CPUMF - Recommendations for Linux® on IBM Z

—

Solution Assurance

André Wild, David Stark

v1.4, March 2022

IBM Z

IBM

Notices and disclaimers

U.S. Government Users Restricted Rights – use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Notices and disclaimers, *continued*

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 about this publication 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. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

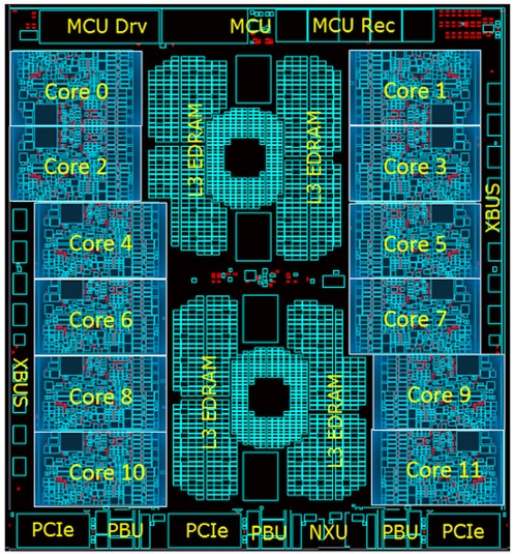
IBM, the IBM logo, ibm.com and IBM Z, IBM z15, and z/VM are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml

Agenda

- CPU Measurement Facility (CPUMF)
  - Introduction
  - Available counter sets
  - How to enable counters and sampling
  - Sampling visualized
  - Management tools
- Monitoring utilities overview
- perf
  - Basic workflow
  - Exploitation overview
  - Collect performance statistics
  - Use flame-graphs
  - Verify hardware exploitation using counters

CPU Measurement Facility
Introduction

On the s390x architecture you can use the CPU Measurement Facility to get insights to your current hardware. These functionality called Performance Monitoring Units (PMU) are located on the cores and the co-processors which allows you to monitor hardware performance directly. This enables you to look into your CPUs and see what happens or happened during the execution of your applications. With this ability you can verify your capacity plan, measure new function exploitation or find hot spots in your application.



z15© CP architecture

Counters

- CPU Cycles | Instruction counter
- Data- and instruction-cache counter
- Cryptography exploitation counter
- Extended counter sets per architecture release

Sampling

- The basic sampling facility takes snapshots of the CPU, the number of CPU cycles and the current state at a specified interval.
- Diagnostic sampling is only required by IBM

CPU Measurement Facility
Available counter sets

To measure system performance in Linux on Z you can choose from the following hardware counter sets. They relate to different kind of topics and therefore measure specific details. You can use this table to get an better overview of what to expect in which counter set.

Counter set	Basic counter set	Problem-State counter set	Crypto-Activity counter set	Extended counter set	MT-diagnostic counter set	Coprocessor group counter set
Topic	CPU Supervisor-State related (kernel- and user-space)	CPU Problem-State related (only user space)	CPU Cryptography instruction related	Depends on hardware generation	Multithreading diagnostic	
Details	• Cycles • Instruction count • L1 instruction cache • L1 data cache	• Cycles • Instruction count	• Counter for RNG | SHA | DES | AES | ECC • Cycles during crypto functions • Function call counter • Stalled/blocked function call counter	• Translation Lookaside Buffer misses/writes • L2 to L1 transitions and requested writes • TX instruction counter • Deflate instruction counter • Binary Coded Decimal (BCD) to decimal floating point conversions counter	• **Always enabled!** • Single thread counter • Multithread counter (only when SMT is enabled)	Not accessible from Linux on Z.

CPU Measurement Facility
How to enable counter and sampling

Before you can utilize the counter sets and sampling facilities you need to activate them first in your LPAR activation profile.

Authorize counter sets

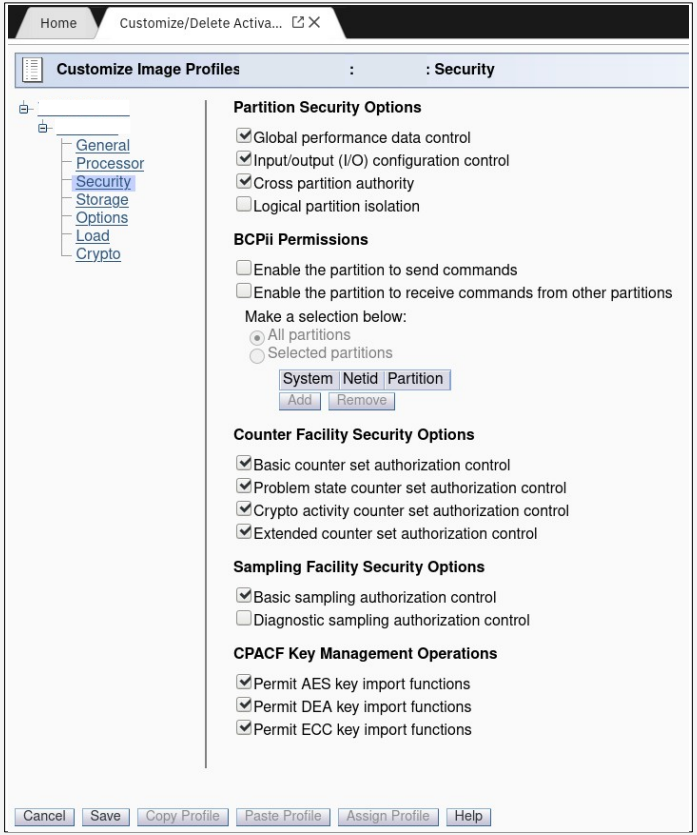
1. Log in to your HMC/SE
2. Choose the **Activation Profile** you want to modify
3. In the "Security" settings look for **Counter Facility Security Options**
   - Authorize each counter set you want to look at.
4. Save
5. Power off and deactivate your LPAR
6. Activate and Initial Program Load (IPL) again

