'
SOCK#SO_TYPE         DC   X'00001008'
*
* Non-standard sockopts
```

```
*
SO_PROPAGATEID       DC   X'00004000'                          @D7A
SO_CLUSTERCONNTYPE   DC   X'00004001'                          @P8A
SO_SECINFO           DC   X'00004002'                             @D9A
SO_RECVUSERNAME      EQU  X'00004003'                          @DHA
*                                                           @04A
*  EQUated Versions                                         @04A
*                                                           @04A
SOCK#SO_DEBUG_EQU       EQU  X'00000001'                    @04A
SOCK#SO_ACCEPTCONN_EQU  EQU  X'00000002'                    @04A
SOCK#SO_REUSEADDR_EQU   EQU  X'00000004'                    @04A
SOCK#SO_KEEPALIVE_EQU   EQU  X'00000008'                    @04A
SOCK#SO_DONTROUTE_EQU   EQU  X'00000010'                    @04A
SOCK#SO_BROADCAST_EQU   EQU  X'00000020'                    @04A
SOCK#SO_USELOOPBACK_EQU EQU  X'00000040'                    @04A
SOCK#SO_LINGER_EQU      EQU  X'00000080'                    @04A
SOCK#SO_OOBINLINE_EQU   EQU  X'00000100'                    @04A
SOCK#SO_SNDBUF_EQU      EQU  X'00001001'                    @04A
SOCK#SO_RCVBUF_EQU      EQU  X'00001002'                    @04A
SOCK#SO_SNDLOWAT_EQU    EQU  X'00001003'                    @04A
SOCK#SO_RCVLOWAT_EQU    EQU  X'00001004'                    @04A
SOCK#SO_SNDTIMEO_EQU    EQU  X'00001005'                    @04A
SOCK#SO_RCVTIMEO_EQU    EQU  X'00001006'                    @04A
SOCK#SO_ERROR_EQU       EQU  X'00001007'                    @04A
SOCK#SO_TYPE_EQU        EQU  X'00001008'                    @04A
SO_PROPAGATEID_EQU      EQU  X'00004000'                    @04A
SO_CLUSTERCONNTYPE_EQU  EQU  X'00004001'                    @04A
SO_SECINFO_EQU          EQU  X'00004002'                    @04A
*
* SO_CLUSTERCONNTYPE Output Values
*
SO_CLUSTERCONNTYPE_NOCONN        EQU  0                        @P8A
SO_CLUSTERCONNTYPE_NONE          EQU  1                        @P8A
SO_CLUSTERCONNTYPE_SAME_CLUSTER  EQU  2                        @P8A
SO_CLUSTERCONNTYPE_SAME_IMAGE    EQU  4                        @P8A
SO_CLUSTERCONNTYPE_INTERNAL      EQU  8                        @P8A
*
*
* IPPROTO_IP Options
*
IP_TOS                    EQU 2                           @P9C@D6A
IP_MULTICAST_TTL          EQU 3                               @D6A
IP_MULTICAST_LOOP         EQU 4                               @D6A
IP_ADD_MEMBERSHIP         EQU 5                               @D6A
IP_DROP_MEMBERSHIP        EQU 6                               @D6A
IP_MULTICAST_IF           EQU 7                           @P9C@D6A
IP_DEFAULT_MULTICAST_TTL  EQU 1                               @D6A
IP_DEFAULT_MULTICAST_LOOP EQU 1                               @D6A
IP_MAX_MEMBERSHIPS        EQU 20                              @D6A
IP_BLOCK_SOURCE           EQU 10                              @DEA
IP_UNBLOCK_SOURCE         EQU 11                              @DEA
IP_ADD_SOURCE_MEMBERSHIP  EQU 12                              @DEA
IP_DROP_SOURCE_MEMBERSHIP EQU 13                              @DEA
*********************************************************************
*    Multicast Source Filter Structures from RFC 3678               *
********************************************************************* @DEA
         AIF   ('&DSECT' EQ 'NO').B425                          @DEA
SOCKADDR_STORAGE_STRUCT DSECT ,                                 @DEA
         AGO   .C425                                           @DEA
.B425    ANOP  ,                                               @DEA
SOCKADDR_STORAGE_STRUCT DS 0D                                  @DEA
.C425    ANOP  ,                                               @DEA
SOCKADDR_STORAGE DS CL128                                      @DEA
*
* setibmsockopt options
*
SOCK#SO_BULKMODE          DC X'00008000'                    @D5A
```

```
SOCK#SO_IGNOREINCOMINGPUSH DC X'00000001'                       @D5A
SOCK#SO_NONBLOCKLOCAL      DC X'00008001'                       @P7A
SOCK#SO_IGNORESOURCEVIPA   DC X'00000002'                       @P7A
*                       Toggles the use of non-VIPA addresses.  When
*                       enabled, non-VIPA addresses will be used for
*                       outbound IP packets.
SOCK#SO_OPTMSS             DC X'00008003'                       @P7A
*                  Toggles the use of optimal TCP segment size.
*                  When enabled, the TCP segment size may be optimally
*                  increased on outbound data transfers.  This may
*                  reduce the amount of TCP outbound and inbound
*                  acknowledgement packet processing; therefore,
*                  minimizing CPU consumption.                  @P7A
SOCK#SO_OPTACK             DC X'00008004'   Optimize Acks       @P7A
SOCK#SO_EIOIFNEWTP         DC X'00000005'   Notify of new tp    @PAA
SOCK#SO_ACCEPTECONNABORTED DC X'00000006'   Notify of conn abtd @PEA
SOCK#SO_EXCLWRT            DC X'00000007'   Write Serialization @03A
*                          Control Stream Write Serialization   @03A
*                      SetIbmSockOpt option to toggle system supplied
*                      serialization on TCP stream socket writes.
*                      NOTE: This function has been disabled and its @DIC
*                            use is thus discouraged.  This function @DIC
*                            may be withdrawn in a future release.   @DIC
*  EQUated Versions                                           @04A
SOCK#SO_BULKMODE_EQU           EQU X'00008000'              @04A
SOCK#SO_IGNOREINCOMINGPUSH_EQU EQU X'00000001'              @04A
SOCK#SO_NONBLOCKLOCAL_EQU      EQU X'00008001'              @04A
SOCK#SO_IGNORESOURCEVIPA_EQU   EQU X'00000002'              @04A
SOCK#SO_OPTMSS_EQU             EQU X'00008003'              @04A
SOCK#SO_OPTACK_EQU             EQU X'00008004'              @04A
SOCK#SO_EIOIFNEWTP_EQU         EQU X'00000005'              @04A
SOCK#SO_ACCEPTECONNABORTED_EQU EQU X'00000006'              @04A
SOCK#SO_EXCLWRT_EQU            EQU X'00000007'              @04A
*
*********************************************************************
*                                                                   *
*   Equates for So_   option values                                 *
*                                                                   *
*********************************************************************
SOCK#SO_SETOPTIONON       DC  X'00000001'                       @PAA
SOCK#SO_SETOPTIONON_EQU   EQU X'00000001'                       @04A
SOCK#SO_SETOPTIONOFF      DC  X'00000000'                       @PAA
SOCK#SO_SETOPTIONOFF_EQU  EQU X'00000000'                       @04A
*********************************************************************
*                                                                   *
*   Equates for IPPROTO_TCP options                                 *
*                                                                   *
*********************************************************************
SOCK#TCP_NODELAY          DC  X'00000001'                       @P4A
SOCK#TCP_NODELAY_EQU      EQU X'00000001'                       @04A
SOCK#TCP_KEEPALIVE        DC  X'00000008'                       @P9A
SOCK#TCP_KEEPALIVE_EQU    EQU X'00000008'                       @04A
*
*********************************************************************
*                                                                   *
*   Equates for Socket Port Constant                                *
*                                                                   *
*********************************************************************
*
SOCK#LASTRESERVEPORT   EQU  1023                                 @D2A
*
*
         AIF   ('&DSECT' EQ 'NO').B412                           @01A
IP_MREQ         DSECT ,                                       @P9M@D6A
         AGO   .C412                                            @01A
.B412    ANOP  ,                                                @01A
IP_MREQ          DS 0F                                         @01A
```

```
.C412    ANOP  ,                                                 @01A
IMR_MULTIADDR    DS CL4            IP MULTICAST ADDR OF GROUP   @D6A
IMR_INTERFACE    DS CL4            LOCAL IP ADDR OF INTERFACE   @D6A
*
*********************************************************************
*                                                                   *
* IP_MREQ_SOURCE STRUCTURE                                          *
*                                                                   *
******************************************************************* @DEA
         AIF   ('&DSECT' EQ 'NO').B422                           @DEA
IP_MREQ_SOURCE DSECT ,                                             @DEA
         AGO   .C422                                               @DEA
.B422    ANOP  ,                                                   @DEA
IP_MREQ_SOURCE DS 0F                                               @DEA
.C422    ANOP   ,                                                  @DEA
IMRS_MULTIADDR  DS CL4              IP MULTICAST ADDR              @DEA
IMRS_SOURCEADDR DS CL4              IP SOURCE ADDR                 @DEA
IMRS_INTERFACE  DS CL4              LOCAL IP ADDR OF INTERFACE     @DEA
*                                                                  @DEA
*********************************************************************
*                                                                   *
* GROUP_REQ STRUCTURE                                               *
*                                                                   *
******************************************************************* @DEA
         AIF   ('&DSECT' EQ 'NO').B423                           @DEA
GROUP_REQ DSECT ,                                                  @DEA
         AGO   .C423                                               @DEA
.B423    ANOP  ,                                                   @DEA
GROUP_REQ DS   0F                                                  @DEA
.C423    ANOP  ,                                                   @DEA
GR_INTERFACE DS CL4                 INTERFACE INDEX                @DEA
         DS    CL4                  PADDING                        @DEA
GR_GROUP DS    CL(L'SOCKADDR_STORAGE) GROUP ADDRESS                @DEA
         ORG   GR_GROUP                                           @DEA
GR_MULTISOCKADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN)                     @DEA
         ORG   GR_GROUP                                           @DEA
GR_MULTISOCKADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN)                    @DEA
*                                                                  @DEA
*********************************************************************
*                                                                   *
* GROUP_SOURCE_REQ STRUCTURE                                        *
*                                                                   *
******************************************************************* @DEA
         AIF   ('&DSECT' EQ 'NO').B424                             @DEA
GROUP_SOURCE_REQ DSECT ,                                           @DEA
         AGO   .C424                                               @DEA
.B424    ANOP  ,                                                   @DEA
GROUP_SOURCE_REQ DS 0D                                             @DEA
.C424    ANOP  ,                                                   @DEA
GSR_INTERFACE DS CL4                INTERFACE INDEX                @DEA
         DS    CL4                  PADDING                        @DEA
GSR_GROUP DS   CL(L'SOCKADDR_STORAGE) GROUP ADDRESS                @DEA
         ORG   GSR_GROUP                                          @DEA
GSR_GROUPADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN)                        @DEA
         ORG   GSR_GROUP                                          @DEA
GSR_GROUPADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN)                       @DEA
GSR_SOURCE DS  CL(L'SOCKADDR_STORAGE) SOURCE ADDRESS               @DEA
         ORG   GSR_SOURCE                                         @DEA
GSR_SOURCEADDR4 DS CL(SOCK#LEN+SOCK_SIN#LEN)                       @DEA
         ORG   GSR_SOURCE                                         @DEA
GSR_SOURCEADDR6 DS CL(SOCK#LEN+SOCK_SIN6#LEN)                      @DEA
*                                                                  @DEA
*********************************************************************
*                                                                   *
*   Structure for So_Linger                                         *
*                                                                 @D3A*
*********************************************************************
```

```
*
         AIF   ('&DSECT' EQ 'NO').B413                          @01A
SOCK_LINGER_STRUCT DSECT ,                                       @P9A
         AGO   .C413                                            @01A
.B413    ANOP  ,                                                @01A
SOCK_LINGER_STRUCT   DS   0F                                    @01A
.C413    ANOP  ,                                                @01A
SOCK_L_ONOFF         DS    F          On/Off indicator         @P9M@D3A
SOCK_L_LINGER        DS    F          Length of time to linger @P9M@D3C
*********************************************************************
*                                                                   *
*   Equates for IPPROTO_IPV6 Options                            @D9A *
*                                                                   *
*********************************************************************
SOCK#IPV6_UNICAST_HOPS    EQU 3                                  @D9A
SOCK#IPV6_MULTICAST_LOOP  EQU 4
SOCK#IPV6_JOIN_GROUP      EQU 5
SOCK#IPV6_LEAVE_GROUP     EQU 6
SOCK#IPV6_MULTICAST_IF    EQU 7
SOCK#IPV6_MULTICAST_HOPS  EQU 9
SOCK#IPV6_V6ONLY          EQU 10
SOCK#IPV6_HOPLIMIT        EQU 11            ANC DATA ONLY
SOCK#IPV6_PATHMTU         EQU 12
SOCK#IPV6_PKTINFO         EQU 13
SOCK#IPV6_RECVHOPLIMIT    EQU 14
SOCK#IPV6_RECVPKTINFO     EQU 15
SOCK#IPV6_RECVPATHMTU     EQU 16
SOCK#IPV6_REACHCONF       EQU 17
SOCK#IPV6_USE_MIN_MTU     EQU 18
SOCK#IPV6_CHECKSUM        EQU 19

SOCK#IPV6_NEXTHOP          EQU 20
SOCK#IPV6_RTHDR            EQU 21
SOCK#IPV6_HOPOPTS          EQU 22
SOCK#IPV6_DSTOPTS          EQU 23
SOCK#IPV6_RTHDRDSTOPTS     EQU 24
SOCK#IPV6_RECVRTHDR        EQU 25
SOCK#IPV6_RECVHOPOPTS      EQU 26
*SOCK#IPV6_RECVRTHDRDSOPTS EQU 27                                @DCD
SOCK#IPV6_RECVDSTOPTS      EQU 28
SOCK#IPV6_DONTFRAG         EQU 29                                @DCA
SOCK#IPV6_TCLASS           EQU 30                                @DCA
SOCK#IPV6_RECVTCLASS       EQU 31                                @DCA
SOCK#IPV6_ADDR_PREFERENCES EQU 32                                @DJA

SOCK#IPV6_RTHDR_TYPE_0     EQU 0        IPv6 Routing hdr type 0  @D9A
*********************************************************************
*   Protocol Independent Options                                @DEA *
*********************************************************************
SOCK#MCAST_JOIN_GROUP         EQU 40                             @DEA
SOCK#MCAST_LEAVE_GROUP        EQU 41                             @DEA
SOCK#MCAST_JOIN_SOURCE_GROUP  EQU 42                             @DEA
SOCK#MCAST_LEAVE_SOURCE_GROUP EQU 43                             @DEA
SOCK#MCAST_BLOCK_SOURCE       EQU 44                             @DEA
SOCK#MCAST_UNBLOCK_SOURCE     EQU 45                             @DEA
*********************************************************************
*                                                                   *
*   Equates for IPPROTO_ICMPV6 options                          @D9A *
*                                                                   *
*********************************************************************
SOCK#ICMP6_FILTER          EQU 1

*********************************************************************
*                                                                   *
*   Structure for Packet Source/Destination Information         @D9A *
*                                                                   *
*********************************************************************
```

```
*
IN6_PKTINFO        DSECT ,
         AGO   .C414
.B414    ANOP  ,
IN6_PKTINFO          DS   0F
.C414    ANOP  ,
IPI6_ADDR            DS    CL16       IPv6 Addr
IPI6_IFINDEX         DS    F          Interface Index

*********************************************************************
*                                                                   *
*   Structure for Multicast Mreq                              @D9A *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B415
IPV6_MREQ          DSECT ,
         AGO   .C415
.B415    ANOP  ,
IPV6_MREQ            DS   0F
.C415    ANOP  ,
IPV6MR_MULTIADDR     DS    CL16       IPv6 Addr
IPV6MR_INTERFACE     DS    F          Interface index
*********************************************************************
*                                                                   *
*   Structure for CInet Interface Index                       @D9A *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B416
IFINDEX            DSECT ,
         AGO   .C416
.B416    ANOP  ,
IFINDEX              DS   0F
.C416    ANOP  ,
IFI_TDX              DS    H          Cinet Td Index
IFI_INDEX            DS    H          Stacks  Interface Index

*********************************************************************
*                                                                   *
*   Structure for Icmp6 Filtering                             @D9A *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B417
ICMP6_FILTER       DSECT ,
         AGO   .C417
.B417    ANOP  ,
ICMP6_FILTER         DS    0F                                  @DBC
.C417    ANOP  , ICMP6_FILT           DS    8F         8*32 = 256 bits
*
*
ICMP6_DST_UNREACH       EQU   1                                  @DBA
ICMP6_PACKET_TOO_BIG    EQU   2                                  @DBA
ICMP6_TIME_EXCEEDED     EQU   3                                  @DBA
ICMP6_PARAM_PROB        EQU   4                                  @DBA
ICMP6_INFOMSG_MASK      EQU 128                                  @DBA
ICMP6_ECHO_REQUEST      EQU 128                                  @DBA
ICMP6_ECHO_REPLY        EQU 129                                  @DBA
MLD_LISTENER_QUERY      EQU 130                                  @DBA
MLD_LISTENER_REPORT     EQU 131                                  @DBA
MLD_LISTENER_REDUCTION  EQU 132                                  @DBA
ND_ROUTER_SOLICIT       EQU 133                                  @DBA
ND_ROUTER_ADVERT        EQU 134                                  @DBA
ND_NEIGHBOR_SOLICIT     EQU 135                                  @DBA
ND_NEIGHBOR_ADVERT      EQU 136                                  @DBA
ND_REDIRECT             EQU 137                                  @DBA
```

```
*
*********************************************************************
*                                                                   *
*   Routing header                                          @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B418
IP6_RTHDR          DSECT ,
         AGO   .C418
.B418    ANOP  ,
IP6_RTHDR            DS   0F
.C418    ANOP  ,
IP6R_NXT             DS   BL1         Next header
IP6R_LEN             DS   BL1         Length in units of 8 octets
IP6R_TYPE            DS   BL1         Routing type
IP6R_SEGLEFT         DS   BL1         Segments left
*
*********************************************************************
*                                                                   *
*   Type 0 Routing header                                   @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B419
IP6_RTHDR0         DSECT ,
         AGO   .C419
.B419    ANOP  ,
IP6_RTHDR0           DS   0F
.C419    ANOP  ,
IP6R0_NXT            DS   BL1         Next header
IP6R0_LEN            DS   BL1         Length in units of 8 octets
IP6R0_TYPE           DS   BL1         Always zero
IP6R0_SEGLEFT        DS   BL1         Segments left
IP6R0_RESERVED       DS   FL4         Reserved field
IP6R0_ADDR           DS   0CL16       Upto 127 in6_addr          @DDA
*
*********************************************************************
*                                                                   *
*   Hop-by-Hop options header                               @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B41A
IP6_HBH            DSECT ,
         AGO   .C41A
.B41A    ANOP  ,
IP6_HBH              DS   0F
.C41A    ANOP  ,
IP6H_NXT             DS   BL1         Next header
IP6H_LEN             DS   BL1         Length in units of 8 octets
IP6H_OPTIONS         DS   0C          Options                    @DDA
*
*********************************************************************
*                                                                   *
*   Destination options header                              @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B41B
IP6_DEST           DSECT ,
         AGO   .C41B
.B41B    ANOP  ,
IP6_DEST             DS   0F
.C41B    ANOP  , IP6D_NXT             DS   BL1         Next header
IP6D_LEN             DS   BL1         Length in units of 8 octets
IP6D_OPTIONS         DS   0C          Options                    @DDA
```

```
*
*********************************************************************
*                                                                   *
*   MTU Information                                          @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B41C
IP6_MTUINFO        DSECT ,
         AGO   .C41C
.B41C    ANOP  ,
IP6_MTUINFO          DS   0F
.C41C    ANOP  ,
IP6M_ADDR            DS   CL28        Dst address including zone ID
IP6M_MTU             DS   F           Path MTU in host byte order
*
*********************************************************************
*                                                                   *
*   IPv6 Options Header                                      @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B41D
IP6_OPT            DSECT ,          AGO   .C41D
.B41D    ANOP  ,
IP6_OPT              DS   0F
.C41D    ANOP  ,
IP6O_TYPE            DS   BL1
IP6O_LEN             DS   BL1
*
IP6OPT_TYPE               EQU x'C0'                              @DCA
IP6OPT_TYPE_SKIP          EQU x'00'                              @DCA
IP6OPT_TYPE_DISCARD       EQU x'40'                              @DCA
IP6OPT_TYPE_FORCEICMP     EQU x'80'                              @DCA
IP6OPT_TYPE_ICMP          EQU x'C0'                              @DCA
IP6OPT_MUTABLE            EQU x'20'                              @DCA
*
IP6OPT_PAD1               EQU x'00'                              @DCA
IP6OPT_PADN               EQU x'01'                              @DCA
*
IP6OPT_JUMBO              EQU x'C2'                              @DCA
IP6OPT_NSAP_ADDR          EQU x'C3'                              @DCA
IP6OPT_TUNNEL_LIMIT       EQU x'04'                              @DCA
IP6OPT_ROUTER_ALERT       EQU x'05'                              @DCA
*
*********************************************************************
*                                                                   *
*   Jumbo Payload Option                                     @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B41E
IP6_OPT_JUMBO      DSECT ,
         AGO   .C41E
.B41E    ANOP  ,
IP6_OPT_JUMBO        DS   0F
.C41E    ANOP  ,
IP6OJ_TYPE           DS   BL1
IP6OJ_LEN            DS   BL1
IP6OJ_JUMBO_LEN      DS  4BL1 *
IP6OPT_JUMBO_LEN          EQU 6
*
*********************************************************************
*                                                                   *
*   NSAP Address Option                                      @DCA *
*                                                                   *
*********************************************************************
```

```
*
         AIF   ('&DSECT' EQ 'NO').B41F
IP6_OPT_NSAP       DSECT ,
         AGO   .C41F
.B41F    ANOP  ,
IP6_OPT_NSAP         DS   0F
.C41F    ANOP  ,
IP6ON_TYPE           DS   BL1
IP6ON_LEN            DS   BL1
IP6ON_SRC_NSAP_LEN   DS   BL1
IP6ON_DST_NSAP_LEN   DS   BL1
IP6ON_SRC_NSAP       DS   0C
IP6ON_DST_NSAP       DS   0C
*
*********************************************************************
*                                                                   *
*   Tunnel Limit Option                                      @DCA *
*                                                                   *
*********************************************************************
*
         AIF   ('&DSECT' EQ 'NO').B420
IP6_OPT_TUNNEL     DSECT ,
         AGO   .C420
.B420    ANOP  ,
IP6_OPT_TUNNEL       DS   0F
.C420    ANOP  ,
IP6OT_TYPE           DS   BL1
IP6OT_LEN            DS   BL1
IP6OT_ENCAP_LIMIT    DS   BL1
*
*********************************************************************
*                                                                   *
*   Router alert values (in network byte order)              @DCA *
*                                                                   *
*********************************************************************
*
IP6_ALERT_MLD             EQU  0
IP6_ALERT_RSVP            EQU  1
IP6_ALERT_AN              EQU  2
*
*********************************************************************
*
*  Source address selection preferences                     @DJA
*
*    Used with setsockopt/getsockopt(Sock#IPV6_ADDR_PREFERENCES)
*      and BPX1PCT(PC#IsSrcAddr) for inet6_is_srcaddr() function
*
*********************************************************************
*
IPV6_PREFER_SRC_HOME    EQU x'00000001'  Prefer home address
IPV6_PREFER_SRC_COA     EQU x'00000002'  Prefer care of address
IPV6_PREFER_SRC_TMP     EQU x'00000004'  Prefer temporary address
IPV6_PREFER_SRC_PUBLIC  EQU x'00000008'  Prefer public address
IPV6_PREFER_SRC_CGA     EQU x'00000010'  Prefer Cryptographically
*                                        generated address
IPV6_PREFER_SRC_NONCGA  EQU x'00000020'  Prefer non-cryptographically
*                                        generated address
*********************************************************************
*
*  BPX1PCT(PC#IsSrcAddr) Argument for inet6_is_srcaddr()    @DJA
*
*     inet6_is_srcaddr(IsSrcAddrIpAddr, IsSrcAddrFlags)
*     is implemented with BPX1PCT(' ',PC#IsSrcAddr,ISSRCADDR#LEN,
*                                     ISSRCADDR, Rv, Rc, Rsn)
*     where the ISSRCADDR argument is defined as follows:
*
*********************************************************************
```

```
         AIF   ('&DSECT' EQ 'NO').B426
ISSRCADDR         DSECT ,
         AGO   .C426
.B426    ANOP  ,
ISSRCADDR         DS  0F
.C426    ANOP  ,
ISSRCADDRVER      DS XL1           Version. 1
ISSRCADDRVER1     EQU 1            Version value
                  DS XL3           Reserved. Must be 0
ISSRCADDRIPADDR   DS CL(SOCK#LEN+SOCK_SIN6#LEN)  sockaddr_in6
ISSRCADDRFLAGS    DS F             Flags. See IPV6_PREFER_SRC_*
                  DS 6F            Reserved. Must be 0
*
ISSRCADDR#LEN     EQU *-ISSRCADDR  Length of ISSRCADDR area
*
** BPXYSOCK End
```

BPXYSSET — Map the sigaction set

DSECT=.. is not supported. The generated code will allocate SSETOPTION_FLAGS and a DSECT for SSET. This should be followed by CSECT statement to return to the current DSECT or CSECT.

AMODE 64 callers use “BPXYSSET — Map the sigaction set” on page 1114.

```
            BPXYSSET   ,
**  BPXYSSET: Macro which enables multiple signal calls
**  Used By:  SA2
SSETOPTION_FLAGS   DS    0F
SSETOPTION_FLAGS1  DS    FL1   FLAGS INDICATING CALLER OPTIONS
SSET_IGINVALID     EQU   X'80' IGNORE INVALID SIGNALS & SIGACTIONS     X
                             0=DO NOT IGNORE, 1=IGNORE
                   DS    3FL1  RESERVED
SSET               DSECT ,
SSETCONSOLMASK     DS  XL8    SIGNALS HAVING THE SAME FLAGS,MASK,      X
                              USERDATA, AND SIGNAL ACTION
SSETCOMPARE        DS  0CL20
SSETFLAGS          DS  XL4    VALUE FOR SIGACTION FLAGS (BPXYSIGH)
SSETSAHANDLER      DS  A      ADDRESS OF A SIGNAL HANDLER ROUTINE
SSETSAMASK         DS  XL8    VALUE FOR SIGACTION MASK
SSETUSERDATA       DS  F      USER DEFINED DATA
SSET#LENGTH        EQU *-SSET LENGTH OF ONE SSET ENTRY
** BPXYSSET End
```

BPXYSSTF — Map response structure for file system status

```
            BPXYSSTF   ,
** BPXYSSTF: file system status response structure
**  Used By: STF STV FTV VSF
SSTF                 DSECT ,
SSTFID               DC    C'SSTF'  EBCDIC ID - SSTF (f_OEcbid)
SSTFLEN              DC    A(SSTF#LENGTH)  Length of SSTF (f_OEcblen)
SSTFBLOCKSIZE        DS    F      Block size (f_bsize)
                     DS    F      Reserved
SSTFDBLTOTSPACE      DS    0D     Name of dblword field - total
                     DS    F      Reserved
SSTFTOTALSPACE       DS    F      Total space. The total number of     X
                                  blocks on file system in units of    X
                                  f_frsize (f_blocks)
SSTFDBLUSEDSPACE     DS    0D     Name of dblword field - used
                     DS    F      Reserved
SSTFUSEDSPACE        DS    F      Allocated space in block size units  X
                                  (f_OEusedspace)
SSTFDBLFREESPACE     DS    0D     Name of dblword field - free
```

```
                    DS    F      Reserved
SSTFFREESPACE       DS    F      Space available to unprivileged      X
                                 users in block size units            X
                                 (f_bavail)
SSTFENDVER1         EQU   *      End of Version 1 SSTF
SSTFFSID            DS    F      File system ID (f_fsid)              X
                                 Set by LFS
SSTFFLAG            DS    0BL.32 Bit mask of f_flag vals
SSTFFLAGB1          DS    XL1    byte 1
SSTFEXPORTED        EQU   X'40'  Filesys is exported                  X
                                 (ST_OEEXPORTED)                      X
                                 Set by LFS
SSTFV3PROP          DS    XL1    NFS V3 Properties
SSTFFSF_V3RET       EQU   X'80'   V3 Prop Returned
SSTFFSF_CANSETTIME  EQU   X'10'   time_delta accuracy
SSTFFSF_HOMOGENEOUS EQU   X'08'   Pathconf same for all
SSTFFSF_SYMLINK     EQU   X'02'   Supports Symlinks
SSTFFSF_LINK        EQU   X'01'   Supports Hard Links
SSTFFLAGB3          DS    XL1    byte 3
SSTFFLAGB4          DS    XL1    byte 4
SSTFNOSEC           EQU   X'04'  No Security checks enforced
SSTFNOSUID          EQU   X'02'  SetUID/SetGID not supported          X
                                 (ST_NOSUID)                          X
                                 Set by LFS
SSTFRDONLY          EQU   X'01'  Filesys is read only                 X
                                 (ST_RDONLY)                          X
                                 Set by LFS
SSTFMAXFILESIZE     DS    0D     Name of dblword field - maximum      X
                                 file size                            X
                                 May be set by LFS
SSTFMAXFILESIZEHW   DS    F      High word of max file size           X
                                 (f_OEmaxfilesizehw)
SSTFMAXFILESIZELW   DS    F      Low word of max file size            X
                                 (f_OEmaxfilesizelw)
                    DS    CL16   Reserved
SSTFENDLFSINFO      EQU   *      End of LFS information
SSTFFRSIZE          DS    F      Fundamental filesystem block size    X
                                 (f_frsize)
                    DS    F      Reserved
SSTFDBLBFREE        DS    0D     Name of dblword field -              X
                                 total number of free blocks
                    DS    F      Reserved
SSTFBFREE           DS    F      Total number of free blocks          X
                                 (f_bfree)
SSTFFILENODES       DS    0CL12  File nodes
SSTFFILES           DS    F      Total number of file nodes           X
                                 in the file system (f_files)
SSTFFFREE           DS    F      Total number of free file nodes      X
                                 (f_ffree)
SSTFFAVAIL          DS    F      Number of free file nodes available  X
                                 to unprivileged users (f_favail)
SSTFNAMEMAX         DS    F      Maximum file name len (f_namemax)
SSTFINVARSEC        DS    F      Number of seconds file system        X
                                 will remain unchanged                X
                                 (f_OEinvarsec)
SSTFTIME_DELTA      DS    0CL8   Set file time granularity
SSTFTIME_DELTA_SEC  DS    F        Seconds
SSTFTIME_DELTA_NS   DS    F        Nano-seconds
                    DS    CL12   Reserved
SSTF#LENGTH         EQU   *-SSTF Length of this structure
SSTF#MINLEN         EQU   SSTFENDVER1-SSTF
SSTF#LFSLEN         EQU   SSTFENDLFSINFO-SSTF
** BPXYSSTF End
```

BPXYSTAT — Map the response structure for stat

```
            BPXYSTAT   ,
** BPXYSTAT: stat system call structure
**  Used By: FST LST STA
STAT                DSECT ,
ST_BEGIN            DS    0D
*
ST_EYE              DC    C'STAT'  Eye catcher
ST_LENGTH           DC    AL2(STAT#LENGTH)                          X
                          Length of this structure
ST_VERSION          DC    AL2(ST#VER)                               X
                          Version of this structure
ST_MODE             DS    F      File Mode, mapped by BPXYMODE
ST_INO              DS    F      File Serial Number
ST_DEV              DS    F      Device ID of the file
ST_NLINK            DS    F      Number of links
ST_UID              DS    F      User ID of the owner of the file
ST_GID              DS    F      Group ID of the Group of the file
ST_SIZE             DS    0D     File Size in bytes, for regular
*                                files.  Unspecified, for others
ST_SIZE_H           DS    F      First word of size
ST_SIZE_L           DS    F      Second word of size
ST_ATIME            DS    F      Time of last access
ST_MTIME            DS    F      Time of last data modification
ST_CTIME            DS    F      Time of last file status change
*                                Time is in seconds since
*                                00:00:00 GMT, Jan. 1, 1970
ST_RDEV             DS    0F     Device Information
ST_MAJORNUMBER      DS    H      Major number for this file, if it
*                                   is a character special file.
ST_MINORNUMBER      DS    H      Minor number for this file, if it
*                                   is a character special file.
ST_AUDITORAUDIT     DS    F      Area for auditor audit info
ST_USERAUDIT        DS    F      Area for user audit info
ST_BLKSIZE          DS    F      File Block size
ST_CREATETIME       DS    F      File Creation Time
ST_AUDITID          DS    4F     RACF File ID for auditing
ST_RES01            DS    F
ST_CHARSETID        DS    0XL12  Coded Character Set ID (obsolete
ST_FILETAG          DS    0F     File Tag
FT_CCSID            DS    H      Coded character set ID in binary
FT_UNTAGGED         EQU   X'0000' File has no tag
FT_BINARYTAG        EQU   X'FFFF' File is binary data
FT_FLAGS            DS    XL2     File tagging flags
FT_TXTFLAG          EQU   X'8000' File is pure text data
FT_DEFERTAG         EQU   X'4000' File to be tagged at 1st write
ST_RES01A           DS    2F     reserved
ST_BLOCKS_D         DS    0D     Double word number - blocks allocated
ST_RES02            DS    F
ST_BLOCKS           DS    F      Number of blocks allocated
ST_GENVALUE         DS    0XL4   General attribute values
ST_OPAQUE           DS    XL3    Opaque attribute flags- Reserved
ST_VISIBLE          DS    X      Visible attribute flags
ST_SHARELIB         EQU   X'10'  Shared Library Flag
ST_NOSHAREAS        EQU   X'08'  No shareas flag
ST_APFAUTH          EQU   X'04'  APF authorized flag
ST_PROGCTL          EQU   X'02'  Program controlled flag
ST_EXTLINK          EQU   X'01'  External Symlink
ST_REFTIME          DS    F      Reference time
ST_FID              DS    2F     File identifier
ST_FILEFMT          DS    XL1    File Format
ST_FSPFLAG2         DS    XL1    IFSP_FLAG2 ACL support
ST_ACCESSACL        EQU   X'80'  Access Acl exists
ST_FMODELACL        EQU   X'40'  File Model Acl exists
ST_DMODELACL        EQU   X'20'  Directory Model Acl exists
```

```
ST_RES03             DS    CL2    reserved
ST_CTIMEMSEC         DS    F      Micro-Sec of full Ctime
ST_SECLABEL          DS    CL8    Security Label
ST_RES04             DS    CL4    Reserved for future
ST_ENDVER1          EQU    *      End of Ver 1 Stat
*
                     DS    F      Reserved
ST_ATIME64           DS    D      Access Time
ST_MTIME64           DS    D      Data Modification Time
ST_CTIME64           DS    D      Meda-data Change Time
ST_CREATETIME64      DS    D      File Creation Time
ST_REFTIME64         DS    D      Reference Time
                     DS    D      Reserved
ST_RES05             DS    CL16   Reserved
ST_ENDVER2          EQU    *      End of Ver 2 Stat
*
*   Constants
*
STAT#LENGTH          EQU   *-STAT   Length of STAT
ST#VER01             EQU   1        Version 1 of this structure
ST#VER02             EQU   2        Version 1 of this structure
ST#VER               EQU   ST#VER02 Current version
ST#LEN               EQU   STAT#LENGTH Length of STAT
ST#VER01LEN          EQU   ST_ENDVER1-ST_BEGIN
ST#VER02LEN          EQU   ST_ENDVER2-ST_BEGIN
** BPXYSTAT End
```

BPXYTCCP — Map the terminal control code page structure

```
           BPXYTCCP   ,
** BPXYTCCP: terminal control code page structure
**  Used By: TGC TSC TST
TCCP                 DSECT ,
TCCPFLAG             DS    0BL.32 Bit mask of __tccp_flags
TCCPFLAGB1           DS    XL1    byte 1
TCCPFLAGB2           DS    XL1    byte 2
TCCPFLAGB3           DS    XL1    byte 3
TCCPFLAGB4           DS    XL1    byte 4
TCCPFASTP            EQU   X'02'  If set, indicates that the          X
                                  application can optionally          X
                                  use iconv() services to             X
                                  build the translation tables        X
                                  once and perform all                X
                                  subsequent translation locally.     X
                                  (_TCCP_FASTP)
TCCPBINARY           EQU   X'01'  If set, indicates that binary       X
                                  mode is desired.                    X
                                  The code pages are ignored.         X
                                  (_TCCP_BINARY)
TCCPSRCNAME          DS    CL32   Source code page name               X
                                  The code page name is case sensitive X
                                  and must be null (X'00') terminated. X
                                  (__tccp_fromname)
TCCPTRGNAME          DS    CL32   Target code page name               X
                                  The code page name is case sensitive X
                                  and must be null (X'00') terminated. X
                                  (__tccp_toname)
TCCPEND              EQU   *      End of TCCP
*
*   Constants
*
TCCP#LENGTH          EQU   *-TCCP Length of this structure
*
*   CPCN capability constants
*
TCCP#CPNAMESONLY     EQU   1     Code page names only (_CPCN_NAMES)
```

```
TCCP#CPNAMESANDTBLS  EQU   2     Code page names and conversion tables X
                                 (_CPCN_TABLES)
TCCP#CPNAMEMAX       EQU  32     Maximum length of code page name      X
                                 including terminating null            X
                                 (_TCCP_CPNAMEMAX)
** BPXYTCCP End
```

BPXYTHDQ — Mapping of THDQ structure for BPX1PQG

BPXYTHDQ maps the THDQ structure that is supplied to the BPX1PQG callable service.

```
           BPXYTHDQ   ,
THDQ     DSECT           THDQ - THDQ structure for BPX1PQG callable    X
                         service
THDQHDR  DS    1CL0048   +0 Header section
         ORG   THDQHDR
THDQEYE  DS    1CL0004   +0 eye catcher - 'THDQ'
THDQLENGTH DS  1FL2      +4 Length of THDQ structure
THDQVERSION DS 1FL2      +6 Version number
THDQNUMENTS DS 1FL4      +8 Number of entries in thread array          X
                         ThdqArray
THDQFLAGS DS   1FL4      +C Flags relating to contents of structure
         ORG   THDQFLAGS
THDQFLAGS1 DS  1FL1      +C 1st flag byte
         ORG   THDQFLAGS1
THDQALLSAFE EQU X'80'    All threads are frozen in a safe state
         ORG   THDQFLAGS1+X'00000001'
THDQFLAGS2 DS  1FL1      +D 2nd flag byte
THDQFLAGS3 DS  1FL1      +E 3rd flag byte (used by exit). Cleared on   X
                         initial call to LE exit
THDQFLAGS4 DS  1FL1      +F 4th flag byte
         ORG   THDQFLAGS4
THDQGETSTATE EQU X'80'   Get State Data requested by caller (input to  X
                         exit)
         ORG   THDQFLAGS+X'00000004'
THDQEXITWKA DS 1CL0016   +10 Reserved for registered LE exit
         DS    1CL0016   +20 Reserved
THDQDYN  DS    0C        +30 Dynamic section
*
*      ***************************************************************
*      *                                                             *
*      * Declare array of thread areas                               *
*      *                                                             *
*      ***************************************************************
*
*
THDQARRAY DS   1CL0256   Array of Thread Areas
         ORG   THDQARRAY
THDQATHID DS   1CL0008   Thread ID of target thread
THDQAFLAGS DS  1FL4      Flags returned for target thread
         ORG   THDQAFLAGS
THDQAFLAGS1 DS 1FL1      Flag1 returned for target thread
         ORG   THDQAFLAGS1
THDQANOTFOUND EQU X'80'  Thread was not found, no data was returned
THDQAQFRZSAFE EQU X'40'  Thread is now frozen in a safe state          X
                         determined by Language Env Exit
THDQAOTHERLE EQU X'20'   Thread is part of other language environment  X
                         process
THDQANODATA EQU X'10'    Status data is not available for this thread  X
                         (if Get State is requested). The PSW/Regs and X
                         other status info are not valid. The thread   X
                         may be in the process of being created.
THDQACONDWAIT EQU X'08'  Task is in Condition Wait.                     X
                         If this bit is set, only the DSA ptr --       X
                         Reg13 or Reg4 -- is returned. ThdQAPswIA      X
```

```
                        and all the other regs in ThdQARegs are set   X
                        to 0.  (Caller should already have properly   X
                        saved the PSW and registers information.)
         ORG   THDQAFLAGS1+X'00000001'
THDQAFLAGS2 DS 1FL1      Flag2 returned for target thread
THDQAFLAGS3 DS 1FL1      Flag3 returned for target thread
THDQAFLAGS4 DS 1FL1      Flag4 -- internal flags used by system
         ORG   THDQAFLAGS4
THDQAQUICKFRZ EQU X'80'  Thread was frozen without going through the   X
                        RTL SIR or the slow freeze exit routine.      X
                        1=quick freeze (Status stopped) 0=traditional X
                        freeze
THDQAREGSOK EQU X'40'    Registers and PSW retrieved OK
THDQASKIP EQU  X'20'     1 = LE exit should not look at this thread.
         ORG   THDQAFLAGS+X'00000004'
THDQAREGSSRC DS 1FL2     (Internal) Source from which the regs are     X
                        retrieved. Valid if ThdqARegsOK set.
         DS    1CL0002   Reserved
THDQAREGS DS   1CL0128   Registers of target thread
         ORG   THDQAREGS
THDQAREGSH DS  1CL0064   High registers 0-15
THDQAREGSL DS  1CL0064   Low registers 0-15
THDQADOWNSTACKPTR DS 1CL0008 Down Stack start address of target thread X
                        (For XPLINK)
         ORG   THDQADOWNSTACKPTR
THDQADSTACKPTRH DS 1AL4  High part of Stack address
THDQADSTACKPTRL DS 1AL4  Low part of Stack address
THDQAUPSTACKPTR DS 1CL0008 Up Stack start address of target thread
         ORG   THDQAUPSTACKPTR
THDQAUSTACKPTRH DS 1AL4  High part of Stack address
THDQAUSTACKPTRL DS 1AL4  Low part of Stack address
THDQAPSWIA DS  1CL0008   Instruction address of target thread
         ORG   THDQAPSWIA
THDQAPSWIAH DS 1AL4      High part of instruction addr
THDQAPSWIAL DS 1AL4      Low part of instruction addr
THDQACAAPTR DS 1CL0008   LE CAA pointer for thread
         ORG   THDQACAAPTR
THDQACAAPTRH DS 1AL4     High part of CAA pointer
THDQACAAPTRL DS 1AL4     Low part of CAA pointer
         DS    1CL0004   Reserved
THDQATCBPTR DS 1AL4      TCB Pointer for target thread
THDQAEXITWKA DS 1CL0008  Thread work area used by LE exit (for PD)
         DS    1CL0064   Reserved
*
*   Declare constants
*
*
THDQ#REGSPPSD EQU 1      Regs from PPSD
THDQ#REGSIRB EQU 2       Regs from IRB
THDQ#REGSUSTA EQU 3      Regs from USTA
THDQ#REGSLS EQU 4        Regs from link stack
THDQ#REGSTCB EQU 5       Regs from TCB/STCB
THDQ#REGSRB EQU 6        Regs from RB/XSB
THDQ#REGSCW EQU 7        Regs for CondWait. Status returned as zeroes
THDQ#ID  EQU   C'THDQ'   Eye catcher
THDQ#VER EQU   1         Current version of control block
THDQ#VER01 EQU 1         Version 1 of control block
THDQ_LEN EQU   *-THDQ
```

BPXYTHLI — Thread-level information

```
           BPXYTHLI   ,
THLI     DSECT ,
THLIID   DS    CL4       EBCDIC ID - Thli
THLISP   DS    X         Subpool number of this Thli
THLILEN  DS    FL3       Length of this Thli
```

```
THLIFLAGS DS   0BL4      Flag bits
THLIFLAGB1 DS  0B
THLISIGPENDING EQU X'80' Signal pending flag
THLICANCELDISABLED EQU X'40' Cancel request type 0=enabled, 1=disabled
THLICANCELPENDING EQU X'20' Cancel pending for thread
THLICANCELASYNC EQU X'10' Cancellation request state 0 = controlled, 1 X
                         = aysnc
THLIITERATESIR EQU X'08' Use back door signal dlv 0 = Sir can exit 1 = X
                         New sig in PPSD(Iterate Sir)
THLINOSIG EQU  X'04'     Suppress signal generation for this socket    X
                         call.
THLITIMEOUTSET EQU X'02' Kernel Time Out Service requested
THLITIMERPOPPED EQU X'01' Kernel Time Out Service timer popped
         ORG   THLIFLAGB1+1
THLIFLAGB2 DS  0B
THLIPTQTIMEOUT EQU X'80' If on, invokers of the BPX1PTQ call will      X
                         recieve EAGAIN/JRTimeOut if quiece times oyt  X
                         and all threads are not quieced.
THLIFREEZESTOP EQU X'40' Thread has been frozen via Status Stop
THLIDEFERSIGNALS EQU X'20' Defer signals for user
THLIPOSTANDDEFER EQU X'10' Post regardless of key defer delivery until X
                         key ok
THLITCBEXITPERC EQU X'08' Set by application to allow abends in        X
                         tcbexits to perc to Tcb
THLIIRBNORETRY EQU X'04' Set by NSSIR to indicate the calling IRB is   X
                         not to retry any abends
THLISIGIRBABEND EQU X'02' Abend on sigkill regardless of state
*
*      ***************************************************************
*      *                                                             *
*      * Use WorkPtr64 instead of WorkPtr when ThliUseWorkPtr64 is   *
*      * ON. Thread was pthread_created in 64 bit mode               *
*      *                                                             *
*      ***************************************************************
*
*
THLIUSEWORKPTR64 EQU X'01'
         ORG   THLIFLAGB2+1
THLIFLAGB3 DS  0B
THLIFORKACCTG EQU X'80'  Child accounting data based on setuid         X
                         identity
THLIPROPAUTH EQU X'40'   Propagate JSCBAUTH to child on fork
THLIUNDUBCALLERONLY EQU X'20' mvsprocclp should cleanup caller's       *
                         process only
         ORG   THLIFLAGS+4
THLIPPSD DS    F         Address of Ppsd
THLISIGMASK DS BL8       Signal mask. Primarliy set by sigprocmask().
THLIPRLI DS    A         -> Prli. Process related information
*
*      ***************************************************************
*      *                                                             *
*      * Use ThliWorkPtr64 if this thread was created via a pthread  *
*      * create done in amode 64                                     *
*      *                                                             *
*      ***************************************************************
*
*
THLIWORKPTR DS A         -> To user work area specified on             X
                         pthread_create
THLICOMECB DS  0F        User communication ECB
THLICOMECBWAIT EQU X'80' ECB wait bit
THLICOMECBPOST EQU X'40' ECB post bit
         ORG   THLICOMECB+4
THLICOMFLAGS DS 0BL4     ECB control flags
THLICOMFLAGSB1 DS B      reserved
THLICOMFLAGSB2 DS B      reserved
THLICOMFLAGSB3 DS 0B     reserved for user
```

```
THLICOMFLAGSU0 EQU X'80' reserved for user
THLICOMFLAGSU1 EQU X'40' reserved for user
THLICOMFLAGSU2 EQU X'20' reserved for user
THLICOMFLAGSU3 EQU X'10' reserved for user
THLICOMFLAGSU4 EQU X'08' reserved for user
THLICOMFLAGSU5 EQU X'04' reserved for user
THLICOMFLAGSU6 EQU X'02' reserved for user
THLICOMFLAGSU7 EQU X'01' reserved for user
         ORG   THLICOMFLAGSB3+1
THLICOMFLAGSB4 DS 0B
THLIWILLFREEZEME EQU X'08' LE will issue FreezeMe for this task
THLIFROZEN EQU X'04'     BPX1PQG freeze request has been issued        X
                         against the task
THLISIGPOSTED EQU X'02'  User posted due to signal
THLISIGWAIT EQU X'01'    User wants ECB posted when a signal will be   X
                         delivered
         ORG   THLICOMFLAGS+4
THLIKEY  DS    CL1       PSW key of Thli control block. The key is in  X
                         bits 0-3, bits 4-7 are zero
THLIIP@LEN DS  X
THLIFLAGS2 DS  0B        Flag Byte
*
*      ***************************************************************
*      *                                                             *
*      * Specifying JOBNAME for APPLID is only used by MRPWD and     *
*      * PRSUI.                                                      *
*      *                                                             *
*      ***************************************************************
*
*
THLIF2_SETAPPL EQU X'80' Set RACROUTE APPL parm with JOBNAME
THLICVTON EQU  X'40'     Activates auto conversion for this thread
THLICVTOFF EQU X'20'     Deactivates auto conversion for this          X
                         thread-both ThliCvton and ThliCvtOff should   X
                         not be on
         ORG   THLIFLAGS2+1
THLIEXECPARMNUM DS X     Number of parms being passed in exec style    X
                         parm list. Zero indicates either mvs style    X
                         parm list or not exec
THLITIMERECB DS 0F       ECB posted when timer pops from BPX1STE call
THLITIMERECBWAIT EQU X'80' ECB wait bit
THLITIMERECBPOST EQU X'40' ECB post bit
         DS    BL.030
         ORG   THLITIMERECB+4
THLIASPIRBECB DS 0F      ECB posted when aio done
THLIASPIRBWAIT EQU X'80' Wait bit
THLIASPIRBPOST EQU X'40' Post bit
         ORG   THLIASPIRBECB+4
THLIJAVA DS    A         JAVA thread control block address. *** DO NOT X
                         MOVE this field *** Modified by assembler     X
                         code in JAVA
THLIIP
THLITIMEOUT DS 0CL8      Kernel Time Out Service parameters
THLISECS DS    BL4       Seconds to wait for event
THLINANOS DS   BL4       Nanoseconds to wait for event
THLICCSID DS   H         Program character set Id for filesystem       X
                         reads/writes
THLIAPPLIDLEN DS X       Len of string in ThliAppid ignored if 0 or >8
         DS    CL5       Reserved
THLISRMFLAGS DS 0B       SRM flags Ownership: SRM
THLIVCMOVERRIDE EQU X'80' This bit indicates that this unit of work    X
                         should not follow the standard SRM management X
                         in an VCM=on environment. Instead of trying   X
                         to assign the work to the same affinity node  X
                         for cache efficiency concerns, assign this    X
                         work to any affinity node, ignore any cache   X
                         concerns.
```

```
         ORG   THLISRMFLAGS+1
THLISHMDUMPPRIO DS X     Dump priority assigned to shared memory      *
                         segment on next shmget() or shmat() call
         DS    CL3       Reserved
THLIWORKPTR64 DS CL8     -> To user work area specified on            *
                         pthread_create
THLIUTOKENINFO DS 0CL8   Fork Hi Memory token
THLIPARENTTKN DS CL4     parents IARV64 user token
THLICHILDTKN DS CL4      token assigned to child
THLIRETCODE DS F         Failing retcode from CPR
THLIRSNCODE DS F         Failing rsncode from CPR
THLIAPPLID DS  CL8       Applid passed for ptsec, and __passwd after  *
                         ptsec
         DS    CL24      reserved
THLIEP_FUNCTIONCODE DS F
THLIEXTENEDEDPARMS DS 0CL24
         ORG   THLIEXTENEDEDPARMS
THLIEXTENDEDPARMAREA DS 0CL24
         DS    CL24
*
*
*      Add a substructure of 5 under the Main UNION for each
*      set of input parms  (each of these maps over the storage
*      defned as ThliExtendedParmArea)
*
*
*
         ORG   THLIEXTENEDEDPARMS
THLIEXTENDEDAPPLPARMS DS 0CL24 Entry parms
THLIEP_APPLIDLEN DS X
THLIEP_APPLID DS CL8
         DS    CL15
         ORG   THLIEXTENDEDPARMS
THLIEXTENDEDMRSMFPARMS DS 0CL24 BPX1SMF parms
THLIEP_MRSMFFLAGS DS 0B
THLIEP_MRSMFIEFU83 EQU X'80' If on for invokers of BPX1SMF, SMF is     *
                         invoked to write the SMF record such that the *
                         SMF record is given to installation exit      *
                         IEFU83 rather than exit IEFU84
         ORG   THLIEP_MRSMFFLAGS+1
         DS    CL23
         ORG   THLIEXTENDEDPARMS
THLIEXTENDEDGIDNAME DS 0CL24 Output for GIDNameSet
THLIEP_GIDNAMELEN DS F
THLIEP_GIDNAME DS CL8
THLIEP_GIDLEN DS F
THLIEP_GID DS  F
THLIEP_GROUPCOUNT DS F
*
*   End of Extended parameter area declarations
*
*
* ********************************************************************
* * The ThliSecErrDetail area provides detailed information for      *
* * select SAF service errors. This information is only provided when*
* * the syscall rv/rc/rs values are ambiguous and do not provide the *
* * user with enough information to determine the potential cause    *
* * of the error. Refer to the appropriate SAF service documentation *
* * to decode the RACF/SAF return codes.                             *
* ********************************************************************
*
*
THLISECERRDETAIL DS 0CL64
THLISECERRCT DS 0CL40    This level also maps the first 40 bytes of
                         the ctrace SAF exception record, see the
                         bpxtrace command for details
THLISECSERVICENAME DS CL8 SAF serice name
```

```
THLISECSERVICEQUAL DS CL8 REQ or function (when applicable)
THLISECSAFRC DS F        SAF return code
THLISECRACFRC DS F       RACF return code
THLISECRACFRS DS F       RACF reason code
THLISECSCRV DS F         Syscall RV
THLISECSCRC DS F         Syscall RC
THLISECSCRS DS F         Syscall RS
THLISECERRNONCT DS F
THLISECSYSCALL DS CL8    Syscall name
         DS    CL4       reserved (dword bdy)
THLISECERRTOD DS CL8     Time of error (TOD)
         DS    CL20      reserved
*
*
*     NOTE: The size of this control block is retrieved dynamically
*           during runtime by the modules that need it.
*           When adding additional fields to this control block,
*
*     ====> THE ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
*
*
THLIEND  DS    0C        End of Thli
*
*   Extended paramter codes.
*
*
THLIEP_APPLSET EQU 1
*
*
*
*
THLI#ID  EQU   C'THLI'   Control Block Acronym
THLI#LEN EQU   144       Length of Thli
THLI#SP  EQU   230       Subpool for the Thli
THLI_LEN EQU   *-THLI
```

BPXYTIMS — Map the response structure for times

```
           BPXYTIMS   ,
** BPXYTIMS: times syscall structure
**  Used By: TIM
TIMS                DSECT ,
TIMSBEGIN           DS    0F
TIMSUTIME           DS    F      User CPU time of current process
*                                in hundredths of a second.
*                                This includes the TCB and SRB time
*                                of the calling process minus the
*                                TCB time accumulated while running
*                                in the  kernel address space.
TIMSSTIME           DS    F      System CPU time of current process
*                                in hundredths of a second.
*                                This is the TCB time accumulated
*                                while running in the
*                                kernel address space.
TIMSCUTIME          DS    F      Sum of user CPU time values (as
*                                defined in TIMSUTIME) and child user
*                                CPU time values (as defined in
*                                TIMSCUTIME) for all waited-for
*                                child processes.  Zero if the
*                                current process has no waited-for
*                                children.
TIMSCSTIME          DS    F      Sum of system CPU time values (as
*                                defined in TIMSSTIME) and child
*                                system CPU time values (as defined in
*                                TIMSCSTIME) for all waited-for
*                                child processes.  Zero if the
```

```
*                                current process has no waited-for
*                                children.
TIMS#LENGTH          EQU   *-TIMS Length of this structure
** BPXYTIMS End
```

BPXYTIOS — Map the termios structure

Use PREFIX to make the labels unique. The characters specified will be appended before each label.

```
           BPXYTIOS   , PREFIX=
** BPXYTIOS: Termios structure
**  Used By: TGA TSA TFH TFW
BPXYTIOS DSECT ,                  Define DSECT
* baud rate values
B0          EQU    0              0     baud (hang-up)
B50         EQU    1              50    baud
B75         EQU    2              75    baud
B110        EQU    3              110   baud
B134        EQU    4              134.5 baud
B150        EQU    5              150   baud
B200        EQU    6              200   baud
B300        EQU    7              300   baud
B600        EQU    8              600   baud
B1200       EQU    9              1200  baud
B1800       EQU   10              1800  baud
B2400       EQU   11              2400  baud
B4800       EQU   12              4800  baud
B9600       EQU   13              9600  baud
B19200      EQU   14              19200 baud
B38400      EQU   15              38400 baud
* c_cflag offsets for baud rate.  These values are
* used to refer to the correct byte within c_cflag.  For
* instance, "MVI   C_CFLAG+ISPEED_O,B50".
OSPEED_O    EQU   0               Offset to OUTPUT baud rate
ISPEED_O    EQU   1               Offset to INPUT baud rate
* Values for c_cflag field are bitwise distinct except for
* character size bits - which form a number.
CLOCAL      EQU   X'01'           Ignore modem status lines
CREAD       EQU   X'02'           Enable receiver
CSIZE       EQU   X'30'           Character size bits
CS5         EQU   X'00'             B'00' - 5 bits/character
CS6         EQU   X'10'             B'01' - 6 bits/character
CS7         EQU   X'20'             B'10' - 7 bits/character
CS8         EQU   X'30'             B'11' - 8 bits/character
CSTOPB      EQU   X'80'           Send two stop bits, else one
HUPCL       EQU   X'01'           Hang up on last close
PARENB      EQU   X'02'           Parity enable
PARODD      EQU   X'04'           Odd parity, else even
PACKET      EQU   X'08'           Packet mode enabled
PKT3270     EQU   X'10'           3270 Passthru mode allowed
PTU3270     EQU   X'20'           3270 Passthru mode enabled
PKTXTND     EQU   X'40'           Extended Packet mode enabled
* c_cflag offsets for bits defined above.  These values are
* used to refer to the correct byte within c_cflag.  For
* instance, "TM    C_CFLAG+HUPCL_O,HUPCL".
CLOCAL_O    EQU   3
CREAD_O     EQU   3
CSIZE_O     EQU   3
CS5_O       EQU   3
CS6_O       EQU   3
CS7_O       EQU   3
CS8_O       EQU   3
CSTOPB_O    EQU   3
HUPCL_O     EQU   2
PARENB_O    EQU   2
```

```
PARODD_O    EQU   2
PACKET_O    EQU   2
PKT3270_O    EQU  2
PTU3270_O    EQU  2
PKTXTND_O    EQU  2
* Values for c_lflag field are bitwise distinct.
ECHO        EQU  X'08'             Enable echo
ECHOE       EQU  X'02'             Echo ERASE as error correcting      X
                                     backspace
ECHOK       EQU  X'04'             Echo KILL
ECHONL      EQU  X'01'             Echo new line
ICANON      EQU  X'10'             Canonical input
IEXTEN      EQU  X'20'             Enable extended functions
ISIG        EQU  X'40'             Enable signals
NOFLSH      EQU  X'80'             Disable flush after interrupt,      X
                                     quit, or suspend
TOSTOP      EQU  X'40'             Send SIGTTOU for background         X
                                     output
XCASE       EQU  X'80'             Canonical Upper/Lower               X
                                     presentation
* c_lflag offsets for bits defined above.  These values are
* used to refer to the correct byte within c_lflag.  For
* instance, "TM    C_LFLAG+TOSTOP_O,TOSTOP".
ECHO_O      EQU  3
ECHOE_O     EQU  3
ECHOK_O     EQU  3
ECHONL_O    EQU  3
ICANON_O    EQU  3
IEXTEN_O    EQU  3
ISIG_O      EQU  3
NOFLSH_O    EQU  0
TOSTOP_O    EQU  1
XCASE_O     EQU  3
* Values for c_iflag field are bitwise distinct.
BRKINT      EQU  X'01'             Signal interrupt on break
ICRNL       EQU  X'02'             Map CR to NL on input
IGNBRK      EQU  X'04'             Ignore break condition
IGNCR       EQU  X'08'             Ignore CR
IGNPAR      EQU  X'10'             Ignore characters with parity       X
                                     errors
INLCR       EQU  X'20'             Map NL to CR in input
INPCK       EQU  X'40'             Enable input parity check
ISTRIP      EQU  X'80'             Strip character
IXOFF       EQU  X'01'             Enable start/stop input             X
                                     control
IXON        EQU  X'02'             Enable start/stop output            X
                                     control
PARMRK      EQU  X'04'             Mark parity errors
IUCLC       EQU  X'08'             Map UC->LC on input
IXANY       EQU  X'10'             Any char restarts output
* c_iflag offsets for bits defined above.  These values are
* used to refer to the correct byte within c_iflag.  For
* instance, "TM    C_IFLAG+BRKINT_O,BRKINT".
BRKINT_O    EQU  3
ICRNL_O     EQU  3
IGNBRK_O    EQU  3
IGNCR_O     EQU  3
IGNPAR_O    EQU  3
INLCR_O     EQU  3
INPCK_O     EQU  3
ISTRIP_O    EQU  3
IXOFF_O     EQU  2
IXON_O      EQU  2
PARMRK_O    EQU  2
IUCLC_O     EQU  2
IXANY_O     EQU  2
* Values for c_oflag are bitwise distinct.
```

```
OPOST       EQU X'01'             Perform output processing
OLCUC       EQU X'02'             Map LC->UC on output
ONLCR       EQU X'04'             Map NL->CR on output
OCRNL       EQU X'08'             Map CR->NL on output
ONOCR       EQU X'10'             No CR at column 0
ONLRET      EQU X'20'             NL performs CR function
OFILL       EQU X'40'             Use fill chars for delay
OFDEL       EQU X'80'             Use DEL, not NUL, for fill
NLDLY       EQU X'01'             Newline delay type
NL0         EQU X'00'               NL delay type 0
NL1         EQU X'01'               NL delay type 1
TABDLY      EQU X'0C'             Tab delay type
TAB0        EQU X'00'               Tab delay type 0
TAB1        EQU X'04'               Tab delay type 1
TAB2        EQU X'08'               Tab delay type 2
TAB3        EQU X'0C'               Expand tabs to spaces
CRDLY       EQU X'30'             CR delay type
CR0         EQU X'00'               CR delay type 0
CR1         EQU X'10'               CR delay type 1
CR2         EQU X'20'               CR delay type 2
CR3         EQU X'30'               CR delay type 3
FFDLY       EQU X'40'             Form-feed delay type
FF0         EQU X'00'               FF delay type 0
FF1         EQU X'40'               FF delay type 1
BSDLY       EQU X'80'             Backspace delay type
BS0         EQU X'00'               BS delay type 0
BS1         EQU X'80'               BS delay type 1
VTDLY       EQU X'01'             Vertical-tab delay type
VT0         EQU X'00'               VT delay type 0
VT1         EQU X'01'               VT delay type 1
*
* c_oflag offsets for bits defined above.  These values are
* used to refer to the correct byte within c_oflag.  For
* instance, "TM    C_OFLAG+OPOST_O,OPOST".
*
OPOST_O     EQU 3
OLCUC_O     EQU 3
ONLCR_O     EQU 3
OCRNL_O     EQU 3
ONOCR_O     EQU 3
ONLRET_O    EQU 3
OFILL_O     EQU 3
OFDEL_O     EQU 3
NLDLY_O     EQU 2
NL0_0       EQU 2
NL1_0       EQU 2
TABDLY_O    EQU 2
TAB0_O      EQU 2
TAB1_O      EQU 2
TAB2_O      EQU 2
TAB3_O      EQU 2
CRDLY_O     EQU 2
CR0_0       EQU 2
CR1_0       EQU 2
CR2_0       EQU 2
CR3_0       EQU 2
FFDLY_O     EQU 2
FF0_0       EQU 2
FF1_0       EQU 2
BSDLY_O     EQU 2
BS0_0       EQU 2
BS1_0       EQU 2
VTDLY_O     EQU 1
VT0_0       EQU 1
VT1_0       EQU 1
* Optional actions used by tcsetattr()
TCSANOW       EQU  0              Change occurs immediately
```

```
TCSADRAIN     EQU  1              Change occurs after all output      X
                                    has been written
TCSAFLUSH     EQU  2              Change occurs after all output      X
                                    has been written and input        X
                                    has been discarded
* queue selector values for tcflush
TCIFLUSH      EQU   0             Flush data received but not read
TCOFLUSH      EQU   1             Flush data written but not sent
TCIOFLUSH     EQU   2             Flush both data received but not    X
                                    read and data written but not sent
* action values for tcflow()
TCOOFF      EQU 0                 Suspend output
TCOON       EQU 1                 Restart suspended output
TCIOFF      EQU 2                 Transmit STOP character
TCION       EQU 3                 Transmit START character
* Special Control Characters subscripts for cc_c
*   field
VINTR       EQU  0                INTR  character
VQUIT       EQU  1                QUIT  character
VERASE      EQU  2                ERASE character
VKILL       EQU  3                KILL  character
VEOF        EQU  4                EOF   character
VEOL        EQU  5                EOL   character
VMIN        EQU  6                MIN   value
VSTART      EQU  7                START character
VSTOP       EQU  8                STOP  character
VSUSP       EQU  9                SUSP  character
VTIME       EQU 10                TIME  value

NCCS      EQU   11                Number of special control chars
C_CFLAG     DC  F'0'              Control modes
C_IFLAG     DC  F'0'              Input modes
C_LFLAG     DC  F'0'              Local modes
C_OFLAG     DC  F'0'              Output modes
C_CC        DC  (NCCS)X'0'        Control characters and values
BPXYTIOS#LENGTH      EQU   *-BPXYTIOS Length of this structure
** BPXYTIOS End
```

BPXYUTSN — Map the response structure for uname

```
            BPXYUTSN   ,
** BPXYUTSN: uname() structure
**  Used By: UNA
UTSN                 DSECT ,
UTSNAMESYSNAMELEN    DS    F      Length of UTSNAMESYSNAME string
UTSNAMESYSNAME       DS    CL16   Name of this implementation of the
*                                 operating system (MVS)
UTSNAMENODENAMELEN   DS    F      Length of UTSNAMENODENAME string
UTSNAMENODENAME      DS    CL32   Name of this node within the
*                                 communications network
UTSNAMERELEASELEN    DS    F      Length of UTSNAMERELEASE string
UTSNAMERELEASE       DS    CL8    Current release level of this
*                                 implementation
UTSNAMEVERSIONLEN    DS    F      Length of UTSNAMEVERSION string
UTSNAMEVERSION       DS    CL8    Current version level of this release
UTSNAMEMACHINELEN    DS    F      Length of UTSNAMEMACHINE string
UTSNAMEMACHINE       DS    CL16   Name of the hardware type on which
*                                 the system is running
UTSN#LENGTH          EQU   *-UTSN Length of this structure
** BPXYUTSN End
```

BPXYWAST — Map the wait status word

```
           BPXYWAST   ,
** BPXYWAST: Wait status word
**  Used By: EXI MPC WAT
WAST                DSECT ,
                    DS    XL2    Reserved - set to zeros
WASTEXITSTATUS      DS    0XL2   Exit Status value passed on the
*                                BPX1EXI or BPX1MPC system calls
WASTEXITCODE        DS    0XL1   Exit return code for ending process
WASTSIGSTOP         DS    XL1    Signal that stopped process
WASTSIGTERM         DS    0XL1   Signal that terminated process
WASTSTOPFLAG        DS    XL1    Special flag value that qualifies the
*                                reason for the process being stopped
*                                or if the process is continued
*                                from stop, the value would be
*                                set to WastStopFlagContinued
* * WASTSTOPFLAG Values * * * * * * * * * * * * * * * * * * * * * *
WASTDUMP            EQU   X'80'  Bit 0 of WASTSTOPFLAG on, a core dump
*                                was taken when the process terminated
WASTSTOPFLAGSIG     EQU   X'7F'  Process stopped for a signal
WASTSTOPFLAGFORK    EQU   X'7E'  Process stopped for a fork
WASTSTOPFLAGEXEC    EQU   X'7D'  Process stopped for an exec
WASTSTOPFLAGLOCALFORK EQU  X'7B'  Process stopped for a local fork
WASTSTOPFLAGEXTENDED EQU   X'7A'  Process stopped for extended event
*
WASTSTOPFLAGCONTINUED EQU  X'79'  Process continued from stop
WASTSTOPFLAGLOAD    EQU   X'78'  Process stopped for a loadHFS
WASTSTOPFLAGDELETE  EQU   X'77'  Process stopped for a deleteHFS
WAST#LENGTH         EQU   *-WAST Length of this structure
** BPXYWAST End
```

BPXYWLM — WLM constants and parameter list DSECTs

BPXYWLM work load manager constants and DSECTs. AMODE 64 callers use "BPXYWLM — WLM constants and parameter list DSECTs" on page 1114.

```
           BPXYWLM    ,
** BPXYWLM: BPX1WLM Interface Declares
**  Used By: Callers of the BPX1WLM Interface
*
*   BPX1WLM Function Code Constants
*
WLM_QUERY_METRICS       EQU 1
WLM_QUERY_SCHEDENV      EQU 2
WLM_CHECK_SCHEDENV      EQU 3
WLM_DISCONNECT          EQU 4
WLM_DELETE_WORKUNIT     EQU 5
WLM_JOIN_WORKUNIT       EQU 6
WLM_LEAVE_WORKUNIT      EQU 7
WLM_CONNECT_WORKMGR     EQU 8
WLM_CONNECT_SERVERMGR   EQU 9
WLM_CREATE_WORKUNIT     EQU 10
WLM_CONTINUE_WORKUNIT   EQU 11
WLM_EXTRACT_WORKUNIT    EQU 12
WLM_EXPORT_WORKUNIT     EQU 13
WLM_UNDOEXPORT_WORKUNIT EQU 14
WLM_IMPORT_WORKUNIT     EQU 15
WLM_UNDOIMPORT_WORKUNIT EQU 16
WLM_QUERY_ENCLAVECLASS  EQU 17
WLM_CONNECT_EXPORTIMPORT EQU 18
*  Function codes 100-112 are reserved
ARM_BIND_THREAD               EQU 200
ARM_BLOCK_TRANSACTION         EQU 201
```

```
ARM_DESTROY_APPLICATION       EQU 202
ARM_DISCARD_TRANSACTION       EQU 203
ARM_GENERATE_CORRELATOR       EQU 204
ARM_GET_ARRIVAL_TIME          EQU 205
ARM_REGISTER_APPLICATION      EQU 206
ARM_REGISTER_METRIC           EQU 207
ARM_REGISTER_TRANSACTION      EQU 208
ARM_REPORT_TRANSACTION        EQU 209
ARM_START_APPLICATION         EQU 210
ARM_START_TRANSACTION         EQU 211
ARM_STOP_APPLICATION          EQU 212
ARM_STOP_TRANSACTION          EQU 213
ARM_UNBIND_THREAD             EQU 214
ARM_UNBLOCK_TRANSACTION       EQU 215
ARM_UPDATE_TRANSACTION        EQU 216
EWLM_CLASSIFY_CORRELATOR      EQU 217
*   BPX1WLM/BPX4WLM Parameter List Mappings
*
_WQM             DSECT ,        WLM_QUERY_METRICS Parameter List
_WQM_SYSI_PTR    DS    A        Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM system information.
*                               This data is returned in the format
*                               of the IWMWSYSI mapping macro.
_WQM_SYSI_LEN    DS    A        Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM system information
_WQM_END         DS    0C       End of WQM
*
_WQS             DSECT ,        WLM_QUERY_SCHEDENV Parameter List
_WQS_SETH_PTR    DS    A        Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM scheduling
*                               environment information.
*                               This data is returned in the format
*                               of the IWMSET mapping macro.
_WQS_SETH_LEN    DS    A        Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM scheduling environment data.
_WQS_END         DS    0C       End of _WQS
*
_WCS             DSECT ,        WLM_CHECK_SCHEDENV Parameter List
_WCS_SCH_ENV     DS    A        Address of a 16 byte character string
*                               that contains the scheduling
*                               environment to be checked.
_WCS_SYS_NAME    DS    A        Address of a 8 byte character string
*                               that contains the system name to be
*                               checked.
_WCS_END         DS    0C       End of _WCS
*
_WDC             DSECT ,        WLM_DISCONNECT Parameter List
_WDC_CONN_TKN    DS    A        Address of an fullword that contains
*                               the connect token to be disconnected
*                               from.
_WDC_END         DS    0C       End of _WDC
*
_WDW             DSECT ,        WLM_DELETE_WORKUNIT Parameter List
_WDW_ENC_TKN     DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               work unit to be deleted.
_WDW_END         DS    0C       End of _WDW
*
_WJW             DSECT ,        WLM_JOIN_WORKUNIT Parameter List
_WJW_ENC_TKN     DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               work unit to join.
_WJW_END         DS    0C       End of _WJW
```

```
*
_WLW             DSECT ,        WLM_LEAVE_WORKUNIT Parameter List
_WLW_ENC_TKN     DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               work unit to leave.
_WLW_END         DS    0C       End of _WLW
*
_WNW             DSECT ,        WLM_CONTINUE_WORKUNIT Parameter List
_WNW_ENC_TKN     DS    A        Address of a doubleword to return the
*                               the WLM enclave token of the created
*                               work unit.
_WNW_END         DS    0C       End of _WNW
*
_WCW             DSECT ,        WLM_CREATE_WORKUNIT Parameter List
_WCW_ENC_TKN     DS    A        Address of a doubleword to return the
*                               the WLM enclave token of the created
*                               work unit.
_WCW_CLASSIFY    DS    A        Address of a fullword pointer that
*                               contains the address of a IWMCLSFY
*                               Parameter List.
_WCW_ARR_TIME    DS    A        Address of a doubleword field that
*                               contains the arrival time of the
*                               work request in STCK format.
_WCW_FUNC_NAME   DS    A        Address of a 8 byte character string
*                               that contains the descriptive function
*                               name of the work request.
_WCW_END         DS    0C       End of _WCW
*
_WSC             DSECT ,        WLM_CONNECT_SERVERMGR Parameter List
_WSC_SUB_SYS     DS    A        Address of a 4 byte character string
*                               that contains the subsystem type the
*                               server manager is requesting connection
*                               for.
_WSC_SUB_SYS_NM  DS    A        Address of a 8 byte character string
*                               that contains the subsystem name the
*                               server manager is requesting connection
*                               for.
_WSC_APPL_ENV    DS    A        Address of a 32 byte character string
*                               that contains the application
*                               environment name associated with the
*                               server.
_WSC_PAR_EU      DS    A        Address of a fullword that contains
*                               number of parallel execution units
*                               in the server environment.
_WSC_END         DS    0C       End of _WSC
*
_WWC             DSECT ,        WLM_CONNECT_WORKMGR Parameter List
_WWC_SUB_SYS     DS    A        Address of a 4 byte character string
*                               that contains the subsystem type the
*                               work manager is requesting connection
*                               for.
_WWC_SUB_SYS_NM  DS    A        Address of a 8 byte character string
*                               that contains the subsystem name the
*                               work manager is requesting connection
*                               for.
_WWC_END         DS    0C       End of _WWC
*
_WEW             DSECT ,        WLM_EXTRACT_WORKUNIT Parameter List
_WEW_ENC_TKN     DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               active work unit.
_WEW_END         DS    0C       End of _WEW
*
_WXW             DSECT ,        WLM_EXPORT_WORKUNIT Parameter List
_WXW_ENC_TKN     DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               work unit to be exported.
```

```
_WXW_EXP_TKN     DS    A        Address of the 32 bytes to return the
*                               WLM export token of the exported work
*                               unit.
_WXW_CONN_TKN    DS    A        Address of a fullword that contains
*                               the connect token associated with the
*                               workmanager.
_WXW_END         DS    0C       End of _WXW
*
_WUXW            DSECT ,        WLM_UNEXPORT_WORKUNIT Parameter List
_WUXW_EXP_TKN    DS    A        Address of the 32 bytes that contains
*                               the WLM export token representing the
*                               exported work unit.
_WUXW_CONN_TKN   DS    A        Address of a fullword that contains
*                               the connect token associated with the
*                               workmanager.
_WUXW_END        DS    0C       End of _WUXW
*
_WIW             DSECT ,        WLM_IMPORT_WORKUNIT Parameter List
_WIW_EXP_TKN     DS    A        Address of the 32 bytes that contains
*                               the WLM export token representing the
*                               exported work unit.
_WIW_ENC_TKN     DS    A        Address of a doubleword to return the
*                               WLM enclave token of the imported work
*                               unit.
_WIW_CONN_TKN    DS    A        Address of a fullword that contains
*                               the connect token associated with the
*                               workmanager.
_WIW_END         DS    0C       End of _WIW
*
_WUIW            DSECT ,        WLM_UNIMPORT_WORKUNIT Parameter List
_WUIW_EXP_TKN    DS    A        Address of the 32 bytes that contains
*                               the WLM export token representing the
*                               imported work unit.
_WUIW_CONN_TKN   DS    A        Address of a fullword that contains
*                               the connect token associated with the
*                               workmanager.
_WUIW_END        DS    0C       End of _WUIW
*
_WQEC            DSECT ,        WLM_QUERY_ENCLAVECLASS Parameter List
_WQEC_ENC_TKN    DS    A        Address of a doubleword that contains
*                               the WLM enclave token representing the
*                               work unit to be queried.
_WQEC_SYSEC_PTR  DS    A        Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM Query Enclave Data.
*                               This data is returned in the format
*                               of the IWMECD mapping macro.
_WQEC_SYSEC_LEN  DS    A        Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM Query Enclave Data.
_WQEC_END        DS    0C       End of WQEC
*
_WCEI            DSECT ,        WLM_CONNECT_EXPORTIMPORT Parameter List
_WCEI_SUB_SYS    DS    A        Address of a 4 byte character string
*                               that contains the subsystem type the
*                               work manager is requesting connection
*                               for.
_WCEI_SUB_SYS_NM DS    A        Address of a 8 byte character string
*                               that contains the subsystem name the
*                               work manager is requesting connection
*                               for.
_WCEI_END        DS    0C       End of _WCEI
*
_ABI             DSECT ,        ARM_BIND_THREAD Parameter
*                               List
_ABI_CONTEXT     DS    A        Reserved.
*                               Must be zero.
```

```
_ABI_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_ABI_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ABI_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ABI_END         DS    0C       End of _ABI
*
_ABT             DSECT ,        ARM_BLOCK_TRANSACTION Parameter
*                               List
_ABT_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ABT_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_ABT_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ABT_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ABT_BLOCK_HDL   DS    A        Address of a fullword pointer that
*                               contains the address of the 8 byte
*                               field to return the block handle.
_ABT_END         DS    0C       End of _ABT
*
_ADA             DSECT ,        ARM_DESTROY_APPLICATION Parameter
*                               List
_ADA_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ADA_APPL_ID     DS    A        Address of a 16 byte field
*                               that contains the application ID.
_ADA_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ADA_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ADA_END         DS    0C       End of _ADA
*
_ADT             DSECT ,        ARM_DISCARD_TRANSACTION Parameter
*                               List
_ADT_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ADT_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_ADT_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ADT_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ADT_END         DS    0C       End of _ADT
*
_AGC             DSECT ,        ARM_GENERATE_CORRELATOR Parameter
*                               List
_AGC_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AGC_APP_HDL     DS    A        Address of a 8 byte field that
*                               contains the application handle.
_AGC_TRAN_ID     DS    A        Address of a 16 byte field that
*                               contains the transaction ID.
_AGC_PAR_CORR    DS    A        Address of a data area that
*                               contains the parent correlator.
_AGC_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AGC_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AGC_CUR_CORR    DS    A        Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the current correlator.
_AGC_END         DS    0C       End of _AGC
*
_AGT             DSECT ,        ARM_GET_ARRIVAL_TIME Parameter
```

```
*                               List
_AGT_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AGT_TIMESTAMP   DS    A        Address of a fullword pointer that
*                               contains the address of a 64 bit
*                               field to return the arrival time.
_AGT_END         DS    0C       End of _AGT
*
_ARA             DSECT ,        ARM_REGISTER_APPLICATION Parameter
*                               List
_ARA_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ARA_APP_NAME    DS    A        Address of a character string that
*                               contains the application name.
_ARA_IN_APP_ID   DS    A        Address of a 16 byte field that
*                               contains an input application ID.
_ARA_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ARA_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ARA_OUT_APP_ID  DS    A        Address of a fullword pointer that
*                               contains the address of a 16 byte
*                               field to return the output
*                               application ID.
_ARA_END         DS    0C       End of _ARA
*
_AMR             DSECT ,        ARM_REGISTER_METRIC Parameter
*                               List
_AMR_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AMR_APP_ID      DS    A        Address of a 16 byte field that
*                               contains the application ID.
_AMR_MET_NAME    DS    A        Address of a character string that
*                               contains the metric name.
_AMR_MET_FORMAT  DS    A        Address of a 1 byte field that
*                               contains the metric format.
_AMR_MET_USAGE   DS    A        Address of a 2 byte field that
*                               contains the metric usage.
_AMR_UNIT        DS    A        Address of a character string that
*                               contains the units of the metric.
_AMR_IN_MET_ID   DS    A        Address of a 16 byte field that
*                               contains an input metric ID.
_AMR_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AMR_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AMR_OUT_MET_ID  DS    A        Address of a fullword pointer that
*                               contains the address of a 16 byte
*                               field to return the output
*                               metric ID.
_AMR_END         DS    0C       End of _AMR
*
_ART             DSECT ,        ARM_REGISTER_TRANSACTION Parameter
*                               List
_ART_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ART_APP_ID      DS    A        Address of a 16 byte field that
*                               contains the application ID.
_ART_TRAN_NAME   DS    A        Address of a character string that
*                               contains the transaction name.
_ART_IN_TRAN_ID  DS    A        Address of a 16 byte field that
*                               contains an input transaction ID.
_ART_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ART_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ART_OUT_TRAN_ID DS    A        Address of a fullword pointer that
```

```
*                               contains the address of a 16 byte
*                               field to return the output
*                               transaction ID.
_ART_END         DS    0C       End of _ART
*
_ATR             DSECT ,        ARM_REPORT_TRANSACTION Parameter
*                               List
_ATR_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ATR_APP_HDL     DS    A        Address of a 8 byte field that
*                               contains the application handle.
_ATR_TRAN_ID     DS    A        Address of a 16 byte field that
*                               contains the transaction ID.
_ATR_TRAN_STA    DS    A        Address of a 4 byte field that
*                               contains the transaction status.
_ATR_RESP_TIME   DS    A        Address of a 64 bit field that
*                               contains the response time.
_ATR_STOP_TIME   DS    A        Address of a 64 bit field that
*                               contains the stop time.
_ATR_PAR_CORR    DS    A        Address of a data area that
*                               contains the parent correlator.
_ATR_CUR_CORR    DS    A        Address of a data area that
*                               contains the current correlator.
_ATR_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ATR_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ATR_END         DS    0C       End of _ATR
*
_AAS             DSECT ,        ARM_START_APPLICATION Parameter
*                               List
_AAS_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AAS_APP_ID      DS    A        Address of a 16 byte field that
*                               contains the application ID.
_AAS_APP_GRP     DS    A        Address of a character string that
*                               contains the application group
*                               name.
_AAS_APP_INS     DS    A        Address of a character string that
*                               contains the application
*                               instance name.
_AAS_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AAS_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AAS_APP_HDL     DS    A        Address of a fullword pointer that
*                               contains the address of the 8 byte
*                               field to return the application
*                               handle.
_AAS_END         DS    0C       End of _AAS
*
_AST             DSECT ,        ARM_START_TRANSACTION Parameter
*                               List
_AST_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AST_APP_HDL     DS    A        Address of a 8 byte field that
*                               contains the application handle.
_AST_TRAN_ID     DS    A        Address of a 16 byte field that
*                               contains the transaction ID.
_AST_PAR_CORR    DS    A        Address of a data area that
*                               contains the parent correlator.
_AST_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AST_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AST_TRAN_HDL    DS    A        Address of a fullword pointer that
*                               contains the address of the 8 byte
```

```
*                               field to return the transaction
*                               handle.
_AST_CUR_CORR    DS    A        Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the current correlator.
_AST_END         DS    0C       End of _AST
*
_APA             DSECT ,        ARM_STOP_APPLICATION Parameter
*                               List
_APA_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_APA_APP_HDL     DS    A        Address of a 8 byte field that
*                               contains the application handle.
_APA_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_APA_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_APA_END         DS    0C       End of _APA
*
_APT             DSECT ,        ARM_STOP_TRANSACTION Parameter
*                               List
_APT_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_APT_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_APT_TRAN_STA    DS    A        Address of a 4 byte number that
*                               contains the transaction status.
_APT_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_APT_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_APT_END         DS    0C       End of _APT
*
_AUB             DSECT ,        ARM_UNBIND_THREAD Parameter
*                               List
_AUB_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AUB_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_AUB_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AUB_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AUB_END         DS    0C       End of _AUB
*
_AUT             DSECT ,        ARM_UNBLOCK_TRANSACTION Parameter
*                               List
_AUT_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AUT_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_AUT_BLOCK_HDL   DS    A        Address of a 8 byte field
*                               that contains the block handle.
_AUT_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_AUT_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AUT_END         DS    0C       End of _AUT
*
_AUP             DSECT ,        ARM_UPDATE_TRANSACTION Parameter
*                               List
_AUP_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_AUP_TRAN_HDL    DS    A        Address of a 8 byte field that
*                               contains the transaction handle.
_AUP_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
```

```
_AUP_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_AUP_END         DS    0C       End of _AUP
*
_ACC             DSECT ,        EWLM_CLASSIFY_CORRELATOR Parameter
*                               List
_ACC_CONTEXT     DS    A        Reserved.
*                               Must be zero.
_ACC_APP_HDL     DS    A        Address of a 8 byte field that
*                               contains the application handle.
_ACC_TRAN_ID     DS    A        Address of a 16 byte field that
*                               contains the transaction ID.
_ACC_FLAGS       DS    A        Address of a 4 byte field
*                               that contains flags.
_ACC_BUFFER4     DS    A        Address of a data area that
*                               contains additional input data.
_ACC_CLASS_CORR  DS    A        Address of a fullword pointer that
*                               contains the address of the buffer to
*                               return the classify correlator.
_ACC_END         DS    0C       End of _ACC
** BPXYWLM End
```

BPXYWNSZ — Map the winsize structure

BPXYWNSZ maps window/terminal size information. It corresponds to the C winsize structure, which is in sys/ioctl.h.

