Linux on POWER: Distribution migration and binary compatibility considerations

John Engel

March 24, 2005

Learn about binary compatibility as it relates to the different operating environments that run on Linux® on POWER™. Examine the two Linux on POWER distributions supported by IBM®, Red Hat Enterprise Linux (RHEL) and SUSE LINUX Enterprise Server (SLES), with regard to the binary compatibility between their respective releases. In general, a smooth transition from the 2.4 kernel-based RHEL3 to the 2.6 kernel-based RHEL4 is observed due to the stable Application Binary Interface (ABI) maintained between the releases and the back-porting of many features from RHEL4 into RHEL3. While there are differences in the threading model in the 2.4 kernel-based SLES8 to the 2.6 kernel-based SLES9, binary compatibility can still be maintained between the releases in many cases. Learn about new technologies that can provide performance enhancements for a Linux on POWER application, and follow steps to ensure binary compatibility across multiple distributions in the future.

Introduction

There are many different distributions of Linux available today. While Red Hat and SUSE LINUX are the two vendors for Linux on POWER solutions supported by IBM, other distributions, such as Gentoo and Debian are becoming popular for Linux on POWER as well. Application developers are typically interested in ensuring their application runs on multiple distributions and that their application runs on different versions of a given distribution. These goals can be achieved by understanding the issues that are involved with binary compatibility. This paper defines binary compatibility, discusses considerations that arise when trying to maintain compatibility, and covers the migration between different releases of Red Hat Enterprise Linux, versions 3 and 4 and SUSE LINUX Enterprise Server, versions 8 and 9. It also includes practices that can be followed to ensure compatibility when running an application across multiple distributions.

Table 1 provides information on the levels of software and supported features available in RHEL3, RHEL4, SLES8, and SLES9 that are covered in detail in this paper:

Table 1. Supported features and code levels of RHEL and SLES distributions

<table>
<thead>
<tr>
<th></th>
<th>SLES8 SP4</th>
<th>RHEL3 U4</th>
<th>SLES9</th>
<th>RHEL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>2.4.21</td>
<td>2.4.21</td>
<td>2.6.5</td>
<td>2.6.9</td>
</tr>
</tbody>
</table>
You can use the flowchart in Figure 1 to determine if an application is binary compatible on RHEL or SLES.

**Figure 1. Determining binary compatibility of an application**

Overview of binary compatibility

Linux on POWER binary compatibility is made possible by conforming to an Application Binary Interface (ABI). An ABI is the interface by which a compiled binary gets access to an operating system and its services. When multiple operating environments support the same ABI, it allows for
the same binary to run on these multiple environments. More information on the 64-bit supplement to the PowerPC® Executable and Linking Format (ELF) ABI can be found in the "64-bit PowerPC ELF Application Binary Interface Supplement 1.7" document (IBM). (See Related topics.)

Binary compatibility is the ability to take a binary and run it on multiple environments of a given processor family. These environments could be different versions of the same Linux distribution or they could be completely different distributions. One example includes running a binary that is compiled and run on a POWER4™ processor-based system running SLES9 and executing it on a POWER5™ processor-based system, also running SLES9. Another example is taking a binary that is compiled and run on a POWER4 processor-based system running RHEL3 and executing the same binary on a POWER5 processor-based system running SLES9.

**Processor compatibility**

Processor compatibility is a subject that is closely related to binary compatibility for Linux on POWER. This section covers the compatibility between different 64-bit POWER processors and the compatibility between the 32-bit PowerPC processor and 64-bit POWER processors.

**POWER processor compatibility**

The last example under "Overview of binary compatibility" involved running a binary on two different processor types -- a POWER4 processor and a POWER5 processor. The POWER5 architecture contains enhancements to the POWER4 architecture while maintaining binary compatibility between the two, which allows you to run the same application on both platforms. This same binary can also run on an IBM® eServer™ BladeCenter™ JS20, which contains the PowerPC® 970-based processor. This processor is a single-core version of the POWER4 processor, which maintains binary compatibility with other members of the Power Architecture family.

**PowerPC and POWER processor compatibility**

Binary compatibility between 32-bit applications running in native 32-bit PowerPC environments and 64-bit POWER environments exists. It is also possible to execute 32-bit binaries on native Linux PowerPC environments that were generated on a 64-bit Linux on POWER system. This compatibility is possible due to the following reasons:

- The 64-bit Power Architecture contains support for the complete 32-bit PowerPC architecture.
- The 64-bit Linux kernel contains support to handle 32-bit system calls.
- 32-bit runtime environments contain the necessary 32-bit libraries.
- 64-bit runtime environments contain the necessary 32-bit and 64-bit libraries.