```
           BPXYWNSZ   ,
** BPXYWNSZ: Winsize structure
**  Used By: ioctl with TIOCGWINSZ and TIOCSWINSZ
BPXYWNSZ DSECT ,                  Define DSECT
WS_ROW       DC  H'0'             Rows, in characters
WS_COL       DC  H'0'             Columns, in characters
WS_XPIXEL    DC  H'0'             Horizontal size, pixels
WS_YPIXEL    DC  H'0'             Vertical size, pixels
BPXYWNSZ#LENGTH      EQU   *-BPXYWNSZ Length of this structure
** BPXYWNSZ End
```

BPXZOAPB — z/OS UNIX address space per-process extension

BPXZOAPB maps z/OS UNIX space per-process extension. Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OapbDefaultUseridLen
- OapbDefaultUserid
- OapbDefaultGroupidLen
- OapbDefaultGroupid

```
           BPXZOAPB   ,
OAPB     DSECT ,
OAPB1    DS    1CL0256
*
*
OAPB2    DS    1CL0020
         ORG   OAPB2
OAPBDEFAULTUSERIDLEN DS 1FL1 Length of default userid
OAPBDEFAULTUSERID DS 1CL0008 Default userid
OAPBDEFAULTGROUPIDLEN DS 1FL1 Length default groupid
OAPBDEFAULTGROUPID DS 1CL0008 Default groupid
         DS    1CL0002   reserved
OAPB3    DS    1CL0132
OAPB_LEN EQU   *-OAPB
```

BPXZOCVT — Base control block for z/OS UNIX

BPXZOCVT maps addresses of common areas for use by z/OS UNIX subcomponents. Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OcvtKernelReady

```
           BPXZOCVT   ,
OCVT        DSECT ,
            DS    CL60
OCVTFLGS    DS    0FL4               Offset +3C
OCVTFLGSB1  DS    0BL1               Offset +3C
OCVTKERNELREADY EQU X'20'     z/OS UNIX Kernel Ready to accept system calls
         ORG   OCVTFLGSB1+X'00000001'
OCVTFLGSB2  DS    1BL1
OCVTFLGSB3  DS    1BL1
OCVTFLGSB4  DS    1BL1
         ORG   OCVTFLGS+X'00000004'
            DS    CL680
OCVTUIWRWCDS DS   CL8                Offset +2E8
```

BPXZOTCB — z/OS UNIX extension to the TCB

BPXZOTCB maps z/OS UNIX extensions to the TCB.

Only the following fields are externally documented. All other fields are reserved for IBM use only.

- OtcbThli
- OtcbWLMEToken
- OtcbSigPending
- OtcbOapb

```
           BPXZOTCB   ,
OTCB     DSECT ,
OTCBID   DS    CL4       EBCDIC ID - OTCB
OTCBSP   DS    X         Subpool number of this OTCB
OTCBLEN  DS    FL3       Length of this OTCB
OTCBPTXL DS    A         -> pthread parameters
OTCBKSER DS    A         -> KSER
OTCBMEDCLEAR DS 0CL84    Section of Otcb we clear for medium weight    X
                         processes
OTCBFLAGS DS   0BL4      Compare and swap flg
OTCBFLAGSB1 DS 0B
OTCBINITIALTHREAD EQU X'80' Initial thread of a process
OTCBINKERNELCALL EQU X'40' moved to PPRT
OTCBSLEEP EQU  X'20'     Signal sleep() flag which is checked by       X
                         pause().
OTCBCALLEDKERNEL EQU X'10' At sometime in its life, this thread has    X
                         made a system call /CS
OTCBNOPTLSIR EQU X'08'   Signal is being sent from the ptrace PtlSir   X
                         (Ptrace Signal Interface Routine), so signal  X
                         delivery should not deliver the signal to the X
                         PtlSir if ptrace mode is on (we're already    X
                         there)
OTCBPROCESSCLEANUP EQU X'04' Process being torn down. /CS
OTCBINTASKTERM EQU X'02' Thread is in the process of task termination. X
                         Set by BPXRRTRM during task term
OTCBBYPASSRACF EQU X'01' Do not do RACF check in kill() routine
         ORG   OTCBFLAGSB1+1
OTCBFLAGSB2 DS 0B
OTCBPTEXITONLY EQU X'80' Thread did XAG exitonly
```

```
OTCBTHREADPTEXITED EQU X'40' Marked Ptexited
OTCBTHREADTERM EQU X'20' Thread in terminated state
OTCBIPT  EQU   X'10'     Indicates this thread is or was the Initial   X
                         Pthread Task, used by BPXPRMPC to check for   X
                         IPT cleanup
OTCBPROCESSCREATOR EQU X'08' 1=>Indicates the dubbing of this thread   X
                         caused the creation of the process
OTCBCANCELINTR EQU X'04' Cancel interrupt point
OTCBQUIESCEPOSTED EQU X'02' This task posted by qut
OTCBDUBNEWPROCESS EQU X'01' 0=>Dub as thread, 1=>Dub as process
         ORG   OTCBFLAGSB2+1
OTCBFLAGSB3 DS 0B
OTCBATTACHEXEC EQU X'80' attach_exec in progress
OTCBMULTIPROCCLP EQU X'40' 1=> Lower level processes are to be cleaned X
                         up by this thread
OTCBACTIVEACEEMANAGED EQU X'20' 1=Active ACEE managed by RACF          X
                         (initACEE)
OTCBTOGGLEACEEMANAGED EQU X'10' 1=Toggled ACEE managed by RACF         X
                         (initACEE)
OTCBSAVEDACEEMANAGED EQU X'08' MrPwd saved ACEE managed
OTCBINPROCESSTERM EQU X'04' 1=> When PRTRM is terminating a process.   X
                         Used to tell F.S. Termination when PRTRM is   X
                         cleaning up.
OTCBTASKACEEUSP EQU X'02' USP created by TLS_TASK_ACEE#
OTCBMRPWDUIDSET EQU X'01' OtcbMrPwdUID field set
         ORG   OTCBFLAGSB3+1
OTCBFLAGSB4 DS 0B
OTCBPSEUDODUBBED EQU X'80' Thread is a pseudo-dubbed kernel task
OTCBTASKSEC EQU X'40'    Thread called BPX1TLS to build a task level   X
                         Acee
OTCBENCLAVEOWNER EQU X'20' Thread is an owner of a WLM Enclave
OTCBWLMEMANAGED EQU X'10' Enclave managed by WLM
OTCBTASKACEEINIT EQU X'08' InitUsp done for Task Level ACEE
OTCBDUBTASKACEE EQU X'04' 0=>Don't Dub Task Level ACEE 1=>Dub Task     X
                         Level ACEE
OTCBPTCREACEE EQU X'02'  0=> No ACEE propagated on Ptcre 1=> ACEE was  X
                         propagated
OTCBPROCINITACEE EQU X'01' 0=> No INITACEE done during dub 1=>         X
                         INITACEE done during dub
         ORG   OTCBFLAGS+4
OTCBTHID DS    0CL8      Thread ID
OTCBPPRT DS    0A        -> PPRT
OTCBLIGHTWEIGHT EQU X'80' 1 Light weight thread
         ORG   OTCBPPRT+4
OTCBSEQNO DS   0F        Sequence number
OTCBSEQNOHIGHERHALF DS H higher half of seq num
OTCBSEQNOLOWERHALF DS H  lower half of seq num
OTCBSIGFLAGS DS 0BL4     Signal Flags1 that are modified by signal     X
                         IRBs. Serialized by Compare & Swap
OTCBSIGFLAGSB1 DS 0B
OTCBSIGDISABLE EQU X'80' Signal Delivery is disabled
OTCBSIGPENDING EQU X'40' Signal pending flag
OTCBTIMERSIGNAL EQU X'20' SIGXCPU or SIGKILL is to be generated by the X
                         syscall layer. Either OtcbSIGXCPU or          X
                         OtcbSIGKILL is on. This flag exists for       X
                         sycall layer performance.
OTCBSIGALRM EQU X'10'    generate in SC layer
OTCBALRMACTIVE EQU X'08' ALR & setitimer REAL
OTCBIGNRBSTATE EQU X'04' Ignore RB state
OTCBSIGDUMP EQU X'02'    Dump for terminating signal
OTCBRAISETIMERIRB EQU X'01' Raise() function from Timer IRB
         ORG   OTCBSIGFLAGSB1+1
OTCBSIGFLAGSB2 DS 0B
OTCBIRBSIGNAL EQU X'80'  Signal Checker routine should recheck signals X
                         because one of the signal IRBs may have       X
                         changed the signals pending
OTCBPTDELAYIRB EQU X'40' Delay IRB for PTRACE
```

```
OTCBSIRDISABLE EQU X'20' Disable invocation of SIR
OTCBCANCELASYNC EQU X'10' Cancellation request state 0 = controlled, 1 X
                       = async
OTCBCANCELDISABLED EQU X'08' Cancellation request type 0=enabled,      X
                       1=disabled
OTCBSETSIGDISABLE EQU X'04' Syscall must turn on OtcbSigDisable on     X
                       return
OTCBCTWACTIVE EQU X'02'  cond_timed_wait (BPX1CTW) is active
OTCBIGNDLVKEY EQU X'01'  Ignore Dlv key
         ORG   OTCBSIGFLAGSB2+1
OTCBSIGFLAGSB3 DS 0B
OTCBCANCELPENDING EQU X'80' Cancel pending for thrd
OTCBPTREGSINUSTA EQU X'40' Ptrace regs/PSW are in the Usta
OTCBSIGXCPU EQU X'20'    SIGXCPU is to be generated by syscall layer
OTCBSIGKILL EQU X'10'    SIGKILL is to be generated by syscall layer
OTCBSIGVTALRM EQU X'08'  generate in SC layer
OTCBSIGPROF EQU X'04'    generate in SC layer
OTCBALLSIGSBLOCKED EQU X'02' All signals are blocked, the same as if   X
                       all bits were on in PpstSigMask
OTCBUDPINKERNEL EQU X'01' In UDP syscall
         ORG   OTCBSIGFLAGSB3+1
OTCBSIGFLAGSB4 DS 0B
OTCBVTALRMACTIVE EQU X'80' setitimer VIRTUAL
OTCBPROFACTIVE EQU X'40' setitimer PROF
OTCBRETURNPPSD EQU X'20'
OTCBCALLRTM EQU X'10'    CallRTM done by IR1
OTCBNOIRB EQU  X'08'     Avoid Irb interrupts
OTCBREDRIVE EQU X'04'    IRB redrive is in prog
OTCBCPUTIMEOUT EQU X'02' A terminating signal is to be generated due   X
                       to the process time limit being exceeded
OTCBDLVTERM EQU X'01'    This thread is terminating due to a           X
                       terminating signal
         ORG   OTCBSIGFLAGS+4
OTCBSIR2ID DS  F         Alarm ID set by STIMERM. Changed by           X
                       incrementing at start of alarm() and sleep()
OTCBRACGROUP DS A        Pointer to RACF structure to be deleted by    X
                       next getgr* call
OTCBRACPASSWD DS A       Pointer to RACF structure to be deleted by    X
                       next getpw* call
OTCBCOMMREQ DS A         Address of communications resource associated X
                       with this task (only valid when there is an   X
                       active request)
OTCBPTMULTISTATUS DS B   Ptrace multi process mode status word value - X
                       will be one of the WastStopFlag... values
OTCBFLAGS2 DS  0CL3      2nd Set of flags
OTCBFLAGS2B1 DS 0B
OTCBSTAXDEFERRED EQU X'80' Stax defer performed
OTCBLUKWKEY0 EQU X'40'   User Kernwait caller is KEY 0
OTCBKSERWAITINGF EQU X'20' Is Kser waiting?
OTCBAFFINPGMRUNNING EQU X'10' An IPT/thread affinity program is        X
                       running on this thread, do no joblogging
OTCBREGSINPPSD EQU X'08' User regs are in Ppsd at time of              X
                       Freeze_This_Thread
OTCBREGSINIRB EQU X'04'  User regs are in IRB at time of               X
                       Freeze_This_Thread
OTCBTIMEDKERNWAIT EQU X'02' Task is in Timed Kernwait
OTCBSLOWPATHSYSCALL EQU X'01' This is a slow-path syscall. User regs   X
                       are in USTA
         ORG   OTCBFLAGS2B1+1
OTCBFLAGS2B2 DS 0B
OTCBOSENVACTIVE EQU X'80' Task is active in the osenv
OTCBOSENVGET EQU X'40'   Task issued osenv_get
OTCBOSENVWLMJOIN EQU X'20' Task is joined to a WLM enclave as a result X
                       of osenv_set
OTCBOSENVSECURITY EQU X'10' Security environment was saved by osenv
OTCBCHKPTUNSAFE EQU X'08' Task is checkpoint unsafe due to being in    X
                       kernel
```

```
OTCBINRSTWAIT EQU X'04'  Task is waiting for OMVS to be restarted
OTCBDORSTWAIT EQU X'02'  Fastpath syscall requests task be put in      X
                         restart wait
OTCBSPBUPDATE EQU X'01'  Used for SPB/IR1 serialization
         ORG   OTCBFLAGS2B2+1
OTCBFLAGS2B3 DS 0B
OTCBDEFERSIGS EQU X'80'  Defer sigs is in effect
OTCBLATCHPROBLEM EQU X'40' Latch Cleanup Problem detected at           X
                         termination time
OTCBF2_ATTACHEDTASK EQU X'20' This task has done a localspawn or       X
                         attach
OTCBPTHDFORKCHILD EQU X'10' This child process was created via fork    X
                         from a pthread
         ORG   OTCBFLAGS2+3
OTCBALRMGTYEAR DS F      Alarm time in seconds greater than 365 days   X
                         used by alarm() and sleep() functions
OTCBCOFPTR DS  A         Address of CopyOnFork area
OTCBDLVIRB DS  A         Address of RB that called Signal Delivery
OTCBDUBRBSQN DS F        Sequence number of RB that was DUBed
OTCBREGRBSQN DS F        Sequence number of RB that registered for     X
                         signals
OTCBSPB
                         the ALET for this SPB is PRIMARY. For         X
                         BPXJCSA, the ALET is HOME.
OTCBSYSCALLCODE DS F     System call number
OTCBLECB DS    A         Ptr to ECB used to wait for a latch to be     X
                         obtained
OTCBPPSDPTR DS 0A        -> PPSD
OTCBPPSD DS    A         -> PPSD
OTCBCTWID DS   F         cond_timed_wait stimerm ID
OTCBSTACKNONSW DS A      Dynamic stack for Non-space switched          X
                         syscalls. Only valid when OTCBSYSCALLCODE is  X
                         non-zero. Contains address of 1st #SAMAP area X
                         following RUCA. Addressable in user home      X
                         space.
OTCBOTIM DS    A         -> interval timers
OTCBOAPB DS    A         -> OAPB
OTCBMEDCLEAR2 DS 0CL248  We can't clear Oapb, multiproc quiesce        X
                         references
OTCBPTPICPARMSPTR DS A   Pointer to ptrace recovery environment        X
                         parameters (PIC parms
OTCBPTEVENTID DS F       Ptrace event ID, that identifies why this     X
                         thread stopped for ptrace
OTCBPTLCLPPSDPTR DS A    Ptrace local Ppsd pointer
OTCBMVSPAUSEECBLIST DS A Pointer to the BPXZECBL - System copy of user X
                         ECB addresses passed to MVSpauseInit
OTCBSAVEDSCB DS A        Saved SCB addr of STAI on entry to Local      X
                         Child Process
OTCBUECBLIST DS A        Pointer to the BPXZECBL - System copy of user X
                         and system ECBs address for the BPXLUKW -     X
                         User KernWait service
OTCBUIDS DS    0CL12     User IDs for Thread
OTCBRUID DS    F         Real Uid
OTCBEUID DS    F         Effective Uid
OTCBSUID DS    F         Saved Uid
OTCBSAVEDACEE DS A       MRPWD saved Acee
OTCBPPRX DS    A         Address of the Pprx, an extension of the Pprt
OTCBMRPWDUID DS F        Password verified UID
OTCBPSWBYT03 DS F        Caller's PSW bytes 0-4 (Used by JCPR to setup X
                         BPXZUSTA)
OTCBMRPWDUSERNAME DS CL8 Password verified userid
OTCBSAVEDSECENV DS A     Pointer to ACEE saved by BPX1ENV for a toggle X
                         request
OTCBMVSUSERIDPTR DS A    Pointer to userid of this thread, points to   X
                         either OtcbLoginNInfo or OasbLoginNInfo
OTCBLOGINNINFO DS 0CL13  Task userid and length
OTCBLOGINNLEN DS F       Task userid length
```

```
OTCBLOGINNAME DS CL9     Tasks userid, must be '00'x (null)           X
                         terminated. Preceding length does not include X
                         ' terminating null
OTCBPRIN2FLAGS DS B      This field is modified by BPXPRIN1, and it is X
                         used by BPXPRIN2. See PPSQ for the mapping    X
                         and more details
         DS    CL2       Reserved, keep word bdy
OTCBTHLI DS    A         -> Thli. This field must never change offsets X
                         within the Otcb since the Thli is an external X
                         control block and the user will have to go    X
                         through the Otcb to get to the Thli
OTCBACTSCTBNODEPTR DS A  Active Acee SCTB node ptr, zero if ACEE is    X
                         private
OTCBTOGGLEDSCTBNODEPTR DS A Toggled Acee SCTB node ptr, zero if ACEE   X
                         is private
OTCBPAG  DS    F         Process Auth Groups
OTCBGIDS DS    0CL12     Group IDs for Thread
OTCBRGID DS    F         Real Gid
OTCBEGID DS    F         Effective Gid
OTCBSGID DS    F         Saved Gid
OTCBRACGIDSPTR DS A      Addr of saved group list
OTCBWLMETOKEN DS BL8     WLM Enclave token associated with the thread
OTCBSAVEDGID DS F        Gid set by getpwname, used by setgid
OTCBALIASNINFO DS 0CL13
OTCBALIASNLEN DS F       Task alias length
OTCBALIASNAME DS CL9     Tasks alias, must be '00'x (null) terminated. X
                         Preceding length does not include '           X
                         terminating null
         DS    CL3       Reserved, keep word bdy
OTCBOSENVTOKEN DS 0CL8   osenv environment token
OTCBOSENVCELLPTR DS A    Ptr to osenv environment cell element
OTCBOSENVSEQN DS F       seq number associated with the osenv cell
OTCBREDRIVETIME DS F     Time to delay signal IRB
OTCBSHLLOADERINFO DS 0CL48 Shared Library data
OTCBSHLLOADER1DSPADDR DS A Shared Library loader data1 addr in data    X
                         space
OTCBSHLLOADER1DSPPAGES DS F Shared Library loader data1 len in data    X
                         space
OTCBSHLLOADER1DSPALET DS F Shared Library loader data1 alet of data    X
                         space
OTCBSHLLOADER1DSPSTOKEN DS CL8 Shared Library loader data1 stkn of     X
                         data space
OTCBSHLLOADER2DSPADDR DS A Shared Library loader data1 addr in data    X
                         space
OTCBSHLLOADER2DSPPAGES DS F Shared Library loader data1 len in data    X
                         space
OTCBSHLLOADER2DSPALET DS F Shared Library loader data1 alet of data    X
                         space
OTCBSHLLOADER2DSPSTOKEN DS CL8 Shared Library loader data1 stkn of     X
                         data space
OTCBSHLLOADERTOK DS CL8  Shared Library loader token len used by       X
                         BPXXSHLB INIT
OTCBSMKLATCHFLAGCOUNT DS F OcvtSMKLatchCount incremented
OTCBSAVEPPRT DS A        Addr of Pprt saved during task term when      X
                         OtcbPprt is changed so that BPXMIPCE can find X
                         real Pprt of running task
OTCBSYSCALLSTART DS BL8  Start Timeused Value for active syscall
OTCBPECBPTR DS A         Prt to a PECB
         DS    CL24      Reserved for future use
*
*
*     NOTE: The size of this control block is retrieved dynamically
*           during runtime by the modules that need it.
*           When adding additional fields to this control block,
*
*     ====> THE ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
*
```

```
*
*
OTCB_END DS    0C        Make CB end on doubleword
OTCB_LEN EQU   *-OTCB
*
* ********************************************************************
* *                                                                  *
* * NOTE: The "OtcbCopyOnFork" is contiguous to the end of the Otcb. *
* *       IPCS modules respecify its base on the address of Otcb_End *
* *       because the field OtcbCofPtr does NOT contain an address   *
* *       that is usable by IPCS without another ?ASAXACC.           *
* *                                                                  *
* ********************************************************************
*
*
OTCBCOPYONFORK DSECT     These fields will be copied to the child Otcb X
                         on fork()
OTCBSYSCALLENTRYSTATUS DS A Ptr to regs and stuff at entry to the      X
                         syscall layer
OTCBUSTAPTR DS A         Pointer to user status area containing the    X
                         syscall issuer's regs and PSW. (Mapped by     X
                         BPXZUSTA. Used by ptrace.)
OTCBGROUPDBSEARCH DS CL8 Group name for group data base search         X
                         (getgrent)
OTCBUSERDBSEARCH DS CL8  User name for user data base search           X
                         (getpwent)
OTCBSTORAGEFORCPR DS A   Dynamic area for BPXJCPR
OTCBSTORAGEFORCPRKEY DS X Storage key of dynamic area for JCPR
OTCBSTORAGEFORCPRLEN DS FL3 Length of JCPR dynamic area
OTCBSTORAGEFORCPRSP DS X Storage SP of dynamic area
OTCBCOPYFLAGS DS 0B      Copy on Fork Flags
OTCBSETUIDEXEC EQU X'80' Setuid Exec in progress
OTCBMVSAUTHLIB EQU X'40' Pgm loaded from MVS authorized library by     X
                         exec/execmvs
OTCBEXECPROCESS EQU X'20' This process image was created by            X
                         exec/execmvs
OTCBSIGPOSTINPC EQU X'10' Sig IRB will post ThliComECB even when a     X
                         Linkage Stack is detected, signal will be     X
                         rescheduled for future delivery
OTCBPOEATTRSET EQU X'08' The OtcbPoeAttr have been set by __poe
         ORG   OTCBCOPYFLAGS+1
         DS    CL2       Reserved
OTCBDAEMONINFO DS 0CL12  This info will be set and reused to improve   X
                         performance by bypassing multiple RACF calls  X
                         for the same info
OTCBSAVEDUID DS F        Remembered UID
OTCBSAVEDUSERNAME DS CL8 Remembered UserID
OTCBWLMUSERDATAINFO DS 0CL16 WLM Server information. These fields are  X
                         needed to manage the application data and     X
                         file descriptor list. A single area is        X
                         obtained for both needs. The area is pointed  X
                         to by OtcbWLMUserDataPtr. The application     X
                         data is always first in the area followed by  X
                         the FDL area.
OTCBWLMUSERDATAPTR DS A  Address of user data storage
OTCBWLMUSERDATALEN DS F  Length of user data storage -- the entire     X
                         area
OTCBWLMUSERDATAKEY DS X  Key of user data area
         DS    CL3       reserved
OTCBWLMAPPLLEN DS F      Length of the application data portion of the X
                         user data area. The file descriptors occupy   X
                         the remainder of the user data area
OTCBSMFBUFFPTR DS A      Address of key 0 copy of user SMF record
OTCBSMFBUFFLEN DS F      Length of key 0 copy of user SMF record
OTCBPOEATTR DS 0CL96     __poe() port of entry info
OTCBPOEPEERIPADDR DS 0CL16 Peer IP Address
OTCBPOEPEERIPV6PREFIX DS CL12
```

```
OTCBPOEPEERIPV4ADDR DS F
OTCBPOETERMID DS CL8     TERMINAL Profile Name
OTCBPOELABEL DS CL8      Security Label of poe
OTCBPOEPROFILE DS CL64   SERVAUTH Profile Name
*
*      ************************************************************
*      *                                                          *
*      * NOTE: The size of this control block is retrieved       *
*      * dynamically during runtime by the modules that need it. When*
*      * adding additional fields to this control block, ===> THE  *
*      * ONLY MODULE THAT *MUST* BE RECOMPILED IS BPXPRIT
*      *                                                          *
*      ************************************************************
*
*
OTCBCOFEND DS  0C        Make CB end on dword
OTCBCOPYONFORK_LEN EQU *-OTCBCOPYONFORK
*
*    ****************************************************************
*    *                                                              *
*    * **** This is end of "OtcbCopyOnFork" area ****               *
*    *                                                              *
*    ****************************************************************
*
*
*
*    ****************************************************************
*    *                                                              *
*    * The following based area is used by BPXPRGUG and BPXPRSGR. A  *
*    * copy of the last getgroupsbyname will be saved here if done   *
*    * after a __passwd() with the same name.
*    *                                                              *
*    ****************************************************************
*
*
OTCBRACGIDS DSECT
OTCBRACGIDSHEADER DS 0CL8
OTCBRACGIDSTOTNUM DS F
OTCBRACGIDSCURNUM DS F
OTCBRACGIDSLIST DS F
OTCB#ID  EQU   C'OTCB'   Control Block Acronym
OTCB#MEDCLEAR2LEN EQU 416 Length to clear
OTCB#LEN EQU   520
*
*                                     Length of OTCB
*
OTCB#ONLYLEN EQU 352     Length of OTCB only
OTCB#SP  EQU   230       Subpool for the OTCB
OTCBRACGIDS_LEN EQU *-OTCBRACGIDS
```

Appendix C. Mapping macros—AMODE 64

Mapping macros map the parameter options in many callable services. The fields with the comment "Reserved for IBM Use"are not programming interfaces. A complete list of the options for each macro is listed in the macro in “Macros mapping parameter options” on page 945.

Most of the mapping macros can be expanded with or without a *DSECT* statement. The invocation operand *DSECT=YES* (default) can be used with either reentrant or nonreentrant programs with the appropriate rules governing the storage backed by the *USING* statement.

Many of the mapping macros exploit the fact that *DC* expands as a *DS* in a *DSECT* and as a *DC* with its initialized value in a *CSECT*. When these fields are expanded as or within DSECTs, the program is responsible for initializing the necessary fields.

Macros mapping parameter options

Specifying DSECT=YES (the default for all macros) creates a DSECT. Addressability requires a USING and a register pointing to storage.

Specifying DSECT=NO (exceptions are listed when this is not allowed) allocates space in the current DSECT or CSECT. In reentrant programs, programmers can place these macros in the DSECT with DSECT=NO, and addressability is accomplished without the individual USING required by DSECT=YES. Nonreentrant programs can place their macros in the program's CSECT and addressability is obtained through the program base register(s).

Specifying LIST=YES (the default for most macros) causes the expansion of the macro to appear in the listing. You can override this by using PRINT OFF.

Specifying LIST=NO removes the macro expansion from the listing.

Additional keywords VARLEN and PREFIX are described in the individual sections where they apply.

BPXYAIO — Map asyncio parameter list

AMODE 31 callers use “BPXYAIO — Map asyncio parameter list” on page 946.

```
            SYSSTATE AMODE64=YES
            BPXYAIO    ,
* ------------------------------ 64-Bit Version
* ------------------------------
** BPXYAIO:  Asyncio parameter block
**  Used by: AIO
AIO                  DS    0D
AIOFD                DS    F      File Descriptor
AIOBUFFDW            DS    0CL8   Eight byte addresses
AIOBUFFALET          DS    F      Alet for AioBuffPtr
&AIOBUFFPTR31        DS    F      Buffer Pointer
AIOBUFFSIZE          DS    F      Buffer Length or Iov count
AIOOFFSETDW          DS    0CL8   Offset in File
```

```
AIOOFFSETH           DS    F      Offset in File highword
AIOOFFSET            DS    F      Offset in File lowword
AIOMSGEVENT          DS    0C     Message Event overlays SigEv
&AIOSIGEVENT31       DS    CL20   POSIX Signals
AIOREQPRIO           DS    F      REQUEST PRIORITY
AIOLIOOPCODE         DS    F      LIO_LISTIO() OP
*
                     ORG   AIOLIOOPCODE
AIOCMD               DS    F      Command Code
AIONOTIFYTYPE        DS    H      Notification Type
AIOCFLAGS            DS    XL1    Control Flags
AIOOK2COMPIMD        EQU   X'80'  Ok to complete immediately
AIOCALLB4            EQU   X'40'  Call exit before redrive
AIOSYNC              EQU   X'10'  Do synchronously
AIOEXITMODETCB       EQU   X'08'  0=SRB, 1=TCB
AIOCANCELNOWAIT      EQU   X'04'  Nowait option on cancel
AIOCANCELNONOTIFY    EQU   X'02'  NoNotify option on cancel
AIOTCBAFFINITY       EQU   X'01'  TCB Affinity I/O
AIOCFLAGS2           DS    XL1    Control Flags2
AIOUSERKEY           EQU   X'F0'  Caller's User's Key bit positions
AIOUSEUSERKEY        EQU   X'08'  Use User's Key for moves
AIOTHLICOMECB        EQU   X'04'  AioEcbPtr points tp ThliComEcb
AIOCOMMBUFF          EQU   X'02'  Common Area Buffer
AIOMSGIOVALET        DS    F      Alet for recvmsg/sendmsg IOV
AIOIOVBUFALET        DS    F      Alet for all IOV buffers
*
AIORV                DS    F      Return value
AIORC                DS    F      Return code
AIORSN               DS    F      Reason code
*
AIOPOSIXFLAGS        DS    XL4    Posix flags
&AIOEXITPTR31        DS    F      Pointer to user exit
AIOEXITDATA          DS    CL8    User Data for exit program
AIOECBPTR            DS    F      ECB address
AIOSOCKADDRLEN       DS    F      Sockaddr length
&AIOSOCKADDRPTR31    DS    F      Sockaddr pointer
AIOTIMEOUT           DS    F      TimeOut Value in Milli-seconds
AIOACEE              DS    F      SRB ACEE for MLS
AIOSICODE            DS    XL2    Signal si_code
AIORES06             DS    CL2    Reserved
AIOLEN               DS    F      (Output,debug) Len of AIO rcvd
AIOENDVER1           DS    0D     End of Original Aiocb
* -------- 64-Bit Extension
AIOLP64              DS    0D
&AIOBUFFPTR64        DS    AD     Buffer Ptr
&AIOEXITPTR64        DS    AD     Exit Program Address
&AIOSIGEVENT64       DS    CL32   SigEvent Structure
&AIOSOCKADDRPTR64    DS    AD     Sockaddr Ptr
                     DS    CL8
* -------- Version 3 Extension
AIOLOCSOCKADDRPTR    DS    AD     Local Sockaddr Ptr for ANR
AIOLOCSOCKADDRLEN    DS    F      Local Sockaddr Len for ANR
AIOANRSOCKET         DS    F      Accepted Socket for ANR
                     DS    CL48
AIOENDVER3           DS    0D     End of Version 3 extension
AIOEND               DS    0D     End of Aiocb
*
AIO#LENGTH           EQU   *-AIO  Length of this structure
*
** AIO command values
AIO#ACCEPT           EQU   126
AIO#CONNECT          EQU   128
AIO#READ             EQU   43
AIO#WRITE            EQU   54
AIO#READV            EQU   133
AIO#WRITEV           EQU   144
AIO#RECV             EQU   134
```

```
AIO#SEND             EQU   138
AIO#RECVFROM         EQU   135
AIO#SENDTO           EQU   140
AIO#RECVMSG          EQU   243
AIO#SENDMSG          EQU   244
AIO#ANR              EQU   264
Aio#BRLOCK           EQU   3
AIO#SELPOLL          EQU   2
AIO#CANCEL           EQU   1
*
** AIO notify type
AIO#POSIX            EQU   0
AIO#MVS              EQU   1
AIO#MSGQ             EQU   2
*
** AIO Message Event Structure
*        For AioNotifyType of AIO#MSGQ the AioMsgEvent
*          structure overlays AioSigEvent (31-bit location).
*        Msgbuf and Msgbuf64 are defined in BPXYMSG.
*        IPC_NOWAIT is defined in BPXYIPCP.
*
AIO_BEFORE_MSGEV     DS    0C             Note current position
                     ORG   AIOMSGEVENT
AIOMSGEV_QID         DS    F              Msg Queue Id
AIOMSGEV_SIZE        DS    H              Length of Msg_mtext
AIOMSGEV_FLAG        DS    H              0 or IPC_NOWAIT
AIOMSGEV_ADDR64      DS    D              Amode(64)-> MsgBuf64
                     ORG   AIOMSGEV_ADDR64
AIOMSGEV_ADDRH       DS    F
AIOMSGEV_ADDR        DS    F              Amode(31)-> MsgBuf
                     ORG   AIO_BEFORE_MSGEV    Return to above
*
AIO#MSGTEXTMAX       EQU  240             Max Msg_MText
*
** AIO Signal Event
SIGEVENT        DSECT ,
SIGEVENT         DS 0F
SIGEV_NOTIFY         DS    F              NOTIFICATION TYPE
SIGEV_SIGNO          DS    F              SIGNAL NUMBER
SIGEV_VALUE          DS    &AIOPTRSIZE    SIG VALUE
                     ORG   SIGEV_VALUE
SIVAL_INT            DS    F
                     ORG   SIGEV_VALUE
SIVAL_PTR            DS    &AIOPTRSIZE
SIGEV_NOTIFY_FUNCTION   DS &AIOPTRSIZE    NOTIF. FUNCTION
SIGEV_NOTIFY_ATTRIBUTES DS &AIOPTRSIZE    NOTIF. ATTRIBUTES
*
SIGEV#LENGTH         EQU   *-SIGEVENT  Length of this structure
*
*    SIGEV_NOTIFY Values
SIGEV_SIGNAL         EQU   0              GENERATE A SIGNAL
SIGEV_NONE           EQU   1              DON'T GENERATE SIGNAL
SIGEV_THREAD         EQU   2              Call Notif. function
*
** AIOTIMEOUT VALUES
AIO#FOREVER          EQU   0              NO TIMEOUT, JUST WAIT
AIO#NOWAITING        EQU   X'FFFFFFFF'    NO WAITING, JUST CHECK
** AIO CANCEL RETURN VALUES
AIO_CANCELED         EQU   1              ALL CANCELS SUCCESSFUL
AIO_NOTCANCELED      EQU   2              AT LEAST 1 CANCEL FAILED
AIO_ALLDONE          EQU   3              NONE CANCELED, ALL COMP
*
** BPXYAIO End
```

BPXYCCA — Map input/output structure for __console()

AMODE 31 callers use "BPXYCCA — Map input/output structure for __console()" on page 950.

```
            SYSSTATE AMODE64=YES
            BPXYCCA    ,
**  BPXYCCA:  Msg Attributes for console_np service
**  Used By:  CCS
CCA                 DSECT ,
CCABEGIN            DS    0D
*
CCAVERSION          DC    AL2(CCA#VER)
*                         Version of this structure
CCARES01            DS    CL2    Reserved
CCAMSGLENGTH        DS    F      Length of msg pointed to by CCAMSGPTR
CCAMSGPTR           DS    AD     Pointer to Msg text
CCARES02            DS    CL4    Reserved
CCAENDVER1          DS    0F     End of Version 1
CCASTARTVER2        DS    0F     Start of Version 2
CCARES03            DS    F      Reserved
CCAWTOPARMS         DS    0F     Start of WTO message attributes
CCAROUTCDELIST      DS    AD     Pointer to list of message routing   X
                                 codes
CCADESCLIST         DS    AD     Pointer to list of message           X
                                 descriptor codes
CCAWMCSFLAGS        DS    0F     WTO MCS Flags
CCAMCSFLAGB1        DS    XL1    MCS flags byte 1
CCAHRDCPY           EQU   X'80'  Send message to hard copy log only
CCAMCSFLAGB2        DS    XL1    MCS flags byte 2
CCAMCSFLAGB3        DS    XL1    MCS flags byte 3
CCAMCSFLAGB4        DS    XL1    MCS flags byte 4
CCAWTOTOKEN         DS    F      Token for message to be issued
CCAMSGIDPTR         DS    AD     Pointer to location where message    X
                                 is is stored by BPX1CCS
CCARES07            DS    F      Reserved
CCADOMPARMS         DS    0F     Delete message parameters
CCADOMTOKEN         DS    F      Token of message(s) to be deleted
CCAMSGIDLIST        DS    AD     Pointer to list of message ids to    X
                                 be deleted
CCAENDVER2          DS    0D     End of version 2
CCASTARTVER3        DS    0CL40  Start of version 3
CCAMODCARTPTRG      DS    AD     Pointer 8 byte CART returned for     X
                                 MODIFY/STOP command
CCAMODCONSOLEIDPTRG DS    AD     Pointer to 4 byte ConsoleID returned X
                                 for MODIFY/STOP command
CCAMSGCART          DS    CL8    Supplied - CART to be specified on   X
                                 WTO when message is issued
CCAMSGCONSOLEID     DS    CL4    Supplied - ConsoleID to be specified X
                                 on WTO when message is issued
CCARES08            DS    CL12   Reserved
CCAENDVER3          DS    OD     End of Version 3
*
*   Constants
*
CCA#VER             EQU   CCA#VER02 Current version
CCA#VER01           EQU   1        Version 1 of this structure
CCA#VER02           EQU   2        Version 2 of this structure
CCA#VER03           EQU   3        Version 3 of this structure
CCA#LENGTH          EQU   *-CCABEGIN                                  X
                                 Length of CCA
CCA#VER1LEN         EQU   CCAENDVER1-CCABEGIN                         X
                                 Length of Version 1 CCA
CCA#VER2LEN         EQU   CCAENDVER2-CCABEGIN                         X
```

```
                                  Length of Version 2 CCA
CCA#VER3LEN          EQU   CCAENDVER3-CCABEGIN                        X
                                  Length of Version 3 CCA
** BPXYCCA  End
```

BPXYDCOR — dbx cordump cache information

BPXYDCOR contains the mapping of dump related information used by dbx when a dump is being formatted. AMODE 31 callers use "BPXYDCOR — dbx cordump cache information" on page 959.

```
             BPXYDCOR   PARMSG=YES
*
* ********************************************************************
* *                                                                  *
* * Level information                                               *
* *                                                                  *
* ********************************************************************
*
*
DCOR_LEVEL1 EQU 65536    65536='00010000'x.
DCOR_LEVEL2 EQU 131072   131072='00020000'x.
*
* ********************************************************************
* *                                                                  *
* * Function codes for BPXGMCDE routine                             *
* *                                                                  *
* ********************************************************************
*
*
DCOR_OPEN# EQU 1
DCOR_CLOSE# EQU 2
DCOR_STATUS# EQU 3
*
*    ******************************************************************
*    *                                                                *
*    * Open return codes                                             *
*    *                                                                *
*    ******************************************************************
*
*
DCOR_CDERC_OK EQU 0      The specified function completed successfully
DCOR_CDERC_PARMERR EQU 4 A parmeter error was detected. See return     X
                         value 1 for more detail
DCOR_CDERC_PROCERR EQU 8 A DCORE processing error occurred. See return X
                         value 1 for more detail
DCOR_CDERC_IKJTSOEVERR EQU 12 An error was encountered trying to       X
                         establish a TSO environment with the IKJTSOEV X
                         service. See return values for more           X
                         information
DCOR_CDERC_IKJEFTSRERR EQU 16 An error was encountered trying to run   X
                         the REXX EXEC with the IKJEFTSR service. See  X
                         return values for more information
DCOR_CDERC_ALLOCATEERR EQU 20 An error was encountered trying to       X
                         allocate one of the user specified data sets.
DCOR_CDERC_IRXINITERR EQU 28 An error was encountered trying to        X
                         establish a REXX environment
*
*    ******************************************************************
*    *                                                                *
*    * Status return codes                                           *
*    *                                                                *
*    ******************************************************************
*
*
```

```
DCOR_CDERC_STATUS_OPENCOMPLETE EQU 0
DCOR_CDERC_STATUS_OPENCONTINUING EQU 1
DCOR_CDERC_STATUS_OPENTERMINATED EQU 2
DCOR_CDERC_STATUS_INVALIDTOKEN EQU 3
*
*    ****************************************************************
*    *                                                              *
*    * Status Rc values when Status return code is                  *
*    * Dcor_CDErc_Status_OpenContinuing                            *
*    *                                                              *
*    ****************************************************************
*
*
DCOR_STATUS_CONT_STARTTSOENV EQU 0 Starting the TSO environment
DCOR_STATUS_CONT_EXECSTARTED EQU 1 BPXTIPCS started
DCOR_STATUS_CONT_EXECCLIST EQU 2 BPXTIPCS allocating CLIST data set
DCOR_STATUS_CONT_DUMPDDIR EQU 3 BPXTIPCS allocating/creating dump      X
                        directory via BLSCDDIR
DCOR_STATUS_CONT_ALLOCDUMPDS EQU 4 BPXTIPCS allocating the dump data   X
                        set
DCOR_STATUS_CONT_INVOKEIPCS EQU 5 BPXTIPCS invoking IPCS
DCOR_STATUS_CONT_INVOKEVERBX EQU 6 BPXTIPC2 invoking VERBX routine
DCOR_STATUS_CONT_ANALYSISSTART EQU 7 Dump analysis started
DCOR_STATUS_CONT_ANALYSISPROCASIDS EQU 8 Analysis processing Asids
DCOR_STATUS_CONT_EXECEXITING EQU 9 BPXTIPCS exiting
DCOR_STATUS_CONT_RECALL EQU 10 BPXTIPCS recalling data set
*
*    ****************************************************************
*    *                                                              *
*    * R1 values when return code is Dcor_CDErc_ParmErr             *
*    *                                                              *
*    ****************************************************************
*
*
DCOR_R1_PARMERR_DUMPDSNREQ EQU 1 The name of a dump data set is        X
                        required
DCOR_R1_PARMERR_HFSDSNREQ EQU 2 The name of a dump data set in the HFS X
                        could not be found
*
*      ****************************************************************
*      *                                                              *
*      * R1 values when return code is Dcor_CDErc_ProcErr            *
*      *                                                              *
*      ****************************************************************
*
*
DCOR_R1_PROCERR_SYSTEMERRATC EQU 1 An unexpected system error has      X
                        occured while trying to establish the IPCS    X
                        environment. The R2 value contains an ABEND   X
                        reason code
*
*      ****************************************************************
*      *                                                              *
*      * R1 values when return code is Dcor_CDErc_AllocateErr         *
*      *                                                              *
*      ****************************************************************
*
*
DCOR_R1_ALLOCATEERR_LOGDSN EQU 1 Error allocating the log data set.    X
                        The R2 field is the return code from          X
                        allocation and the R3 field is the reason     X
                        code.
DCOR_R1_ALLOCATEERR_EXECDSN EQU 2 Error allocating the EXEC data set.  X
                        The R2 field is the return code from          X
                        allocation and the R3 field is the reason     X
                        code.
*
```

```
* ********************************************************************
* *                                                                  *
* * Function codes for BPXGMPTR Ptrace Dump Access Routine           *
* *                                                                  *
* ********************************************************************
*
*
DCOR_ASID_LIST# EQU 1
DCOR_SET_ASID# EQU 2
DCOR_PID_LIST# EQU 3
DCOR_SET_PID# EQU 4
DCOR_LDINFO# EQU 5
DCOR_THREAD_LIST# EQU 6
DCOR_THREAD_CURRENT# EQU 7
DCOR_SET_THREAD# EQU 8
DCOR_PSW# EQU  9
DCOR_GPR_LIST# EQU 10
DCOR_THREAD_STATUS# EQU 11
DCOR_READ_D# EQU 12
DCOR_ERROR_PSW# EQU 13
DCOR_CAPTURE# EQU 14
DCOR_ERROR_GPR_LIST# EQU 15
DCOR_FLT_LIST# EQU 16
DCOR_ERROR_FLT_LIST# EQU 17
DCOR_CONDINFO# EQU 18
DCOR_IPCSCMD# EQU 19
DCOR_PTRRC_OKVALUE EQU 0 The specified function completed successfully
DCOR_PTRRC_ASIDNOTFOUND EQU 1 The requested asid(s) not in dump
DCOR_PTRRC_ASIDNOTSET EQU 2 An ASID or PID has not been established    X
                        for this session
DCOR_PTRRC_REQTYPENOTDEFINED EQU 3 The function type provided on this  X
                        request is not supported by BPXGMPT2
DCOR_PTRRC_REQINVALIDTOKEN EQU 4 The open token provided on this       X
                        request is not not valid
DCOR_PTRRC_REQDCORTERMINATED EQU 5 Dcor dump access services are not   X
                        available
DCOR_PTRRC_THREADNOTFOUND EQU 6 The request thread(s) were not in the  X
                        dump
DCOR_PTRRC_THREADNOTSET EQU 7 The current thread has not been          X
                        established
DCOR_PTRRC_PIDNOTSET EQU 9 The request PID(s) were not in the dump
DCOR_PTRRC_PIDNOTFOUND EQU 10 The current process has not been         X
                        established
DCOR_PTRRC_STORAGENOTINDUMP EQU 11 The requested storage was not       X
                        dumped
DCOR_PTRRC_NASTANDALONEDUMP EQU 12 Not supported in a standalone dump
DCOR_PTRRC_ABENDOCCURRED EQU 13 Not supported in a standalone dump
DCOR_PTRRC_STORAGELENGTHBAD EQU 14 The requested storage length was    X
                        zero
DCOR_PTRRC_SOMESTORAGEINDUMP EQU 15 The number of bytes of storage     X
                        successfully retrieved is returned in the     X
                        reason code field
RSNOKVALUE EQU 0
RSNDCORERROR EQU 1       See Dcor return codes
RSNMVSERROR EQU 2        Ususally an out of storage condition or an    X
                        abend
RSNIPCSERROR EQU 3       When An IPCS error occurs use the DCOR log to X
                        view the messages generated by IPCS (normally X
                        suppress)
RSNCSVERROR EQU 4
RSNCSVMODI12ERR EQU 1
RSNCSVMODI3ERR EQU 2
RSNCSVTOOMANYEXTENTS EQU 3
*
* ********************************************************************
* *   parameter definitions for BPXGMPTR Ptrace Dump Access Routine  *
* *   1. Parm     1   function code                                  *
```

```
* *   2. Parms    2   Token returned from DCOR_OPEN#                *
* *   3. Parms    3-5 Function parameters                           *
* *   3. Parms    6-8 retvalue, retcode, rsncode                    *
* ********************************************************************
*
*
PARMS    DSECT
PARMS_FUNCTYPEPTR DS 1AL4
PARMS_DCOMTOKENPTR DS 1AL4
PARMS_INTERFACE DS 0CL0012
         ORG   PARMS_INTERFACE
PARMS_CAPTURE DS 0CL0012
PARMS_CAPTURE_PSTORADR DS 1AL4
PARMS_CAPTURE_PSTORLEN DS 1AL4
PARMS_CAPTURE_PDATAADR DS 1AL4 Address output buffer
         ORG   PARMS_INTERFACE
PARMS_READD DS 0CL0012
PARMS_READD_PSTORADR DS 1AL4
PARMS_READD_PSTORLEN DS 1AL4
PARMS_READD_PDATAADR DS 1AL4 user provided buffer
         ORG   PARMS_INTERFACE
PARMS_LDINFO DS 0CL0004
PARMS_LDINFO_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADLIST DS 0CL0008
PARMS_THREADLIST_OUTBUFPTR DS 1AL4
PARMS_THREADLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_PIDLIST DS 0CL0008
PARMS_PIDLIST_OUTBUFPTR DS 1AL4
PARMS_PIDLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ASIDLIST DS 0CL0008
PARMS_ASIDLIST_OUTBUFPTR DS 1AL4
PARMS_ASIDLIST_OUTBUFCNT DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADCURRENT DS 0CL0004
PARMS_THREADCURRENT_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETASID DS 0CL0004
PARMS_SETASID_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETPID DS 0CL0004
PARMS_SETPID_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_SETTHREAD DS 0CL0004
PARMS_SETTHREAD_INBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_PSW DS   0CL0004
PARMS_PSW_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_PSW DS 0CL0004
PARMS_ERROR_PSW_OUTBUFPTR DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_THREADSTATUS DS 0CL0008
PARMS_THREADSTATUS_OUTBUFPTR DS 1AL4
PARMS_THREADSTATUS_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_GPRLIST DS 0CL0008
PARMS_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_GPRLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_GPRLIST DS 0CL0008
PARMS_ERROR_GPRLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_GPRLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_FLTLIST DS 0CL0008
```

```
PARMS_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_FLTLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_ERROR_FLTLIST DS 0CL0008
PARMS_ERROR_FLTLIST_OUTBUFPTR DS 1AL4
PARMS_ERROR_FLTLIST_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_CONDITIONINFO DS 0CL0008
PARMS_CONDITIONINFO_OUTBUFPTR DS 1AL4
PARMS_CONDITIONINFO_OUTBUFLEN DS 1AL4
         ORG   PARMS_INTERFACE
PARMS_IPCSCMD DS 0CL0012
PARMS_IPCSCMDTEXT_INBUFPTR DS 1AL4
PARMS_IPCSCMDTEXT_INBUFLEN DS 1AL4
PARMS_IPCSCMDPRNT_LRECL DS 1AL4
PARMS_XRVPTR DS 1AL4     Return Value
PARMS_XRCPTR DS 1AL4     Return Code
PARMS_XRSNPTR DS 1AL4    Reason Code
PARMS_LEN EQU  *-PARMS
PARMSG   DSECT
PARMS_FUNCTYPEPTRG DS 1AD
PARMS_DCOMTOKENPTRG DS 1AD
PARMS_INTERFACEG DS 0CL0024
         ORG   PARMS_INTERFACEG
PARMS_CAPTUREG DS 0CL0024
PARMS_CAPTURE_PSTORADRG DS 1AD
PARMS_CAPTURE_PSTORLENG DS 1AD
PARMS_CAPTURE_PDATAADRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_READDG DS 0CL0024
PARMS_READD_PSTORADRG DS 1AD
PARMS_READD_PSTORLENG DS 1AD
PARMS_READD_PDATAADRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_LDINFOG DS 0CL0008
PARMS_LDINFO_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_THREADLISTG DS 0CL0016
PARMS_THREADLIST_OUTBUFPTRG DS 1AD
PARMS_THREADLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_PIDLISTG DS 0CL0016
PARMS_PIDLIST_OUTBUFPTRG DS 1AD
PARMS_PIDLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ASIDLISTG DS 0CL0016
PARMS_ASIDLIST_OUTBUFPTRG DS 1AD
PARMS_ASIDLIST_OUTBUFCNTG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_THREADCURRENTG DS 0CL0008
PARMS_THREADCURRENT_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETASIDG DS 0CL0008
PARMS_SETASID_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETPIDG DS 0CL0008
PARMS_SETPID_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_SETTHREADG DS 0CL0008
PARMS_SETTHREAD_INBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_PSWG DS  0CL0008
PARMS_PSW_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_PSWG DS 0CL0008
PARMS_ERROR_PSW_OUTBUFPTRG DS 1AD
         ORG   PARMS_INTERFACEG
```

```
PARMS_THREADSTATUSG DS 0CL0016
PARMS_THREADSTATUS_OUTBUFPTRG DS 1AD
PARMS_THREADSTATUS_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_GPRLISTG DS 0CL0016
PARMS_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_GPRLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_GPRLISTG DS 0CL0016
PARMS_ERROR_GPRLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_GPRLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_FLTLISTG DS 0CL0016
PARMS_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_FLTLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_ERROR_FLTLISTG DS 0CL0016
PARMS_ERROR_FLTLIST_OUTBUFPTRG DS 1AD
PARMS_ERROR_FLTLIST_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_CONDITIONINFOG DS 0CL0016
PARMS_CONDITIONINFO_OUTBUFPTRG DS 1AD
PARMS_CONDITIONINFO_OUTBUFLENG DS 1AD
         ORG   PARMS_INTERFACEG
PARMS_IPCSCMDG DS 0CL0024
PARMS_IPCSCMDTEXT_INBUFPTRG DS 1AD
PARMS_IPCSCMDTEXT_INBUFLENG DS 1AD
PARMS_IPCSCMDPRNT_LRECLG DS 1AD
PARMS_XRVPTRG DS 1AD
PARMS_XRCPTRG DS 1AD
PARMS_XRSNPTRG DS 1AD
PARMSG_LEN EQU *-PARMSG
ASIDLIST_MAP DSECT
ASID_NEXTOFF DS 1FL4     Offset to the next ASID in DcomAsidList
ASID_NUM DS    1FL2
ASID_CPU DS    1FL1      CPUID
ASID_FLAGS DS  0BL1      Status flags
ASID_HOME EQU  X'80'     Current HOMEAsid
ASID_PRIM EQU  X'40'     Current PRIMARY ASID
ASID_SEC EQU   X'20'     Current SECONDARY ASID
         ORG   ASID_FLAGS+X'00000001'
ASID_JOBNAME DS 1CL0009
         DS    1CL0003   Reserved
ASID_PIDCNT DS 1FL4      Number of Pids in this Asid
ASID_ASCB DS   1AL4      Pointer to ASCB
         DS    1CL0004   Reserved
ASID_PIDLISTPTR DS 1AL4  Pointer to the pidlist for This Asid
         DS    1CL0004   Reserved
ASID_MAPEND DS 0C        end of block
ASIDLIST_MAP_LEN EQU *-ASIDLIST_MAP
PIDLIST_MAP DSECT
PID_NEXTOFF DS 1FL4      Offset to the next Pid in DcomPidList
PID_     DS    1FL4      Process id
PID_ASID DS    1FL2      Asid of this Pid
PID_THIDCNT DS 1FL2      Count of thids in this pid
PID_FOCUSTHREAD DS 1CL0008 Ptrace focus thread
PID_ERRORTHREAD DS 1CL0008 Ptrace error thread
PID_LOGINNAME DS 1CL0009 Tso logon
         DS    1CL0003   Reserved
PID_THIDLISTPTR DS 1AL4  list info for each THID
         DS    1CL0004   Reserved
PID_PENDINGSIGMASK DS 1BL8 Signals pending at the process that could   X
                         not be delivered to any thread
PID_BLOCKEDSIGMASK DS 1BL8 Signals blocked on all thread
PID_MAPEND DS  0C        end of block
PIDLIST_MAP_LEN EQU *-PIDLIST_MAP
CONDINFO_MAP DSECT
```

```
COND_CURABENDINFO DS 0CL0016 If current task abended
COND_CURINTCODE DS 1FL2  Interrupt code
COND_CURSIGNUMBER DS 1FL2 Signal number raised
COND_CURABENDCODE DS 0BL4 Abend code
COND_CURABENDFLAGS DS 1BL1 System or user
COND_CURABENDCC DS 1BL3  Abend Number
COND_CURABENDREASON DS 1BL4 Abend Reason
COND_CURILC DS 1FL2      Instruction length
         DS    1CL0002   Reserved
CONDINFO_MAPEND DS 0C    end of block
CONDINFO_MAP_LEN EQU *-CONDINFO_MAP
```

BPXYINHE — Spawn inheritance structure

AMODE 31 callers use "BPXYINHE — Spawn Inheritance Structure" on page 970.

```
           SYSSTATE AMODE64=YES
           BPXYINHE   ,
** BPXYINHE: Inheritance Area
**  Used By: spawn() callable service
INHE                DSECT ,
INHEBEGIN           DS    0D
*
INHEEYE             DC    C'INHE'  Eye catcher
INHELENGTH          DC    AL2(INHE#LENGTH)                          X
                                Length of this structure
INHEVERSION         DC    AL2(INHE#VER)
INHE#VER            EQU   3     Version of this structure
INHEFLAGS           DS    0BL4  Flags indicating contents of structure
INHEFLAGS0          DS    XL1   1st byte
INHESETPGROUP       EQU   X'80' Set Process Group using INHEPGROUP
INHESETSIGMASK      EQU   X'40' Set Signal Mask using INHESIGMASK
INHESETSIGDEF       EQU   X'20' Set Signal Defaults using INHESIGDEF
INHESETTCPGRP       EQU   X'10' Set TTY Pgrp using INHECTLTTYFD
INHESETCWD          EQU   X'08' Set CWD using INHECWDPTR
INHESETUMASK        EQU   X'04' Set UMASK using INHEUMASK
INHESETUSERID       EQU   X'02' Set Userid using INHEUSERID
INHESETREGIONSZ     EQU   X'01' Set Region using INHEREGIONSZ
INHEFLAGS1          DS    XL1   2nd byte
INHESETTIMELIMIT    EQU   X'80' Set Timelimit with INHETIMELIMIT
INHESETACCTDATA     EQU   X'40' SET ACCTDATA using INHEACCTDATA
INHESETJOBNAME      EQU   X'20' SET JOBNAME using INHEJOBNAME
INHEMUSTBELOCAL     EQU   X'10' Spawn locally or else fail
INHESETDEBUGENV     EQU   X'08' Setup Debug Environment
INHESETMEMLIMIT     EQU   X'04' Set MemLimit with INHEMEMLIMIT
INHEFLAGS2          DS    XL1   3rd byte
INHEFLAGS3          DS    XL1   4th byte
INHEPGROUP          DS    F     Process Group for child
INHE#NEWPGROUP      EQU   0     Put child in a new proc grp of its own
INHESIGMASK         DS    BL8   Signal Mask for child
INHESIGDEF          DS    BL8   Set of default signals for child
INHECTLTTYFD        DS    F     Cntl TTY FD for tcsetgrp() in child
                    DS    0F    64-Bit Addressing Version
INHERES01           DS    F     Reserved
INHECWDLEN          DS    H     Length of the users CWD
INHEACCTDATALEN     DS    H     LENGTH OF THE USERS ACCTDATA
INHERES02           DS    F     Reserved
INHEUMASK           DS    XL4   Users Umask
INHEUSERID          DS    CL8   New A.S. user identity
INHEJOBNAME         DS    CL8   New A.S. jobname
INHEREGIONSZ        DS    F     New A.S. region size
INHETIMELIMIT       DS    F     New A.S. Time limit
INHERES03           DS    F     Reserved
INHECWDPTR          DS    AD    Pointer to the users CWD
```

```
INHEACCTDATAPTR      DS    AD    Pointer to the users ACCTDATA
INHEMEMLIMIT         DS    D     New A.S. Memlimit #bytes
INHE#LENGTH          EQU   *-INHE
** BPXYINHE End
```

BPXYIOC6 — Map IPV6 prerouter structures

BPXYIOC6 is used by transport providers. DSECT= is allowed but ignored. AMODE 31 callers use "BPXYIOC6 — Map IPV6 prerouter structures" on page 982.

```
         SYSSTATE AMODE64=YES
         BPXYIOC6   ,
NETCONFHDR           DSECT ,
* ------------ 64-Bit Version                                    @P2A
NCHEYECATCHER        DS    CL4    Eye catcher                     @P2A
NCHIOCTL             DS    F      Ioctl being processed (RAS)     @P2A
NCHNUMENTRYRET       DS    F      Number of HomeIF returned via   @P2A*
                                       SIOCGHOMEIF6 or number of  @P2A*
                                       GRT6RtEntry's returned via @P2A*
                                       SIOCGRT6TABLE.             @P2A
NCHBUFFERLENGTH      DS    F      Buffer Length                   @P2A
NCHBUFFERPTR         DS    D      64-bit Buffer Pointer           @P2A
NETCONFHDR#LENGTH    EQU   *-NETCONFHDR   Length of NETCONFHDR
*
*********************************************************************
* HomeIf Structure                                                  *
*********************************************************************
*
HOMEIF               DSECT ,      HomeIf structure
HomeIfAddress        DS    CL16   Home Interface Address
*
HomeIf#LENGTH        EQU   *-HOMEIF   Length of HOMEIF
*
*********************************************************************
* GRT6RtEntry Structure                                             *
*********************************************************************
*
GRT6RTENTRY          DSECT ,      GRT6RtEntry Structure
*
GRT6DESTINATION      DS    CL16 Destination IP Address
GRT6GATEWAY          DS    CL16 First HOP on the trip if going through *
                                  a gateway
GRT6DESTPREFIXLEN    DS    F    Destination's Prefix Length which is a *
                                  decimal value that specifies how many *
                                  of the leftmost contiguous bits of the*
                                  address comprise the prefix
GRT6RTMETRIC         DS    F    Metric - hop count. Currently Tcp/Ip   *
                                  returns 1 for indirect routes and 0   *
                                  for direct routes. If route is from   *
                                  routing daemon, metric is whatever    *
                                  routing daemon set it to.
GRT6RTFLAGS          DS    F    IPV6 Route Flags.
*
GRT6RTENTRY#LENGTH   EQU   *-GRT6RTENTRY   Length of GRT6RTENTRY
*
*********************************************************************
* RT6Entry Structure                                                *
*********************************************************************
*
RT6ENTRY          DSECT ,         Rt6Entry Structure
*
RT6DESTINATION       DS    CL28   Destination IP address (in an IPV6   *
                                    sockaddr structure)
RT6GATEWAY           DS    CL28   First HOP on the trip if going       *
                                    through a gateway (in an IPV6      *
```

```
                                  sockaddr structure)
RT6DESTPREFIXLEN     DS    F      Destination's Prefix Length,         *
                                    which is a decimal value         *
                                    that specifies how many of       *
                                    the leftmost contiguous          *
                                    bits of the address              *
                                    comprise the prefix.
RT6METRIC            DS    F      Metric - hop count                   *
                                   Currently Tcp/IP returns          *
                                    1 for indirect route and         *
                                    0 for direct route.              *
                                   If route is from routing          *
                                   daemon, metric is whatever        *
                                   routing daemon set it to.
RT6FLAGS             DS    F      IPV6 Route Flags.
*
RT6ENTRY#LENGTH      EQU   *-RT6ENTRY   Length of RT6ENTRY
*
*********************************************************************
* GRT6RtEntryV2 Structure                                           *
*********************************************************************
*
GRT6RTENTRYV2        DSECT ,      New Route Entry used with DCR A846 - *
                                  Route Modification
*
GRT6OLDRTENTRY       DS    CL44   Old GRT6 Route Entry
GRT6RTHOMEIFIDX      DS    F      Route's Home Interface Idx
GRT6RTIFINDEX        DS    F      Route's Interface Index
GRT6RTMTU            DS    H      Route's MTU Value
*                    DS    H      Reserved
*                    DS    F      Reserved
*                    DS    F      Reserved
*
GRT6RTENTRYV2#LENGTH   EQU   *-GRT6RTENTRYV2   Length of GRT6RTENTRYV2
*
*********************************************************************
* RT6EntryV2 Structure                                              *
*********************************************************************
*
RT6ENTRYV2           DSECT ,      New Route Entry Used with A846       *
                                  MSADDRT6V2/MSDELRT6V2 IOCTLs
*
RT6OLDENTRY          DS    CL68   Old Route Entry used before A846     *
                                  with SIOCMSADDRT6/SIOCMSDELRT6 IOCTL
RT6RTHOMEIFIDX       DS    F      Route's Home Interface Idx
*                    DS    F      Reserved
*                    DS    F      Reserved
*                    DS    F      Reserved
*                    DS    F      Reserved
*
RT6ENTRYV2#LENGTH    EQU   *-RT6ENTRYV2    Length of RT6ENTRYV2
*
*********************************************************************
* IPV6RtFlags Structure                                             *
*********************************************************************
*
IPV6RTFLAGS       DSECT ,         IPV6RtFlags Structure
*
IPV6FLGROUTETYPE     DS    XL1   Route Type                    @D1C
IPV6FLGBYTE2         DS    XL1    Reserved
IPV6FLGBYTE3         DS    XL1    Reserved
IPV6FLGBYTE4         DS    XL1    FLAGS:
*                    EQU   X'80'  Reserved
*                    EQU   X'40'  Reserved
*                    EQU   X'20'  Reserved
IPV6BITLOOPBACK      EQU   X'10'  1 = Loopback Interface
IPV6BITHOME          EQU   X'08'  1 = Home interface
```

```
IPV6BITHOST          EQU   X'04'  1 = Host Route. 0 = Network Route
IPV6BITGATE          EQU   X'02'  1 = Gateway
IPV6BITRTUP          EQU   X'01'  1 = Route is active
*
* *-------------------------------------------------------------------*
* * SiocGifConf6 - Get IPv6 Interface Configuration.            @D3A*
* *                                                                   *
* *  Net_IfConf6Header is passed as the argument of the ioctl and    *
* *  is returned with the number of entries and entry length of the  *
* *  Net_IfConf6Entry structs that were written to the output buffer.*
* *                                                                   *
* *  If Buflen=0=Buffer a Query function is performed and the        *
* *  header is returned with: (1) the maximum supported version,     *
* *  (2) the total number of entries that would be output and        *
* *  (3) the length of each individual entry.                        *
* *                                                                   *
* *  If a call to get information fails with RC=ERANGE or with       *
* *  (RC=EINVAL & Nif6h_Version is changed) the call is converted    *
* *  into a Query function and the content of the output buffer      *
* *  is unpredictable.                                               *
* *                                                                   *
* *  For information on the data returned in this structure refer    *
* *  to the z/OS Communication Server's IP Configuration Guide and   *
* *  IPv6 Network and Application Design Guide.                      *
* *                                                                   *
* *-------------------------------------------------------------------*
NET_IFCONF6HEADER DSECT   Header                                  @D3A
NIF6H_VERSION DS F       Input for Get IfConf6 Output for Query
NIF6H_ENTRIES DS F       Output: number of entries returned in output  *
                         buffer
NIF6H_ENTRYLEN DS F      Output: length of an entry
NIF6H_BUFLEN DS F        Input: length of buffer
NIF6H_BUFFER64 DS 0CL8   Input: Amode(64) Buffer ptr
NIF6H_BUFFER64H DS F
NIF6H_BUFFER DS A        Input: Amode(31) Buffer ptr to output buffer  *
                         that will be filled with an array of          *
                         Net_IfConf6Entrys.
NET_IFCONF6HEADER_LEN EQU *-NET_IFCONF6HEADER
*
NET_IFCONF6ENTRY DSECT   Entry                                    @D3A
NIF6E_NAME DS  CL16      x00 interface name (blank padded - no null)
NIF6E_STACKNAME DS CL8   x10 tcpip stack name (blank padded - no null)
NIF6E_ADDR DS  CL28      x18 Sock_Inet6_SockAddr of the interface
NIF6E_ROUTEMETRIC DS F   x34 route metric
NIF6E_PREFIXLEN DS H     x38 routing prefix length
NIF6E_PREFIXORIGIN DS X  x3A prefix origin, see below
NIF6E_STATUS DS X        x3B status, see below
NIF6E_FLAGS DS 0BL4      x3C Flags:
NIF6E_FLAGS1 DS B
NIF6E_FLAGS2 DS B
NIF6E_FLAGS3 DS 0B
NIF6E_VIRTUAL EQU X'40'
NIF6E_MULTIPOINT EQU X'08'
NIF6E_MULTICASTCAPABLE EQU X'04'
         ORG   NIF6E_FLAGS3+1
NIF6E_FLAGS4 DS 0B
NIF6E_POINT2POINT EQU X'10'
NIF6E_LOOPBACK EQU X'08'
NIF6E_ONLINK EQU X'01'
         ORG   NIF6E_FLAGS+4
NIF6E_MTU DS   F         x40 mtu
*
*    ******************************************************************
*    *                                                                *
*    * Constants for nif6h_version                                 @D3A
*    *                                                                *
*    ******************************************************************
```

```
*
*
NIF6H#VER EQU  1         Current Version
NIF6H#VER1 EQU 1         Initial Version
*
*    ********************************************************************
*    *                                                                  *
*    * Constants for nif6e_prefixorigin                           @D3A  *
*    *                                                                  *
*    ********************************************************************
*
*
NIF6H#WELLKNOWN EQU 1
NIF6H#MANUAL EQU 2
NIF6H#RTRADV EQU 3
NIF6H#OTHER EQU 8
*
*    ********************************************************************
*    *                                                                  *
*    * Constants for nif6e_status                                 @D3A  *
*    *                                                                  *
*    ********************************************************************
*
*
NIF6H#PREFERRED EQU 1
NIF6H#DEPRECATED EQU 2
NIF6H#INVALID EQU 3
NIF6H#INACCESSIBLE EQU 4
NIF6H#UNKNOWN EQU 5
NIF6H#TENTATIVE EQU 6
NIF6H#DUPLICATE EQU 7
NET_IFCONF6ENTRY_LEN EQU *-NET_IFCONF6ENTRY
*                                    End SiocGifConf6 -------- @D3A
*
* ********************************************************************
* *                                                                  *
* *   Constants                                                      *
* *                                                                  *
* ********************************************************************
*
*
IOC6_#HOMEIFPREFIXLEN EQU 128 The prefix length for a home interface   *
                         address returned on the SIOCGHOMEIF6 IOCTL.
IOC6_NCH#EYE EQU C'6NCH' IPV6 Network Configuration Header EyeCatcher.
IOC6_NCH64#EYE EQU C'6N64' IPV6 NetConfHdr EyeCatcher 64-BIT
*
* ********************************************************************
* *                                                                  *
* * Maximum hop count for the Metric fields:                         *
* *    GRT6RtMetric                                                  *
* *    Rt6Metric                                                     *
* *                                                                  *
* ********************************************************************
*
*
IOC6_#MAXHOPMETRIC EQU 16
*
* ********************************************************************
* *                                                                  *
* *   Constants used for size of control areas                       *
* *                                                                  *
* ********************************************************************
*
*
IOC6_#MAXROUTES EQU 600
IOC6_#GRT6ROUTELEN EQU 44
*
```

```
*    ****************************************************************
*     Initial buffer size for SIOCGHOMEIF6 and SIOCGRT6TABLE.
*    ****************************************************************
*
*
IOC6_#MAXGRT6LEN EQU 26400
IOC6_#NETCONFHDRLEN EQU 20
IOC6_#GRT6V2ROUTELEN EQU 64
IOC6_#MAXGRT6V2LEN EQU 38400
*
** BPXYIOC6 End
```

BPXYIOV — Map the I/O vector structure

BPXYIOV is used by readv(), writev(), sendmsg() and recvmsg(). AMODE 31 callers use "BPXYIOV — Map the I/O vector structure" on page 986.

```
            SYSSTATE AMODE64=YES
            BPXYIOV    ,
** BPXYIOV: Socket I/O Vectors
**  Used By: FCT OPN
IOV                  DSECT ,
IOV_ENTRY            DS    0F     Array Entry
*  ------------------------- 64-bit format
IOV_BASE             DS    D      64-bit Address of buffer
IOV_LEN              DS    D      64-bit length of buffer
*
IOV#LENGTH          EQU   *-IOV_ENTRY Length of this structure
IOV_MAX             EQU   120     Maximum number of entries
** BPXYIOV End
```

BPXYIPCQ — Map w_getipc structure

AMODE 31 callers use "BPXYIPCQ — Map w_getipc structure" on page 987.

```
            SYSSTATE AMODE64=YES
            BPXYIPCQ   ,
*********************************************************************
*                                                                   *
*  BPXYIPCQ: w_getipc interface mapping                             *
*   Used By: BPXGXGET                                               *
*                                                                   *
*********************************************************************
IPCQ          DSECT ,      Interprocess Communications - Query
IPCQLENGTH    DS    F      IPCQ#LENGTH used by system call.  If not
*                          equal, check BPXYIPCQ and system levels.
IPCQTYPE      DS    CL4    "IMSG", "ISEM", "ISHM", "OVER", "IMAP"
IPCQOVER      DS    0D     OVERVIEW MAPPING STARTS HERE
*-------------------------------------------------------------------*
*  For IPCQTYPE = OVER, data starts here and the rest of the fields  *
*  in this section of code are not filled in.                       *
*-------------------------------------------------------------------*
IPCQMID       DS    FL4    MEMBER ID
IPCQKEY       DS    XL4    KEY
IPCQIPCP      DS    CL20   MAPPED BY BPXYIPCP
IPCQGTIME     DS    XL4    TIME_T OF LAST ...GET()
IPCQCTIME     DS    XL4    TIME_T OF LAST ...CTL()
IPCQTTIME     DS    XL4    TIME_T CHANGED BY TERMINATION
*-------------------------------------------------------------------*
*  Start of Unique data for IPCQTYPE requested                      *
*-------------------------------------------------------------------*
IPCQREST      DS    0C     IPCQMSG, IPCQSHM, IPCQSEM, MAPPED MEMORY
*********************************************************************
* Message Queue unique data                                        *
```

```
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQBYTES     DS    F       # BYTES OF MESSAGES ON QUEUE
IPCQQBYTES    DS    F       MAX # BYTES OF MESSAGES ALLOWED ON QUEUE
IPCQLSPID     DS    F       PID OF LAST MSGSND()
IPCQLRPID     DS    F       PID OF LAST MSGRCV()
IPCQSTIME     DS    F       TIME_T OF LAST MSGSND()
IPCQRTIME     DS    F       TIME_T OF LAST MSGRCV()
IPCQNUM       DS    F       # OF MESSAGES ON QUEUE
IPCQRCNT      DS    F       COUNT OF WAITING MSGRCV
IPCQSCNT      DS    F       COUNT OF WAITING MSGSND
              DS    0CL16   MSGRCV AND MSGSND WAITERS
              DS    0CL8     MSGRCV - WAIT FOR TYPE
IPCQQRPID     DS    F         PROCESS ID
IPCQQRMSGTYPE DS    F         MESSAGE TYPE
              DS    0CL8     MSGSND - WAIT FOR ROOM TO SEND
IPCQQSPID     DS    F         PROCESS ID
IPCQQSMSGLEN  DS    F         MESSAGE LENGTH
              DS    9CL16   MSGSND AND MSGRCV WAITERS
              DS    0CL8    MESSAGES WAITING TO BE RECEIVED
IPCQQMPID     DS    F         PROCESS ID
IPCQQMMSGTYPE DS    F         MESSAGE TYPE
              DS    9CL8    MESSAGES
              DS    F       Reserved
              DS    0D
* The 64 bit time fields will be set for either 31 or 64 bit mode
* Must define storage different, depending on how assembled
* AMODE 64
IPCQSTIME64   DS    FD      TIME64_T OF LAST MSGSND()
IPCQRTIME64   DS    FD      TIME64_T OF LAST MSGRCV()
IPCQQRMSGTYPE64 DS  10FD    MSGRCV 64 BIT MSG TYPE
IPCQQMMSGTYPE64 DS  10FD    MSG WAITING 64 BIT MSG TYPE
              DS    CL96    Reserved for expansion
*********************************************************************
* Semaphore unique data                                             *
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQLOPID     DS    XL4     PID OF LAST SEMOP
IPCQOTIME     DS    F       TIME_T LAST SEMOP
IPCQADJBADCNT DS    F       TERMINATION BUMPS SEM_VAL LIMITS
IPCQNSEMS     DS    FL2     NUMBER OF SEMAPHORES IN THIS SET
IPCQADJCNT    DS    FL2     NUMBER OF UNDO STRUCTURES
IPCQNCNT      DS    FL2     COUNT OF WAITERS FOR >0
IPCQZCNT      DS    FL2     COUNT OF WAITERS FOR =0
              DS    0CL16   WAITERS AND ADJUSTERS
              DS    0CL8     WAITER
IPCQSWPID     DS    F         PROCESS ID
IPCQSWNUM     DS    H         SEMAPHORE NUMBER
IPCQSWOP      DS    H         SEMAPHORE OPERATION
              DS    0CL8     ADJUSTER
IPCQSAPID     DS    F         PROCESS ID
IPCQSANUM     DS    H         SEMAPHORE NUMBER
IPCQSAADJ     DS    H         SEMAPHORE OPERATION
              DS    9CL16   WAITERS AND ADJUSTERS
              DS    0D
* AMode 64
IPCQOTIME64   DS    FD      TIME64_T LAST SEMOP
              DS    CL360   Reserved for expansion
*********************************************************************
* Shared Memory unique data                                         *
*********************************************************************
              ORG   IPCQREST
              DS    0F
IPCQACNT      DS    F       USE COUNT (#SHMAT - #SHMDT)
              DS    F       RESERVED IN 64 BIT MODE
```

```
IPCQDTIME     DS    F      TIME_T OF LAST SHMDT()
IPCQATIME     DS    F      TIME_T OF LAST SHMAT()
IPCQLPID      DS    F      PID OF LAST SHMAT() OR SHMDT()
IPCQCPID      DS    XL4    PID OF CREATOR
*-------------------------------------------------------------------*
* 31 bit callers - 10 Element array of segments attached             *
*    Each element is the 4 byte PID followed by the 31 bit address   *
*-------------------------------------------------------------------*
IPCQATPID64   DS    F      ATTACHED PROCESS ID
              DS    F      Reserved
IPCQATADDRESS64 DS  AD     Segment addresses for process
              DS  18FD     Rest of elements
*
IPCQDTIME64   DS    FD     TIME_T OF LAST SHMDT()
IPCQATIME64   DS    FD     TIME_T OF LAST SHMAT()
*
IPCQSEGSZ     DS    FD     MEMORY SEGMENT SIZE
              DS    CL344  Reserved
*********************************************************************
* Mapped Memory unique data                                         *
*********************************************************************
              ORG   IPCQREST Mapped Memory unique data
              DS    0F
IPCQMAPCPID   DS    F      CREATOR PROCESS ID
IPCQMAPUPID   DS    F      USER PROCESS ID
IPCQMAPTOKEN  DS    2F     MAP TOKEN
IPCQMAPUID    DS    F      USER'S EFFECTIVE UID
IPCQMAPGID    DS    F      USER'S EFFECTIVE GID
IPCQMAPFLAGS  DS    XL4    FLAGS
* Flags in first byte
IPCQMAPSHUT   EQU   X'80'  SHUTDOWN OF OBJECT
IPCQBLKSZ     DS    F      SIZE OF BLOCKS IN MEGS
IPCQBLKSINUSE DS    F      NUMBER OF BLOCKS IN USE
IPCQBLKSINMAP DS    F      NUMBER OF BLOCKS IN MAP AREA
IPCQBLKSMAPPED DS   F      NUMBER OF BLOCKS MAPPED
*                          BY THIS PROCESS
              DS    CL508  Reserved for expansion
*********************************************************************
* Continuation of Common data                                       *
* This next ORG gets us past the largest unique section of data     *
* We need to preserve the field offsets from prior releases so      *
* needed to add the rest of this common data at the end of the      *
* unique data instead of within the common area defined above.      *
*********************************************************************
              ORG
IPCQGTIME64   DS    FD     TIME64_T OF LAST ...GET()
IPCQCTIME64   DS    FD     TIME64_T OF LAST ...CTL()
IPCQTTIME64   DS    FD     TIME64_T CHANGED BY TERMINATION
IPCQSECLABEL  DS    FD     SECLABEL
*********************************************************************
* Overview - summary data for msgqs, semaphores, shared memory      *
*********************************************************************
              ORG   IPCQOVER Overview
                 DS    0F  MESSAGE QUEUES
IPCQOMSGNIDS     DS    F        Maximum number MSQs allowed
IPCQOMSGHIGHH2O  DS    F        Most MSQs at one time
IPCQOMSGFREE     DS    F        Number MSQs available
IPCQOMSGPRIVATE  DS    F        Number MSQs with Ipc_PRIVATE
IPCQOMSGKEYED    DS    F        Number MSQs with KEYs
IPCQOMSGREJECTS  DS    F        TIMES MSGGET DENIED
IPCQOMSGQBYTES   DS    F        MAX BYTES PER QUEUE
IPCQOMSGQMNUM    DS    F        MAX NUMBER MESSAGES PER QUEUE
IPCQOMSGNOALC    DS    F        # MSGSNDS THAT RETURNED ENOMEM
                 DS    F
                 DS    0F  SEMAPHORE
IPCQOSEMNIDS     DS    F        Maximum number SEMs allowed
IPCQOSEMHIGHH2O  DS    F        Most SEMs at one time
```

```
IPCQOSEMFREE     DS    F        Number SEMs available
IPCQOSEMPRIVATE  DS    F        Number SEMs with Ipc_PRIVATE
IPCQOSEMKEYED    DS    F        Number SEMs with KEYs
IPCQOSEMREJECTS  DS    F        TIMES SEMGET DENIED
IPCQOSEMSNSEMS   DS    F        MAX NUMBER OF SEMAPHORES PER SET
IPCQOSEMSNOPS    DS    F        MAX NUMBER OPERATION IN SEMOP
IPCQOSEMSBYTES   DS    F        STORAGE LIMIT
IPCQOSEMCBYTES   DS    F        STORAGE COUNT
                 DS    F
                 DS    0F  SHARED MEMORY
IPCQOSHMNIDS     DS    F        Maximum number SHMs allowed
IPCQOSHMHIGHH2O  DS    F        Most SHMs at one time
IPCQOSHMFREE     DS    F        Number SHMs available
IPCQOSHMPRIVATE  DS    F        Number SHMs with Ipc_PRIVATE
IPCQOSHMKEYED    DS    F        Number SHMs with KEYs
IPCQOSHMREJECTS  DS    F        TIMES SHMGET DENIED
IPCQOSHMSPAGES   DS    F        MAX # PAGES PER SYSTEM LIMIT
IPCQOSHMMPAGES   DS    F        MAX # PAGES PER SEGMENT LIMIT - ZERO
*                               IF 32 BITS EXCEEDED - USE
*                               IPCQOSHMMPAGES64 FOR GREATER THAN 32
*                               BITS
IPCQOSHMNSEGS    DS    F        MAX # SEGMENTS PER PROCESS LIMIT
IPCQOSHMCPAGES   DS    F        CURRENT # BYTES SYSTEM WIDE
*                               This field does not include pages for
*                               shared memory requests processed with
*                               the ipc_MEGA option
IPCQOSHMBIGGEST  DS    F        LARGEST SEGMENT ALLOCATED - ZERO IF
*                               32 BITS EXCEEDED - USE
*                               IPCQOSHMBIGGEST64 FOR GREATER THAN 32
*                               BITS
                 DS    0D
IPCQOSHMMPAGES64  DS   FD       MAX # PAGES PER SEGMENT LIMIT
IPCQOSHMBIGGEST64 DS   FD       LARGEST SEGMENT ALLOCATED
              ORG   ,
IPCQ#LENGTH   EQU   *-IPCQ Storage needed for w_getipc function
* w-getipc Command:
IPCQ#MSG      EQU   1      Retrieve next message queue
IPCQ#SHM      EQU   2      Retrieve next shared memory segment
IPCQ#SEM      EQU   3      Retrieve next semaphore set
IPCQ#ALL      EQU   4      Retrieve next member, all mechanisms
IPCQ#OVER     EQU   5      Retrieve overview
IPCQ#MAP      EQU   6      Retrieve mapped memory
** BPXYIPCQ End
```

BPXYITIM — Map getitimer, setitimer structure

AMODE 31 callers use "BPXYITIM — Map getitimer, setitimer structure" on page 990.

```
            SYSSTATE AMODE64=YES
            BPXYITIM   ,
** BPXYITIM: getitimer and setitimer interval structure
**  Used By: GTR STR
ITIM                 DSECT ,
** STRUCTURE OF GETITIMER (PARAMETER 2), SETITIMER (PARAMETERS 2,3)
ITIMIPAIR            DS    0CL16     Initial value or value at cancel
ITIMISECONDS         DS    FD          Seconds       0-7FFFFFFF   x
                     DS    F           Padding
ITIMIMICROSEC        DS    0F          Microseconds  0-000F423F   x
ITIMINANOSEC         DS    F           Nanoseconds   0-369AC9FF   x
ITIMRPAIR            DS    0CL16     Reload Interval
ITIMRSECONDS         DS    FD          Seconds       0-2147483647 d
                     DS    F           Padding
ITIMRMICROSEC        DS    0F          Microseconds  0-999999     d
ITIMRNANOSEC         DS    F           Nanoseconds   0-999999999  d
ITIMER_REAL          EQU   0         REAL TIME
```

```
ITIMER_VIRTUAL       EQU   1         VIRTUAL TIME (CPU - SYSTEM)
ITIMER_PROF          EQU   2         CPU TIME
ITIMER_MICRO         EQU   0         1/1,000,000 of seconds
ITIMER_NANO          EQU   4         1/1,000,000,000 of seconds
ITIM#LENGTH          EQU   32        LENGTH THIS STRUCTURE
** BPXYITIM End
```

BPXYMMG — Map Interface for _map_init and _map_service

AMODE 31 callers use "BPXYMMG — Map interface for _map_init and _map_service" on page 991.