Depending on the development platform, there a few different ways to generate 32-bit and 64-bit Linux binaries:

- The native GNU Compiler Collection (GCC) C compiler on a 32-bit PowerPC platform, such as an Apple Powerbook running Linux, can generate 32-bit binaries that can execute on its native 32-bit platform or on 64-bit Linux on POWER platforms that contain the proper 32-bit user-space libraries.
The IBM XL C/C++, Version 7.0 and GCC C compilers for 64-bit Linux on POWER can generate 32-bit and 64-bit binary executables which can execute on either 32-bit or 64-bit runtime environments. Cross compilers are available for 32-bit PowerPC Linux systems and 64-bit Linux on POWER systems. These cross-compilers can generate both 32-bit and 64-bit binaries. Regardless of where the binary was built, the 32-bit binary can run on either a 32-bit Linux platform or a 64-bit platform, while a 64-bit generated binary can only run on a 64-bit Linux on POWER system. Crosstool is an example of a cross-compiler and can be found under "Downloads" in Dan Kegel's "Building and Testing gcc/glibc cross toolchains" document. (See Related topics.)

Table 2 shows how to generate 32-bit and 64-bit Linux binaries for various development platforms:

Table 2. Generating 32-bit and 64-bit Linux binaries

<table>
<thead>
<tr>
<th>Development platform</th>
<th>Compiler</th>
<th>Generated Linux binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit Linux on PowerPC</td>
<td>Native GCC C compiler</td>
<td>32-bit</td>
</tr>
<tr>
<td>64-bit Linux on POWER</td>
<td>Native XL C/C++ or GCC C compiler</td>
<td>32-bit or 64-bit</td>
</tr>
<tr>
<td>32-bit Linux on PowerPC or 64-bit Linux on POWER</td>
<td>Cross-compiler such as crosstool</td>
<td>32-bit or 64-bit</td>
</tr>
</tbody>
</table>

While the compatibility between 32-bit and 64-bit environments has been demonstrated, this does not mean that distributions officially support this type of compatibility. Red Hat supports 32-bit and 64-bit forward and backward compatibility between RHEL3 and RHEL4, while SLES8 only supports 32-bit forward compatibility when migrating from SLES8 to SLES9.

Table 3 shows the backward and forward compatibility of 32-bit and 64-bit applications on the different releases of RHEL and SLES:

Table 3. 32-bit and 64-bit compatibility in RHEL and SLES distributions

<table>
<thead>
<tr>
<th>Distribution</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL3 &gt; RHEL4</td>
<td>Forward compatible</td>
<td>Forward compatible</td>
</tr>
<tr>
<td>RHEL4 &lt; RHEL3</td>
<td>Backward compatible</td>
<td>Backward compatible</td>
</tr>
<tr>
<td>SLES9 &gt; SLES8</td>
<td>Forward compatible</td>
<td></td>
</tr>
<tr>
<td>SLES8 &lt; SLES9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimizing performance

The 2.6 kernel and the POWER5 architecture contain features that can improve an application's performance. These gains come about because of different libraries, processor features, and updated compilers. Some of the performance gains are immediate without any modification to the application, while other performance gains require a recompile of the source code. It is important to keep in mind that recompiling to gain performance enhancements may compromise binary compatibility on some environments. This section provides some examples of new features of the 2.6 kernel and POWER5 architecture from which an application's performance could benefit.
NUMA

Non-uniform Memory Access (NUMA) is a new feature supported in the 2.6 kernel for Linux on POWER. NUMA addresses the problem that arises when certain processors in a system, depending upon where they are on the bus, take longer to reach certain regions of memory than other processors. NUMA reduces the contention for a system’s shared memory bus by having more memory buses and fewer processors on each bus. While this is less of an issue on a system with just a few processors, it can provide performance improvements when systems have a large number of processors. With the POWER5 architecture being scalable to 64 processors, Linux on POWER can benefit from this new feature in the 2.6 kernel. Most applications will benefit from the 2.6 kernel level NUMA support. Applications seeking additional performance gains can use user-land NUMA API's. RHEL4 ships with user-land NUMA API's, and more information can be found on how to use these API's at the NUMA Group home page. (see Related topics).

Compiler enhancements

You might consider recompiling to take advantage of performance gains from the new features in the latest compilers for Linux on POWER. The high performance compiler IBM XL C/C++ Version 7.0 is available for RHEL3, RHEL4, and SLES9 and adds performance improvements for POWER5 processor-based systems. The -qarch and -qtune options are used to optimize for their respective architectures. For example, to optimize for POWER5, use the following options -qarch=pwr5 and -qtune=pwr5.