```
           SYSSTATE AMODE64=YES
           BPXYMMG    ,
** BPXYMMG: BPX1MMI & BPX1MMS Interface Declares
**  Used By: Callers of the BPX1MMI & BPX1MMS Interface
*
*********************************************************************
*
*   Function Code Constants
*
*********************************************************************
*
MMG_INIT               EQU 1
MMG_SERVICE            EQU 2
*
*********************************************************************
*                                                                   *
*   Parameter list mapping for the BPX1MMI MMG_INIT call            *
*                                                                   *
*********************************************************************
*
_MMG_INIT_PARM   DSECT ,        MMG_INIT Parameter List
_MMG_NUMBLKS     DS    F        Fullword that contains the number of
*                               blocks to be contained in the map
*                               area.
_MMG_MEGSPERBLK  DS    F        Fullword that contains the size in
*                               megabytes of each block in the map
*                               area
_MMG_MAPTOKEN    DS    CL8      Token for map area
_MMG_AREAADDR    DS    AD       Doubleword that contains, on input,
*                               the suggested starting address of the
*                               map area or 0.  On output, this field
*                               is set to the actual map starting
*                               address.
_MMG_INIT_PARM_LEN EQU *-_MMG_INIT_PARM
*
*********************************************************************
*
*   Parameter list mapping for the BPX1MMS MMG_SERVICE request
*
*   The parameter list is an array of entries, each entry having the
*   format as mapped by _MMG_SERVICE_BLK.  Each entry is a request for
*   one of the supported request types:  MMG_NEWBLOCK, MMG_CONN,
*   MMG_DISCONN, MMG_CNTL or MMG_FREE.  In addition, an entry can be
*   marked as inactive by setting its value to MMG_NOP, which will
*   cause the entry to be skipped.  The result of a given request will
*   be reflected in the array entry.
*
*   The meaning of array entry fields is dependant on the requested
*   function.  The following table defines the field meanings for each
*   of the supported functions. A field not used by a service is marked
*   N/A.  Fields so marked are ignored and their value is not
*   important for the specified service.  All reserved fields must be
*   zero.
*
```

```
*   Function      Field               Field usage
*   ---------------------------------------------------
*   _newblock
*                 _MMG_SERVICETYPE    MMG_NEWBLOCK
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     MMG_NOCONN may be set to one if
*                                     the new block is to be allocated
*                                     in the backing storage but not
*                                     connected to the map area
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          output
*                 _MMG_BlkAddr        input  - 0 or address where the
*                                              new block is to be
*                                              allocated
*                                     output - An address in the map
*                                              area where the new
*                                              block was allocated
*   _conn
*                 _MMG_SERVICETYPE    MMG_CONN
*                 _MMG_SERVICEIFLAG   All bits should be zero
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          input
*                 _MMG_BlkAddr        input  - 0 or address where the
*                                              block identified by
*                                              token is to be
*                                              allocated
*                                     output - An address in the map
*                                              area where the block
*                                              was allocated
*   _disconn
*                 _MMG_SERVICETYPE    MMG_DISCONN
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     the MMG_FREE bit may be on if
*                                     backing storage is to be
*                                     released for the data
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          N/A
*                 _MMG_BlkAddr        input  - Address of the block
*                                              containing data to
*                                              be disconnected
*   _free
*                 _MMG_SERVICETYPE    MMG_FREE
*                 _MMG_SERVICEIFLAG   All bits should be zero
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          input  - Token of the data
*                                              contained in the
*                                              backing storage which
*                                              is to be release
*                 _MMG_BlkAddr        N/A
*   _cntl
*                 _MMG_SERVICETYPE    MMG_CNTL
*                 _MMG_SERVICEIFLAG   All bits should be zero except
*                                     those that define the access
*                                     state of the data (read or
*                                     read/write flags)
*                 _MMG_SERVICEOFLAG   Should be zero, but not checked
*                 _MMG_Token          N/A
*                 _MMG_BlkAddr        input  - Address of the block
*                                              containing data to be
*                                              affected by the state
*                                              change
*
********************************************************************
*
_MMG_SERVICE_PARM DSECT ,       MMG_SERVICE Parameter List
_MMG_SERVICE_ENTRY DS 0H
_MMG_SERVICETYPE DS    FL2      Type of service requested. eg, MMG_CONN
```

```
_MMG_SERVICEIFLAG DS   BL1      Flags
                  ORG  _MMG_SERVICEIFLAG
_MMG_READONLY     EQU  X'80'    All pages of each area are to be made
*                              read-only
_MMG_READWRITE    EQU  X'40'    All pages of each area are to be made
*                              read-write
_MMG_FREEBLOCK    EQU  X'20'    The backing storage for the specified
*                              block is to be freed
_MMG_NOCONN       EQU  X'10'    The new block is to be allocated in the
*                              backing storage but not connected to
*                              the map area
                  ORG  _MMG_SERVICEIFLAG+L'_MMG_SERVICEIFLAG
_MMG_SERVICEOFLAG DS   BL1      Flags
                  ORG  _MMG_SERVICEOFLAG
_MMG_REQFAIL      EQU  X'80'    If on, a failure occured on this entry
*                              or this entry was not processed
                  ORG  _MMG_SERVICEOFLAG+L'_MMG_SERVICEOFLAG
_MMG_TOKEN        DS   CL8      Token for a data block
_MMG_RES02B       DS   A        Reserved
_MMG_BLKADDR      DS   AD       Doubleword that contains the virtual
*                              address of a map area block
_MMG_MAXARRAYCOUNT EQU 1000     Maximum number of requests that can be
*                              in a service request array
_MMG_SERVICE_PARM_LEN EQU *-_MMG_SERVICE_PARM
*
*********************************************************************
*
*   BPX1MMS SERVICE Request Constants (values for field
*   _MMG_SERVICETYPE)
*
*********************************************************************
*
MMG_NOP                EQU 0
MMG_NEWBLOCK           EQU 1
MMG_CONN               EQU 2
MMG_DISCONN            EQU 3
MMG_FREE               EQU 4
MMG_CNTL               EQU 5
*
*********************************************************************
*
** BPXYMMG End
```

BPXYMSG — Map interprocess communication message queues

DSECT (MSGBUF) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. Default for the message size is 100 bytes. Specify VARLEN= to override this value.

AMODE 31 callers use "BPXYMSG — Map interprocess communication message queues" on page 997.

```
           SYSSTATE AMODE64=YES
           BPXYMSG    ,
** BPXYMSG: Interprocess Communication Message Queue Structure
**  Used By: msgctl
MSQID_DS            DSECT ,      message queue structure
MSG_PERM            DS    CL(IPC#LENGTH) Mapped by BPXYIPCP
MSG_QNUM            DS    F      # of messages on queue
MSG_QBYTES          DS    F      max bytes allowed on queue
MSG_LSPID           DS    F      process ID of last msgsnd()
MSG_LRPID           DS    F      process ID of last msgrcv()
MSG_STIME           DS    F      time of last msgsnd()
MSG_RTIME           DS    F      time of last msgrcv()
```

```
MSG_CTIME           DS    F      time of last change  get/ctl
MSG_STIME64         DS    FD     time64_t of last msgsnd()
MSG_RTIME64         DS    FD     time64_t of last msgrcv()
MSG_CTIME64         DS    FD     time64_t of last change  get/ctl
MSQ#LENGTH          EQU *-MSQID_DS Length of this DSECT
MSGBUF              DSECT ,      Message buffer - msgsnd, msgrcv
MSG_TYPE            DS    FD     64 bit message type
MSG_MTEXT           DS    CL100  Message text
MSGB#LENGTH         EQU *-MSGBUF Length of this DSECT
MSGXBUF             DSECT ,      Message buffer - msgxrcv
MSGX_MTIME          DS    FD     time message sent
MSGX_UID            DS    F      sender's effective UID
MSGX_GID            DS    F      sender's effective GID
MSGX_PID            DS    F      sender's PID
MSGX_TYPE           DS    FD     Message type
MSGX_MTEXT          DS    CL100  Message text
MSGX#LENGTH         EQU *-MSGXBUF Length of this DSECT
* Flag bits - msgrcv (also IPC_NOWAIT
MSG_NOERROR         EQU   4      No error if big message.
MSG_INFO            EQU   8      Use MSGXBUF not MSGBUF format
** BPXYMSG End
```

BPXYMSGH — Map the message header

BPXYMSGH is used by the sendmsg and recvmsg syscalls. AMODE 31 callers use "BPXYMSGH — Map the message header" on page 999.

```
           SYSSTATE AMODE64=YES
           BPXYMSGH   ,
** BPXYMSGH: MSGH system call structure
**  Used By: SendMsg / RecvMsg
MSGH                DSECT ,
MSGHBEGIN           DS    0D
* ------------- 64-Bit Version
MSGHNAMEPTR         DS    D'0'   Pointer to sockaddr
MSGHIOVPTR          DS    D'0'   Pointer to an array of IOVEC buffers.
MSGHCONTROLPTR      DS    D'0'   Pointer to ancillary data buffer
MSGHFLAGS           DS    F'0'   Output flags on received message
MSGHNAMELEN         DS    F'0'   Buffer length.
MSGHIOVNUM          DS    F'0'   Number of elements in IOVEC array.
MSGHCONTROLLEN      DS    F'0'   Length of ancillary data buffer
*
*   Constants
*
MSGH#LENGTH         EQU   *-MSGH   Length of MsgH
*
CMSGPTR             DS    A(0)   CMsg pointer
*
CMSGHDR             DSECT ,
CMSGLEN             DS    F'0'   Length, including header
CMSGLEVEL           DS    F'0'   Level
CMSGTYPE            DS    F'0'   Type
CMSGDATA            DS    0C     Data
*
*   Constants
*
SCM_RIGHTS          EQU   1        Access Rights
SCM_SECINFO         EQU   16386    Security Information
*
** BPXYMSGH End
```

BPXYOCRT — Map the OE certificate support structure

AMODE 31 callers use "BPXYOCRT — Map the OE certificate support structure" on page 1002.

```
           SYSSTATE AMODE64=YES
           BPXYOCRT   ,
** BPXYOCRT: OE Certificate support structure
**  Used By: TLS
OCRT       DSECT ,
OCRTTYPE   DS    F      type of certificate attached
OCRTUSERID DS    CL9    MVS userid, null terminated, input/output
           DS    CL3    reserved
OCRTCLEN   DS    F      length of certificate associated with type
OCRTCPTR   DS    A      31-Bit ptr to the actual certifcate
OCRTCERTPTR64 DS    D      64-Bit ptr to the actual certifcate
OCRT_LEN   EQU   *-OCRT
OCRT_X509  EQU   1      Certificate type X509
** BPXYOCRT End
```

BPXYPPSD — Map signal delivery data

This structure is passed to a signal interface routine (SIR). AMODE 31 callers use "BPXYPPSD — Map signal delivery data" on page 1014.

```
           SYSSTATE AMODE64=YES
           BPXYPPSD   ,
** BPXYPPSD: Signal Data Area
**  Used By: User written signal interrupt routines
PPSD                DSECT ,
PPSDID              DC    C'PPSD'  Eye catcher
PPSD#ID             EQU   C'PPSD'    Control Block Acronym
PPSDSP              DS    FL1      Subpool number of this PPSD
PPSD#SP             EQU   230        Subpool for the PPSD
PPSDLEN             DC    AL3(PPSD#LENGTH)   Length this structure
*
*    ****************************************************************
*     PpsdSIRParms is used to setup up a parameter list to the
*     Signal Interface Routine (SIR).  When the SIR is invoked, the
*     address of PpsdSIRParms field is set in Register 1.  The
*     PpsdAddrPpsd contains the address of the Ppsd.
*    ****************************************************************
*
PPSDSIRPARMS        DS    0A       SIR Parameters
PPSDADDRPPSD        DC    A(PPSD)  Pointer to the top of the Ppsd
PPSDSIRPARMEND      EQU   X'80'    End of Parameters flag set on
PPSDTRMEXITSTATUS   DS    F        4 Byte status passed to PRTRM
PPSDSIGNUM          DS    F        Signal number
PPSDFL              DS    XL2      X'7FFF' reserved
                    ORG   PPSDFL
PPSDFLAGS2A         DS    0B
PPSDQUIESCEFREEZE   EQU   X'80'      Interrupt due to freeze
PPSDSIRCOMPLETE     EQU   X'40'    Sir done with async I/O exits
PPSDPROCDFT         EQU   X'20'    Process default
PPSDSIGQUEUE        EQU   X'10'    NSSGQ queued signal
PPSDREDRIVE         EQU   X'08'    SPB will Resend signal later
PPSDJUMPBACK        EQU   X'04'    SPB return to point of interrupt
PPSDMASKONLY        EQU   X'02'    SPB restore mask only
PPSDSIGTHSTOP       EQU   X'01'    Interrupt due to thread-stop
*                                  signal
                    ORG   PPSDFL+0001
PPSDFLAGS2B         DS    B
PPSDQUIESCEANDGET   EQU   X'80'    Interrupt due to
*                                  pthread_quiesce_and_get_np
PPSDF2_64           EQU   X'40'    Use PSWxxx64 fields
PPSDACTION          DS    B        Action for this signal
*                                     catch
*                                     SIR determines default action
PPSDFLAGS           DS    B        X'00' reserved
PPSDASYNC           EQU   X'80'    Signal delivered Asynchronously
```

```
PPSDDUMP             EQU   X'40'    Dump for terminating signals
PPSDPTHREADKILL      EQU   X'20'    Signal sent via BPX1PTK
PPSDTHISTHREADGEN    EQU   X'10'    Sending=Receiving thread
PPSDSIGNAL           EQU   X'08'    Interrupt due to signal
PPSDCANCEL           EQU   X'04'    Interrupt due to cancel
PPSDQUIESCE          EQU   X'02'    Interrupt due to quiesce
PPSDIPT              EQU   X'01'    If ON then this is the IPT
PPSDRES1             DS    F        Reserved in 64 bit mode
PPSDSAMASK           DS    XL8      Signal mask set by BPX1SIA for
*                                   this signal
PPSDSAFLAGS          DS    XL4      X'00FFFFFF' reserved
PPSDNOCLDSTOP        EQU   X'80'      Do not generate SIGCHLD on stops
PPSDOLDSTYLE         EQU   X'40'      Signal defined by signal() funct.
PPSDONSTACK          EQU   X'20'      Deliver on alternate stack
PPSDRESETHAND        EQU   X'10'      Reset action on delivery
PPSDRESTART          EQU   X'08'      Restart interruptable funcs
PPSDSIGINF           EQU   X'04'      Pass sig info to catcher
PPSDNOCLDWAIT        EQU   X'02'      Don't create zombie on exit
PPSDNODEFER          EQU   X'01'      Don't block sig on delivery
PPSDCURRENTMASK      DS    XL8      This is the signal mask to be set
*                                   when the signal catcher returns.
*                                   Signal mask at time of interrupt
*                                   except for sigsuspend case. If
*                                   signal during sigsuspend, then
*                                   this mask is the signal mask prior
*                                   to call to sigsuspend.
PPSDRES2             DS    F        Reserved in 64 bit mode
PPSDRES3             DS    F        Reserved in 64 bit mode
PPSDGENREGS          DS    CL64     Users general regs at interrupt
PPSDRES4             DS    XL8      Reserved in 64 bit mode
PPSDARREGS           DS    16F      Users AR regs at interrupt
PPSDKILDATA          DS    FL2      User specified data on BPX1KIL
PPSDKILOPTS          DS    XL2      X'7FFF' reserved
*                                   User specified options on BPX1KIL
PPSDPTBYPASS         EQU   X'80'      Ptrace Bypass option in effect
PPSDKERNSICODE       EQU   X'40'      PpsdKilData=Kern set SiCode
PPSDAPPLSICODE       EQU   X'20'      PpsdKilData=Appl set SiCode
PPSDCONSCANCEL       EQU   X'10'      Console MODIFY cancel qualifier
PPSDTRACEOVERRIDE    EQU   X'04'      SYSCALL Trace Override Option
PPSDTRACEACTION      EQU   X'02'      SYSCALL Trace Action Setting

*                                       in PpdsKilData
PPSDSUPERKILL        EQU   X'08'      Superkill option on BPX1KIL
PPSDRES5             DS    F        Reserved in 64 bit mode
PPSDLASTPTSIG        DS    F        Last Ptraced Signal
PPSDRES6             DS    2F       Reserved in 64 bit mode
PPSDSENDINGTHREAD    DS    CL8      Sending thread id
PPSDTARGETTHREAD     DS    CL8      Target thread id
PPSDSENDINGPID       DS    F        Sending process id
PPSDSENDINGUID       DS    F        Sending real uid
PPSDRES7             DS    F        Reserved in 64 bit mode
PPSDSISTATUS         DS    F        Exit status or signal
PPSDRES8             DS    F        Reserved in 64 bit mode
PPSDERRNO            DS    F        Error return code
PPSDCATCHERMASK      DS    XL8      Signal Mask to be set before signal
*                                   catcher is called. If signal during
*                                   sigsuspend then this field is same
*                                   as mask specified on sigsuspend. If
*                                   not sigsuspend, then PpsdCatcherMask
*                                   and PpsdCurrentMask are equal.
PPSDRES10            DS    25F       Reserved
PPSDRES9             DS    F         Reserved in 64 bit mode
PPSDREDRIVETIME      DS    F         Time to delay signal 1000 per mic
PPSDG64H             DS    16F       Users G64H at interrupt
PPSDRRTRMMSGTHID     DS    CL8       Sending thread id for MSG
*                                   BPXP010I
PPSDSENDINGJOBNAME   DS    CL8       Jobname of thread sending signal
```

```
*
                    DS    4F       Reserved
PPSDSAHANDLER       DS    AD       Addr of catcher function
PPSDSIR             DS    AD       Addr Signal interrupt routine
PPSDUSERDATA        DS    AD       User data speicified on BPX1MSS
PPSDPSW             DS    XL16     Users PSW at interrupt
PPSDQUIESCEDATA     DS    FD       Quiesce_Data specified on BPX1QUT
*
PPSDSIGACTIONDATA   DS    FD       User_Data specified on BPX1SIA
PPSDPTXLWAPTR       DS    AD       Threads workarea address specified
*                                  on BPX1PTC (pthread_create).  This
*                                  address is zero if the thread was
*                                  not pt_created.
PPSDSIADDR          DS    AD       Address of faulting instruction
*                                  for SIGILL, SIGFPE, SIGSEGV
PPSDSIBAND          DS    FD       Band event
PPSDSQV             DS    FD       Signal si_value
                    DS    4F       Reserved
                    DS    FL2      Reserved
PPSDAIOCB64         DS    FL2      Amode(64) Exit Flags
PPSDEXCOUNT         DS    FL2      Count of PpsdAiocb's
PPSDEXLASTIX        DS    FL2      Last array index used
PPSDAIOCB           DS    12D      Aiocb Array for Async Exit
PPSDEND             DS    0D       End of PPSD on double word
PPSD#LENGTH         EQU   *-PPSD   Length of this structure
** BPXYPPSD End
```

BPXYPTXL — Map the parameter pist for pthread_create

AMODE 31 callers use “BPXYPTXL — Map the parameter list for pthread_create” on page 1032.

```
           SYSSTATE AMODE64=YES
           BPXYPTXL   ,
** BPXYPTXL: Pthread Parameter List
**  Used By: PTX
PTXL                DSECT ,     Parm List returned by BPX1PTX
PTXLWORKAREAPTR     DS    AD    Pointer to User Work Area
PTXLATTRIBUTEPTR    DS    AD    Pointer to User Attributes
PTXLTHIDPTR         DS    A     Pointer to Thread ID
PTXLSTATUSPTR       DS    A     Pointer to Thread Run Status
PTXL#LENGTH         EQU   *-PTXL
PTXLRS              DSECT ,     Thread Run Status
                    DS    0F
PTXLRSFLAGS         DS    0BL4  Thread Run Status Flags
PTXLRSFLAGS0        DS    B     1st byte
PTXLRSREADY         EQU   X'80'   Thread is ready to run
PTXLRSFLAGS1        DS    B     2nd byte
PTXLRSFLAGS2        DS    B     3rd byte
PTXLRSFLAGS3        DS    B     4th byte
PTXLRS#LENGTH       EQU   *-PTXLRS
** BPXYPTXL End
```

BPXYRLIM — Map the rlimit, rusage, and timeval structures

AMODE 31 callers use “BPXYRLIM — Map the rlimit, rusage, and timeval structures” on page 1033.

```
           SYSSTATE AMODE64=YES
           BPXYRLIM   ,
** BPXYRLIM: Rlimit, Timeval, and Rusage Structures
**  Used By: setrlimit, getrlimit, and getrusage
RLIMIT              DSECT ,      Rlimit structure
RLIM_CUR_DW         DS    0CL8   Current limit (doubleword)
```

```
RLIM_CUR_HW          DS    F      Current (soft) limit highword -      X
                                    used only for RLIMIT_FSIZE         X
                                    and RLIMIT_MEMLIMIT, it is         X
                                    ignored for all other resources
RLIM_CUR             DS    0F     Current (soft) limit lowword
RLIM_CUR_LW          DS    F      Current (soft) limit lowword
RLIM_MAX_DW          DS    0CL8   Current limit (doubleword)
RLIM_MAX_HW          DS    F      Current (hard) limit highword -      X
                                    used only for RLIMIT_FSIZE         X
                                    and RLIMIT_MEMLIMIT, it is         X
                                    ignored for all other resources
RLIM_MAX             DS    0F     Maximum (hard) limit lowword
RLIM_MAX_LW          DS    F      Maximum (hard) limit lowword
RLIMIT#LENGTH        EQU *-RLIMIT Length of this DSECT
TIMEVAL              DSECT ,      Timeval structure
TMVL_SEC             DS    FD     Seconds
                     DS    F      Padding
TMVL_USEC            DS    F      Microseconds
TIMEVAL#LENGTH       EQU *-TIMEVAL Length of this DSECT
RUSAGE               DSECT ,      Rusage structure
RU_UTIME             DS    CL(TIMEVAL#LENGTH) User time used
RU_STIME             DS    CL(TIMEVAL#LENGTH) System time used
RUSAGE#LENGTH        EQU *-RUSAGE Length of this DSECT
** BPXYRLIM End
```

BPXYSELT — Map the timeout value for the select syscall

AMODE 31 callers use "BPXYSELT — Map the timeout value for the select syscall" on page 1037.

```
           SYSSTATE AMODE64=YES
           BPXYSELT   ,
** BPXYSELT: Select Time Structure
**  Used By: Select Syscall
SELT                 DSECT ,
SELTBEGIN            DS    0D
*-------------------------64-bit format
*
TV_SEC               DS    D'0'   Seconds
                     DS    F'0'   Padding
TV_USEC              DS    F'0'   Microseconds
*   Constants
*
SELT#LENGTH          EQU   *-SELT Length of SELT
** BPXYSELT End
```

BPXYSEM — Map interprocess communication semaphores

DSECTs (SEMID_DS, SEM_ARRAY and SEM_BUF_ELE) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use "BPXYSEM — Map interprocess communication semaphores" on page 1037.

```
           SYSSTATE AMODE64=YES
           BPXYSEM    ,
** BPXYSEM: Interprocess Communications Permission
**  Used By: XSO, XSC
SEMID_DS             DSECT ,      semctl structure
SEM_PERM             DS    CL(IPC#LENGTH)   Mapped by BPXYIPCP
SEM_NSEMS            DS    H      number of semaphores in set
                     DS    H      spacer
SEM_OTIME            DS    FL4    last semop() time
SEM_CTIME            DS    FL4    last time changed by semctl()
```

```
SEM_OTIME64          DS    FD     last semop() time64_t
SEM_CTIME64          DS    FD     last semctl() time64_t
SEM#LENGTH           EQU   *-SEMID_DS   Length of this DSECT
* SETVAL - a one element array for Semaphore_Number
* SETALL, GETALL - an array with Number_of_Semaphore elements
SEM_ARRAY            DSECT ,      SETALL, GETALL, SETVAL
SEM_ARRAY_VAL        DS    FL2    semaphore value
SEM_BUF_ELE          DSECT ,      sembuf element - semop
SEM_NUM              DS    FL2    semaphore number (0 to n-1)
SEM_OP               DS    FL2    semaphore operation
SEM_FLG              DS    H      operation flags
SEM#BUFLEN           EQU   *-SEM_BUF_ELE
* Flag bits - semop (also IPC_NOWAIT
SEM_UNDO             EQU   2      Set up adjust on exit entry.
* Control Commands - (also IPC_RMID, IPC_SET, IPC_STAT):
SEM_GETVAL           EQU   21     Get the current semaphore value
SEM_SETVAL           EQU   22     Change the semaphore value
SEM_GETPID           EQU   23     Get PID of last process to alter sem
SEM_GETNCNT          EQU   24     Get count of tasks waiting for val>0
SEM_GETZCNT          EQU   25     Get count of tasks waiting for val=0
SEM_GETALL           EQU   26     Get the current semaphore values
SEM_SETALL           EQU   27     Change the semaphore values
* Maximum and minimum values
SEM#MAX_VAL          EQU   32767  Maximum sem_val (min = 0)
SEM#MAX_ADJ          EQU   16383  Maximum sem_adj (min = -MAX)
** BPXYSEM End
```

BPXYSFPL — Map the send_file parameter list

AMODE 31 callers use "BPXYSFPL — Map the send_file parameter list" on page 1038.

```
            SYSSTATE AMODE64=YES
            BPXYSFPL   ,
** BPXYSFPL: SFPL system call structure
**  Used By: BPX1SF
SFPL                 DSECT ,
SFSOCKETDES          DS    F         Socket Descriptor
SFHEADERLEN          DS    F         Header Length
SFHEADERVPTR         DS    0F
SFHEADERALET         DS    F         Header Alet
                     DS    F         64-bit pointer below
SFFILEDES            DS    F         File Descriptor
SFFILEBYTESDW        DS    0F        Bytes to send Double Word (-1=all)
SFFILEBYTESH         DS    F          High Word
SFFILEBYTESL         DS    F          Low  Word
SFFILEOFFSETDW       DS    0F        Offset Double Word
SFFILEOFFSETH        DS    F          High Word
SFFILEOFFSETL        DS    F          Low  Word
SFFILESIZEDW         DS    0F        File Size Double Word
SFFILESIZEH          DS    F          High Word
SFFILESIZEL          DS    F          Low  Word
SFTRAILERLEN         DS    F         Trailer Length
SFTRAILERVPTR        DS    0F
SFTRAILERALET        DS    F          Trailer Alet
                     DS    F         64-bit pointer below
SFBYTESSENTDW        DS    0F        Bytes Sent Double Word
SFBYTESSENTH         DS    F          High Word
SFBYTESSENTL         DS    F          Low  Word
SFFLAGS              DS    0XL4      Control Flags
SFPLVERSION          DS    XL1        Version
SFFLAGBYTE2          DS    XL1       Reserved
SFFLAGBYTE3          DS    XL1       Reserved
SFFLAGBYTE4          DS    XL1       Flags
SF_CLOSE             EQU   2         Close Socket Descriptor
SF_REUSE             EQU   1         Reuse Socket Descriptor
```

```
SFHEADERPTR          DS    FL8       Header Ptr
SFTRAILERPTR         DS    FL8        Trailer Ptr
SFRESERVE            DS    CL12      Reserved
*
SFPLEND              EQU   *
*
SFPL#LENGTH          EQU   SFPLEND-SFPL
*
*  Constants
*
*
** BPXYSFPL End
```

BPXYSHM—Map interprocess communication shared memory segments

AMODE 31 callers use "BPXYSHM—Map interprocess communication shared memory segments" on page 1039.

```
           SYSSTATE AMODE64=YES
           BPXYSHM    ,
** BPXYSHM: Interprocess Communications Permission
**  Used By: XMC
SHMID_DS             DSECT ,      SHMID_DS - shmctl structure
SHM_PERM             DS    CL(IPC#LENGTH)   Mapped by BPXYIPC
SHM_SEGSZ            DS    F      size of segment in bytes
SHM_LPID             DS    F      process ID of last operation
SHM_CPID             DS    F      process ID of creator
SHM_NATTCH           DS    F      number of current attaches
SHM_ATIME            DS    F      time of last shmat
SHM_DTIME            DS    F      time of last shmdt
SHM_CTIME            DS    F      time of last change shmget/shmctl
SHM_RES2             DS    F      Reserved
SHM_FLAGS            DS    F      Flags
SHM_SEG64            EQU   X'80'  Shared memory above the bar
         ORG   SHM_FLAGS+1
SHM_DUMP_PRIO64      DS    FL1    Dump priority for this seg
SHM_RES3             DS    FL2    Reserved
SHM_SEGADDR64        DS    AD     Address of segment
SHM_SEGSIZE64        DS    FD     Size of segment in bytes
SHM_ATIME64          DS    FD     time64_t of last shmat
SHM_DTIME64          DS    FD     time64_t of last shmdt
SHM_CTIME64          DS    FD     time64_t of last change shmget/shmctl
*
* Mode bits (mapped over S_TYPE in BPXYMODE):
SHM_RDONLY           EQU   1      Attach read-only (else read-write)
SHM_RND              EQU   2      Round attach address to SHMLBA
SHMLBA               EQU   4096   Rounding boundary
SHM#LENGTH           EQU   *-SHMID_DS   Length of this DSECT
** BPXYSHM End
```

BPXYSINF — Map SIGINFO_T structure

DSECT (SIGINFO_T) will be generated with either DSECT=NO or DSECT=YES. If DSECT=NO is specified, you may need an additional DSECT / CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use "BPXYSINF — Map SIGINFO_T structure" on page 1042.

```
           SYSSTATE AMODE64=YES
           BPXYSINF   ,
** BPXYSINF: siginfo_t Structure
**  Used By: waitid
SIGINFO_T            DSECT ,      Siginfo_t structure
```

```
SI_SIGNO            DS    F      signal number
SI_ERRNO            DS    F      error number
SI_CODE             DS    F      signal code
SI_PID              DS    F      sending process ID
SI_UID              DS    F      real user ID of sending process
SI_RES01            DS    F      reserved in 64 bit mode
SI_STATUS           DS    F      exit value or signal
SI_RES02            DS    F      reserved in 64 bit mode
SI_RES03            DS    F      reserved in 64 bit mode
SI_RES04            DS    F      reserved in 64 bit mode
SI_ADDR             DS    AD     address of faulting instruction
SI_BAND             DS    FD     band event for SIGPOLL
SI_VALUE            DS    FD     signal value
SIGINFO#LENGTH      EQU *-SIGINFO_T Length of this DSECT
** BPXYSINF End
```

BPXYSSET — Map the sigaction set

DSECT=.. is not supported. The generated code will allocate SSETOPTION_FLAGS and a DSECT for SSET. This should be followed by CSECT statement to return to the current DSECT or CSECT. AMODE 31 callers use "BPXYSSET — Map the sigaction set" on page 1055.

```
           SYSSTATE AMODE64=YES
           BPXYSSET   ,
**  BPXYSSET: Macro which enables multiple signal calls
**  Used By:  SA2
SSETOPTION_FLAGS   DS    0F
SSETOPTION_FLAGS1  DS    FL1   FLAGS INDICATING CALLER OPTIONS
SSET_IGINVALID     EQU   X'80' IGNORE INVALID SIGNALS & SIGACTIONS     X
                             0=DO NOT IGNORE, 1=IGNORE
                   DS    3FL1  RESERVED
SSET               DSECT ,
SSETCONSOLMASK     DS  XL8    SIGNALS HAVING THE SAME FLAGS,MASK,      X
                              USERDATA, AND SIGNAL ACTION
SSETCOMPARE        DS  0CL28
SSETSAHANDLER      DS  AD     ADDRESS OF A SIGNAL HANDLER ROUTINE
SSETSAMASK         DS  XL8    VALUE FOR SIGACTION MASK
SSETUSERDATA       DS  FD     USER DEFINED DATA
SSETFLAGS          DS  XL4    VALUE FOR SIGACTION FLAGS (BPXYSIGH)
SSETRES01          DS  F      Reserved
SSET#LENGTH        EQU *-SSET LENGTH OF ONE SSET ENTRY
** BPXYSSET End
```

BPXYWLM — WLM constants and parameter list DSECTs

BPXYWLM work load manager constants and DSECTs. AMODE 31 callers use "BPXYWLM — WLM constants and parameter list DSECTs" on page 1069.

```
           SYSSTATE AMODE64=YES
           BPXYWLM    ,
** BPXYWLM: BPX1WLM Interface Declares
**  Used By: Callers of the BPX1WLM Interface
*
*   BPX1WLM Function Code Constants
*
WLM_QUERY_METRICS       EQU 1
WLM_QUERY_SCHEDENV      EQU 2
WLM_CHECK_SCHEDENV      EQU 3
WLM_DISCONNECT          EQU 4
WLM_DELETE_WORKUNIT     EQU 5
WLM_JOIN_WORKUNIT       EQU 6
WLM_LEAVE_WORKUNIT      EQU 7
WLM_CONNECT_WORKMGR     EQU 8
```

```
WLM_CONNECT_SERVERMGR   EQU 9
WLM_CREATE_WORKUNIT     EQU 10
WLM_CONTINUE_WORKUNIT   EQU 11
WLM_EXTRACT_WORKUNIT    EQU 12
WLM_EXPORT_WORKUNIT     EQU 13
WLM_UNDOEXPORT_WORKUNIT EQU 14
WLM_IMPORT_WORKUNIT     EQU 15
WLM_UNDOIMPORT_WORKUNIT EQU 16
WLM_QUERY_ENCLAVECLASS  EQU 17
WLM_CONNECT_EXPORTIMPORT EQU 18
*  Function codes 100-112 are reserved
ARM_BIND_THREAD               EQU 200
ARM_BLOCK_TRANSACTION         EQU 201
ARM_DESTROY_APPLICATION       EQU 202
ARM_DISCARD_TRANSACTION       EQU 203
ARM_GENERATE_CORRELATOR       EQU 204
ARM_GET_ARRIVAL_TIME          EQU 205
ARM_REGISTER_APPLICATION      EQU 206
ARM_REGISTER_METRIC           EQU 207
ARM_REGISTER_TRANSACTION      EQU 208
ARM_REPORT_TRANSACTION        EQU 209
ARM_START_APPLICATION         EQU 210
ARM_START_TRANSACTION         EQU 211
ARM_STOP_APPLICATION          EQU 212
ARM_STOP_TRANSACTION          EQU 213
ARM_UNBIND_THREAD             EQU 214
ARM_UNBLOCK_TRANSACTION       EQU 215
ARM_UPDATE_TRANSACTION        EQU 216
EWLM_CLASSIFY_CORRELATOR      EQU 217
*   BPX1WLM/BPX4WLM Parameter List Mappings
*
_WQM             DSECT ,        WLM_QUERY_METRICS Parameter List
_WQM_SYSI_PTR    DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM system information.
*                               This data is returned in the format
*                               of the IWMWSYSI mapping macro.
_WQM_SYSI_LEN    DS    AD       Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM system information
_WQM_END         DS    0C       End of WQM
*
_WQS             DSECT ,        WLM_QUERY_SCHEDENV Parameter List
_WQS_SETH_PTR    DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM scheduling
*                               environment information.
*                               This data is returned in the format
*                               of the IWMSET mapping macro.
_WQS_SETH_LEN    DS    AD       Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM scheduling environment data.
*
_WQS_END         DS    0C       End of _WQS
*
_WCS             DSECT ,        WLM_CHECK_SCHEDENV Parameter List
_WCS_SCH_ENV     DS    AD       Address of a 16 byte character string
*                               that contains the scheduling
*                               environment to be checked.
_WCS_SYS_NAME    DS    AD       Address of a 8 byte character string
*                               that contains the system name to be
*                               checked.
_WCS_END         DS    0C       End of _WCS
*
_WDC             DSECT ,        WLM_DISCONNECT Parameter List
_WDC_CONN_TKN    DS    AD       Address of an fullword that contains
*                               the connect token to be disconnected
```

```
*                                 from.
_WDC_END         DS    0C         End of _WDC
*
_WDW             DSECT ,          WLM_DELETE_WORKUNIT Parameter List
*
_WDW_ENC_TKN     DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 work unit to be deleted.
_WDW_END         DS    0C         End of _WDW
*
_WJW             DSECT ,          WLM_JOIN_WORKUNIT Parameter List
_WJW_ENC_TKN     DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 work unit to join.
_WJW_END         DS    0C         End of _WJW
*
_WLW             DSECT ,          WLM_LEAVE_WORKUNIT Parameter List
_WLW_ENC_TKN     DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 work unit to leave.
_WLW_END         DS    0C         End of _WLW
*
_WNW             DSECT ,          WLM_CONTINUE_WORKUNIT Parameter List
*
_WNW_ENC_TKN     DS    AD         Address of a doubleword to return the
*                                 the WLM enclave token of the created
*                                 work unit.
_WNW_END         DS    0C         End of _WNW
*
_WCW             DSECT ,          WLM_CREATE_WORKUNIT Parameter List
*
_WCW_ENC_TKN     DS    AD         Address of a doubleword to return the
*                                 the WLM enclave token of the created
*                                 work unit.
_WCW_CLASSIFY    DS    AD         Address of a fullword pointer that
*                                 contains the address of a IWMCLSFY
*                                 Parameter List.
_WCW_ARR_TIME    DS    AD         Address of a doubleword field that
*                                 contains the arrival time of the
*                                 work request in STCK format.
_WCW_FUNC_NAME   DS    AD         Address of a 8 byte character string
*                                 that contains the descriptive function
*                                 name of the work request.
_WCW_END         DS    0C         End of _WCW
*
_WSC             DSECT ,          WLM_CONNECT_SERVERMGR Parameter List
*
_WSC_SUB_SYS     DS    AD         Address of a 4 byte character string
*                                 that contains the subsystem type the
*                                 server manager is requesting connection
*                                 for.
_WSC_SUB_SYS_NM  DS    AD         Address of a 8 byte character string
*                                 that contains the subsystem name the
*                                 server manager is requesting connection
*                                 for.
_WSC_APPL_ENV    DS    AD         Address of a 32 byte character string
*                                 that contains the application
*                                 environment name associated with the
*                                 server.
_WSC_PAR_EU      DS    AD         Address of a fullword that contains
*                                 number of parallel execution units
*                                 in the server environment.
_WSC_END         DS    0C         End of _WSC
*
_WWC             DSECT ,          WLM_CONNECT_WORKMGR Parameter List
*
_WWC_SUB_SYS     DS    AD         Address of a 4 byte character string
```

```
*                                 that contains the subsystem type the
*                                 work manager is requesting connection
*                                 for.
_WWC_SUB_SYS_NM  DS    AD         Address of a 8 byte character string
*                                 that contains the subsystem name the
*                                 work manager is requesting connection
*                                 for.
_WWC_END         DS    0C         End of _WWC
*
_WEW             DSECT ,          WLM_EXTRACT_WORKUNIT Parameter List
*
_WEW_ENC_TKN     DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 active work unit.
_WEW_END         DS    0C         End of _WEW
*
_WXW             DSECT ,          WLM_EXPORT_WORKUNIT Parameter List
*
_WXW_ENC_TKN     DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 work unit to be exported.
_WXW_EXP_TKN     DS    AD         Address of the 32 bytes to return the
*                                 WLM export token of the exported work
*                                 unit.
_WXW_CONN_TKN    DS    AD         Address of a fullword that contains
*                                 the connect token associated with the
*                                 workmanager.
_WXW_END         DS    0C         End of _WXW
*
_WUXW            DSECT ,          WLM_UNEXPORT_WORKUNIT Parameter List
*
_WUXW_EXP_TKN    DS    AD         Address of the 32 bytes that contains
*                                 the WLM export token representing the
*                                 exported work unit.
_WUXW_CONN_TKN   DS    AD         Address of a fullword that contains
*                                 the connect token associated with the
*                                 workmanager.
_WUXW_END        DS    0C         End of _WUXW
*
_WIW             DSECT ,          WLM_IMPORT_WORKUNIT Parameter List
*
_WIW_EXP_TKN     DS    AD         Address of the 32 bytes that contains
*                                 the WLM export token representing the
*                                 exported work unit.
_WIW_ENC_TKN     DS    AD         Address of a doubleword to return the
*                                 WLM enclave token of the imported work
*                                 unit.
_WIW_CONN_TKN    DS    AD         Address of a fullword that contains
*                                 the connect token associated with the
*                                 workmanager.
_WIW_END         DS    0C         End of _WIW
*
_WUIW            DSECT ,          WLM_UNIMPORT_WORKUNIT Parameter List
*
_WUIW_EXP_TKN    DS    AD         Address of the 32 bytes that contains
*                                 the WLM export token representing the
*                                 imported work unit.
_WUIW_CONN_TKN   DS    AD         Address of a fullword that contains
*                                 the connect token associated with the
*                                 workmanager.
_WUIW_END        DS    0C         End of _WUIW
*
_WQEC            DSECT ,          WLM_QUERY_ENCLAVECLASS Parameter List
*
_WQEC_ENC_TKN    DS    AD         Address of a doubleword that contains
*                                 the WLM enclave token representing the
*                                 work unit to be queried.
```

```
_WQEC_SYSEC_PTR  DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the WLM Query Enclave Data.
*                               This data is returned in the format
*                               of the IWMECD mapping macro.
_WQEC_SYSEC_LEN  DS    AD       Address of a fullword that contains
*                               the length of the buffer to return
*                               the WLM Query Enclave Data.
_WQEC_END        DS    0C       End of WQEC
*
_WCEI            DSECT ,        WLM_CONNECT_EXPORTIMPORT Parameter List
*
_WCEI_SUB_SYS    DS    AD       Address of a 4 byte character string
*                               that contains the subsystem type the
*                               work manager is requesting connection
*                               for.
_WCEI_SUB_SYS_NM DS    AD       Address of a 8 byte character string
*                               that contains the subsystem name the
*                               work manager is requesting connection
*                               for.
_WCEI_END        DS    0C       End of _WCEI
*
_ABI             DSECT ,        ARM_BIND_THREAD Parameter
*                               List
_ABI_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ABI_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_ABI_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ABI_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ABI_END         DS    0C       End of _ABI
*
_ABT             DSECT ,        ARM_BLOCK_TRANSACTION Parameter
*                               List
_ABT_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ABT_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_ABT_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ABT_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ABT_BLOCK_HDL   DS    AD       Address of a fullword pointer that
*                               contains the address of the 8 byte
*                               field to return the block handle.
_ABT_END         DS    0C       End of _ABT
*
_ADA             DSECT ,        ARM_DESTROY_APPLICATION Parameter
*                               List
_ADA_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ADA_APPL_ID     DS    AD       Address of a 16 byte field
*                               that contains the application ID.
_ADA_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ADA_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ADA_END         DS    0C       End of _ADA
*
_ADT             DSECT ,        ARM_DISCARD_TRANSACTION Parameter
*                               List
_ADT_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ADT_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
```

```
_ADT_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ADT_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ADT_END         DS    0C       End of _ADT
*
_AGC             DSECT ,        ARM_GENERATE_CORRELATOR Parameter
*                               List
_AGC_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AGC_APP_HDL     DS    AD       Address of a 8 byte field that
*                               contains the application handle.
_AGC_TRAN_ID     DS    AD       Address of a 16 byte field that
*                               contains the transaction ID.
_AGC_PAR_CORR    DS    AD       Address of a data area that
*                               contains the parent correlator.
_AGC_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AGC_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AGC_CUR_CORR    DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the current correlator.
_AGC_END         DS    0C       End of _AGC
*
_AGT             DSECT ,        ARM_GET_ARRIVAL_TIME Parameter
*                               List
_AGT_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AGT_TIMESTAMP   DS    AD       Address of a fullword pointer that
*                               contains the address of a 64 bit
*                               field to return the arrival time.
_AGT_END         DS    0C       End of _AGT
*
_ARA             DSECT ,        ARM_REGISTER_APPLICATION Parameter
*                               List
_ARA_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ARA_APP_NAME    DS    AD       Address of a character string that
*                               contains the application name.
_ARA_IN_APP_ID   DS    AD       Address of a 16 byte field that
*                               contains an input application ID.
_ARA_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ARA_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ARA_OUT_APP_ID  DS    AD       Address of a fullword pointer that
*                               contains the address of a 16 byte
*                               field to return the output
*                               application ID.
_ARA_END         DS    0C       End of _ARA
*
_AMR             DSECT ,        ARM_REGISTER_METRIC Parameter
*                               List
_AMR_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AMR_APP_ID      DS    AD       Address of a 16 byte field that
*                               contains the application ID.
_AMR_MET_NAME    DS    AD       Address of a character string that
*                               contains the metric name.
_AMR_MET_FORMAT  DS    AD       Address of a 1 byte field that
*                               contains the metric format.
_AMR_MET_USAGE   DS    AD       Address of a 2 byte field that
*                               contains the metric usage.
_AMR_UNIT        DS    AD       Address of a character string that
*                               contains the units of the metric.
_AMR_IN_MET_ID   DS    AD       Address of a 16 byte field that
```

```
*                               contains an input metric ID.
_AMR_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AMR_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AMR_OUT_MET_ID  DS    AD       Address of a fullword pointer that
*                               contains the address of a 16 byte
*                               field to return the output
*                               metric ID.
_AMR_END         DS    0C       End of _AMR
*
_ART             DSECT ,        ARM_REGISTER_TRANSACTION Parameter
*                               List
_ART_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ART_APP_ID      DS    AD       Address of a 16 byte field that
*                               contains the application ID.
_ART_TRAN_NAME   DS    AD       Address of a character string that
*                               contains the transaction name.
_ART_IN_TRAN_ID  DS    AD       Address of a 16 byte field that
*                               contains an input transaction ID.
_ART_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ART_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ART_OUT_TRAN_ID DS    AD       Address of a fullword pointer that
*                               contains the address of a 16 byte
*                               field to return the output
*                               transaction ID.
_ART_END         DS    0C       End of _ART
*
_ATR             DSECT ,        ARM_REPORT_TRANSACTION Parameter
*                               List
_ATR_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ATR_APP_HDL     DS    AD       Address of a 8 byte field that
*                               contains the application handle.
_ATR_TRAN_ID     DS    AD       Address of a 16 byte field that
*                               contains the transaction ID.
_ATR_TRAN_STA    DS    AD       Address of a 4 byte field that
*                               contains the transaction status.
_ATR_RESP_TIME   DS    AD       Address of a 64 bit field that
*                               contains the response time.
_ATR_STOP_TIME   DS    AD       Address of a 64 bit field that
*                               contains the stop time.
_ATR_PAR_CORR    DS    AD       Address of a data area that
*                               contains the parent correlator.
_ATR_CUR_CORR    DS    AD       Address of a data area that
*                               contains the current correlator.
_ATR_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ATR_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ATR_END         DS    0C       End of _ATR
*
_AAS             DSECT ,        ARM_START_APPLICATION Parameter
*                               List
_AAS_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AAS_APP_ID      DS    AD       Address of a 16 byte field that
*                               contains the application ID.
_AAS_APP_GRP     DS    AD       Address of a character string that
*                               contains the application group
*                               name.
_AAS_APP_INS     DS    AD       Address of a character string that
*                               contains the application
*                               instance name.
```

```
_AAS_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AAS_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AAS_APP_HDL     DS    AD       Address of a fullword pointer that
*                               contains the address of the 8 byte
*                               field to return the application
*                               handle.
_AAS_END         DS    0C       End of _AAS
*
_AST             DSECT ,        ARM_START_TRANSACTION Parameter
*                               List
_AST_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AST_APP_HDL     DS    AD       Address of a 8 byte field that
*                               contains the application handle.
_AST_TRAN_ID     DS    AD       Address of a 16 byte field that
*                               contains the transaction ID.
_AST_PAR_CORR    DS    AD       Address of a data area that
*                               contains the parent correlator.
_AST_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AST_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AST_TRAN_HDL    DS    AD       Address of a fullword pointer that
*                               contains the address of the 8 byte
*                               field to return the transaction
*                               handle.
_AST_CUR_CORR    DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer
*                               to return the current correlator.
_AST_END         DS    0C       End of _AST
*
_APA             DSECT ,        ARM_STOP_APPLICATION Parameter
*                               List
_APA_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_APA_APP_HDL     DS    AD       Address of a 8 byte field that
*                               contains the application handle.
_APA_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_APA_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_APA_END         DS    0C       End of _APA
*
_APT             DSECT ,        ARM_STOP_TRANSACTION Parameter
*                               List
_APT_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_APT_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_APT_TRAN_STA    DS    AD       Address of a 4 byte number that
*                               contains the transaction status.
_APT_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_APT_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_APT_END         DS    0C       End of _APT
*
_AUB             DSECT ,        ARM_UNBIND_THREAD Parameter
*                               List
_AUB_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AUB_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_AUB_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
```

```
_AUB_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AUB_END         DS    0C       End of _AUB
*
_AUT             DSECT ,        ARM_UNBLOCK_TRANSACTION Parameter
*                               List
_AUT_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AUT_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_AUT_BLOCK_HDL   DS    AD       Address of a 8 byte field
*                               that contains the block handle.
_AUT_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AUT_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AUT_END         DS    0C       End of _AUT
*
_AUP             DSECT ,        ARM_UPDATE_TRANSACTION Parameter
*                               List
_AUP_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_AUP_TRAN_HDL    DS    AD       Address of a 8 byte field that
*                               contains the transaction handle.
_AUP_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_AUP_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_AUP_END         DS    0C       End of _AUP
*
_ACC             DSECT ,        EWLM_CLASSIFY_CORRELATOR Parameter
*                               List
_ACC_CONTEXT     DS    AD       Reserved.
*                               Must be zero.
_ACC_APP_HDL     DS    AD       Address of a 8 byte field that
*                               contains the application handle.
_ACC_TRAN_ID     DS    AD       Address of a 16 byte field that
*                               contains the transaction ID.
_ACC_FLAGS       DS    AD       Address of a 4 byte field
*                               that contains flags.
_ACC_BUFFER4     DS    AD       Address of a data area that
*                               contains additional input data.
_ACC_CLASS_CORR  DS    AD       Address of a fullword pointer that
*                               contains the address of the buffer to
*                               return the classify correlator.
_ACC_END         DS    0C       End of _ACC
** BPXYWLM End
```

Appendix D. Callable services examples—AMODE 31

For an example using nonreentrant code, see “Example of nonreentrant entry linkage—AMODE 31” on page 1307. These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry.

The examples are arranged alphabetically and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

Reentrant entry linkage

This entry linkage is reentrant and saves the caller's registers, allocates a save area and dynamic storage, and establishes program and dynamic storage base registers. This entry linkage is paired with the return linkage that is located at the end of the executable program; see “Reentrant return linkage” on page 1212. For an example of nonreentrant entry and return linkage, see “Example of nonreentrant entry linkage—AMODE 31” on page 1307.

```
         TITLE 'Alphabetical syscall of z/OS UNIX callable services'
BPXB1SM1 CSECT ,                    Reentrant entry linkage
BPXB1SM1 AMODE 31
BPXB1SM1 RMODE ANY
         USING *,R15                Program addressability
@ENTRY0  B     @ENTRY1              Branch around program header
         DROP  R15                  R15 not needed for addressability
         DC    C'BPXB1SM1 - Reentrant callable service examples'
         DS    0H                   Ensure half word boundary
@ENTRY1  STM   R14,R12,12(R13)      Save caller's registers
         LR    R2,R13               Hold address of caller's area
         LR    R3,R1                Hold parameter register
         LR    R12,R15              R12 program base register
         LA    R11,2048(,R12)       Second program base register
         LA    R11,2048(,R11)       Second program base register
         LA    R9,2048(,R11)        Third  program base register
         LA    R9,2048(,R9)         Third  program base register
         LA    R4,2048(,R9)         Fourth program base register
         LA    R4,2048(,R4)         Fourth program base register
         LA    R7,2048(,R4)         Fifth  program base register
         LA    R7,2048(,R7)         Fifth  program base register
         USING @ENTRY0,R12,R11,R9,R4,R7 Program addressability
         L     R0,@SIZEDAT          Size this program's getmain area
         GETMAIN RU,LV=(0)          Getmain storage
         LR    R13,R1               R13 -> this program's save area
         LA    R10,2048(,R13)       Second getmain base register
         LA    R10,2048(,R10)       Second getmain base register
         LA    R6,2048(,R10)        Third  getmain base register
         LA    R6,2048(,R6)         Third  getmain base register
         USING @STORE,R13,R10,R6    Getmain addressability
         ST    R2,@BACK             Save caller's save area pointer
         ST    R13,8(,R2)           Give caller our save area
         LR    R1,R3                Restore parameter register
@ENTRY2  EQU   *  *  *  *  *  *  *  End of the entry linkage code
         SPACE ,
PSEUDO   EQU   *                    Dummy label used throughout
```

BPX1ACC (access) example

The following code determines if **/usr/inv/network.t** can be accessed. For the callable service, see "access (BPX1ACC, BPX4ACC) — Determine if a file can be accessed" on page 23. For the data structure, see "BPXYACC — Map flag values for access" on page 945. AMODE 64 callers use "BPX4ACC (access) example" on page 1216.

```
MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
MVC   BUFLENA,=F'18'
XC    ACC(ACC#LENGTH),ACC
MVI   ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK  Read and write access
SPACE ,
CALL  BPX1ACC,              Determine accessibility of a file +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      ACC,                  Input: Access, BPXYACC           +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Set condition code for RETVAL
BZ    PSEUDO                Branch if RETVAL is zero
CLC   RETCODE,=A(EACCES)    Compare RETCODE to EACCES
BE    PSEUDO                Branch if access denied
```

BPX1ACK (auth_check_resource_np) example

The following code determines if user JOEUSER has UPDATE access to the FACILITY class profile TEST.THIS.PROFILE. For the callable service, see "auth_check_resource_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF-protected resource" on page 66. AMODE 64 callers use "BPX4ACK (auth_check_resource_np) example" on page 1216.

```
MVI   CELLUUID,X'00'
MVI   PRINUUID,X'00'
MVC   USERNLEN,=F'7'
MVC   USERNAME(7),=CL7'JOEUSER'
MVC   CLSLEN,=F'8'
MVC   CLS(8),=CL8'FACILITY'
MVC   ENTLEN,=F'17'
MVC   ENT(17),=CL17'TEST.THIS.PROFILE'
SPACE ,
CALL  BPX1ACK,              Determine access to a resource    +
      (CELLUUID,            Input: Cell UUID                  +
      PRINUUID,             Input: Principal UUID             +
      USERNLEN,             Input: Userid length              +
      USERID,               Input: Userid                     +
      CLSLEN,,              Input: Class length               +
      CLS,                  Input: Class                      +
      ENTLEN,               Input: Entity length              +
      ENT,                  Input: Entity                     +
      =A(ACK_UPDATE#),      Input: Access type to check for   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

01

BPX1ACP (accept) example

The following code does an accept to accept a connect request from a client. SOCKDESC was previously set by a call to BPX1SOC. A bind and a listen must also have been previously done. The SOCKADDR was built by the call to BPX1BND. For the callable service, see "accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket" on page 15. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4ACP (accept) example" on page 1217.

```
CALL  BPX1ACP,              Accept a socket connect request   +
      (SOCKDESC,            Input: Socket descriptor          +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr     +
      SOCKADDR,             Input: Sockaddr structure         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
L  R2,RETVAL
ST R2,SOCKDES2              Store the new socket descriptor
```

BPX1AIO (asyncio) example

The following code will accept the next conversation. For the callable service, see "asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets" on page 31. AMODE 64 callers use "BPX4AIO (asyncio) example" on page 1217.

```
XC    AIO(AIO#LENGTH),AIO   Null AIO control block
MVC   AIOCMD,=A(AIO#ACCEPT) Command = Accept
MVC   AIOFD,FILEDESC        File descriptor
MVC   AIONOTIFYTYPE,=AL2(AIO#MVS)  Notify type = MVS
XC    ECB01,ECB01           ECB = 0
LA    R15,ECB01             ECB Address
ST    R15,AIOECBPTR         Null AIO control block
MVC   AIOSOCKADDRLEN,=A(SOCK#LEN)
LA    R15,SOCKADDR          From recvform (see BPX1RFM)
ST    R15,AIOSOCKADDRPTR
SPACE ,
CALL  BPX1AIO,              Asynchronous I/O for Sockets      +
      (=A(AIO#LENGTH),      Input: Time before SIGAIOM        +
      AIO,                  Input: Time before SIGAIOM        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1ALR (alarm) example

The following code schedules an alarm in 5 seconds. For the callable service, see "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29. AMODE 64 callers use "BPX4ALR (alarm) example" on page 1217.

```
MVC   SECONDS,=F'5'
SPACE ,
CALL  BPX1ALR,              Schedule Alarm                    +
      (SECONDS,             Input: Time before SIGALRM        +
      RETVAL),              Return value: 0 or -1             +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1ANR (accept_and_recv) example

The following code accepts a connection and reads the first block of data from a client. The new socket's descriptor, the peer's remote address and the caller's local address are also returned. SOCKDESC was previously set by a call to BPX1SOC. ACPSOCK must be set to -1 and the system will assign a new descriptor for the accepted connection in this parameter. A bind and a listen must also have been previously done. The SOCKADDR was built by the call to BPX1BND. For the callable service, see “accept_and_recv (BPX1ANR, BPX4ANR) — Accept a connection and receive the first block of data” on page 18. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043. AMODE 64 callers use “BPX4ANR (accept_and_recv) example” on page 1218.

```
L     R8,=XL4'FFFFFFFF'     Set ACPSOCK = -1
ST    R8,ACPSOCK
CALL  BPX1ANR,              Accept_and_receive request        +
      (SOCKDESC,            Input: Socket descriptor          +
      ACPSOCK,              Input: -1 Output: accepted soc des+
      SOCK#LEN+SOCK_SUN#LEN, Input/Output: Len of Remote_addr +
      RSOCKADR,             Input: Remote sockaddr structure  +
      SOCK#LEN+SOCK_SUN#LEN, Input/Output: Len of Local_addr  +
      LSOCKADR,             Input: Local sockaddr structure   +
      =A(L'BUFFERA),        Input: Length of the buffer       +
      BUFFERA,              Input/Output: Addr of the buffer  +
      PRIMARYALET,          Input: Alet of the buffer         +
      RETVAL,               Return value: -1 or num bytes recd+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
L     R2,RETVAL
ST    R2,BYTERECD              Store number of bytes received
```

BPX1ASP (aio_suspend) example

The following code will wait up to 10 seconds for one of the events specified in the AIOCB. For the callable service, see “aio_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request” on page 26. AMODE 64 callers use “BPX4ASP (aio_suspend) example” on page 1218.

```
LA    R15,AIO
ST    R15,ARGSLST
MVC   ARGCNT,=F'1'
MVC   SECONDS,=F'10'
XC    NANOSECONDS,NANOSECONDS
SPACE ,
CALL  BPX1ASP,              Suspend for an aio request        +
      (ARGSLST,             Input: List of pointers to AIOCBs +
      ARGCNT,               Input: Count of pointers in list  +
      SECONDS,              Input: Seconds to wait            +
      NANOSECONDS,          Input: Nanoseconds to wait        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1ATM (attach_execmvs) example

The following code invokes program APPL92 on a subtask and as a child process of the caller, passing the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine that is associated with program APPL92. For the callable service, see "attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program" on page 59. AMODE 64 callers use "BPX4ATM (attach_execmvs) example" on page 1218.

```
MVC   PGMNAMEL,=F'6'
MVC   PGMNAME(06),=CL6'APPL92'
MVC   BUFLENA,=F'24'
MVC   BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE ,
CALL  BPX1ATM,              Invoke a MVS program              +
      (PGMNAMEL,            Input: Length of program name     +
      PGMNAME,              Input: Program name               +
      BUFLENA,              Input: Length of program argument +
      BUFFERA,              Input: Program argument           +
      =A(0),                Input: Exit routine address or 0  +
      =A(0),                Input: Exit Parm list address or 0+
      RETVAL,               Return value: Child PID Or -1     +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1ATX (attach_exec) example

The program ictasma located at **ict/bin** gets control on a subtask and as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see "attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program" on page 50. AMODE 64 callers use "BPX4ATX (attach_exec) example" on page 1219.

```
        MVC   BUFLENA,=F'16'
        MVC   BUFFERA(16),=C'/ict/bin/ictasma'
        MVC   ARGCNT,=F'3'
*                                 First
        LA    R15,=F'4'               Length
        ST    R15,ARGLLST+00          Length parm list
        LA    R15,=CL4'WK18'          Argument
        ST    R15,ARGSLST+00          Argument address parm list
*                                 Second
        LA    R15,=F'7'               Length
        ST    R15,ARGLLST+04          Length parm list
        LA    R15,=CL7'DEPT37A'       Argument
        ST    R15,ARGSLST+04          Argument address parm list
*                                 Third
        LA    R15,=F'22'              Length
        ST    R15,ARGLLST+08          Length parm list
        LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
        ST    R15,ARGSLST+08          Argument address parm list
*
        MVC   ENVCNT,=F'0'          Number of env. data items passed
        MVC   ENVLENS,=F'0'         Addr of env. data length list
        MVC   ENVPARMS,=F'0'        Add of env. data
*
        MVC   EXITRTNA,=V(EXITRTN)  ->exit routine
*       MVC   EXITPLA,=A(exit paramter list as expected by EXITRTN)
        SPACE ,
        CALL  BPX1ATX,                                          +
              (BUFLENA,             Input: Pathname length      +
```

```
     BUFFERA,             Input: Pathname                  +
     ARGCNT,              Input: Argument count            +
     ARGLLST,             Input: Argument length list      +
     ARGSLST,             Input: Argument address list     +
     ENVCNT,              Input: Environment count         +
     ENVLENS,             Input: Environment length list   +
     ENVPARMS,            Input: Environment address list  +
     EXITRTNA,            Input: Exit routine address or 0 +
     EXITPLA,             Input: Exit Parm list address or 0+
     RETVAL,              Return value: Child PID or -1    +
     RETCODE,             Return code                      +
     RSNCODE),            Reason code                      +
     VL,MF=(E,PLIST)      ---------------------------------
```

BPX1BND (bind) example

The following code does a bind to associate a name with a socket. SOCKDESC was previously set by a call to BPX1SOC. For the callable service, see "bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor" on page 71. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4BND (bind) example" on page 1219.

```
SPACE ,
MVI   SOCK_LEN,12          Store the length of the address
MVI   SOCK_FAMILY,AF_UNIX  Set the domain to AF_UNIX
MVC   SOCK_SUN_NAME(12),=CL12'/tmp/socket1'  Set the name
CALL  BPX1BND,             Bind a name to a socket          +
      (SOCKDESC,           Input: Socket Descriptor         +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr   +
      SOCKADDR,            Input: Sockaddr structure        +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1BAS (bind with source address selection) example

The following code does a bind to associate the best source address for the provided destination IP address with a socket. SOCKDESC was previously set by a call to BPX1SOC. For the callable service, see "bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address" on page 73. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043.

```
SPACE ,
                           Store the length of the address
MVI SOCK_LEN,=A(SOCK#LEN+SOCK_SIN6#LEN)
MVI SOCK_FAMILY,AF_INET6   Set the domain to AF_INET6
MVC SOCK_SIN6_ADDR,=XL16'00A100B200C300D400E500F61234ABCD'
CALL BPX1BAS,              Bind with source address selection+
     (SOCKDESC,            Input: Socket Descriptor         +
     =A(SOCK#LEN+SOCK_SIN6#LEN), Input:Length - Sockaddr    +
     SOCKADDR,             Input: Sockaddr structure        +
     RETVAL,               Return value: 0 or -1            +
     RETCODE,              Return code                      +
     RSNCODE),             Reason code                      +
     VL,MF=(E,PLIST)       ---------------------------------
```

BPXICCA (cond_cancel) example

The following code demonstrates how to cancel a program's interest in events that were selected by a call to the cond_setup service. For the callable service, see "cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events" on page 107. AMODE 64 callers use "BPX4CCA (cond_cancel) example" on page 1220.

```
        CALL  BPX1CCA,              Cancel cond_setup                +
              (RETVAL,              Return value: 0 or -1            +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
*  The return value (RETVAL) does not matter.  When your program
*  receives control following the call to cond_cancel, it is no
*  longer eligible to receive event notifications via cond_post.
```

BPX1CCS (__console()) example

The following code sends a message to the console. For the callable service, see "__console() (BPX1CCS, BPX4CCS) — Communicate with console (modify/stop/WTO/DOM)" on page 124. For the data structure, see "BPXYCCA — Map input/output structure for __console()" on page 950. AMODE 64 callers use "BPX4CCS (__console()) example" on page 1220.

```
        CALL  BPX1CCS,              Send msg to console              +
              (MSGATTRLEN,          Input: BPXYCCA length            +
              MSGATTR,              Input: BPXYCCA                   +
              MODSTRINGPTR,         Output: Modify msg from console  +
              MODIFYSTGLEN,         Output: Length of modify msg     +
              CONMSGTYPE,           Output: Console msg type         +
              RETVAL,               Return value: 0 or -1            +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CHA (chaudit) example

The following code changes the audit flags for the file identified by pathname. For the callable service, see "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84. For the data structure, see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. AMODE 64 callers use "BPX4CHA (chaudit) example" on page 1221.

```
        MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
        MVC   BUFLENA,=F'18'
        MVI   AUDTREADACCESS,AUDTREADFAIL
        MVI   AUDTWRITEACCESS,AUDTWRITEFAIL
        MVI   AUDTEXECACCESS,AUDTEXECFAIL
        MVI   AUDTRSRV,0
        SPACE ,
        CALL  BPX1CHA,              Change audit                     +
              (BUFLENA,             Input: Pathname length           +
              BUFFERA,              Input: Pathname                  +
              AUDT,                 Input: Audit flags, BPXYAUDT     +
              =F'0',                Input: 0 user, 1 security auditor +
              RETVAL,               Return value: 0 or -1            +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CHD (chdir) example

The following code changes the working directory for the task. For the callable service, see "chdir (BPX1CHD, BPX4CHD) — Change the working directory" on page 88. AMODE 64 callers use "BPX4CHD (chdir) example" on page 1221.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX1CHD,              Change working directory          +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1CHM (chmod) example

The following code changes the file mode for the file identified by pathname. For the callable service, see "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90. For the data structure, see "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 64 callers use "BPX4CHM (chmod) example" on page 1221.

```
MVC   BUFFERA(26),=CL26'newprogs/path/eightfold.c'
MVC   BUFLENA,=F'26'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       All read and write
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX1CHM,              Change File Modes                 +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      S_MODE,               Input: Mode, mapped by BPXYMODE   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1CHO (chown) example

The following code changes the owner of **/somedir/somefile.c** from the current owner to that specified by USERID and GROUPID. For the callable service, see "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93. AMODE 64 callers use "BPX4CHO (chown) example" on page 1222.

```
  MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
  MVC   BUFLENA,=F'20'
MVC   USERID,..             New owner UID from stat           07
MVC   GROUPID,..            New owner GID from stat           07
  SPACE ,
  CALL  BPX1CHO,              Change owner and group of a file  +
        (BUFLENA,             Input: Pathname length            +
        BUFFERA,              Input: Pathname                   +
        USERID,               Input: New owner UID              +
        GROUPID,              Input: New owner GID              +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1CHP (chpriority) example

The following code changes the CPU priority based on the input which, who, and priority type values. The which value used is PRIO_PROCESS, indicating that the priority will be set by process ID. The who value used is 7, to set the priority for process ID 7. The priority type is CPRIO_ABSOLUTE, indicating that the priority will be set to the value specified, 1. For the callable service, see "chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process" on page 97. AMODE 64 callers use "BPX4CHP (chpriority) example" on page 1222.

```
MVC   PROCID,=XL4'00000007' Process ID to change priority for
MVC   PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL  BPX1CHP,              Change priority value             +
      (=A(PRIO_PROCESS),    Input: Set by Process ID          +
      PROCID,               Input: PID to set priority for    +
      =A(CPRIO_ABSOLUTE),   Input: Change by absolute value   +
      PRIORITY,             Input: Priority value to change to+
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1CHR (chattr) example

The following code changes the attributes of **/somedir/somefile.c**. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948 and "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 64 callers use "BPX4CHR (chattr) example" on page 1223.

```
MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
MVC   BUFLENA,=F'20'
MVC   ATTID,=CL4'ATT '      Eye Catcher
MVC   ATTVERSION,=AL2(ATT#VER)  version
XC    S_MODE,S_MODE         Clear mode
MVI   S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC   ATTMODE,S_MODE        Move mode data to attribute       +
      structure
MVC   ATTUID,=F'7'          Specify new UID
MVC   ATTGID,=F'77'         Specify new GID
OI    ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG                     +
      Flag Mode, UID and GID changes
OI    ATTSETFLAGS2,ATTCTIMETOD                                +
      Set change time to current time
SPACE ,
CALL  BPX1CHR,              Change file attributes            +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      =A(ATT#LENGTH),       Input: BPXYATT length             +
      ATT,                  Input/output: BPXYATT             +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1CLD (closedir) example

The following code closes the directory identified by FILEDESC. For the callable service, see "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105. AMODE 64 callers use "BPX4CLD (closedir) example" on page 1223.

```
MVC   FILEDESC,..           Directory descriptor from opendir  08
  SPACE ,
  CALL  BPX1CLD,              Close a directory                +
        (FILEDESC,            Input: Directory file descriptor +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CLO (close) example

The following code closes the standard input file. For the callable service, see "close (BPX1CLO, BPX4CLO) — Close a file" on page 103. AMODE 64 callers use "BPX4CLO (close) example" on page 1223.

```
  CALL  BPX1CLO,              Close a file                     +
        (=A(STDIN_FILENO),    Input: File descriptor           +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CON (connect) example

The following code connects to a socket. SOCKDESC was returned by a previous call to BPX1SOC, and SOCKADDR contains the name of the peer, possibly obtained by a call to BPX1GNM. For the callable service, see "connect (BPX1CON, BPX4CON) — Establish a connection between two sockets" on page 121. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4CON (connect) example" on page 1224.

```
  SPACE ,
  MVI   SOCK_LEN,12           Store the length of the address
  MVI   SOCK_FAMILY,AF_UNIX   Set the domain to AF_UNIX
  MVC   SOCK_SUN_NAME(12),=CL12'/tmp/socket1'  Set the name
  CALL  BPX1CON,              Connect to a socket              +
        (SOCKDESC,            Input: Socket Descriptor         +
        SOCK#LEN+SOCK_SUN#LEN, Input: Length - Sockaddr        +
        SOCKADDR,             Input: Sockaddr structure        +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CPO (cond_post) example

The following code demonstrates how to send an event notification to a thread waiting in the cond_wait or cond_timed_wait service. For the callable service, see "cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event" on page 109. AMODE 64 callers use "BPX4CPO (cond_post) example" on page 1224. The following code notifies thread (THID) that a CW_CONDVAR event has occurred.

```
  CALL  BPX1CPO,              Send condition event notification +
        (THID,                Input: Thread ID of target pgm    +
```

```
      =A(CW_CONDVAR),       Input: Event            in BPXYCW +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CRT (chroot) example

The following code changes the root directory for the task. For the callable service, see "chroot (BPX1CRT, BPX4CRT) — Change the root directory" on page 100. AMODE 64 callers use "BPX4CRT (chroot) example" on page 1224.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX1CRT,              Change root directory             +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CSE (cond_setup) example

The following code sets up the invoker to suspend processing until any of the specified events (CW_INTRPT or CW_CONDVAR) occurs. The BPX1CTW (cond_timed_wait) or BPX1CWA (cond_wait) service is used to actually suspend processing. For the callable service, see "cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications" on page 111. AMODE 64 callers use "BPX4CSE (cond_setup) example" on page 1225.

```
MVC   EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX1CSE,              Condition setup                   +
      (EVENTLIST,           Input: Event list          BPXYCW +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CTW (cond_timed_wait) example

The following code suspends the calling thread until a signal arrives (CW_INTRPT), or else 2.5 seconds have elapsed. For the callable service, see "cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event" on page 114. AMODE 64 callers use "BPX4CTW (cond_timed_wait) example" on page 1225.

```
MVC   EVENTLIST,=A(CW_INTRPT)          Signals
CALL  BPX1CTW,              Wait for condition events         +
      (=A(2),               Input: Number of seconds          +
      =A(500000000),        Input: Number of nanoseconds      +
      EVENTLIST,            Input: Event list          BPXYCW +
      SECONDS,              Output: Unexpired seconds         +
      NANOSECONDS,          Output: Unexpired nanoseconds     +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1CWA (cond_wait) example

The following code suspends the calling thread until either of two events occurs: the arrival of a signal (CW_INTRPT) or some other thread using the cond_post service to send this thread a CW_CONDVAR notification. For the callable service, see "cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event" on page 118. AMODE 64 callers use "BPX4CWA (cond_wait) example" on page 1225.

```
MVC   EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX1CWA,              Wait for condition events         +
      (EVENTLIST,           Input: Event list          BPXYCW +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1DEL (deleteHFS) example

The program ictasma located at **ict/bin** is loaded into storage using BPX1LOD, branched to and then deleted from storage using BPX1DEL. For the callable service, see "deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage" on page 130. AMODE 64 callers use "BPX4DEL (deleteHFS) example" on page 1225.

```
MVC   BUFLENA,=F'16'
MVC   BUFFERA(16),=C'/ict/bin/ictasma'
MVC   OPTIONS,=A(0)
MVC   LIBPTHLN,=A(0)
SPACE ,
CALL  BPX1LOD,              Load Program                      +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      OPTIONS,              Input: Options                    +
      LIBPTHLN,             Input: Library Path Length        +
      LIBPATH,              Input: Library Path               +
      EPADDR,               Return value: -1 or entry pt addr +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
L     R15,EPADDR            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
SPACE ,
L     R15,EPADDR
BALR  R14,R15               Branch to loaded program
SPACE ,
CALL  BPX1DEL,              Delete program                    +
      (EPADDR,              Input: Entry point address        +
      RETVAL,               Return value: -1 or 0             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1ENV (oe_env_np) example

The following code enables interruption of threads waiting in MVS ENQs in the caller's process. For the callable service, see "oe_env_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute" on page 435. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4ENV (oe_env_np) example" on page 1226.

```
         LA    R15,=F'1'
         ST    R15,INARG
         LA    R15,INARG
         ST    R15,INARGLIST
         LA    R15,INARGLIST
         ST    R15,INARGLISTPTR
         SPACE ,
         CALL  BPX1ENV,              oe_env_np                       +
               (=A(ENQWAIT_PROCESS), Input: Function_code     BPXYCONS +
               =A(1),                Input: InArgCount                +
               INARGLISTPTR,         Input: InArgListPtr              +
               =A(0),                Input: OutArgCount               +
               =A(0),                Input: OutArgListPtr             +
               RETVAL,               Return value: 0 or -1            +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1EXC (exec) example

The program ictasma located at **ict/bin** gets control and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132. AMODE 64 callers use "BPX4EXC (exec) example" on page 1227.

```
         MVC   BUFLENA,=F'16'
         MVC   BUFFERA(16),=C'/ict/bin/ictasma'
         MVC   ARGCNT,=F'3'
*                                First
         LA    R15,=F'4'              Length
         ST    R15,ARGLLST+00         Length parm list
         LA    R15,=CL4'WK18'         Argument
         ST    R15,ARGSLST+00         Argument address parm list
*                                Second
         LA    R15,=F'7'              Length
         ST    R15,ARGLLST+04         Length parm list
         LA    R15,=CL7'DEPT37A'      Argument
         ST    R15,ARGSLST+04         Argument address parm list
*                                Third
         LA    R15,=F'22'             Length
         ST    R15,ARGLLST+08         Length parm list
         LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
         ST    R15,ARGSLST+08         Argument address parm list
*
         MVC   ENVCNT,=F'0'       Number of env. data items passed
         MVC   ENVLENS,=F'0'      Addr of env. data length list
         MVC   ENVPARMS,=F'0'     Add of env. data
*
         MVC   EXITRTNA,=V(EXITRTN)  ->exit routine
*        MVC   EXITPLA,=A(exit parameter list as expected by EXITRTN)
         SPACE ,
         CALL  BPX1EXC,                                           +
               (BUFLENA,          Input: Pathname length         +
               BUFFERA,           Input: Pathname                +
               ARGCNT,            Input: Argument count          +
               ARGLLST,           Input: Argument length list    +
               ARGSLST,           Input: Argument address list   +
               ENVCNT,            Input: Environment count       +
               ENVLENS,           Input: Environment length list +
               ENVPARMS,          Input: Environment address list+
               EXITRTNA,          Input: Exit routine address or 0 +
               EXITPLA,           Input: Exit Parm list address or 0+
               RETVAL,            Return value: -1 or not return +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1EXI (_exit) example

The following code ends the program and returns an exit code of 44 to the waiting parent process. For the callable service, see "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150. AMODE 64 callers use "BPX4EXI (_exit) example" on page 1227.

```
XC    WAST(WAST#LENGTH),WAST
MVI   WASTEXITCODE,44      User defined exit code
SPACE
CALL  BPX1EXI,             End a process                    +
      (WAST),              Input: Status field              +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1EXM (execmvs) example

The following code invokes program APPL92 and passes the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine associated with program APPL92. For the callable service, see "execmvs (BPX1EXM, BPX4EXM) — Run an MVS program" on page 144. AMODE 64 callers use "BPX4EXM (execmvs) example" on page 1228.

```
MVC   PGMNAMEL,=F'6'
MVC   PGMNAME(06),=CL6'APPL92'
MVC   BUFLENA,=F'24'
MVC   BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE ,
CALL  BPX1EXM,             Invoke a MVS program             +
      (PGMNAMEL,           Input: Length of program name    +
      PGMNAME,             Input: Program name              +
      BUFLENA,             Input: Length of program argument +
      BUFFERA,             Input: Program argument          +
      =A(0),               Input: Exit routine address or 0  +
      =A(0),               Input: Exit Parm list address or 0+
      RETVAL,              Return value: -1 or not return   +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1EXT (extlink_np) example

The following code creates an external link to data set **MY.DATASET** for pathname **/mvs/mydataset**. For the callable service, see "extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link" on page 153. AMODE 64 callers use "BPX4EXT (extlink_np) example" on page 1228.

```
MVC   BUFFERA(10),=CL10'MY.DATASET'
MVC   BUFLENA,=F'10'
MVC   BUFFERB(14),=CL14'/mvs/mydataset'
MVC   BUFLENB,=F'14'
SPACE ,
CALL  BPX1EXT,             Create external link to name     +
      (BUFLENA,            Input: External name length      +
      BUFFERA,             Input: External name             +
      BUFLENB,             Input: Link name length          +
      BUFFERB,             Input: Link name                 +
      RETVAL,              Return value: 0 or -1            +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1FAI (freeaddrinfo) example

The following code frees the Addr_Info structure(s) that were obtained by the getaddrinfo callable service. For the callable service, see "freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures" on page 194. AMODE 64 callers use "BPX4FAI (freeaddrinfo) example" on page 1228.

```
SPACE ,
CALL  BPX1FAI,             Free Addr_Info                   +
      (ADDR_INFO_PTR,      Input: -> Addr_Info structure    +
      RETVAL,              Return code                      +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1FCA (fchaudit) example

The following code changes the audit for the standard input file to ReadFail, WriteFail and ExecFail. For the callable service, see "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164. For the data structure, see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. AMODE 64 callers use "BPX4FCA (fchaudit) example" on page 1229.

```
MVI   AUDTREADACCESS,AUDTREADFAIL
MVI   AUDTWRITEACCESS,AUDTWRITEFAIL
MVI   AUDTEXECACCESS,AUDTEXECFAIL
MVI   AUDTRSRV,X'00'
SPACE ,
CALL  BPX1FCA,             Change audit                     +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      AUDT,                Input: Audit flags, BPXYAUDT     +
      =A(0),               Input: 0 user, 1 security auditor +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1FCD (fchdir) example

The following code changes the working directory for the task to the directory identified by FILEDESC. For the callable service, see "fchdir (BPX1FCD, BPX4FCD) — Change the working directory" on page 167. AMODE 64 callers use "BPX4FCD (fchdir) example" on page 1229.

```
MVC   FILEDESC,..          Directory descriptor from opendir  14
  SPACE ,
  CALL  BPX1FCD,             Change working directory         +
        (FILEDESC,           Input: Directory file descriptor  +
        RETVAL,              Return value: 0 or -1            +
        RETCODE,             Return code                      +
        RSNCODE),            Reason code                      +
        VL,MF=(E,PLIST)      ---------------------------------
```

BPX1FCM (fchmod) example

The following code changes the permissions for the standard input file. For the callable service, see "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169. For the data structure, see "BPXYMODE — Map the mode constants of the file services" on page 996 and "BPXYFTYP — File type definitions" on page 967. AMODE 64 callers use "BPX4FCM (fchmod) example" on page 1229.

```
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       All permissions
MVI   S_MODE3,S_IRWXU2+S_IRWXG+S_IRWXO
SPACE ,
CALL  BPX1FCM,              Change file modes                +
      (=A(STDIN_FILENO),    Input: File descriptor           +
      S_MODE,               Input: Mode, BPXYMODE, BPXYFTYP  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1FCO (fchown) example

The following code changes the owner and group for the standard input file. For the callable service, see "fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor" on page 171. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4FCO (fchown) example" on page 1230.

```
MVC   GROUPID,..            Group ID                          15
MVC   USERID,..             User ID                           15
  SPACE ,
  CALL  BPX1FCO,              Change the owner and group of file+
        (=A(STDIN_FILENO),    Input: File descriptor           +
        USERID,               Input: New user ID for file      +
        GROUPID,              Input: New group ID for file     +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1FCR (fchattr) example

The following code changes the attributes of the standard input file. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see "fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor" on page 156. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948 and "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 64 callers use "BPX4FCR (fchattr) example" on page 1230.

```
MVC   ATTID,=CL4'ATT '      Eye Catcher
MVC   ATTVERSION,=AL2(ATT#VER)  version
XC    S_MODE,S_MODE         Clear mode
MVI   S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC   ATTMODE,S_MODE        Move mode data to attribute      +
                            structure
MVC   ATTUID,=F'7'          Specify new UID
```

```
         MVC   ATTGID,=F'77'         Specify new GID
         OI    ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG                    +
                                     Flag Mode, UID and GID changes
         OI    ATTSETFLAGS2,ATTCTIMETOD                               +
                                     Set change time to current time
         SPACE ,
         CALL  BPX1FCR,              Change file attributes           +
               (=A(STDIN_FILENO),    Input: File descriptor           +
               =A(ATT#LENGTH),       Input: BPXYATT length            +
               ATT,                  Input/output: BPXYATT            +
               RETVAL,               Return value: 0 or -1            +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1FCT (fcntl) example

The code for the first example duplicates the standard error file descriptor to a file descriptor greater than or equal to FILEDES2.

The code for the second example sets a shared byte range lock. For the callable service, see "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174. For the data structure, see "BPXYFCTL — Command values and flags for fcntl" on page 966, "BPXYBRLK — Map byte range lock request for fcntl" on page 950, and "BPXYOPNF — Map flag values for open" on page 1004. AMODE 64 callers use "BPX4FCT (fcntl) example" on page 1230.

```
* for 2nd parm F_DUPFD, F_DUPFD2           3rd parm file desc no..
* for 2nd parm F_GETFD, F_GETFL            3rd parm 0
* for 2nd parm F_SETFD                     3rd parm BPXYFCTL
* for 2nd parm F_GETLK, F_SETLK, F_SETLKW  3rd parm BPXYBRLK
* for 2nd parm F_SETFL                     3rd parm BPXYOPNF
         SPACE ,
* Example 1 - duplicate file descriptor
         MVC   FILEDES2,=F'20'       Get free file descriptor >= 20
         SPACE ,
         CALL  BPX1FCT,              General purpose file control     +
               (=A(STDERR_FILENO),   Input: File descriptor           +
               =A(F_DUPFD),          Input: Action, BPXYFCTL          +
               FILEDES2,             Input: Argument #/0/FCTL/BRLK/OPNF+
               RETVAL,               Return value: 0, -1 or action    +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               VL,MF=(E,PLIST)       ---------------------------------
         SPACE ,
* Example 2 - duplicate file descriptor
         MVC   FILEDES2,=F'20'       Get next higher file descriptor
         LA    R15,BRLK
         ST    R15,BRLKA
         XC    BRLK(BRLK#LENGTH),BRLK      Null out BRLK
         MVC   L_TYPE,=AL2(F_RDLCK)        Lock type = shared
         MVC   L_WHENCE,=AL2(SEEK_CUR)     Whence = from current cursor
         SPACE ,
         CALL  BPX1FCT,              General purpose file control     +
               (=A(STDERR_FILENO),   Input: File descriptor           +
               =A(F_SETLK),          Input: Action, BPXYFCTL          +
               BRLKA,                Input: Argument #/0/FCTL/BRLK/OPNF+
               RETVAL,               Return value: 0, -1 or action    +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1FPC (fpathconf) example

The following code obtains the configurable option associated with the pipe buffer. For the callable service, see “fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor” on page 191. For the data structure, see “BPXYPCF — Command values for pathconf and pathconf” on page 1005. AMODE 64 callers use “BPX4FPC (fpathconf) example” on page 1231.

```
MVC   FILEDESC,..           From opendir                       16
  SPACE ,
  CALL  BPX1FPC,              Get configurable pathname variable+
        (FILEDESC,            Input: Directory file descriptor  +
        =A(PC_PIPE_BUF),      Input: Configurables      BPXYPCF +
        RETVAL,               Return value: 0, -1 or variable   +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1FRK (fork) example

The following code forks a new process. The next sequential instruction gets control from both the parent process (RETVAL=child process ID) and from the child process (RETVAL=0). If RETVAL=-1, the fork failed. For the callable service, see “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185. AMODE 64 callers use “BPX4FRK (fork) example” on page 1232.

```
  CALL  BPX1FRK,              Create a new process (fork)       +
        (RETVAL,              Return value: -1, 0, child's PID  +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1FST (fstat) example

The following code gets the file status for the file opened as FILEDESC. For the callable service, see “fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor” on page 196. For the data structure, see “BPXYSTAT — Map the response structure for stat” on page 1057. AMODE 64 callers use “BPX4FST (fstat) example” on page 1232.

```
MVC   FILEDESC,..           File descriptor from open          17
  SPACE ,
  CALL  BPX1FST,              Get file status of file descriptor+
        (FILEDESC,            Input: File descriptor            +
        STATL,                Input: Length of buffer needed    +
        STAT,                 Buffer, mapped by BPXYSTAT        +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1FSY (fsync) example

The following code writes file descriptor changes to permanent storage. For the callable service, see “fsync (BPX1FSY, BPX4FSY) — Write changes to permanent storage” on page 201. AMODE 64 callers use “BPX4FSY (fsync) example” on page 1232.

```
MVC   FILEDESC,..           File descriptor from open          17
  SPACE ,
  CALL  BPX1FSY,              Write changes to permanent storage+
        (FILEDESC,            Input: File descriptor            +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1FTR (ftruncate) example

The following code truncates the file described by FILEDESC after 512 bytes. For the callable service, see "ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file" on page 203. AMODE 64 callers use "BPX4FTR (ftruncate) example" on page 1232.

```
MVC   FILEDESC,..           File descriptor from open          17
  MVC   NEWLEN(8),=FL8'512'
  SPACE ,
  CALL  BPX1FTR,              Truncate a file                   +
        (FILEDESC,            Input: File descriptor            +
        NEWLEN,               Input: Length to keep             +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1FTV (fstatvfs) example

The following code obtains information about the file system containing the file identified by FILEDESC. For the callable service, see "fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status" on page 199. For the data structure, see "BPXYSSTF — Map response structure for file system status" on page 1055. AMODE 64 callers use "BPX4FTV (fstatvfs) example" on page 1233.

```
MVC   FILEDESC,..           File descriptor from open          18
  SPACE ,
  CALL  BPX1FTV,              Get file system status            +
        (FILEDESC,            Input: File descriptor            +
        SSTFL,                Input: Length of BPXYSSTF         +
        SSTF,                 Buffer, BPXYSSTF                  +
        RETVAL,               Return value: -1 or length status +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1GAI (getaddrinfo) example

The following code returns the IP address and other associated information for the specified node name. For the callable service, see "getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location" on page 205. AMODE 64 callers use "BPX4GAI (getaddrinfo) example" on page 1233.

```
  SPACE ,
  CALL  BPX1GAI,              Get Addr_info                     +
        (NODE_NAME,           Input: Name of Host being queried +
        NODE_NAME_LENGTH,     Input: Length of host name        +
        SERVICE_NAME,         Input: Service name being queried +
        SERVICE_NAME_LENGTH,  Input: Length of service name     +
```

```
      HINTS_PTR,           Input: Ptr to Addr_Info Structure +
      RESULTS_PTR,         Output:Ptr to Addr_Info Structure +
      CANONICAL_LENGTH,    Output: Length canonical name     +
      RETVAL,              Return code                       +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GCL (getclientid) example

The following code obtains the clientid information for caller. This information is used on givesocket (BPX1GIV) and takesocket (BPX1TAK) services. For the callable service, see "getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier" on page 213. For the data structure, see "BPXYCID — Map the returning structure for getclientid()" on page 951. AMODE 64 callers use "BPX4GCL (getclientid) example" on page 1233.