The XL C/C++ compiler also contains support for Vector Multimedia Extensions (VMX) on the PowerPC 970 processor using the -qaltivec option.

The GNU Compiler Collection contains compilers for many different languages. Improvements have been made from version 3.2 to 3.3, including POWER5 processor-specific optimizations to its C compiler, gcc. The -mcpu=power5 and -mtune=power5 flags are now supported and result in register usage and instruction scheduling parameters targeted for the POWER5 architecture. There are also VMX vector extensions for the IBM PowerPC 970 processor that can increase the performance of vectorized code. While these optimizations improve performance on their respective architectures, they may compromise binary compatibility for applications running on other platforms.

More information on using the XL C/C++ compiler for Linux on POWER can be found in the developerWorks article "How to use IBM XL C/C++ Advanced Edition V7.0 for Linux on POWER: A guide for GCC users."

SMT

Simultaneous Multi-threading (SMT) provides another possible performance gain when moving to the 2.6 based Linux kernel. SMT is supported on the POWER5 processor and greatly enhances the performance of heavily multi-threaded applications. The POWER5 chip has two hardware instruction threads that are both capable of issuing multiple instructions per cycle, which causes a boost in performance. However, SMT can cause performance degradation in some applications.
that are not heavily threaded. SMT can be disabled by passing `smt-enabled=off` to the Linux kernel when booting.

**Other 2.6 kernel enhancements**

The 2.6 kernel contained in RHEL4 and SLES9 provides these other performance improvements:

- Scalability is improved with the capability of 64 SMP POWER5 processors.
- Large page support is provided for memory intensive applications, allowing 16MB page sizes while still providing the traditional 4KB page sizes.
- PCI hot plug is added in 2.6, which allows a PCI adapter to be added or removed from a running system without having to shutdown the system.
- The virtual memory subsystem has been enhanced with a reverse-mapping algorithm, which provides improvements in memory constrained systems.
- Block I/O and asynchronous I/O are other areas where improvements have been made for resulting performance gains.

**Migrating from RHEL3 to RHEL4**

Red Hat has maintained a stable ABI from RHEL3 to RHEL4, allowing for a smooth transition of applications between releases. However, to ensure binary compatibility, Red Hat recommends that you link your application interfaces to the core libraries they have defined. This list includes:

- libc, libgcc, libstdc++, libdl, libm, libutil, libcrypto, libz, libpthread, libncurses
- libX11, libXext, libXt, libICE, libSM, libGL
- libgtk, libgdk, libgdk_pixmap, libpango, libatk, libglib, libgmodule, libgthread, libgnomeprint, libgnomeprintui, libgconf, libglade

The core libraries and other compatibility issues are described in detail in the “Red Hat Enterprise Linux 4 Application Compatibility” article (see *Related topics*). An application that uses libraries outside of the core set might still maintain compatibility, but further regression testing should be done. Other cases where applications may not retain binary compatible include applications that statically link against any library, depend on kernel level interfaces, or conflict with POSIX standards or the 64-bit POWER ABI definitions.

Not only is the ABI stable between RHEL3 and RHEL4, but many of the 2.6 kernel features found in RHEL4 have also been back-ported to RHEL3. This allows RHEL3 applications already taking advantage of such features as Simultaneous Multi-threading (SMT) and Native Posix Thread Library (NPTL) to gain the performance enhancements included in RHEL4 without having to recompile their source code. These applications will also pick up the performance enhancements that go along with the 2.6 kernel, as described earlier in this paper.

There are, however, two areas in which recompiling on RHEL4 could enhance the performance of an application:

- Systems with a large number of processors may find it advantageous to recompile against the NUMA user-land API’s added in RHEL4. While applications in general will see performance
gains from the kernel level NUMA support, additional gains may be achieved by compiling against these user-land API's. Applications that are processor intensive and access memory frequently will benefit the greatest since NUMA decreases the time it takes a processor to access a region in memory.

- Other applications might also find performance gains when recompiling their applications with the latest compilers available for RHEL4. These compilers have added flags that take advantage of POWER5 optimizations, mentioned previously in this paper.

The risk of compromising binary compatibility must be considered when recompiling your application for performance enhancements.

In most cases, though, you should not need to do additional work with your application when migrating from RHEL3 to RHEL4.

**Migrating from SLES8 to SLES9**

The transition from SLES8 to SLES9 has a few more considerations due to some of the major changes between the releases. Specifically, SLES8 is based on a 2.4.21 kernel with a 2.2.5 version of glibc, while SLES9 is based on a 2.6 kernel with a 2.3 version of glibc. In addition, SLES9 has moved to the NPTL threading model, while SLES8 still uses the older LinuxThreads model. The problems that arise due to the changes in the threading model are described under "Threads model" below. While non-threaded applications that use a smaller subset of common libraries have a better chance of maintaining binary compatibility between SLES releases, implement regression testing to ensure quality.

As with RHEL, there are some circumstances when a recompile of the source may benefit an application that is intended to run on SLES9. For example, an application may gain performance boosts by taking advantage of the enhanced compilers available for SLES9. SMT is also a new feature available in SLES9 that will enhance heavily threaded applications.

In general, most nonthreaded applications will be binary compatible between SLES8 and SLES9. However, with the major changes in libraries and kernel versions, a recompile from source is probably the best option for a developer trying to get optimal performance out of their application.

**Ensuring compatibility across distributions**

Writing an application that is portable across multiple Linux distributions can seem like a difficult task, but it can be achieved by following a few simple practices. The latest releases of most distributions typically contain the same level of libraries and kernel versions. Most distributions also use a similar format for configuration and data files. The following section provides some guidelines to follow when trying to write an application that will be portable across different distributions.

Many Linux distributions include the same ABI and contain a common set of core libraries. However, each distribution's definition of core could vary slightly, which means that regression testing is always necessary when claiming your application is supported on a given distribution.
For example, if you take a closer look at the glibc libraries that are contained in the RHEL and SLES distributions, you see that RHEL3 contains version 2.3.2, SLES9 is at 2.3.3, and RHEL4 ships with 2.3.4, which are all very similar. The differences in the minor revisions are typically bug fixes with no new added feature. RHEL3, RHEL4, and SLES9 also contain similar threading models, so any application that is linked against the common libraries should be able to run on all three of these operating environments. You will also find common libraries in other distributions, such as Gentoo and Debian.

The File Hierarchy Standard (FHS) defines how files and directories should be laid out on common UNIX®-like systems. The FHS should be used if your application relies on other configuration and data files. The main purpose of the FHS is to provide applications a common location to find standard configuration files. More information on the FHS can be found at the Filesystem Hierarchy Standard home page. (See Related topics.)

While a statement that guarantees binary compatibility across all Linux distributions typically cannot be made, you can, however, claim support on most current distributions by following these practices.

**Binary compatibility considerations**

While binary compatibility has been defined with regards to an ABI and processor families, there are other compatibility considerations that must also be addressed when determining if an application will run across multiple environments. A few examples include thread models, low level kernel dependencies, middleware, and core libraries. This section addresses these considerations and how Linux on POWER handles them.

**Thread model**

The threading model in Linux has changed with the release of glibc 2.3 and the transition of the Linux kernel from version 2.4 to version 2.6. The traditional LinuxThreads model used in the glibc 2.2 version and 2.4 kernel has been replaced with the Native Posix Thread Library (NPTL). NPTL was rewritten from scratch and provides significant performance gains for heavily threaded applications. More details on the NPTL specifications can be found in the Red Hat article “The Native POSIX Thread Library for Linux. (See Related topics.)

SLES8 is based on a 2.4 kernel and contains the Linux Thread model, which presents a problem when trying to run a threaded application on SLES9 that contains the NPTL threading model. There are two solutions to this problem:

- Make minor modifications to the source code and recompile against the NPTL based libraries and benefit from the performance enhancements contained in NPTL. More information on migrating from the LinuxThreads model to an NPTL based model can be found in the LinuxDevices.com article "Migrating applications to the 2.6 kernel and NPTL." (See Related topics.)
- Set the **LD_ASSUME_KERNEL** environment variable, which provides backward compatibility to the LinuxThreads model in SLES9. By setting this variable, the linker assumes that a 2.4 kernel
is running along with the older LinuxThreads model. Red Hat developer Ulrich Drepper has a detailed explanation of `LD_ASSUME_KERNEL` (see Related topics).

Red Hat RHEL3 also contains a 2.4 kernel, but it has the NPTL threading support back-ported from the 2.6 kernel. This provides for a smooth transition of threaded applications between the different releases of RHEL.