```
CALL  BPX1GCL,             get clientid information          +
      (=F'2',              Input: Function code of 2         +
      =A(AF_INET),         Input: Domain of AF_INET          +
      CID,                 Output: Clientid information      +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GCW (getcwd) example

The following code gets the working directory for the caller. For the callable service, see "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215. AMODE 64 callers use "BPX4GCW (getcwd) example" on page 1234.

```
MVC   BUFLENA,=F'1024'     Max directory name return area
SPACE ,
CALL  BPX1GCW,             Get working directory name        +
      (BUFLENA,            Input: Length directory work area +
      BUFFERA,             Buffer                            +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GEG (getegid) example

The following code gets the effective group ID of the caller. For the callable service, see "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217. AMODE 64 callers use "BPX4GEG (getegid) example" on page 1234.

```
CALL  BPX1GEG,             Get the effective group ID        +
      (RETVAL),            Return value: effective group ID  +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GEP (getpgid) example

The following code returns the process group ID for the process identified by the input process ID. The process ID value is set to 1. For the callable service, see "getpgid (BPX1GEP, BPX4GEP) — Get the process group ID" on page 250. AMODE 64 callers use "BPX4GEP (getpgid) example" on page 1234.

```
MVC   PROCID,=XL4'00000001'  Value of process ID
SPACE ,
CALL  BPX1GEP,              Get process group ID             +
      (PROCID,              Input: Process ID                +
      RETVAL,               Return value: process group ID   +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1GES (getsid) example

The following code returns the process group ID for the session leader of the process identified by the input process ID. The process ID value is set to 1. For the callable service, see "getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader" on page 270. AMODE 64 callers use "BPX4GES (getsid) example" on page 1234.

```
MVC   PROCID,=XL4'00000000'  Value of process ID
SPACE ,
CALL  BPX1GES,              Get group ID of session leader   +
      (PROCID,              Input: Process ID                +
      RETVAL,               Return value: process group ID   +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1GET (w_getipc) example

The following code retrieves information on the first semaphore defined to the system to which the caller has read access. For the callable service, see "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890. For the data structure, see "BPXYIPCQ — Map w_getipc structure" on page 987. AMODE 64 callers use "BPX4GET (w_getipc) example" on page 1235.

```
XC    TOKEN,TOKEN           Zero, token for 1st member
LA    R5,BUFFERA            Area for query IPC return data
ST    R5,BUFA               R5 -> IPCQ
SPACE ,
CALL  BPX1GET,              Interprocess Communications      +
      (TOKEN,               Input: member token              +
      BUFA,                 Input: ->IPCQ            BPXYIPCQ+
      =A(IPCQ#LENGTH),      Input: Length of IPCQ    BPXYIPCQ+
      =A(IPCQ#SEM),         Input: Request           BPXYIPCQ+
      RETVAL,               Return value: 0, -1 or value     +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
SPACE ,
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
LTR   R15,R15               Test for 0 return
BZ    PSEUDO                Branch on end of file
ST    R15,TOKEN             Save token for next w_semipc
```

BPX1GEU (geteuid) example

The following code gets the effective user ID of the caller. For the callable service, see “geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID” on page 219. AMODE 64 callers use “BPX4GEU (geteuid) example” on page 1235.

```
         CALL  BPX1GEU,             Get the effective user ID        +
               (RETVAL),            Return value: effective user ID  +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1GGE (getgrent) example

The following code accesses the group database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each group entry in the database. For the callable service, see “getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database” on page 221. For the data structure, see “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 969. AMODE 64 callers use “BPX4GGE (getgrent) example” on page 1235.

```
GGELOOP  DS    0H
         CALL  BPX1GGE,             Access the group database        +
               (RETVAL,             Return value: 0 or ->BPXYGIDS    +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      ---------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    CHKGGERR             Error or end of file
         USING GIDS,R8
*        access the group structure
         DROP  R8
         B     GGELOOP              Check next group entry
CHKGGERR DS    0H
         ICM   R8,B'1111',RETCODE
         BZ    GGEEOF               End of file
*        handle error as needed
GGEEOF   DS    0H
```

BPX1GGI (getgrgid) example

The following code accesses the group database by the ID of the caller and returns a structure identifying the groups by ID. The group ID value is set to 5. For the callable service, see “getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID” on page 223. For the data structure, see “BPXYGIDS — Map data returned for getgrnam and getgrpid” on page 969. AMODE 64 callers use “BPX4GGI (getgrgid) example” on page 1236.

```
         MVC   GROUPID,=XL4'00000005'  Value of group ID
         SPACE ,
         CALL  BPX1GGI,             Access the group database        +
               (GROUPID,            Input: Group ID                  +
               RETVAL,              Return value: 0 or ->BPXYGIDS    +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      ---------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    NOGIDS
         USING GIDS,R8
*        access the group structure
         DROP  R8
NOGIDS   EQU   *
```

BPX1GGN (getgrnam) example

The following code accesses the group database by the name of the caller and returns a structure identifying the groups by ID. For the callable service, see "getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name" on page 226. For the data structure, see "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. AMODE 64 callers use "BPX4GGN (getgrnam) example" on page 1236.

```
MVC   GRNAMELN,=F'7'
MVC   GRPGMNAME(7),=CL7'EXTSERV'
SPACE ,
CALL  BPX1GGN,              Access the group database         +
      (GRNAMELN,            Input: Length of group name       +
      GRPGMNAME,            Input: Name of group              +
      RETVAL,               Return value: 0 or ->BPXYGIDS     +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1GGR (getgroups) example

The following code provides the caller with a list of supplementary group IDs. The code sets BUFW size to 256. The actual BUFW size is determined from the previous BPX1GGR RETVAL when BUFW was 0. For the callable service, see "getgroups (BPX1GGR, BPX4GGR) — Get a list of supplementary group IDs" on page 229. AMODE 64 callers use "BPX4GGR (getgroups) example" on page 1237.

```
*       MVC   BUFW,=XL4'00000256'   Value of buffer BUFW
        LA    R15,BUFFERA           Space for BUFW words
        ST    R15,BUFA              ->Array for group IDs
        SPACE ,
        CALL  BPX1GGR,              Get list of supplementary grp IDs +
              (BUFW,                Input: Group ID list size         +
              BUFA,                 ->Buffer for Group ID list address+
              RETVAL,               Return value: -1, 0, ID count     +
              RETCODE,              Return code                       +
              RSNCODE),             Reason code                       +
              VL,MF=(E,PLIST)       ----------------------------------
```

BPX1GHA (gethostbyaddr) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose address is specified as input. For the callable service, see "gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address" on page 234. AMODE 64 callers use "BPX4GHA (gethostbyaddr) example" on page 1237.

The HOSTENT structure has the following format:

- h_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'. #$
- h_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h_addrtype - The value 2, which signifies AF_INET.
- h_length - The length of the host internet addresses pointed to by h_addr_list.
- h-addr_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

```
*        MVC   HOST_ADDR,=XL4'C90E0256'   IP Address of Host
         MVC   HOST_ADDRLEN,=F'4'         Address length
         SPACE ,
         CALL  BPX1GHA,              Get host by address              +
               (HOST_ADDR,           Input: IP address of queried HOST +
               HOST_ADDRLEN,         Input: Length of IP address      +
               HOSTENT_PTR,          Output: 0 or -> HOSTENT structure +
               =A(AF_INET),          Input: Domain - AF_INET          +
               RETVAL,               Return code                      +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GHN (gethostbyname) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose domain name is specified as input. For the callable service, see “gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names” on page 237. AMODE 64 callers use “BPX4GHN (gethostbyname) example” on page 1238.

The HOSTENT structure has the following format:

- h_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'.
- h_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h_addrtype - The value 2, which signifies AF_INET.
- h_length - The length of the host internet addresses pointed to by h_addr_list.
- h-addr_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

```
MVC   HOST_NAME(8),=CL8'HOST1234'
MVC   HOST_NAMELEN,=F'8'
SPACE ,
CALL  BPX1GHN,              Get host by name                 +
      (HOST_NAME,           Input: Name of Host being queried +
      HOST_NAMELEN,         Input: Length of host name       +
      HOSTENT_PTR,          Output: 0 or -> HOSTENT structure +
      RETVAL,               Return code                      +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GID (getgid) example

The following code gets the real group ID of the caller. For the callable service, see “getgid (BPX1GID, BPX4GID) — Get the real group ID” on page 220. AMODE 64 callers use “BPX4GID (getgid) example” on page 1238.

```
CALL  BPX1GID,              Get the real group ID            +
      (RETVAL),             Return value: real group ID      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GIV (givesocket) example

The following code gives a socket to the program identified by CID (clientid). The target program may then use takesocket (BPX1TAK) to take the socket. SOCKDESC was previously set by a call to BPX1ACP. CID is set by the getclientid (BPX1GCL) service. For the callable service, see "givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program" on page 285. For the data structure, see "BPXYCID — Map the returning structure for getclientid()" on page 951. AMODE 64 callers use "BPX4GIV (givesocket) example" on page 1238.

```
CALL  BPX1GIV,             give a socket to another program  +
      (SOCKDESC,           Input: Socket descriptor          +
      CID,                 Input: Clientid of recipient      +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GLG (getlogin) example

The following code gets the login name of the caller. For the callable service, see "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245. AMODE 64 callers use "BPX4GLG (getlogin) example" on page 1239.

```
CALL  BPX1GLG,             Get the login name                +
      (RETVAL),            Returns value, 0 or ->login name  +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GMN (w_getmntent) example

The following code gets the mount entries for the caller. For the callable service, see "w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems" on page 894. For the data structure, see "BPXYMNTE — Map response and element structure of w_getmntent" on page 993. AMODE 64 callers use "BPX4GMN (w_getmntent) example" on page 1239.

If BPXYMNTE is assembled with MNTE2=YES, fields MNTEHID and MNTEHBLEN must be initialized.

```
LA    R14,MNTEH            R14->MNTEH and MNTE
L     R15,MNTEL            R15 = Length of MNTEH and MNTE
XR    R0,R0                Dummy 2nd operand
XR    R1,R1                Pad=null, length=0
MVCL  R14,R0               Null out MNTEH and MNTE
MVC   MNTEHID,=CL4'MNT2'         Version indicator
MVC   MNTEHBLEN,=A(MNTE#LENGTH)  Length of MNTE
CALL  BPX1GMN,             Get mount entries                 +
      (MNTEL,              Input: Length BPXYMNTE + MNTEH    +
      MNTEH,               Header in BPXYMNTH                +
      RETVAL,              Return value: -1 or mount entries +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GNI (getnameinfo) example

The following code resolves a socket address into a host name and a service name. For the callable service, see “getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address” on page 246. AMODE 64 callers use “BPX4GNI (getnameinfo) example” on page 1239.

```
SPACE ,
CALL  BPX1GNI,              Get name info                    +
      (SOCKADDR,            Input: Socket address            +
      SOCKADDR_LENGTH,      Input: Length of socket address  +
      SERVICE_BUFFER,       I/O:   Buffer   for service name +
      SERVICE_BUFFER_LENGTH, I/O:  Length of service buffer  +
      HOST_BUFFER,          I/O:   Buffer   for host name    +
      HOST_BUFFER_LENGTH,   I/O:   Length of host buffer     +
      FLAGS,                Input: Flags                     +
      RETVAL,               Return code                      +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GPG (getpgrp) example

The following code gets the process group ID of the caller. For the callable service, see “getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID” on page 252. AMODE 64 callers use “BPX4GPG (getpgrp) example” on page 1240.

```
CALL  BPX1GPG,              Get the process group ID         +
      (RETVAL),             Return value: group ID           +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GNM (getpeername or getsockname) example

The following code gets the peer name, and then requests the socket name. SOCKDESC was returned by a previous call to BPX1SOC. For the callable service, see “getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer” on page 272. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043. AMODE 64 callers use “BPX4GNM (getpeername or getsockname) example” on page 1240.

```
SPACE ,
CALL  BPX1GNM,                Get peername                   +
      (SOCKDESC,              Input: Socket Descriptor       +
      SOCK#GNMOPTGETPEERNAME, Input: Indicate getpeername    +
      SOCK#LEN+SOCK_SUN#LEN,  Input: Length - Sockaddr       +
      SOCKADDR,               Input: Sockaddr structure      +
      RETVAL,                 Return value: 0 or -1          +
      RETCODE,                Return code                    +
      RSNCODE),               Reason code                    +
      VL,MF=(E,PLIST)         -------------------------------
SPACE ,
CALL  BPX1GNM,                Get sockname                   +
      (SOCKDESC,              Input: Socket Descriptor       +
      SOCK#GNMOPTGETSOCKNAME, Input: Indicate getpeername    +
      SOCK#LEN+SOCK_SUN#LEN,  Input: Length - Sockaddr       +
      SOCKADDR,               Input: Sockaddr structure      +
      RETVAL,                 Return value: 0 or -1          +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GPE (getpwent) example

The following code accesses the user database starting with the next available entry and continuing until end of fileon the database. It returns a structure identifying information about each user entry in the database. For the callable service, see "getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database" on page 258. For the data structure, see "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969. AMODE 64 callers use "BPX4GPE (getpwent) example" on page 1240.

```
GPELOOP  DS    0H
         CALL  BPX1GPE,             Access the user database          +
               (RETVAL,             Return value: 0 or ->BPXYGIDN     +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               VL,MF=(E,PLIST)      ----------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    CHKGPERR             Error or end of file
         USING GIDN,R8
*        access the user structure
         DROP  R8
         B     GPELOOP              Check next user entry
CHKGPERR DS    0H
         ICM   R8,B'1111',RETCODE
         BZ    GPEEOF               End of file
*        handle error as needed
GPEEOF   DS    0H
```

BPX1GPI (getpid) example

The following code gets the process ID of the caller. For the callable service, see "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253. AMODE 64 callers use "BPX4GPI (getpid) example" on page 1241.

```
         CALL  BPX1GPI,             Get the process ID                +
               (RETVAL),            Returns value, Process ID         +
               VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GPN (getpwnam) example

The following code accesses the group database by the user ID of the caller and returns a structure identifying the groups by name. For the callable service, see "getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name" on page 260. For the data structure, see "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969. AMODE 64 callers use "BPX4GPN (getpwnam) example" on page 1241.

```
         MVC   USERNLEN,=F'8'
         MVC   USERNAME(8),=CL8'Pebbles'
         SPACE ,
         CALL  BPX1GPN,             Access the user database          +
               (USERNLEN,           Input: Length of user name        +
               USERNAME,            Input: Name of user               +
               RETVAL,              Return value 0 or ->BPXYGIDN      +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GPP (getppid) example

The following code gets the process ID of the caller's parent. For the callable service, see "getppid (BPX1GPP, BPX4GPP) — Get the parent process ID" on page 254. AMODE 64 callers use "BPX4GPP (getppid) example" on page 1241.

```
CALL  BPX1GPP,              Get PID of the parent process     +
      (RETVAL),             Returns value, parent's process ID+
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GPS (w_getpsent) example

The following code gets process data associated with the first relative process (PROCTOK=0) to which the caller is authorized access (by the security access facility). For the callable service, see "w_getpsent (BPX1GPS) — Get process data" on page 897. For the data structure, see "BPXYPGPS — Map the response structure for w_getpsent" on page 1007.

```
LA    R15,PGPS              Getmain area mapped by BPXYPGPS
ST    R15,PGPSA             Hold pointer to this area
XC    PROCTOK,PROCTOK       First relative process (Zero)
LA    R2,PGPSCONTTYBUF      Controlling TTY  ->buffer
ST    R2,PGPSCONTTYPTR                       Store into PGPS
MVC   PGPSCONTTYBLEN,=A(L'PGPSCONTTYBUF)     Length
LA    R2,PGPSPATHBUF        Pathname         ->buffer
ST    R2,PGPSPATHPTR                         Store into PGPS
MVC   PGPSPATHBLEN,=A(L'PGPSPATHBUF)         Length
LA    R2,PGPSCMDBUF         Command          ->buffer
ST    R2,PGPSCMDPTR                          Store into PGPS
MVC   PGPSCMDBLEN,=A(L'PGPSCMDBUF)           Length
SPACE ,
CALL  BPX1GPS,              Get process data                  +
      (PROCTOK,             Input: Relative process token     +
      PGPSL,                Input: Length of buffer needed    +
      PGPSA,                I/O: ->Buffer, mapped by BPXYPGPS +
      RETVAL,               Return value: -1, 0, next proctok +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test Return value: 0 or -1
ST    R15,PROCTOK           The next relative process token
BZ    PSEUDO                RETVAL = 0, end of file
BM    PSEUDO                RETVAL < 0, error
BP    PSEUDO                RETVAL > 0, next logical process
```

BPX1GPT (grantpt) example

The following code grants access to the slave pseudoterminal device that is identified by the file descriptor. For the callable service, see "grantpt (BPX1GPT, BPX4GPT) — Grant access to the slave pseudoterminal" on page 289. AMODE 64 callers use "BPX4GPT (grantpt) example" on page 1241.

```
CALL  BPX1GPT,              Grant access to slave pty         +
      (MASTER_FD,           Input: File descriptor            +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GPU (getpwuid) example

The following code accesses the group database by the user name of the caller and returns a structure identifying the groups by name. The code sets the user ID value to 1. For the callable service, see "getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID" on page 263. For the data structure, see "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969. AMODE 64 callers use "BPX4GPU (getpwuid) example" on page 1242.

```
MVC   USERID,..             Value of user ID                  27
  SPACE ,
  CALL  BPX1GPU,              Access database by user ID       +
        (USERID,              Input: User ID                   +
        RETVAL,               Return value 0 or ->BPXYGIDN     +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GPY (getpriority) example

The following code gets the CPU priority based on the input which and who values. The which value used is PRIO_PROCESS, which indicates to get the priority by process ID. The who value used is 7, indicating to get the priority for process ID 7. For the callable service, see "getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process" on page 255. AMODE 64 callers use "BPX4GPY (getpriority) example" on page 1242.

```
MVC   PROCID,=XL4'00000007' Process ID to get priority for
SPACE ,
CALL  BPX1GPY,              Get priority value               +
      (=A(PRIO_PROCESS),    Input: Get by Process ID         +
      PROCID,               Input: PID to get priority for   +
      RETVAL,               Return value: Priority of process +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1GRL (getrlimit) example

The following code fills in the rlimit structure for the calling process based on the input resource value. The resource value is set to RLIMIT_CPU. For the callable service, see "getrlimit (BPX1GRL, BPX4GRL) — Get resource limits" on page 266. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 64 callers use "BPX4GRL (getrlimit) example" on page 1242.

```
MVC   RESOURCE,=A(RLIMIT_CPU) Value of resource
SPACE ,
CALL  BPX1GRL,              Get resource limits              +
      (RESOURCE,            Input: resource                  +
      RLIMIT,               Structure, mapped by BPXYRLIM    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1GRU (getrusage) example

The following code fills in the rusage structure based on the input who value. The who value is set to RUSAGE_SELF. For the callable service, see "getrusage (BPX1GRU, BPX4GRU) — Get resource usage" on page 268. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 64 callers use "BPX4GRU (getrusage) example" on page 1243.

```
MVC   WHO,=A(RUSAGE_SELF)   Value of who
SPACE ,
CALL  BPX1GRU,              Get resource usage               +
      (WHO,                 Input: who                       +
      RUSAGE,               Structure, mapped by BPXYRLIM    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1GTH (__getthent) example

The following code retrieves information on the first process accessible to the caller. For the callable service, see "__getthent (BPX1GTH, BPX4GTH) — Get thread data" on page 278. For the data structure, see "BPXYPGTH — Map the __getthent input/output structure" on page 1009. AMODE 64 callers use "BPX4GTH (__getthent) example" on page 1243.

```
LA    R5,BUFFERB            R5 -> Input parameters
ST    R5,BUFB               ->input buffer
USING PGTHA,R5              R5 base for PGTHA
XC    PGTHA(PGTHA#LEN),PGTHA  Null Input area
MVI   PGTHAFLAG1,PGTHAPROCESS+PGTHACOMMAND+PGTHATHREAD
MVI   PGTHAACCESSTHID,PGTH#LAST   Last thread
LA    R15,BUFFERA           PgthB, Output buffer
ST    R15,BUFA              ->output buffer
DROP  R5
SPACE ,
CALL  BPX1GTH,              __getthent                       +
      (=A(PGTHA#LEN),       Input: length input parms BPXYPGTH+
      BUFA,                 Input: ->input parms      BPXYPGTH+
      =A(1024),             Input: length output area BPXYPGTH+
      BUFB,                 Input: ->output area      BPXYPGTH+
      RETVAL,               Return value: 0, -1              +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1GTR (getitimer) example

The following code returns the time remaining an alarm, or ITIMER_REAL as set by setitimer. For the callable service, see "getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer" on page 242. For the data structure, see "BPXYITIM — Map getitimer, setitimer structure" on page 990. AMODE 64 callers use "BPX4GTR (getitimer) example" on page 1243.

```
LA    R15,ITIM              Output mapping structure
ST    R15,ITIMA             ->structure
CALL  BPX1GTR,              Get process data                 +
      (=A(ITIMER_REAL),     Input: Relative process token    +
```

```
      ITIMA,               Out: ->Buffer, mapped by BPXYITIM +
      RETVAL,              Return value: -1, 0               +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GUG (getgroupsbyname) example

The following code returns the number of supplementary group IDs, up to 9, for user Pebbles. For the callable service, see "getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs by user name" on page 231. AMODE 64 callers use "BPX4GUG (getgroupsbyname) example" on page 1244.

```
MVC   USERNLEN,=F'7'
MVC   USERNAME(07),=CL07'Pebbles'
MVC   BUFLENA,=F'9'
LA    R15,BUFFERA
ST    R15,BUFA
SPACE ,
CALL  BPX1GUG,             Get list of groups by user name   +
      (USERNLEN,           Input: User name length           +
      USERNAME,            Input: User name                  +
      BUFLENA,             Input: Group ID list size         +
      BUFA,                Group ID list address             +
      RETVAL,              Return value: -1, or # of grp IDs +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GUI (getuid) example

The following code gets the invoker's real user ID. For the callable service, see "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282. AMODE 64 callers use "BPX4GUI (getuid) example" on page 1244.

```
CALL  BPX1GUI,             Get the real user ID              +
      (RETVAL),            Return value: real user ID        +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1GWD (getwd) example

The following code gets the working directory for the caller. For the callable service, see "getwd (BPX1GWD, BPX4GWD) — Get the pathname of the working directory" on page 283. AMODE 64 callers use "BPX4GWD (getwd) example" on page 1244.

```
MVC   BUFLENA,=F'1024'     Max directory name return area
SPACE ,
CALL  BPX1GWD,             Get working directory name        +
      (BUFLENA,            Input: Length directory work area +
      BUFFERA,             Buffer                            +
      RETVAL,              Return value: length or -1        +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1HST (gethostid or gethostname) example

The following code requests the host id and the host name for an AF_INET domain. For the callable service, see “gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host” on page 240. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043. AMODE 64 callers use “BPX4HST (gethostid or gethostname) example” on page 1245.

```
XC    BUFLENA,BUFLENA
CALL  BPX1HST,              Request host id                  +
      (=A(AF_INET),         Input: Domain - AF_INET          +
      BUFLENA,              Input: Length - No buffer - get id+
      BUFFERA,              Output: (not used with Length=0)  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------

MVC   BUFLENA,=A(L'BUFFERA)
CALL  BPX1HST,              Request host name                +
      (=A(AF_INET),         Input: Domain - AF_INET          +
      BUFLENA,              Input: Length - for output name   +
      BUFFERA,              Output: Buffer for host name     +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1IOC (w_ioctl) example

The following code conveys a command to the standard output device. To run properly this example needs a command defined by the user for the COMMAND parameter. This command must be understood by the device driver providing support for the output device. For the callable service, see “w_ioctl (BPX1IOC, BPX4IOC) — Control I/O” on page 902. AMODE 64 callers use “BPX4IOC (w_ioctl) example” on page 1245.

```
MVC   BUFLENA,=F'1024'
MVC   COMMAND,=F'123'       User defined command
SPACE ,
CALL  BPX1IOC,              I/O Control                      +
      (=A(STDOUT_FILENO),   Input: File descriptor           +
      COMMAND,              Input: Command                   +
      BUFLENA,              Input: Argument length           +
      BUFFERA,              Argument buffer name             +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1IPT (mvsiptaffinity) example

The following code executes the assembler routine EXITRTN on the IPT of the requesting thread, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread” on page 410. AMODE 64 callers use “BPX4IPT (mvsiptaffinity) example” on page 1245.

```
         MVC   EXITRTNA,=V(EXITRTN)  ->Routine address
*        MVC   EXITPLA,=A(EXITPARM)  ->Input parameter list
         SPACE ,
         CALL  BPX1IPT,                                          +
               (EXITRTNA,            Input: Routine address      +
               EXITPLA,              Input: Parm list address or 0 +
               RETVAL,               Return value: -1 or not return +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal” on page 301.

```
CALL  BPX1ITY,              Determine if device is a TTY      +
      (=A(STDOUT_FILENO),   Input: File descriptor            +
      RETVAL),              Return value: 0 isn't, 1 is       +
      VL,MF=(E,PLIST)       ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BZ    PSEUDO                RETVAL=0 means device not terminal
```

BPX2ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX2ITY, BPX4ITY) (X/Open Version) — Determine whether a file descriptor represents a terminal” on page 303. AMODE 64 callers use “BPX4ITY (isatty) example” on page 1246.

```
CALL  BPX2ITY,              Determine if device is a TTY      +
      (=A(STDOUT_FILENO),   Input: File descriptor            +
      RETVAL,               Return value: 0 isn't, 1 is, -1   +
      RETCODE,              Return code: describes why VAL=-1 +
      RSNCODE),             Reason code: qualifier on RETCODE +
      VL,MF=(E,PLIST)       ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BZ    PSEUDO                RETVAL=0 means device not terminal
```

BPX1KIL (kill) example

The following code sends a signal (SIGUSR1) to all processes for which access is allowed in the invoker's process group. For the callable service, see “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 64 callers use “BPX4KIL (kill) example” on page 1246.

```
MVC   PROCID,=A(0)          Invoker's process group
CALL  BPX1KIL,              Send a signal to a process        +
      (PROCID,              Input: Process ID                 +
      =A(SIGUSR1#),         Input: Signal            BPXYSIGH +
      =A(0),                Input: Signal options             +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1LCO (lchown) example

The following code changes the owner of symbolic link **/somedir/somesymlink.c** from the current owner to that specified by USERID and GROUPID. For the callable service, see "lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link" on page 324. AMODE 64 callers use "BPX4LCO (lchown) example" on page 1246.

```
  MVC   BUFFERA(22),=CL22'/somedir/somesymlink.c'
  MVC   BUFLENA,=F'22'
MVC   USERID,..             New owner UID from stat            33
MVC   GROUPID,..            New owner GID from stat            33
  SPACE ,
  CALL  BPX1LCO,              Change owner and group of a file  +
        (BUFLENA,             Input: Pathname length            +
        BUFFERA,              Input: Pathname                   +
        USERID,               Input: New owner UID              +
        GROUPID,              Input: New owner GID              +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1LCR (lchattr) example

The following code changes the attributes of symbolic link **/somedir/somesymlink.c**. The security label is set and the file change time is set. For the callable service, see "lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link" on page 315. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948. AMODE 64 callers use "BPX4LCR (lchattr) example" on page 1247.

```
  MVC   BUFFERA(22),=CL22'/somedir/somesymlink.c'
  MVC   BUFLENA,=F'22'
  XC    ATT,ATT               Clear ATT
  MVC   ATTID,=CL4'ATT '      Eye Catcher
  MVC   ATTVERSION,=AL2(ATT#VER)  version
  MVC   ATTSECLABEL,=CL08'SYSMULTI'
  OI    ATTSETFLAGS3,ATTSECLABELCHG                           +
        Flag Seclabel update
  OI    ATTSETFLAGS2,ATTCTIMETOD                              +
        Set change time to current time
  SPACE ,
  CALL  BPX1LCR,              Change file attributes            +
        (BUFLENA,             Input: Pathname length            +
        BUFFERA,              Input: Pathname                   +
        =A(ATT#LENGTH),       Input: BPXYATT length             +
        ATT,                  Input/output: BPXYATT             +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1LDX (loadHFS extended) example

The following is an example specifying the Lod_Directed option. For an example of BPX1LDX/BPX4LDX without the Lod_Directed option flag specified, see "BPX1LOD (loadHFS) example" on page 1158, substituting BPX1LDX/BPX4LDX for BPX1LOD/BPX4LOD. The program ictasma located at **ict/bin** is loaded into storage and then branched to. Then the CSVDYLPA service is called to provide serviceability information to the system. The loaded module can then be branched

to. When the load module is no longer needed, the serviceability information should be deleted and the module's storage released. For the callable service, see "loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage" on page 338. AMODE 64 callers use "BPX4LDX (loadHFS extended) example" on page 1247.

```
 MVC   BUFLENA,=F'13'
 MVC   BUFFERA(16),=C'/ict/bin/ictasma'
 MVC   OPTIONS,=AL4(LOD_DIRECTED)  Directed loadhfs to common
 OI    OPTIONS+3,X'F1'             Subpool 241
 MVC   LIBPTHLN,=A(0)
 SPACE ,
 CALL  BPX1LDX,   Load program                              +
        (BUFLENA,  Input: Pathname length                   +
       BUFFERA,   Input: Pathname                           +
       OPTIONS,   Input: Options                            +
       LIBPTHLN,  Input: Library Path Length                +
       LIBPATH,   Input: Library Path                       +
       RTNPARM@,  Return value: -1 or direct load ret parms +
       RETCODE,   Return code                               +
       RSNCODE),  Reason code                               +
       MF=(E,PLIST)  ----------------------------------------
 SPACE ,
 L     R15,RTNPARM@          Load return value
 C     R15,=F'-1'            Test for -1 return
               BE   PSEUDO                Branch on error
             L     R5,RTNPARM@
 MVC   LOCALPARMS(24),0(R5)  Local copy of returned parameters
*
*              Provide serviceability information to system
*
 LA    R4,LOCALPARMS
 USING DIRECTEDLOADRETURNEDPARMS,R4
 L     R5,DIRECTEDLOADMODULEENTRYPT
 L     R6,DIRECTEDLOADMODULESTART
 L     R7,DIRECTEDLOADMODULELENGTH
 XC    LPMEA(LPMEA_LEN),LPMEA
 ST    R5,LPMEAENTRYPOINTADDR
 ST    R6,LPMEALOADPOINTADDR
 ST    R7,LPMEAMODLEN
 MVC   LPMEANAME,=C'ICTASMA '
 CSVDYLPA REQUEST=ADD,                                      +
       BYADDR=YES,                                          +
       MODINFOTYPE=MEMBERLIST,                              +
       MODINFO=LPMEA,                                       +
       NUMMOD=1,                                            +
       REQUESTOR=REQID,                                     +
       RETCODE=RETCODE,                                     +
       RSNCODE=RSNCODE,                                     +
       MF=(E,DYLPAL)    Provide serviceability information
 L     R15,RETCODE      Load return code
 LTR   R15,R15
 BNZ   PSEUDO
 MVC   LOCALDELTOKEN(8),LPMEADELETETOKEN
 SPACE ,
 .
 .
 .
*
*          Call directed loadhfs target module
*
 L     R15,DIRECTEDLOADMODULEENTRYPT
 BALR  R14,R15   Branch to loaded program
 SPACE ,
  .
 .
```

```
 .
*
*         When done with directed load hfs module
*         remove serviceability information and
*         release module storage
*
 XC    LPMED(LPMED_LEN),LPMED
 MVC   LPMEDNAME,=C'ICTASMA '
 MVC   LPMEDDELETETOKEN(8),LOCALDELTOKEN
 CSVDYLPA REQUEST=DELETE,                                      +
       TYPE=BYTOKEN,                                           +
       MODINFO=LPMED,                                          +
       NUMMOD=1,                                               +
       RETCODE=RETCODE,                                        +
       RSNCODE=RSNCODE,                                        +
      MF=(E,DYLPAL)    Remove serviceability information
 L     R15,RETCODE      Load return code
 LTR   R15,R15
 BNZ   PSEUDO
 SPACE ,
 MODESET MODE=SUP
 L     R7,DIRECTEDLOADMODULELENGTH
               STORAGE RELEASE,                                +
       LENGTH=(R7),                                            +
       ADDR=DIRECTEDLOADMODULESTART,                           +
       SP=241           Free module
 MODESET MODE=PROB
 DROP  R4
```

BPX1LOD (loadHFS) example

The program ictasma located at **ict/bin** is loaded into storage and then branched to. For the callable service, see "loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name" on page 333. AMODE 64 callers use "BPX4LOD (loadHFS) example" on page 1249.

```
        MVC   BUFLENA,=F'16'
        MVC   BUFFERA(16),=C'/ict/bin/ictasma'
        MVC   OPTIONS,=A(0)
        MVC   LIBPTHLN,=A(0)
        SPACE ,
        CALL  BPX1LOD,              Load program                      +
              (BUFLENA,             Input: Pathname length            +
              BUFFERA,              Input: Pathname                   +
              OPTIONS,              Input: Options                    +
              LIBPTHLN,             Input: Library Path Length        +
              LIBPATH,              Input: Library Path               +
              EPADDR,               Return value: -1 or entrypt addr  +
              RETCODE,              Return code                       +
              RSNCODE),             Reason code                       +
              VL,MF=(E,PLIST)       ---------------------------------
        SPACE ,
        L     R15,EPADDR            Load return value
        C     R15,=F'-1'            Test for -1 return
        BE    PSEUDO                Branch on error
        SPACE ,
        L     R15,EPADDR
        BALR  R14,R15               Branch to loaded program
```

BPX1LNK (link) example

The following code creates a new way for **usr/dataproc.next.t** to link to an existing file, **usr/user05/yearrecs.t** . For the callable service, see "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327. AMODE 64 callers use "BPX4LNK (link) example" on page 1249.

```
MVC   BUFLENA,=F'21'
MVC   BUFFERA(21),=CL21'usr/user05/yearrecs.t'
MVC   BUFLENB,=F'19'
MVC   BUFFERB(19),=CL19'usr/dataproc.next.t'
SPACE ,
CALL  BPX1LNK,              Create a link to a file          +
      (BUFLENA,             Input: Name length: existing     +
      BUFFERA,              Input: Name of existing file     +
      BUFLENB,              Input: Name length: link         +
      BUFFERB,              Input: Name of link to file      +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1LSK (lseek) example

The following code changes the file (FILEDESC) offset to 80 bytes past the current offset. For the callable service, see "lseek (BPX1LSK, BPX4LSK) — Change a file's offset" on page 345. For the data structure, see "BPXYSEEK — Constants for lseek" on page 1036. AMODE 64 callers use "BPX4LSK (lseek) example" on page 1250.

```
MVC   FILEDESC,..           File descriptor from open         34
  MVC   OFFSET(08),=FL8'80'   Forward 80 Bytes
  MVC   REFPT,=A(SEEK_CUR)    Current offset of the file
  SPACE ,
  CALL  BPX1LSK,              Change a file's offset           +
        (FILEDESC,            File descriptor                  +
        OFFSET,               I/O: Offset in file              +
        REFPT,                Input: Reference point, BPXYSEEK +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1LSN (listen) example

The following code issues a listen on a socket that was previously created and that had a bind done for it. SOCKDESC was returned from the call to BPX1SOC. Set the backlog count to 5. For the callable service, see "listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients" on page 330. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4LSN (listen) example" on page 1250.

```
CALL  BPX1LSN,              Listen on a socket               +
      (SOCKDESC,            Input: Socket Descriptor         +
      =A(5),                Input: Backlog count of 5        +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1LST (lstat) example

The following code obtains the file status for the file described by the symbolic name **labrec/sym**. For the callable service, see "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349. For the data structure, see "BPXYSTAT — Map the response structure for stat" on page 1057. AMODE 64 callers use "BPX4LST (lstat) example" on page 1250.

```
*   symbolic name established using symlink (BPX1SYM) system call
        MVC   BUFFERA(10),=CL10'labrec/sym'
        MVC   BUFLENA,=F'10'
        SPACE ,
        CALL  BPX1LST,              Get file status                  +
              (BUFLENA,             Input: Pathname length           +
              BUFFERA,              Input: Pathname                  +
              STATL,                Input: Length of buffer needed   +
              STAT,                 Buffer, mapped by BPXYSTAT       +
              RETVAL,               Return value: 0 or -1            +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1MAT (shmat) example

The following code attaches a shared memory segment. For the callable service, see "shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment" on page 714. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039. AMODE 64 callers use "BPX4MAT (shmat) example" on page 1251.

```
        CALL  BPX1MAT,              Shared memory segment control    +
              (SHM_ID,              Input: Shared memory segment ID  +
              SEGADDR,              Input: ST loc for seg address    +
              =A(0),                Input: Flags              BPXYSHM +
              RETVAL,               Return value: 0, -1 or ->segment +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1MCT (shmctl) example

The following code retrieves the size of the shared memory segment. For the callable service, see "shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations" on page 718. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039. AMODE 64 callers use "BPX4MCT (shmctl) example" on page 1251.

```
        LA    R15,BUFFERA
        ST    R15,BUFA
        SPACE ,
        CALL  BPX1MCT,              Shared memory segment control    +
              (SHM_ID,              Input: Shared memory segment ID  +
              =A(IPC_STAT),         Input: Command            BPXYIPC +
              BUFA,                 Input: ->SHMID_DS or 0    BPXYSHM +
              RETVAL,               Return value: 0, -1 or value     +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1MDT (shmdt) example

The following code detaches a shared memory segment. For the callable service, see “shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment” on page 722. For the data structure, see “BPXYSHM—Map interprocess communication shared memory segments” on page 1039. AMODE 64 callers use “BPX4MDT (shmdt) example” on page 1251.

```
CALL  BPX1MDT,              Shared memory segment detach      +
      (SEGADDR,             Input: Shared memory segment addr +
      RETVAL,               Return value: 0, -1 or value      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1MGT (shmget) example

The following code creates a private shared memory segment of 500 bytes. For the callable service, see “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 738. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1037. AMODE 64 callers use “BPX4MGT (shmget) example” on page 1252.

```
MVC   KEY(4),=A(IPC_PRIVATE)  Local to this family
MVI   S_TYPE,IPC_CREAT+IPC_EXCL   Must not already exist
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX1MGT,              Create a set of semaphores        +
      (KEY,                 Input: Shared memory segment KEY  +
      =A(500),              Input: Segment size               +
      S_MODE,               Input: Creation flags     BPXYIPC +
      RETVAL,               Return value: -1 or MessageQue ID +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on shmget failure
ST    R15,SHM_ID            Store SHM_ID associated with key
```

BPX1MKD (mkdir) example

The following code creates a new and empty directory pathname of **/usr/newprots/** with user read-execute, group write, other read-execute permissions. For the callable service, see “mkdir (BPX1MKD, BPX4MKD) — Make a directory” on page 361. For the data structure, see “BPXYFTYP — File type definitions” on page 967 and “BPXYMODE — Map the mode constants of the file services” on page 996. AMODE 64 callers use “BPX4MKD (mkdir) example” on page 1252.

```
MVC   BUFFERA(14),=CL14'/usr/newprots/'
MVC   BUFLENA,=F'14'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       Read search write read search
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
SPACE ,
CALL  BPX1MKD,              Make a directory                  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      S_MODE,               Input: BPXYMODE and BPXYFTYP      +
```

```
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1MKN (mknod) example

The following code creates a FIFO (pipe) named **/u/fifos/fifo1** and user read-write, group read, other read permissions. For the callable service, see "mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file" on page 364. For the data structure, see "BPXYFTYP — File type definitions" on page 967 and "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 64 callers use "BPX4MKN (mknod) example" on page 1252.

```
MVC   BUFFERA(14),=CL14'/u/fifos/fifo1'
MVC   BUFLENA,=F'14'
XC    S_MODE,S_MODE
MVI   S_TYPE,FT_FIFO       First in - first out
MVI   S_MODE2,S_IRUSR      Read write read read
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH
SPACE ,
CALL  BPX1MKN,             Create FIFO or char special file  +
      (BUFLENA,            Input: Pathname length            +
      BUFFERA,             Input: Pathname                   +
      S_MODE,              Input: BPXYMODE and BPXYFTYP      +
      =A(0),               Input: Device id not used here    +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1MMI (__map_init) example

The following code creates a shared memory map with 10 map blocks each with a size of 1 meg. For the callable service, see "__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area" on page 352. For the data structure, see "BPXYMMG — Map interface for _map_init and _map_service" on page 991. AMODE 64 callers use "BPX4MMI (__map_init) example" on page 1253.

```
LA    R2,INITPARM          Set address of init parm list
ST    R2,INITADDR
USING _MMG_INIT_PARM,R2
XC    _MMG_INIT_PARM(_MMG_INIT_PARM_LEN),_MMG_INIT_PARM
L     R1,=F'10'            Map area to contain 10 blocks
ST    R1,_MMG_NUMBLKS      *
L     R1,=F'1'             Each block is to be 1 meg in size
ST    R1,_MMG_MEGSPERBLK   *
SPACE ,
CALL  BPX1MMI,                                           +
      (=A(MMG_INIT),       Input: Function code             +
      INITADDR,            Input: __map_init parameter list +
      RETVAL,              Return value: 0, -1              +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1MMP (mmap) example

The following code changes the protection of a memory mapped area. For the callable service, see "mmap (BPX1MMP, BPX4MMP) — Map pages of memory" on page 368. AMODE 64 callers use "BPX4MMP (mmap) example" on page 1253.

```
MVC   FILEDESC,..          File descriptor                   37
  SPACE ,
  CALL  BPX1MMP,             map pages of memory              +
        (MAP_ADDRESS,        Input: address of mapped area    +
        MAP_LENGTH,          Input: area length               +
        =A(MAP_PRIVATE),     Input: Map type                  +
        FILEDESC,            Input: File descriptor           +
        FILEOFFSET,          Input: File offset               +
        RETVAL,              Return value: 0 or -1            +
        RETCODE,             Return code                      +
        RSNCODE),            Reason code                      +
        VL,MF=(E,PLIST)      ---------------------------------
```

BPX1MMS (__map_service) example

The following code creates three new data blocks within a shared memory map. For the callable service, see "__map_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services" on page 356. For the data structure, see "BPXYMMG — Map interface for _map_init and _map_service" on page 991. AMODE 64 callers use "BPX4MMS (__map_service) example" on page 1254.

```
LA    R3,SRVCPARM          Set address of init parm list
ST    R3,SRVCADDR
USING _MMG_SERVICE_PARM,R3
XC  _MMG_SERVICE_PARM(_MMG_SERVICE_PARM_LEN),_MMG_SERVICE_PARM
LA    R4,MMG_NEWBLOCK      Request that a block be created
STH   R4,_MMG_SERVICETYPE
LA    R3,_MMG_SERVICE_PARM_LEN(R3) Bump to next entry
STH   R4,_MMG_SERVICETYPE  Create a second block
LA    R3,_MMG_SERVICE_PARM_LEN(R3) Bump to next entry
STH   R4,_MMG_SERVICETYPE  Create the third block
SPACE ,
CALL  BPX1MMS,                                             +
      (=A(MMG_SERVICE),    Input: Function code             +
      SRVCADDR,            Input: __map_service parm list   +
      =F'3',               Input: Three requests to process +
      _MMG_MAPTOKEN,       Map area token from INIT call    +
      RETVAL,              Return value: 0, -1              +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1MNT (mount) example

The following code requests that the file system mount the system file TESTLIB.FILESYS1 and ready it for use. For the callable service, see "mount (BPX1MNT) — Make a file system available" on page 377. For the data structure, see "BPXYMTM — Map the modes for mount and unmount" on page 1000.

```
XC    MTM(MTM#LENGTH),MTM
MVI   MTM1,MTMRDWR         Mount mode - read-write
MVC   BUFLENA,=F'2'        Max 1023
MVC   BUFFERA(02),=CL02'/u'
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
MVC   FSTYPE(8),=CL08'HFS'
CALL  BPX1MNT,             Ready a file system for use      +
```

```
      (BUFLENA,             Input: Mount point length         +
      BUFFERA,              Input: Mount point name           +
      FSNAME,               Input: File system name (44 char) +
      FSTYPE,               Input: File system type (8 char)  +
      MTM,                  Input: Mount mode         BPXYMTM +
      =A(0),                Input: Parm length, future        +
      =A(0),                Input: Parm, future               +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX2MNT (__mount) example

The following code requests that the file system __mount the system file and ready it for use. The file system name and mount parameters are encoded into the various fields in the MNTE. See "mount (BPX1MNT) — Make a file system available" on page 377. AMODE 64 callers use "BPX4MNT (__mount) example" on page 1254.

```
LA    R14,MNTEH             R14->MNTEH and MNTE
L     R15,MNTEL             R15 = Length of MNTEH and MNTE
XR    R0,R0                 Dummy 2nd operand
XR    R1,R1                 Pad=null, length=0
MVCL  R14,R0                Null out MNTEH and MNTE
MVC   MNTEHID,=CL4'MNT2'          Version indicator
MVC   MNTEHBLEN,=A(MNTE#LENGTH)   Length of MNTE
MVC   MNTENTFSTNAME(08),=CL08'HFS'     HFS type name
MVC   MNTENTFSNAME(44),=CL44'TESTLIB.FILESYS1'  Filesystem
MVC   MNTENTMOUNTPOINT(02),=CL02'/u'   Mount point
MVC   MNTENTPATHLEN,=F'2'
MVC   MNTENTFSMODE4,=A(MNTENTFSMODERDONLY)   Filesystem mode
CALL  BPX2MNT,              Ready a file system for use       +
      (MNTEL,               Input: MNTE length (hdr + body)   +
      MNTEH,                Input: MNTE                       +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1MP (mvspause) example

The following code places this thread into an MVS WAIT, to be terminated when a user ECB specified on a prior MVSpauseInit call is POSTed. The MVS WAIT is also terminated if a signal occurs. For the callable service, see "mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals" on page 413. AMODE 64 callers use "BPX4MP (mvspause) example" on page 1255.

```
CALL  BPX1MP,               MVS Pause                         +
      (RETVAL,              Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1MPC (mvsprocclp) examples

1. The following code causes all z/OS UNIX-related resources to be released for this thread, and if this is the last thread in the process, for the process.

```
XC    WAST(WAST#LENGTH),WAST
MVI   WASTEXITCODE,57       User defined exit code
SPACE ,
```

```
          CALL  BPX1MPC,             MVS Process cleanup              +
                (WAST,               Input: Ending status code 0-255  +
                RETVAL,              Return value: 0, -1 or 1         +
                RETCODE,             Return code                      +
                RSNCODE),            Reason code                      +
                VL,MF=(E,PLIST)      ---------------------------------
```

2. To indicate that the process ended with a specific code, the application should set up the WAST as follows, and then call BPX1MPC:

```
*************************************************************
*  Set up the WAST (exit status word)                       *
*  with a user defined exit code                            *
*************************************************************
LA    R3,0                   Set R3 with zero and
ST    R3,LOCALWAST           clear the WAST
LA    R14,LOCALWAST          Get address of WAST
MVI   WASTEXITCODE(R14),44   Set exit status
```

3. To indicate that the process ended with a terminating signal, the application should set up the WAST as follows prior to calling BPX1MPC:

```
*************************************************************
*  Set up the WAST (exit status word)                       *
*  with a terminating signal                                *
*************************************************************
LA    R3,0                  Set R3 with zero and
ST    R3,LOCALWAST          clear the WAST
LA    R14,LOCALWAST         Get address of WAST
MVI   WASTSIGTERM(R14),09   Exit with sigterm (x'09')
```

4. If an application does not care about the terminating status of a process, and the parent will not check the status after issuing a call to the wait service, then the application should set the WAST to zero prior to calling BPX1MPC:

```
*************************************************************
*  Set up the WAST (exit status word)                       *
*  Do not set any exit codes                                *
*************************************************************
LA    R3,0                  Set R3 with zero and
ST    R3,LOCALWAST          clear the WAST
LA    R14,LOCALWAST         Get address of WAST
```

For the callable service, see “mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources” on page 418. For the data structure, see “BPXYWAST — Map the wait status word” on page 1069. AMODE 64 callers use “BPX4MPC (mvsprocclp) example” on page 1255.

BPX1MPI (mvspauseinit) example

The following code prepares the thread for a subsequent MVSpause invocation. A list of Event Control Block addresses is passed to the system with the last address having the high order bit on. This syscall will use the first ECB pointed to from the list as the signal ECB, therefore at least one ECB address must be passed to the system. For the callable service, see “mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals” on page 413. AMODE 64 callers use “BPX4MPI (mvspauseinit) example” on page 1255.

```
         LA    R15,BUFFERA          Load address of ECB address list
         ST    R15,BUFA             Save address for future parameter
*                                    to be passed to BPX1MPI
         SR    R15,R15              Clear R15
         ST    R15,ECB01            Clear ECB01
         ST    R15,ECB02            Clear ECB02
         LA    R15,ECB01            Load address of first ECB
         ST    R15,BUFFERA          Save ECB address in list of
```

```
*                                  pointers
         LA    R15,ECB02           Load address of second ECB
         ST    R15,BUFFERA+4       Save ECB address in list of
*                                  pointers
         OI    BUFFERA+4,X'80'     Denote end of ECB pointers
         SPACE ,
         CALL  BPX1MPI,            MVS Pause initialize             +
               (BUFA,              Input ->list of ECB@, x'80' ended +
               RETVAL,             Return value: 0, -1              +
               RETCODE,            Return code                      +
               RSNCODE),           Reason code                      +
               VL,MF=(E,PLIST)     ---------------------------------
```

BPX1MPR (mprotect) example

The following code changes the protection of a memory mapped area. For the callable service, see "mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping" on page 384. AMODE 64 callers use "BPX4MPR (mprotect) example" on page 1256.

```
         CALL  BPX1MPR,            set protection of a mapped area  +
               (MAP_ADDRESS,       Input: address of mapped area    +
               MAP_LENGTH,         Input: area length               +
               =A(PROT_READ),      Input: Protection options        +
               RETVAL,             Return value: 0 or -1            +
               RETCODE,            Return code                      +
               RSNCODE),           Reason code                      +
               VL,MF=(E,PLIST)     ---------------------------------
```

BPX1MSD (mvsunsigsetup) example

The following code detaches the invoker from being able to catch signals. For the callable service, see "mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup" on page 430. C AMODE 64 callers use "BPX4MSD (mvsunsigsetup) example" on page 1256.

```
         CALL  BPX1MSD,            Reregister MVS signals, this task +
               (SIRTNA,            Signal interface routine address  +
               USERWORD,           User data                         +
               INTMASK,            Default override signal set       +
               TERMMASK,           Default terminate signal set      +
               RETVAL,             Return value: 0 or -1             +
               RETCODE,            Return code                       +
               RSNCODE),           Reason code                       +
               VL,MF=(E,PLIST)     ---------------------------------
```

BPX1MSS (mvssigsetup) example

The following code allows the invoker to catch signals. For the callable service, see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421. AMODE 64 callers use "BPX4MSS (mvssigsetup) example" on page 1256.

```
* Each bit of the mask represents a signal 1-64.
         MVC   INTMASK(8),=XL8'F000000000000000'    Default sig 1-4
         MVC   TERMMASK(8),=XL8'F000000000000000'   Terminate sig 1-4
         LA    R15,BUFFERA
         ST    R15,USERWORD
         SPACE ,
         CALL  BPX1MSS,            Register MVS signals, this task  +
               (=V(SIRTN),         Input: Signal interrupt routine  +
               USERWORD,           Input: User data                 +
               INTMASK,            Input: Default override signals  +
```

```
      TERMMASK,            Input: Default terminate signals  +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
```

BPX1MSY (msync) example

The following code causes the file associated with this mapped area to be updated with the contents of storage. For the callable service, see "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403. AMODE 64 callers use "BPX4MSY (msync) example" on page 1257.

```
MVC   FILEDESC,..          File descriptor                   41
  SPACE ,
  CALL  BPX1MSY,             synchronize memory with storage   +
        (MAP_ADDRESS,        Input: address of mapped area     +
        MAP_LENGTH,          Input: area length                +
        =A(MS_SYNC),         Input: sync options               +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ----------------------------------
```

BPX1MUN (munmap) example

The following code causes a mapped area to be unmapped. For the callable service, see "munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses" on page 407. AMODE 64 callers use "BPX4MUN (munmap) example" on page 1257.

```
  CALL  BPX1MUN,             unmap previously mapped addresses +
        (MAP_ADDRESS,        Input: address of mapped area     +
        MAP_LENGTH,          Input: area length                +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ----------------------------------
```

BPX1NIC (nice) example

The following code increases the priority value of the calling process by 1. For the callable service, see "nice (BPX1NIC, BPX4NIC) — Change the nice value of a process" on page 432. AMODE 64 callers use "BPX4NIC (nice) example" on page 1257.

```
  MVC   INCR,=F'1'           Increase priority by 1
  SPACE ,
  CALL  BPX1NIC,             Change priority value             +
        (INCR,               Input: Priority change value      +
        RETVAL,              Return value: new nice value or -1+
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ----------------------------------
  L     R15,RETVAL           Load return value
  C     R15,=F'-1'           Test for -1 return
  BE    PSEUDO               Branch on error
```

BPX1OPD (opendir) example

The following code opens directory **/etc/passwd** so that it can be read by readdir. For the callable service, see "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452. AMODE 64 callers use "BPX4OPD (opendir) example" on page 1257.

```
MVC   BUFLENA,=F'11'
MVC   BUFFERA(11),=CL11'/etc/passwd'
SPACE ,
CALL  BPX1OPD,              Open a directory                 +
      (BUFLENA,             Input: Directory name length     +
      BUFFERA,              Input: Directory name            +
      RETVAL,               Return value:-1 or directory f.d. +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,DIRECTDES         Store the directory descriptor
```

BPX1OPN (open) example

The following code opens file **usr/inv/nov.d** with user read-write, group read and other read. A file descriptor (FILEDESC) is returned. For the callable service, see "open (BPX1OPN, BPX4OPN) — Open a file" on page 447. For the data structure, see "BPXYOPNF — Map flag values for open" on page 1004, AMODE 64 callers use "BPX4OPN (open) example" on page 1258. "BPXYMODE — Map the mode constants of the file services" on page 996, and "BPXYFTYP — File type definitions" on page 967.

```
MVC   BUFFERA(13),=CL13'usr/inv/nov.d'
MVC   BUFLENA,=F'13'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       User read/write, group read,
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH           other read
XC    O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI   O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
SPACE ,
CALL  BPX1OPN,              Open a file                      +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      O_FLAGS,              Input: Access            BPXYOPNF +
      S_MODE,               Input: Mode    BPXYMODE, BPXYFTYP +
      RETVAL,               Return value:-1 or file descriptor+
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,FILEDESC          Store the file descriptor
```

BPX2OPN (openstat) example

The following code opens file **usr/inv/nov.d** with user read-write, group read and other read, and obtains status about the file. A file descriptor (FILEDESC) is returned. For the callable service, see "openstat (BPX2OPN, BPX4OPS) — Open a file and obtain status information" on page 454. For the data structures, see "BPXYOPNF — Map flag values for open" on page 1004, "BPXYSTAT — Map the response structure for stat" on page 1057, "BPXYMODE — Map the mode

constants of the file services" on page 996, and "BPXYFTYP — File type definitions" on page 967. AMODE 64 callers use "BPX4OPS (openstat) example" on page 1258.

```
MVC   BUFFERA(13),=CL13'usr/inv/nov.d'
MVC   BUFLENA,=F'13'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       User read/write, group read,
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH           other read
XC    O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI   O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
SPACE ,
CALL  BPX2OPN,              Open a file and get status        +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      O_FLAGS,              Input: Access            BPXYOPNF +
      S_MODE,               Input: Mode    BPXYMODE, BPXYFTYP +
      STATL,                Input: Length of buffer needed    +
      STAT,                 Buffer, BPXYSTAT                  +
      RETVAL,               Return value:-1 or file descriptor+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,FILEDESC          Store the file descriptor
```

BPX1OPT (getsockopt or setsockopt) example

The following code gets and then sets socket options. SOCKDESC was returned on a previous call to BPX1SOC. For the callable service, see "getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket" on page 275. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4OPT (getsockopt or setsockopt) example" on page 1259.

```
MVC   BUFLENA,=A(L'BUFFERA)
CALL  BPX1OPT,                 Get socket options            +
      (SOCKDESC,               Input: Socket Descriptor      +
      =A(SOCK#OPTOPTGETSOCKOPT), Input: Indicate Get socket  +
      SOCK#SOL_SOCKET,         Input: Level                  +
      SOCK#SO_TYPE,            Input: Option name            +
      BUFLENA,                 Input: Length - option value  +
      BUFFERA,                 Input: Option value           +
      RETVAL,                  Return value: 0 or -1         +
      RETCODE,                 Return code                   +
      RSNCODE),                Reason code                   +
      VL,MF=(E,PLIST)       ----------------------------------
SPACE ,
MVC   BUFLENA,=A(4)            SO_OOBINLINE has length=4
CALL  BPX1OPT,                 Set socket options            +
      (SOCKDESC,               Input: Socket Descriptor      +
      =A(SOCK#OPTOPTSETSOCKOPT), Input: Indicate set socket  +
      SOCK#SOL_SOCKET,         Input: Level                  +
      SOCK#SO_TYPE,            Input: Option name            +
      BUFLENA,                 Input: Length - option value  +
      SOCK#SO_OOBINLINE,       Input: Option value           +
      RETVAL,                  Return value: 0 or -1         +
      RETCODE,                 Return code                   +
      RSNCODE),                Reason code                   +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1PAF (__pid_affinity) example

The following code will add your PID to the target process' affinity list. For the callable service, see "__pid_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list" on page 477. AMODE 64 callers use "BPX4PAF (__pid_affinity) example" on page 1259.

```
*       MVC   TARPID,....          PID of target
*       MVC   SIGPID,....          PID of this routine
        CALL  BPX1PAF,                                          +
              (=A(PAF_ADD_PID#),   Function code (add entry)    +
              TARPID,              PID of target                +
              SIGPID,              PID to receive signal        +
              =A(SIGUSR1#),        signal to be generated       +
              RETVAL,              Return value: 0 or -1        +
              RETCODE,             Return code                  +
              RSNCODE),            Reason code                  +
              VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PAS (pause) example

The following code suspends execution of the invoker's thread until a signal is delivered. For the callable service, see "pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal" on page 468. AMODE 64 callers use "BPX4PAS (pause) example" on page 1260.

```
        CALL  BPX1PAS,             Suspend execution            +
              (RETVAL,             Return value: -1 or not return +
              RETCODE,             Return code                  +
              RSNCODE),            Reason code                  +
              VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PCF (pathconf) example

The following code extracts the current value for the configurable maximum number of bytes in a file name associated with **/usr/inv/network.t**. For the callable service, see "pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name" on page 464. For the data structure, see "BPXYPCF — Command values for pathconf and pathconf" on page 1005. AMODE 64 callers use "BPX4PCF (pathconf) example" on page 1260.

```
MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
MVC   BUFLENA,=F'18'
SPACE ,
CALL  BPX1PCF,             Get configurable pathname variable+
      (BUFLENA,            Input: Pathname length           +
      BUFFERA,             Input: Pathname                  +
      =A(PC_NAME_MAX),     Input: Options           BPXYPCF +
      RETVAL,              Return value: 0, -1 or variable  +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PCT (pfsctl) example

The following code conveys a command to a Physical File System named ACMEFILE. ACMEFILE doesn't really exist; to actually run this example you would need a real PFS product that supports this function. For the callable service, see "pfsctl (BPX1PCT, BPX4PCT) — Physical file system control" on page 470. AMODE 64 callers use "BPX4PCT (pfsctl) example" on page 1260.

```
MVC   FSTYPE(8),=CL08'ACMEFILE'
MVC   BUFLENA,=F'25'
MVC   BUFFERA(25),=CL25'COMPRESS(ON) CONVERT(OFF)'
MVC   COMMAND,=F'123'       PFS product defined command
SPACE ,
CALL  BPX1PCT,              PFS Control                      +
      (FSTYPE,              Input: PFS Type Name             +
      COMMAND,              Input: Command                   +
      BUFLENA,              Input: Argument length           +
      BUFFERA,              Input/Output: Argument buffer    +
      RETVAL,               Return value: product defined    +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1PIP (pipe) example

The following code creates a pipe. For the callable service, see “pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe” on page 481. AMODE 64 callers use “BPX4PIP (pipe) example” on page 1261.

```
CALL  BPX1PIP,              Create a pipe                    +
      (READFD,              Output: Read file descriptor     +
      WRITEFD,              Output: Write file descriptor    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1POE (__poe) example

The following code registers a socket (SOCKDESC) as the process scope port of entry. SOCKDESC was returned previously from a call to either BPX1SOC or BPX1ACP. For the callable service, see “__poe() (BPX1POE, BPX4POE) — Port of entry information” on page 483. For the data structure, see “BPXYPOE — Map poe syscall parameters” on page 1013. AMODE 64 callers use “BPX4POE (__poe) example” on page 1261.

```
MVC   POEOPTIONS,=A(POE#SCOPEPROCESS)
MVC   POEENTRYTYPE,=A(POE#ENTRYSOCKET)
MVC   POEENTRYLEN,=A(POE#ENTRYSOCKETLEN)
LA    R15,SOCKDESC
ST    R15,POEENTRYPTR
CALL  BPX1POE,              Port of Entry registration       +
      (=A(POE#LEN),         Input: Length of poe structure   +
      POE,                  Input: mapped by BPXYPOE         +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1POL (poll) example

The following code issues a poll. For the callable service, see “poll (BPX1POL, BPX4POL) — Monitor activity on file descriptors and message queues” on page 488. For the data structure, see “BPXYPOLL — Map poll syscall parameters” on page 1014. AMODE 64 callers use “BPX4POL (poll) example” on page 1261.

```
LA    R15,BUFFERA
USING POLLFD,R15
```

BPX1POL (poll) example

```
         ST    R15,BUFA             ->BPXYPOLL structure
*        MVC   POLLHFD(4),file_descriptor_number2
         MVI   POLLEVENTS,0
         MVI   POLLEVENTS+1,POLLERDNORM
         A     R15,=A(POLLFD#LENGTH)
*        MVC   POLLHFD(4),file_descriptor_number1
         MVI   POLLEVENTS,0
         MVI   POLLEVENTS+1,POLLEWRNORM
         SPACE ,
         CALL  BPX1POL,             Create a pipe                    +
               (BUFA,               Input: address of BPXYPOLL       +
               =A(2),               Input: number of BPXYPOLL structs +
               =A(0),               Input: -1, 0, milliseconds       +
               RETVAL,              Return value: 0 or -1            +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PSI (pthread_setintr) example

The following code sets the interruption type of the calling thread. For the callable service, see "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4PSI (pthread_setintr) example" on page 1262.

```
         CALL  BPX1PSI,             Examine and change interrupt state+
               (INTRSTATE,          Input: Interrupt state   BPXYCONS +
               RETVAL,              Return value: 0 or -1            +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PST (pthread_setintrtype) example

The following code sets the interruption type of the calling thread and returns the previous interruption type. For the callable service, see "pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type" on page 530"pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type" on page 530. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4PST (pthread_setintrtype) example" on page 1262.

```
         CALL  BPX1PST,             Examine and change interrupt type +
               (INTRTYPE,           Input: Interrupt type    BPXYCONS +
               RETVAL,              Return value: 0 or -1            +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTB (pthread_cancel) example

The following code generates a cancelation request for the target thread (THID). For the callable service, see "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495. AMODE 64 callers use "BPX4PTB (pthread_cancel) example" on page 1262.

```
         CALL  BPX1PTB,             pthread_cancel                   +
               (THID,               Input: Thread ID                 +
               RETVAL,              Return Value: 0, -1, or Buf length+
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTC (pthread_create) example

The following code creates a new thread. For the callable service, see "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495. For the data structure, see "BPXYPTAT — Map attributes for pthread_exit_and_get" on page 1017. AMODE 64 callers use "BPX4PTC (pthread_create) example" on page 1262.

```
LA    R15,BUFFERA          Work area
ST    R15,BUFA             ->above
LA    R15,PTAT             Area mapped by BPXYPTAT
ST    R15,PTATA            ->above
MVC   PTATEYE,=C'BPXYPTAT'             Set the eye-catcher
MVC   PTATLENGTH,=A(PTATUSEROFFVAL)    Length of structure
MVC   PTATSYSOFFSET,=A(PTATSYSOFFVAL)  Sys attr offset
MVC   PTATSYSLENGTH,=A(PTATSYSLENVAL)  Sys attr length
MVC   PTATUSEROFFSET,=A(0)             User attr offset
MVC   PTATUSERLENGTH,=A(0)             User attr length
LOAD  EP=INITRTN           Get address of Init Rtn
ST    R0,INITRTNA
SPACE ,
CALL  BPX1PTC,                                          +
      (INITRTNA,           Input: Init  routine address  +
      BUFA,                Input: Work area address      +
      PTATA,               Input: Attr area Address BPXYPTAT +
      THID,                Thread ID,  if Return value = 0   +
      RETVAL,              Return value: 0 or -1         +
      RETCODE,             Return code                   +
      RSNCODE),            Reason code                   +
      VL,MF=(E,PLIST)      ------------------------------
```

BPX1PTD (pthread_detach) example

The following code detaches a thread (THID) in the calling process. For the callable service, see "pthread_detach (BPX1PTD, BPX4PTD) — Detach a thread" on page 503. AMODE 64 callers use "BPX4PTD (pthread_detach) example" on page 1263.

```
CALL  BPX1PTD,             pthread_detach                   +
      (THID,               Input: Thread ID                 +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTI (pthread_testintr) example

The following code causes a cancelation point. For the callable service, see "pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur" on page 536. AMODE 64 callers use "BPX4PTI (pthread_testintr) example" on page 1263.

```
CALL  BPX1PTI,             Cause an interrupt point to occur +
      (RETVAL,             Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTJ (pthread_join) example

The following code gets the termination status of a specified thread (THID). For the callable service, see “pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread” on page 509. AMODE 64 callers use “BPX4PTJ (pthread_join) example” on page 1263.

```
         CALL  BPX1PTJ,             pthread_join                    +
               (THID,               Input: Thread ID                +
               =A(0),               Input: ->Status Field or 0      +
               RETVAL,              Return value: 0 or -1           +
               RETCODE,             Return code                     +
               RSNCODE),            Reason code                     +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTK (pthread_kill) example

The following code sends a signal to a specified thread (THID). For the callable service, see “pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread” on page 512. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 64 callers use “BPX4PTK (pthread_kill) example” on page 1264.

```
*        MVC   SIGNAL,=A(SIGALRM#)   Input: SIGALRM           BPXYSIGH
*        MVC   SIGNALOPTIONS,=XL4'00000000' Input: Signal options
*        CALL  BPX1PTK,              pthread_kill                    +
               (THID,               Input: Thread ID                +
               SIGNAL,              Input: Signal or 0       BPXYSIGH +
               SIGNALOPTIONS,       Input: Signal options           +
               RETVAL,              Return value: 0 or -1           +
               RETCODE,             Return code                     +
               RSNCODE),            Reason code                     +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTQ (pthread_quiesce) example

The following code terminates all other pthreads in the caller's process. For the callable service, see “pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 515. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 64 callers use “BPX4PTQ (pthread_quiesce) example” on page 1264.

```
         CALL  BPX1PTQ,             pthread_quiesce                 +
               (=A(QUIESCE_TERM),   Input: Quiesce type     BPXYCONS +
               =A(0),               Input: User data - Catch data PPSD+
               RETVAL,              Return value: 0 or -1           +
               RETCODE,             Return code                     +
               RSNCODE),            Reason code                     +
               VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTR (ptrace) example

The following code enables a process (PROCID) to be debugged with ptrace. For the callable service, see “ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging” on page 537. For the data structure, see “BPXYPTRC — Map parameters for ptrace” on page 1018. AMODE 64 callers use “BPX4PTR (ptrace) example” on page 1264.

```
*        MVC   PROCID,  Process ID from fork
         SPACE ,
         CALL  BPX1PTR,             Debug another process           +
               (=A(PT_ATTACH),      Input: Request           BPXYPTRC +
```

```
      PROCID,              Input: Process ID                +
      =A(0),               Input: Address                   +
      =A(0),               Input: Data                      +
      =A(0),               Input: Buffer                    +
      RETVAL,              Return value: 0, -1, or Request  +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTS (pthread_self) example

The following code gets the thread ID of the calling thread. For the callable service, see “pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID” on page 526. AMODE 64 callers use “BPX4PTS (pthread_self) example” on page 1265.

```
CALL  BPX1PTS,             pthread_self                     +
      (THID),              Output: Thread ID                +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTT (pthread_tag_np) example

The following code updates the pthread tag. For the callable service, see “pthread_tag_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data” on page 533. AMODE 64 callers use “BPX4PTT (pthread_tag_np) example” on page 1265.

```
LA    R15,=CL30'UPDATING MONTH-END STATISTICS'
ST    R15,PT_NEWA
LA    R15,PT_OLD
ST    R15,PT_OLDA
CALL  BPX1PTT,             pthread_tag_np                   +
      (=A(30),             Input: Length of New Tag         +
      PT_NEWA,             Input: Address of New Tag        +
      PT_OLDL,             Input: Length of Old Tag         +
      PT_OLDA,             Input: Address to store Old Tag  +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code:                     +
      RSNCODE),            Reason code:                     +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PTX (pthread_exit_and_get) example

The following code terminates a thread and creates a new thread. For the callable service, see “pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread” on page 505. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 64 callers use “BPX4PTX (pthread_exit_and_get) example” on page 1265.

```
CALL  BPX1PTX,             pthread_exit_and_get             +
      (STATFLD,            Input: Status field              +
      OPTIONS,             Input: Options field             +
      SIGNALREG,           Input: Signal registration usrdata+
      RETVAL,              Return value: 0 or -1  ->BPXYPTXL +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1PWD (__passwd, __passwd__applid) example

The following code queries/changes the password of a given user ID. For the callable service, see "__passwd, __passwd__applid (BPX1PWD, BPX4PWD) — Verify or change security information" on page 459. AMODE 64 callers use "BPX4PWD (__passwd, __passwd__applid) example" on page 1265.

```
MVC   USERNLEN,=F'8'
MVC   USERNAME(8),=CL8'Myuserid'
MVC   OLDPASSLEN,=F'8'
MVC   OLDPASS(8),=CL8'MyOldPwd'
MVC   NEWPASSLEN,=F'8'
MVC   NEWPASS(8),=CL8'MyNewPwd'
SPACE ,
CALL  BPX1PWD,              Query/change user ID password     +
      (USERNLEN,            Input: Length of user ID          +
      USERNAME,             Input: User ID                    +
      OLDPASSLEN,           Input: Length of old password     +
      OLDPASS,              Input: Old password               +
      NEWPASSLEN,           Input: Length of new password     +
      NEWPASS,              Input: New password               +
      RETVAL,               Return value 0 or -1              +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1QCT (msgctl) example

The following code removes the message queue from the system. For the callable service, see "msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations" on page 388. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 64 callers use "BPX4QCT (msgctl) example" on page 1266.

```
CALL  BPX1QCT,              Message queue control (msgctl)    +
      (MSG_ID,              Input: MessageQueueID             +
      =A(IPC_RMID),         Input: Action to take     BPXYIPC +
      =A(0),                Input: ->MSQID_DS or 0    BPXYMSG +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1QDB (querydub) example

The following code obtains the dub status information for the current task. The status indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread). AMODE 64 callers use "BPX4QDB (querydub) example" on page 1266.

```
CALL  BPX1QDB,              Query DUB status for this task    +
      (RETVAL,              Return value: -1 or see BPXYCONS  +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1QGT (msgget) example

The following code creates a private message queue. For the callable service, see "msgget (BPX1QGT, BPX4QGT) — Create or find a message queue" on page 391. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 64 callers use "BPX4QGT (msgget) example" on page 1266.

```
MVI   S_TYPE,IPC_CREAT+IPC_EXCL     Error if exists
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX1QGT,              Create a message queue            +
      (=A(IPC_PRIVATE),     Input: Key                        +
      S_MODE,               Input: Creation flags BPXYMODE/IPC+
      RETVAL,               Return value: -1 or msg ID  +532200
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on msgget failure
ST    R15,MSG_ID            Store MSG_ID associated with key
```

BPX1QRC (msgrcv) example

The following code adds a message to the message queue identified by MSG_ID. For the callable service, see "msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue" on page 395. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 64 callers use "BPX4QRC (msgrcv) example" on page 1267.

```
LA    R15,BUFFERA           R15 -> Utility buffer
ST    R15,BUFA
USING MSGBUF,R15
MVC   MSG_TYPE(4),=A(0)
MVC   BUFLENA(4),=A(MSQ#LENGTH)
MVC   FLAGS(4),=A(0)        Wait for message
DROP  R15
SPACE ,
CALL  BPX1QSN,              Send a message (msgrcv)           +
      (MSG_ID,              Input: MessageQueueID             +
      BUFA,                 Input: ->MSGBUF           BPXYMSG +
      PRIMARYALET,          Input: ALET of message buffer     +
      BUFLENA,              Input: Length MSGBUF              +
      =A(0),                Input: Message Type       BPXYMSG +
      FLAGS,                Input: Flags              BPXYIPC +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1QSE (quiesce) example

The following code quiesces file system TESTLIB.FILESYS1, making the files in it unavailable for use. For the callable service, see "quiesce (BPX1QSE, BPX4QSE) — Quiesce a file system" on page 570. AMODE 64 callers use "BPX4QSE (quiesce) example" on page 1267.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX1QSE,              Quiesce a file system            +
      (FSNAME,              Input: File system name (44 char) +
      RETVAL,               Return value: 0, -1, or 4        +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1QSN (msgsnd) example

The following code adds a message to the message queue identified by MSG_ID. For the callable service, see "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 64 callers use "BPX4QSN (msgsnd) example" on page 1267.

```
LA    R15,BUFFERA           R15 -> Utility buffer
ST    R15,BUFA
USING MSGBUF,R15
MVC   MSG_TYPE(4),=A(0)
MVC   MSG_MTEXT(11),=CL11'QSN MSG TEXT'
MVC   BUFLENA(4),=A(15)
MVC   FLAGS(4),=A(IPC_NOWAIT)   Don't wait on queue full
DROP  R15
SPACE ,
CALL  BPX1QSN,              Send a message (msgsnd)          +
      (MSG_ID,              Input: MessageQueueID            +
      BUFA,                 Input: ->MSGBUF           BPXYMSG +
      PRIMARYALET,          Input: ALET of message buffer    +
      BUFLENA,              Input: Length MSGBUF             +
      FLAGS,                Input: Flags              BPXYIPC +
      RETVAL,               Return value: 0, -1              +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1RCV (recv) example

The following code issues a recv for a socket. SOCKDESC was returned previously from a call to either BPX1SOC or BPX1ACP. For the callable service, see "recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer" on page 597. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 64 callers use "BPX4RCV (recv) example" on page 1268.

```
SPACE ,
CALL  BPX1RCV,              Receive data on from a socket    +
      (SOCKDESC,            Input: Socket Descriptor         +
      =A(L'BUFFERA),        Input: Length of input buffer    +
      BUFFERA,              Input: Address of input buffer   +
      PRIMARYALET,          Input: Alet of input buffer      +
      MSG_FLAGS,            Input: Flags                     +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1RDD (readdir) example

The following code reads multiple name entries from the specified directory (DIRECTDES). For the callable service, see "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577. For the data structure, see "BPXYDIRE — Map directory entries for readdir" on page 965. AMODE 64 callers use "BPX4RDD (readdir) example" on page 1268.

```
MVC   DIRECTDES,..          Directory descriptor from  opendir 55
  LA    R15,BUFFERA
  ST    R15,BUFA
  MVC   BUFLENA,=F'1023'
  CALL  BPX1RDD,              Read entries from a directory     +
        (DIRECTDES,           Input: Directory file descriptor  +
        BUFA,                 Output: ->buffer         BPXYDIRE +
        PRIMARYALET,          Input: buffer ALET                +
        BUFLENA,              Input: buffer size                +
        RETVAL,               Return value: 0, -1, entries read +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RDL (readlink) example

The following code reads the contents of symbolic link **/personnel/templink** into the buffer provided. This will be the pathname that was specified when the symbolic link was defined. For the callable service, see "readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link" on page 587. AMODE 64 callers use "BPX4RDL (readlink) example" on page 1269.

```
  MVC   BUFFERB(19),=CL19'/personnel/templink'
  MVC   BUFLENB,=F'19'
  LA    R15,BUFFERA
  ST    R15,BUFA
  MVC   BUFLENA,=F'1023'
  SPACE ,
  CALL  BPX1RDL,              Read the value of a symbolic link +
        (BUFLENB,             Input: Linkname length            +
        BUFFERB,              Input: Link name                  +
        BUFLENA,              Input: Buffer size - 1023         +
        BUFA,                 ->Buffer for symbolic link        +
        RETVAL,               Return value: 0, -1 or char count +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RDV (readv) example

The following code issues a readv for a socket. SOCKDESC was returned previously from a call to either BPX1SOC or BPX1ACP. For the callable service, see "readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers" on page 590. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 64 callers use "BPX4RDV (readv) example" on page 1269.

```
 SPACE ,
  LA    R2,BUFFERA
  ST    R2,IOV_BASE
  LA    R2,L'BUFFERA
  ST    R2,IOV_LEN
  CALL  BPX1RDV,              Read into a vector of buffers     +
```

```
      (SOCKDESC,            Input: Socket Descriptor          +
      =A(1),                Input: Number of elements in iov  +
      IOV,                  Input: Iov containing info        +
      PRIMARYALET,          Input: Alet where iov resides     +
      PRIMARYALET,          Input: Alet of buffers for data   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RDX (read extlink) example

The following code reads the contents of external symbolic link **/personnel/tmpxlink** into the buffer provided. This will be the pathname that was specified when the external symbolic link was defined. For the callable service, see "read_extlink (BPX1RDX, BPX4RDX) — Read an external symbolic link" on page 584. AMODE 64 callers use "BPX4RDX (read extlink) example" on page 1269.

```
  MVC   BUFFERB(19),=CL19'/personnel/tmpxlink'
  MVC   BUFLENB,=F'19'
  LA    R15,BUFFERA
  ST    R15,BUFA
  MVC   BUFLENA,=F'1023'
  SPACE ,
  CALL  BPX1RDX,              Read value of an external link    +
        (BUFLENB,             Input: Linkname length            +
        BUFFERB,              Input: Link name                  +
        BUFLENA,              Input: Buffer size - 1023         +
        BUFA,                 ->Buffer for symbolic link        +
        RETVAL,               Return value: 0, -1 or char count +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RD2 (readdir2) example

The following code reads multiple name entries from the specified directory (DIRECTDES). FUIOCURSOR, set to zero by the BPXYFUIO macro, indicates that the system is to begin reading with the first entry in the directory. For the callable service, see "readdir2 (BPX1RD2, BPX4RD2) — Read an entry from a directory" on page 580. For the data structure, see "BPXYDIRE — Map directory entries for readdir" on page 965. AMODE 64 callers use "BPX4RD2 (readdir2) example" on page 1270.

```
MVC   DIRECTDES,..          Directory descriptor from  opendir 56
  MVC   FUIOID,=CL4'FUIO'     Eye Catcher
  MVC   FUIOLEN,=AL4(FUIO#LENGTH) length
  LA    R15,BUFFERA           Set address of buffer
  ST    R15,FUIOBUFFERADDR    for directory data in FUIO
  MVC   FUIOIBYTESRW,=F'1023' Max number of bytes to read
  SPACE ,
  CALL  BPX1RD2,              Read directory entries            +
        (DIRECTDES,           Input: Directory file descriptor  +
        FUIO,                 Input/output: BPXYFUIO            +
        RETVAL,               Return value: 0, -1 or char count +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RED (read) example

The following code reads 80 bytes from the specified file (FILEDESC) and places them in the area provided (BUFFERA). For the callable service, see "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572. AMODE 64 callers use "BPX4RED (read) example" on page 1270.

```
MVC   FILEDESC,..          File descriptor                    57
  LA    R15,BUFFERA          Buffer
  ST    R15,BUFA             Buffer address
  MVC   BUFLENA,=F'80'       Read buffer length
  SPACE ,
  CALL  BPX1RED,             Read from a file                  +
        (FILEDESC,           Input: File descriptor            +
        BUFA,                ->Buffer to read into             +
        PRIMARYALET,         Input: Buffer ALET                +
        BUFLENA,             Input: Number of bytes to read    +
        RETVAL,              Return value: 0, -1, or char count+
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ----------------------------------
```

BPX1REN (rename) example

The following code changes the directory name of a file from **usr/sam** to **usr/samantha**. For the callable service, see "rename (BPX1REN, BPX4REN) — Rename a file or directory" on page 607. AMODE 64 callers use "BPX4REN (rename) example" on page 1271.

```
  MVC   BUFFERB(07),=CL07'usr/sam'
  MVC   BUFLENB,=F'07'
  MVC   BUFFERA(12),=CL12'usr/samantha'
  MVC   BUFLENA,=F'12'
  SPACE ,
  CALL  BPX1REN,             Rename a file                     +
        (BUFLENB,            Input: Old name length            +
        BUFFERB,             Input: Old name                   +
        BUFLENA,             Input: New name length            +
        BUFFERA,             Input: New name                   +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ----------------------------------
```

BPX1RFM (recvfrom) example

The following code issues a recv from a socket. SOCKDESC was returned from a previous call, either BPX1SOC or BPX1ACP. For the callable service, see "recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer" on page 600. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 64 callers use "BPX4RFM (recvfrom) example" on page 1271.

```
  SPACE ,
  MVC   MSG_FLAGS4,MSG_PEEK
  CALL  BPX1RFM,             Read from a socket                +
        (SOCKDESC,           Input: Socket Descriptor          +
        =A(L'BUFFERA),       Input: Length of the input buffer +
        BUFFERA,             Input: Address of the input buffer+
        PRIMARYALET,         Input: Alet of the input buffer   +
        MSG_FLAGS,           Input: Flags                      +
```

```
      =A(L'SOCKADDR),       Input: Length of the socket addr  +
      SOCKADDR,             Input: The socket address         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RMD (rmdir) example

The following code removes directory **applib/user02**. For the callable service, see "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615. AMODE 64 callers use "BPX4RMD (rmdir) example" on page 1271.

```
MVC   BUFFERA(13),=CL13'applib/user02'
MVC   BUFLENA,=F'13'
SPACE ,
CALL  BPX1RMD,              Remove a directory                +
      (BUFLENA,             Input: Directory name length      +
      BUFFERA,              Input: Directory to be removed    +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RMG (resource) example

The following code retrieves system-wide resource measurement data. For the callable service, see "resource (BPX1RMG, BPX4RMG) — Measure resources" on page 611. For the data structure, see "BPXYRMON — Map resource monitor data" on page 1034. AMODE 64 callers use "BPX4RMG (resource) example" on page 1272.

```
CALL  BPX1RMG,              Resource measurement gatherer     +
      (RMONL,               Input: Length of BPXYRMON         +
      RMON,                 Input: Buffer, BPXYRMON           +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX2RMS (recvmsg) example

The following code issues a recvmsg for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see "recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers" on page 604. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043, "BPXYMSGF — Map the message flags" on page 997, "BPXYMSGH — Map the message header" on page 999, and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 64 callers use "BPX4RMS (recvmsg) example" on page 1272.

```
SPACE ,
XC    MSGH(MSGH#LENGTH),MSGH  Clear msgh
LA    R2,SOCKADDR
ST    R2,MSGHNAMEPTR        Store the address of sockaddr
LA    R2,SOCK#LEN+SOCK_SUN#LEN
ST    R2,MSGHNAMELEN
LA    R2,IOV
ST    R2,MSGHIOVPTR
MVI   MSGHIOVNUM,1
LA    R2,BUFFERA
```

```
         ST    R2,IOV_BASE
         LA    R2,L'BUFFERA
         ST    R2,IOV_LEN
*
         CALL  BPX2RMS,              Receive a message from a socket   +
               (SOCKDESC,            Input: Socket Descriptor          +
               MSGH,                 Input: Address of BPXYMSGH        +
               MSG_FLAGS,            Input: Flags                      +
               PRIMARYALET,          Input: Alet of the iov            +
               PRIMARYALET,          Input: Alet of the buffers in iov +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RPH (realpath) example

The following code gets the absolute pathname without dot (.), dot-dot (..), or symbolic links for the input pathname. For the callable service, see "realpath (BPX1RPH, BPX4RPH) — Resolve a pathname" on page 594. AMODE 64 callers use "BPX4RPH (realpath) example" on page 1272.

```
         MVC   BUFFERA(8),=CL2'..'
         MVC   BUFLENA,=F'2'
         MVC   BUFLENB,=F'1024'      Resolved pathname return area
         SPACE ,
         CALL  BPX1RPH,              Resolve pathname                  +
               (BUFLENA,             Input: Pathname length            +
               BUFFERA,              Input: Pathname                   +
               BUFLENB,              Input: Length resolved name area  +
               BUFFERB,              Output: Resolved name buffer      +
               RETVAL,               Return value: -1 or length        +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1RW (Pwrite) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). It will start writing at specified offset, 30 bytes from start of the file. To positional read from a file, change the FUIORWIND to indicate FUIO#RD. For the callable service, see "Pread() and Pwrite() (BPX1RW, BPX4RW) — Read from or write to a file without changing the file pointer" on page 492. AMODE 64 callers use "BPX4RW (Pwrite) example" on page 1273.

```
       MVC   FILEDESC,             File descriptor from open
         XC    FUIO,FUIO             Zero out Fuio fields
         MVC   FUIOID,=CL4'FUIO'     Eye Catcher
         MVC   FUIOLEN,=AL4(FUIO#LENGTH) length
         LA    R15,BUFFERA           Set address of buffer
         ST    R15,FUIOBUFFERADDR    for buffer data in FUIO
         MVI   FUIORWIND,FUIO#WRT    Flag to indicate to PWrite
         MVC   FUIOIBYTESRW,=F'80'   Number of bytes to Write
         MVC   FUIOCUR2,=F'30'       Offset to start writing
         LA    R15,FUIO              Set address of Fuio
         ST    R15,LFUIOPTR          For access to Fuio fields
         SPACE ,
         CALL  BPX1RW,               PWrite to a file                  +
               (FILEDESC,            Input: File descriptor            +
               LFUIOPTR,             Input: Address of FUIO struct     +
               PRIMARYALET,          Input: Fuio ALET                  +
               FUIOLEN,              Input: Fuio Length                +
               RETVAL,               Return value: -1 or bytes written +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1RWD (rewinddir) example

The following code resets the open directory to the beginning. For the callable service, see "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613. AMODE 64 callers use "BPX4RWD (rewinddir) example" on page 1273.

```
MVC   DIRECTDES,..         File descriptor from opendir       59
  CALL  BPX1RWD,             Reposition directory at beginning +
        (DIRECTDES,          Input: Directory file descriptor  +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ---------------------------------
```

BPX1SA2 (__sigactionset) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see "__sigactionset (BPX1SA2, BPX4SA2) — Examine or change a set of signal actions" on page 751. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 64 callers use "BPX4SA2 (__sigactionset) example" on page 1274.

```
  XC    R15,R15
  ST    R15,SSETOPTION_FLAGS
  OI    SSETOPTION_FLAGS1,SSET_IGINVALID
  LA    R14,1
  ST    R11,BUFCNTB
  LA    R14,BUFFERA
  USING SSET,R14
  MVC   SSETFLAGS,=XL4'00000000'
  MVC   SSETSAMASK,=XL8'0FFF0F0000000000'
  MVC   SSETSAHANDLER,EPADDR
  MVC   SSETUSERDATA,=CL4'DATA'
  DROP  R14
  SPACE ,
  CALL  BPX1SA2,             Examine/change multiple sig acts  +
        (=A(1),              Input: One SSET set               +
        BUFFERA,             Input: Signal set input BPXYSSET  +
        BUFCNTB,             In/Out: Number of array elements  +
        BUFFERB,             Output: Address of output struct  +
        SSETOPTION_FLAGS,    Input: Mapped by BPXYSSET         +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        VL,MF=(E,PLIST)      ---------------------------------
```

BPX1SCT (semctl) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM_ID semaphore set. For the callable service, see "semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations" on page 626. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037. AMODE 64 callers use "BPX4SCT (semctl) example" on page 1274.

```
  LA    R15,BUFFERA
  ST    R15,BUFA
```

```
MVC   SEM_NUMBER(4),4       Semaphore number 4 in set
SPACE ,
CALL  BPX1SCT,              Semaphore control operations      +
      (SEM_ID,              Input: Semaphore set ID           +
      SEM_NUMBER,           Input: Semaphore number (0 based) +
      =A(SEM_GETPID),       Input: Action to take     BPXYSEM +
      BUFA,                 Input: Value | Buffer | Array | 0 +
      RETVAL,               Return value: 0, -1 or value      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SDD (setdubdefault) example

The following code sets the dub default setting for the subtasks of the caller to process. For the callable service, see "set_dub_default (BPX1SDD, BPX4SDD) — Set the dub default service" on page 666. AMODE 64 callers use "BPX4SDD (setdubdefault) example" on page 1274.

```
CALL  BPX1SDD,              Set effective group ID            +
      (=A(DUBPROCESS),      Input: Set Dub Constant  BPXYCONS +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SEC (__login, __login__applid, __certificate) example

The following code will invoke RACF (or other security product) to create a security environment (ACEE) for the calling process with the identity of JOEUSER. For the callable service, see "__login, __login__applid, __certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product" on page 309. AMODE 64 callers use "BPX4SEC (__login, __login__applid, __certificate) example" on page 1275.

```
MVC   USERNLEN,=F'7'
MVC   USERNAME(7),=CL7'JOEUSER'
MVC   OLDPASSLEN,=F'8'
MVC   OLDPASS,=CL8'JOESPASS'
MVC   OPTIONS,=F'0'
SPACE ,
CALL  BPX1SEC,              Create security environment       +
      (=A(SECURITY_CREATE#), Input: Function_code    BPXYCONS +
      SECURITY_USERID#,      Input: ID-Type          BPXYCONS +
      USERNLEN,              Input: UserID Length             +
      USERNAME,              Input: UserID                    +
      OLDPASSLEN,            Input: Password Length           +
      OLDPASS,               Input: Password                  +
      =A(0),                 Input: Holder                    +
      =A(0),                 Input: Holder                    +
      OPTIONS,               Input: Options                   +
      RETVAL,                Return value: 0 or -1            +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      VL,MF=(E,PLIST)        ---------------------------------
```

BPX1SEG (setegid) example

The following code sets the effective group ID of the invoker to 1. For the callable service, see "setegid (BPX1SEG, BPX4SEG) — Set the effective group ID" on page 670. AMODE 64 callers use "BPX4SEG (setegid) example" on page 1275.

```
MVC   GROUPID,=XL4'00000001' Value of new effective ID
SPACE ,
CALL  BPX1SEG,              Set effective group ID           +
      (GROUPID,             Input: Group ID                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SEL (select) example

The following code issues a select for a previously connected socket. SOCKDESC was returned when the socket was created. In this case, the select is for a single socket for read, write and exception. Do not request waiting. There are no ECBs. For the callable service, see “select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues” on page 618. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043 and “BPXYSEL — Map the select options” on page 1036. AMODE 64 callers use “BPX4SEL (select) example” on page 1275.

```
        SPACE ,
*
        MVC   SELLIST(4),=XL4'81000000'                          +
                              Turn on the bit representing sd 0 +
                              and sd 7
        LA    R8,8            One more than largest descriptor
        ST    R8,SOCKDESC     Set number of sockets to check
*
        CALL  BPX1SEL,        Select on a set of sockets         +
              (SOCKDESC,      Input: Number of file descriptors  +
              =A(4),          Input: Length of read list         +
              SELLIST,        Input: Read list                   +
              =A(4),          Input: Length of write list        +
              SELLIST,        Input: Write list                  +
              =A(4),          Input: Length of exception list    +
              SELLIST,        Input: Exception list              +
              =A(0),          Input: Address of Timeout value    +
              =A(0),          Input: ECB pointer                 +
              =A(SEL#BITSFORWARD), Input: Option - bits forward  +
              RETVAL,         Return value: 0 or -1              +
              RETCODE,        Return code                        +
              RSNCODE),       Reason code                        +
              VL,MF=(E,PLIST) ---------------------------------
```

BPX1SEU (seteuid) example

The following code sets the effective user ID of the invoker to 1. For the callable service, see “seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID” on page 672. AMODE 64 callers use “BPX4SEU (seteuid) example” on page 1276.

```
MVC   USERID,=XL4'00000001' Value of new effective user ID
SPACE ,
CALL  BPX1SEU,              Set effective user ID            +
      (USERID,              Input: User ID                   +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SF (send_file) example

The following code create a parameter list to send the contents of the specified file to the designated socket. to 1. For the callable service, see “send_file (BPX1SF, BPX4SF) — Send a file on a socket” on page 643. AMODE 64 callers use “BPX4SF (send_file) example” on page 1276.

```
         LA    R5,BUFFERA
         ST    R5,BUFA
         USING SFPL,R5
         XC    SFPL(SFPL#LENGTH),SFPL  Initialize to nulls (required)
* NULLS= no header, no trailer, start at offset 0
*        MVC   SFFileDes,...                   Read from file
*        MVC   SFSocketDes,...                 Write to Socket
         MVC   SFFileBytesH,=XL4'FFFFFFFF'     To file end
         MVC   SFFileBytesL,=XL4'FFFFFFFF'     To file end
         OI    SFflagByte4,SF_Close   Close socket after write
         SPACE ,
         CALL  BPX1SF,               Send_file                       +
               (=A(SFPL#LENGTH),     Input: Length of BPXYSFPL       +
               BUFA,                 Input: ->SFPL                   +
               RETVAL,               Return value: 0 or -1           +
               RETCODE,              Return code                     +
               RSNCODE),             Reason code                     +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SGE (setgrent) example

The following code resets the group database to the beginning, so that a subsequent BPX1GGE call will restart the group database search from the first entry. For the callable service, see “setgrent (BPX1SGE, BPX4SGE) — Reset the group database” on page 677. AMODE 64 callers use “BPX4SGE (setgrent) example” on page 1277.

```
         CALL  BPX1SGE,              Reset the group database        +
               (RETVAL),             Return value: 0                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SGI (setgid) example

The following code sets the real, effective, and save group IDs to 1. For the callable service, see “setgid (BPX1SGI, BPX4SGI) — Set the group ID” on page 674. AMODE 64 callers use “BPX4SGI (setgid) example” on page 1277.

```
         MVC   USERID,=XL4'00000001' Value of new group user ID
         SPACE ,
         CALL  BPX1SGI,              Set group ID                    +
               (GROUPID,             Input: Group ID                 +
               RETVAL,               Return value: 0 or -1           +
               RETCODE,              Return code                     +
               RSNCODE),             Reason code                     +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SGQ (sigqueue) example

The following code queues a signal (SIGUSR1#) to the process specified by PROCID with a signal value of 0. For the callable service, see “sigqueue (BPX1SGQ, BPX4SGQ) — Queue a signal to a process” on page 760. AMODE 64 callers use “BPX4SGQ (sigqueue) example” on page 1277.

```
SPACE ,
CALL  BPX1SGQ,              Queue a signal to a process       +
      (PROCID,              Input: Process ID                 +
      =A(SIGUSR1#),         Input: Signal            BPXYSIGH +
      =A(0),                Input: Signal value               +
      =A(0),                Input: Signal options             +
      RETVAL,               Return value: -1 or 0             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SGR (setgroups) example

The following code sets the supplementary group id list to the three gids (00000001, 000000002, 00000003) in BUFFERA. For the callable service, see "setgroups (BPX1SGR, BPX4SGR) — Set the supplementary group IDs list" on page 678. AMODE 64 callers use "BPX4SGR (setgroups) example" on page 1277.

```
LA    R15,BUFFERA
ST    R15,BUFA
MVC   BUFFERA(12),=XL12'000000010000000200000003'
SPACE ,
CALL  BPX1SGR,              Set supplementary groups list     +
      (=A(3),               Input: number of sgids in list    +
      BUFA,                 Input: address of sgids list      +
      RETVAL,               Return value: -1 or 0             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SGT (semget) example

The following code creates a private set of 10 semaphores. For the callable service, see "semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores" on page 631. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037. AMODE 64 callers use "BPX4SGT (semget) example" on page 1278.

```
MVC   KEY(4),=A(IPC_PRIVATE)  Local to this family
MVI   S_TYPE,IPC_CREAT+IPC_EXCL   Must not already exist
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
MVC   NUMB_SEMS(4),=A(10)         10 semaphores this set
SPACE ,
CALL  BPX1SGT,              Create a set of semaphores        +
      (KEY,                 Input: Semaphore key              +
      NUMB_SEMS,            Input: Number semaphores in set   +
      S_MODE,               Input: Flags    BPXYMODE / BPXYIPC+
      RETVAL,               Return value: -1 or Semaphore ID  +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on semget failure
ST    R15,SEM_ID            Store SEM_ID associated with key
```

BPX1SHT (shutdown) example

The following code issues a shutdown to stop socket writes to this socket connection. SOCKDESC was returned from a previous call to BPX1SOC. For the callable service, see "shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection" on page 743. AMODE 64 callers use "BPX4SHT (shutdown) example" on page 1278.

```
SPACE ,
CALL  BPX1SHT,              Shutdown communication           +
      (SOCKDESC,            Input: Socket Descriptor         +
      SOCK#SHUTDOWNWRITE,   Input: How - shutdown writes     +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SIA (sigaction) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 64 callers use "BPX4SIA (sigaction) example" on page 1278.

```
XC    NEWMASK,NEWMASK       Don't block additional signals
LA    R15,NCATCHER          New catcher (NCATCHER=0,1  ->)
ST    R15,NEWHANDL
LA    R15,OCATCHER          Old catcher (NCATCHER=0,1  ->)
ST    R15,OLDHANDL
SPACE ,
CALL  BPX1SIA,              Examine or change signal action   +
      (=A(SIGALRM#),        Input: Signal constant   BPXYSIGH +
      NEWHANDL,             Input: 0, ->0, ->1 or ->catcher   +
      NEWMASK,              Input: 64Bit mask of signals      +
      =A(0),                Input: Action, BPXYSIGH           +
      OLDHANDL,             0, ->XL4 (return 0, 1 ->catcher)  +
      OLDMASK,              64 bit mask of signals            +
      OLDFLAGS,             Action, BPXYSIGH                  +
      =A(0),                Data passed to signal routine     +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SIN (server_init) example

The following code connects a server address space to WLM as a server manager for the WEB subsystem type, WEB1 subsystem name, and IMWHTTP application environment. For the callable service, see "server_init (BPX1SIN, BPX4SIN) — Server initialization" on page 656. AMODE 64 callers use "BPX4SIN (server_init) example" on page 1279.

```
MVC   SUBSYSTYPE,=CL4'WEB '  WEB Subsystem Type
MVC   SUBSYSNAME,=CL8'WEB1    '  WEB1 Subsystem Name
MVC   APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
LA    R15,=F'7'             R15 = 7
ST    R15,PARALLELEU        7 Parallel Execution Units
SPACE ,
CALL  BPX1SIN,              Server_init                       +
      (=A(SRV_SERVERMGR),   Input: Manager Type (Server Mgr)  +
```

```
      SUBSYSTYPE,           Input: Subsystem Type            +
      SUBSYSNAME,           Input: Subsystem Type            +
      APPLENV,              Input: Application Environment   +
      PARALLELEU,           Input: Parallel Eu               +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1SIP (sigpending) example

The following code retrieves the mask used for pending and blocked signals. For the callable service, see "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755. AMODE 64 callers use "BPX4SIP (sigpending) example" on page 1279.

```
CALL  BPX1SIP,              Determine pending signals        +
      (SIGRET,              Signal mask return area (XL8)    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SLK (shmem_lock) example

The following code initializes a shared memory resident lock. For the callable service, see "shmem_lock (BPX1SLK, BPX4SLK) — Shared memory lock service" on page 724. AMODE 64 callers use "BPX4SLK (shmem_lock) example" on page 1280.

```
XR    R15,R15               R15 = 0
ST    R15,LOCKATTRADDR      No lock attribute Data
SPACE ,
CALL  BPX1SLK,              shmem_lock                       +
      (=A(SLK_INIT),        INPUT: Function Code (Init)      +
      =A(SLK_NORMAL),       INPUT: Request Type (Normal)     +
      =A(SLK_SHARED),       INPUT: Lock Type (Shared)        +
      LOCKADDR,             INPUT: ->user lockword (shared mem+
      LOCKATTRADDR,         INPUT: Address of lock attr area +
      LOCKTOKENADDR,        INPUT: Address of Lock Token     +
      RETVAL,               Return value: >=0 or -1          +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1SLP (sleep) example

The following code suspends running for 8 seconds or until a signal is delivered (whichever comes first). For the callable service, see "sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time" on page 771. AMODE 64 callers use "BPX4SLP (sleep) example" on page 1280.

```
MVC   SECONDS,=F'8'         8 seconds
SPACE ,
CALL  BPX1SLP,              Temporarily suspend execution    +
```

```
              (SECONDS,            Input: Sleep interval in seconds  +
              RETVAL),             Return value: 0 or sleep time     +
              VL,MF=(E,PLIST)      ---------------------------------
```

BPX1SMF (smf_record) example

The following code tests whether SMF recording is active for a specified SMF record type, and if it is, writes an SMF record. For the callable service, see "smf_record (BPX1SMF, BPX4SMF) — Write an SMF record" on page 774. AMODE 64 callers use "BPX4SMF (smf_record) example" on page 1280.

```
         MVC   SMF_TYPE,=F'108'      Set SMF record type
         MVC   SMF_SUBTYPE,=F'0'     Set SMF record subtype
         MVC   BUFLENA,=F'0'         Set SMF record length
         MVC   BUFA,=F'0'            Zero SMF record address
         CALL  BPX1SMF,              smf_record                        +
               (SMF_TYPE,            SMF record type                   +
               SMF_SUBTYPE,          SMF record subtype                +
               BUFLENA,              SMF record length                 +
               BUFA,                 SMF record address set to zero    +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       --------------------------------
         ICM   R15,B'1111',RETVAL    Test return value
         BNZ   QUIT                  Not recording or error, quit
         SPACE ,
         MVI   BUFFERA,C' '
         MVC   BUFFERA+1(255),BUFFERA Clear SMF record
         MVI   BUFFERA+1,100         Set length in SMF header
         MVI   BUFFERA+5,108         Set SMF type in SMF header
         MVC   BUFFERA+18(16),=CL16'Here is the data'  Set SMF record
         MVC   SMF_TYPE,=F'108'      Set SMF record type
         MVC   SMF_SUBTYPE,=F'0'     Set SMF record subtype
         MVC   BUFLENA,=F'100'       Set SMF record length
         LA    R15,BUFFERA
         ST    R15,BUFA              Set SMF record address
         CALL  BPX1SMF,              smf_record                        +
               (SMF_TYPE,            SMF record type                   +
               SMF_SUBTYPE,          SMF record subtype                +
               BUFLENA,              SMF record length                 +
               BUFA,                 SMF record address                +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       --------------------------------
QUIT     EQU  *
```

BPX2SMS (sendmsg) example

The following code sends a message on a socket. SOCKDESC was returned from a previous call to BPX1SOC. For the callable service, see "sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket" on page 648. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043, "BPXYIOV — Map the I/O vector structure" on page 986, and "BPXYMSGH — Map the message header" on page 999. AMODE 64 callers use "BPX4SMS (sendmsg) example" on page 1281.

```
         XC    MSGH(MSGH#LENGTH),MSGH  Clear msgh
         LA    R2,SOCKADDR
         ST    R2,MSGHNAMEPTR        Store the address of sockaddr
         LA    R2,SOCK#LEN+SOCK_SUN#LEN
         ST    R2,MSGHNAMELEN
```

BPX2SMS (sendmsg) example

```
         LA    R2,IOV
         ST    R2,MSGHIOVPTR
         MVI   MSGHIOVNUM,1
*
         LA    R2,BUFFERA
         ST    R2,IOV_BASE
         LA    R2,16
         ST    R2,IOV_LEN
         MVC   BUFFERA(16),=CL16'Here is the data'
*
         CALL  BPX2SMS,              Send a message on a socket        +
               (SOCKDESC,            Input: Socket Descriptor          +
               MSGH,                 Input: Address of BPXYMSGH        +
               MSG_FLAGS,            Input: Flags                      +
               PRIMARYALET,          Input: Alet of the iov            +
               PRIMARYALET,          Input: Alet of the buffers in iov +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SND (send) example

The following code issues a send for a socket. SOCKDESC was returned previously from a call to BPX1SOC. For the callable service, see "send (BPX1SND, BPX4SND) — Send data on a socket" on page 640. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 64 callers use "BPX4SND (send) example" on page 1282.

```
         MVC   BUFLENA,=F'16'
         MVC   BUFFERA(16),=CL16'Here is the data'
         SPACE ,
         CALL  BPX1SND,              Send data on a socket             +
               (SOCKDESC,            Input: Socket Descriptor          +
               =A(L'BUFFERA),        Input: Length of input buffer     +
               BUFFERA,              Input: input buffer               +
               PRIMARYALET,          Input: Alet of input buffer       +
               MSG_FLAGS,            Input: Flags                      +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SOC (socket or socketpair) example

The following code creates a pair of stream sockets in the AF_UNIX domain. For the callable service, see "socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets" on page 777. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4SOC (socket or socketpair) example" on page 1282.

```
         CALL  BPX1SOC,              Create a socket pair              +
               (=A(AF_UNIX),         Input: Domain of AF_UNIX          +
               =A(SOCK#_STREAM),     Input: Type of socket stream      +
               =A(0),                Input: Protocol of 0              +
               =A(2),                Input: Dimension of 2 for pair    +
               SOCKETS,              Input: Socket vector for return   +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SOP (semop) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM_ID semaphore set. For the callable service, see "semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations" on page 636. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037. AMODE 64 callers use "BPX4SOP (semop) example" on page 1282.

```
LA    R5,BUFFERA            ->Utitliy buffer
ST    R5,BUFA
USING SEM_BUF_ELE,R5        ->1st SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(0)    Semaphore number 0
MVC   SEM_OP(2),=AL2(-1)              take the resource
MVC   SEM_FLG(2),=AL2(SEM_UNDO)       flags (undo,wait)
LA    R5,SEM#BUFLEN(,R5)    ->next SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(2)              number 2
MVC   SEM_OP(2),=AL2(1)               release the resource
MVC   SEM_FLG(2),=AL2(IPC_NOWAIT)     flags (nowait)
LA    R5,SEM#BUFLEN(,R5)              ->next SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(8)              number 8
MVC   SEM_OP(2),=AL2(0)               test for no resource
MVC   SEM_FLG(2),=AL2(0)              flags (wait)
SPACE ,
MVC   NUMB_SEM_OPS(4),=AL2(3) number of SEM_BUF_ELE in BUFFERA
SPACE ,
CALL  BPX1SOP,              Semaphore control operations      +
      (SEM_ID,              Input: Semaphore set ID           +
      BUFA,                 Input: ->SEM_BUF_ELE      BPXYSEM +
      NUMB_SEM_OPS,         Input: Action to take             +
      RETVAL,               Return value: 0, -1 or value      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SPB (queue_interrupt) example

The following code uses the queue_interrupt to return the last signal delivered to the signal interface routine (SIR). For the callable service, see "queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered" on page 568. AMODE 64 callers use "BPX4SPB (queue_interrupt) example" on page 1283.

```
CALL  BPX1SPB,              Queue the signal                  +
      (RETVAL,              Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SPE (setpwent) example

The following code resets the user database to the beginning, so that a subsequent BPX1GPE call will restart the user database search from the first entry. For the callable service, see "setpwent (BPX1SPE, BPX4SPE) — Reset the user database" on page 691. AMODE 64 callers use "BPX4SPE (setpwent) example" on page 1283.

```
CALL  BPX1SPE,              Reset the user database           +
      (RETVAL),             Return value: 0                   +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SPG (setpgid) example

The following code places the invoking process in its own process group (zeros indicate that the process group ID is to be set to the process ID). For the callable service, see “setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control” on page 686. AMODE 64 callers use “BPX4SPG (setpgid) example” on page 1283.

```
MVC   PROCID,=A(0)          Process ID - current to leader
MVC   GROUP,=A(0)           Group ID - current to leader
SPACE ,
CALL  BPX1SPG,              Set process group ID for Job Ctl  +
      (PROCID,              Input: Process to be placed in grp+
      GROUP,                Input: Target group               +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SPM (sigprocmask) example

The following code changes the signal mask to block signals 1 through 16. For the callable service, see “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask” on page 757. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 64 callers use “BPX4SPM (sigprocmask) example” on page 1284.

```
LA    R15,=XL8'FFFF000000000000'  Block signals 1 thru 16
ST    R15,NEWMASKA          New mask address
LA    R15,OLDMASK           Old signal mask
ST    R15,OLDMASKA          Old mask address
SPACE ,
CALL  BPX1SPM,              Examine or change signal mask     +
      (=A(SIG_BLOCK#),      Input: How parameter BPXYSIGH     +
      NEWMASKA,             Input: 0, ->CL8                   +
      OLDMASKA,             Input: 0 | ->returned mask        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SPN (spawn) example

The program ictasma located at **ict/bin** gets control as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. The file descriptor count is set to 0, indicating that the child shall inherit all of the parent's file descriptors. The inheritance area passed is set to all zeroes, indicating that the child shall inherit the parent's attributes without change. For the callable service, see “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 780. AMODE 64 callers use “BPX4SPN (spawn) example” on page 1284.

```
      MVC   BUFLENA,=F'16'
      MVC   BUFFERA(16),=C'/ict/bin/ictasma'
      MVC   ARGCNT,=F'3'
*                               First
      LA    R15,=F'4'               Length
      ST    R15,ARGLLST+00          Length parm list
      LA    R15,=CL4'WK18'          Argument
      ST    R15,ARGSLST+00          Argument address parm list
*                               Second
```

```
         LA    R15,=F'7'                Length
         ST    R15,ARGLLST+04           Length parm list
         LA    R15,=CL7'DEPT37A'        Argument
         ST    R15,ARGSLST+04           Argument address parm list
*                                  Third
         LA    R15,=F'22'               Length
         ST    R15,ARGLLST+08           Length parm list
         LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
         ST    R15,ARGSLST+08           Argument address parm list
*
         MVC   ENVCNT,=F'0'          Zero environment args passed
         MVC   ENVLENS,=F'0'         Addr of env. data length list
         MVC   ENVPARMS,=F'0'        Add of env. data
*
         MVC   FDCNT,=F'0'           Zero file descriptors passed
         MVC   FDLST,=F'0'           File Descriptor list
*
         XC    INHE(INHE#LENGTH),INHE  Clear Inheritance structure
         SPACE ,
         CALL  BPX1SPN,                                          +
               (BUFLENA,             Input: Pathname length      +
               BUFFERA,              Input: Pathname             +
               ARGCNT,               Input: Argument count       +
               ARGLLST,              Input: Argument length list +
               ARGSLST,              Input: Argument address list+
               ENVCNT,               Input: Environment count    +
               ENVLENS,              Input: Environment length list +
               ENVPARMS,             Input: Environment address list +
               FDCNT,                Input: File desriptor count +
               FDLST,                Input: File descriptor list +
               =A(INHE#LENGTH),      Input: Length of Inheritance area +
               INHE,                 Input: Inheritance area     +
               RETVAL,               Return value: Child PID or -1 +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SPR (setpeer) example

The following code issues a setpeer to set up the host address. For the callable service, see "setpeer (BPX1SPR, BPX4SPR) — Preset the peer address associated with a socket" on page 684. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 64 callers use "BPX4SPR (setpeer) example" on page 1285.

```
         CALL  BPX1SPR,              Select on a set of sockets  +
               (SOCKDESC,            Input: Socket Descriptor    +
               SOCK#LEN+SOCK_SUN#LEN, Input: Length of socket address +
               SOCKADDR,             Input: Socket address       +
               SOCK#SO_SET,          Input: Option - set the address +
               RETVAL,               Return value: 0 or -1       +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SPW (server_pwu) example

The following code puts work to the WLM work queue for the IMWHTTP application environment for transaction class A. For the callable service, see "server_pwu (BPX1SPW, BPX4SPW) — Server process work unit" on page 660. AMODE 64 callers use "BPX4SPW (server_pwu) example" on page 1285.

```
MVC   APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
MVC   TRXCLASS,=CL8'A      ' Transaction Class A
XR    R15,R15              R15 = 0
ST    R15,CLASSIFYLEN      No Classification Data
ST    R15,APPLDATALEN      No Application Data
ST    R15,FDLISTPTR        No File Descriptor List
SPACE ,
CALL  BPX1SPW,             Server_pwu                        +
      (=A(SRV_PUT_NEWWRK), Input: Function Code (Putwork)    +
      TRXCLASS,            Input: Transaction Class          +
      APPLENV,             Input: Application Environment    +
      CLASSIFYLEN,         Input: Classification Area Length +
      CLASSIFYAREAPTR,     Input: Classification Area Address+
      APPLDATALEN,         Input: Application Data Length    +
      APPLDATAPTR,         Input: Application Data Address   +
      FDLISTPTR,           Input: Mapped by BPXYSFDL         +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
L     R15,RETVAL           Load return value
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error
```

BPX1SPY (setpriority) example

The following code sets the CPU priority based on the input which and who values. The which value used is PRIO_PROCESS, which indicates that the priority is to be set by process ID. The who value used is 7, to set the priority for process ID 7. For the callable service, see “setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process” on page 688. AMODE 64 callers use “BPX4SPY (setpriority) example” on page 1286.

```
MVC   PROCID,=XL4'00000007' Process ID to set priority for
MVC   PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL  BPX1SPY,             Set priority value                +
      (=A(PRIO_PROCESS),   Input: Set by Process ID          +
      PROCID,              Input: PID to set priority for    +
      PRIORITY,            Input: Priority value to set to   +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      VL,MF=(E,PLIST)      ----------------------------------
L     R15,RETVAL           Load return value
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error
```

BPX1SRG (setregid) example

The following code sets the real and/or effective group IDs to 1. For the callable service, see “setregid (BPX1SRG, BPX4SRG) — Set the real and/or effective GIDs” on page 693. AMODE 64 callers use “BPX4SRG (setregid) example” on page 1286.

```
  MVC   RGID,=XL4'00000001'   Value of new real group ID
MVC   RGID,..               Group ID to be set from a getgid   73
  MVC   EGID,=XL4'00000001'   Value of new effective group ID
MVC   EGID,..               Group ID to be set from a getegid  73
  SPACE ,
  CALL  BPX1SRG,              Set Group IDs                     +
        (RGID,                Input: Real Group ID to be set    +
        EGID,                 Input: Eff. Group ID to be set    +
```

```
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1SRL (setrlimit) example

The following code sets the resource limits for the calling process based on the input resource value and the resource limits set in the input rlimit structure. The resource value is set to RLIMIT_CPU. The resource limits are set to RLIM_INFINITY. For the callable service, see "setrlimit (BPX1SRL, BPX4SRL) — Set resource limits" on page 698. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 64 callers use "BPX4SRL (setrlimit) example" on page 1286.

```
MVC   RESOURCE,=A(RLIMIT_CPU)    Value of resource
XC    RLIM_CUR_HW,RLIM_CUR_HW    Current limit highword (Zero)
XC    RLIM_MAX_HW,RLIM_MAX_HW    Maximum limit highword (Zero)
MVC   RLIM_CUR,=A(RLIM_INFINITY) Current limit
MVC   RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
SPACE ,
CALL  BPX1SRL,              Set resource limits              +
      (RESOURCE,            Input: resource                  +
      RLIMIT,               Structure, mapped by BPXYRLIM    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX1SRU (setreuid) example

The following code sets the real and/or effective user IDs to 1. For the callable service, see "setreuid (BPX1SRU, BPX4SRU) —Set the real and/or effective UIDs" on page 695. AMODE 64 callers use "BPX4SRU (setreuid) example" on page 1287.

```
  MVC   RUID,=XL4'00000001'   Value of new real user ID
MVC   RUID,..               User ID to be set from a getuid    73
  MVC   EUID,=XL4'00000001'   Value of new effective user ID
MVC   EUID,..               User ID to be set from a geteuid   73
  SPACE ,
  CALL  BPX1SRU,              Set user IDs                     +
        (RUID,                Input: Real User ID to be set    +
        EUID,                 Input: Eff. User ID to be set    +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SRX (srx_np) example

srx_np callable service sends or receives data on a socket using CSM buffers. The following example receives data into CSM buffers. The MSGXNAMEPTR is set up to point to a buffer to receive the source address of the data. The MSGXIOVX is an IVTBUFL structure, which describes an IOVX array in a CSM buffer. The IOVX array contains IVTBUFL structures, each of which describes a CSM buffer with data that was received. SOCKDESC is a socket descriptor that was returned from a previous call to either BPX1SOC or BPX1ACP. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and

"BPXYMSGX — Map the message header" on page 999. For the callable service, see "srx_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket" on page 799. AMODE 64 callers use "BPX4SRX (srx_np) example" on page 1287.

```
XC    MSGX(MSGX#LEN),MSGX   Clear msgx storage
LA    R2,SOCKADDR
ST    R2,MSGXNAMEPTR        Store the address of sockaddr
LA    R2,SOCK#LEN+SOCK_SIN#LEN
ST    R2,MSGXNAMELEN        Length of sockaddr buffer
SPACE ,
CALL  BPX1SRX,              Receive data in CSM buffers       +
      (SOCKDESC,            Input: Socket Descriptor          +
      MSGX_RECV,            Input: Direction                  +
      L'MSGX,               Input: Msghdrx length             +
      MSGX,                 Input: Msghdrx                    +
      RETVAL,               Return value: -1 or bytes read    +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SSI (setsid) example

The following code creates a session and a process group (and is the leader of both). For the callable service, see "setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID" on page 702. AMODE 64 callers use "BPX4SSI (setsid) example" on page 1288.

```
CALL  BPX1SSI,              Create session, set process grp ID+
      (RETVAL,              Return value: -1 or new session ID+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SSU (sigsuspend) example

The following code replaces the invoker's current mask to block signals 1 through 16 and suspend until a signal is delivered. For the callable service, see "sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered" on page 763. AMODE 64 callers use "BPX4SSU (sigsuspend) example" on page 1288.

```
MVC   WAITMASK(8),=XL8'FFFF000000000000'  Blocks 1 thru 16
SPACE ,
CALL  BPX1SSU,              Wait for a signal                 +
      (WAITMASK,            Input: Wait mask, XL8             +
      RETVAL,               Return value: -1 or not returned  +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1STA (stat) example

The following code obtains status about file **labrec/qual/current** . For the callable service, see "stat (BPX1STA, BPX4STA) — Get status information about a file by pathname" on page 805. For the data structure, see "BPXYSTAT — Map the response structure for stat" on page 1057. AMODE 64 callers use "BPX4STA (stat) example" on page 1288.

```
MVC   BUFFERA(19),=CL19'labrec/qual/current'
MVC   BUFLENA,=F'19'
```

```
SPACE ,
CALL  BPX1STA,             Get file status                  +
      (BUFLENA,            Input: Pathname length           +
      BUFFERA,             Input: Pathname                  +
      STATL,               Input: Length of buffer needed   +
      STAT,                Buffer, BPXYSTAT                 +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1STE (set_timer_event) example

The following code sets a timer event, which when it expires will post the ECB represented by THLITIMERECB. For the callable service, see "set_timer_event (BPX1STE, BPX4STE) — Set DIE-mode timer event" on page 707. AMODE 64 callers use "BPX4STE (set_timer_event) example" on page 1288.

```
CALL  BPX1STE,             Set timer event                  +
      (=A(2),              Input: Number of seconds         +
      =A(500000000),       Input: Number of nanoseconds     +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1STF (w_statvfs) example

The following code obtains information about file system TESTLIB.FILESYS1. For the callable service, see "w_statvfs (BPX1STF, BPX4STF) — Get the file system status" on page 926. For the data structure, see "BPXYSSTF — Map response structure for file system status" on page 1055. AMODE 64 callers use "BPX4STW (sigtimedwait) example" on page 1291.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX1STF,             Get file system status           +
      (FSNAME,             Input: File system name (44 char) +
      SSTFL,               Input: Length of BPXYSSTF        +
      SSTF,                Buffer, BPXYSSTF                 +
      RETVAL,              Return value: -1 or length status +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1STL (set_thread_limits) example

The following code sets the MAX_THREAD and MAX_THREAD_TASKS limits for pthread_created threads in the invoker's process. For the callable service, see "set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads" on page 704. AMODE 64 callers use "BPX4STL (set_thread_limits) example" on page 1289.

```
CALL  BPX1STL,             Set_thread_limits                +
      (=A(STL_SET_BOTH),   Input: action           BPXYCONS +
      =A(50),              Input: new task limit            +
      =A(100),             Input: new thread limit          +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1STO (sendto) example

The following code issues a sendto for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see “sendto (BPX1STO, BPX4STO) — Send data on a socket” on page 652. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043 and “BPXYMSGF — Map the message flags” on page 997. AMODE 64 callers use “BPX4STO (sendto) example” on page 1289.

```
MVC   BUFFERA(16),=CL16'Here is the data'
LA    R2,BUFFERA
ST    R2,IOV_BASE
MVI   IOV_LEN,16
SPACE ,
CALL  BPX1STO,              Send data to a socket             +
      (SOCKDESC,            Input: Socket Descriptor          +
      =A(L'BUFFERA),        Input: Length of the input buffer +
      BUFFERA,              Input: input buffer               +
      PRIMARYALET,          Input: Alet of the input buffer   +
      MSG_FLAGS,            Input: Flags                      +
      =A(L'SOCKADDR),       Input: Length of the socket addr  +
      SOCKADDR,             Input: The socket address         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1STR (setitimer) example

The following code returns the time remaining an alarm, or ITIMER_REAL as set by setitimer. For the callable service, see “setitimer (BPX1STR, BPX4STR) — Set the value of the iterval timer” on page 680. For the data structure, see “BPXYITIM — Map getitimer, setitimer structure” on page 990. AMODE 64 callers use “BPX4STR (setitimer) example” on page 1290.

```
LA    R15,2                 Initial value 2.5 seconds
ST    R15,ITIMISECONDS
L     R15,=A(500000)
ST    R15,ITIMIMICROSEC
L     R15,0                 No reload value
ST    R15,ITIMRSECONDS
ST    R15,ITIMRMICROSEC
LA    R15,ITIM              Output mapping structure
ST    R15,ITIMA             ->structure
CALL  BPX1STR,              Get process data                  +
      (=A(ITIMER_REAL),     Input: Relative process token     +
      ITIMA,                In : ->Buffer, mapped by BPXYITIM +
      ITIMA,                Out: ->Buffer, mapped by BPXYITIM +
      RETVAL,               Return value: -1, 0               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1STV (statvfs) example

The following code obtains information about the file system containing the file identified by pathname. For the callable service, see “statvfs (BPX1STV, BPX4STV) — Get the file system status” on page 809. For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1055. AMODE 64 callers use “BPX4STV (statvfs) example” on page 1290.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX1STV,              Get file system status           +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      SSTFL,                Input: Length of BPXYSSTF        +
      SSTF,                 Buffer, BPXYSSTF                 +
      RETVAL,               Return value: -1 or length status +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1STW (sigtimedwait) example

The following code will wait for signals 1-4 to arrive or 3 seconds, whichever occurs first. For the callable service, see "sigtimedwait (BPX1STW, BPX4STW) — Wait for a signal with a specified timeout" on page 766. AMODE 64 callers use "BPX4STW (sigtimedwait) example" on page 1291.

```
MVC   WAITMASK(8),=XL8'F000000000000000'  Signals 1-4
LA    R15,SIGINFO_T
ST    R15,SINFA
MVC   SECONDS,=F'3'         Wait three seconds
XC    NANOSECONDS,NANOSECONDS Zero nanoseconds
SPACE ,
CALL  BPX1STW,              Signal timed wait                +
      (WAITMASK,            Input: mask of signal to wait for +
      SINFA,                Input: address of siginfo_t area  +
      SIGINFO#LENGTH,       Input: length of siginfo_t area   +
      SECONDS,              Input: seconds to wait for sig    +
      NANOSECONDS,          Input: nanoseconds to wait for sig+
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SUI (setuid) example

The following code sets the real, effective, and saved user IDs to 1. For the callable service, see "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710. AMODE 64 callers use "BPX4SUI (setuid) example" on page 1291.

```
  MVC   USERID,=XL4'00000001' Value of new user ID
MVC   USERID,..             User ID to be set from a getuid    78
  SPACE ,
  CALL  BPX1SUI,              Set user ID                      +
        (USERID,              Input: User ID to be set         +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        VL,MF=(E,PLIST)       ---------------------------------
```

BPX1SWT (sigwait) example

The following code waits for an asynchronous signal, **SIGALRM** bit 14 in the mask. For the callable service, see "sigwait (BPX1SWT, BPX4SWT) — Wait for a signal" on page 769. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 64 callers use "BPX4SWT (sigwait) example" on page 1291.

```
MVC   WAITMASK(8),=XL8'0004000000000000'
SPACE ,
CALL  BPX1SWT,              Wait for asynchronous signal      +
      (WAITMASK,            Input: Signal mask SIGALRM        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SYC (sysconf) example

The following code gets the maximum number of children allowed by the configuration variable. For the callable service, see "sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options" on page 819. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4SYC (sysconf) example" on page 1292.

```
CALL  BPX1SYC,              Get configuration variable        +
      (=A(SC_CHILD_MAX),    Input: Config variable BPXYCONS   +
      RETVAL,               Return value: -1 or variable      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SYM (symlink) example

The following code creates a symbolic link **/sysaccts** for path name **/sys12/acctn**. For the callable service, see "symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name" on page 812. AMODE 64 callers use "BPX4SYM (symlink) example" on page 1292.

```
MVC   BUFFERA(12),=CL12'/sys12/acctn'
MVC   BUFLENA,=F'12'
MVC   BUFFERB(09),=CL09'/sysaccts'
MVC   BUFLENB,=F'09'
SPACE ,
CALL  BPX1SYM,              Create symbolic link to pathname  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      BUFLENB,              Input: Link name length           +
      BUFFERB,              Input: Link name                  +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1SYN (sync) example

The following code causes all information in memory that updates file systems to be scheduled for writing out to disk. For the callable service, see "sync (BPX1SYN, BPX4SYN) — Schedule file system updates" on page 818. AMODE 64 callers use "BPX4SYN (sync) example" on page 1292.

```
CALL  BPX1SYN,              Sync                              +
      (RETVAL,              Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1TAF (MVSThreadAffinity) example

The following code executes the assembler routine EXITRTN on another thread, identified by thread ID THID, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see "MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service" on page 427. AMODE 64 callers use "BPX4TAF (MVSThreadAffinity) example" on page 1292.

```
         MVC   EXITRTNA,=V(EXITRTN)  ->Routine address
*        MVC   EXITPLA,=A(EXITPARM)  ->Input parameter list
         SPACE ,
         CALL  BPX1TAF,                                          +
               (EXITRTNA,            Input: Routine address      +
               EXITPLA,              Input: Parm list address or 0 +
               THID,                 Input: Target pthread to run exit +
               RETVAL,               Return value: -1 or not return +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1TAK (takesocket) example

The following code takes a socket that was given by the program identified by CID (clientid). SOCKDESC and CID information are passed by the program that did the givesocket (BPX1GIV). SOCKDESC is the giver's descriptor. When takesocket completes successfully, RETVAL will contain the taker's new socket descriptor. For the callable service, see "takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program" on page 821. For the data structure, see "BPXYCID — Map the returning structure for getclientid()" on page 951. AMODE 64 callers use "BPX4TAK (takesocket) example" on page 1293.

```
         CALL  BPX1TAK,              take a socket from another program+
               (CID,                 Input: Clientid of giver    +
               SOCKDESC,             Input: Giver's socket descriptor  +
               RETVAL,               Return value: -1 or new descriptor+
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
         L  R2,RETVAL
         ST R2,SOCKDES2              Store the new socket descriptor
```

BPX1TDR (tcdrain) example

The following code waits until all output sent to the standard output file has been transmitted. For the callable service, see "tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted" on page 824. AMODE 64 callers use "BPX4TDR (tcdrain) example" on page 1293.

```
         CALL  BPX1TDR,              Wait for output transmittal    +
               (=A(STDOUT_FILENO),   Input: File descriptor         +
               RETVAL,               Return value: 0 or -1          +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               VL,MF=(E,PLIST)       ---------------------------------
```

BPX1TFH (tcflush) example

The following code flushes all the data in the standard input file. the callable service, see “tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal” on page 829. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1065. AMODE 64 callers use “BPX4TFH (tcflush) example” on page 1293.

```
CALL  BPX1TFH,             Line control flush               +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      =A(TCIFLUSH),        Input: Queue selector BPXYTIOS   +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1TFW (tcflow) example

The following code resumes data flow (TCION transmits a START character) on the standard input file. For the callable service, see “tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal” on page 826. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1065. AMODE 64 callers use “BPX4TFW (tcflow) example” on page 1294.

```
CALL  BPX1TFW,             Suspend or resume data flow      +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      =A(TCION),           Input: Action BPXYTIOS           +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1TGA (tcgetattr) example

The following code retrieves control information about the standard input file. For the callable service, see “tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal” on page 831. For the data structure, see “BPXYTIOS — Map the termios structure” on page 1065. AMODE 64 callers use “BPX4TGA (tcgetattr) example” on page 1294.

```
CALL  BPX1TGA,             Get a terminal control structure +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      TIOS,                Termio structure, BPXYTIOS       +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      VL,MF=(E,PLIST)      ---------------------------------
```

BPX1TGC (tcgetcp) example

The following code retrieves information about code page change notification (CPCN) capability and the BPXYTCCP structure. For the callable service, see “tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names” on page 833. For the data structure, see “BPXYTCCP — Map the terminal control code page structure” on page 1058. AMODE 64 callers use “BPX4TGC (tcgetcp) example” on page 1294.

```
CALL  BPX1TGC,             Get code page names              +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      =A(TCCP#LENGTH),     Input: Length of BPXYTCCP        +
```

```
      TCCP,               Output: Termcp structure BPXYTCCP +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      VL,MF=(E,PLIST)     ----------------------------------
```

BPX1TGP (tcgetpgrp) example

The following code gets the foreground process group ID associated with the controlling terminal. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see "tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID" on page 836. AMODE 64 callers use "BPX4TGP (tcgetpgrp) example" on page 1295.

```
CALL  BPX1TGP,            Get the foreground process grp ID +
      (=A(STDIN_FILENO),  Input: File descriptor            +
      RETVAL,             Return value -1, fgrd proc grp ID +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      VL,MF=(E,PLIST)     ----------------------------------
```

BPX1TGS (tcgetsid) example

The following code retrieves the process group ID of the session for which the terminal specified by file descriptor is the controlling terminal. For the callable service, see "tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal" on page 838. AMODE 64 callers use "BPX4TGS (tcgetsid) example" on page 1295.

```
CALL  BPX1TGS,            Get session process group ID      +
      (=A(STDIN_FILENO),  Input: File descriptor            +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      VL,MF=(E,PLIST)     ----------------------------------
```

BPX1TIM (times) example

The following code gathers selected times about the invoker's CPU utilization. For the callable service, see "times (BPX1TIM, BPX4TIM) — Get process and child process times" on page 856. For the data structure, see "BPXYTIMS — Map the response structure for times" on page 1064. AMODE 64 callers use "BPX4TIM (times) example" on page 1295.

```
CALL  BPX1TIM,            Process CPU times                 +
      (TIMS,              Input: Buffer           BPXYTIMS +
      RETVAL,             Return value: -1 or clock_t       +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      VL,MF=(E,PLIST)     ----------------------------------
```

BPX1TLS (pthread_security_np) example

The following code creates a thread-level security environment for the calling thread using the identity specified by the caller. For the callable service, see "pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create | delete thread-level security" on page 518. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4TLS (pthread_security_np) example" on page 1295.

```
MVC   IDENT,=CL8'USERID05'
MVC   PASSWORD,=CL7'MYPSWRD'
SPACE ,
CALL  BPX1TLS,              pthread_security_np              +
      (=A(TLS_CREATE_THREAD_SEC#), Input: Func_code  BPXYCONS +
      TLS_IDENTITY_USERID#, Input: Identity_type     BPXYCONS +
      =A(8),                Input: Identity length           +
      IDENT,                Input: Identity                  +
      =A(7),                Input: Password length           +
      PASSWORD,             Input: Password                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1TRU (truncate) example

The following code truncates the file described by **/somedir/somefile.c** to a length of 512 bytes. For the callable service, see "truncate (BPX1TRU, BPX4TRU) — Change the size of a file" on page 859. AMODE 64 callers use "BPX4TRU (truncate) example" on page 1296.

```
MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
MVC   BUFLENA,=F'20'
MVC   NEWLEN(8),=FL8'512'
SPACE ,
CALL  BPX1TRU,              Truncate a file                  +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      NEWLEN,               Input: Length to keep            +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      VL,MF=(E,PLIST)       ---------------------------------
```

BPX1TSA (tcsetattr) example

The following code turns off the HUPCL (hang up on last close) bit for the standard input file. For the callable service, see "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842. For the data structure, see "BPXYTIOS — Map the termios structure" on page 1065. AMODE 64 callers use "BPX4TSA (tcsetattr) example" on page 1296.

```
      NI    C_CFLAG+HUPCL_O,X'FF'-HUPCL   Turn off HUPCL
*     termios was retrived by a prior tcgetattr
      CALL  BPX1TSA,              Set terminal attributes          +
            (=A(STDIN_FILENO),    Input: File descriptor           +
            =A(TCSADRAIN),        Input: Action            BPXYTIOS +
            TIOS,                 Input: Terminos struct   BPXYTIOS +
            RETVAL,               Return value: 0 or -1            +
            RETCODE,              Return code                      +
            RSNCODE),             Reason code                      +
            VL,MF=(E,PLIST)       ---------------------------------
```

BPX1TSB (tcsendbreak) example

The following code requests that a break be sent to the standard input file. For the callable service, see "tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal" on page 840. AMODE 64 callers use "BPX4TSB (tcsendbreak) example" on page 1296.

```
CALL  BPX1TSB,              Send break condition to terminal  +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      =A(0),                Duration, not used in z/OS UNIX   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1TSC (tcsetcp) example

The following code sets code page names and Code Page Change Notification (CPCN) capability. For the callable service, see "tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names" on page 845. For the data structure, see "BPXYTCCP — Map the terminal control code page structure" on page 1058. AMODE 64 callers use "BPX4TSC (tcsetcp) example" on page 1297.

```
XC    TCCP(TCCP#LENGTH),TCCP  Clear area
OI    TCCPFLAGB4,TCCPFASTP  Set local translation
MVC   TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC   TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
SPACE ,
CALL  BPX1TSC,              Set code page names               +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      =A(TCCP#LENGTH),      Input: Length of BPXYTCCP         +
      TCCP,                 Termcp structure, BPXYTCCP        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1TSP (tcsetpgrp) example

The following code sets the controlling terminal's foreground process group to a new value. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see "tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID" on page 849. AMODE 64 callers use "BPX4TSP (tcsetpgrp) example" on page 1297.

```
MVC   PROCID,..             Process group ID set by setpgrp
SPACE ,
CALL  BPX1TSP,              Set foreground process group ID   +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      PROCID,               Input: Foreground process group ID+
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1TST (tcsettables) example

The following code sets code page names, conversion tables and Code Page Change Notification (CPCN) capability. For the callable service, see "tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables" on page 852. For the data structure, see "BPXYTCCP — Map the terminal control code page structure" on page 1058. AMODE 64 callers use "BPX4TST (tcsettables) example" on page 1297.

```
XC    TCCP(TCCP#LENGTH),TCCP  Clear area
OI    TCCPFLAGB4,TCCPFASTP  Set local translation
MVC   TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
```

```
MVC   TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
MVC   TBLSOURCE,..          Initialize source conversion table
MVC   TBLTARGET,..          Initialize target conversion table
SPACE ,
CALL  BPX1TST,              Set code page names and tables    +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      =A(TCCP#LENGTH),      Input: Length of BPXYTCCP         +
      TCCP,                 Termcp structure, BPXYTCCP        +
      TBLSOURCE,            Source conversion table           +
      TBLTARGET,            Target conversion table           +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal” on page 862.

```
MVC   BUFLENA,=A(1023)      Maximum pathname
CALL  BPX1TYN,              Determine terminal name           +
      (=A(STDERR_FILENO),   Input: File descriptor            +
      BUFLENA,              Length of buffer for pathname     +
      BUFFERA),             Buffer for pathname of terminal   +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX2TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal” on page 862. AMODE 64 callers use “BPX4TYN (ttyname) example” on page 1298.

```
MVC   BUFLENA,=A(1023)      Maximum pathname
CALL  BPX2TYN,              Determine terminal name           +
      (=A(STDERR_FILENO),   Input: File descriptor            +
      BUFLENA,              Length of buffer for pathname     +
      BUFFERA,              Buffer for pathname of terminal   +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code: describes why VAL=-1 +
      RSNCODE),             Reason code: qualifier on RETCODE +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UMK (umask) example

The following code changes the process's file mode creation mask (to user read, group execute, other execute). For the callable service, see “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 866. For the data structure, see “BPXYMODE — Map the mode constants of the file services” on page 996. AMODE 64 callers use “BPX4UMK (umask) example” on page 1298.

```
XC    S_MODE,S_MODE
MVI   S_MODE3,S_IXUSR+S_IXGRP+S_IXOTH  Search permission
SPACE
CALL  BPX1UMK,              Set file creation mask            +
      (S_MODE,              Input: Mode              BPXYMODE +
      RETVAL),              Return value: previous mode mask  +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UMT (umount) example

The following code removes virtual file system TESTLIB.FILESYS1 from the file tree. For the callable service, see "umount (BPX1UMT, BPX4UMT) — Remove a virtual file system" on page 867. For the data structure, see "BPXYMTM — Map the modes for mount and unmount" on page 1000. AMODE 64 callers use "BPX4UMT (umount) example" on page 1298.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC    MTM(MTM#LENGTH),MTM
MVI   MTM1,MTMUMOUNT        Unmount request
SPACE ,
CALL  BPX1UMT,              Remove a virtual file system      +
      (FSNAME,              Input: File system name (44 char) +
      MTM,                  Input: Flags, BPXYMTM             +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UNA (uname) example

The following code obtains information about the system on which the invoker is running. For the callable service, see "uname (BPX1UNA, BPX4UNA) — Obtain the name of the current operating system" on page 870. For the data structure, see "BPXYUTSN — Map the response structure for uname" on page 1068. AMODE 64 callers use "BPX4UNA (uname) example" on page 1299.

```
LA    R15,UTSN
ST    R15,UTSNA
SPACE ,
CALL  BPX1UNA,              Identify system                   +
      (UTSNL,               Input: Length of required buffer  +
      UTSNA,                Output: ->UTSN           BPXYUTSN +
      RETVAL,               Return value: -1 or >-1           +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UNL (unlink) example

The following code removes pathname **usr/dataproc/next.t** from the system. For the callable service, see "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872. AMODE 64 callers use "BPX4UNL (unlink) example" on page 1299.

```
MVC   BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC   BUFLENA,=F'19'
SPACE ,
CALL  BPX1UNL,              Remove a directory entry          +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UPT (unlockpt) example

The following code unlocks the slave pseudoterminal device associated with the master to which the file descriptor refers. For the callable service, see "unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal master/slave pair" on page 875. AMODE 64 callers use "BPX4UPT (unlockpt) example" on page 1299.

```
CALL  BPX1UPT,              Unlocks slave pty from master     +
      (MASTER_FD,           Input: File descriptor            +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UQS (unquiesce) example

The following code unquiesces TESTLIB.FILESYS1, making its files available for use again. For the callable service, see "unquiesce (BPX1UQS, BPX4UQS) — Unquiesce a file system" on page 877. For the data structure, see "BPXYMTM — Map the modes for mount and unmount" on page 1000. AMODE 64 callers use "BPX4UQS (unquiesce) example" on page 1300.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC    MTM(MTM#LENGTH),MTM   Zero MTM = don't force unquiesce
SPACE ,
CALL  BPX1UQS,              Unquiesce a file system           +
      (FSNAME,              Input: File system name (44 char) +
      MTM,                  Input: Flags, BPXYMTM             +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1UTI (utime) example

The following code changes the access and modification times of **/usr/private/workfile.t** to the current time. For the callable service, see "utime (BPX1UTI, BPX4UTI) — Set file access and modification times" on page 879. AMODE 64 callers use "BPX4UTI (utime) example" on page 1300.

```
MVC   BUFFERA(23),=CL23'/usr/private/workfile.t'
MVC   BUFLENA,=F'23'
MVC   NEWTIMES,=FL8'-1'     Current time
SPACE ,
CALL  BPX1UTI,              Set file access and modify times  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      NEWTIMES,             Input: Access/Modification time   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1WAT (wait) example

The following code waits for any of its children to end or stop. For the callable service, see "wait (BPX1WAT, BPX4WAT) — Wait for a child process to end" on page 882. For the data structure, see "BPXYWAST — Map the wait status word" on page 1069 and "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4WAT (wait) example" on page 1300.

```
LA    R15,WAST              Resolve address of STATUS
ST    R15,WASTA             Save address of STATUS
MVC   PROCID,=F'-1'         Wait for any child
SPACE ,
CALL  BPX1WAT,              Wait for a child process to end   +
      (PROCID,              Input: PID being waited on        +
      =A(WNOHANG),          Input: options       BPXYCONS     +
      WASTA,                ->Exit status field, BPXTWAST     +
      RETVAL,               Return value: -1, 0, child PID    +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
```

BPX1WLM (__WLM) example

The following code connects to WLM as a work manager for the WEB subsystem type and WEB1 subsystem name. For the callable service, see "__wlm (BPX1WLM, BPX4WLM) — WLM interface service" on page 916. AMODE 64 callers use "BPX4WLM (__WLM) example" on page 1301.

```
LA    R8,BUFFERA            Storage for _WWC
USING _WWC,R8               WLM_CONNECT_WORKMGR DSECT
ST    R8,INARGLISTPTR       -> _WWC list of parameters
MVC   SUBSYSTYPE,=CL4'WEB '  WEB Subsystem Type
MVC   SUBSYSNAME,=CL8'WEB1    '  WEB1 Subsystem Name
LA    R15,SUBSYSTYPE
ST    R15,_WWC_SUB_SYS      Pointer to Subsystem Type
LA    R15,SUBSYSNAME
ST    R15,_WWC_SUB_SYS_NM   Pointer to Subsystem Name
SPACE ,
CALL  BPX1WLM,              work_load_manager system call     +
      (=A(WLM_CONNECT_WORKMGR), Input: Fcn Codes in BPXYWLM   +
      INARGLISTPTR,         Input: ->list of parameters       +
      RETVAL,               Return value: Varies with fcn code+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      VL,MF=(E,PLIST)       ----------------------------------
DROP  R8
```

BPX1WRT (write) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). For the callable service, see "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928. AMODE 64 callers use "BPX4WRT (write) example" on page 1301.

```
*       MVC   FILEDESC,             File descriptor from open
        MVC   BUFLENA,=F'80'
        LA    R15,BUFFERA
        ST    R15,BUFA
        SPACE ,
        CALL  BPX1WRT,              Write to a file                  +
              (FILEDESC,            Input: File descriptor           +
              BUFA,                 Input: ->Buffer                  +
              PRIMARYALET,          Input: Buffer ALET               +
              BUFLENA,              Input: Number of bytes to write  +
              RETVAL,               Return value: -1 or bytes written +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              VL,MF=(E,PLIST)       ---------------------------------
```

BPX1WRV (writev) example

The following code issues a writev for a socket. SOCKDESC was returned from a previous call to either BPX1SOC or BPX1ACP. For the callable service, see "writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers" on page 933. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 64 callers use "BPX4WRV (writev) example" on page 1301.

```
         MVC   BUFFERA(16),=CL16'Here is the data'
         LA    R2,BUFFERA
         ST    R2,IOV_BASE
         MVI   IOV_LEN,16
*
         CALL  BPX1WRV,              Write from a vector of buffers    +
               (SOCKDESC,            Input: Socket Descriptor          +
               =A(1),                Input: Single element in iov      +
               IOV,                  Input: Iov containing info        +
               PRIMARYALET,          Input: Alet where iov resides     +
               PRIMARYALET,          Input: Alet of buffers for data   +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

BPX1WTE (wait extension) example

The following code uses the #WAIT3 function to wait for any of its children to end or stop. For the callable service, see "wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children" on page 885. For the data structures, see "BPXYWAST — Map the wait status word" on page 1069 and "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. and "BPXYCONS — Constants used by services" on page 952. AMODE 64 callers use "BPX4WTE (wait extension) example" on page 1302.

```
         LA    R15,WAST              Resolve address of WAST
         ST    R15,WASTA             Save address of WAST
         LA    R15,RUSAGE            Resolve address of RUSAGE
         ST    R15,RUSAGEA           Save address of RUSAGE
         SPACE ,
         CALL  BPX1WTE,              Wait for a child process to end   +
               (=A(#WAIT3),          Input: function      BPXYCONS     +
               0,                    Input: id type                    +
               0,                    Input: id                         +
               WASTA,                ->Exit status field, BPXTWAST     +
               =A(WNOHANG),          Input: options       BPXYCONS     +
               RUSAGEA,              ->Rusage structure,  BPXYRLIM     +
               RETVAL,               Return value: -1, 0, child PID    +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL,MF=(E,PLIST)       ----------------------------------
```

Reentrant return linkage

```
         XR    R15,R15               Zero return code
         L     R0,@SIZEDAT           Size this program's getmain area
         LR    R1,R13                R1 -> this program's getmain area
         L     R13,@BACK             R2 -> caller's save area
         DROP  R13
         FREEMAIN RU,LV=(0),A=(1)
```

```
        L     R14,12(,R13)          Restore caller's R14
        LM    R0,R12,20(R13)        Restore caller's R0-R12
        BSM   0,R14                 Branch back to caller

        SPACE ,  * * * * * * * * * *.* Program constants * * * * * * *
@SIZEDAT DC   A(@ENDSTOR-@STORE)    Size of this getmain storage
MNTEL    DC   A(MNTE#LENGTH+MNTEH#LENGTH)
*                                   Length of MNTEH and 1 MNTE area
PGPSL    DC   A(PGPS#LENGTH)        Length of PGPS structure
RMONL    DC   A(RMON#LENGTH)        Length of RMON structure
SSTFL    DC   A(SSTF#LENGTH)        Length of SSTF structure
STATL    DC   A(STAT#LENGTH)        Length of STAT structure
UTSNL    DC   A(UTSN#LENGTH)        Length of UTSN structure
        SPACE ,
PRIMARYALET  DC   A(0)              Primary ALET

* * * * * * * * * * * * * * * * * * *.* Structures requiring a USING  *
        BPXYDIRE DSECT=YES          Dictionary for readdir
        BPXYGIDN DSECT=YES          Group names
        BPXYGIDS DSECT=YES          Group IDs and member names
        BPXYOSMF DSECT=YES          Job step accounting for BPXESMF
        BPXYPOLL DSECT=YES          Poll syscall parameters (I/O)
        BPXYPGTH DSECT=YES          Mapping for __getthent data
        BPXYPPSD DSECT=YES          Signal data area (R1 in SIR)
        BPXYSEL  DSECT=YES          Select options
        BPXYSFPL DSECT=YES          Send_file parameter list
        BPXYTHDQ DSECT=YES          Data structure for BPX1PQG
        BPXYWLM  ,                  Work load manager
* * * * * * * * * * * * * * * * * * *.* EQUates * * * * * * * *
* With EQUate only macros, DSECT= is allowed but is ignored
        BPXYCONS ,                  z/OS UNIX constants
        BPXYCW   ,                  Serialization constants
        BPXYERNO LIST=NO            Errno, Errnojr constants
        BPXYFTYP ,                  File type constants
        BPXYPCF  ,                  Command, pathconf constants
        BPXYSEEK ,                  lseek constants
        BPXYSIGH ,                  Signal constants

* * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                   Standard 72-byte save area
         DS    A
@BACK    DS    A                    Back to caller's save area
@FORWARD DS    A                    Forwards to callee's save area
         DS    15A                  Regs 14,15,0-12

* * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                   Standard 72-byte save area
         DS    A
@BACK    DS    A                    Back to caller's save area
@FORWARD DS    A                    Forwards to callee's save area
         DS    15A                  Regs 14,15,0-12

* * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                   Standard 72-byte save area
         DS    A
@BACK    DS    A                    Back to caller's save area
@FORWARD DS    A                    Forwards to callee's save area
         DS    15A                  Regs 14,15,0-12

* * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                   Standard 72-byte save area
```

Reentrant return linkage

```
         DS    A
@BACK    DS    A                   Back to caller's save area
@FORWARD DS    A                   Forwards to callee's save area
         DS    15A                 Regs 14,15,0-12

* * * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                  Standard 72-byte save area
         DS    A
@BACK    DS    A                   Back to caller's save area
@FORWARD DS    A                   Forwards to callee's save area
         DS    15A                 Regs 14,15,0-12

* * * * * * * * * * * * * * * * * * * *.* Standard linkage save area  * *
@STORE   DSECT ,
@SAVE00  DS    0D                  Standard 72-byte save area
         DS    A
@BACK    DS    A                   Back to caller's save area
@FORWARD DS    A                   Forwards to callee's save area
         DS    15A                 Regs 14,15,0-12
```

Appendix E. Callable services examples—AMODE 64

For an example using nonreentrant code, see “Example of nonreentrant entry linkage—AMODE 64” on page 1309. These examples follow the rules of reentrancy. They use DSECT=NO and place the variables in the program's dynamic storage DSECT, which is allocated upon entry.

The examples are arranged alphabetically and have references to the mapping macros they use. The declaration for all local variables used in the examples follows the examples.

Reentrant entry linkage

This entry linkage is reentrant and saves the caller's registers, allocates a save area and dynamic storage, and establishes program and dynamic storage base registers. This entry linkage is paired with the return linkage that is located at the end of the executable program; see “Reentrant return linkage” on page 1302. For an example of nonreentrant entry and return linkage, see “Example of nonreentrant entry linkage—AMODE 31” on page 1307.

```
         TITLE 'Alphabetical syscall of z/OS UNIX callable services'
BPXB1SM1 CSECT ,                    Reentrant entry linkage
BPXB1SM1 AMODE 64
         SYSSTATE AMODE64=YES
BPXB1SM1 RMODE ANY
@ENTRY0  J     @ENTRY1              Branch around program header
         DC    C'BPXB1SM4 - Reentrant callable service examples'
         DS    0H                   Ensure half word boundary
@ENTRY1  STMG  R14,R12,12(R13)      Save caller's registers
         LGR   R2,R13               Hold address of caller's area
         LGR   R3,R1                Hold parameter register
         LGR   R12,R15              R12 program base register
         LA    R11,2048(,R12)       Second program base register
         LA    R11,2048(,R11)       Second program base register
         LA    R9,2048(,R11)        Third  program base register
         LA    R9,2048(,R9)         Third  program base register
         LA    R4,2048(,R9)         Fourth program base register
         LA    R4,2048(,R4)         Fourth program base register
         LA    R7,2048(,R4)         Fifth  program base register
         LA    R7,2048(,R7)         Fifth  program base register
         USING @ENTRY0,R12,R11,R9,R4,R7 Program addressability
         L     R0,@SIZEDAT          Size this program's getmain area
         GETMAIN RU,LV=(0)          Getmain storage
         LGR   R13,R1               R13 -> this program's save area
         LA    R10,2048(,R13)       Second getmain base register
         LA    R10,2048(,R10)       Second getmain base register
         LA    R6,2048(,R10)        Third  getmain base register
         LA    R6,2048(,R6)         Third  getmain base register
         USING @STORE,R13,R10,R6    Getmain addressability
         STG   R2,@BACK             Save caller's save area pointer
         STG   R13,136(,R2)         Give caller our save area
         LR    R1,R3                Restore parameter register
@ENTRY2  EQU   *  *  *  *  *  *  *  End of the entry linkage code
         SPACE ,
PSEUDO   EQU   *                    Dummy label used throughout
```

BPX4ACC (access) example

The following code determines if file **/usr/inv/network.t** can be accessed. For the callable service, see “access (BPX1ACC, BPX4ACC) — Determine if a file can be accessed” on page 23. For the data structure, see “BPXYACC — Map flag values for access” on page 945. AMODE 31 callers use “BPX1ACC (access) example” on page 1124.

```
MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
MVC   BUFLENA,=F'18'
XC    ACC(ACC#LENGTH),ACC
MVI   ACCINTENTFLAGS,ACC_R_OK+ACC_W_OK  Read and write access
SPACE ,
CALL  BPX4ACC,              Determine accessibility of a file +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      ACC,                  Input: Access, BPXYACC           +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Set condition code for RETVAL
BZ    PSEUDO                Branch if RETVAL is zero
CLC   RETCODE,=A(EACCES)    Compare RETCODE to EACCES
BE    PSEUDO                Branch if access denied
```

BPX4ACK (auth_check_resource_np) example

The following code determines if user 'JOEUSER' has UPDATE access to the FACILITY class profile 'TEST.THIS.PROFILE'. For the callable service, see “auth_check_resource_np (BPX1ACK, BPX4ACK) — Determine a user's access to a RACF-protected resource” on page 66. AMODE 31 callers use “BPX1ACK (auth_check_resource_np) example” on page 1124.

```
MVI   CELLUUID,X'00'
MVI   PRINUUID,X'00'
MVC   USERNLEN,=F'7'
MVC   USERNAME(7),=CL7'JOEUSER'
MVC   CLSLEN,=F'8'
MVC   CLS(8),=CL8'FACILITY'
MVC   ENTLEN,=F'17'
MVC   ENT(17),=CL17'TEST.THIS.PROFILE'
SPACE ,
CALL  BPX4ACK,              Determine access to a resource    +
      (CELLUUID,            Input: Cell UUID                  +
      PRINUUID,             Input: Principal UUID             +
      USERNLEN,             Input: Userid length              +
      USERID,               Input: Userid                     +
      CLSLEN,,              Input: Class length               +
      CLS,                  Input: Class                      +
      ENTLEN,               Input: Entity length              +
      ENT,                  Input: Entity                     +
      =A(ACK_UPDATE#),      Input: Access type to check for   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4ACP (accept) example

The following code does an accept to accept a connect request from a client. SOCKDESC was previously set by a call to BPX4SOC. A bind and a listen must also have been previously done. The SOCKADDR was built by the call to BPX4BND. For the callable service, see "accept (BPX1ACP, BPX4ACP) — Accept a connection request from a client socket" on page 15. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1ACP (accept) example" on page 1125.

```
CALL  BPX4ACP,              Accept a socket connect request   +
      (SOCKDESC,            Input: Socket descriptor          +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr     +
      SOCKADDR,             Input: Sockaddr structure         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
L  R2,RETVAL
ST R2,SOCKDES2              Store the new socket descriptor
```

BPX4AIO (asyncio) example

The following code will accept the next conversation. For the callable service, see "asyncio (BPX1AIO, BPX4AIO) — Asynchronous I/O for sockets" on page 31. AMODE 31 callers use "BPX1AIO (asyncio) example" on page 1125.

```
XC    AIO(AIO#LENGTH),AIO   Null AIO control block
MVC   AIOCMD,=A(AIO#ACCEPT) Command = Accept
MVC   AIOFD,FILEDESC        File descriptor
MVC   AIONOTIFYTYPE,=AL2(AIO#MVS)  Notify type = MVS
XC    ECB01,ECB01           ECB = 0
LA    R15,ECB01             ECB Address
ST    R15,AIOECBPTR         Null AIO control block
MVC   AIOSOCKADDRLEN,=A(SOCK#LEN)
LA    R15,SOCKADDR          From recvform (see BPX4RFM)
STG   R15,AIOSOCKADDRPTR
SPACE ,
CALL  BPX4AIO,              Asynchronous I/O for Sockets      +
      (=A(AIO#LENGTH),      Input: Time before SIGAIOM        +
      AIO,                  Input: Time before SIGAIOM        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4ALR (alarm) example

The following code schedules an alarm in 5 seconds. For the callable service, see "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29. AMODE 31 callers use "BPX1ALR (alarm) example" on page 1125.

```
MVC   SECONDS,=F'5'
SPACE ,
CALL  BPX4ALR,              Schedule Alarm                    +
      (SECONDS,             Input: Time before SIGALRM        +
      RETVAL),              Return value: 0 or -1             +
      MF=(E,PLIST)          ----------------------------------
```

BPX4ANR (accept_and_recv) example

The following code schedules an alarm in 5 seconds. For the callable service, see "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29. AMODE 31 callers use "BPX1ALR (alarm) example" on page 1125.

```
MVC   SECONDS,=F'5'
SPACE ,
CALL  BPX4ALR,              Schedule Alarm                  +
      (SECONDS,             Input: Time before SIGALRM      +
      RETVAL),              Return value: 0 or -1           +
      MF=(E,PLIST)          ---------------------------------
```

BPX4ASP (aio_suspend) example

The following code will wait up to 10 seconds for one of the events specified in the AIOCB. For the callable service, see "aio_suspend (BPX1ASP, BPX4ASP) — Wait for an asynchronous I/O request" on page 26. AMODE 31 callers use "BPX1ASP (aio_suspend) example" on page 1126.

```
LA    R15,AIO
STG   R15,ARGSLST
MVC   ARGCNT,=F'1'
MVC   SECONDS,=F'10'
XC    NANOSECONDS,NANOSECONDS
SPACE ,
CALL  BPX4ASP,              Suspend for an aio request      +
      (ARGSLST,             Input: List of pointers to AIOCBs +
      ARGCNT,               Input: Count of pointers in list  +
      SECONDS,              Input: Seconds to wait          +
      NANOSECONDS,          Input: Nanoseconds to wait      +
      RETVAL,               Return value: 0 or -1           +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4ATM (attach_execmvs) example

The following code invokes program APPL92 on a subtask and as a child process of the caller, passing the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine associated with program APPL92. For the callable service, see "attach_execmvs (BPX1ATM, BPX4ATM) — Attach an MVS program" on page 59. AMODE 31 callers use "BPX1ATM (attach_execmvs) example" on page 1127.

```
MVC   PGMNAMEL,=F'6'
MVC   PGMNAME(06),=CL6'APPL92'
MVC   BUFLENA,=F'24'
MVC   BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE ,
CALL  BPX4ATM,              Invoke a MVS program            +
      (PGMNAMEL,            Input: Length of program name   +
      PGMNAME,              Input: Program name             +
      BUFLENA,              Input: Length of program argument +
      BUFFERA,              Input: Program argument         +
      =AD(0),               Input: Exit routine address or 0  +
      =AD(0),               Input: Exit Parm list address or 0+
      RETVAL,               Return value: Child PID Or -1   +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4ATX (attach_exec) example

The program ictasma located at **ict/bin** gets control on a subtask and as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see "attach_exec (BPX1ATX, BPX4ATX) — Attach a z/OS UNIX program" on page 50. AMODE 31 callers use "BPX1ATX (attach_exec) example" on page 1127.

```
         MVC   BUFLENA,=F'16'
         MVC   BUFFERA(16),=C'/ict/bin/ictasma'
         MVC   ARGCNT,=F'3'
*                                  First
         LA    R15,=F'4'               Length
         STG   R15,ARGLLST+00          Length parm list
         LA    R15,=CL4'WK18'          Argument
         STG   R15,ARGSLST+00          Argument address parm list
*                                  Second
         LA    R15,=F'7'               Length
         STG   R15,ARGLLST+08          Length parm list
         LA    R15,=CL7'DEPT37A'       Argument
         STG   R15,ARGSLST+08          Argument address parm list
*                                  Third
         LA    R15,=F'22'              Length
         STG   R15,ARGLLST+16          Length parm list
         LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
         STG   R15,ARGSLST+16          Argument address parm list
*
         MVC   ENVCNT,=F'0'          Number of env. data items passed
         MVC   ENVLENS,=FD'0'         Addr of env. data length list
         MVC   ENVPARMS,=FD'0'        Add of env. data
*
         MVC   EXITRTNA,=AD(EXITRTN)  ->exit routine
*        MVC   EXITPLA,=A(exit paramter list as expected by EXITRTN)
         SPACE ,
         CALL  BPX4ATX,                                          +
               (BUFLENA,             Input: Pathname length      +
               BUFFERA,              Input: Pathname             +
               ARGCNT,               Input: Argument count       +
               ARGLLST,              Input: Argument length list +
               ARGSLST,              Input: Argument address list+
               ENVCNT,               Input: Environment count    +
               ENVLENS,              Input: Environment length list+
               ENVPARMS,             Input: Environment address list+
               EXITRTNA,             Input: Exit routine address or 0+
               EXITPLA,              Input: Exit Parm list address or 0+
               RETVAL,               Return value: Child PID or -1 +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               MF=(E,PLIST)          ---------------------------------
```

BPX4BND (bind) example

The following code does a bind to associate a name with a socket. SOCKDESC was previously set by a call to BPX4SOC. For the callable service, see "bind (BPX1BND, BPX4BND) — Bind a unique local name to a socket descriptor" on page 71. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1BND (bind) example" on page 1128.

```
         SPACE ,
         MVI   SOCK_LEN,12           Store the length of the address
         MVI   SOCK_FAMILY,AF_UNIX   Set the domain to AF_UNIX
         MVC   SOCK_SUN_NAME(12),=CL12'/tmp/socket1'  Set the name
```

```
CALL  BPX4BND,              Bind a name to a socket          +
      (SOCKDESC,            Input: Socket Descriptor         +
      =A(SOCK#LEN+SOCK_SUN#LEN), Input: Length - Sockaddr    +
      SOCKADDR,             Input: Sockaddr structure        +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4BAS (bind with source address selection) example

The following code does a bind to associate the best source address for the provided destination IP address with a socket. SOCKDESC was previously set by a call to BPX4SOC. For the callable service, see "bind2addrsel (BPX1BAS, BPX4BAS) — Bind the socket descriptor to the best source address" on page 73. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043.

```
SPACE ,
                            Store the length of the address
MVI SOCK_LEN,=A(SOCK#LEN+SOCK_SIN6#LEN)
MVI SOCK_FAMILY,AF_INET6    Set the domain to AF_INET6
MVC SOCK_SIN6_ADDR,=XL16'00A100B200C300D400E500F61234ABCD'
CALL BPX4BAS,               Bind with source address selection+
     (SOCKDESC,             Input: Socket Descriptor         +
     =A(SOCK#LEN+SOCK_SIN6#LEN), Input:Length - Sockaddr     +
     SOCKADDR,              Input: Sockaddr structure        +
     RETVAL,                Return value: 0 or -1            +
     RETCODE,               Return code                      +
     RSNCODE),              Reason code                      +
     VL,MF=(E,PLIST)        ---------------------------------
```

BPX4CCA (cond_cancel) example

The following code demonstrates how to cancel a program's interest in events that were selected by a call to the cond_setup service. For the callable service, see "cond_cancel (BPX1CCA, BPX4CCA) — Cancel interest in events" on page 107. AMODE 31 callers use "BPXICCA (cond_cancel) example" on page 1129.

```
CALL  BPX4CCA,              Cancel cond_setup                +
      (RETVAL,              Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
*  The return value (RETVAL) does not matter.  When your program
*  receives control following the call to cond_cancel, it is no
*  longer eligible to receive event notifications via cond_post.
```

BPX4CCS (__console()) example

The following code sends a message to the console. For the callable service, see "__console() (BPX1CCS, BPX4CCS) — Communicate with console (modify/stop/WTO/DOM)" on page 124. For the data structure, see "BPXYCCA — Map input/output structure for __console()" on page 950. AMODE 31 callers use "BPX1CCS (__console()) example" on page 1129.

```
CALL  BPX4CCS,              Send msg to console              +
      (MSGATTRLEN,          Input: BPXYCCA length            +
      MSGATTR,              Input: BPXYCCA                   +
      MODSTRINGPTR,         Output: Modify msg from console  +
      MODIFYSTGLEN,         Output: Length of modify msg     +
      CONMSGTYPE,           Output: Console msg type         +
```

```
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4CHA (chaudit) example

The following code changes the audit flags for the file identified by pathname. For the callable service, see "chaudit (BPX1CHA, BPX4CHA) — Change audit flags for a file by path" on page 84. For the data structure, see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. AMODE 31 callers use "BPX1CHA (chaudit) example" on page 1129.

```
MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
MVC   BUFLENA,=F'18'
MVI   AUDTREADACCESS,AUDTREADFAIL
MVI   AUDTWRITEACCESS,AUDTWRITEFAIL
MVI   AUDTEXECACCESS,AUDTEXECFAIL
MVI   AUDTRSRV,0
SPACE ,
CALL  BPX4CHA,             Change audit                     +
      (BUFLENA,            Input: Pathname length           +
      BUFFERA,             Input: Pathname                  +
      AUDT,                Input: Audit flags, BPXYAUDT     +
      =F'0',               Input: 0 user, 1 security auditor +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4CHD (chdir) example

The following code changes the working directory for the task. For the callable service, see "chdir (BPX1CHD, BPX4CHD) — Change the working directory" on page 88. AMODE 31 callers use "BPX1CHD (chdir) example" on page 1130.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX4CHD,             Change working directory         +
      (BUFLENA,            Input: Pathname length           +
      BUFFERA,             Input: Pathname                  +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4CHM (chmod) example

The following code changes the file mode for the file identified by pathname. For the callable service, see "chmod (BPX1CHM, BPX4CHM) — Change the mode of a file or directory" on page 90. For the data structure, see "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 31 callers use "BPX1CHM (chmod) example" on page 1130.

```
MVC   BUFFERA(26),=CL26'newprogs/path/eightfold.c'
MVC   BUFLENA,=F'26'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       All read and write
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
```

```
      CALL  BPX4CHM,             Change File Modes                +
            (BUFLENA,            Input: Pathname length           +
            BUFFERA,             Input: Pathname                  +
            S_MODE,              Input: Mode, mapped by BPXYMODE  +
            RETVAL,              Return value: 0 or -1            +
            RETCODE,             Return code                      +
            RSNCODE),            Reason code                      +
            MF=(E,PLIST)         ---------------------------------
```

BPX4CHO (chown) example

The following code changes the owner of **/somedir/somefile.c** from the current owner to that specified by USERID and GROUPID. For the callable service, see "chown (BPX1CHO, BPX4CHO) — Change the owner or group of a file or directory" on page 93. AMODE 31 callers use "BPX1CHO (chown) example" on page 1130.

```
        MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
        MVC   BUFLENA,=F'20'
      MVC   USERID,..            New owner UID from stat
      MVC   GROUPID,..           New owner GID from stat
        SPACE ,
        CALL  BPX4CHO,             Change owner and group of a file  +
              (BUFLENA,            Input: Pathname length           +
              BUFFERA,             Input: Pathname                  +
              USERID,              Input: New owner UID             +
              GROUPID,             Input: New owner GID             +
              RETVAL,              Return value: 0 or -1            +
              RETCODE,             Return code                      +
              RSNCODE),            Reason code                      +
              MF=(E,PLIST)         ---------------------------------
```

BPX4CHP (chpriority) example

The following code changes the CPU priority based on the input which, who, and priority type values. The which value used is PRIO_PROCESS, indicating that the priority will be set by process ID. The who value used is 7, to set the priority for process ID 7. The priority type is CPRIO_ABSOLUTE, indicating that the priority will be set to the value specified, 1. For the callable service, see "chpriority (BPX1CHP, BPX4CHP) — Change the scheduling priority of a process" on page 97. AMODE 31 callers use "BPX1CHP (chpriority) example" on page 1131.

```
      MVC   PROCID,=XL4'00000007' Process ID to change priority for
      MVC   PRIORITY,=XL4'00000001' Priority value of 1
      SPACE ,
      CALL  BPX4CHP,             Change priority value            +
            (=A(PRIO_PROCESS),   Input: Set by Process ID         +
            PROCID,              Input: PID to set priority for   +
            =A(CPRIO_ABSOLUTE),  Input: Change by absolute value  +
            PRIORITY,            Input: Priority value to change to+
            RETVAL,              Return value: 0 or -1            +
            RETCODE,             Return code                      +
            RSNCODE),            Reason code                      +
            MF=(E,PLIST)         ---------------------------------
      L     R15,RETVAL           Load return value
      C     R15,=F'-1'           Test for -1 return
      BE    PSEUDO               Branch on error
```

BPX4CHR (chattr) example

The following code changes the attributes of **/somedir/somefile.c**. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see "chattr (BPX1CHR, BPX4CHR) — Change the attributes of a file or directory" on page 76. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948 and "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 31 callers use "BPX1CHR (chattr) example" on page 1131.

```
MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
MVC   BUFLENA,=F'20'
MVC   ATTID,=CL4'ATT '      Eye Catcher
MVC   ATTVERSION,=AL2(ATT#VER)  version
XC    S_MODE,S_MODE         Clear mode
MVI   S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC   ATTMODE,S_MODE        Move mode data to attribute       +
      structure
MVC   ATTUID,=F'7'          Specify new UID
MVC   ATTGID,=F'77'         Specify new GID
OI    ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG                     +
      Flag Mode, UID and GID changes
OI    ATTSETFLAGS2,ATTCTIMETOD                                +
      Set change time to current time
SPACE ,
CALL  BPX4CHR,              Change file attributes            +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      =A(ATT#LENGTH),       Input: BPXYATT length             +
      ATT,                  Input/output: BPXYATT             +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4CLD (closedir) example

The following code closes the directory identified by FILEDESC. For the callable service, see "closedir (BPX1CLD, BPX4CLD) — Close a directory" on page 105. AMODE 31 callers use "BPX1CLD (closedir) example" on page 1132.

```
MVC   FILEDESC,..           Directory descriptor from opendir
  SPACE ,
  CALL  BPX4CLD,              Close a directory               +
        (FILEDESC,            Input: Directory file descriptor +
        RETVAL,               Return value: 0 or -1           +
        RETCODE,              Return code                     +
        RSNCODE),             Reason code                     +
        MF=(E,PLIST)          --------------------------------
```

BPX4CLO (close) example

The following code closes the standard input file. For the callable service, see "close (BPX1CLO, BPX4CLO) — Close a file" on page 103. AMODE 31 callers use "BPX1CLO (close) example" on page 1132.

```
CALL  BPX4CLO,              Close a file                      +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      RETVAL,               Return value: 0 or -1             +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4CON (connect) example

The following code connects to a socket. SOCKDESC was returned by a previous call to BPX4SOC, and SOCKADDR contains the name of the peer, possibly obtained by a call to BPX4GNM. For the callable service, see "connect (BPX1CON, BPX4CON) — Establish a connection between two sockets" on page 121. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1CON (connect) example" on page 1132.

```
SPACE ,
MVI   SOCK_LEN,12           Store the length of the address
MVI   SOCK_FAMILY,AF_UNIX   Set the domain to AF_UNIX
MVC   SOCK_SUN_NAME(12),=CL12'/tmp/socket1'  Set the name
CALL  BPX4CON,              Connect to a socket              +
      (SOCKDESC,            Input: Socket Descriptor         +
      SOCK#LEN+SOCK_SUN#LEN, Input: Length - Sockaddr        +
      SOCKADDR,             Input: Sockaddr structure        +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4CPO (cond_post) example

The following code demonstrates how to send an event notification to a thread waiting in the cond_wait or cond_timed_wait service. For the callable service, see "cond_post (BPX1CPO, BPX4CPO) — Post a thread for an event" on page 109. AMODE 31 callers use "BPX1CPO (cond_post) example" on page 1132. The following code notifies thread (THID) that a CW_CONDVAR event has occurred.