**Low-level kernel dependencies**

Low-level kernel dependencies are another area of concern when running applications in different operating environments. The movement of information from the /proc file system to the sysfs file system is an example of how an application might not be compatible if it was written against a hard-coded value in one of these file systems. The /proc file system was originally an in-memory file system that allowed user-space access to kernel data structures that contained information, such as system and device status information. This information is being migrated from the /proc file system into the sysfs file system. The /proc file system will still exist, but only contain process information.

The storing of the Universal Serial Bus (USB) device list is an example of where system information has moved from the /proc file system to the sysfs file system. An application that was originally written on a system based on a 2.4 kernel would find this device list in /proc/bus/usb/devices. With the migration to a 2.6 kernel and the sysfs file system, this information has moved to /sys/bus/usb/devices. This movement of information could cause the application to not work properly. While this is not typical, you need to be aware of such potential problems at the kernel level.

**Middleware and application compatibility**

Many applications today are not self-contained, but depend on application middleware. Middleware is the piece of software that lies between two applications or an application and the operating system. Examples of middleware include IBM DB2® Universal Database for Linux, the Java™ Development Kit (JDK), and IBM WebSphere® applications. A developer can determine on which version of middleware their application is supported, but the middleware provider decides on which Linux distributions their product will run. A list of IBM middleware for Linux is available at the IBM software for Linux Web site (see Related topics). Along with middleware, an application may depend on other applications that a distribution ships. Some examples include the Apache web server, database systems, such as mysql and postgresql, and interpreted languages, such as perl and python.

As an example, let's take a closer look at Java and how it could impact an application developer. Java is a common middleware package that receives a lot of focus due to the high number of Java applications that exist because of its platform independence. Not only are there different releases and providers of JDKs, but they also come in 32-bit and 64-bit versions for Linux on POWER. RHEL3 does not ship a JDK, but the IBM Developer Kit for Linux, Java 2 Technology Edition, Version 1.4.2 is supported on it, while RHEL4 ships with both the IBM 32-bit SDK for Linux V1.4.2 and IBM 64-bit SDK for Linux V1.4.2. SLES8 Update4 ships with the IBM 32-bit SDK for
Linux V1.3.1, while SLES9 ships with the 32-bit and 64-bit versions of the IBM SDK V1.4.2. An application developer must be aware of these differences, in case their application depends on features only available in version 1.4.2, which is not available for SLES8.

As another example, the Apache web server is an application that is not considered middleware, but many applications have dependencies on it. Version 2.0 is the latest version of Apache and ships with RHEL3, RHEL4, and SLES9, while SLES8 ships with version 1.3. If an application contains new features that are only available in Apache 2.0, the developer should be aware that their application may not be compatible with version 1.3 of Apache that is shipped with SLES8.

**Core libraries**

Major changes in core libraries can also present a problem when trying to achieve binary compatibility. Backward compatibility is maintained between major changes in libraries. This allows applications compiled against an older version of a given library to run against a new version of the same library. An application compiled against the latest version of a given library will not run against an older version of the same library if there are major changes between the revisions. For example, a binary that was compiled on a SLES8 system with a version of glibc at 2.2.5 will also run on a SLES9 system with a glibc version at 2.3 since the 2.3 version is backward compatible. However, the same source file compiled on a SLES9 system will not run on a SLES8 system because of the older level of glibc. This typically only presents a problem when you develop an application on a current distribution and want to support that application on an older version of the distribution that does not supply older compatible libraries.

**Acknowledgments**

I would like to acknowledge Linda Kinnunen for her document template and helpful reviews and Brent Baude and Matt Davis for their technical assistance and reviews of this document.
Related topics

- Download the IBM Developer Kit for Linux, Java 2 Technology Edition.
- Read the "64-bit PowerPC ELF Application Binary Interface Supplement 1.7" (IBM, 2004) for more information on the 64-bit supplement to the PowerPC® Executable and Linking Format (ELF) ABI.
- Download the Crosstool cross compiler.
- More information about Non-uniform Memory Access can be found at the NUMA Group home page.
- Read "How to use IBM XL C/C++ Advanced Edition V7.0 for Linux on POWER: A guide for GCC users" (developerWorks, Dec 2004) for more information on using the IBM XL C/C++ compiler for Linux on POWER.
- The Filesystem Hierarchy Standard has been designed to be used by Unix distribution developers, package developers, and system implementers.
- Read "The Native POSIX Thread Library for Linux" (Red Hat, Feb 2005) for more details on NPTL specifications.
- Read Ulrich Drepper's detailed explanation of LD_ASSUME_KERNEL.
- The IBM Middleware Available on Linux matrix provides information regarding IBM Middleware availability on Linux.

© Copyright IBM Corporation 2005
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)