```
CALL  BPX4CPO,              Send condition event notification +
      (THID,                Input: Thread ID of target pgm    +
      =A(CW_CONDVAR),       Input: Event            in BPXYCW +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4CRT (chroot) example

The following code changes the root directory for the task. For the callable service, see "chroot (BPX1CRT, BPX4CRT) — Change the root directory" on page 100. AMODE 31 callers use "BPX1CRT (chroot) example" on page 1133.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX4CRT,              Change root directory            +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4CSE (cond_setup) example

The following code sets up the invoker to suspend processing until any of the specified events (CW_INTRPT or CW_CONDVAR) occurs. The BPX4CTW (cond_timed_wait) or BPX4CWA (cond_wait) service is used to actually suspend processing. For the callable service, see "cond_setup (BPX1CSE, BPX4CSE) — Set up to receive event notifications" on page 111. AMODE 31 callers use "BPX1CSE (cond_setup) example" on page 1133.

```
MVC   EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX4CSE,              Condition setup                 +
      (EVENTLIST,           Input: Event list          BPXYCW +
      RETVAL,               Return value: 0 or -1           +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4CTW (cond_timed_wait) example

The following code suspends the calling thread until a signal arrives (CW_INTRPT), or else 2.5 seconds have elapsed. For the callable service, see "cond_timed_wait (BPX1CTW, BPX4CTW) — Suspend a thread for a limited time or an event" on page 114. AMODE 31 callers use "BPX1CTW (cond_timed_wait) example" on page 1133.

```
MVC   EVENTLIST,=A(CW_INTRPT)           Signals
CALL  BPX4CTW,              Wait for condition events       +
      (=A(2),               Input: Number of seconds        +
      =A(500000000),        Input: Number of nanoseconds    +
      EVENTLIST,            Input: Event list          BPXYCW +
      SECONDS,              Output: Unexpired seconds       +
      NANOSECONDS,          Output: Unexpired nanoseconds   +
      RETVAL,               Return value: 0 or -1           +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4CWA (cond_wait) example

The following code suspends the calling thread until either of two events occurs: the arrival of a signal (CW_INTRPT) or some other thread using the cond_post service to send this thread a CW_CONDVAR notification. For the callable service, see "cond_wait (BPX1CWA, BPX4CWA) — Suspend a thread for an event" on page 118. AMODE 31 callers use "BPX1CWA (cond_wait) example" on page 1134.

```
MVC   EVENTLIST,=A(CW_INTRPT+CW_CONDVAR)
CALL  BPX4CWA,              Wait for condition events       +
      (EVENTLIST,           Input: Event list          BPXYCW +
      RETVAL,               Return value: 0 or -1           +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4DEL (deleteHFS) example

The program ictasma located at **ict/bin** is loaded into storage using BPX4LOD, branched to and then deleted from storage using BPX4DEL. For the callable service, see "deletehfs (BPX1DEL, BPX4DEL) — Delete a program from storage" on page 130. AMODE 31 callers use "BPX1DEL (deleteHFS) example" on page 1134.

```
MVC   BUFLENA,=F'16'
MVC   BUFFERA(16),=C'/ict/bin/ictasma'
MVC   OPTIONS,=A(0)
MVC   LIBPTHLN,=A(0)
SPACE ,
CALL  BPX4LOD,              Load Program                     +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      OPTIONS,              Input: Options                   +
      LIBPTHLN,             Input: Library Path Length       +
      LIBPATH,              Input: Library Path              +
      EPADDR,               Entry Point address              +
      RETVAL,               Return value                     +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
SPACE ,
LG    R15,EPADDR
BASSM  R14,R15               Branch to loaded program
SPACE ,
CALL  BPX4DEL,              Delete program                   +
      (EPADDR,              Input: Entry point address       +
      RETVAL,               Return value: -1 or 0            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4ENV (oe_env_np) example

The following code enables interruption of threads waiting in MVS ENQs in the caller's process. For the callable service, see “oe_env_np (BPX1ENV, BPX4ENV) — Examine, change, or examine and change an environmental attribute” on page 435. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 31 callers use “BPX1ENV (oe_env_np) example” on page 1134.

```
LA    R15,=F'1'
ST    R15,INARG
LA    R15,INARG
STG    R15,INARGLIST
LA    R15,INARGLIST
STG   R15,INARGLISTPTR
SPACE ,
CALL  BPX4ENV,              oe_env_np                        +
      (=A(ENQWAIT_PROCESS), Input: Function_code     BPXYCONS +
      =A(1),                Input: InArgCount                +
      INARGLISTPTR,         Input: InArgListPtr              +
      =A(0),                Input: OutArgCount               +
      =AD(0),               Input: OutArgListPtr             +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4EXC (exec) example

The program ictasma located at **ict/bin** gets control and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. For the callable service, see "exec (BPX1EXC, BPX4EXC) — Run a program" on page 132. AMODE 31 callers use "BPX1EXC (exec) example" on page 1135.

```
         MVC   BUFLENA,=F'16'
         MVC   BUFFERA(16),=C'/ict/bin/ictasma'
         MVC   ARGCNT,=F'3'
*                                  First
         LA    R15,=F'4'                Length
         ST    R15,ARGLLST+00           Length parm list
         LA    R15,=CL4'WK18'           Argument
         STG   R15,ARGSLST+00           Argument address parm list
*                                  Second
         LA    R15,=F'7'                Length
         ST    R15,ARGLLST+08           Length parm list
         LA    R15,=CL7'DEPT37A'        Argument
         STG   R15,ARGSLST+08           Argument address parm list
*                                  Third
         LA    R15,=F'22'               Length
         ST    R15,ARGLLST+16           Length parm list
         LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
         STG   R15,ARGSLST+16           Argument address parm list
*
         MVC   ENVCNT,=F'0'          Number of env. data items passed
         MVC   ENVLENS,=FD'0'         Addr of env. data length list
         MVC   ENVPARMS,=FD'0'        Add of env. data
*
         MVC   EXITRTNA,=AD(EXITRTN)  ->exit routine
*        MVC   EXITPLA,=AD(exit parameter list as expected by EXITRTN)
         SPACE ,
         CALL  BPX4EXC,                                          +
               (BUFLENA,             Input: Pathname length      +
               BUFFERA,              Input: Pathname             +
               ARGCNT,               Input: Argument count       +
               ARGLLST,              Input: Argument length list +
               ARGSLST,              Input: Argument address list+
               ENVCNT,               Input: Environment count    +
               ENVLENS,              Input: Environment length list+
               ENVPARMS,             Input: Environment address list+
               EXITRTNA,             Input: Exit routine address or 0+
               EXITPLA,              Input: Exit Parm list address or 0+
               RETVAL,               Return value: -1 or not return+
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               MF=(E,PLIST)          ---------------------------------
```

BPX4EXI (_exit) example

The following code ends the program and returns an exit code of 44 to the waiting parent process. For the callable service, see "_exit (BPX1EXI, BPX4EXI) — End a process and bypass the cleanup" on page 150. AMODE 31 callers use "BPX1EXI (_exit) example" on page 1136.

```
         XC    WAST(WAST#LENGTH),WAST
         MVI   WASTEXITCODE,44       User defined exit code
         SPACE
         CALL  BPX4EXI,              End a process               +
               (WAST),               Input: Status field         +
               MF=(E,PLIST)          ---------------------------------
```

BPX4EXM (execmvs) example

The following code invokes program APPL92 and passes the length and parameter MONTH9,PRELIM,(232/74.99). There is no exit routine associated with program APPL92. For the callable service, see "execmvs (BPX1EXM, BPX4EXM) — Run an MVS program" on page 144. AMODE 31 callers use "BPX1EXM (execmvs) example" on page 1136.

```
MVC   PGMNAMEL,=F'6'
MVC   PGMNAME(06),=CL6'APPL92'
MVC   BUFLENA,=F'24'
MVC   BUFFERA(24),=CL24'MONTH9,PRELIM,(232/74.99)'
SPACE ,
CALL  BPX4EXM,              Invoke a MVS program             +
      (PGMNAMEL,            Input: Length of program name    +
      PGMNAME,              Input: Program name              +
      BUFLENA,              Input: Length of program argument +
      BUFFERA,              Input: Program argument          +
      =AD(0),               Input: Exit routine address or 0  +
      =AD(0),               Input: Exit Parm list address or 0+
      RETVAL,               Return value: -1 or not return   +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4EXT (extlink_np) example

The following code creates an external link to data set **MY.DATASET** for pathname **/mvs/mydataset**. For the callable service, see "extlink_np (BPX1EXT, BPX4EXT) — Create an external symbolic link" on page 153. AMODE 31 callers use "BPX1EXT (extlink_np) example" on page 1136.

```
MVC   BUFFERA(10),=CL10'MY.DATASET'
MVC   BUFLENA,=F'10'
MVC   BUFFERB(14),=CL14'/mvs/mydataset'
MVC   BUFLENB,=F'14'
SPACE ,
CALL  BPX4EXT,              Create external link to name     +
      (BUFLENA,             Input: External name length      +
      BUFFERA,              Input: External name             +
      BUFLENB,              Input: Link name length          +
      BUFFERB,              Input: Link name                 +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4FAI (freeaddrinfo) example

The following code frees the Addr_Info structure(s) that were obtained by the getaddrinfo callable service. For the callable service, see "freeaddrinfo (BPX1FAI, BPX4FAI) — Free Addr_Info structures" on page 194. AMODE 31 callers use "BPX1FAI (freeaddrinfo) example" on page 1137.

```
SPACE ,
CALL  BPX4FAI,              Free Addr_Info                   +
      (ADDR_INFO_PTR,       Input: -> Addr_Info structure    +
      RETVAL,               Return code                      +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4FCA (fchaudit) example

The following code changes the audit for the standard input file to ReadFail, WriteFail and ExecFail. For the callable service, see "fchaudit (BPX1FCA, BPX4FCA) — Change audit flags for a file by descriptor" on page 164. For the data structure, see "BPXYAUDT — Map flag values for chaudit and fchaudit" on page 949. AMODE 31 callers use "BPX1FCA (fchaudit) example" on page 1137.

```
MVI   AUDTREADACCESS,AUDTREADFAIL
MVI   AUDTWRITEACCESS,AUDTWRITEFAIL
MVI   AUDTEXECACCESS,AUDTEXECFAIL
MVI   AUDTRSRV,X'00'
SPACE ,
CALL  BPX4FCA,              Change audit                      +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      AUDT,                 Input: Audit flags, BPXYAUDT      +
      =A(0),                Input: 0 user, 1 security auditor +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4FCD (fchdir) example

The following code changes the working directory for the task to the directory identified by FILEDESC. For the callable service, see "fchdir (BPX1FCD, BPX4FCD) — Change the working directory" on page 167. AMODE 31 callers use "BPX1FCD (fchdir) example" on page 1137.

```
MVC   FILEDESC,..           Directory descriptor from opendir
  SPACE ,
  CALL  BPX4FCD,              Change working directory          +
        (FILEDESC,            Input: Directory file descriptor  +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4FCM (fchmod) example

The following code changes the permissions for the standard input file. For the callable service, see "fchmod (BPX1FCM, BPX4FCM) — Change the mode of a file or directory by descriptor" on page 169. For the data structure, see "BPXYMODE — Map the mode constants of the file services" on page 996 and "BPXYFTYP — File type definitions" on page 967. AMODE 31 callers use "BPX1FCM (fchmod) example" on page 1138.

```
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       All permissions
MVI   S_MODE3,S_IRWXU2+S_IRWXG+S_IRWXO
SPACE ,
CALL  BPX4FCM,              Change file modes                 +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      S_MODE,               Input: Mode, BPXYMODE, BPXYFTYP   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4FCO (fchown) example

The following code changes the owner and group for the standard input file. For the callable service, see "fchown (BPX1FCO, BPX4FCO) — Change the owner and group of a file or directory by descriptor" on page 171. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 31 callers use "BPX1FCO (fchown) example" on page 1138.

```
MVC   GROUPID,..            Group ID
MVC   USERID,..             User ID
  SPACE ,
  CALL  BPX4FCO,              Change the owner and group of file+
        (=A(STDIN_FILENO),    Input: File descriptor           +
        USERID,               Input: New user ID for file      +
        GROUPID,              Input: New group ID for file     +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4FCR (fchattr) example

The following code changes the attributes of the standard input file. The owning user and group ids are changed; the file change time is set to the current time; and the user read-execute, group write, and other read-execute permissions are set. For the callable service, see "fchattr (BPX1FCR, BPX4FCR) — Change the attributes of a file or directory by descriptor" on page 156. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948 and "BPXYMODE — Map the mode constants of the file services" on page 996. AMODE 31 callers use "BPX1FCR (fchattr) example" on page 1138.

```
MVC   ATTID,=CL4'ATT '      Eye Catcher
MVC   ATTVERSION,=AL2(ATT#VER)  version
XC    S_MODE,S_MODE         Clear mode
MVI   S_MODE2,S_IRUSR       Read-execute/write/read-execute
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
MVC   ATTMODE,S_MODE        Move mode data to attribute      +
                            structure
MVC   ATTUID,=F'7'          Specify new UID
MVC   ATTGID,=F'77'         Specify new GID
OI    ATTSETFLAGS1,ATTMODECHG+ATTOWNERCHG                    +
                            Flag Mode, UID and GID changes
OI    ATTSETFLAGS2,ATTCTIMETOD                               +
                            Set change time to current time
SPACE ,
CALL  BPX4FCR,              Change file attributes           +
      (=A(STDIN_FILENO),    Input: File descriptor           +
      =A(ATT#LENGTH),       Input: BPXYATT length            +
      ATT,                  Input/output: BPXYATT            +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4FCT (fcntl) example

The code for the first example duplicates the standard error file descriptor to a file descriptor greater than or equal to FILEDES2.

The code for the second example sets a shared byte range lock. For the callable service, see "fcntl (BPX1FCT, BPX4FCT) — Control open file descriptors" on page 174

174. For the data structure, see "BPXYFCTL — Command values and flags for fcntl" on page 966, "BPXYBRLK — Map byte range lock request for fcntl" on page 950, and "BPXYOPNF — Map flag values for open" on page 1004. AMODE 31 callers use "BPX1FCT (fcntl) example" on page 1139.

```
* for 2nd parm F_DUPFD, F_DUPFD2          3rd parm file desc no..
* for 2nd parm F_GETFD, F_GETFL           3rd parm 0
* for 2nd parm F_SETFD                    3rd parm BPXYFCTL
* for 2nd parm F_GETLK, F_SETLK, F_SETLKW  3rd parm BPXYBRLK
* for 2nd parm F_SETFL                    3rd parm BPXYOPNF
         SPACE ,
* Example 1 - duplicate file descriptor
         MVC   FILEDES2,=F'20'       Get free file descriptor >= 20
         SPACE ,
         CALL  BPX4FCT,              General purpose file control      +
               (=A(STDERR_FILENO),   Input: File descriptor            +
               =A(F_DUPFD),          Input: Action, BPXYFCTL           +
               FILEDES2,             Input: Argument #/0/FCTL/BRLK/OPNF+
               RETVAL,               Return value: 0, -1 or action     +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               MF=(E,PLIST)          ----------------------------------
         SPACE ,
* Example 2 - duplicate file descriptor
         MVC   FILEDES2,=F'20'       Get next higher file descriptor
         LA    R15,BRLK
         STG   R15,BRLKA
         XC    BRLK(BRLK#LENGTH),BRLK      Null out BRLK
         MVC   L_TYPE,=AL2(F_RDLCK)        Lock type = shared
         MVC   L_WHENCE,=AL2(SEEK_CUR)     Whence = from current cursor
         SPACE ,
         CALL  BPX4FCT,              General purpose file control      +
               (=A(STDERR_FILENO),   Input: File descriptor            +
               =A(F_SETLK),          Input: Action, BPXYFCTL           +
               BRLKA,                Input: Argument #/0/FCTL/BRLK/OPNF+
               RETVAL,               Return value: 0, -1 or action     +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               MF=(E,PLIST)          ----------------------------------
```

BPX4FPC (fpathconf) example

The following code obtains the configurable option associated with the pipe buffer. For the callable service, see "fpathconf (BPX1FPC, BPX4FPC) — Determine configurable path name variables using a descriptor" on page 191. For the data structure, see "BPXYPCF — Command values for pathconf and pathconf" on page 1005. AMODE 31 callers use "BPX1FPC (fpathconf) example" on page 1140.

```
      MVC   FILEDESC,..           From opendir
        SPACE ,
        CALL  BPX4FPC,              Get configurable pathname variable+
              (FILEDESC,            Input: Directory file descriptor  +
              =A(PC_PIPE_BUF),      Input: Configurables      BPXYPCF +
              RETVAL,               Return value: 0, -1 or variable   +
              RETCODE,              Return code                       +
              RSNCODE),             Reason code                       +
              MF=(E,PLIST)          ----------------------------------
```

BPX4FRK (fork) example

The following code forks a new process. The next sequential instruction gets control from both the parent process (RETVAL=child process ID) and from the child process (RETVAL=0). If RETVAL=-1, the fork failed. For the callable service, see “fork (BPX1FRK, BPX4FRK) — Create a new process” on page 185. AMODE 31 callers use “BPX1FRK (fork) example” on page 1140.

```
  CALL  BPX4FRK,              Create a new process (fork)       +
        (RETVAL,              Return value: -1, 0, child's PID  +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4FST (fstat) example

The following code gets the file status for the file opened as FILEDESC. For the callable service, see “fstat (BPX1FST, BPX4FST) — Get status information about a file by descriptor” on page 196. For the data structure, see “BPXYSTAT — Map the response structure for stat” on page 1057. AMODE 31 callers use “BPX1FST (fstat) example” on page 1140.

```
MVC   FILEDESC,..           File descriptor from open
  SPACE ,
  CALL  BPX4FST,              Get file status of file descriptor+
        (FILEDESC,            Input: File descriptor            +
        STATL,                Input: Length of buffer needed    +
        STAT,                 Buffer, mapped by BPXYSTAT        +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4FSY (fsync) example

The following code writes file descriptor changes to permanent storage. For the callable service, see “fsync (BPX1FSY, BPX4FSY) — Write changes to permanent storage” on page 201. AMODE 31 callers use “BPX1FSY (fsync) example” on page 1140.

```
MVC   FILEDESC,..           File descriptor from open
  SPACE ,
  CALL  BPX4FSY,              Write changes to permanent storage+
        (FILEDESC,            Input: File descriptor            +
        RETVAL,               Return value: 0 or -1             +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4FTR (ftruncate) example

The following code truncates the file described by FILEDESC after 512 bytes. For the callable service, see “ftruncate (BPX1FTR, BPX4FTR) — Change the size of a file” on page 203. AMODE 31 callers use “BPX1FTR (ftruncate) example” on page 1141.

```
MVC   FILEDESC,..           File descriptor from open
  MVC   NEWLEN(8),=FL8'512'
  SPACE ,
```

```
      CALL  BPX4FTR,             Truncate a file                  +
            (FILEDESC,           Input: File descriptor           +
            NEWLEN,              Input: Length to keep            +
            RETVAL,              Return value: 0 or -1            +
            RETCODE,             Return code                      +
            RSNCODE),            Reason code                      +
            MF=(E,PLIST)         ---------------------------------
```

BPX4FTV (fstatvfs) example

The following code obtains information about the file system containing the file identified by FILEDESC. For the callable service, see “fstatvfs (BPX1FTV, BPX4FTV) — Get the file system status” on page 199. For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1055. AMODE 31 callers use “BPX1FTV (fstatvfs) example” on page 1141.

```
    MVC   FILEDESC,..          File descriptor from open
      SPACE ,
      CALL  BPX4FTV,             Get file system status           +
            (FILEDESC,           Input: File descriptor           +
            SSTFL,               Input: Length of BPXYSSTF        +
            SSTF,                Buffer, BPXYSSTF                 +
            RETVAL,              Return value: -1 or length status +
            RETCODE,             Return code                      +
            RSNCODE),            Reason code                      +
            MF=(E,PLIST)         ---------------------------------
```

BPX4GAI (getaddrinfo) example

The following code returns the IP address and other associated information for the specified node name. For the callable service, see “getaddrinfo (BPX1GAI, BPX4GAI) — Get the IP address and information for a service name or location” on page 205. AMODE 31 callers use “BPX1GAI (getaddrinfo) example” on page 1141.

```
      SPACE ,
      CALL  BPX4GAI,             Get Addr_info                    +
            (NODE_NAME,          Input: Name of Host being queried +
            NODE_NAME_LENGTH,    Input: Length of host name       +
            SERVICE_NAME,        Input: Service name being queried +
            SERVICE_NAME_LENGTH, Input: Length of service name    +
            HINTS_PTR,           Input: Ptr to Addr_Info Structure +
            RESULTS_PTR,         Output:Ptr to Addr_Info Structure +
            CANONICAL_LENGTH,    Output: Length canonical name    +
            RETVAL,              Return code                      +
            RETCODE,             Return code                      +
            RSNCODE),            Reason code                      +
            MF=(E,PLIST)         ---------------------------------
```

BPX4GCL (getclientid) example

The following code obtains the clientid information for caller. This information is used on givesocket (BPX4GIV) and takesocket (BPX4TAK) services. For the callable service, see “getclientid (BPX1GCL, BPX4GCL) — Obtain the calling program's identifier” on page 213. For the data structure, see “BPXYCID — Map the returning structure for getclientid()” on page 951. AMODE 31 callers use “BPX1GCL (getclientid) example” on page 1142.

```
      CALL  BPX4GCL,             get clientid information         +
            (=F'2',              Input: Function code of 2        +
            =A(AF_INET),         Input: Domain of AF_INET         +
```

```
      CID,                Output: Clientid information      +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ----------------------------------
```

BPX4GCW (getcwd) example

The following code gets the working directory for the caller. For the callable service, see "getcwd (BPX1GCW, BPX4GCW) — Get the pathname of the working directory" on page 215. AMODE 31 callers use "BPX1GCW (getcwd) example" on page 1142.

```
MVC   BUFLENA,=F'1024'      Max directory name return area
SPACE ,
CALL  BPX4GCW,              Get working directory name        +
      (BUFLENA,             Input: Length directory work area +
      BUFFERA,              Buffer                            +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4GEG (getegid) example

The following code gets the effective group ID of the caller. For the callable service, see "getegid (BPX1GEG, BPX4GEG) — Get the effective group ID" on page 217. AMODE 31 callers use "BPX1GEG (getegid) example" on page 1142.

```
CALL  BPX4GEG,              Get the effective group ID        +
      (RETVAL),             Return value: effective group ID  +
      MF=(E,PLIST)          ----------------------------------
```

BPX4GEP (getpgid) example

The following code returns the process group ID for the process identified by the input process ID. The process ID value is set to 1. For the callable service, see "getpgid (BPX1GEP, BPX4GEP) — Get the process group ID" on page 250. AMODE 31 callers use "BPX1GEP (getpgid) example" on page 1142.

```
MVC   PROCID,=XL4'00000001'  Value of process ID
SPACE ,
CALL  BPX4GEP,              Get process group ID              +
      (PROCID,              Input: Process ID                 +
      RETVAL,               Return value: process group ID    +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4GES (getsid) example

The following code returns the process group ID for the session leader of the process identified by the input process ID. The process ID value is set to 1. For the callable service, see "getsid (BPX1GES, BPX4GES) — Get the process group ID of the session leader" on page 270. AMODE 31 callers use "BPX1GES (getsid) example" on page 1143.

```
MVC   PROCID,=XL4'00000000'  Value of process ID
SPACE ,
CALL  BPX4GES,               Get group ID of session leader    +
      (PROCID,               Input: Process ID                 +
      RETVAL,                Return value: process group ID    +
      RETCODE,               Return code                       +
      RSNCODE),              Reason code                       +
      MF=(E,PLIST)           ---------------------------------
L     R15,RETVAL             Load return value
C     R15,=F'-1'             Test for -1 return
BE    PSEUDO                 Branch on error
```

BPX4GET (w_getipc) example

The following code retrieves information on the first semaphore defined to the system to which the caller has read access. For the callable service, see "w_getipc (BPX1GET, BPX4GET) — Query interprocess communications" on page 890. For the data structure, see "BPXYIPCQ — Map w_getipc structure" on page 987. AMODE 31 callers use "BPX1GET (w_getipc) example" on page 1143.

```
XC    TOKEN,TOKEN            Zero, token for 1st member
LA    R5,BUFFERA             Area for query IPC return data
STG   R5,BUFA                R5 -> IPCQ
SPACE ,
CALL  BPX4GET,               Interprocess Communications       +
      (TOKEN,                Input: member token               +
      BUFA,                  Input: ->IPCQ             BPXYIPCQ+
      =A(IPCQ#LENGTH),       Input: Length of IPCQ     BPXYIPCQ+
      =A(IPCQ#SEM),          Input: Request            BPXYIPCQ+
      RETVAL,                Return value: 0, -1 or value      +
      RETCODE,               Return code                       +
      RSNCODE),              Reason code                       +
      MF=(E,PLIST)           ---------------------------------
SPACE ,
L     R15,RETVAL             Load return value
C     R15,=F'-1'             Test for -1 return
BE    PSEUDO                 Branch on error
LTR   R15,R15                Test for 0 return
BZ    PSEUDO                 Branch on end of file
ST    R15,TOKEN              Save token for next w_semipc
```

BPX4GEU (geteuid) example

The following code gets the effective user ID of the caller. For the callable service, see "geteuid (BPX1GEU, BPX4GEU) — Get the effective user ID" on page 219. AMODE 31 callers use "BPX1GEU (geteuid) example" on page 1144.

```
CALL  BPX4GEU,               Get the effective user ID         +
      (RETVAL),              Return value: effective user ID   +
      MF=(E,PLIST)           ---------------------------------
```

BPX4GGE (getgrent) example

The following code accesses the group database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each group entry in the database. For the callable service, see "getgrent (BPX1GGE, BPX4GGE) — Sequentially access the group database" on page 221. For the data structure, see "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. AMODE 31 callers use "BPX1GGE (getgrent) example" on page 1144.

```
GGELOOP  DS    0H
         CALL  BPX4GGE,             Access the group database         +
               (RETVAL,             Return value: 0 or ->BPXYGIDS     +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    CHKGGERR             Error or end of file
         USING GIDS,R8
*        access the group structure
         DROP  R8
         B     GGELOOP              Check next group entry
CHKGGERR DS    0H
         ICM   R8,B'1111',RETCODE
         BZ    GGEEOF               End of file
*        handle error as needed
GGEEOF   DS    0H
```

BPX4GGI (getgrgid) example

The following code accesses the group database by the ID of the caller and returns a structure identifying the groups by ID. The group ID value is set to 5. For the callable service, see "getgrgid (BPX1GGI, BPX4GGI) — Access the group database by ID" on page 223. For the data structure, see "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. AMODE 31 callers use "BPX1GGI (getgrgid) example" on page 1144.

```
         MVC   GROUPID,=XL4'00000005'  Value of group ID
         SPACE ,
         CALL  BPX4GGI,             Access the group database         +
               (GROUPID,            Input: Group ID                   +
               RETVAL,              Return value: 0 or ->BPXYGIDS     +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    NOGIDS
         USING GIDS,R8
*        access the group structure
         DROP  R8
NOGIDS   EQU   *
```

BPX4GGN (getgrnam) example

The following code accesses the group database by the name of the caller and returns a structure identifying the groups by ID. For the callable service, see "getgrnam (BPX1GGN, BPX4GGN) — Access the group database by name" on page 226. For the data structure, see "BPXYGIDS — Map data returned for getgrnam and getgrpid" on page 969. AMODE 31 callers use "BPX1GGN (getgrnam) example" on page 1145.

```
         MVC   GRNAMELN,=F'7'
         MVC   GRPGMNAME(7),=CL7'EXTSERV'
         SPACE ,
         CALL  BPX4GGN,             Access the group database         +
               (GRNAMELN,           Input: Length of group name       +
               GRPGMNAME,           Input: Name of group              +
               RETVAL,              Return value: 0 or ->BPXYGIDS     +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
```

BPX4GGR (getgroups) example

The following code provides the caller with a list of supplementary group IDs. The code sets BUFW size to 256. The actual BUFW size is determined from the previous BPX4GGR RETVAL when BUFW was 0. For the callable service, see “getgroups (BPX1GGR, BPX4GGR) — Get a list of supplementary group IDs” on page 229. AMODE 31 callers use “BPX1GGR (getgroups) example” on page 1145.

```
*        MVC   BUFW,=XL4'00000256'   Value of buffer BUFW
         LA    R15,BUFFERA           Space for BUFW words
         STG   R15,BUFA              ->Array for group IDs
         SPACE ,
         CALL  BPX4GGR,              Get list of supplementary grp IDs +
               (BUFW,                Input: Group ID list size         +
               BUFA,                 ->Buffer for Group ID list address+
               RETVAL,               Return value: -1, 0, ID count     +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               MF=(E,PLIST)          ----------------------------------
```

BPX4GHA (gethostbyaddr) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose address is specified as input. For the callable service, see “gethostbyaddr (BPX1GHA, BPX4GHA) Get the IP address and alias of a host name for the specified IP address” on page 234.

The HOSTENT structure has the following format:

- h_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'.
- h_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h_addrtype - The value 2, which signifies AF_INET.
- h_length - The length of the host internet addresses pointed to by h_addr_list.
- h-addr_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

AMODE 31 callers use “BPX1GHA (gethostbyaddr) example” on page 1145.

```
*        MVC   HOST_ADDR,=XL4'C90E0256'   IP Address of Host
         MVC   HOST_ADDRLEN,=F'4'         Address length
         SPACE ,
         CALL  BPX4GHA,              Get host by address               +
               (HOST_ADDR,           Input: IP address of queried HOST +
               HOST_ADDRLEN,         Input: Length of IP address       +
               HOSTENT_PTR,          Output: 0 or -> HOSTENT structure +
               =A(AF_INET),          Input: Domain - AF_INET           +
               RETVAL,               Return code                       +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               MF=(E,PLIST)          ----------------------------------
```

BPX4GHN (gethostbyname) example

The following code returns a pointer to a HOSTENT structure, which contains the alias names and the internet addresses of a host whose domain name is specified as input. For the callable service, see "gethostbyname (BPX1GHN, BPX4GHN) Get IP information for specified host domain names" on page 237.

The HOSTENT structure has the following format:

- h_name - The address of the host name returned by the service. The host name is a variable length field that is ended by x'00'.
- h_aliases - The address of a list of addresses that point to the alias names returned by the service. The list is ended by the pointer x'00000000'. Each alias name is a variable length field that is ended by x'00'.
- h_addrtype - The value 2, which signifies AF_INET.
- h_length - The length of the host internet addresses pointed to by h_addr_list.
- h-addr_list - The address of a list of addresses that point to the host internet addresses returned by this service. The list is ended by the pointer x'00000000'.

AMODE 31 callers use "BPX1GHN (gethostbyname) example" on page 1146.

```
MVC   HOST_NAME(8),=CL8'HOST1234'
MVC   HOST_NAMELEN,=F'8'
SPACE ,
CALL  BPX4GHN,              Get host by name                  +
      (HOST_NAME,           Input: Name of Host being queried +
      HOST_NAMELEN,         Input: Length of host name        +
      HOSTENT_PTR,          Output: 0 or -> HOSTENT structure +
      RETVAL,               Return code                       +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4GID (getgid) example

The following code gets the real group ID of the caller. For the callable service, see "getgid (BPX1GID, BPX4GID) — Get the real group ID" on page 220. AMODE 31 callers use "BPX1GID (getgid) example" on page 1146.

```
CALL  BPX4GID,              Get the real group ID             +
      (RETVAL),             Return value: real group ID       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4GIV (givesocket) example

The following code gives a socket to the program identified by CID (clientid). The target program may then use takesocket (BPX4TAK) to take the socket. SOCKDESC was previously set by a call to BPX4ACP. CID is set by the getclientid (BPX4GCL) service. For the callable service, see "givesocket (BPX1GIV, BPX4GIV) — Give a socket to another program" on page 285. For the data structure, see "BPXYCID — Map the returning structure for getclientid()" on page 951. AMODE 31 callers use "BPX1GIV (givesocket) example" on page 1147.

```
CALL  BPX4GIV,              give a socket to another program  +
      (SOCKDESC,            Input: Socket descriptor          +
      CID,                  Input: Clientid of recipient      +
      RETVAL,               Return value: 0 or -1             +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4GLG (getlogin) example

The following code gets the login name of the caller. For the callable service, see "getlogin (BPX1GLG, BPX4GLG) — Get the user login name" on page 245. AMODE 31 callers use "BPX1GLG (getlogin) example" on page 1147.

```
CALL  BPX4GLG,             Get the login name               +
      (RETVAL),            Returns value, 0 or ->login name +
      MF=(E,PLIST)         ---------------------------------
```

BPX4GMN (w_getmntent) example

The following code gets the mount entries for the caller. For the callable service, see "w_getmntent (BPX1GMN, BPX4GMN) — Get information on mounted file systems" on page 894. For the data structure, see "BPXYMNTE — Map response and element structure of w_getmntent" on page 993.

If BPXYMNTE is assembled with MNTE2=YES, fields MNTEHID and MNTEHBLEN must be initialized. AMODE 31 callers use "BPX1GMN (w_getmntent) example" on page 1147.

```
LA    R14,MNTEH            R14->MNTEH and MNTE
L     R15,MNTEL            R15 = Length of MNTEH and MNTE
XR    R0,R0                Dummy 2nd operand
XR    R1,R1                Pad=null, length=0
MVCL  R14,R0               Null out MNTEH and MNTE
MVC   MNTEHID,=CL4'MNT2'          Version indicator
MVC   MNTEHBLEN,=A(MNTE#LENGTH)   Length of MNTE
CALL  BPX4GMN,             Get mount entries                +
      (MNTEL,              Input: Length BPXYMNTE + MNTEH   +
      MNTEH,               Header in BPXYMNTH               +
      RETVAL,              Return value: -1 or mount entries +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4GNI (getnameinfo) example

The following code resolves a socket address into a host name and a service name. For the callable service, see "getnameinfo (BPX1GNI, BPX4GNI) — Get the host name and service name from a socket address" on page 246. AMODE 31 callers use "BPX1GNI (getnameinfo) example" on page 1148.

```
SPACE ,
CALL  BPX4GNI,             Get name info                    +
      (SOCKADDR,           Input: Socket address            +
      SOCKADDR_LENGTH,     Input: Length of socket address  +
      SERVICE_BUFFER,      I/O:   Buffer   for service name +
      SERVICE_BUFFER_LENGTH, I/O: Length of service buffer  +
      HOST_BUFFER,         I/O:   Buffer   for host name    +
      HOST_BUFFER_LENGTH,  I/O:   Length of host buffer     +
      FLAGS,               Input: Flags                     +
      RETVAL,              Return code                      +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4GPG (getpgrp) example

The following code gets the process group ID of the caller. For the callable service, see “getpgrp (BPX1GPG, BPX4GPG) — Get the process group ID” on page 252. AMODE 31 callers use “BPX1GPG (getpgrp) example” on page 1148.

```
        CALL  BPX4GPG,              Get the process group ID         +
              (RETVAL),             Return value: group ID           +
              MF=(E,PLIST)          ---------------------------------
```

BPX4GNM (getpeername or getsockname) example

The following code gets the peer name, and then requests the socket name. SOCKDESC was returned by a previous call to BPX4SOC. For the callable service, see “getsockname or getpeername (BPX1GNM, BPX4GNM) - Get the name of a socket or connected peer” on page 272. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043. AMODE 31 callers use “BPX1GNM (getpeername or getsockname) example” on page 1148.

```
        SPACE ,
        CALL  BPX4GNM,                Get peername                   +
              (SOCKDESC,              Input: Socket Descriptor       +
              SOCK#GNMOPTGETPEERNAME, Input: Indicate getpeername    +
              SOCK#LEN+SOCK_SUN#LEN,  Input: Length - Sockaddr       +
              SOCKADDR,               Input: Sockaddr structure      +
              RETVAL,                 Return value: 0 or -1          +
              RETCODE,                Return code                    +
              RSNCODE),               Reason code                    +
              MF=(E,PLIST)          ---------------------------------
        SPACE ,
        CALL  BPX4GNM,                Get sockname                   +
              (SOCKDESC,              Input: Socket Descriptor       +
              SOCK#GNMOPTGETSOCKNAME, Input: Indicate getpeername    +
              SOCK#LEN+SOCK_SUN#LEN,  Input: Length - Sockaddr       +
              SOCKADDR,               Input: Sockaddr structure      +
              RETVAL,                 Return value: 0 or -1          +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              MF=(E,PLIST)          ---------------------------------
```

BPX4GPE (getpwent) example

The following code accesses the user database starting with the next available entry and continuing until end of file on the database. It returns a structure identifying information about each user entry in the database. For the callable service, see “getpwent (BPX1GPE, BPX4GPE) — Sequentially access the user database” on page 258. For the data structure, see “BPXYGIDN — Map data returned for getpwnam and getpwuid” on page 969. AMODE 31 callers use “BPX1GPE (getpwent) example” on page 1149.

```
GPELOOP  DS    0H
         CALL  BPX4GPE,             Access the user database         +
               (RETVAL,             Return value: 0 or ->BPXYGIDN    +
               RETCODE,             Return code                      +
               RSNCODE),            Reason code                      +
               MF=(E,PLIST)         ---------------------------------
         ICM   R8,B'1111',RETVAL
         BZ    CHKGPERR             Error or end of file
         USING GIDN,R8
*        access the user structure
         DROP  R8
```

```
         B     GPELOOP              Check next user entry
CHKGPERR DS    0H
         ICM   R8,B'1111',RETCODE
         BZ    GPEEOF               End of file
*        handle error as needed
GPEEOF   DS    0H
```

BPX4GPI (getpid) example

The following code gets the process ID of the caller. For the callable service, see "getpid (BPX1GPI, BPX4GPI) — Get the process ID" on page 253. AMODE 31 callers use "BPX1GPI (getpid) example" on page 1149.

```
         CALL  BPX4GPI,             Get the process ID              +
               (RETVAL),            Returns value, Process ID       +
               MF=(E,PLIST)         -------------------------------
```

BPX4GPN (getpwnam) example

The following code accesses the group database by the user ID of the caller and returns a structure identifying the groups by name. For the callable service, see "getpwnam (BPX1GPN, BPX4GPN) — Access the user database by user name" on page 260. For the data structure, see "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969. AMODE 31 callers use "BPX1GPN (getpwnam) example" on page 1149.

```
         MVC   USERNLEN,=F'8'
         MVC   USERNAME(8),=CL8'Pebbles'
         SPACE ,
         CALL  BPX4GPN,             Access the user database        +
               (USERNLEN,           Input: Length of user name      +
               USERNAME,            Input: Name of user             +
               RETVAL,              Return value 0 or ->BPXYGIDN    +
               RETCODE,             Return code                     +
               RSNCODE),            Reason code                     +
               MF=(E,PLIST)         -------------------------------
```

BPX4GPP (getppid) example

The following code gets the process ID of the caller's parent. For the callable service, see "getppid (BPX1GPP, BPX4GPP) — Get the parent process ID" on page 254. AMODE 31 callers use "BPX1GPP (getppid) example" on page 1150.

```
         CALL  BPX4GPP,             Get PID of the parent process   +
               (RETVAL),            Returns value, parent's process ID+
               MF=(E,PLIST)         -------------------------------
```

BPX4GPT (grantpt) example

The following code grants access to the slave pseudoterminal device that is identified by the file descriptor. For the callable service, see "grantpt (BPX1GPT, BPX4GPT) — Grant access to the slave pseudoterminal" on page 289. AMODE 31 callers use "BPX1GPT (grantpt) example" on page 1150.

```
         CALL  BPX4GPT,             Grant access to slave pty       +
               (MASTER_FD,          Input: File descriptor          +
               RETVAL,              Return value: 0 or -1           +
               RETCODE,             Return code                     +
               RSNCODE),            Reason code                     +
               MF=(E,PLIST)         -------------------------------
```

BPX4GPU (getpwuid) example

The following code accesses the group database by the user name of the caller and returns a structure identifying the groups by name. The code sets the user ID value to 1. For the callable service, see "getpwuid (BPX1GPU, BPX4GPU) — Access the user database by user ID" on page 263. For the data structure, see "BPXYGIDN — Map data returned for getpwnam and getpwuid" on page 969. AMODE 31 callers use "BPX1GPU (getpwuid) example" on page 1151.

```
MVC   USERID,..             Value of user ID
  SPACE ,
  CALL  BPX4GPU,              Access database by user ID       +
        (USERID,              Input: User ID                   +
        RETVAL,               Return value 0 or ->BPXYGIDN     +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4GPY (getpriority) example

The following code gets the CPU priority based on the input which and who values. The which value used is PRIO_PROCESS, which indicates to get the priority by process ID. The who value used is 7, indicating to get the priority for process ID 7. For the callable service, see "getpriority (BPX1GPY, BPX4GPY) — Get the scheduling priority of a process" on page 255. AMODE 31 callers use "BPX1GPY (getpriority) example" on page 1151.

```
  MVC   PROCID,=XL4'00000007' Process ID to get priority for
  SPACE ,
  CALL  BPX4GPY,              Get priority value               +
        (=A(PRIO_PROCESS),    Input: Get by Process ID         +
        PROCID,               Input: PID to get priority for   +
        RETVAL,               Return value: Priority of process +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
  L     R15,RETVAL            Load return value
  C     R15,=F'-1'            Test for -1 return
  BE    PSEUDO                Branch on error
```

BPX4GRL (getrlimit) example

The following code fills in the rlimit structure for the calling process based on the input resource value. The resource value is set to RLIMIT_CPU. For the callable service, see "getrlimit (BPX1GRL, BPX4GRL) — Get resource limits" on page 266. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 31 callers use "BPX1GRL (getrlimit) example" on page 1151.

```
  MVC   RESOURCE,=A(RLIMIT_CPU) Value of resource
  SPACE ,
  CALL  BPX4GRL,              Get resource limits              +
        (RESOURCE,            Input: resource                  +
        RLIMIT,               Structure, mapped by BPXYRLIM    +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
  L     R15,RETVAL            Load return value
  C     R15,=F'-1'            Test for -1 return
  BE    PSEUDO                Branch on error
```

BPX4GRU (getrusage) example

The following code fills in the rusage structure based on the input who value. The who value is set to RUSAGE_SELF. For the callable service, see "getrusage (BPX1GRU, BPX4GRU) — Get resource usage" on page 268. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 31 callers use "BPX1GRU (getrusage) example" on page 1152.

```
MVC   WHO,=A(RUSAGE_SELF)   Value of who
SPACE ,
CALL  BPX4GRU,              Get resource usage               +
      (WHO,                 Input: who                       +
      RUSAGE,               Structure, mapped by BPXYRLIM    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4GTH (__getthent) example

The following code retrieves information on the first process accessible to the caller. For the callable service, see "__getthent (BPX1GTH, BPX4GTH) — Get thread data" on page 278. For the data structure, see "BPXYPGTH — Map the __getthent input/output structure" on page 1009. AMODE 31 callers use "BPX1GTH (__getthent) example" on page 1152.

```
LA    R5,BUFFERB            R5 -> Input parameters
STG   R5,BUFB               ->input buffer
USING PGTHA,R5              R5 base for PGTHA
XC    PGTHA(PGTHA#LEN),PGTHA  Null Input area
MVI   PGTHAFLAG1,PGTHAPROCESS+PGTHACOMMAND+PGTHATHREAD
MVI   PGTHAACCESSTHID,PGTH#LAST   Last thread
LA    R15,BUFFERA           PgthB, Output buffer
STG   R15,BUFA              ->output buffer
DROP  R5
SPACE ,
CALL  BPX4GTH,              __getthent                       +
      (=A(PGTHA#LEN),       Input: length input parms BPXYPGTH+
      BUFA,                 Input: ->input parms      BPXYPGTH+
      =A(1024),             Input: length output area BPXYPGTH+
      BUFB,                 Input: ->output area      BPXYPGTH+
      RETVAL,               Return value: 0, -1              +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4GTR (getitimer) example

The following code returns the time remaining an alarm, or ITIMER_REAL as set by setitimer. For the callable service, see "getitimer (BPX1GTR, BPX4GTR) — Get the value of the interval timer" on page 242. For the data structure, see "BPXYITIM — Map getitimer, setitimer structure" on page 990. AMODE 31 callers use "BPX1GTR (getitimer) example" on page 1152.

```
LA    R15,ITIM              Output mapping structure
STG   R15,ITIMA             ->structure
CALL  BPX4GTR,              Get process data                 +
      (=A(ITIMER_REAL),     Input: Relative process token    +
```

```
      ITIMA,               Out: ->Buffer, mapped by BPXYITIM +
      RETVAL,              Return value: -1, 0               +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4GUG (getgroupsbyname) example

The following code returns the number of supplementary group IDs, up to 9, for user Pebbles. For the callable service, see "getgroupsbyname (BPX1GUG, BPX4GUG) — Get a list of supplementary group IDs by user name" on page 231. AMODE 31 callers use "BPX1GUG (getgroupsbyname) example" on page 1153.

```
MVC   USERNLEN,=F'7'
MVC   USERNAME(07),=CL07'Pebbles'
MVC   BUFLENA,=F'9'
LA    R15,BUFFERA
STG   R15,BUFA
SPACE ,
CALL  BPX4GUG,             Get list of groups by user name   +
      (USERNLEN,           Input: User name length           +
      USERNAME,            Input: User name                  +
      BUFLENA,             Input: Group ID list size         +
      BUFA,                Group ID list address             +
      RETVAL,              Return value: -1, or # of grp IDs +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4GUI (getuid) example

The following code gets the invoker's real user ID. For the callable service, see "getuid (BPX1GUI, BPX4GUI) — Get the real user ID" on page 282. AMODE 31 callers use "BPX1GUI (getuid) example" on page 1153.

```
CALL  BPX4GUI,             Get the real user ID              +
      (RETVAL),            Return value: real user ID        +
      MF=(E,PLIST)         ----------------------------------
```

BPX4GWD (getwd) example

The following code gets the working directory for the caller. For the callable service, see "getwd (BPX1GWD, BPX4GWD) — Get the pathname of the working directory" on page 283. AMODE 31 callers use "BPX1GWD (getwd) example" on page 1153.

```
MVC   BUFLENA,=F'1024'     Max directory name return area
SPACE ,
CALL  BPX4GWD,             Get working directory name        +
      (BUFLENA,            Input: Length directory work area +
      BUFFERA,             Buffer                            +
      RETVAL,              Return value: length or -1        +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4HST (gethostid or gethostname) example

The following code requests the host id and the host name for an AF_INET domain. For the callable service, see "gethostid or gethostname (BPX1HST, BPX4HST) — Get ID or name information about a socket host" on page 240. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1HST (gethostid or gethostname) example" on page 1154.

```
XC    BUFLENA,BUFLENA
CALL  BPX4HST,              Request host id                  +
      (=A(AF_INET),         Input: Domain - AF_INET          +
      BUFLENA,              Input: Length - No buffer - get id+
      BUFFERA,              Output: (not used with Length=0)  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------

MVC   BUFLENA,=A(L'BUFFERA)
CALL  BPX4HST,              Request host name                +
      (=A(AF_INET),         Input: Domain - AF_INET          +
      BUFLENA,              Input: Length - for output name   +
      BUFFERA,              Output: Buffer for host name      +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4IOC (w_ioctl) example

The following code conveys a command to the standard output device. To run properly this example needs a command defined by the user for the COMMAND parameter. This command must be understood by the device driver providing support for the output device. For the callable service, see "w_ioctl (BPX1IOC, BPX4IOC) — Control I/O" on page 902. AMODE 31 callers use "BPX1IOC (w_ioctl) example" on page 1154.

```
MVC   BUFLENA,=F'1024'
MVC   COMMAND,=F'123'       User defined command
SPACE ,
CALL  BPX4IOC,              I/O Control                      +
      (=A(STDOUT_FILENO),   Input: File descriptor           +
      COMMAND,              Input: Command                   +
      BUFLENA,              Input: Argument length           +
      BUFFERA,              Argument buffer name             +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4IPT (mvsiptaffinity) example

The following code executes the assembler routine EXITRTN on the IPT of the requesting thread, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see "mvsiptaffinity (BPX1IPT, BPX4IPT) — Run a program on the IPT thread" on page 410. AMODE 31 callers use "BPX1IPT (mvsiptaffinity) example" on page 1154.

```
         MVC   EXITRTNA,=AD(EXITRTN)  ->Routine address
*        MVC   EXITPLA,=AD(EXITPARM) ->Input parameter list
         SPACE ,
         CALL  BPX4IPT,                                          +
               (EXITRTNA,            Input: Routine address      +
               EXITPLA,              Input: Parm list address or 0  +
               RETVAL,               Return value: -1 or not return  +
               RETCODE,              Return code                 +
               RSNCODE),             Reason code                 +
               MF=(E,PLIST)          ---------------------------------
```

BPX4ITY (isatty) example

The following code determines if the standard output device is a terminal. For the callable service, see “isatty (BPX1ITY) (POSIX Version) — Determine whether a file descriptor represents a terminal” on page 301. AMODE 31 callers use “BPX2ITY (isatty) example” on page 1155.

```
         CALL  BPX4ITY,              Determine if device is a TTY      +
               (=A(STDOUT_FILENO),   Input: File descriptor            +
               RETVAL,               Return value: 0 isn't, 1 is, -1   +
               RETCODE,              Return code: describes why VAL=-1 +
               RSNCODE),             Reason code: qualifier on RETCODE +
               MF=(E,PLIST)          ---------------------------------
         ICM   R15,B'1111',RETVAL    Test RETVAL
         BZ    PSEUDO                RETVAL=0 means device not terminal
```

BPX4KIL (kill) example

The following code sends a signal (SIGUSR1) to all processes for which access is allowed in the invoker's process group. For the callable service, see “kill (BPX1KIL, BPX4KIL) — Send a signal to a process” on page 304. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 31 callers use “BPX1KIL (kill) example” on page 1155.

```
         MVC   PROCID,=A(0)          Invoker's process group
         CALL  BPX4KIL,              Send a signal to a process        +
               (PROCID,              Input: Process ID                 +
               =A(SIGUSR1#),         Input: Signal            BPXYSIGH +
               =A(0),                Input: Signal options             +
               RETVAL,               Return value: 0 or -1             +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               MF=(E,PLIST)          ---------------------------------
```

BPX4LCO (lchown) example

The following code changes the owner of symbolic link **/somedir/somesymlink.c** from the current owner to that specified by USERID and GROUPID. For the callable service, see “lchown (BPX1LCO, BPX4LCO) — Change the owner or group of a file, directory, or symbolic link” on page 324. AMODE 31 callers use “BPX1LCO (lchown) example” on page 1156.

```
           MVC   BUFFERA(22),=CL22'/somedir/somesymlink.c'
           MVC   BUFLENA,=F'22'
       MVC   USERID,..             New owner UID from stat
       MVC   GROUPID,..            New owner GID from stat
           SPACE ,
           CALL  BPX4LCO,            Change owner and group of a file  +
                 (BUFLENA,           Input: Pathname length            +
```

```
      BUFFERA,             Input: Pathname                  +
      USERID,              Input: New owner UID             +
      GROUPID,             Input: New owner GID             +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4LCR (lchattr) example

The following code changes the attributes of symbolic link **/somedir/somesymlink.c**. The security label is set and the file change time is set. For the callable service, see "lchattr (BPX1LCR, BPX4LCR) — Change the attributes of a file or directory or symbolic link" on page 315. For the data structures, see "BPXYATT — Map file attributes for chattr and fchattr" on page 948. AMODE 31 callers use "BPX1LCR (lchattr) example" on page 1156.

```
      MVC   BUFFERA(22),=CL22'/somedir/somesymlink.c'
      MVC   BUFLENA,=F'22'
      XC    ATT,ATT              Clear ATT
      MVC   ATTID,=CL4'ATT '     Eye Catcher
      MVC   ATTVERSION,=AL2(ATT#VER)  version
      MVC   ATTSECLABEL,=CL08'SYSMULTI'
      OI    ATTSETFLAGS3,ATTSECLABELCHG                      +
      Flag Seclabel update
      OI    ATTSETFLAGS2,ATTCTIMETOD                         +
      Set change time to current time
      SPACE ,
      CALL  BPX4LCR,             Change file attributes      +
      (BUFLENA,                  Input: Pathname length      +
      BUFFERA,                   Input: Pathname             +
      =A(ATT#LENGTH),            Input: BPXYATT length       +
      ATT,                       Input/output: BPXYATT       +
      RETVAL,                    Return value: 0 or -1       +
      RETCODE,                   Return code                 +
      RSNCODE),                  Reason code                 +
      MF=(E,PLIST)               ---------------------------------
```

BPX4LDX (loadHFS extended) example

The following is an example specifying the Lod_Directed option. For an example of BPX1LDX/BPX4LDX without the Lod_Directed option flag specified, see "BPX1LOD (loadHFS) example" on page 1158, substituting BPX1LDX/BPX4LDX for BPX1LOD/BPX4LOD. The program ictasma located at **ict/bin** is loaded into storage and then branched to. Then the CSVDYLPA service is called to provide serviceability information to the system. The loaded module can then be branched to. When the load module is no longer needed, the serviceability information should be deleted and the module's storage released. For the callable service, see "loadhfs extended (BPX1LDX, BPX4LDX) — Direct the loading of an executable into storage" on page 338. AMODE 31 callers use "BPX1LDX (loadHFS extended) example" on page 1156.

```
MVC   BUFLENA,=F'16'
MVC   BUFFERA(16),=C'/ict/bin/ictasma'
MVC   OPTIONS,=AL4(LOD_DIRECTED)  Directed loadhfs to common
OI    OPTIONS+3,X'F1'             Subpool 241
MVC   LIBPTHLN,=A(0)
SPACE ,
CALL  BPX4LDX,   Load program                              +
       (BUFLENA,  Input: Pathname length                    +
        BUFFERA,   Input: Pathname                          +
        OPTIONS,   Input: Options                           +
```

BPX4LDX (loadHFS extended) example

```
        LIBPTHLN,  Input: Library Path Length                  +
        LIBPATH,   Input: Library Path                         +
        RTNPARM@,   Output: directed load ret parm structure   +
        RETVAL,    Return value: -1 or 0                       +
        RETCODE,   Return code                                 +
       RSNCODE),  Reason code                                  +
        MF=(E,PLIST)  ----------------------------------------
          SPACE ,
L     R15,RETVAL   Load return value
C     R15,=F'-1'   Test for -1 return
BE    PSEUDO   Branch on error
SPACE ,
LG    R5,RTNPARM@
MVC   LOCALPARMS(24),0(R5)  Local copy of returned parameters
*
*              Provide serviceability information to system
*
LA    R4,LOCALPARMS
USING DIRECTEDLOADRETURNEDPARMS,R4
LGHI  R7,-2
LG    R5,DIRECTEDLOADMODULEENTRYPT64
NGR   R5,R7                  Clear entry point amode 64 flag
STG   R5,DIRECTEDLOADMODULEENTRYPT64
LG    R6,DIRECTEDLOADMODULESTART64
LG    R7,DIRECTEDLOADMODULELENGTH64
XC    LPMEA(LPMEA_LEN),LPMEA
ST    R5,LPMEAENTRYPOINTADDR
ST    R6,LPMEALOADPOINTADDR
ST    R7,LPMEAMODLEN
MVC   LPMEANAME,=C'ICTASMA '
CSVDYLPA REQUEST=ADD,                                        +
       BYADDR=YES,                                            +
       MODINFOTYPE=MEMBERLIST,                                +
       MODINFO=LPMEA,                                         +
       NUMMOD=1,                                              +
       REQUESTOR=REQID,                                       +
       RETCODE=RETCODE,                                       +
       RSNCODE=RSNCODE,                                       +
       MF=(E,DYLPAL)    Provide serviceability information
L     R15,RETCODE      Load return code
LTR   R15,R15
BNZ   PSEUDO
MVC   LOCALDELTOKEN(8),LPMEADELETETOKEN
SPACE ,
 .
 .
 .
*
*          Call directed loadhfs target module
*
LG    R15,DIRECTEDLOADMODULEENTRYPT64
BALR  R14,R15   Branch to loaded program
SPACE ,
 .
 .
 .
*
*         When done with directed load hfs module
*         remove serviceability information and
*         release module storage
*
XC    LPMED(LPMED_LEN),LPMED
MVC   LPMEDNAME,=C'ICTASMA '
MVC   LPMEDDELETETOKEN(8),LOCALDELTOKEN
CSVDYLPA REQUEST=DELETE,                                     +
       TYPE=BYTOKEN,                                          +
       MODINFO=LPMED,                                         +
```

```
       NUMMOD=1,                                                +
       RETCODE=RETCODE,                                         +
       RSNCODE=RSNCODE,                                         +
       MF=(E,DYLPAL)    Remove serviceability information
L      R15,RETCODE      Load return code
LTR    R15,R15
BNZ    PSEUDO
SPACE ,
MODESET MODE=SUP
L      R7,DIRECTEDLOADMODULELENGTH
STORAGE RELEASE,                                               +
       LENGTH=(R7),                                             +
       ADDR=DIRECTEDLOADMODULESTART,                            +
       SP=241           Free module
MODESET MODE=PROB
DROP   R4
```

BPX4LOD (loadHFS) example

The program ictasma located at **ict/bin** is loaded into storage and then branched to. For the callable service, see "loadhfs (BPX1LOD, BPX4LOD) — Load a program into storage by path name" on page 333. AMODE 31 callers use "BPX1LOD (loadHFS) example" on page 1158.

```
MVC   BUFLENA,=F'16'
MVC   BUFFERA(16),=C'/ict/bin/ictasma'
MVC   OPTIONS,=A(0)
MVC   LIBPTHLN,=A(0)
SPACE ,
CALL  BPX4LOD,             Load program                      +
      (BUFLENA,            Input: Pathname length            +
      BUFFERA,             Input: Pathname                   +
      OPTIONS,             Input: Options                    +
      LIBPTHLN,            Input: Library Path Length        +
      LIBPATH,             Input: Library Path               +
      ENTRYPT,             Output:Entry Point                +
      RETVAL,              Return value: -1 or 0             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ---------------------------------
SPACE ,
L     R15,RETVAL
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error
SPACE ,
LG    R15,ENTRYPT
BALR  R14,R15              Branch to loaded program
```

BPX4LNK (link) example

The following code creates a new way for **usr/dataproc.next.t** to link to an existing file, **usr/user05/yearrecs.t** . For the callable service, see "link (BPX1LNK, BPX4LNK) — Create a link to a file" on page 327. AMODE 31 callers use "BPX1LNK (link) example" on page 1159.

```
MVC   BUFLENA,=F'21'
MVC   BUFFERA(21),=CL21'usr/user05/yearrecs.t'
MVC   BUFLENB,=F'19'
MVC   BUFFERB(19),=CL19'usr/dataproc.next.t'
SPACE ,
CALL  BPX4LNK,             Create a link to a file           +
      (BUFLENA,            Input: Name length: existing      +
      BUFFERA,             Input: Name of existing file      +
```

```
         BUFLENB,             Input: Name length: link         +
         BUFFERB,             Input: Name of link to file      +
         RETVAL,              Return value: 0 or -1            +
         RETCODE,             Return code                      +
         RSNCODE),            Reason code                      +
         MF=(E,PLIST)         ---------------------------------
```

BPX4LSK (lseek) example

The following code changes the file (FILEDESC) offset to 80 bytes past the current offset. For the callable service, see "lseek (BPX1LSK, BPX4LSK) — Change a file's offset" on page 345. For the data structure, see "BPXYSEEK — Constants for lseek" on page 1036. AMODE 31 callers use "BPX1LSK (lseek) example" on page 1159.

```
MVC   FILEDESC,..           File descriptor from open
  MVC   OFFSET(08),=FL8'80'   Forward 80 Bytes
  MVC   REFPT,=A(SEEK_CUR)    Current offset of the file
  SPACE ,
  CALL  BPX4LSK,              Change a file's offset           +
        (FILEDESC,            File descriptor                  +
        OFFSET,               I/O: Offset in file              +
        REFPT,                Input: Reference point, BPXYSEEK +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4LSN (listen) example

The following code issues a listen on a socket that was previously created and that had a bind done for it. SOCKDESC was returned from the call to BPX4SOC. Set the backlog count to 5. For the callable service, see "listen (BPX1LSN, BPX4LSN) — Prepare a server socket to queue incoming connection requests from clients" on page 330. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1LSN (listen) example" on page 1159.

```
        CALL  BPX4LSN,              Listen on a socket               +
              (SOCKDESC,            Input: Socket Descriptor         +
              =A(5),                Input: Backlog count of 5        +
              RETVAL,               Return value: 0 or -1            +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              MF=(E,PLIST)          ---------------------------------
```

BPX4LST (lstat) example

The following code obtains the file status for the file described by the symbolic name **labrec/sym**. For the callable service, see "lstat (BPX1LST, BPX4LST) — Get status information about a file or symbolic link by path name" on page 349. For the data structure, see "BPXYSTAT — Map the response structure for stat" on page 1057. AMODE 31 callers use "BPX1LST (lstat) example" on page 1160.

```
*   symbolic name established using symlink (BPX4SYM) system call
        MVC   BUFFERA(10),=CL10'labrec/sym'
        MVC   BUFLENA,=F'10'
        SPACE ,
        CALL  BPX4LST,              Get file status                  +
              (BUFLENA,             Input: Pathname length           +
              BUFFERA,              Input: Pathname                  +
              STATL,                Input: Length of buffer needed   +
```

```
      STAT,               Buffer, mapped by BPXYSTAT       +
      RETVAL,             Return value: 0 or -1            +
      RETCODE,            Return code                      +
      RSNCODE),           Reason code                      +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MAT (shmat) example

The following code attaches a shared memory segment. For the callable service, see "shmat (BPX1MAT, BPX4MAT) — Attach to a shared memory segment" on page 714. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039. AMODE 31 callers use "BPX1MAT (shmat) example" on page 1160.

```
CALL  BPX4MAT,            Shared memory segment control     +
      (SHM_ID,            Input: Shared memory segment ID   +
      SEGADDR,            Input: ST loc for seg address     +
      =A(0),              Input: Flags              BPXYSHM +
      ATTADDR,            Output: memory segment address    +
      RETVAL,             Return value: 0, -1               +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MCT (shmctl) example

The following code retrieves the size of the shared memory segment. For the callable service, see "shmctl (BPX1MCT, BPX4MCT) — Perform shared memory control operations" on page 718. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039. AMODE 31 callers use "BPX1MCT (shmctl) example" on page 1160.

```
LA    R15,BUFFERA
STG   R15,BUFA
SPACE ,
CALL  BPX4MCT,            Shared memory segment control     +
      (SHM_ID,            Input: Shared memory segment ID   +
      =A(IPC_STAT),       Input: Command            BPXYIPC +
      BUFA,               Input: ->SHMID_DS or 0    BPXYSHM +
      RETVAL,             Return value: 0, -1 or value      +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MDT (shmdt) example

The following code detaches a shared memory segment. For the callable service, see "shmdt (BPX1MDT, BPX4MDT) — Detach a shared memory segment" on page 722. For the data structure, see "BPXYSHM—Map interprocess communication shared memory segments" on page 1039. AMODE 31 callers use "BPX1MDT (shmdt) example" on page 1161.

```
CALL  BPX4MDT,            Shared memory segment detach      +
      (SEGADDR,           Input: Shared memory segment addr +
      RETVAL,             Return value: 0, -1 or value      +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MGT (shmget) example

The following code creates a private shared memory segment of 500 bytes. For the callable service, see “shmget (BPX1MGT, BPX4MGT) — Create/find a shared memory segment” on page 738. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1037. AMODE 31 callers use “BPX1MGT (shmget) example” on page 1161.

```
MVC   KEY(4),=A(IPC_PRIVATE)  Local to this family
MVI   S_TYPE,IPC_CREAT+IPC_EXCL   Must not already exist
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX4MGT,              Create a set of semaphores        +
      (KEY,                 Input: Shared memory segment KEY  +
      =AD(500),             Input: Segment size               +
      S_MODE,               Input: Creation flags     BPXYIPC +
      RETVAL,               Return value: -1 or MessageQue ID +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on shmget failure
ST    R15,SHM_ID            Store SHM_ID associated with key
```

BPX4MKD (mkdir) example

The following code creates a new and empty directory pathname of **/usr/newprots/** with user read-execute, group write, other read-execute permissions. For the callable service, see “mkdir (BPX1MKD, BPX4MKD) — Make a directory” on page 361. For the data structure, see “BPXYFTYP — File type definitions” on page 967 and “BPXYMODE — Map the mode constants of the file services” on page 996. AMODE 31 callers use “BPX1MKD (mkdir) example” on page 1161.

```
MVC   BUFFERA(14),=CL14'/usr/newprots/'
MVC   BUFLENA,=F'14'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       Read search write read search
MVI   S_MODE3,S_IXUSR+S_IWGRP+S_IROTH+S_IXOTH
SPACE ,
CALL  BPX4MKD,              Make a directory                  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      S_MODE,               Input: BPXYMODE and BPXYFTYP      +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4MKN (mknod) example

The following code creates a FIFO (pipe) named **/u/fifos/fifo1** and user read-write, group read, other read permissions. For the callable service, see “mknod (BPX1MKN, BPX4MKN) — Make a directory, a FIFO, a character special, or a regular file” on page 364. For the data structure, see “BPXYFTYP — File type definitions” on page 967 and “BPXYMODE — Map the mode constants of the file services” on page 996. AMODE 31 callers use “BPX1MKN (mknod) example” on page 1162.

```
MVC   BUFFERA(14),=CL14'/u/fifos/fifo1'
MVC   BUFLENA,=F'14'
XC    S_MODE,S_MODE
MVI   S_TYPE,FT_FIFO        First in - first out
MVI   S_MODE2,S_IRUSR       Read write read read
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH
SPACE ,
CALL  BPX4MKN,              Create FIFO or char special file  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      S_MODE,               Input: BPXYMODE and BPXYFTYP      +
      =A(0),                Input: Device id not used here    +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4MMI (__map_init) example

The following code creates a shared memory map with 10 map blocks each with a size of 1 meg. For the callable service, see "__map_init (BPX1MMI, BPX4MMI) — Create a mapped megabyte area" on page 352. For the data structure, see "BPXYMMG — Map interface for _map_init and _map_service" on page 991. AMODE 31 callers use "BPX1MMI (__map_init) example" on page 1162.

```
LA    R2,INITPARM           Set address of init parm list
STG   R2,INITADDR
USING _MMG_INIT_PARM,R2
XC    _MMG_INIT_PARM(_MMG_INIT_PARM_LEN),_MMG_INIT_PARM
L     R1,=F'10'             Map area to contain 10 blocks
ST    R1,_MMG_NUMBLKS       *
L     R1,=F'1'              Each block is to be 1 meg in size
ST    R1,_MMG_MEGSPERBLK    *
SPACE ,
CALL  BPX4MMI,                                          +
      (=A(MMG_INIT),        Input: Function code        +
      INITADDR,             Input: __map_init parameter list  +
      RETVAL,               Return value: 0, -1         +
      RETCODE,              Return code                 +
      RSNCODE),             Reason code                 +
      MF=(E,PLIST)          ----------------------------------
```

BPX4MMP (mmap) example

The following code changes the protection of a memory mapped area. For the callable service, see "mmap (BPX1MMP, BPX4MMP) — Map pages of memory" on page 368. AMODE 31 callers use "BPX1MMP (mmap) example" on page 1163.

```
MVC   FILEDESC,..           File descriptor
  SPACE ,
  CALL  BPX4MMP,              map pages of memory             +
        (MAP_ADDRESS,         Input: address of mapped area   +
        MAP_LENGTH,           Input: area length              +
        =A(MAP_PRIVATE),      Input: Map type                 +
        FILEDESC,             Input: File descriptor          +
        =AD(0),               Input: File offset              +
        RETURNEDADDRESS,      Output:  value mapped address  +
        RETVAL,               Return value: 0 or -1           +
        RETCODE,              Return code                     +
        RSNCODE),             Reason code                     +
        MF=(E,PLIST)          --------------------------------
```

BPX4MMS (__map_service) example

The following code creates three new data blocks within a shared memory map. For the callable service, see "__map_service (BPX1MMS, BPX4MMS) — Mapped megabyte area services" on page 356. For the data structure, see "BPXYMMG — Map interface for _map_init and _map_service" on page 991. AMODE 31 callers use "BPX1MMS (__map_service) example" on page 1163.

```
LA    R3,SRVCPARM           Set address of init parm list
STG   R3,SRVCADDR
USING _MMG_SERVICE_PARM,R3
XC  _MMG_SERVICE_PARM(_MMG_SERVICE_PARM_LEN),_MMG_SERVICE_PARM
LA    R4,MMG_NEWBLOCK       Request that a block be created
STH   R4,_MMG_SERVICETYPE
LA    R3,_MMG_SERVICE_PARM_LEN(R3) Bump to next entry
STH   R4,_MMG_SERVICETYPE   Create a second block
LA    R3,_MMG_SERVICE_PARM_LEN(R3) Bump to next entry
STH   R4,_MMG_SERVICETYPE   Create the third block
SPACE ,
CALL  BPX4MMS,                                              +
      (=A(MMG_SERVICE),     Input: Function code            +
      SRVCADDR,             Input: __map_service parm list  +
      =F'3',                Input: Three requests to process +
      _MMG_MAPTOKEN,        Map area token from INIT call   +
      RETVAL,               Return value: 0, -1             +
      RETCODE,              Return code                     +
      RSNCODE),             Reason code                     +
      MF=(E,PLIST)          ---------------------------------
```

BPX4MNT (__mount) example

The following code requests that the file system __mount the system file and ready it for use. The file system name and mount parameters are encoded into the various fields in the MNTE. See "mount (BPX1MNT) — Make a file system available" on page 377. AMODE 31 callers use "BPX2MNT (__mount) example" on page 1164.

```
LA    R14,MNTEH             R14->MNTEH and MNTE
L     R15,MNTEL             R15 = Length of MNTEH and MNTE
XR    R0,R0                 Dummy 2nd operand
XR    R1,R1                 Pad=null, length=0
MVCL  R14,R0                Null out MNTEH and MNTE
MVC   MNTEHID,=CL4'MNT2'          Version indicator
MVC   MNTEHBLEN,=A(MNTE#LENGTH)   Length of MNTE
MVC   MNTENTFSTNAME(08),=CL08'HFS'     HFS type name
MVC   MNTENTFSNAME(44),=CL44'TESTLIB.FILESYS1'  Filesystem
MVC   MNTENTMOUNTPOINT(02),=CL02'/u'   Mount point
MVC   MNTENTPATHLEN,=F'2'
MVC   MNTENTFSMODE4,=A(MNTENTFSMODERDONLY)   Filesystem mode
CALL  BPX4MNT,              Ready a file system for use       +
      (MNTEL,               Input: MNTE length (hdr + body)   +
      MNTEH,                Input: MNTE                       +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4MP (mvspause) example

The following code places this thread into an MVS WAIT, to be terminated when a user ECB specified on a prior MVSpauseInit call is POSTed. The MVS WAIT is also terminated if a signal occurs. For the callable service, see "mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals" on page 413. AMODE 31 callers use "BPX1MP (mvspause) example" on page 1164.

```
CALL  BPX4MP,              MVS Pause                        +
      (RETVAL,             Return value: 0, -1              +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4MPC (mvsprocclp) example

The following code causes all z/OS UNIX-related resources to be released for this thread, and if this is the last thread in the process, for the process. For the callable service, see "mvsprocclp (BPX1MPC, BPX4MPC) — Clean up kernel resources" on page 418. For the data structure, see "BPXYWAST — Map the wait status word" on page 1069. AMODE 31 callers use "BPX1MPC (mvsprocclp) examples" on page 1164.

```
XC    WAST(WAST#LENGTH),WAST
MVI   WASTEXITCODE,57      User defined exit code
SPACE ,
CALL  BPX4MPC,             MVS Process cleanup              +
      (WAST,               Input: Ending status code 0-255  +
      RETVAL,              Return value: 0, -1 or 1         +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4MPI (mvspauseinit) example

The following code prepares the thread for a subsequent MVSpause invocation. A list of Event Control Block addresses is passed to the system with the last address having the high order bit on. This syscall will use the first ECB pointed to from the list as the signal ECB, therefore at least one ECB address must be passed to the system. For the callable service, see "mvspause (BPX1MP, BPX4MP) — Wait on user events plus signals" on page 413. AMODE 31 callers use "BPX1MPI (mvspauseinit) example" on page 1165.

```
        LA    R15,BUFFERA          Load address of ECB address list
        STG   R15,BUFA             Save address for future parameter
*                                   to be passed to BPX4MPI
        SR    R15,R15              Clear R15
        ST    R15,ECB01            Clear ECB01
        ST    R15,ECB02            Clear ECB02
        LA    R15,ECB01            Load address of first ECB
        ST    R15,BUFFERA          Save ECB address in list of
*                                     pointers
        LA    R15,ECB02            Load address of second ECB
        ST    R15,BUFFERA+4        Save ECB address in list of
*                                     pointers
        OI    BUFFERA+4,X'80'      Denote end of ECB pointers
        SPACE ,
        CALL  BPX4MPI,             MVS Pause initialize             +
              (BUFA,               Input ->list of ECB@, x'80' ended +
              RETVAL,              Return value: 0, -1              +
```

```
      RETCODE,            Return code                     +
      RSNCODE),           Reason code                     +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MPR (mprotect) example

The following code changes the protection of a memory mapped area. For the callable service, see "mprotect (BPX1MPR, BPX4MPR) — Set protection of memory mapping" on page 384. AMODE 31 callers use "BPX1MPR (mprotect) example" on page 1166.

```
CALL  BPX4MPR,            set protection of a mapped area   +
      (MAP_ADDRESS,       Input: address of mapped area     +
      MAP_LENGTH,         Input: area length                +
      =A(PROT_READ),      Input: Protection options         +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MSD (mvsunsigsetup) example

The following code detaches the invoker from being able to catch signals. For the callable service, see "mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup" on page 430. AMODE 31 callers use "BPX1MSD (mvsunsigsetup) example" on page 1166.

```
CALL  BPX4MSD,            Reregister MVS signals, this task +
      (SIRTNA,            Signal interface routine address  +
      USERWORD,           User data                         +
      INTMASK,            Default override signal set       +
      TERMMASK,           Default terminate signal set      +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4MSS (mvssigsetup) example

The following code allows the invoker to catch signals. For the callable service, see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421. AMODE 31 callers use "BPX1MSS (mvssigsetup) example" on page 1166.

```
* Each bit of the mask represents a signal 1-64.
        MVC   INTMASK(8),=XL8'F000000000000000'    Default sig 1-4
        MVC   TERMMASK(8),=XL8'F000000000000000'   Terminate sig 1-4
        LA    R15,BUFFERA
        STG   R15,USERWORD
        SPACE ,
        CALL  BPX4MSS,            Register MVS signals, this task   +
              (=AD(SIRTN),        Input: Signal interrupt routine   +
              USERWORD,           Input: User data                  +
              INTMASK,            Input: Default override signals   +
              TERMMASK,           Input: Default terminate signals  +
              RETVAL,             Return value: 0 or -1             +
              RETCODE,            Return code                       +
              RSNCODE),           Reason code                       +
              MF=(E,PLIST)        ---------------------------------
```

BPX4MSY (msync) example

The following code causes the file associated with this mapped area to be updated with the contents of storage. For the callable service, see "msync (BPX1MSY, BPX4MSY) — Synchronize memory with physical storage" on page 403. AMODE 31 callers use "BPX1MSY (msync) example" on page 1167.

```
MVC   FILEDESC,..          File descriptor
  SPACE ,
  CALL  BPX4MSY,             synchronize memory with storage   +
        (MAP_ADDRESS,        Input: address of mapped area     +
        MAP_LENGTH,          Input: area length                +
        =A(MS_SYNC),         Input: sync options               +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        MF=(E,PLIST)         ----------------------------------
```

BPX4MUN (munmap) example

The following code causes a mapped area to be unmapped. For the callable service, see "munmap (BPX1MUN, BPX4MUN)— Unmap previously mapped addresses" on page 407. AMODE 31 callers use "BPX1MUN (munmap) example" on page 1167.

```
  CALL  BPX4MUN,             unmap previously mapped addresses +
        (MAP_ADDRESS,        Input: address of mapped area     +
        MAP_LENGTH,          Input: area length                +
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        MF=(E,PLIST)         ----------------------------------
```

BPX4NIC (nice) example

The following code increases the priority value of the calling process by 1. For the callable service, see "nice (BPX1NIC, BPX4NIC) — Change the nice value of a process" on page 432. AMODE 31 callers use "BPX1NIC (nice) example" on page 1167.

```
MVC   INCR,=F'1'           Increase priority by 1
SPACE ,
CALL  BPX4NIC,             Change priority value             +
      (INCR,               Input: Priority change value      +
      RETVAL,              Return value: new nice value or -1+
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
L     R15,RETVAL           Load return value
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error
```

BPX4OPD (opendir) example

The following code opens directory **/etc/passwd** so that it can be read by readdir. For the callable service, see "opendir (BPX1OPD, BPX4OPD) — Open a directory" on page 452. AMODE 31 callers use "BPX1OPD (opendir) example" on page 1168.

```
MVC   BUFLENA,=F'11'
MVC   BUFFERA(11),=CL11'/etc/passwd'
```

```
SPACE ,
CALL  BPX4OPD,              Open a directory                 +
      (BUFLENA,             Input: Directory name length     +
      BUFFERA,              Input: Directory name            +
      RETVAL,               Return value:-1 or directory f.d. +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,DIRECTDES         Store the directory descriptor
```

BPX4OPN (open) example

The following code opens file **usr/inv/nov.d** with user read-write, group read and other read. A file descriptor (FILEDESC) is returned. For the callable service, see "open (BPX1OPN, BPX4OPN) — Open a file" on page 447. For the data structure, see "BPXYOPNF — Map flag values for open" on page 1004, "BPXYMODE — Map the mode constants of the file services" on page 996, and "BPXYFTYP — File type definitions" on page 967. AMODE 31 callers use "BPX1OPN (open) example" on page 1168.

```
MVC   BUFFERA(13),=CL13'usr/inv/nov.d'
MVC   BUFLENA,=F'13'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       User read/write, group read,
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH           other read
XC    O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI   O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
SPACE ,
CALL  BPX4OPN,              Open a file                      +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      O_FLAGS,              Input: Access            BPXYOPNF +
      S_MODE,               Input: Mode    BPXYMODE, BPXYFTYP +
      RETVAL,               Return value:-1 or file descriptor+
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,FILEDESC          Store the file descriptor
```

BPX4OPS (openstat) example

The following code opens file **usr/inv/nov.d.** with user read-write, group read and other read, and obtains status about the file. A file descriptor (FILEDESC) is returned. For the callable service, see "openstat (BPX2OPN, BPX4OPS) — Open a file and obtain status information" on page 454. For the data structures, see "BPXYOPNF — Map flag values for open" on page 1004, "BPXYSTAT — Map the response structure for stat" on page 1057, "BPXYMODE — Map the mode constants of the file services" on page 996, and "BPXYFTYP — File type definitions" on page 967. AMODE 31 callers use "BPX2OPN (openstat) example" on page 1168.

```
MVC   BUFFERA(13),=CL13'usr/inv/nov.d'
MVC   BUFLENA,=F'13'
XC    S_MODE,S_MODE
MVI   S_MODE2,S_IRUSR       User read/write, group read,
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IROTH           other read
XC    O_FLAGS(OPNF#LENGTH),O_FLAGS
MVI   O_FLAGS4,O_CREAT+O_RDWR Create, open for read and write
```

```
SPACE ,
CALL  BPX4OPS,              Open a file and get status        +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      O_FLAGS,              Input: Access            BPXYOPNF +
      S_MODE,               Input: Mode    BPXYMODE, BPXYFTYP +
      STATL,                Input: Length of buffer needed    +
      STAT,                 Buffer, BPXYSTAT                  +
      RETVAL,               Return value:-1 or file descriptor+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
ICM   R15,B'1111',RETVAL    Test RETVAL
BL    PSEUDO                Branch if negative (-1 = failure)
ST    R15,FILEDESC          Store the file descriptor
```

BPX4OPT (getsockopt or setsockopt) example

The following code gets and then sets socket options. SOCKDESC was returned on a previous call to BPX4SOC. For the callable service, see "getsockopt or setsockopt (BPX1OPT, BPX4OPT) — Get or set options associated with a socket" on page 275. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1OPT (getsockopt or setsockopt) example" on page 1169.

```
MVC   BUFLENA,=A(L'BUFFERA)
CALL  BPX4OPT,                  Get socket options            +
      (SOCKDESC,                Input: Socket Descriptor      +
      =A(SOCK#OPTOPTGETSOCKOPT), Input: Indicate Get socket   +
      SOCK#SOL_SOCKET,          Input: Level                  +
      SOCK#SO_TYPE,             Input: Option name            +
      BUFLENA,                  Input: Length - option value  +
      BUFFERA,                  Input: Option value           +
      RETVAL,                   Return value: 0 or -1         +
      RETCODE,                  Return code                   +
      RSNCODE),                 Reason code                   +
      MF=(E,PLIST)          ----------------------------------
SPACE ,
MVC   BUFLENA,=A(4)             SO_OOBINLINE has length=4
CALL  BPX4OPT,                  Set socket options            +
      (SOCKDESC,                Input: Socket Descriptor      +
      =A(SOCK#OPTOPTSETSOCKOPT), Input: Indicate set socket   +
      SOCK#SOL_SOCKET,          Input: Level                  +
      SOCK#SO_TYPE,             Input: Option name            +
      BUFLENA,                  Input: Length - option value  +
      SOCK#SO_OOBINLINE,        Input: Option value           +
      RETVAL,                   Return value: 0 or -1         +
      RETCODE,                  Return code                   +
      RSNCODE),                 Reason code                   +
      MF=(E,PLIST)          ----------------------------------
```

BPX4PAF (__pid_affinity) example

The following code will add your PID to the target process' affinity list. For the callable service, see "__pid_affinity (BPX1PAF, BPX4PAF) — Add or delete an entry in a process's affinity list" on page 477. AMODE 31 callers use "BPX1PAF (__pid_affinity) example" on page 1170.

```
*       MVC   TARPID,....       PID of target
*       MVC   SIGPID,....       PID of this routine
        CALL  BPX4PAF,                                        +
              (=A(PAF_ADD_PID#), Function code (add entry)    +
              TARPID,           PID of target                 +
```

```
      SIGPID,              PID to receive signal            +
      =A(SIGUSR1#),        signal to be generated           +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PAS (pause) example

The following code suspends execution of the invoker's thread until a signal is delivered. For the callable service, see “pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal” on page 468. AMODE 31 callers use “BPX1PAS (pause) example” on page 1170.

```
CALL  BPX4PAS,             Suspend execution                +
      (RETVAL,             Return value: -1 or not return   +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PCF (pathconf) example

The following code extracts the current value for the configurable maximum number of bytes in a file name associated with **/usr/inv/network.t**. For the callable service, see “pathconf (BPX1PCF, BPX4PCF) — Determine configurable path name variables using a path name” on page 464. For the data structure, see “BPXYPCF — Command values for pathconf and pathconf” on page 1005. AMODE 31 callers use “BPX1PCF (pathconf) example” on page 1170.

```
MVC   BUFFERA(18),=CL18'/usr/inv/network.t'
MVC   BUFLENA,=F'18'
SPACE ,
CALL  BPX4PCF,             Get configurable pathname variable+
      (BUFLENA,            Input: Pathname length           +
      BUFFERA,             Input: Pathname                  +
      =A(PC_NAME_MAX),     Input: Options           BPXYPCF +
      RETVAL,              Return value: 0, -1 or variable  +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PCT (pfsctl) example

The following code conveys a command to a Physical File System named ACMEFILE. ACMEFILE doesn't really exist; to actually run this example you would need a real PFS product that supports this function. For the callable service, see “pfsctl (BPX1PCT, BPX4PCT) — Physical file system control” on page 470. AMODE 31 callers use “BPX1PCT (pfsctl) example” on page 1170.

```
MVC   FSTYPE(8),=CL08'ACMEFILE'
MVC   BUFLENA,=F'25'
MVC   BUFFERA(25),=CL25'COMPRESS(ON) CONVERT(OFF)'
MVC   COMMAND,=F'123'      PFS product defined command
SPACE ,
CALL  BPX4PCT,             PFS Control                      +
      (FSTYPE,             Input: PFS Type Name             +
      COMMAND,             Input: Command                   +
      BUFLENA,             Input: Argument length           +
      BUFFERA,             Input/Output: Argument buffer    +
      RETVAL,              Return value: product defined    +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PIP (pipe) example

The following code creates a pipe. For the callable service, see "pipe (BPX1PIP, BPX4PIP) — Create an unnamed pipe" on page 481. AMODE 31 callers use "BPX1PIP (pipe) example" on page 1171.

```
CALL  BPX4PIP,             Create a pipe                    +
      (READFD,             Output: Read file descriptor     +
      WRITEFD,             Output: Write file descriptor    +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4POE (__poe) example

The following code registers a socket (SOCKDESC) as the process scope port of entry. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see "__poe() (BPX1POE, BPX4POE) — Port of entry information" on page 483. For the data structure, see "BPXYPOE — Map poe syscall parameters" on page 1013. AMODE 31 callers use "BPX1POE (__poe) example" on page 1171.

```
MVC   POEOPTIONS,=A(POE#SCOPEPROCESS)
MVC   POEENTRYTYPE,=A(POE#ENTRYSOCKET)
MVC   POEENTRYLEN,=A(POE#ENTRYSOCKETLEN)
LA    R15,SOCKDESC
STG   R15,POEENTRYPTR64
CALL  BPX4POE,             Port of Entry registration       +
      (=A(POE#LEN),        Input: Length of poe structure   +
      POE,                 Input: mapped by BPXYPOE         +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4POL (poll) example

The following code issues a poll. For the callable service, see "poll (BPX1POL, BPX4POL) — Monitor activity on file descriptors and message queues" on page 488. For the data structure, see "BPXYPOLL — Map poll syscall parameters" on page 1014. AMODE 31 callers use "BPX1POL (poll) example" on page 1171.

```
      LA    R15,BUFFERA
      USING POLLFD,R15
      STG   R15,BUFA             ->BPXYPOLL structure
*     MVC   POLLHFD(4),file_descriptor_number2
      MVI   POLLEVENTS,0
      MVI   POLLEVENTS+1,POLLERDNORM
      A     R15,=A(POLLFD#LENGTH)
*     MVC   POLLHFD(4),file_descriptor_number1
      MVI   POLLEVENTS,0
      MVI   POLLEVENTS+1,POLLEWRNORM
      SPACE ,
      CALL  BPX4POL,             Create a pipe                +
            (BUFA,               Input: address of BPXYPOLL   +
            =A(2),               Input: number of BPXYPOLL structs +
```

```
      =A(0),              Input: -1, 0, milliseconds       +
      RETVAL,             Return value: 0 or -1            +
      RETCODE,            Return code                      +
      RSNCODE),           Reason code                      +
      MF=(E,PLIST)        ---------------------------------
```

BPX4PSI (pthread_setintr) example

The following code sets the interruption type of the calling thread. For the callable service, see "pthread_setintr (BPX1PSI, BPX4PSI) — Examine and change the interrupt state" on page 527. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 31 callers use "BPX1PSI (pthread_setintr) example" on page 1172.

```
CALL  BPX4PSI,            Examine and change interrupt state+
      (INTRSTATE,         Input: Interrupt state   BPXYCONS +
      RETVAL,             Return value: 0 or -1            +
      RETCODE,            Return code                      +
      RSNCODE),           Reason code                      +
      MF=(E,PLIST)        ---------------------------------
```

BPX4PST (pthread_setintrtype) example

The following code sets the interruption type of the calling thread and returns the previous interruption type. For the callable service, see "pthread_setintrtype (BPX1PST, BPX4PST) — Examine and change the interrupt type" on page 530. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 31 callers use "BPX1PST (pthread_setintrtype) example" on page 1172.

```
CALL  BPX4PST,            Examine and change interrupt type +
      (INTRTYPE,          Input: Interrupt type    BPXYCONS +
      RETVAL,             Return value: 0 or -1            +
      RETCODE,            Return code                      +
      RSNCODE),           Reason code                      +
      MF=(E,PLIST)        ---------------------------------
```

BPX4PTB (pthread_cancel) example

The following code generates a cancelation request for the target thread (THID). For the callable service, see "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495. AMODE 31 callers use "BPX1PTB (pthread_cancel) example" on page 1172.

```
CALL  BPX4PTB,            pthread_cancel                   +
      (THID,              Input: Thread ID                 +
      RETVAL,             Return Value: 0, -1, or Buf length+
      RETCODE,            Return code                      +
      RSNCODE),           Reason code                      +
      MF=(E,PLIST)        ---------------------------------
```

BPX4PTC (pthread_create) example

The following code creates a new thread. For the callable service, see "pthread_cancel (BPX1PTB, BPX4PTB) — Cancel a thread" on page 495. For the data structure, see "BPXYPTAT — Map attributes for pthread_exit_and_get" on page 1017. AMODE 31 callers use "BPX1PTC (pthread_create) example" on page 1173.

```
LA    R15,BUFFERA           Work area
STG   R15,BUFA              ->above
LA    R15,PTAT              Area mapped by BPXYPTAT
STG   R15,PTATA             ->above
MVC   PTATEYE,=C'BPXYPTAT'              Set the eye-catcher
MVC   PTATLENGTH,=A(PTATUSEROFFVAL)     Length of structure
MVC   PTATSYSOFFSET,=A(PTATSYSOFFVAL)   Sys attr offset
MVC   PTATSYSLENGTH,=A(PTATSYSLENVAL)   Sys attr length
MVC   PTATUSEROFFSET,=A(0)              User attr offset
MVC   PTATUSERLENGTH,=A(0)              User attr length
LOAD  EP=INITRTN            Get address of Init Rtn
STG   R0,INITRTNA
SPACE ,
CALL  BPX4PTC,                                          +
      (INITRTNA,            Input: Init  routine address      +
      BUFA,                 Input: Work area address          +
      PTATA,                Input: Attr area Address BPXYPTAT +
      THID,                 Thread ID,  if Return value = 0   +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4PTD (pthread_detach) example

The following code detaches a thread (THID) in the calling process. For the callable service, see "pthread_detach (BPX1PTD, BPX4PTD) — Detach a thread" on page 503. AMODE 31 callers use "BPX1PTD (pthread_detach) example" on page 1173.

```
CALL  BPX4PTD,              pthread_detach                    +
      (THID,                Input: Thread ID                  +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4PTI (pthread_testintr) example

The following code causes a cancelation point. For the callable service, see "pthread_testintr (BPX1PTI, BPX4PTI) — Cause a cancellation point to occur" on page 536. AMODE 31 callers use "BPX1PTI (pthread_testintr) example" on page 1173.

```
CALL  BPX4PTI,              Cause an interrupt point to occur +
      (RETVAL,              Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4PTJ (pthread_join) example

The following code gets the termination status of a specified thread (THID). For the callable service, see "pthread_join (BPX1PTJ, BPX4PTJ) — Wait on a thread" on page 509. AMODE 31 callers use "BPX1PTJ (pthread_join) example" on page 1174.

```
CALL  BPX4PTJ,              pthread_join                      +
      (THID,                Input: Thread ID                  +
      =AD(0),               Input: ->Status Field or 0        +
      RETVAL,               Return value: 0 or -1             +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PTK (pthread_kill) example

The following code sends a signal to a specified thread (THID). For the callable service, see “pthread_kill (BPX1PTK, BPX4PTK) — Send a signal to a thread” on page 512. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 31 callers use “BPX1PTK (pthread_kill) example” on page 1174.

```
MVC   SIGNAL,=A(SIGALRM#)   Input: SIGALRM           BPXYSIGH
MVC   SIGNALOPTIONS,=XL4'00000000' Input: Signal options
CALL  BPX4PTK,              pthread_kill                     +
      (THID,                Input: Thread ID                 +
      SIGNAL,               Input: Signal or 0       BPXYSIGH +
      SIGNALOPTIONS,        Input: Signal options            +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4PTQ (pthread_quiesce) example

The following code terminates all other pthreads in the caller's process. For the callable service, see “pthread_quiesce (BPX1PTQ, BPX4PTQ) — Quiesce threads in a process” on page 515. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 31 callers use “BPX1PTQ (pthread_quiesce) example” on page 1174.

```
CALL  BPX4PTQ,              pthread_quiesce                  +
      (=A(QUIESCE_TERM),    Input: Quiesce type      BPXYCONS +
      =AD(0),               Input: User data - Catch data PPSD+
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4PTR (ptrace) example

The following code enables a process (PROCID) to be debugged with ptrace. For the callable service, see “ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging” on page 537. For the data structure, see “BPXYPTRC — Map parameters for ptrace” on page 1018. AMODE 31 callers use “BPX1PTR (ptrace) example” on page 1174.

```
*       MVC   PROCID,  Process ID from fork
        SPACE ,
        CALL  BPX4PTR,              Debug another process            +
              (=A(PT_ATTACH),       Input: Request           BPXYPTRC +
              PROCID,               Input: Process ID                +
              =AD(0),               Input: Address                   +
              =AD(0),               Input: Data                      +
              =AD(0),               Input: Buffer                    +
              RETVAL,               Return value: 0, -1, or Request  +
              RETCODE,              Return code                      +
              RSNCODE),             Reason code                      +
              MF=(E,PLIST)          ---------------------------------
```

BPX4PTS (pthread_self) example

The following code gets the thread ID of the calling thread. For the callable service, see “pthread_self (BPX1PTS, BPX4PTS) — Query the thread ID” on page 526. AMODE 31 callers use “BPX1PTS (pthread_self) example” on page 1175.

```
CALL  BPX4PTS,             pthread_self                    +
      (THID),              Output: Thread ID               +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PTT (pthread_tag_np) example

The following code updates the pthread tag. For the callable service, see “pthread_tag_np (BPX1PTT, BPX4PTT) — Set, query, or both set and query the caller's thread tag data” on page 533. AMODE 31 callers use “BPX1PTT (pthread_tag_np) example” on page 1175.

```
LA    R15,=CL30'UPDATING MONTH-END STATISTICS'
STG   R15,PT_NEWA
LA    R15,PT_OLD
STG   R15,PT_OLDA
CALL  BPX4PTT,             pthread_tag_np                  +
      (=A(30),             Input: Length of New Tag        +
      PT_NEWA,             Input: Address of New Tag       +
      PT_OLDL,             Input: Length of Old Tag        +
      PT_OLDA,             Input: Address to store Old Tag +
      RETVAL,              Return value: 0 or -1           +
      RETCODE,             Return code:                    +
      RSNCODE),            Reason code:                    +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PTX (pthread_exit_and_get) example

The following code terminates a thread and creates a new thread. For the callable service, see “pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread” on page 505. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 31 callers use “BPX1PTX (pthread_exit_and_get) example” on page 1175.

```
CALL  BPX4PTX,             pthread_exit_and_get            +
      (STATFLD,            Input: Status field             +
      OPTIONS,             Input: Options field            +
      SIGNALREG,           Input: Signal registration usrdata+
      RETVAL,              Return value: 0 or -1  ->BPXYPTXL +
      RETCODE,             Return code                     +
      RSNCODE),            Reason code                     +
      MF=(E,PLIST)         ---------------------------------
```

BPX4PWD (__passwd, __passwd__applid) example

The following code queries/changes the password of a given user ID. For the callable service, see “__passwd, __passwd__applid (BPX1PWD, BPX4PWD) — Verify or change security information” on page 459. AMODE 31 callers use “BPX1PWD (__passwd, __passwd__applid) example” on page 1176.

```
MVC   USERNLEN,=F'8'
MVC   USERNAME(8),=CL8'Myuserid'
MVC   OLDPASSLEN,=F'8'
MVC   OLDPASS(8),=CL8'MyOldPwd'
MVC   NEWPASSLEN,=F'8'
```

```
MVC   NEWPASS(8),=CL8'MyNewPwd'
SPACE ,
CALL  BPX4PWD,              Query/change user ID password     +
      (USERNLEN,            Input: Length of user ID          +
      USERNAME,             Input: User ID                    +
      OLDPASSLEN,           Input: Length of old password     +
      OLDPASS,              Input: Old password               +
      NEWPASSLEN,           Input: Length of new password     +
      NEWPASS,              Input: New password               +
      RETVAL,               Return value 0 or -1              +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4QCT (msgctl) example

The following code removes the message queue from the system. For the callable service, see "msgctl (BPX1QCT, BPX4QCT) — Perform message queue control operations" on page 388. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 31 callers use "BPX1QCT (msgctl) example" on page 1176.

```
CALL  BPX4QCT,              Message queue control (msgctl)    +
      (MSG_ID,              Input: MessageQueueID             +
      =A(IPC_RMID),         Input: Action to take     BPXYIPC +
      =AD(0),               Input: ->MSQID_DS or 0    BPXYMSG +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4QDB (querydub) example

The following code obtains the dub status information for the current task. The status indicates whether the current task has already been dubbed, is ready to be dubbed, or cannot be dubbed as a process (or thread). AMODE 31 callers use "BPX1QDB (querydub) example" on page 1176.

```
CALL  BPX4QDB,              Query DUB status for this task    +
      (RETVAL,              Return value: -1 or see BPXYCONS  +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4QGT (msgget) example

The following code creates a private message queue. For the callable service, see "msgget (BPX1QGT, BPX4QGT) — Create or find a message queue" on page 391. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 31 callers use "BPX1QGT (msgget) example" on page 1177.

```
MVI   S_TYPE,IPC_CREAT+IPC_EXCL     Error if exists
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
SPACE ,
CALL  BPX4QGT,              Create a message queue            +
      (=A(IPC_PRIVATE),     Input: Key                        +
      S_MODE,               Input: Creation flags BPXYMODE/IPC+
      RETVAL,               Return value: -1 or msg ID  +532200
```

```
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on msgget failure
ST    R15,MSG_ID            Store MSG_ID associated with key
```

BPX4QRC (msgrcv) example

The following code adds a message to the message queue identified by MSG_ID. For the callable service, see "msgrcv (BPX1QRC, BPX4QRC) — Receive from a message queue" on page 395. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 31 callers use "BPX1QRC (msgrcv) example" on page 1177.

```
LA    R15,BUFFERA           R15 -> Utility buffer
STG   R15,BUFA
USING MSGBUF,R15
MVC   MSG_TYPE(4),=A(0)
MVC   BUFLENA(4),=A(MSQ#LENGTH)
MVC   FLAGS(4),=A(0)        Wait for message
DROP  R15
SPACE ,
CALL  BPX4QSN,              Send a message (msgrcv)           +
      (MSG_ID,              Input: MessageQueueID             +
      BUFA,                 Input: ->MSGBUF           BPXYMSG +
      PRIMARYALET,          Input: ALET of message buffer     +
      BUFLENA,              Input: Length MSGBUF              +
      =AD(0),               Input: Message Type       BPXYMSG +
      FLAGS,                Input: Flags              BPXYIPC +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4QSE (quiesce) example

The following code quiesces file system TESTLIB.FILESYS1, making the files in it unavailable for use. For the callable service, see "quiesce (BPX1QSE, BPX4QSE) — Quiesce a file system" on page 570. AMODE 31 callers use "BPX1QSE (quiesce) example" on page 1177.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX4QSE,              Quiesce a file system             +
      (FSNAME,              Input: File system name (44 char) +
      RETVAL,               Return value: 0, -1, or 4         +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4QSN (msgsnd) example

The following code adds a message to the message queue identified by MSG_ID. For the callable service, see "msgsnd (BPX1QSN, BPX4QSN) — Send to a message queue" on page 399. For the data structure, see "BPXYMSG — Map interprocess communication message queues" on page 997. AMODE 31 callers use "BPX1QSN (msgsnd) example" on page 1178.

```
LA    R15,BUFFERA           R15 -> Utility buffer
STG   R15,BUFA
USING MSGBUF,R15
MVC   MSG_TYPE(4),=A(0)
MVC   MSG_MTEXT(11),=CL11'QSN MSG TEXT'
MVC   BUFLENA(4),=A(15)
MVC   FLAGS(4),=A(IPC_NOWAIT)   Don't wait on queue full
DROP  R15
SPACE ,
CALL  BPX4QSN,              Send a message (msgsnd)           +
      (MSG_ID,              Input: MessageQueueID             +
      BUFA,                 Input: ->MSGBUF           BPXYMSG +
      PRIMARYALET,          Input: ALET of message buffer     +
      BUFLENA,              Input: Length MSGBUF              +
      FLAGS,                Input: Flags              BPXYIPC +
      RETVAL,               Return value: 0, -1               +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4RCV (recv) example

The following code issues a recv for a socket. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see "recv (BPX1RCV, BPX4RCV) — Receive data on a socket and store it in a buffer" on page 597. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 31 callers use "BPX1RCV (recv) example" on page 1178.

```
SPACE ,
CALL  BPX4RCV,              Receive data on from a socket     +
      (SOCKDESC,            Input: Socket Descriptor          +
      =A(L'BUFFERA),        Input: Length of input buffer     +
      BUFFERA,              Input: Address of input buffer    +
      PRIMARYALET,          Input: Alet of input buffer       +
      MSG_FLAGS,            Input: Flags                      +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4RDD (readdir) example

The following code reads multiple name entries from the specified directory (DIRECTDES). For the callable service, see "readdir (BPX1RDD, BPX4RDD) — Read an entry from a directory" on page 577. For the data structure, see "BPXYDIRE — Map directory entries for readdir" on page 965. AMODE 31 callers use "BPX1RDD (readdir) example" on page 1179.

```
MVC   DIRECTDES,..          Directory descriptor from  opendir
  LA    R15,BUFFERA
  STG   R15,BUFA
  MVC   BUFLENA,=F'1023'
  CALL  BPX4RDD,              Read entries from a directory     +
        (DIRECTDES,           Input: Directory file descriptor  +
        BUFA,                 Output: ->buffer         BPXYDIRE +
        PRIMARYALET,          Input: buffer ALET                +
        BUFLENA,              Input: buffer size                +
        RETVAL,               Return value: 0, -1, entries read +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4RDL (readlink) example

The following code reads the contents of symbolic link **/personnel/templink** into the buffer provided. This will be the pathname that was specified when the symbolic link was defined. For the callable service, see "readlink (BPX1RDL, BPX4RDL) — Read the value of a symbolic link" on page 587. AMODE 31 callers use "BPX1RDL (readlink) example" on page 1179.

```
MVC   BUFFERB(19),=CL19'/personnel/templink'
MVC   BUFLENB,=F'19'
LA    R15,BUFFERA
STG   R15,BUFA
MVC   BUFLENA,=F'1023'
SPACE ,
CALL  BPX4RDL,              Read the value of a symbolic link +
      (BUFLENB,             Input: Linkname length            +
      BUFFERB,              Input: Link name                  +
      BUFLENA,              Input: Buffer size - 1023         +
      BUFA,                 ->Buffer for symbolic link        +
      RETVAL,               Return value: 0, -1 or char count +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4RDV (readv) example

The following code issues a readv for a socket. SOCKDESC was returned previously from a call to either BPX4SOC or BPX4ACP. For the callable service, see "readv (BPX1RDV, BPX4RDV) — Read data and store it in a set of buffers" on page 590. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 31 callers use "BPX1RDV (readv) example" on page 1179.

```
SPACE ,
 LA    R2,BUFFERA
 STG   R2,IOV_BASE
 LA    R2,L'BUFFERA
 STG    R2,IOV_LEN
 CALL  BPX4RDV,              Read into a vector of buffers     +
       (SOCKDESC,            Input: Socket Descriptor          +
       =A(1),                Input: Number of elements in iov  +
       IOV,                  Input: Iov containing info        +
       PRIMARYALET,          Input: Alet where iov resides     +
       PRIMARYALET,          Input: Alet of buffers for data   +
       RETVAL,               Return value: 0 or -1             +
       RETCODE,              Return code                       +
       RSNCODE),             Reason code                       +
       MF=(E,PLIST)          ----------------------------------
```

BPX4RDX (read extlink) example

The following code reads the contents of external symbolic link **/personnel/tmpxlink** into the buffer provided. This will be the pathname that was specified when the external symbolic link was defined. For the callable service, see "read_extlink (BPX1RDX, BPX4RDX) — Read an external symbolic link" on page 584. AMODE 31 callers use "BPX1RDX (read extlink) example" on page 1180.

```
MVC   BUFFERB(19),=CL19'/personnel/tmpxlink'
MVC   BUFLENB,=F'19'
LA    R15,BUFFERA
STG   R15,BUFA
```

```
  MVC   BUFLENA,=F'1023'
  SPACE ,
  CALL  BPX4RDX,              Read value of an external link    +
        (BUFLENB,             Input: Linkname length            +
        BUFFERB,              Input: Link name                  +
        BUFLENA,              Input: Buffer size - 1023         +
        BUFA,                 ->Buffer for symbolic link        +
        RETVAL,               Return value: 0, -1 or char count +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4RD2 (readdir2) example

The following code reads multiple name entries from the specified directory (DIRECTDES). FUIOCURSOR, set to zero by the BPXYFUIO macro, indicates that the system is to begin reading with the first entry in the directory. For the callable service, see "readdir2 (BPX1RD2, BPX4RD2) — Read an entry from a directory" on page 580. For the data structure, see "BPXYDIRE — Map directory entries for readdir" on page 965. AMODE 31 callers use "BPX1RD2 (readdir2) example" on page 1180.

```
MVC   DIRECTDES,..          Directory descriptor from  opendir
  MVC   FUIOID,=CL4'FUIO'     Eye Catcher
  MVC   FUIOLEN,=AL4(FUIO#LENGTH) length
  LA    R15,BUFFERA           Set address of buffer
  STG   R15,FUIOBUFF64VADDR    for directory data in FUIO
  MVC   FUIOIBYTESRW,=F'1023' Max number of bytes to read
  MVI  FUIOFLAG2,FUIOADDR64   Set 64bit addressing
  SPACE ,
  CALL  BPX4RD2,              Read directory entries            +
        (DIRECTDES,           Input: Directory file descriptor  +
        FUIO,                 Input/output: BPXYFUIO            +
        RETVAL,               Return value: 0, -1 or char count +
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4RED (read) example

The following code reads 80 bytes from the specified file (FILEDESC) and places them in the area provided (BUFFERA). For the callable service, see "read (BPX1RED, BPX4RED) — Read from a file or socket" on page 572. AMODE 31 callers use "BPX1RED (read) example" on page 1181.

```
MVC   FILEDESC,..           File descriptor
  LA    R15,BUFFERA           Buffer
  STG   R15,BUFA              Buffer address
  MVC   BUFLENA,=F'80'        Read buffer length
  SPACE ,
  CALL  BPX4RED,              Read from a file                  +
        (FILEDESC,            Input: File descriptor            +
        BUFA,                 ->Buffer to read into             +
        PRIMARYALET,          Input: Buffer ALET                +
        BUFLENA,              Input: Number of bytes to read    +
        RETVAL,               Return value: 0, -1, or char count+
        RETCODE,              Return code                       +
        RSNCODE),             Reason code                       +
        MF=(E,PLIST)          ----------------------------------
```

BPX4REN (rename) example

The following code changes the directory name of a file from **usr/sam** to **usr/samantha**. For the callable service, see "rename (BPX1REN, BPX4REN) — Rename a file or directory" on page 607. AMODE 31 callers use "BPX1REN (rename) example" on page 1181.

```
MVC   BUFFERB(07),=CL07'usr/sam'
MVC   BUFLENB,=F'07'
MVC   BUFFERA(12),=CL12'usr/samantha'
MVC   BUFLENA,=F'12'
SPACE ,
CALL  BPX4REN,              Rename a file                    +
      (BUFLENB,             Input: Old name length           +
      BUFFERB,              Input: Old name                  +
      BUFLENA,              Input: New name length           +
      BUFFERA,              Input: New name                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4RFM (recvfrom) example

The following code issues a recv from a socket. SOCKDESC was returned from a previous call, either BPX4SOC or BPX4ACP. For the callable service, see "recvfrom (BPX1RFM, BPX4RFM) — Receive data from a socket and store it in a buffer" on page 600. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 31 callers use "BPX1RFM (recvfrom) example" on page 1181.

```
SPACE ,
MVC   MSG_FLAGS4,MSG_PEEK
CALL  BPX4RFM,              Read from a socket               +
      (SOCKDESC,            Input: Socket Descriptor         +
      =A(L'BUFFERA),        Input: Length of the input buffer +
      BUFFERA,              Input: Address of the input buffer+
      PRIMARYALET,          Input: Alet of the input buffer   +
      MSG_FLAGS,            Input: Flags                     +
      =A(L'SOCKADDR),       Input: Length of the socket addr  +
      SOCKADDR,             Input: The socket address        +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4RMD (rmdir) example

The following code removes directory **applib/user02**. For the callable service, see "rmdir (BPX1RMD, BPX4RMD) — Remove a directory" on page 615. AMODE 31 callers use "BPX1RMD (rmdir) example" on page 1182.

```
MVC   BUFFERA(13),=CL13'applib/user02'
MVC   BUFLENA,=F'13'
SPACE ,
CALL  BPX4RMD,              Remove a directory               +
      (BUFLENA,             Input: Directory name length     +
      BUFFERA,              Input: Directory to be removed   +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4RMG (resource) example

The following code retrieves system-wide resource measurement data. For the callable service, see "resource (BPX1RMG, BPX4RMG) — Measure resources" on page 611. For the data structure, see "BPXYRMON — Map resource monitor data" on page 1034. AMODE 31 callers use "BPX1RMG (resource) example" on page 1182.

```
         CALL  BPX4RMG,             Resource measurement gatherer     +
               (RMONL,              Input: Length of BPXYRMON         +
               RMON,                Input: Buffer, BPXYRMON           +
               RETVAL,              Return value: 0 or -1             +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
```

BPX4RMS (recvmsg) example

The following code issues a recvmsg for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see "recvmsg (BPX2RMS, BPX4RMS) — Receive messages on a socket and store them in message buffers" on page 604. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043, "BPXYMSGF — Map the message flags" on page 997, "BPXYMSGH — Map the message header" on page 999, and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 31 callers use "BPX2RMS (recvmsg) example" on page 1182.

```
         SPACE ,
         XC    MSGH(MSGH#LENGTH),MSGH  Clear msgh
         LA    R2,SOCKADDR
         STG   R2,MSGHNAMEPTR        Store the address of sockaddr
         LA    R2,SOCK#LEN+SOCK_SUN#LEN
         ST    R2,MSGHNAMELEN
         LA    R2,IOV
         STG   R2,MSGHIOVPTR
         MVI   MSGHIOVNUM,1
         LA    R2,BUFFERA
         STG   R2,IOV_BASE
         LA    R2,L'BUFFERA
         STG   R2,IOV_LEN
*
         CALL  BPX4RMS,             Receive a message from a socket   +
               (SOCKDESC,           Input: Socket Descriptor          +
               MSGH,                Input: Address of BPXYMSGH        +
               MSG_FLAGS,           Input: Flags                      +
               PRIMARYALET,         Input: Alet of the iov            +
               PRIMARYALET,         Input: Alet of the buffers in iov +
               RETVAL,              Return value: 0 or -1             +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
```

BPX4RPH (realpath) example

The following code gets the absolute pathname without dot (.), dot-dot (..), or symbolic links for the input pathname. For the callable service, see "realpath (BPX1RPH, BPX4RPH) — Resolve a pathname" on page 594. AMODE 31 callers use "BPX1RPH (realpath) example" on page 1183.

```
         MVC   BUFFERA(8),=CL2'..'
         MVC   BUFLENA,=F'2'
```

```
MVC   BUFLENB,=F'1024'      Resolved pathname return area
SPACE ,
CALL  BPX4RPH,              Resolve pathname                 +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      BUFLENB,              Input: Length resolved name area +
      BUFFERB,              Output: Resolved name buffer     +
      RETVAL,               Return value: -1 or length       +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4RW (Pwrite) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). It will start writing at specified offset, 30 bytes from start of the file. To positional read from a file, change the FUIORWIND to indicate FUIO#RD. For the callable service, see "Pread() and Pwrite() (BPX1RW, BPX4RW) — Read from or write to a file without changing the file pointer" on page 492. AMODE 31 callers use "BPX1RW (Pwrite) example" on page 1183.

```
MVC   FILEDESC,             File descriptor from open
  XC    FUIO,FUIO             Zero out Fuio fields
  MVC   FUIOID,=CL4'FUIO'     Eye Catcher
  MVC   FUIOLEN,=AL4(FUIO#LENGTH) length
  LA    R15,BUFFERA           Set address of buffer
  STG   R15,FUIOBUFFERADDR    for buffer data in FUIO
  MVI   FUIORWIND,FUIO#WRT    Flag to indicate to PWrite
  MVC   FUIOIBYTESRW,=F'80'   Number of bytes to Write
  MVC   FUIOCUR2,=F'30'       Offset to start writing
  MVI  FUIOFLAG2,FUIOADDR64   Set 64bit addressing
  LA    R15,FUIO              Set address of Fuio
  STG   R15,LFUIOPTR          For access to Fuio fields
  SPACE ,
  CALL  BPX4RW,               PWrite to a file                 +
        (FILEDESC,            Input: File descriptor           +
        LFUIOPTR,             Input: Address of FUIO struct    +
        PRIMARYALET,          Input: Fuio ALET                 +
        FUIOLEN,              Input: Fuio Length               +
        RETVAL,               Return value: -1 or bytes written +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4RWD (rewinddir) example

The following code resets the open directory to the beginning. For the callable service, see "rewinddir (BPX1RWD, BPX4RWD) — Reposition a directory stream to the beginning" on page 613. AMODE 31 callers use "BPX1RWD (rewinddir) example" on page 1184.

```
MVC   DIRECTDES,..          File descriptor from opendir
  CALL  BPX4RWD,              Reposition directory at beginning +
        (DIRECTDES,           Input: Directory file descriptor  +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4SA2 (__sigactionset) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see "__sigactionset (BPX1SA2, BPX4SA2) — Examine or change a set of signal actions" on page 751. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 31 callers use "BPX1SA2 (__sigactionset) example" on page 1184.

```
XC    R15,R15
ST    R15,SSETOPTION_FLAGS
OI    SSETOPTION_FLAGS1,SSET_IGINVALID
LA    R14,1
ST    R11,BUFCNTB
LA    R14,BUFFERA
USING SSET,R14
MVC   SSETFLAGS,=XL4'00000000'
MVC   SSETSAMASK,=XL8'0FFF0F0000000000'
MVC   SSETSAHANDLER,EPADDR
MVC   SSETUSERDATA,=CL4'DATA'
DROP  R14
SPACE ,
CALL  BPX4SA2,              Examine/change multiple sig acts  +
      (=A(1),               Input: One SSET set               +
      BUFFERA,              Input: Signal set input BPXYSSET  +
      BUFCNTB,              In/Out: Number of array elements  +
      BUFFERB,              Output: Address of output struct  +
      SSETOPTION_FLAGS,     Input: Mapped by BPXYSSET         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SCT (semctl) example

The following code retrieves the PID of the last process toupdate semaphore 4 from the SEM_ID semaphore set. For the callable service, see "semctl (BPX1SCT, BPX4SCT) — Perform semaphore control operations" on page 626. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037. AMODE 31 callers use "BPX1SCT (semctl) example" on page 1184.

```
LA    R15,BUFFERA
STG   R15,BUFA
MVC   SEM_NUMBER(4),4       Semaphore number 4 in set
SPACE ,
CALL  BPX4SCT,              Semaphore control operations      +
      (SEM_ID,              Input: Semaphore set ID           +
      SEM_NUMBER,           Input: Semaphore number (0 based) +
      =A(SEM_GETPID),       Input: Action to take     BPXYSEM +
      BUFA,                 Input: Value | Buffer | Array | 0 +
      RETVAL,               Return value: 0, -1 or value      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SDD (setdubdefault) example

The following code sets the dub default setting for the subtasks of the caller to process. For the callable service, see "set_dub_default (BPX1SDD, BPX4SDD) — Set the dub default service" on page 666. AMODE 31 callers use "BPX1SDD (setdubdefault) example" on page 1185.

```
CALL  BPX4SDD,              Set effective group ID           +
      (=A(DUBPROCESS),      Input: Set Dub Constant  BPXYCONS +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SEC (__login, __login__applid, __certificate) example

The following code will invoke RACF (or other security product) to create a security environment (ACEE) for the calling process with the identity of JOEUSER. For the callable service, see "__login, __login__applid, __certificate (BPX1SEC, BPX4SEC) — Provides an interface to the security product" on page 309. AMODE 31 callers use "BPX1SEC (__login, __login__applid, __certificate) example" on page 1185.

```
MVC   USERNLEN,=F'7'
MVC   USERNAME(7),=CL7'JOEUSER'
MVC   OLDPASSLEN,=F'8'
MVC   OLDPASS,=CL8'JOESPASS'
MVC   OPTIONS,=F'0'
SPACE ,
CALL  BPX4SEC,              Create security environment       +
      (=A(SECURITY_CREATE#), Input: Function_code    BPXYCONS +
      SECURITY_USERID#,      Input: ID-Type          BPXYCONS +
      USERNLEN,              Input: UserID Length             +
      USERNAME,              Input: UserID                    +
      OLDPASSLEN,            Input: Password Length           +
      OLDPASS,               Input: Password                  +
      =A(0),                 Input: Holder                    +
      =A(0),                 Input: Holder                    +
      OPTIONS,               Input: Options                   +
      RETVAL,                Return value: 0 or -1            +
      RETCODE,               Return code                      +
      RSNCODE),              Reason code                      +
      MF=(E,PLIST)           ---------------------------------
```

BPX4SEG (setegid) example

The following code sets the effective group ID of the invoker to 1. For the callable service, see "setegid (BPX1SEG, BPX4SEG) — Set the effective group ID" on page 670. AMODE 31 callers use "BPX1SEG (setegid) example" on page 1185.

```
MVC   GROUPID,=XL4'00000001' Value of new effective ID
SPACE ,
CALL  BPX4SEG,              Set effective group ID           +
      (GROUPID,             Input: Group ID                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SEL (select) example

The following code issues a select for a previously connected socket. SOCKDESC was returned when the socket was created. In this case, the select is for a single socket for read, write and exception. Do not request waiting. There are no ECBs. For the callable service, see "select/selectex (BPX1SEL, BPX4SEL) — Select on file descriptors and message queues" on page 618. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and

"BPXYSEL — Map the select options" on page 1036. AMODE 31 callers use "BPX1SEL (select) example" on page 1186.

```
         SPACE ,
*
         MVC   SELLIST(4),=XL4'81000000'                            +
                                Turn on the bit representing sd 0 +
                                and sd 7
         LA    R8,8              One more than largest descriptor
         ST    R8,SOCKDESC       Set number of sockets to check
*
         CALL  BPX4SEL,          Select on a set of sockets        +
               (SOCKDESC,        Input: Number of file descriptors +
               =A(4),            Input: Length of read list        +
               SELLIST,          Input: Read list                  +
               =A(4),            Input: Length of write list       +
               SELLIST,          Input: Write list                 +
               =A(4),            Input: Length of exception list   +
               SELLIST,          Input: Exception list             +
               =AD(0),           Input: Address of Timeout value   +
               =AD(0),           Input: ECB pointer                +
               =A(SEL#BITSFORWARD), Input: Option - bits forward   +
               RETVAL,           Return value: 0 or -1             +
               RETCODE,          Return code                       +
               RSNCODE),         Reason code                       +
               MF=(E,PLIST)      ----------------------------------
```

BPX4SEU (seteuid) example

The following code sets the effective user ID of the invoker to 1. For the callable service, see "seteuid (BPX1SEU, BPX4SEU) — Set the effective user ID" on page 672. AMODE 31 callers use "BPX1SEU (seteuid) example" on page 1186.

```
         MVC   USERID,=XL4'00000001' Value of new effective user ID
         SPACE ,
         CALL  BPX4SEU,          Set effective user ID             +
               (USERID,          Input: User ID                    +
               RETVAL,           Return value: 0 or -1             +
               RETCODE,          Return code                       +
               RSNCODE),         Reason code                       +
               MF=(E,PLIST)      ----------------------------------
```

BPX4SF (send_file) example

The following code create a parameter list to send the contents of the specified file to the designated socket. to 1. For the callable service, see "send_file (BPX1SF, BPX4SF) — Send a file on a socket" on page 643. AMODE 31 callers use "BPX1SF (send_file) example" on page 1187.

```
         LA    R5,BUFFERA
         ST    R5,BUFFR
         USING SFPL,R5
         XC    SFPL(SFPL#LENGTH),SFPL  Initialize to nulls (required)
* NULLS= no header, no trailer, start at offset 0
*        MVC   SFFileDes,...                  Read from file
*        MVC   SFSocketDes,...                Write to Socket
         MVC   SFFileBytesH,=XL4'FFFFFFFF'    To file end
         MVC   SFFileBytesL,=XL4'FFFFFFFF'    To file end
         OI    SFflagByte4,SF_Close   Close socket after write
         SPACE ,
         CALL  BPX4SF,               Send_file                    +
               (=A(SFPL#LENGTH),     Input: Length of BPXYSFPL    +
               BUFFERA,              Input: ->SFPL                +
```

```
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4SGE (setgrent) example

The following code resets the group database to the beginning, so that a subsequent BPX4GGE call will restart the group database search from the first entry. For the callable service, see "setgrent (BPX1SGE, BPX4SGE) — Reset the group database" on page 677. AMODE 31 callers use "BPX1SGE (setgrent) example" on page 1187.

```
CALL  BPX4SGE,             Reset the group database         +
      (RETVAL),            Return value: 0                  +
      MF=(E,PLIST)         ---------------------------------
```

BPX4SGI (setgid) example

The following code sets the real, effective, and save group IDs to 1. For the callable service, see "setgid (BPX1SGI, BPX4SGI) — Set the group ID" on page 674. AMODE 31 callers use "BPX1SGI (setgid) example" on page 1187.

```
MVC   USERID,=XL4'00000001' Value of new group user ID
SPACE ,
CALL  BPX4SGI,             Set group ID                     +
      (GROUPID,            Input: Group ID                  +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4SGQ (sigqueue) example

The following code queues a signal (SIGUSR1#) to the process specified by PROCID with a signal value of 0. For the callable service, see "sigqueue (BPX1SGQ, BPX4SGQ) — Queue a signal to a process" on page 760. AMODE 31 callers use "BPX1SGQ (sigqueue) example" on page 1187.

```
SPACE ,
CALL  BPX4SGQ,             Queue a signal to a process      +
      (PROCID,             Input: Process ID                +
      =A(SIGUSR1#),        Input: Signal           BPXYSIGH +
      =AD(0),              Input: Signal value              +
      =A(0),               Input: Signal options            +
      RETVAL,              Return value: -1 or 0            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4SGR (setgroups) example

The following code sets the supplementary group id list to the three gids (00000001, 000000002, 00000003) in BUFFERA. For the callable service, see "setgroups (BPX1SGR, BPX4SGR) — Set the supplementary group IDs list" on page 678. AMODE 31 callers use "BPX1SGR (setgroups) example" on page 1188.

```
LA    R15,BUFFERA
STG   R15,BUFA
MVC   BUFFERA(12),=XL12'000000010000000200000003'
```

```
SPACE ,
CALL  BPX4SGR,              Set supplementary groups list     +
      (=A(3),               Input: number of sgids in list    +
      BUFA,                 Input: address of sgids list      +
      RETVAL,               Return value: -1 or 0             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SGT (semget) example

The following code creates a private set of 10 semaphores. For the callable service, see "semget (BPX1SGT, BPX4SGT) — Create or find a set of semaphores" on page 631. For the data structure, see "BPXYSEM — Map interprocess communication semaphores" on page 1037. AMODE 31 callers use "BPX1SGT (semget) example" on page 1188.

```
MVC   KEY(4),=A(IPC_PRIVATE)  Local to this family
MVI   S_TYPE,IPC_CREAT+IPC_EXCL   Must not already exist
MVI   S_MODE1,0             Not used
MVI   S_MODE2,S_IRUSR       All read and write permissions
MVI   S_MODE3,S_IWUSR+S_IRGRP+S_IWGRP+S_IROTH+S_IWOTH
MVC   NUMB_SEMS(4),=A(10)         10 semaphores this set
SPACE ,
CALL  BPX4SGT,              Create a set of semaphores        +
      (KEY,                 Input: Semaphore key              +
      NUMB_SEMS,            Input: Number semaphores in set   +
      S_MODE,               Input: Flags    BPXYMODE / BPXYIPC+
      RETVAL,               Return value: -1 or Semaphore ID  +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
SPACE ,
ICM   R15,B'1111',RETVAL    Test return value
BNP   PSEUDO                Branch on semget failure
ST    R15,SEM_ID            Store SEM_ID associated with key
```

BPX4SHT (shutdown) example

The following code issues a shutdown to stop socket writes to this socket connection. SOCKDESC was returned from a previous call to BPX4SOC. For the callable service, see "shutdown (BPX1SHT, BPX4SHT) — Shut down all or part of a duplex socket connection" on page 743. AMODE 31 callers use "BPX1SHT (shutdown) example" on page 1189.

```
SPACE ,
CALL  BPX4SHT,              Shutdown communication            +
      (SOCKDESC,            Input: Socket Descriptor          +
      SOCK#SHUTDOWNWRITE,   Input: How - shutdown writes      +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SIA (sigaction) example

The following code sets new action for SIGALRM to default processing and returns the previous action for SIGALARM. For the callable service, see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 31 callers use "BPX1SIA (sigaction) example" on page 1189.

```
XC    NEWMASK,NEWMASK       Don't block additional signals
LA    R15,NCATCHER          New catcher (NCATCHER=0,1  ->)
STG   R15,NEWHANDL
LA    R15,OCATCHER          Old catcher (NCATCHER=0,1  ->)
STG   R15,OLDHANDL
SPACE ,
CALL  BPX4SIA,              Examine or change signal action   +
      (=A(SIGALRM#),        Input: Signal constant   BPXYSIGH +
      NEWHANDL,             Input: 0, ->0, ->1 or ->catcher   +
      NEWMASK,              Input: 64Bit mask of signals      +
      =A(0),                Input: Action, BPXYSIGH           +
      OLDHANDL,             0, ->XL4 (return 0, 1 ->catcher)  +
      OLDMASK,              64 bit mask of signals            +
      OLDFLAGS,             Action, BPXYSIGH                  +
      =AD(0),               Data passed to signal routine     +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SIN (server_init) example

The following code connects a server address space to WLM as a server manager for the WEB subsystem type, WEB1 subsystem name, and IMWHTTP application environment. For the callable service, see "server_init (BPX1SIN, BPX4SIN) — Server initialization" on page 656. AMODE 31 callers use "BPX1SIN (server_init) example" on page 1189.

```
MVC   SUBSYSTYPE,=CL4'WEB '  WEB Subsystem Type
MVC   SUBSYSNAME,=CL8'WEB1    '  WEB1 Subsystem Name
MVC   APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
LA    R15,=F'7'             R15 = 7
ST    R15,PARALLELEU        7 Parallel Execution Units
SPACE ,
CALL  BPX4SIN,              Server_init                       +
      (=A(SRV_SERVERMGR),   Input: Manager Type (Server Mgr)  +
      SUBSYSTYPE,           Input: Subsystem Type             +
      SUBSYSNAME,           Input: Subsystem Type             +
      APPLENV,              Input: Application Environment    +
      PARALLELEU,           Input: Parallel Eu                +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4SIP (sigpending) example

The following code retrieves the mask used for pending and blocked signals. For the callable service, see "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755. AMODE 31 callers use "BPX1SIP (sigpending) example" on page 1190.

```
CALL  BPX4SIP,              Determine pending signals         +
      (SIGRET,              Signal mask return area (XL8)     +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SLK (shmem_lock) example

The following code initializes a shared memory resident lock. For the callable service, see "shmem_lock (BPX1SLK, BPX4SLK) — Shared memory lock service" on page 724. AMODE 31 callers use "BPX1SLK (shmem_lock) example" on page 1190.

```
XR    R15,R15              R15 = 0
STG   R15,LOCKATTRADDR     No lock attribute Data
SPACE ,
CALL  BPX4SLK,             shmem_lock                        +
      (=A(SLK_INIT),       INPUT: Function Code (Init)       +
      =A(SLK_NORMAL),      INPUT: Request Type (Normal)      +
      =A(SLK_SHARED),      INPUT: Lock Type (Shared)         +
      LOCKADDR,            INPUT: ->user lockword (shared mem+
      LOCKATTRADDR,        INPUT: Address of lock attr area  +
      LOCKTOKENADDR,       INPUT: Address of Lock Token      +
      RETVAL,              Return value: >=0 or -1           +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
L     R15,RETVAL           Load return value
C     R15,=F'-1'           Test for -1 return
BE    PSEUDO               Branch on error
```

BPX4SLP (sleep) example

The following code suspends running for 8 seconds or until a signal is delivered (whichever comes first). For the callable service, see "sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time" on page 771. AMODE 31 callers use "BPX1SLP (sleep) example" on page 1190.

```
MVC   SECONDS,=F'8'        8 seconds
SPACE ,
CALL  BPX4SLP,             Temporarily suspend execution     +
      (SECONDS,            Input: Sleep interval in seconds  +
      RETVAL),             Return value: 0 or sleep time     +
      MF=(E,PLIST)         ----------------------------------
```

BPX4SMF (smf_record) example

The following code tests whether SMF recording is active for a specified SMF record type, and if it is, writes an SMF record. For the callable service, see "smf_record (BPX1SMF, BPX4SMF) — Write an SMF record" on page 774. AMODE 31 callers use "BPX1SMF (smf_record) example" on page 1191.

```
MVC   SMF_TYPE,=F'108'     Set SMF record type
MVC   SMF_SUBTYPE,=F'0'    Set SMF record subtype
MVC   BUFLENA,=F'0'        Set SMF record length
MVC   BUFA,=FD'0'           Zero SMF record address
CALL  BPX4SMF,             smf_record                        +
      (SMF_TYPE,           SMF record type                   +
      SMF_SUBTYPE,         SMF record subtype                +
      BUFLENA,             SMF record length                 +
      BUFA,                SMF record address set to zero    +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
ICM   R15,B'1111',RETVAL   Test return value
BNZ   QUIT                 Not recording or error, quit
SPACE ,
```

```
         MVI   BUFFERA,C' '
         MVC   BUFFERA+1(255),BUFFERA Clear SMF record
         MVI   BUFFERA+1,100         Set length in SMF header
         MVI   BUFFERA+5,108         Set SMF type in SMF header
         MVC   BUFFERA+18(16),=CL16'Here is the data'  Set SMF record
         MVC   SMF_TYPE,=F'108'      Set SMF record type
         MVC   SMF_SUBTYPE,=F'0'     Set SMF record subtype
         MVC   BUFLENA,=F'100'       Set SMF record length
         LA    R15,BUFFERA
         STG   R15,BUFA              Set SMF record address
         CALL  BPX4SMF,              smf_record                      +
               (SMF_TYPE,            SMF record type                 +
               SMF_SUBTYPE,          SMF record subtype              +
               BUFLENA,              SMF record length               +
               BUFA,                 SMF record address              +
               RETVAL,               Return value: 0 or -1           +
               RETCODE,              Return code                     +
               RSNCODE),             Reason code                     +
               MF=(E,PLIST)          --------------------------------
QUIT     EQU  *
```

BPX4SMS (sendmsg) example

The following code sends a message on a socket. SOCKDESC was returned from a previous call to BPX4SOC. For the callable service, see "sendmsg (BPX2SMS, BPX4SMS) — Send messages on a socket" on page 648. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043, "BPXYIOV — Map the I/O vector structure" on page 986, and "BPXYMSGH — Map the message header" on page 999. AMODE 31 callers use "BPX2SMS (sendmsg) example" on page 1191.

```
         XC    MSGH(MSGH#LENGTH),MSGH  Clear msgh
         LA    R2,SOCKADDR
         STG   R2,MSGHNAMEPTR        Store the address of sockaddr
         LA    R2,SOCK#LEN+SOCK_SUN#LEN
         ST    R2,MSGHNAMELEN
         LA    R2,IOV
         STG   R2,MSGHIOVPTR
         MVI   MSGHIOVNUM,1
*
         LA    R2,BUFFERA
         STG   R2,IOV_BASE
         LA    R2,16
         STG    R2,IOV_LEN
         MVC   BUFFERA(16),=CL16'Here is the data'
*
         CALL  BPX4SMS,              Send a message on a socket      +
               (SOCKDESC,            Input: Socket Descriptor        +
               MSGH,                 Input: Address of BPXYMSGH      +
               MSG_FLAGS,            Input: Flags                    +
               PRIMARYALET,          Input: Alet of the iov          +
               PRIMARYALET,          Input: Alet of the buffers in iov +
               RETVAL,               Return value: 0 or -1           +
               RETCODE,              Return code                     +
               RSNCODE),             Reason code                     +
               MF=(E,PLIST)          --------------------------------
```

BPX4SND (send) example

The following code issues a send for a socket. SOCKDESC was returned previously from a call to BPX4SOC. For the callable service, see “send (BPX1SND, BPX4SND) — Send data on a socket” on page 640. For the data structures, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043 and “BPXYMSGF — Map the message flags” on page 997. AMODE 31 callers use “BPX1SND (send) example” on page 1192.

```
MVC   BUFLENA,=F'16'
MVC   BUFFERA(16),=CL16'Here is the data'
SPACE ,
CALL  BPX4SND,              Send data on a socket            +
      (SOCKDESC,            Input: Socket Descriptor         +
      =A(L'BUFFERA),        Input: Length of input buffer    +
      BUFFERA,              Input: input buffer              +
      PRIMARYALET,          Input: Alet of input buffer      +
      MSG_FLAGS,            Input: Flags                     +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SOC (socket or socketpair) example

The following code creates a pair of stream sockets in the AF_UNIX domain. For the callable service, see “socket or socketpair (BPX1SOC, BPX4SOC) — Create a socket or a pair of sockets” on page 777. For the data structure, see “BPXYSOCK — Map SOCKADDR structure and constants” on page 1043. AMODE 31 callers use “BPX1SOC (socket or socketpair) example” on page 1192.

```
CALL  BPX4SOC,              Create a socket pair             +
      (=A(AF_UNIX),         Input: Domain of AF_UNIX         +
      =A(SOCK#_STREAM),     Input: Type of socket stream     +
      =A(0),                Input: Protocol of 0             +
      =A(2),                Input: Dimension of 2 for pair   +
      SOCKETS,              Input: Socket vector for return  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SOP (semop) example

The following code retrieves the PID of the last process to update semaphore 4 from the SEM_ID semaphore set. For the callable service, see “semop (BPX1SOP, BPX4SOP) — Perform semaphore serialization operations” on page 636. For the data structure, see “BPXYSEM — Map interprocess communication semaphores” on page 1037. AMODE 31 callers use “BPX1SOP (semop) example” on page 1193.

```
LA    R5,BUFFERA            ->Utitliy buffer
STG   R5,BUFA
USING SEM_BUF_ELE,R5        ->1st SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(0)    Semaphore number 0
MVC   SEM_OP(2),=AL2(-1)              take the resource
MVC   SEM_FLG(2),=AL2(SEM_UNDO)       flags (undo,wait)
LA    R5,SEM#BUFLEN(,R5)    ->next SEM_BUF_ELE
MVC   SEM_NUM(2),=AL2(2)              number 2
MVC   SEM_OP(2),=AL2(1)               release the resource
MVC   SEM_FLG(2),=AL2(IPC_NOWAIT)     flags (nowait)
LA    R5,SEM#BUFLEN(,R5)              ->next SEM_BUF_ELE
```

```
MVC   SEM_NUM(2),=AL2(8)             number 8
MVC   SEM_OP(2),=AL2(0)              test for no resource
MVC   SEM_FLG(2),=AL2(0)             flags (wait)
SPACE ,
MVC   NUMB_SEM_OPS(4),=AL2(3) number of SEM_BUF_ELE in BUFFERA
SPACE ,
CALL  BPX4SOP,              Semaphore control operations      +
      (SEM_ID,              Input: Semaphore set ID           +
      BUFA,                 Input: ->SEM_BUF_ELE      BPXYSEM +
      NUMB_SEM_OPS,         Input: Action to take             +
      RETVAL,               Return value: 0, -1 or value      +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SPB (queue_interrupt) example

The following code uses the queue_interrupt to return the last signal delivered to the signal interface routine (SIR). For the callable service, see "queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered" on page 568. AMODE 31 callers use "BPX1SPB (queue_interrupt) example" on page 1193.

```
CALL  BPX4SPB,              Queue the signal                  +
      (RETVAL,              Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SPE (setpwent) example

The following code resets the user database to the beginning, so that a subsequent BPX4GPE call will restart the user database search from the first entry. For the callable service, see "setpwent (BPX1SPE, BPX4SPE) — Reset the user database" on page 691. AMODE 31 callers use "BPX1SPE (setpwent) example" on page 1193.

```
CALL  BPX4SPE,              Reset the user database           +
      (RETVAL),             Return value: 0                   +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SPG (setpgid) example

The following code places the invoking process in its own process group (zeros indicate that the process group ID is to be set to the process ID). For the callable service, see "setpgid (BPX1SPG, BPX4SPG) — Set a process group ID for job control" on page 686. AMODE 31 callers use "BPX1SPG (setpgid) example" on page 1194.

```
MVC   PROCID,=A(0)          Process ID - current to leader
MVC   GROUP,=A(0)           Group ID - current to leader
SPACE ,
CALL  BPX4SPG,              Set process group ID for Job Ctl  +
      (PROCID,              Input: Process to be placed in grp+
      GROUP,                Input: Target group               +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SPM (sigprocmask) example

The following code changes the signal mask to block signals 1 through 16. For the callable service, see “sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask” on page 757. For the data structure, see “BPXYSIGH — Signal constants” on page 1039. AMODE 31 callers use “BPX1SPM (sigprocmask) example” on page 1194.

```
LA    R15,=XL8'FFFF000000000000'  Block signals 1 thru 16
ST    R15,NEWMASKA          New mask address
LA    R15,OLDMASK           Old signal mask
ST    R15,OLDMASKA          Old mask address
SPACE ,
CALL  BPX4SPM,              Examine or change signal mask     +
      (=A(SIG_BLOCK#),      Input: How parameter BPXYSIGH     +
      NEWMASKA,             Input: 0, ->CL8                   +
      OLDMASKA,             Input: 0 | ->returned mask        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SPN (spawn) example

The program ictasma located at **ict/bin** gets control as a child process of the caller, and is passed arguments WK18, DEPT37A, and RATE(STD,NOEXC,NOSPEC). No environment arguments are passed. The file descriptor count is set to 0, indicating that the child shall inherit all of the parent's file descriptors. The inheritance area passed is set to all zeroes, indicating that the child shall inherit the parent's attributes without change. For the callable service, see “spawn (BPX1SPN, BPX4SPN) — Spawn a process” on page 780. AMODE 31 callers use “BPX1SPN (spawn) example” on page 1194.

```
        MVC   BUFLENA,=F'16'
        MVC   BUFFERA(16),=C'/ict/bin/ictasma'
        MVC   ARGCNT,=F'3'
*                                 First
        LA    R15,=F'4'               Length
        ST    R15,ARGLLST+00          Length parm list
        LA    R15,=CL4'WK18'          Argument
        STG   R15,ARGSLST+00          Argument address parm list
*                                 Second
        LA    R15,=F'7'               Length
        ST    R15,ARGLLST+04          Length parm list
        LA    R15,=CL7'DEPT37A'       Argument
        STG   R15,ARGSLST+08          Argument address parm list
*                                 Third
        LA    R15,=F'22'              Length
        ST    R15,ARGLLST+08          Length parm list
        LA    R15,=CL22'RATE(STD,NOEXC,NOSPEC)'    Argument
        STG   R15,ARGSLST+16          Argument address parm list
*
        MVC   ENVCNT,=F'0'        Zero environment args passed
        MVC   ENVLENS,=F'0'       Addr of env. data length list
        MVC   ENVPARMS,=F'0'      Add of env. data
*
        MVC   FDCNT,=F'0'         Zero file descriptors passed
        MVC   FDLST,=F'0'         File Descriptor list
*
        XC    INHE(INHE#LENGTH),INHE  Clear Inheritance structure
        SPACE ,
        CALL  BPX4SPN,                                          +
              (BUFLENA,           Input: Pathname length        +
```

```
      BUFFERA,              Input: Pathname                  +
      ARGCNT,               Input: Argument count            +
      ARGLLST,              Input: Argument length list      +
      ARGSLST,              Input: Argument address list     +
      ENVCNT,               Input: Environment count         +
      ENVLENS,              Input: Environment length list   +
      ENVPARMS,             Input: Environment address list  +
      FDCNT,                Input: File desriptor count      +
      FDLST,                Input: File descriptor list      +
      =A(INHE#LENGTH),      Input: Length of Inheritance area +
      INHE,                 Input: Inheritance area          +
      RETVAL,               Return value: Child PID or -1    +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SPR (setpeer) example

The following code issues a setpeer to set up the host address. For the callable service, see "setpeer (BPX1SPR, BPX4SPR) — Preset the peer address associated with a socket" on page 684. For the data structure, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043. AMODE 31 callers use "BPX1SPR (setpeer) example" on page 1195.

```
CALL  BPX4SPR,              Select on a set of sockets       +
      (SOCKDESC,            Input: Socket Descriptor         +
      SOCK#LEN+SOCK_SUN#LEN, Input: Length of socket address +
      SOCKADDR,             Input: Socket address            +
      SOCK#SO_SET,          Input: Option - set the address  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SPW (server_pwu) example

The following code puts work to the WLM work queue for the IMWHTTP application environment for transaction class A. For the callable service, see "server_pwu (BPX1SPW, BPX4SPW) — Server process work unit" on page 660. AMODE 31 callers use "BPX1SPW (server_pwu) example" on page 1195.

```
MVC   APPLENV,=CL8'IMWHTTP ' IMWHTTP Application Environment
MVC   TRXCLASS,=CL8'A       ' Transaction Class A
XR    R15,R15               R15 = 0
ST    R15,CLASSIFYLEN       No Classification Data
ST    R15,APPLDATALEN       No Application Data
ST    R15,FDLISTPTR         No File Descriptor List
SPACE ,
CALL  BPX4SPW,              Server_pwu                       +
      (=A(SRV_PUT_NEWWRK),  Input: Function Code (Putwork)   +
      TRXCLASS,             Input: Transaction Class         +
      APPLENV,              Input: Application Environment   +
      CLASSIFYLEN,          Input: Classification Area Length +
      CLASSIFYAREAPTR,      Input: Classification Area Address+
      APPLDATALEN,          Input: Application Data Length   +
      APPLDATAPTR,          Input: Application Data Address  +
      FDLISTPTR,            Input: Mapped by BPXYSFDL        +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

```
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4SPY (setpriority) example

The following code sets the CPU priority based on the input which and who values. The which value used is PRIO_PROCESS, which indicates that the priority is to be set by process ID. The who value used is 7, to set the priority for process ID 7. For the callable service, see "setpriority (BPX1SPY, BPX4SPY) — Set the scheduling priority of a process" on page 688. AMODE 31 callers use "BPX1SPY (setpriority) example" on page 1196.

```
MVC   PROCID,=XL4'00000007' Process ID to set priority for
MVC   PRIORITY,=XL4'00000001' Priority value of 1
SPACE ,
CALL  BPX4SPY,              Set priority value               +
      (=A(PRIO_PROCESS),    Input: Set by Process ID         +
      PROCID,               Input: PID to set priority for   +
      PRIORITY,             Input: Priority value to set to  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4SRG (setregid) example

The following code sets the real and/or effective group IDs to 1. For the callable service, see "setregid (BPX1SRG, BPX4SRG) — Set the real and/or effective GIDs" on page 693. AMODE 31 callers use "BPX1SRG (setregid) example" on page 1196.

```
  MVC   RGID,=XL4'00000001'   Value of new real group ID
MVC   RGID,..               Group ID to be set from a getgid
  MVC   EGID,=XL4'00000001'   Value of new effective group ID
MVC   EGID,..               Group ID to be set from a getegid
  SPACE ,
  CALL  BPX4SRG,              Set Group IDs                    +
        (RGID,                Input: Real Group ID to be set   +
        EGID,                 Input: Eff. Group ID to be set   +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4SRL (setrlimit) example

The following code sets the resource limits for the calling process based on the input resource value and the resource limits set in the input rlimit structure. The resource value is set to RLIMIT_CPU. The resource limits are set to RLIM_INFINITY. For the callable service, see "setrlimit (BPX1SRL, BPX4SRL) — Set resource limits" on page 698. For the data structure, see "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033. AMODE 31 callers use "BPX1SRL (setrlimit) example" on page 1197.

```
MVC   RESOURCE,=A(RLIMIT_CPU)    Value of resource
XC    RLIM_CUR_HW,RLIM_CUR_HW    Current limit highword (Zero)
XC    RLIM_MAX_HW,RLIM_MAX_HW    Maximum limit highword (Zero)
MVC   RLIM_CUR,=A(RLIM_INFINITY) Current limit
```

```
MVC   RLIM_MAX,=A(RLIM_INFINITY) Maximum limit
SPACE ,
CALL  BPX4SRL,              Set resource limits              +
      (RESOURCE,            Input: resource                  +
      RLIMIT,               Structure, mapped by BPXYRLIM    +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
L     R15,RETVAL            Load return value
C     R15,=F'-1'            Test for -1 return
BE    PSEUDO                Branch on error
```

BPX4SRU (setreuid) example

The following code sets the real and/or effective user IDs to 1. For the callable service, see "setreuid (BPX1SRU, BPX4SRU) —Set the real and/or effective UIDs" on page 695. AMODE 31 callers use "BPX1SRU (setreuid) example" on page 1197.

```
  MVC   RUID,=XL4'00000001'   Value of new real user ID
MVC   RUID,..               User ID to be set from a getuid
  MVC   EUID,=XL4'00000001'   Value of new effective user ID
MVC   EUID,..               User ID to be set from a geteuid
  SPACE ,
  CALL  BPX4SRU,              Set user IDs                     +
        (RUID,                Input: Real User ID to be set    +
        EUID,                 Input: Eff. User ID to be set    +
        RETVAL,               Return value: 0 or -1            +
        RETCODE,              Return code                      +
        RSNCODE),             Reason code                      +
        MF=(E,PLIST)          ---------------------------------
```

BPX4SRX (srx_np) example

srx_np callable service sends or receives data on a socket using CSM buffers. The following example receives data into CSM buffers. The MSGXNAMEPTR is set up to point to a buffer to receive the source address of the data. The MSGXIOVX is an IVTBUFL structure, which describes an IOVX array in a CSM buffer. The IOVX array contains IVTBUFL structures, each of which describes a CSM buffer with data that was received. SOCKDESC is a socket descriptor that was returned from a previous call to either BPX4SOC or BPX4ACP. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGX — Map the message header" on page 999. For the callable service, see "srx_np (BPX1SRX, BPX4SRX) — Send or receive CSM buffers on a socket" on page 799. AMODE 31 callers use "BPX1SRX (srx_np) example" on page 1197.

```
XC    MSGX(MSGX#LEN),MSGX   Clear msgx storage
LA    R2,SOCKADDR
ST    R2,MSGXNAMEPTR        Store the address of sockaddr
LA    R2,SOCK#LEN+SOCK_SIN#LEN
ST    R2,MSGXNAMELEN        Length of sockaddr buffer
SPACE ,
CALL  BPX4SRX,              Receive data in CSM buffers      +
      (SOCKDESC,            Input: Socket Descriptor         +
      MSGX_RECV,            Input: Direction                 +
      L'MSGX,               Input: Msghdrx length            +
      MSGX,                 Input: Msghdrx                   +
      RETVAL,               Return value: -1 or bytes read   +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SSI (setsid) example

The following code creates a session and a process group (and is the leader of both). For the callable service, see “setsid (BPX1SSI, BPX4SSI) — Create a session and set the process group ID” on page 702. AMODE 31 callers use “BPX1SSI (setsid) example” on page 1198.

```
CALL  BPX4SSI,             Create session, set process grp ID+
      (RETVAL,             Return value: -1 or new session ID+
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4SSU (sigsuspend) example

The following code replaces the invoker's current mask to block signals 1 through 16 and suspend until a signal is delivered. For the callable service, see “sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered” on page 763. AMODE 31 callers use “BPX1SSU (sigsuspend) example” on page 1198.

```
MVC   WAITMASK(8),=XL8'FFFF000000000000'  Blocks 1 thru 16
SPACE ,
CALL  BPX4SSU,             Wait for a signal                 +
      (WAITMASK,           Input: Wait mask, XL8             +
      RETVAL,              Return value: -1 or not returned  +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4STA (stat) example

The following code obtains status about file **labrec/qual/current** . For the callable service, see “stat (BPX1STA, BPX4STA) — Get status information about a file by pathname” on page 805. For the data structure, see “BPXYSTAT — Map the response structure for stat” on page 1057. AMODE 31 callers use “BPX1STA (stat) example” on page 1198.

```
MVC   BUFFERA(19),=CL19'labrec/qual/current'
MVC   BUFLENA,=F'19'
SPACE ,
CALL  BPX4STA,             Get file status                   +
      (BUFLENA,            Input: Pathname length            +
      BUFFERA,             Input: Pathname                   +
      STATL,               Input: Length of buffer needed    +
      STAT,                Buffer, BPXYSTAT                  +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4STE (set_timer_event) example

The following code sets a timer event, which when it expires will post the ECB represented by THLITIMERECB. For the callable service, see “set_timer_event (BPX1STE, BPX4STE) — Set DIE-mode timer event” on page 707. AMODE 31 callers use “BPX1STE (set_timer_event) example” on page 1199.

```
CALL  BPX4STE,             Set timer event                  +
      (=A(2),              Input: Number of seconds         +
      =A(500000000),       Input: Number of nanoseconds     +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4STF (w_statvfs) example

The following code obtains information about file system TESTLIB.FILESYS1. For the callable service, see "w_statvfs (BPX1STF, BPX4STF) — Get the file system status" on page 926. For the data structure, see "BPXYSSTF — Map response structure for file system status" on page 1055. AMODE 31 callers use "BPX1STF (w_statvfs) example" on page 1199.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
SPACE ,
CALL  BPX4STF,             Get file system status           +
      (FSNAME,             Input: File system name (44 char) +
      SSTFL,               Input: Length of BPXYSSTF        +
      SSTF,                Buffer, BPXYSSTF                 +
      RETVAL,              Return value: -1 or length status +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4STL (set_thread_limits) example

The following code sets the MAX_THREAD and MAX_THREAD_TASKS limits for pthread_created threads in the invoker's process. For the callable service, see "set_thread_limits (BPX1STL, BPX4STL) — Change task or thread limits for pthread_created threads" on page 704. AMODE 31 callers use "BPX1STL (set_thread_limits) example" on page 1199.

```
CALL  BPX4STL,             Set_thread_limits                +
      (=A(STL_SET_BOTH),   Input: action           BPXYCONS +
      =A(50),              Input: new task limit            +
      =A(100),             Input: new thread limit          +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4STO (sendto) example

The following code issues a sendto for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see "sendto (BPX1STO, BPX4STO) — Send data on a socket" on page 652. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYMSGF — Map the message flags" on page 997. AMODE 31 callers use "BPX1STO (sendto) example" on page 1200.

```
MVC   BUFFERA(16),=CL16'Here is the data'
LA    R2,BUFFERA
STG   R2,IOV_BASE
MVI   IOV_LEN,16
SPACE ,
CALL  BPX4STO,             Send data to a socket            +
      (SOCKDESC,           Input: Socket Descriptor         +
```

```
      =A(L'BUFFERA),        Input: Length of the input buffer +
      BUFFERA,              Input: input buffer               +
      PRIMARYALET,          Input: Alet of the input buffer   +
      MSG_FLAGS,            Input: Flags                      +
      =A(L'SOCKADDR),       Input: Length of the socket addr  +
      SOCKADDR,             Input: The socket address         +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4STR (setitimer) example

The following code returns the time remaining an alarm, or ITIMER_REAL as set by setitimer. For the callable service, see “setitimer (BPX1STR, BPX4STR) — Set the value of the iterval timer” on page 680. For the data structure, see “BPXYITIM — Map getitimer, setitimer structure” on page 990. AMODE 31 callers use “BPX1STR (setitimer) example” on page 1200.

```
LA    R15,2               Initial value 2.5 seconds
ST    R15,ITIMISECONDS
L     R15,=A(500000)
ST    R15,ITIMIMICROSEC
L     R15,0               No reload value
ST    R15,ITIMRSECONDS
ST    R15,ITIMRMICROSEC
LA    R15,ITIM            Output mapping structure
STG   R15,ITIMA           ->structure
CALL  BPX4STR,            Get process data                  +
      (=A(ITIMER_REAL),   Input: Relative process token     +
      ITIMA,              In : ->Buffer, mapped by BPXYITIM +
      ITIMA,              Out: ->Buffer, mapped by BPXYITIM +
      RETVAL,             Return value: -1, 0               +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ----------------------------------
```

BPX4STV (statvfs) example

The following code obtains information about the file system containing the file identified by pathname. For the callable service, see “statvfs (BPX1STV, BPX4STV) — Get the file system status” on page 809. For the data structure, see “BPXYSSTF — Map response structure for file system status” on page 1055. AMODE 31 callers use “BPX1STV (statvfs) example” on page 1200.

```
MVC   BUFFERA(8),=CL8'/usr/inv'
MVC   BUFLENA,=F'8'
SPACE ,
CALL  BPX4STV,            Get file system status            +
      (BUFLENA,           Input: Pathname length            +
      BUFFERA,            Input: Pathname                   +
      SSTFL,              Input: Length of BPXYSSTF         +
      SSTF,               Buffer, BPXYSSTF                  +
      RETVAL,             Return value: -1 or length status +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ----------------------------------
```

BPX4STW (sigtimedwait) example

The following code will wait for signals 1-4 to arrive or 3 seconds, whichever occurs first. For the callable service, see "sigtimedwait (BPX1STW, BPX4STW) — Wait for a signal with a specified timeout" on page 766. AMODE 31 callers use "BPX1STW (sigtimedwait) example" on page 1201.

```
MVC   WAITMASK(8),=XL8'F000000000000000'  Signals 1-4
LA    R15,SIGINFO_T
STG   R15,SINFA
MVC   SECONDS,=F'3'         Wait three seconds
XC    NANOSECONDS,NANOSECONDS Zero nanoseconds
SPACE ,
CALL  BPX4STW,              Signal timed wait                +
      (WAITMASK,            Input: mask of signal to wait for +
      SINFA,                Input: address of siginfo_t area  +
      SIGINFO#LENGTH,       Input: length of siginfo_t area   +
      SECONDS,              Input: seconds to wait for sig    +
      NANOSECONDS,          Input: nanoseconds to wait for sig+
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SUI (setuid) example

The following code sets the real, effective, and saved user IDs to 1. For the callable service, see "setuid (BPX1SUI, BPX4SUI) — Set user IDs" on page 710. AMODE 31 callers use "BPX1SUI (setuid) example" on page 1201.

```
  MVC   USERID,=XL4'00000001' Value of new user ID
MVC   USERID,..             User ID to be set from a getuid
  SPACE ,
  CALL  BPX4SUI,              Set user ID                     +
        (USERID,              Input: User ID to be set        +
        RETVAL,               Return value: 0 or -1           +
        RETCODE,              Return code                     +
        RSNCODE),             Reason code                     +
        MF=(E,PLIST)          --------------------------------
```

BPX4SWT (sigwait) example

The following code waits for an asynchronous signal, **SIGALRM** bit 14 in the mask. For the callable service, see "sigwait (BPX1SWT, BPX4SWT) — Wait for a signal" on page 769. For the data structure, see "BPXYSIGH — Signal constants" on page 1039. AMODE 31 callers use "BPX1SWT (sigwait) example" on page 1201.

```
MVC   WAITMASK(8),=XL8'0004000000000000'
SPACE ,
CALL  BPX4SWT,              Wait for asynchronous signal      +
      (WAITMASK,            Input: Signal mask SIGALRM        +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
```

BPX4SYC (sysconf) example

The following code gets the maximum number of children allowed by the configuration variable. For the callable service, see “sysconf (BPX1SYC, BPX4SYC) — Determine system configuration options” on page 819. For the data structure, see “BPXYCONS — Constants used by services” on page 952. AMODE 31 callers use “BPX1SYC (sysconf) example” on page 1202.

```
CALL  BPX4SYC,              Get configuration variable       +
      (=A(SC_CHILD_MAX),    Input: Config variable BPXYCONS  +
      RETVAL,               Return value: -1 or variable     +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SYM (symlink) example

The following code creates a symbolic link **/sysaccts** for path name **/sys12/acctn**. For the callable service, see “symlink (BPX1SYM, BPX4SYM) — Create a symbolic link to a path name” on page 812. AMODE 31 callers use “BPX1SYM (symlink) example” on page 1202.

```
MVC   BUFFERA(12),=CL12'/sys12/acctn'
MVC   BUFLENA,=F'12'
MVC   BUFFERB(09),=CL09'/sysaccts'
MVC   BUFLENB,=F'09'
SPACE ,
CALL  BPX4SYM,              Create symbolic link to pathname  +
      (BUFLENA,             Input: Pathname length            +
      BUFFERA,              Input: Pathname                   +
      BUFLENB,              Input: Link name length           +
      BUFFERB,              Input: Link name                  +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4SYN (sync) example

The following code causes all information in memory that updates file systems to be scheduled for writing out to disk. For the callable service, see “sync (BPX1SYN, BPX4SYN) — Schedule file system updates” on page 818. AMODE 31 callers use “BPX1SYN (sync) example” on page 1202.

```
CALL  BPX4SYN,              Sync                             +
      (RETVAL,              Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4TAF (MVSThreadAffinity) example

The following code executes the assembler routine EXITRTN on another thread, identified by thread ID THID, and passes EXITPARM as input in R1. The requesting thread is blocked until EXITRTN runs. For the callable service, see “MVSThreadAffinity (BPX1TAF, BPX4TAF) — MVS thread affinity service” on page 427. AMODE 31 callers use “BPX1TAF (MVSThreadAffinity) example” on page 1203.

```
         MVC   EXITRTNA,=AD(EXITRTN)  ->Routine address
*        MVC   EXITPLA,=AD(EXITPARM)  ->Input parameter list
         SPACE ,
         CALL  BPX4TAF,                                          +
               (EXITRTNA,           Input: Routine address       +
               EXITPLA,             Input: Parm list address or 0+
               THID,                Input: Target pthread to run exit +
               RETVAL,              Return value: -1 or not return    +
               RETCODE,             Return code                  +
               RSNCODE),            Reason code                  +
               MF=(E,PLIST)         ---------------------------------
```

BPX4TAK (takesocket) example

The following code takes a socket that was given by the program identified by CID (clientid). SOCKDESC and CID information are passed by the program that did the givesocket (BPX4GIV). SOCKDESC is the giver's descriptor. When takesocket completes successfully, RETVAL will contain the taker's new socket descriptor. For the callable service, see "takesocket (BPX1TAK, BPX4TAK) — Acquire a socket from another program" on page 821. For the data structure, see "BPXYCID — Map the returning structure for getclientid()" on page 951. AMODE 31 callers use "BPX1TAK (takesocket) example" on page 1203.

```
CALL  BPX4TAK,              take a socket from another program+
      (CID,                 Input: Clientid of giver          +
      SOCKDESC,             Input: Giver's socket descriptor  +
      RETVAL,               Return value: -1 or new descriptor+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
L  R2,RETVAL
ST R2,SOCKDES2              Store the new socket descriptor
```

BPX4TDR (tcdrain) example

The following code waits until all output sent to the standard output file has been transmitted. For the callable service, see "tcdrain (BPX1TDR, BPX4TDR) — Wait until output has been transmitted" on page 824. AMODE 31 callers use "BPX1TDR (tcdrain) example" on page 1203.

```
CALL  BPX4TDR,              Wait for output transmittal       +
      (=A(STDOUT_FILENO),   Input: File descriptor            +
      RETVAL,               Return value: 0 or -1             +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4TFH (tcflush) example

The following code flushes all the data in the standard input file. the callable service, see "tcflush (BPX1TFH, BPX4TFH) — Flush input or output on a terminal" on page 829. For the data structure, see "BPXYTIOS — Map the termios structure" on page 1065. AMODE 31 callers use "BPX1TFH (tcflush) example" on page 1204.

```
CALL  BPX4TFH,              Line control flush                +
      (=A(STDIN_FILENO),    Input: File descriptor            +
      =A(TCIFLUSH),         Input: Queue selector BPXYTIOS    +
      RETVAL,               Return value: 0 or -1             +
```

```
        RETCODE,            Return code                     +
        RSNCODE),           Reason code                     +
        MF=(E,PLIST)        ---------------------------------
```

BPX4TFW (tcflow) example

The following code resumes data flow (TCION transmits a START character) on the standard input file. For the callable service, see "tcflow (BPX1TFW, BPX4TFW) — Suspend or resume data flow on a terminal" on page 826. For the data structure, see "BPXYTIOS — Map the termios structure" on page 1065. AMODE 31 callers use "BPX1TFW (tcflow) example" on page 1204.

```
CALL  BPX4TFW,            Suspend or resume data flow     +
      (=A(STDIN_FILENO),  Input: File descriptor          +
      =A(TCION),          Input: Action BPXYTIOS          +
      RETVAL,             Return value: 0 or -1           +
      RETCODE,            Return code                     +
      RSNCODE),           Reason code                     +
      MF=(E,PLIST)        ---------------------------------
```

BPX4TGA (tcgetattr) example

The following code retrieves control information about the standard input file. For the callable service, see "tcgetattr (BPX1TGA, BPX4TGA) — Get the attributes for a terminal" on page 831. For the data structure, see "BPXYTIOS — Map the termios structure" on page 1065. AMODE 31 callers use "BPX1TGA (tcgetattr) example" on page 1204.

```
CALL  BPX4TGA,            Get a terminal control structure  +
      (=A(STDIN_FILENO),  Input: File descriptor            +
      TIOS,               Termio structure, BPXYTIOS        +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4TGC (tcgetcp) example

The following code retrieves information about code page change notification (CPCN) capability and the BPXYTCCP structure. For the callable service, see "tcgetcp (BPX1TGC, BPX4TGC) — Get terminal code page names" on page 833. For the data structure, see "BPXYTCCP — Map the terminal control code page structure" on page 1058. AMODE 31 callers use "BPX1TGC (tcgetcp) example" on page 1204.

```
CALL  BPX4TGC,            Get code page names               +
      (=A(STDIN_FILENO),  Input: File descriptor            +
      =A(TCCP#LENGTH),    Input: Length of BPXYTCCP         +
      TCCP,               Output: Termcp structure BPXYTCCP +
      RETVAL,             Return value: 0 or -1             +
      RETCODE,            Return code                       +
      RSNCODE),           Reason code                       +
      MF=(E,PLIST)        ---------------------------------
```

BPX4TGP (tcgetpgrp) example

The following code gets the foreground process group ID associated with the controlling terminal. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see "tcgetpgrp (BPX1TGP, BPX4TGP) — Get the foreground process group ID" on page 836. AMODE 31 callers use "BPX1TGP (tcgetpgrp) example" on page 1205.

```
CALL  BPX4TGP,             Get the foreground process grp ID +
      (=A(STDIN_FILENO),   Input: File descriptor            +
      RETVAL,              Return value -1, fgrd proc grp ID +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4TGS (tcgetsid) example

The following code retrieves the process group ID of the session for which the terminal specified by file descriptor is the controlling terminal. For the callable service, see "tcgetsid (BPX1TGS, BPX4TGS) — Get a process group ID for the session leader for the controlling terminal" on page 838. AMODE 31 callers use "BPX1TGS (tcgetsid) example" on page 1205.

```
CALL  BPX4TGS,             Get session process group ID      +
      (=A(STDIN_FILENO),   Input: File descriptor            +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4TIM (times) example

The following code gathers selected times about the invoker's CPU utilization. For the callable service, see "times (BPX1TIM, BPX4TIM) — Get process and child process times" on page 856. For the data structure, see "BPXYTIMS — Map the response structure for times" on page 1064. AMODE 31 callers use "BPX1TIM (times) example" on page 1205.

```
CALL  BPX4TIM,             Process CPU times                 +
      (TIMS,               Input: Buffer           BPXYTIMS +
      RETVAL,              Return value: -1 or clock_t       +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4TLS (pthread_security_np) example

The following code creates a thread-level security environment for the calling thread using the identity specified by the caller. For the callable service, see "pthread_security_np, pthread_security_applid_np (BPX1TLS, BPX4TLS) — Create|delete thread-level security" on page 518. For the data structure, see "BPXYCONS — Constants used by services" on page 952. AMODE 31 callers use "BPX1TLS (pthread_security_np) example" on page 1205.

```
MVC   IDENT,=CL8'USERID05'
MVC   PASSWORD,=CL7'MYPSWRD'
SPACE ,
CALL  BPX4TLS,             pthread_security_np               +
      (=A(TLS_CREATE_THREAD_SEC#), Input: Func_code  BPXYCONS +
```

```
      TLS_IDENTITY_USERID#, Input: Identity_type     BPXYCONS +
      =A(8),                Input: Identity length           +
      IDENT,                Input: Identity                  +
      =A(7),                Input: Password length           +
      PASSWORD,             Input: Password                  +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4TRU (truncate) example

The following code truncates the file described by **/somedir/somefile.c** to a length of 512 bytes. For the callable service, see "truncate (BPX1TRU, BPX4TRU) — Change the size of a file" on page 859. AMODE 31 callers use "BPX1TRU (truncate) example" on page 1206.

```
      MVC   BUFFERA(20),=CL20'/somedir/somefile.c'
      MVC   BUFLENA,=F'20'
      MVC   NEWLEN(8),=FL8'512'
      SPACE ,
      CALL  BPX4TRU,              Truncate a file                  +
            (BUFLENA,             Input: Pathname length           +
            BUFFERA,              Input: Pathname                  +
            NEWLEN,               Input: Length to keep            +
            RETVAL,               Return value: 0 or -1            +
            RETCODE,              Return code                      +
            RSNCODE),             Reason code                      +
            MF=(E,PLIST)          ---------------------------------
```

BPX4TSA (tcsetattr) example

The following code turns off the HUPCL (hang up on last close) bit for the standard input file. For the callable service, see "tcsetattr (BPX1TSA, BPX4TSA) — Set the attributes for a terminal" on page 842. For the data structure, see "BPXYTIOS — Map the termios structure" on page 1065. AMODE 31 callers use "BPX1TSA (tcsetattr) example" on page 1206.

```
      NI    C_CFLAG+HUPCL_O,X'FF'-HUPCL   Turn off HUPCL
*     termios was retrived by a prior tcgetattr
      CALL  BPX4TSA,              Set terminal attributes          +
            (=A(STDIN_FILENO),    Input: File descriptor           +
            =A(TCSADRAIN),        Input: Action            BPXYTIOS +
            TIOS,                 Input: Terminos struct   BPXYTIOS +
            RETVAL,               Return value: 0 or -1            +
            RETCODE,              Return code                      +
            RSNCODE),             Reason code                      +
            MF=(E,PLIST)          ---------------------------------
```

BPX4TSB (tcsendbreak) example

The following code requests that a break be sent to the standard input file. For the callable service, see "tcsendbreak (BPX1TSB, BPX4TSB) — Send a break condition to a terminal" on page 840. AMODE 31 callers use "BPX1TSB (tcsendbreak) example" on page 1206.

```
      CALL  BPX4TSB,              Send break condition to terminal  +
            (=A(STDIN_FILENO),    Input: File descriptor           +
            =A(0),                Duration, not used in z/OS UNIX   +
            RETVAL,               Return value: 0 or -1            +
```

```
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4TSC (tcsetcp) example

The following code sets code page names and Code Page Change Notification (CPCN) capability. For the callable service, see "tcsetcp (BPX1TSC, BPX4TSC) — Set terminal code page names" on page 845. For the data structure, see "BPXYTCCP — Map the terminal control code page structure" on page 1058. AMODE 31 callers use "BPX1TSC (tcsetcp) example" on page 1207.

```
XC    TCCP(TCCP#LENGTH),TCCP  Clear area
OI    TCCPFLAGB4,TCCPFASTP  Set local translation
MVC   TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC   TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
SPACE ,
CALL  BPX4TSC,             Set code page names              +
      (=A(STDIN_FILENO),   Input: File descriptor           +
      =A(TCCP#LENGTH),     Input: Length of BPXYTCCP        +
      TCCP,                Termcp structure, BPXYTCCP       +
      RETVAL,              Return value: 0 or -1            +
      RETCODE,             Return code                      +
      RSNCODE),            Reason code                      +
      MF=(E,PLIST)         ---------------------------------
```

BPX4TSP (tcsetpgrp) example

The following code sets the controlling terminal's foreground process group to a new value. For this example to work, STDIN must be associated with the controlling terminal. For the callable service, see "tcsetpgrp (BPX1TSP, BPX4TSP) — Set the foreground process group ID" on page 849. AMODE 31 callers use "BPX1TSP (tcsetpgrp) example" on page 1207.

```
MVC   PROCID,..            Process group ID set by setpgrp
  SPACE ,
  CALL  BPX4TSP,             Set foreground process group ID   +
        (=A(STDIN_FILENO),   Input: File descriptor            +
        PROCID,              Input: Foreground process group ID+
        RETVAL,              Return value: 0 or -1             +
        RETCODE,             Return code                       +
        RSNCODE),            Reason code                       +
        MF=(E,PLIST)         ---------------------------------
```

BPX4TST (tcsettables) example

The following code sets code page names, conversion tables and Code Page Change Notification (CPCN) capability. For the callable service, see "tcsettables (BPX1TST, BPX4TST) — Set terminal code page names and conversion tables" on page 852. For the data structure, see "BPXYTCCP — Map the terminal control code page structure" on page 1058. AMODE 31 callers use "BPX1TST (tcsettables) example" on page 1207.

```
XC    TCCP(TCCP#LENGTH),TCCP  Clear area
OI    TCCPFLAGB4,TCCPFASTP  Set local translation
MVC   TCCPSRCNAME(8),=CL8'IBM-1047'  Set source code page name
MVC   TCCPTRGNAME(9),=CL9'ISO8859-1' Set target code page name
MVC   TBLSOURCE,..         Initialize source conversion table
MVC   TBLTARGET,..         Initialize target conversion table
SPACE ,
CALL  BPX4TST,             Set code page names and tables    +
```

```
      (=A(STDIN_FILENO),    Input: File descriptor           +
      =A(TCCP#LENGTH),      Input: Length of BPXYTCCP        +
      TCCP,                 Termcp structure, BPXYTCCP       +
      TBLSOURCE,            Source conversion table          +
      TBLTARGET,            Target conversion table          +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4TYN (ttyname) example

The following code retrieves the pathname for the standard error output file. For the callable service, see “ttyname (BPX1TYN, BPX4TYN) (POSIX version) — Get the name of a terminal” on page 862. AMODE 31 callers use “BPX2TYN (ttyname) example” on page 1208.

```
MVC   BUFLENA,=A(1023)      Maximum pathname
CALL  BPX4TYN,              Determine terminal name          +
      (=A(STDERR_FILENO),   Input: File descriptor           +
      BUFLENA,              Length of buffer for pathname    +
      BUFFERA,              Buffer for pathname of terminal  +
      RETVAL,               Return value: 0, -1              +
      RETCODE,              Return code: describes why VAL=-1 +
      RSNCODE),             Reason code: qualifier on RETCODE +
      MF=(E,PLIST)          ---------------------------------
```

BPX4UMK (umask) example

The following code changes the process's file mode creation mask (to user read, group execute, other execute). For the callable service, see “umask (BPX1UMK, BPX4UMK) — Set the file mode creation mask” on page 866. For the data structure, see “BPXYMODE — Map the mode constants of the file services” on page 996. AMODE 31 callers use “BPX1UMK (umask) example” on page 1208.

```
XC    S_MODE,S_MODE
MVI   S_MODE3,S_IXUSR+S_IXGRP+S_IXOTH  Search permission
SPACE
CALL  BPX4UMK,              Set file creation mask           +
      (S_MODE,              Input: Mode             BPXYMODE +
      RETVAL),              Return value: previous mode mask  +
      MF=(E,PLIST)          ---------------------------------
```

BPX4UMT (umount) example

The following code removes virtual file system TESTLIB.FILESYS1 from the file tree. For the callable service, see “umount (BPX1UMT, BPX4UMT) — Remove a virtual file system” on page 867. For the data structure, see “BPXYMTM — Map the modes for mount and unmount” on page 1000. AMODE 31 callers use “BPX1UMT (umount) example” on page 1209.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC    MTM(MTM#LENGTH),MTM
MVI   MTM1,MTMUMOUNT        Unmount request
SPACE ,
CALL  BPX4UMT,              Remove a virtual file system     +
      (FSNAME,              Input: File system name (44 char) +
      MTM,                  Input: Flags, BPXYMTM            +
      RETVAL,               Return value: 0 or -1            +
```

```
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4UNA (uname) example

The following code obtains information about the system on which the invoker is running. For the callable service, see "uname (BPX1UNA, BPX4UNA) — Obtain the name of the current operating system" on page 870. For the data structure, see "BPXYUTSN — Map the response structure for uname" on page 1068. AMODE 31 callers use "BPX1UNA (uname) example" on page 1209.

```
LA    R15,UTSN
STG   R15,UTSNA
SPACE ,
CALL  BPX4UNA,             Identify system                   +
      (UTSNL,              Input: Length of required buffer  +
      UTSNA,               Output: ->UTSN           BPXYUTSN +
      RETVAL,              Return value: -1 or >-1           +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4UNL (unlink) example

The following code removes path name **usr/dataproc/next.t** from the system. For the callable service, see "unlink (BPX1UNL, BPX4UNL) — Remove a directory entry" on page 872. AMODE 31 callers use "BPX1UNL (unlink) example" on page 1209.

```
MVC   BUFFERA(19),=CL19'usr/dataproc/next.t'
MVC   BUFLENA,=F'19'
SPACE ,
CALL  BPX4UNL,             Remove a directory entry          +
      (BUFLENA,            Input: Pathname length            +
      BUFFERA,             Input: Pathname                   +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4UPT (unlockpt) example

The following code unlocks the slave pseudoterminal device associated with the master to which the file descriptor refers. For the callable service, see "unlockpt (BPX1UPT, BPX4UPT) — Unlock a pseudoterminal master/slave pair" on page 875. AMODE 31 callers use "BPX1UPT (unlockpt) example" on page 1210.

```
CALL  BPX4UPT,             Unlocks slave pty from master     +
      (MASTER_FD,          Input: File descriptor            +
      RETVAL,              Return value: 0 or -1             +
      RETCODE,             Return code                       +
      RSNCODE),            Reason code                       +
      MF=(E,PLIST)         ----------------------------------
```

BPX4UQS (unquiesce) example

The following code unquiesces TESTLIB.FILESYS1, making its files available for use again. For the callable service, see “unquiesce (BPX1UQS, BPX4UQS) — Unquiesce a file system” on page 877. For the data structure, see “BPXYMTM — Map the modes for mount and unmount” on page 1000. AMODE 31 callers use “BPX1UQS (unquiesce) example” on page 1210.

```
MVC   FSNAME(44),=CL44'TESTLIB.FILESYS1'
XC    MTM(MTM#LENGTH),MTM   Zero MTM = don't force unquiesce
SPACE ,
CALL  BPX4UQS,              Unquiesce a file system          +
      (FSNAME,              Input: File system name (44 char) +
      MTM,                  Input: Flags, BPXYMTM            +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4UTI (utime) example

The following code changes the access and modification times of **/usr/private/workfile.t** to the current time. For the callable service, see “utime (BPX1UTI, BPX4UTI) — Set file access and modification times” on page 879. AMODE 31 callers use “BPX1UTI (utime) example” on page 1210.

```
MVC   BUFFERA(23),=CL23'/usr/private/workfile.t'
MVC   BUFLENA,=F'23'
MVC   NEWTIMES,=X'FFFFFFFFFFFFFFFF'  Current time
SPACE ,
CALL  BPX4UTI,              Set file access and modify times  +
      (BUFLENA,             Input: Pathname length           +
      BUFFERA,              Input: Pathname                  +
      NEWTIMES,             Input: Access/Modification time   +
      RETVAL,               Return value: 0 or -1            +
      RETCODE,              Return code                      +
      RSNCODE),             Reason code                      +
      MF=(E,PLIST)          ---------------------------------
```

BPX4WAT (wait) example

The following code waits for any of its children to end or stop. For the callable service, see “wait (BPX1WAT, BPX4WAT) — Wait for a child process to end” on page 882. For the data structure, see “BPXYWAST — Map the wait status word” on page 1069 and “BPXYCONS — Constants used by services” on page 952. AMODE 31 callers use “BPX1WAT (wait) example” on page 1210.

```
LA    R15,WAST              Resolve address of STATUS
STG   R15,WASTA             Save address of STATUS
MVC   PROCID,=F'-1'         Wait for any child
SPACE ,
CALL  BPX4WAT,              Wait for a child process to end   +
      (PROCID,              Input: PID being waited on        +
      =A(WNOHANG),          Input: options       BPXYCONS     +
      WASTA,                ->Exit status field, BPXTWAST     +
      RETVAL,               Return value: -1, 0, child PID    +
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ---------------------------------
```

BPX4WLM (__WLM) example

The following code connects to WLM as a work manager for the WEB subsystem type and WEB1 subsystem name. For the callable service, see "__wlm (BPX1WLM, BPX4WLM) — WLM interface service" on page 916. AMODE 31 callers use "BPX1WLM (__WLM) example" on page 1211.

```
LA    R8,BUFFERA            Storage for _WWC
USING _WWC,R8               WLM_CONNECT_WORKMGR DSECT
STG    R8,INARGLISTPTR       ->_WWC list of parameters
MVC   SUBSYSTYPE,=CL4'WEB '  WEB Subsystem Type
MVC   SUBSYSNAME,=CL8'WEB1    '  WEB1 Subsystem Name
LA    R15,SUBSYSTYPE
STG   R15,_WWC_SUB_SYS      Pointer to Subsystem Type
LA    R15,SUBSYSNAME
STG   R15,_WWC_SUB_SYS_NM   Pointer to Subsystem Name
SPACE ,
CALL  BPX4WLM,              work_load_manager system call     +
      (=A(WLM_CONNECT_WORKMGR), Input: Fcn Codes in BPXYWLM   +
      INARGLISTPTR,         Input: ->list of parameters       +
      RETVAL,               Return value: Varies with fcn code+
      RETCODE,              Return code                       +
      RSNCODE),             Reason code                       +
      MF=(E,PLIST)          ----------------------------------
DROP  R8
```

BPX4WRT (write) example

The following code writes 80 bytes from the specified buffer to the file specified (FILEDESC). For the callable service, see "write (BPX1WRT, BPX4WRT) — Write to a file or a socket" on page 928. AMODE 31 callers use "BPX1WRT (write) example" on page 1211.

```
*       MVC   FILEDESC,             File descriptor from open
        MVC   BUFLENA,=F'80'
        LA    R15,BUFFERA
        STG   R15,BUFA
        SPACE ,
        CALL  BPX4WRT,              Write to a file                 +
              (FILEDESC,            Input: File descriptor          +
              BUFA,                 Input: ->Buffer                 +
              PRIMARYALET,          Input: Buffer ALET              +
              BUFLENA,              Input: Number of bytes to write +
              RETVAL,               Return value: -1 or bytes written +
              RETCODE,              Return code                     +
              RSNCODE),             Reason code                     +
              MF=(E,PLIST)          --------------------------------
```

BPX4WRV (writev) example

The following code issues a writev for a socket. SOCKDESC was returned from a previous call to either BPX4SOC or BPX4ACP. For the callable service, see "writev (BPX1WRV, BPX4WRV) — Write data from a set of buffers" on page 933. For the data structures, see "BPXYSOCK — Map SOCKADDR structure and constants" on page 1043 and "BPXYIOV — Map the I/O vector structure" on page 986. AMODE 31 callers use "BPX1WRV (writev) example" on page 1212.

```
MVC   BUFFERA(16),=CL16'Here is the data'
LA    R2,BUFFERA
STG   R2,IOV_BASE
MVI   IOV_LEN,16
```

```
*
         CALL  BPX4WRV,             Write from a vector of buffers    +
               (SOCKDESC,           Input: Socket Descriptor          +
               =A(1),               Input: Single element in iov      +
               IOV,                 Input: Iov containing info        +
               PRIMARYALET,         Input: Alet where iov resides     +
               PRIMARYALET,         Input: Alet of buffers for data   +
               RETVAL,              Return value: 0 or -1             +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
```

BPX4WTE (wait extension) example

The following code uses the #WAIT3 function to wait for any of its children to end or stop. For the callable service, see "wait-extension (BPX1WTE, BPX4WTE) — Obtain status information for children" on page 885. For the data structures, see "BPXYWAST — Map the wait status word" on page 1069 and "BPXYRLIM — Map the rlimit, rusage, and timeval structures" on page 1033 and "BPXYCONS — Constants used by services" on page 952. AMODE 31 callers use "BPX1WTE (wait extension) example" on page 1212.

```
         LA    R15,WAST             Resolve address of WAST
         STG   R15,WASTA            Save address of WAST
         LA    R15,RUSAGE           Resolve address of RUSAGE
         STG   R15,RUSAGEA          Save address of RUSAGE
         SPACE ,
         CALL  BPX4WTE,             Wait for a child process to end   +
               (=A(#WAIT3),         Input: function      BPXYCONS     +
               0,                   Input: id type                    +
               0,                   Input: id                         +
               WASTA,               ->Exit status field, BPXTWAST     +
               =A(WNOHANG),         Input: options       BPXYCONS     +
               RUSAGEA,             ->Rusage structure,  BPXYRLIM     +
               RETVAL,              Return value: -1, 0, child PID    +
               RETCODE,             Return code                       +
               RSNCODE),            Reason code                       +
               MF=(E,PLIST)         ----------------------------------
```

Reentrant return linkage

```
         XGR   R15,R15              Zero return code
         L     R0,@SIZEDAT          Size this program's getmain area
         LGR   R1,R13               R1 -> this program's getmain area
         LG    R13,@BACK            R2 -> caller's save area
         DROP  R13
         FREEMAIN RU,LV=(0),A=(1)
         LG    R14,8(,R13)          Restore caller's R14
         LMG   R0,R12,16(R13)       Restore caller's R0-R12
         BR    R14                  Branch back to caller


         SPACE ,  * * * * * * * * * *.* Program constants * * * * * *
@SIZEDAT DC    A(@ENDSTOR-@STORE)    Size of this getmain storage
MNTEL    DC    A(MNTE#LENGTH+MNTEH#LENGTH)
*                                   Length of MNTEH and 1 MNTE area
PGPSL    DC    A(PGPS#LENGTH)        Length of PGPS structure
RMONL    DC    A(RMON#LENGTH)        Length of RMON structure
SSTFL    DC    A(SSTF#LENGTH)        Length of SSTF structure
STATL    DC    A(STAT#LENGTH)        Length of STAT structure
UTSNL    DC    A(UTSN#LENGTH)        Length of UTSN structure
         SPACE ,
PRIMARYALET  DC   A(0)               Primary ALET
```

```
* * * * * * * * * * * * * * * * * * * * *.* Program getmain variables * * *
             DS    0D
ACPSOCK      DS    F               Accepted socket descriptor
ADDR_INFO_PTR DS   F               ->Addr_Info Structure
APPLENV      DS    CL8             Application Environment
APPLDATALEN  DS    F               Application Data Length
APPLDATAPTR  DS    A               -> Application Data
ARGCNT       DS    F               Argument count
ARGLLST      DS    3AD             Argument lengths list
ARGSLST      DS    3AD             Arguments list
ATTRIBUTES   DS    F               Attributes
ATTADDR      DS    AD              Shared memory attached address
BUFA         DS    FD              ->buffer
BUFB         DS    FD              ->buffer
BUFFR        DS    F
BUFCNTB      DS    F               Count associated with BUFFERB
BUFFERA      DS    CL1024          Utility buffer A, length 1024
BUFFERB      DS    CL1024          Utility buffer B, length 1024
BUFLENA      DS    F               Number of bytes used in buffer A
BUFLENB      DS    F               Number of bytes used in buffer B
BUFW         DS    F               Number of words used in BUF
BYTERECD     DS    F               Bytes Received
CANONICAL_LENGTH DS F              Canonical name length
CELLUUID     DS    CL36            Cell UUID (string form)
CLASSIFYLEN  DS    F               Classify Area Length
CLASSIFYAREAPTR DS  A              -> Classify Area
CLSLEN       DS    F               Class name length
CLS          DS    CL8             Class name
COMMAND      DS    F               User defined command
CONMSGTYPE   DS    F               Console msg type (modify or stop)
DIRECTDES    DS    F               Directory descriptor
ECB01        DS    F               Event Control Block # 1
ECB02        DS    F               Event Control Block # 2
EGID         DS    F               User ID
ENT          DS    CL40            Entity name
ENTLEN       DS    F               Entity name length
ENTRYPT      DS    FD              Entry Point
ENVCNT       DS    F               Number of environment variables
ENVLENS      DS    FD              Length of environment variables
ENVPARMS     DS    FD              Environment variables
EPADDR       DS    AD              Entry point address
EUID         DS    F               User ID
EVENTLIST    DS    A               Event list for thread posting
EXITRTNA     DS    AD              Exit routine address
EXITPLA      DS    AD              Exit Parm list address
FDCNT        DS    F               File descriptor count
FDLST        DS    F               File descriptor list
FDLISTPTR    DS    A               -> File Descriptor List
FILEDESC     DS    F               File descriptor
FILEDES2     DS    F               File descriptor
FLAGS        DS    F               Utility Flags word
FSNAME       DS    CL44            File system name
FSTYPE       DS    CL8             File system type
GRNAMELN     DS    F               Group name length
GROUP        DS    F               Group
GROUPCNT     DS    F               Group count
GROUPID      DS    F               Group ID (PID of group leader)
GRPGMNAME    DS    CL8             Group program name
HINTS_PTR    DS    FD              ->Addr_Info Structure
HOST_BUFFER  DS    CL255           Host buffer (up to 255 Characters)
HOST_BUFFER_LENGTH DS F            Host buffer length
IDENT        DS    CL8             Identity of new thread security
INARG        DS    F               Input argument
INARGLIST    DS    A               Input arglist
INARGLISTPTR DS    AD              Input arglist ptr
INCR         DS    F               Increment value for nice
```

```
INTMASK       DS    XL8              Signal mask
INITADDR  DS   FD                   Address __map_init parm list
INITPARM  DS   0C                   __map_init parm list
          ORG  *+_MMG_INIT_PARM_LEN
INITRTNA      DS    AD               ->Initialization routine
INTRSTATE     DS    A                Interrupt state
INTRTYPE      DS    A                Interrupt type
ITIMA         DS    AD               ->BPXYITIM structure
KEY           DS    F                Interprocess Communication KEY
LIBPTHLN      DS    F                Library Path Length (BPX4LOD)
LIBPATH       DS    CL100            Library Path (BPX4LOD)
LOCKADDR      DS    AD               ->Lockword
LOCKTOKENADDR DS    AD               ->LockToken
LOCKATTRADDR  DS    AD               ->LockAttr
LOCKWORD      DS    F                Lockword (BPX4SLK)
LSOCKADR      DS    F                Local socket structure
LTOKEN        DS    CL8              Local token
MAP_ADDRESS   DS    AD               ->mapped area
MAP_LENGTH    DS    FD               length of mapped area
MASTER_FD     DS    F                Master file descriptor
MSG_ID        DS    F                IPC Message Queue ID
MSGATTRLEN    DS    F                Length of BPX4CCA
MSGATTR       DS    CL100            Storage for BPX4CCA
MODSTRINGPTR  DS    F                Address of user msg buffer
MODIFYSTGLEN  DS    F                Length of user msg buffer
NANOSECONDS   DS    F                Count of nanoseconds
NCATCHER      DS    A                New catcher
NEWFLAGS      DS    F                New flags
NEWHANDL      DS    FD               New Handler
NEWLEN        DS    XL8              Length file
NEWMASK       DS    XL8              New mask for signals
NEWMASKA      DS    A                ->New mask
NEWPASS       DS    CL8              Password
NEWPASSLEN    DS    F                Password length
NEWTIMES      DS    DL2              New access/modification time
NODE_NAME     DS    CL255            Node Name (up to 255 Characters)
NODE_NAME_LENGTH DS F                Node Name Length
NUMB_SEMS     DS    F                IPC Number of semaphores in set
NUMB_SEM_OPS  DS    F                IPC Number of semaphore ops
OCATCHER      DS    A                Old catcher
OFFSET        DS    CL8              File offset
OLDHANDL      DS    FD               Old handler
OLDFLAGS      DS    F                Old flags
OLDMASK       DS    CL8              Old signal mask
OLDMASKA      DS    A                ->Old mask
OLDPASS       DS    CL8              Password
OLDPASSLEN    DS    F                Password length
OPTIONS       DS    F                Options
PARALLELEU    DS    F                Parallel Eu
PASSWORD      DS    CL8              Password
PGMNAME       DS    CL8              Program name
PGMNAMEL      DS    F                Length PGMNAME
PLIST         DS    13A              Max number of parms
PRINUUID      DS    CL36             Principal UUID (string form)
PRIORITY      DS    F                Priority value
PROCID        DS    F                Process ID
PROCTOK       DS    F                Relative process number
PT_NEWA       DS    AD               Address of PT_NEW
PT_OLD        DS    CL66             Pthread tag - old
PT_OLDA       DS    AD               Address of PT_OLD
PT_OLDL       DS    F                Length of tag in PT_NEW
READFD        DS    F                File descriptor - input file
REFPT         DS    F                File reference point
RESOURCE      DS    F                Resource
RESULTS_PTR   DS    FD               ->Addr_Info Structure
RETCODE       DS    F                Return code (ERRNO)
RETURNEDADDRESS DS  AD               Returned address in doubleword
```

```
RETVAL        DS    F               Return value (0, -1 or other)
RETVAL64      DS    FD              64-bit return value
RGID          DS    F               User ID
RSOCKADR      DS    F               Remote socket structure
RUID          DS    F               User ID
RUSAGEA       DS    AD              ->Rusage
RSNCODE       DS    F               Reason code (ERRNOJR)
SECONDS       DS    F               Time in seconds
SEGADDR       DS    AD              IPC Shared Memory segment Addr
SELLIST       DS    F               List to use for select calls
SEM_ID        DS    F               IPC Semaphore set ID
SEM_NUMBER    DS    F               IPC Semaphore number
SERVICE_BUFFER DS   CL32            Service Buffer ( to 32 Characters)
SERVICE_BUFFER_LENGTH DS F          Service buffer length
SERVICE_NAME  DS    CL32            Service Name (up to 32 Characters)
SERVICE_NAME_LENGTH DS F            Service Name Length
SHM_ID        DS    F               IPC Shared Memory segment ID
SIGNAL        DS    A               Signal
SIGNALREG     DS    AD              Signal registration, user data
SIGNALOPTIONS DS    A               Signal options
SIGPID        DS    F               Signal processs id for BPX4PAF
SIGRET        DS    CL8             Signal return mask
SIRTNA        DS    AD              Signal interrupt routine
SMF_TYPE      DS    F               SMF record type
SMF_SUBTYPE   DS    F               SMF record subtype
SOCKADDR_LENGTH DS  F               Lenght of SockAddr
SOCKETS       DS    0XL8            Socket vector for socket call
SOCKDESC      DS    F               Socket descriptor
SOCKDES2      DS    F               Second Socket descriptor
SRVCADDR      DS    FD              Address __map_service parm list
SRVCPARM      DS    0C              __map_service parm list
              ORG   *+3*_MMG_SERVICE_PARM_LEN Room for three entries
STATFLD       DS    AD              Status field
STATUS        DS    F               Status
STATUSA       DS    A               ->STATUS
SUBSYSTYPE    DS    CL4             Subsystem Type
SUBSYSNAME    DS    CL8             Subsystem Name
TARPID        DS    F               Target processs id for BPX4PAF
              ORG   BUFFERB         remap utility buffer B
TBLSOURCE     DS    XL256           Source conversion table
TBLTARGET     DS    XL256           Target conversion table
              ORG
TERMMASK      DS    XL8             Signal termination mask
THID          DS    XL8             Thread ID
TOKEN         DS    F               Relative IPC member or Misc Token
TRXCLASS      DS    CL8             Transaction Class
USERID        DS    F               User ID
USERDATA      DS    FD              User Data
USERNAME      DS    CL8             User name
USERNLEN      DS    F               Length USERNAME
HOST_NAME     DS    CL8             HOST name
HOST_NAMELEN  DS    F               Length HOST_NAME
HOST_ADDR     DS    CL8             HOST IP address
HOST_ADDRLEN  DS    F               Length HOST_ADDR
HOSTENT_PTR   DS    FD              Length HOST_ADDR
USERWORD      DS    FD              User data
WAITMASK      DS    F               Mast for signal waits
WHO           DS    F               Who for rusage
WRITEFD       DS    F               File descriptor - output file
LFUIOPTR      DS    FD              Pointer to FUIO structure
              SPACE ,
@ENDSTOR      EQU   *               End of getmain storage
              IVTBUFL
         SPACE 3  * * * * * * * * * *.* Register equates  * * * * * * *
              SPACE ,
R0            EQU   0
R1            EQU   1               Parameter list pointer
```

```
R2            EQU   2
R3            EQU   3
R4            EQU   4
R5            EQU   5
R6            EQU   6
R7            EQU   7
R8            EQU   8
R9            EQU   9
R10           EQU   10              Second getmain storage register
R11           EQU   11              Second program base register
R12           EQU   12              Program base register
R13           EQU   13              Savearea and getmain storage base
R14           EQU   14              Return address
R15           EQU   15              Branch location
         SPACE 3  * * * * * * * * * *.* External * * * * * * * * * * *
              SPACE ,
              EXTRN EXITRTN
              EXTRN SIRTN
                END
```

Appendix F. Examples of nonreentrant entry linkage

Example of nonreentrant entry linkage—AMODE 31

This example shows the function for the __getthent service in a nonreentrant program. For a reentrant example of __getthent, see "BPX1GTH (__getthent) example" on page 1152. For an example of reentrant entry and return linkage, see Appendix D, "Callable services examples—AMODE 31," on page 1123 and "Reentrant return linkage" on page 1212.

```
BPXB1SM5 CSECT ,                    Nonreentrant linkage
BPXB1SM5 AMODE 31
BPXB1SM5 RMODE ANY
         USING *,R15                Program addressability
@BEGIN0  B     @BEGIN1              Branch around program header
         DC    C'BPXB1SM5 - nonreentrant __getthent invoker'
         DS    0H
@BEGIN1  STM   R14,12,12(R13)       Save callers registers
         ST    R13,@BACK            Save ->Callers save area
         LA    R13,@SAVE00          Program addressability
         DROP  R15
         USING @SAVE00,R13          Program addressability
         B     @BEGIN2
@SAVE00  DS    0D                   Standard save area - 72 Bytes
         DS    A
@BACK    DS    A                    Backwards save area pointer
@FORWARD DS    A                    Forwards save area pointer
         DS    15A                  Regs 14,15,0-12
RETURN   XR    R15,R15              Zero return code
RETURNRC L     R13,@BACK            Restore callers r13
         L     R14,12(,R13)         Restore callers r14
         LM    R0,R12,20(R13)       Restore callers r0-r12
         BSM   0,R14                Branch back to caller
R0       EQU   0
R1       EQU   1                    Parameter list pointer
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13                   Program and save area base
R14      EQU   14                   Return address
R15      EQU   15                   Branch location
@BEGIN2  EQU   *  *  *  *  *  *  *  End of the entry linkage code
         EJECT ,
         LA    R5,BUFFERA           R5-> Input buffer
         ST    R5,PGTHAB            -> input buffer
         USING PGTHA,R5             R5 base for PGTHA
         XC    PGTHA,PGTHA          Null input area
         MVI   PGTHAFLAG1,PGTHAPROCESS+PGTHATHREAD
         MVI   PGTHAPID,PGTH#FIRST  First thread
         LA    R15,BUFFERB          Pgthb, Output buffer
         ST    R15,PGTHBB           Output Buffer
         SPACE ,  *  *  *  *  *  *
         LA    R0,=CL8'BPX1GTH '    LOAD -> entry point name
         XR    R1,R1                No JOBLIB or LINKLIB DCB
         SVC   8                    Issue LOAD SVC
```

Example of nonreentrant entry linkage—AMODE 31

```
         ST    R0,GETENTRY           Store BPX1GTH entry point
GETTH    L     R15,GETENTRY          Address of BPX1GPS load module
         CALL  (15),                 Get process data                  +
               (PGTHAL,              Length of buffer                  +
               PGTHAB,               Buffer, mapped by BPXPGPHA        +
               PGTHBL,               Length of output buffer           +
               PGTHBB,               Buffer, mapped by BPXPGTHC        +
               RETVAL,               Return value (next, eof or error) +
               RETCODE,              Return code                       +
               RSNCODE),             Reason code                       +
               VL                    ---------------------------------
         SPACE ,  *  *  *  *  *  *
         L     R15,RETVAL            Load return value
         C     R15,=F'-1'            Test for -1 return
         BE    RETURNRC              -1 is error
         SPACE ,  *  *  *  *  *  *   Initialize WTO area & message
         MVI   XPID,C' '             Blank out variable portion of msg  ge
         MVC   XPID+1(WTO#BLANK-1),XPID
         SPACE , *  *  *  *  *  *    Process ID to printable hex
         LA    R6,BUFFERB            R6-> Output buffer
         ST    R6,PGTHBB             -> output buffer
         USING PGTHB,R6              R6 base for PGTHB
         L     R8,PGTHBPID           R8 = process ID
         LA    R9,XPID               To be placed at message start
         LA    R15,8                 8 nibbles to convert (4 bytes)
         LA    R10,9                 For 0-9 / A-F compare
NIBBLE   LR    R11,R8                Target bits in 0-3    XYYYYYYZ
         SRL   R11,28                Bits 0-3 to 28-31     0000000X
         SLL   R8,4                  Drop bits 0-3 off end YYYYYYZ0
         CLR   R11,R10               Are 4 bits 0-9 or A-F
         BC    B'0010',AF            Branch if A-F
         LA    R11,57(,R11)          Add for 0-9 (57+183=240 or F0)
AF       LA    R11,183(,R11)         Add for 0-F (183+10=193 or C1)
         STC   R11,0(,R9)            Store to results location
         LA    R9,1(,R9)             Increment R9 to next location
         BCT   R15,NIBBLE            Decrement half byte counter, loop
         SPACE ,  *  *  *  *  *  *   Test status bits
* Go after the state of the process
         LA    R7,PGTHB
         SLR   R9,R9
         ICM   R9,7,PGTHBOFFC
         AR    R7,R9
         USING PGTHC,R7
         LA    R8,PGTHB
         SLR   R9,R9
         ICM   R9,7,PGTHBOFFJ
         AR    R8,R9
         USING PGTHJ,R8
         MVI   THREAD,C'1'           Assume single
         TM    PGTHCFLAG1,PGTHCMULPROCESS   if multiprocess
         BZ    NOTMULT
         MVI   THREAD,C'M'
NOTMULT  MVC   STATE,PGTHJSTATUS2    Z, W, X, S, C, F, K, R ...
         TM    PGTHCFLAG1,PGTHCSWAP  if swapped out
         BZ    NOTSWAP
         MVC   SWAPA,=CL4'SWAP'
NOTSWAP  TM    PGTHCFLAG1,PGTHCSTOPPED       if stopped
         BZ    NOTSTOP
         MVC   STOPA,=CL4'STOP'
NOTSTOP  TM    PGTHCFLAG1,PGTHCTRACE         if ptrace
         BZ    NOTTRAC
         MVC   TRACA,=CL4'TRAC'
NOTTRAC  EQU   *
         SPACE ,  *  *  *  *  *  *   Display message to operator
         LA    R2,WTOAREA            R2->WTO message area
         WTO   TEXT=(R2)             Write to Operator
         SPACE ,  *  *  *  *  *  *   Loop back
```

```
         MVC   PGTHACONTINUE,PGTHBCONTINUE  get next thread
         B     GETTH
WTOAREA  DS    0F                    WTO message
         DC    AL2(WTO#LENGTH)       Length of area
         DC    CL4'PID='             Process ID =
XPID     DS    CL8                   Hex of process ID
         DS    CL1
THREAD   DS    CL1                   1, M or H
         DS    CL1
STATE    DS    CL1                   Z, W, X, C, F, K, R ...
         DS    CL1
SWAPA    DS    CL4                   SWAP or blank
         DS    CL1
STOPA    DS    CL4                   STOP or blank
         DS    CL1
TRACA    DS    CL4                   TRAC or blank
WTO#BLANK EQU  *-XPID                Length to blank
         DC    C'.'
WTO#LENGTH EQU *-WTOAREA             Length of WTO area
         SPACE ,
GETENTRY DS    A                     Address of BPX1GPS
RETVAL   DS    F                     Return value - next
RETCODE  DS    F                     Return code
RSNCODE  DS    F                     Reason code
         SPACE ,
BUFFERA  DS    CL50                  Buffer for Process data
BUFFERB  DS    CL500                 Buffer for Process data
PGTHAL   DC    A(PGTHA#LEN)          Length of PGTH buffer
PGTHAB   DS    A(PGTHA)              ->Process data buffer
PGTHBL   DC    A(500)                Length of PGTH buffer
PGTHBB   DS    A(PGTHB)              ->Process data buffer
         BPXYPGTH DSECT=NO           Place in current CSECT / DSECT
         END
```

Example of nonreentrant entry linkage—AMODE 64

This example shows the function for the __getthent service in a nonreentrant program. For a reentrant example of __getthent, see "BPX4GTH (__getthent) example" on page 1243. For an example of reentrant entry and return linkage, see Appendix E, "Callable services examples—AMODE 64," on page 1215 and "Reentrant return linkage" on page 1302.

```
BPXB1SM6 CSECT ,                     Nonreentrant linkage
BPXB1SM6 AMODE 64
         SYSSTATE AMODE64=YES
@BEGIN0  J     @BEGIN1               Branch around program header
         DC    C'BPXB1SM6 - nonreentrant __getthent invoker'
         DS    0H
@BEGIN1  STMG  R14,12,12(R13)        Save callers registers
         BRAS  R12,PDATA1            Establish addressability save area ea
         DC    A(@SAVE00)
PDATA1   L     R12,0(,R12)
         USING @SAVE00,R12
         STG   R13,@BACK             Save ->Callers save area
         LA    R13,@SAVE00           Program addressability
         DROP  R12
         USING @SAVE00,R13           Program addressability
         J     @BEGIN2
@SAVE00  DS    0D                    Standard save area - 144 Bytes
         DS    A                     Reserved
         DS    CL4'F4SA'             Linkage Type
         DS    15AD                  Regs 14,15,0-12
@BACK    DS    AD                    Backwards save area pointer
@FORWARD DS    AD                    Forwards save area pointer
RETURN   XR    R15,R15               Zero return code
RETURNRC LG    R13,@BACK             Restore callers r13
```

Example of nonreentrant entry linkage—AMODE 64

```
         LG    R14,12(,R13)          Restore callers r14
         LMG   R0,R12,20(R13)        Restore callers r0-r12
         BR    R14                   Branch back to caller
R0       EQU   0
R1       EQU   1                     Parameter list pointer
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13                    Program and save area base
R14      EQU   14                    Return address
R15      EQU   15                    Branch location
@BEGIN2  EQU   *  *  *  *  *  *  *   End of the entry linkage code
         EJECT ,
         LA    R5,BUFFERA            R5-> Input buffer
         STG   R5,PGTHAB             -> input buffer
         USING PGTHA,R5              R5 base for PGTHA
         XC    PGTHA,PGTHA           Null input area
         MVI   PGTHAFLAG1,PGTHAPROCESS+PGTHATHREAD
         MVI   PGTHAPID,PGTH#FIRST   First thread
         LA    R15,BUFFERB           Pgthb, Output buffer
         STG   R15,PGTHBB            Output Buffer
         SPACE ,  *  *  *  *  *  *
         LA    R0,=CL8'BPX4GTH '     LOAD -> entry point name
         XGR   R1,R1                 No JOBLIB or LINKLIB DCB
         SVC   8                     Issue LOAD SVC
         NILL  R0,X'FFFE'            Turn off low order bit
         STG   R0,GETENTRY           Store BPX4GTH entry point
GETTH    LG    R15,GETENTRY          Address of BPX4GTH load module
         CALL  (15),                 Get process data                 +
               (PGTHAL,              Length of buffer                 +
               PGTHAB,               Buffer, mapped by BPXYPGTH       +
               PGTHBL,               Length of output buffer          +
               PGTHBB,               Buffer, mapped by BPXYPGTH       +
               RETVAL,               Return value (next, eof or error) +
               RETCODE,              Return code                      +
               RSNCODE),             Reason code                      +
               LINKINST=BALR
         SPACE ,  *  *  *  *  *  *
         L     R15,RETVAL            Load return value
         C     R15,=F'-1'            Test for -1 return
         BE    RETURNRC              -1 is error
         SPACE ,  *  *  *  *  *  *   Initialize WTO area & message
         MVI   XPID,C' '             Blank out variable portion of msg  ge
         MVC   XPID+1(WTO#BLANK-1),XPID
         SPACE , *  *  *  *  *  *    Process ID to printable hex
         LA    R6,BUFFERB            R6-> Output buffer
         STG   R6,PGTHBB             -> output buffer
         USING PGTHB,R6              R6 base for PGTHB
         L     R8,PGTHBPID           R8 = process ID
         LA    R9,XPID               To be placed at message start
         LA    R15,8                 8 nibbles to convert (4 bytes)
         LA    R10,9                 For 0-9 / A-F compare
NIBBLE   LR    R11,R8                Target bits in 0-3    XYYYYYYZ
         SRL   R11,28                Bits 0-3 to 28-31     0000000X
         SLL   R8,4                  Drop bits 0-3 off end YYYYYYZ0
         CLR   R11,R10               Are 4 bits 0-9 or A-F
         BC    B'0010',AF            Branch if A-F
         LA    R11,57(,R11)          Add for 0-9 (57+183=240 or F0)
AF       LA    R11,183(,R11)         Add for 0-F (183+10=193 or C1)
```

```
         STC   R11,0(,R9)            Store to results location
         LA    R9,1(,R9)             Increment R9 to next location
         BCT   R15,NIBBLE            Decrement half byte counter, loop
         SPACE ,  *  *  *  *  *  *   Test status bits
* Go after the state of the process
         LA    R7,PGTHB              Get the PGTHB address
         SLR   R9,R9                 Clear r9
         ICM   R9,7,PGTHBOFFC        Get offset for PGTHC
         AR    R7,R9                 Calculate address
         USING PGTHC,R7              Addressability for PGTHC
         LA    R8,PGTHB              Get the PGTHB address
         SLR   R9,R9                 Clear r9
         ICM   R9,7,PGTHBOFFJ        Get offset for PGTHJ
         AR    R8,R9                 Calculate address
         USING PGTHJ,R8              Addressability for PGTHJ
         MVI   THREAD,C'1'           Assume single
         TM    PGTHCFLAG1,PGTHCMULPROCESS   if multiprocess
         BZ    NOTMULT
         MVI   THREAD,C'M'
NOTMULT  MVC   STATE,PGTHJSTATUS2    Z, W, X, S, C, F, K, R ...
         TM    PGTHCFLAG1,PGTHCSWAP  if swapped out
         BZ    NOTSWAP
         MVC   SWAPA,=CL4'SWAP'
NOTSWAP  TM    PGTHCFLAG1,PGTHCSTOPPED       if stopped
         BZ    NOTSTOP
         MVC   STOPA,=CL4'STOP'
NOTSTOP  TM    PGTHCFLAG1,PGTHCTRACE         if ptrace
         BZ    NOTTRAC
         MVC   TRACA,=CL4'TRAC'
NOTTRAC  EQU   *
         SPACE ,  *  *  *  *  *  *   Display message to operator
         LA    R2,WTOAREA            R2->WTO message area
         WTO   TEXT=(R2)             Write to Operator
         SPACE ,  *  *  *  *  *  *   Loop back
         MVC   PGTHACONTINUE,PGTHBCONTINUE  get next thread, process
         J     GETTH
WTOAREA  DS    0F                    WTO message
         DC    AL2(WTO#LENGTH)       Length of area
         DC    CL4'PID='             Process ID =
XPID     DS    CL8                   Hex of process ID
         DS    CL1
THREAD   DS    CL1                   1, M
         DS    CL1
STATE    DS    CL1                   Z, W, X, C, F, K, R ...
         DS    CL1
SWAPA    DS    CL4                   SWAP or blank
         DS    CL1
STOPA    DS    CL4                   STOP or blank
         DS    CL1
TRACA    DS    CL4                   TRAC or blank
WTO#BLANK EQU  *-XPID                Length to blank
         DC    C'.'
WTO#LENGTH EQU *-WTOAREA             Length of WTO area
         SPACE ,
GETENTRY DS    AD                    Address of BPX4GTH
RETVAL   DS    F                     Return value - next
RETCODE  DS    F                     Return code
RSNCODE  DS    F                     Reason code
         SPACE ,
BUFFERA  DS    CL50                  Buffer for Process data
BUFFERB  DS    CL500                 Buffer for Process data
PGTHAL   DC    A(PGTHA#LEN)          Length of PGTH buffer
PGTHAB   DS    AD(PGTHA)             ->Process data buffer
PGTHBL   DC    A(500)                Length of PGTH buffer
PGTHBB   DS    AD(PGTHB)             ->Process data buffer
         BPXYPGTH DSECT=NO           Place in current CSECT / DSECT
         END
```

Example of nonreentrant entry linkage—AMODE 64

Appendix G. The relationship of z/OS UNIX signals to callable services

The signal information in this topic is needed by compiler writers who are implementing POSIX in a high-level language.

Signals support the following callable services:

- "alarm (BPX1ALR, BPX4ALR) — Set an alarm" on page 29
- "kill (BPX1KIL, BPX4KIL) — Send a signal to a process" on page 304
- "mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup" on page 430
- "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421
- "pause (BPX1PAS, BPX4PAS) — Suspend a process pending a signal" on page 468
- "ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging" on page 537
- "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746
- "sigpending (BPX1SIP, BPX4SIP) — Examine pending signals" on page 755
- "sleep (BPX1SLP, BPX4SLP) — Suspend execution of a process for an interval of time" on page 771
- "queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered" on page 568
- "sigprocmask (BPX1SPM, BPX4SPM) — Examine or change a process's signal mask" on page 757
- "sigsuspend (BPX1SSU, BPX4SSU) — Change the signal mask and suspend the thread until a signal is delivered" on page 763

High-level-language signal interfaces

In addition to the signal interface callable services that are defined by POSIX, z/OS UNIX provides the following signal interface services:

mvssigsetup service
: Sets up and defines the *signal interface routine (SIR)*. The SIR is a routine that is provided by the high-level language. For information about how to write the SIR and the interface to it, see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421.

mvsunsigsetup service
: Detaches the interface to the SIR and returns the parameters set up in mvssigsetup. See "mvsunsigsetup (BPX1MSD, BPX4MSD) — Detach the signal setup" on page 430.

ptrace service
: Controls the running of another process for debugging programs. See "ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging" on page 537.

queue_interrupt service
: Returns the last signal delivered. See "queue_interrupt (BPX1SPB, BPX4SPB) — Return the last interrupt delivered" on page 568.

These interfaces allow a runtime library (RTL) for a high-level language to control the flow of signals. Each high-level language defines its own linkage interface between callable procedures; for example, the C language has a linkage stack and register interface between function procedures, which are unique to C.

Delivery of signals involves:

- Interrupting a currently running procedure
- Saving the status of the code that was interrupted
- Invoking a callable procedure known as the *signal catcher*, or signal handler.

How high-level languages use signals

Invoking a callable service involves setting up registers that are unique to the high-level language.

1. The RTL, using these callable services, sets up a SIR to receive control when a signal occurs.
2. The SIR procedure performs the necessary language linkages and POSIX functions to call the signal catcher procedure.
3. The signal catcher may return to the SIR.
4. The SIR performs the necessary language and POSIX functions to return to the interrupted procedure after the signal catcher returns.
5. The CSRL16J system service loads all registers and the PSW condition code and jumps to the instruction that was interrupted by the signal.

Signal setup when linking to callable services

When a task invokes the first z/OS UNIX call, the address space (if needed) and task are set up for z/OS UNIX callable services. Setting up for z/OS UNIX callable services is known as dubbing the address space and dubbing the task. When an address space is dubbed, a new process is created and assigned a unique process ID.

A *dubbed task* is a thread that is assigned an 8-character thread ID. This thread ID is unique within the process. Threads in different processes could have the same thread ID. When the first z/OS UNIX call is made and the task is dubbed, the current program request block (PRB) that dubbed the task is also recorded. This not only dubs the task, but also sets it up for signals.

Figure 6 on page 1315 shows the flows for the various signal functions when a synchronous signal SIGPIPE is generated with the kill service.

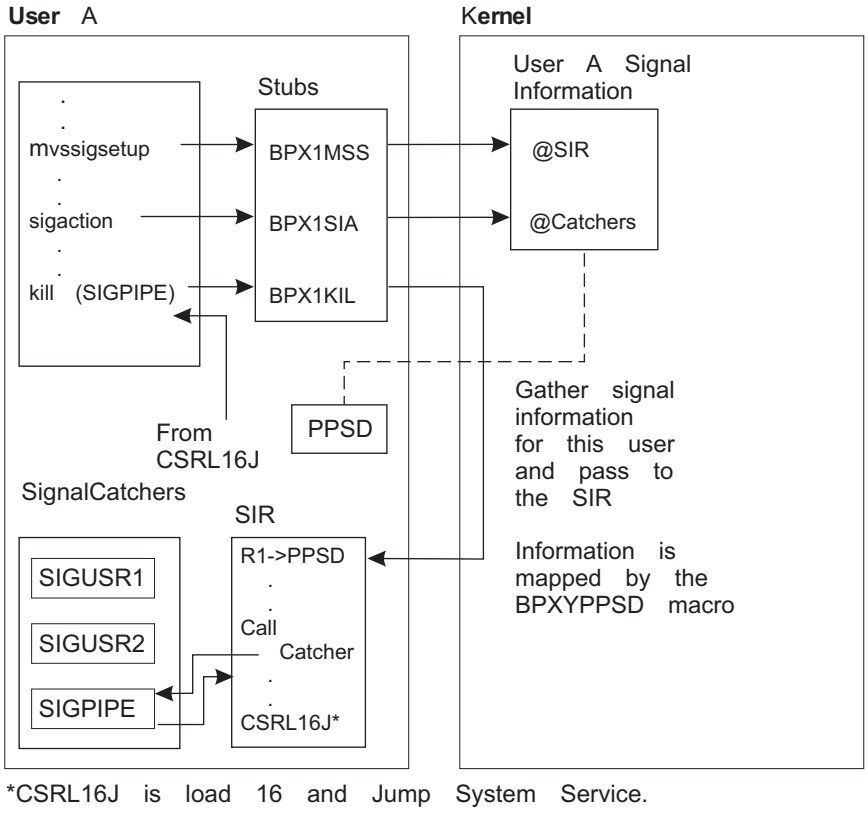


Figure 6. Program flow of mvssigsetup and sigaction with signal interface routine (SIR)

For more information about the setup and use of SIRs, see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421. For more information about signal catchers, see "sigaction (BPX1SIA, BPX4SIA) — Examine or change a signal action" on page 746.

ESPIE or ESTAE and the SIGILL, SIGFPE, and SIGSEGV signals

High-level languages generate the **SIGILL**, **SIGFPE**, and **SIGSEGV** signals. In z/OS UNIX, the kill service is invoked to generate these signals. The ESPIE or ESTAE must also use the kill service to generate **SIGILL**, **SIGFPE**, and **SIGSEGV.** High-level languages can define an ESPIE or ESTAE routine to receive control after an incorrect hardware instruction, arithmetic operation, or memory reference.

Since z/OS UNIX does not generate or process the signals **SIGILL**, **SIGFPE**, and **SIGSEGV**, it is the responsibility of the high-level language's RTL to define what happens when a signal catcher is defined for these signals and the signal catcher returns to the failing instruction. For information on how the compiler defines what happens in this case, see *z/OS XL C/C++ Programming Guide*.

ESPIE or ESTAE routines in high-level languages must also invoke the ptrace service. For more information about the ptrace service see "ptrace (BPX1PTR, BPX4PTR) — Control another process for debugging" on page 537.

When signals are and are not supported

All signal functions are supported when the task is set up for signals, when it is running with the signal delivery key, and when its current program request block (PRB) is the same PRB as when the task was set up for signals. When this is not

the case, some signal functions are not supported, or they function differently. Table 26 defines these signal functions.

The mvssigsetup columns in Table 26 describe a task that is set up with the mvssigsetup service. When a task invokes the mvssigsetup service, the current PRB is recorded for future signal delivery. When a task is set up for signals by mvssigsetup, signals are only delivered when the task's current PRB is the same PRB that called mvssigsetup.

Table 26. Support of signal calls

	Task mvssigsetup			Task not mvssigsetup	
Service	**The current PRB called mvssigsetup**	**The current PRB did not call mvssigsetup**	**Not signal delivery key**	**The current PRB dubbed the task**	**The current PRB did not dub the task**
BPX1ALR	RV=Seconds	Abend	RV=Seconds	RV=Seconds	Abend
BPX1KIL	RV=0	RV=0	RV=0	RV=0	RV=0
BPX1MSD	RV=0	RV=0	RV=0	RV=-1	RV=-1
BPX1MSS	RV=-1	RV=-1	RV=-1	RV=0	RV=0
BPX1PAS	RV=0	RV=-1	RV=-1	RV=0	RV=0
BPX1SEL	RV=0	RV=-1	RV=-1	RV=0	RV=0
BPX1SIA	RV=0	RV=-1	RV=0	RV=-1	RV=-1
BPX1SIP	RV=0	RV=-1	RV=0	RV=0	RV=0
BPX1SLP	RV=Seconds	RV=Abend	RV=Abend	RV=Seconds	RV=Seconds
BPX1SPB	RV=0	N/A	N/A	N/A	N/A
BPX1SPM	RV=0	RV=-1	RV=0	RV=0	RV=0
BPX1SSU	RV=0	RV=-1	RV=-1	RV=0	RV=0

Notes:

PRB Program request blocks are created by MVS system services such as LINK. PRBs are also created for ESTAE routines.

RV Return value returned in the service.

N/A Not applicable

Signal delivery keys

Signal delivery also depends on the signal delivery key. Each process has one signal delivery key. The signal delivery key is set to the PSW key of the caller of the first z/OS UNIX call that created the process. A process created by the fork or exec service has key 8. The attach_exec service works differently from the exec and fork service; it creates a process with a signal delivery key equal to the PSW key of the Attach_exec caller. Key zero is not a valid signal delivery key. Therefore, if the caller's PSW key is zero when mvssigsetup created the process, the mvssigsetup call fails and signal catchers cannot be invoked in this process.

Delayed signal delivery

Asynchronous signals are generated from a process or task different from the task the signal is being delivered to. Delivery of asynchronous signals is not always possible and can have a delay. Signals that must be delayed are delivered later, when signals are permitted and the next z/OS UNIX service is invoked. The following describes some additional cases when signal delivery must be delayed:

- During STORAGE obtains or releases that use a hardware linkage stack.
- During execution of system services that are entered with PC or that use the hardware linkage stack (such as a BAKR instruction).
- When applications use a BAKR instruction on entry to save registers in a hardware linkage stack and use a PR instruction to restore registers on exit. Therefore, asynchronous signals cannot be delivered after the BAKR instruction and before the PR instruction.
- When a task that is set up for signals by a mvssigsetup service is followed by a system service call (for example, LINK) that creates another program request block (PRB).

z/OS UNIX System Services provides a signal deferral capability that allows an application to defer the receipt of signals until it is ready to accept them. You could use it, for instance, to shield an application from signal interruption during a time of critical processing. Once the section of critical code had finished, the application could receive any signals that had been deferred.

To use the signal deferral capability, the application sets the ThliDeferSignals bit on in the THLI data structure. When it is interested in receiving signals again, it sets this bit off. To see if any signals are pending, the application can check the OtcbSigPending or the ThliSigPending bit. If OtcbSigPending or ThliSigPending is set on, it can set ThliDeferSignals = OFF, and call BPX1GPI to drive signal delivery.

To access the THLI bit, traverse the data structures TCB, STCB, OTCB, and THLI. If the STCBOTCB (the field in the STCB that points to the OTCB) is 0, the process is not dubbed and the THLI has not been created. (However, since a process that has not been dubbed cannot receive signals, it is not necessary to set the THLI bit to defer their handling.) If there is an OTCB, the OTCBTHLI points to the THLI. Set the ThliDeferSignals bit accordingly.

For example:

```
If (stcbotcb ^= 0) then             /* Make sure the process is dubbed, the otcb pointer  */
                                    /* will not be zero.                                  */

   otcbthli->thlidefersignals = ON; /* The otcbthli field points to the thli; set the thli*/
                                    /* to defer signals.                                  */

...start of important stuff
                                    /* Remember not to issue any syscalls during this     */
                                    /* segment of code. A syscall will force a delivery   */
                                    /* of any pending signal.                             */
...end of important stuff

otcbthli->thlidefersignals = off;   /* Reset the bit.                                     */

If otcbthli-thlisigpending = on     /* Check to see if any signals were made pending      */
                                    /* during the critical code interval.

   then call bpx1gpi(...)           /* Make any syscall. It will have all pending signals */
                                    /* delivered.                                         */
```

This mechanism is not intended to be used by an application that is requesting z/OS UNIX system services. If a syscall is requested, any pending signals are delivered. The THLI bit is intended to shield the application from unwanted interruptions only when no syscalls are being performed.

When signals cannot be delivered

Compilers and applications that enter states when signals cannot be delivered should invoke z/OS UNIX callable services after returning to a state where signal delivery is possible. This action ensures prompt delivery of signals. For example, a program may invoke a STORAGE obtain and getpid service. After returning from the getpid service, z/OS UNIX delivers any asynchronous signals that were generated during the STORAGE obtain.

When the SIR is unable to deliver a signal to a signal catcher routine for environmental reasons, the queue_interrupt service is invoked from a signal interface routine (SIR). The queue_interrupt service also delays signal delivery until the next z/OS UNIX callable service. z/OS UNIX callable services should be performed shortly after a queue_interrupt call to ensure prompt signal delivery.

Signals and multiple tasks created by ATTACH

This section describes processes that have multiple dubbed tasks created by using the ATTACH system service. It describes how the first dubbed task in a process can be created and how to create additional dubbed tasks using ATTACH. It also describes how signals work in a process with multiple dubbed tasks created by ATTACH.

The first dubbed thread in a process can be created with the fork callable service or the exec or execmvs callable service, or by the first call to az/OS UNIX callable service from any task in the address space. Subsequent tasks can be created in the process with the ATTACH system service. Once a program running on behalf of the task calls a z/OS UNIX callable service, the task becomes dubbed. Every dubbed task is assigned an 8-character thread ID.

The mvssigsetup and sigaction services allow only one thread in a process to set up a signal interface routine (SIR) and signal catchers. When a process contains two tasks with signals unblocked, the signal is delivered to the task that called mvssigsetup.

If signal action on delivery of a signal specifies termination, stop, or continue, the entire process is terminated, stopped, or continued. Delivery of a signal for default signal action occurs for any of the following conditions:

1. None of the threads is set up for signals by mvssigsetup and one or more threads do not have the signal blocked.
2. One of the threads is set up for signals by mvssigsetup and the signal is not blocked by the thread that called mvssigsetup.

Signals and multiple tasks created by pthread_create

The pthread_create service creates dubbed tasks within the process. This section describes how signals work in processes that have multiple dubbed tasks created by the pthread_create service and ATTACH system service.

A thread created by pthread_create also inherits any signal setup information created by a prior mvssigsetup call. If the caller of pthread_create had previously called mvssigsetup successfully, the thread created is also set up for signals. The mvssigsetup and pthread_create services can be used to create multiple threads in a process that is set up for signals.

When a signal is generated by a kill service request to a process that has multiple threads set up for signals and threads that are not set up for signals, z/OS UNIX signal processing must determine which thread has the most interest in the signal. The signal is delivered to the thread with the most interest when a signal catcher is defined by a sigaction call.

The following is a list of signal interest rules for a signal generated by a kill call from most to least interested:

1. When threads are found in a sigwait for this signal, the signal is delivered to the first thread found in a sigwait.
2. When all threads are blocking this signal, the signal is left pending at the process level. The sigpending service moves blocked pending signals at the process level to the thread-level.
3. When the default terminating signal action (not ignore and not catch) is to take place, that action is performed for all threads in the process.
4. When all of the following are true:
   - One or more threads are set up for signals.
   - All threads set up for signals have the signal blocked.
   - A thread not set up for signals has not blocked the signal.

   The signal is left pending on the first thread set up for signals. This signal remains pending on that thread until the thread unblocks the signal.
5. When one or more threads are set up for signals and at least one of the threads set up for signals has the signal unblocked, the signal is delivered to the first thread that is set up for signals that also has the signal unblocked.

Signal defaults

This section contains information about the signals that are supported by z/OS UNIX. These signals are mapped by the BPXYSIGH mapping macro; see "BPXYSIGH — Signal constants" on page 1039. The following table lists the signals and their default actions:

Constant	Value	Default action	Description
SIGABND#	18	1	Abend
SIGABRT#	3	1	Abnormal termination
SIGALRM#	14	1	Timeout
SIGBUS#	10	1	Bus error
SIGCHLD#	20	2	Child process terminated or stopped
SIGCONT#	19	4	Continue if stopped
SIGDANGER	33	1	Termination
SIGDUMP#	39	2	The system takes a SYSMDUMP and writes it to an MVS data set or a z/OS UNIX file. The _BPXK_MDUMP environment variable must be set to the name of the data set or file. This signal cannot be caught.

Constant	Value	Default action	Description
SIGFPE#	8	1	Erroneous arithmetic operation, such as division by zero or an operation resulting in overflow
SIGHUP#	1	1	Hangup detected on controlling terminal
SIGILL#	4	1	Detection of an incorrect hardware instruction
SIGINT#	2	1	Interactive attention
SIGIO#	23	2	Completion of input or output
SIGIOER#	27	2	I/O error
SIGKILL#	9	1	Termination (cannot be caught or ignored). Can result if abend not caught or handled and terminating status not set; CPU time exceeded and SIGXCPU# caught or ignored; or **sigkill** shell command sent.
SIGNULL#	0	2	Null; no signal sent (cannot be caught or ignored)
SIGPIPE#	13	1	Write on a pipe with no readers
SIGPOLL#	5	1	Pollable event
SIGPROF#	32	1	Profiling timer expired
SIGQUIT#	24	1	Interactive termination
SIGSEGV#	11	1	Detection of an incorrect memory reference
SIGSTOP#	7	3	Stop (cannot be caught or ignored)
SIGSYS#	12	1	Bad system call
SIGTERM#	15	1	Termination
SIGTHCONT#	35	1	Thread continue (cannot be caught or blocked or ignored)
SIGTHSTOP#	34	1	Thread stop (cannot be caught or blocked or ignored)
SIGTMOUT#	40	1	Terminates a process waiting for terminal activity. Cannot be caught or ignored. Is reserved for z/OS use. It cannot be sent by an application program.
SIGTRACE#	37	2	Toggles the user syscall trace setting ON or OFF.
SIGTSTP#	25	3	Interactive stop
SIGTTIN#	21	3	Read from a control terminal attempted by a member of a background
SIGTTOU#	22	3	Write from a control terminal attempted by a member of a background process group
SIGTRAP#	26	1	Trap used by the ptrace call
SIGURG#	6	2	High bandwidth data is available at a socket
SIGUSR1#	16	1	Reserved as application-defined signal 1
SIGUSR2#	17	1	Reserved as application-defined signal 2 process group
SIGVTALRM#	31	1	Virtual timer expired
SIGXCPU#	29	1	CPU time limit exceeded
SIGXFSZ#	30	1	File size limit exceeded
SIGWINCH#	28	2	Change size of window

The default actions are:

1. Abnormal termination.
2. Ignore the signal.
3. Stop the process.
4. Continue if it is currently stopped; otherwise, ignore the signal.

Appendix H. Using threads with callable services

z/OS UNIX threads are tasks that are using z/OS UNIX services. Pthreads are z/OS UNIX threads that are created with pthread_create; this also includes the *initial pthread-creating task (IPT)*. The first thread in a process to invoke the pthread_create service becomes the IPT. This topic contains information about creating pthreads, the IPT, terminating pthreads, and multiple pthreads. It also shows scenarios for different termination situations.

Creating threads

Threads are created as follows.

The successful completion of:

- The pthread_create service
- The fork or exec service
- Most z/OS UNIX service requests from an undubbed MVS task

A single-threaded process is created with fork, with exec, or by the invocation of a kernel service from within an MVS address space.

Multiple-threaded processes can be created with pthread_create.

The IPT and all pthread_created threads are referred to as *pthreads*. All future pthread_create requests attach subtasks to the IPT, even though they are not issued by the IPT. This is important in thread termination. For a complete description of the process of creating threads, see "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497.

The pthread_create task initialization routine

The first routine that is given control in the new task when a thread is created with the pthread_create service is the pthread_create pthread-creating task initialization routine. (The pthread-creating task initialization routine is not the same as the initial pthread-creating task (IPT). The pthread-creating task initialization routine is the routine that is given control when a pthread_create is done, whereas the IPT refers to the task that the first task runs on.) The pthread_create pthread-creating task initialization routine does the following:

1. Acquires task-related resources required by the user application.
2. Calls pthread_exit_and_get service to exit the old thread and get the new thread information. The exit of the old thread is ignored if this is the first call to pthread_exit_and_get.
3. Checks for failures. If a failure is found, it skips to step 8.
4. Gets pthread-related resources for the newly created thread.
5. Calls the user-specified Start_routine.
6. Releases resources for the newly created thread.
7. Repeats step 2.
8. Releases task-related resources.
9. Returns to the caller (ends the task).

Note: When control is returned after a successful pthread_exit_and_get call, the thread can be interrupted by any signals that are not blocked. The signal blocking mask of the created thread is inherited from the thread that invoked the created thread.

Terminating pthreads

Note: If multiple threads are created with a combination of pthread_create and dubbed MVS tasks, the following termination methods do not apply. The exception to this, of course, is that the IPT is a dubbed task. If the IPT has any subtasks that are non-pthread threads, the following termination scenarios also do not apply.

There are no prescribed methods for terminating threads that are mixed with other dubbed tasks in a single process.

There are three ways to terminate a thread without exiting the process:

- The pthread_exit_and_get (BPX1PTX) service terminates the thread that invoked it. If it is successful, control is returned to the invoking task.
- The pthread_cancel (BPX1PTB) service generates a cancel request to the target thread. After the cancel request is delivered, the thread and its associated task are terminated by the kernel. This behavior can be circumvented if the thread intercepts the cancelation request (see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421).
- The pthread_quiesce (BPX1PTQ) service sends a quiesce event to all other pthreads in the process. If the other pthreads do not intercept the quiesce event (see "mvssigsetup (BPX1MSS, BPX4MSS) — Set up MVS signals" on page 421), delivery of the event terminates the thread and the task, if the target is not the IPT.

The two types of threads that are created with pthread_create require different actions for terminating.

Heavyweight thread (HWT)

Terminating an HWT requires that the task also terminate. That is, after a pthread_exit_and_get service is issued to exit an HWT, z/OS UNIX services, with the exception of mvsprocclp, can no longer by issued from this task.

Mediumweight thread (MWT)

Terminating an MWT does not require that the task terminate. You can terminate it by using the PTGETNEWTHREAD option on the pthread_exit_and_get service. The pthread-creating task initialization routine that is specified on the pthread_create service can repeatedly call the pthread_exit_and_get service, getting new thread requests as they are created. This avoids the overhead of task creation and termination for each thread.

For information on HWTs and MWTs, see "pthread_create (BPX1PTC, BPX4PTC) — Create a thread" on page 497.

Terminating multiple pthreads and tasks

Terminating a pthread is different from terminating the task that the pthread runs on. The IPT should be the last task to terminate; that is, the IPT should wait for all pthreads and supporting tasks to terminate before it terminates. If the IPT and its associated task should terminate before all its subtasks terminate, those subtasks

abend asynchronously with a 33E abend. This type of termination does not allow an orderly cleanup of pthread and task-related resources.

When a process contains multiple pthreads, and one of the executing pthreads starts process termination, the following steps should be taken:

1. The terminating pthread uses the pthread_quiesce (BPX1PTQ) service to inform all other pthreads that are running in the process of its process termination.
2. The pthread_quiesce service places the issuing pthread in a wait state until all other pthreads are notified and have terminated.
3. As each pthread's signal interface routine receives the quiesce notification, it uses the pthread_exit_and_get service to terminate the pthread. The signal interface routine should not pass control to the user program, because it might continue processing. The task that invoked pthread_quiesce is waiting for all the pthreads in the process to terminate.
4. The pthread that is issuing the pthread_quiesce service gains control after all pthreads have terminated. The terminating pthread can then invoke any exit and cleanup functions that are necessary for an orderly termination of the process.

   Note: The tasks that supported quiesced pthreads can still be running after control is returned to the task that issued pthread_quiesce. Only the pthreads have terminated, not the tasks. Terminating the task is a separate and asynchronous part of terminating the process.
5. The terminating pthread can then issue a terminating service request such as exit, _exit, or exec. If the terminating pthread is the IPT, the mvsprocclp (BPX1MPC) service can be issued instead of the _exit (BPX1EXI) service. This avoids the automatic termination of the task.
6. The IPT gains control only when all the pthreads that were created with pthreads_create have terminated. The IPT can then call mvsprocclp (BPX1MPC) to clean up the remaining z/OS UNIX environment. Control cannot return to the IPT until all the other tasks that supported the pthreads have exited. If any of the pthread subtasks fail to terminate, mvsprocclp sets a failing return code.
7. Now that all the tasks have terminated (except for the IPT), control is returned to the caller of the application (if one exists) or back to the system (which terminates the IPT).

Pthread termination scenarios

The following scenarios describe the steps needed to terminate multithread processing for situations application programmers might encounter.

Using exit or _exit when the thread is not the IPT

Table 27 describes the actions that are taken for exit or _exit issued from a thread created with pthread_create.

Table 27. Using exit or _exit when the thread is not the IPT

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1		A request to exit the process was issued.
2		A pthread_quiesce is issued. Control is not returned until all other pthreads in this process end with pthread_exit_and_get.
3	An asynchronous quiesce event is delivered to this thread.	

Table 27. Using exit or _exit when the thread is not the IPT (continued)

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
4	Either the thread is terminated by the kernel, or the signal interface routine intercepts the quiesce termination event to do necessary thread cleanup and issue another pthread_exit_and_get. Interception of quiesce events must be specified by the mvssigsetup service.	
5	The IPT thread is terminated, and the IPT is placed in a wait state in the kernel.	
6		Control is returned from pthread_quiesce when all other pthreads terminate.
8		An _exit service request is issued to terminate the process and pass the process status. This pthread and task are both terminated, and control is not returned to the _exit service caller.
9	The kernel posts the IPT when the last pthread terminates.	
10	The mvsprocclp service is issued to clean up any remaining portions of the process. Control returns from this service after all subtasks created with pthread_create terminate, or when a reasonable amount of time to do this has elapsed.	
11	The IPT gains control after the mvsprocclp service completes. All pthreads for this process and all subtasks of the IPT have terminated.	
12	The IPT is no longer associated with the kernel and can now return to its caller or to the system.	

Using exit or _exit when the thread is the IPT

Table 28 describes the actions that are taken for exit or _exit issued from the IPT thread.

Table 28. Using exit or _exit when the thread is the IPT

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1	A request to exit the process is issued from the IPT.	
2	A pthread_quiesce is issued. Control is not returned until all other pthreads in this process end with pthread_exit_and_get.	
3		An asynchronous quiesce event is delivered to this thread.
4		Either the thread and its associated task are terminated by the kernel, or the signal interface routine intercepts the quiesce termination event to do necessary thread cleanup and to issue another pthread_exit_and_get. Interception of quiesce events is specified by the mvssigsetup service.
5		Control is returned to the pthread-creating task initialization routine (QUIESCE_TERM only), the remaining parts of the environment are cleaned up, and control is returned to the caller, terminating the task.

Table 28. Using exit or _exit when the thread is the IPT (continued)

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
6	Control is returned after the pthread_quiesce call when all other pthreads terminate (perhaps not all tasks have terminated yet).	
7	Process the remaining thread and clean up (such as running exits).	
8	Terminate the process and pass the process status with mvsprocclp status.	
9	When control is returned from mvsprocclp, all pthreads for this process and all subtasks of the IPT have terminated.	
10	The IPT task is no longer associated with the kernel and can now return to its caller or to the system.	

Using pthread_exit_and_get when the thread is not the IPT and not the last thread

Table 29 describes the actions that are taken when pthread_exit_and_get is issued on a thread that is not the IPT and is not the last thread.

Table 29. Using pthread_exit_and_get when the thread is not the IPT and not the last thread

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1		A request to exit the pthread is issued.
2		Run thread cleanup routines before terminating this thread.
3		Return to the pthread-creating task initialization routine that issued pthread_exit_and_get to terminate the thread, using the PTEXITTHREAD and PTGETNEWTHREAD option for MWTs or the PTEXITTHREAD option for HWTs. If you want to know when the last thread is terminating so that process termination cleanup can be done first, specify PTFAILIFLASTTHREAD. You must then call pthread_exit_and_get again, but this time without the PTFAILIFLASTTHREAD option.
4		For MWTs, this task waits in the kernel until the next new pthread_create request. When pthread_exit_and_get returns a -1 return value, a new thread was not created. You must exit the pthread-creating task initialization routine, terminating the task. (You must always do this for HWTs.)
5		A successful return from pthread_exit_and_get indicates that this was not the last thread that terminated. If the PTEXITTHREAD and PTGETNEWTHREAD option was used, a new thread was returned.

Using pthread_cancel when the thread is not the last thread and is canceled

Table 30 on page 1326 defines the actions that are taken when the pthread_cancel request is handled by the signal interface routine, and the cancel causes the thread

to terminate. This is the same as when the target thread issues pthread_exit_and_get. The status of the thread is -1, and is available for joining threads.

Table 30. Using pthread_cancel when the thread is not the last thread and is canceled

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1		The pthread_cancel request was received and delivered to the signal interface routine. Interception of cancelations must be specified by the mvssigsetup service.
2		Set Status_field in the pthread_exit_and_get service to -1. See "pthread_exit_and_get (BPX1PTX, BPX4PTX) — Exit and get a new thread" on page 505.
3		Now follow the steps in Table 29 on page 1325.

Using pthread_exit_and_get when the thread is the IPT and not the last thread

Table 31 describes the actions that are taken when pthread_exit_and_get is issued on a thread that is the IPT and is not the last thread. The IPT is placed in wait state until all other pthreads in this process terminate.

Table 31. Using pthread_exit_and_get when the thread is the IPT and not the last thread

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1	A request to exit the pthread was issued.	
2	Run thread cleanup routines before terminating this thread.	
3	To terminate the thread, issue the pthread_exit_and_get service with the PTEXITTHREAD option. To determine when the last thread has terminated so that process termination cleanup can be done first, use the pthread_exit_and_get service with the PTFAILIFLASTTHREAD option. Then repeat the pthread_exit_and_get service, but without the PTFAILIFLASTTHREAD option.	
4	The IPT is now in a wait state until the process terminates.	
5	A return from pthread_exit_and_get indicates that all other pthreads for the process have terminated.	
6	The mvsprocclp service is issued to clean up any remaining portions of the process. Control returns from this call after all subtasks that were created with pthread_create terminate, or until time to do so has elapsed.	
7	The IPT task gains control after the mvsprocclp call. All pthreads for this process and all subtasks of the IPT have terminated.	
8	The IPT task is no longer associated to the kernel, and can now return to the caller or to the system.	

Using pthread_exit_and_get when the thread is not the IPT and is the last thread

Table 32 describes the actions that are taken when pthread_exit_and_get is issued on a thread that is not the IPT and is the last thread.

Table 32. Using pthread_exit_and_get when the thread is not the IPT and is the last thread

Step	Thread 1 (initial pthread-creating task, or IPT)	Thread 2 (pthread-created thread)
1	The IPT is in a wait state because of a previous pthread_exit_and_get.	pthread_exit_and_get is issued from this thread.
2		Run thread cleanup routines before this thread terminates.
3		Return to pthread-creating task initialization routine that issues pthread_exit_and_get to exit the thread, using the PTEXITTHREAD and PTGETNEWTHREAD option for MWTs or the PTEXITTHREAD option for HWTs. If you want to know when the last thread is terminating so that process termination cleanup can be done first, specify the PTFAILIFLASTTHREAD option. You must then call pthread_exit_and_get again, but this time without the PTFAILIFLASTTHREAD option.
4		A failing return value and reason code from pthread_exit_and_get indicates that this is the last thread.
5		Process the remaining thread and clean up (such as running exits).
6		Call pthread_exit_and_get without the PTFAILIFLASTTHREAD option to terminate the last thread and the process.
7		Clean up any MVS resources that may have been obtained STAE/SPIE/storage, after control is returned from pthread_exit_and_get to the pthread-creating task initialization routine.
8		The pthread-creating task initialization routine returns to its caller, terminating the task. The IPT is posted when this task terminates.
9	The IPT gains control after its pthread_exit_and_get and all threads have terminated.	
10	Issue the BPX1MPC service to clean up any remaining portions of the process. Control returns from this call after all subtasks created with pthread_create terminate, or until the time to do so has elapsed.	
11	The IPT task gains control when control is returned from mvsprocclp and all pthreads for this process and all subtasks of the IPT have terminated.	
12	The IPT task is no longer associated with the kernel, and can now return to its caller or to the system.	

Using pthread_exit_and_get when the IPT is the last thread

Table 33 describes the actions that are taken when pthread_exit_and_get is issued for a thread that is the IPT and is the last thread.

Table 33. Using pthread_exit_and_get when the IPT is the last thread

Step	IPT task is the only task	Thread 2 doesn't exist
1	A request to exit the pthread was issued.	
2	Run thread cleanup routines before this thread terminates.	
3	Call pthread_exit_and_get with the PTEXITTHREAD and PTFAILIFLASTTHREAD options to terminate the thread on the IPT.	
4	A return value and reason code reporting a failure from pthread_exit_and_get indicates that this is the last thread.	
5	Process the remaining thread and cleanup (such as running exits).	
6	Call pthread_exit_and_get without the PTFAILIFLASTTHREAD option to terminate the last thread and the process.	
7	Control is returned to the IPT from pthread_exit_and_get, and all threads terminate.	
8	The mvsprocclp service is issued to clean up any remaining portions of the process. Control returns from this service after all subtasks created with pthread_create terminate, or until the time to do so has elapsed.	
9	The IPT task gains control when the mvsprocclp service completes. All pthreads for this process and all subtasks of the IPT have terminated.	
10	The IPT task is no longer associated with the kernel, and can now return to its caller or to the system.	

Appendix I. Optimizing performance using process- and thread-level information

The process-level information area (PRLI) and the thread-level information area (THLI) contain information that can be used to optimize the performance of certain callable services. This information describes how to access the information in these areas and how the information can be used.

A thread-level information area (THLI) is created for each task in the system. The THLI is pointed to by the OTCB field OTCBTHLI. The OTCB is pointed to by a secondary task control block field, STCBOTCB.

A process-level information area (PRLI) is created for each process in the system. The PRLI is pointed to by the THLI field THLIPRLI for each task in the process.

The system maintains information in the PRLI and THLI that can be used to reduce the system overhead that is associated with certain callable services and improve their performance. The callable services that can use the information in these control blocks include:

- BPX1PSI, BPX4PSI (pthread_setintr)
- BPX1PST, BPX4PST (pthread_setintrtype)
- BPX1SPM, BPX4SPM (sigprocmask)
- BPX1GPI, BPX4GPI (getpid)

Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr)

Information in the THLI area can be used to optimize pthread_setintr (BPX1PSI, BPX4PSI) callable service invocations. BPX1PSI (BPX4PSI) must not be optimized if a signal is pending for the thread. A signal pending condition is indicated by the ThliSigPending flag. When this flag is on, indicating that a signal is pending, BPX1PSI (BPX4PSI) must be called to process the request and process signal delivery.

Table 34 maps the actions that can be taken for BPX1PSI (BPX4PSI) when there is no signal pending. The result column shows the action that the optimizing program can take. The cancel pending column reflects the setting of ThliCancelPending, and the current state column that of ThliCancelDisabled. The new state is provided by the caller of BPX1PSI (BPX4PSI). The interruptablity type, which is set by BPX1PST (BPX4PST), is not applicable to BPX1PSI (BPX4PSI) processing.

Table 34. Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr)

The current state	New state	Int. type	Cancel pending	Result
Disabled	Disabled	N/A	N/A	Return "disabled"
Enabled	Disabled	N/A	N/A	Issue BPX1PSI (BPX4PSI)
Enabled	Enabled	N/A	Yes	Issue BPX1PSI (BPX4PSI)

Table 34. Optimization processing for BPX1PSI, BPX4PSI (pthread_setintr) (continued)

The current state	New state	Int. type	Cancel pending	Result
Enabled	Enabled	N/A	No	Return "enabled"
Disabled	Enabled	N/A	N/A	Issue BPX1PSI (BPX4PSI)
N/A	Invalid	N/A	N/A	Issue BPX1PSI (BPX4PSI)

In other words, the optimizing program should issue the BPX1PSI (BPX4PSI) if there is a request to change the interruptability state, or if the state is enabled and there is a cancel pending, as indicated by the ThliCancelPending bit.

Optimization processing for BPX1PST, BPX4PST (pthread_setintrtype)

Information in the THLI area can be used to optimize pthread_setintrtype (BPX1PST, BPX4PST) callable service invocations.

Table 35 maps the actions that can be taken for BPX1PST (BPX4PST) when there is no signal pending. The result column shows the action that the optimizing program can take. The cancel pending column reflects the setting of ThliCancelPending, the cancel disabled column the setting of ThliCancelPending, and the current interruptability type column the setting of ThliCancelAsync. The new interruptability type is provided by the caller of BPX1PST (BPX4PST).

Table 35. Optimization processing for BPX1PST, BPX4PST (pthread_setintrtype)

The current int. type	New int. type	Cancel disabled	Cancel pending	Result
Controlled	Controlled	N/A	N/A	Return "controlled"
Asynch	Controlled	N/A	N/A	Issue BPX1PST
Controlled	Asynch	N/A	N/A	Issue BPX1PST (BPX4PST)
Asynch	Asynch	Yes	N/A	Return "asynch"
Asynch	Asynch	No	No	Return "asynch"
Asynch	Asynch	No	Yes	Issue BPX1PST (BPX4PST)
N/A	Invalid	N/A	N/A	Issue BPX1PST (BPX4PST)

In other words, the optimizing program should issue the BPX1PST (BPX4PST) if there is a request to change the interruptability type; or if the type is asynchronous and cancel is not disabled (ThliCancelDisabled off) and there is a cancel pending (ThliCancelPending on).

Optimization processing for BPX1SPM, BPX4SPM (sigprocmask)

Information in the THLI data area can be used to optimize sigprocmask (BPX1SPM, BPX4SPM) invocations.

The optimizing program should first process the new mask that is provided by the caller of BPX1SPM (BPX4SPM), to determine if optimization is possible. If no new mask is provided, no change is being made to the signal mask, and this call can be optimized.

To process the new mask, the optimizing program should first generate the effective new mask using the new mask provided by the caller, clearing bits from this new mask for any signals that cannot be caught (ANDing the provided mask with PrliCatcherMask), and then applying the How requested by the caller as follows:

Table 36. Optimization processing for BPX1SPM, BPX4SPM (sigprocmask)

How	Effective mask
SIG_SETMASK	New mask ANDed with PrliCatcherMask
SIG_BLOCK	New mask ORed with ThliSigMask
SIG_UNBLOCK	Complement new mask ANDed with ThliSigMask
Other	An incorrect How was specified; issue BPX1SPM (BPX4SPM) or fail the request with an appropriate error code.

If the effective mask does not equal the current mask in ThliSigMask, a change in value of the current signal mask must be made, and BPX1SPM (BPX4SPM) should be issued. If the effective mask is the same as the current signal mask, the request is a NOOP and may be optimized.

If the request is being optimized and the caller requested that the previous value of the signal mask be returned, the optimizing program should return ThliSigMask to the caller.

Optimization processing for BPX1GPI, BPX4GPI (getpid)

Information in the PRLI data area can be used to optimize the getpid callable service invocations.

The optimizing program should return PrliProcessID if there is no signal pending; otherwise, getpid should be issued.

Appendix J. Callable services available to SRB mode routines

Overview

A subset of the callable services are now available to SRB mode routines. Supported callable services can be called from SRBs using the same conventions that are used when calling them from task mode routines. However, unlike task mode routines, SRBs do not cause process dubbing on the first issue of a callable service. In order to issue callable services, they must be associated with a dubbed process; that is, the SRB must be running in a dubbed address space. Upon issuing the callable service, it must place into register 2 the address of the OAPB control block that represents the associated process. When the OAPB address in register 2 is zero, the SRB is associated with the initial process in the address space.

Most applications consist of a single process per address space. These applications should default to the initial or only process in the address space, and set register 2 to zero when invoking a callable service. Applications creating multiple processes per address space most likely need to explicitly provide the address of the OAPB of the process to which the SRB is to be associated. In this case, the SRB typically receives the OAPB address from the routine scheduling the SRB. The OAPB address is obtained from the PRLI control block, which contains process-related control information intended for external use. The PRLI is addressed as follows:

```
TcbStcb -> StcbOtcb -> OtcbThli -> ThliPrli -> PrliOapb
```

The TCB referenced must represent a thread of the process to which the SRB is to be associated, and the PrliOapb field contains the address that must be passed by the SRB in register 2 when issuing a callable service. Note that the StcbOtcb field is zero until the task has been dubbed. The Otcb, Thli, and Prli are mapped by BPXZOTCB, BPXYTHLI, and BPXYPRLI, respectively.

A restriction on the use of callable services by an SRB is that the SRB must be running in non-cross memory mode (primary=secondary=home).

Recovery

The use of callable services from SRB routines requires that the SRB and associated task mode routines must assume responsibility for certain recovery actions. Failure to provide for this can result in unwanted and unpredictable system problems; the system will take a dump. This responsibility revolves around the creation and termination of the process with which the SRB is associated. The process should be created (dubbed) prior to the scheduling of any SRBs that may be associated with it for the purpose of issuing calls. In addition, the SRBs must not be allowed to issue calls after the process has terminated, and the owner of the function taking advantage of SRB mode calls is responsible for guaranteeing that this does not happen. The function must also ensure that it cannot terminate until all of the application-created SRBs have completed processing.

Task and address space-level resource managers can be used to help meet this responsibility. You can use the MVS RESMGR service to set up task and address space resource managers. The following example shows the proper order of processing for the task mode routine, and for the task and address space dynamic resource manager.

Task mode routine responsibilities

- Get the task dubbed by issuing a callable service. The task may already have been dubbed by having been pthread-created.
- Establish a task and an address space dynamic resource manager using the MVS RESMGR service. There are several RESMGR options you can choose when creating a resource manager. It is recommended that you choose to monitor only the address space containing the process, in order to limit system overhead during termination of other address spaces; and that you monitor the top task of the process. Note that the resource manager must be established via the RESMGR service; and that this must be done after the task has been dubbed, or your resource manager will be called after the systems resource manager responsible for process-level termination.
- Schedule one or more SRBs, passing the OAPB address obtained from the PRLI.
- Ensure that all SRBs have completed, and that they will not issue any more callable services.
- Undub or terminate the task.

Task and address space dynamic resource manager

- Terminate any SRBs that have not yet been dispatched via the MVS PURGEDQ service. You can provide filters to this service to purge SRBs selectively; for example, a multiprocess application could use the RMTR address filter to purge only SRBs for the terminating process.
- Wait for already-dispatched SRBs to complete.

For information about scheduling an SRB and SRB processing, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Callable services supported in SRB mode

The following callable services support SRB mode callers. The support of SRB mode callers was intended for the use of sockets from within SRB routines. Some of the following calls support files as well as sockets. These services will only support sockets from an SRB and not file operations. The callable services that are limited are so marked. The results of calling an unsupported callable service are unpredictable.

Note: AF_UNIX sockets do not support SRB mode, only AF_INET and AF_INET6.

The following callable services are supported for 31-bit AMODE SRB callers:

- accept (BPX1ACP)
- accept_and_recv (BPX4ANR)
- asyncio (BPX1AIO)
- bind (BPX1BND)
- bind2addrsel (BPX1BAS)
- close (BPX1CLO) - socket support only
- connect (BPX1CON)
- gethostid & gethostname (BPX1HST)
- getpeername & getsockname (BPX1GNM)
- getsockopt & setsockopt (BPX1OPT)
- listen (BPX1LSN)

- msgsnd (BPX1QSN) - send to a message queue
- pfsctl (BPX1PCT)
- read (BPX1RED) - socket support only
- readv (BPX1RDV) - socket support only
- recv (BPX1RCV)
- recvfrom (BPX1RFM)
- recvmsg (BPX1RMS)
- send (BPX1SND)
- sendmsg (BPX1SMS)
- sendto (BPX1STO)
- server_init (BPX1SIN)
- setpeer (BPX1SPR)
- shutdown (BPX1SHT)
- socket & socket_pair (BPX1SOC)
- srx_np (BPX1SRX)
- w_ioctl (BPX1IOC) - socket support only
- write (BPX1WRT) - socket support only
- writev (BPX1WRV) - socket support only

The following callable services are supported for 64-bit AMODE SRB callers. The support is intended for the use of sockets from within SRB routines. Some of the following calls support files as well as sockets. These services only support sockets from an SRB and do not support file operations. The callable services that are limited are so marked. The results of calling an unsupported callable service are unpredictable.

- asyncio (BPX4AIO)
- accept_and_recv (BPX1ANR)
- read (BPX4RED) - socket support only
- recv (BPX4RCV)
- recvmsg (BPX4RMS)
- sendmsg (BPX4SMS)
- write (BPX4WRT) - socket support only

Appendix K. z/OS UNIX process start/end exits

Four installation exits are defined to enable applications to monitor z/OS UNIX process activity. Exit routines can be added to each exit point. z/OS UNIX passes control to the exit routine when an exit point is reached, and information about the current process and its creator is then passed to the exit routine. These are the installation exits:

Pre-process initiation exit (BPX_PREPROC_INIT)

Pre-process initiation exit routines receive control immediately before the creation of any new z/OS UNIX process. When a pre-process initiation exit routine receives control, the Process Exit Data Block (PEDB) contains the data about the initiating job.

Upon return from the exit, if the exit's return code is greater than 4, the process initiation request will be rejected. The z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPreProcInitExitReject.

The pre-process initiation exit should have a recovery routine to clean up any resources that it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will not be released. Should an exit abend, the z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPreProcInitExitAbend.

The sole purpose of the pre-process initiation exit point is to provide an application with the ability to fail an attempt to initialize a process. If this is not the intent of your exit routine, you should not use this exit point. Do not use this exit point if, for example, your primary purpose is to monitor the initialization and termination of processes in the system, because it does not receive enough information to identify the process that is to be initialized. When it receives control, the only information the exit has available (from the PEDB) is the unique ID and information about the initiator of the process.

Recommendation: Resources should not be obtained at this exit point, because it is possible that another exit routine could subsequently fail the process initialization attempt, and no further exit points would be driven for this process, including the process termination exit. Resources that relate to the process should be obtained in the post-process initialization exit, where the process is fully initialized, and the termination exit will eventually run upon termination of the process.

Rule: This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138–ENQ deadlock.

Post-process initiation exit (BPX_POSPROC_INIT)

Post-process initiation exit routines receive control immediately after the creation of any new z/OS UNIX process. When a post-process initiation exit routine receives control, the Process Exit Data Block (PEDB) contains the creator and the new process data.

The post-process initiation exit should have a recovery routine to clean up any resources that it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will

not be released. Should an exit abend, the z/OS UNIX callable service that drove this process initiation request will fail with a return value of -1, a return code of EAGAIN, and a reason code of JrPosProcInitExitAbend.

Rule: This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138–ENQ deadlock.

Process image initiation exit (BPX_IMAGE_INIT)
Process image initiation exit routines receive control immediately before the initiation of a new z/OS UNIX process image. This occurs when a successful spawn, attach_exec, attach_execmvs, exec or execmvs callable service is done. The process image initiation exit receives control before the new process image file is run. When a process image initiation exit routine receives control, the Process Exit Data Block (PEDB) contains the data of the creator and the new image.

The process image initiation exit should have a recovery routine to clean up any resources it obtained. If the exit does not have a recovery routine, first-failure capture is not possible, and resources that were obtained will not be released. Should an exit abend, the z/OS UNIX callable service that drove this process receives a successful return code, but the image is not created, and an EC6 ABEND with a ImageInitExitABEND reason code is issued.

Rule: This exit should not use any z/OS UNIX callable service. To do so could cause unexpected results, such as ABEND 138–ENQ deadlock.

Pre-process termination exit (BPX_PREPROC_TERM)
Pre-process termination exit routines receive control immediately before the termination of a z/OS UNIX process. These exits may receive control in the address space of the process or in the master address space, if the address space of the process was terminated. In the latter case (ASID=1), z/OS UNIX callable services cannot be used by the exit. When a pre-process termination exit receives control, the Process Exit Data Block (PEDB) contains data about the terminating process.

Exit environment

The user exit receives control in the following environment:

- Supervisor state, key zero.
- Running in the ASID of the process, except for the pre-process termination exit, which runs in the master address space if the address space of the process was terminated.

Register usage:

- On entry to the user exit, register 1 points to the Process Exit Data Block (PEDB).
- For the pre-process initiation exit, if the value returned in register 15 is > 4, the process initiation request is rejected. For all other exit points, the return code in register 15 is ignored.

Errno/errnoJrs

Any callable service that causes a process to be dubbed can receive the following errno/errnojr combinations:

Error	Description
Return code EAGAIN, reason code JrPreProcInitExitReject.	The pre-process initiation exit failed the process initiation request.
Return value -1, return code EAGAIN, reason code JrPreProcInitExitAbend	The pre-process initiation exit ended abnormally.
Return value -1, return code EAGAIN, reason code JrPosProcInitExitAbend	The post-process initiation exit ended abnormally.
EC6 ABEND, reason code ImageInitExitABEND	The process image initiation exit ended abnormally.

Restrictions

1. Process start/end exits cannot use any z/OS UNIX callable services.
2. Exit routines are responsible for cleaning up any resources they obtain (such as storage or locks).
3. Exit routines should have recovery routines to ensure first-failure data capture.

Usage notes

The same exit point can be used for all four exits. The value in the PEDB field PEDBEXITPOINTID identifies the exit point that is hit. For example, If PEDBEXITPOINTID is PEDB_BPX_PREPROC_INIT, the preprocess initiation exit point is hit. The constants that identify each exit point are defined at the bottom of the PEDB (see "BPXYPEDB — Mapping of process exit data block" on page 1005).

See Using installation exits in *z/OS UNIX System Services Planning*for more information about the process start/end installation exits.

Appendix L. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out

punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Acknowledgments

InterOpen/POSIX Shell and Utilities is a source code product providing POSIX.2 (Shell and Utilities) functions to z/OS UNIX System Services. InterOpen/POSIX Shell and Utilities is developed and licensed by Mortice Kern Systems (MKS) Inc. of Waterloo, Ontario, Canada.

Trademarks

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Product Number: 5650-ZOS

Printed in USA

SA23-2281-01