Authorize Sampling

1. Log in to your HMC/SE
2. Choose the **Activation Profile** you want to modify
3. In the "Security" settings look for **Sampling Facility Security Options**
   - Authorize the basic sampling control
4. Save
5. Power off and deactivate your LPAR
6. Activate and Initial Program Load (IPL) again

Verify after IPL

List information about authorized facilities: `# lscpumf -i`
List authorized and available counters: `# lscpumf -c`



CPU Measurement Facility
Management tools

For Linux there are two utilities available. One is to get information about the current available and authorized counter sets and sampling facilities. The other one can be used to change the sampling facility buffer size. Which might be needed if you increase the frequency of your sampling.

lscpumf

Show information about counter and sampling facilities:	**`# lscpumf -i`**
List only available and authorized hardware counters:	**`# lscpumf -c`**
List all hardware counters even if not available:	**`# lscpumf -C`**

chcpumf

Change sampling facility buffer size in Sample-Data-Blocks (SDB)

min buffer size: **`# chcpumf -m SDB_COUNT`**

max buffer size: **`# chcpumf -x SDB_COUNT`**

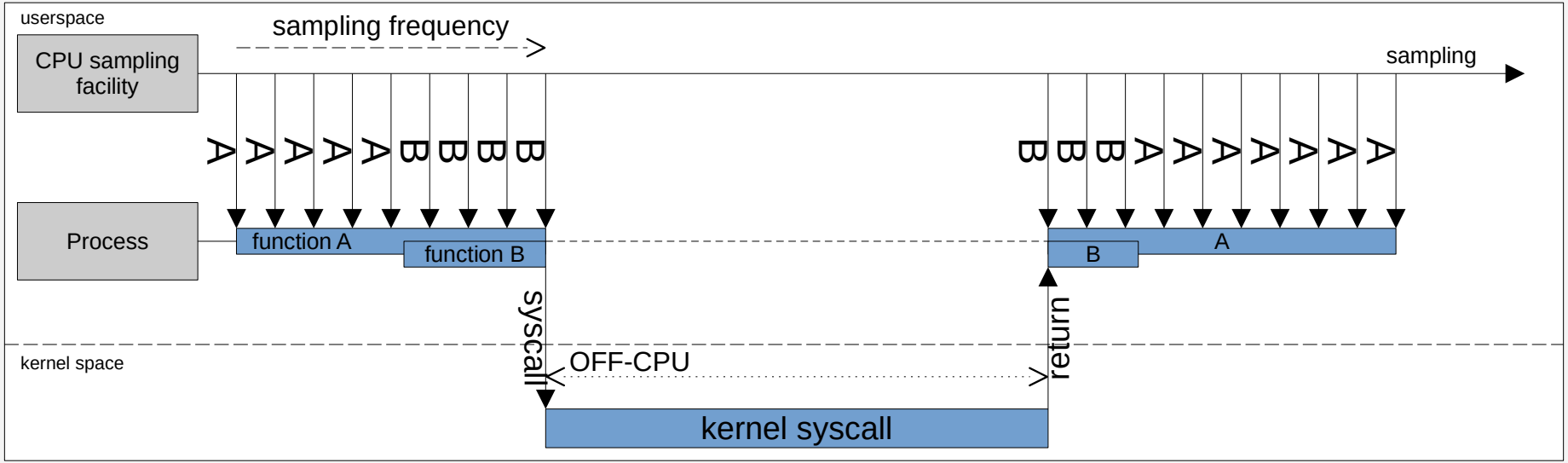
```
root@test:~# lscpumf -i
CPU-measurement Counter Facility
------------------------------------------------------------------------
Version: 3.6

Authorized counter sets:
    Basic counter Set
    Crypto-Activity counter Set
    Extended counter Set
    MT-diagnostic counter Set
    Problem-State counter Set

Linux perf event support: Yes (PMU: cpum_cf)

CPU-measurement Sampling Facility
------------------------------------------------------------------------
Sampling Interval:
     Minimum:      18200 cycles (approx.    285714 Hz)
     Maximum:  170388400 cycles (approx.        30 Hz)

Authorized sampling modes:
     basic:      (sample size:  32 bytes)
     diagnostic: (sample size: 165 bytes)

Linux perf event support: Yes (PMU: cpum_sf)

Current sampling buffer settings for cpum_sf:
    Basic-sampling mode
        Minimum:     15 sample-data-blocks (  64KB)
        Maximum:   8176 sample-data-blocks (  32MB)

    Diagnostic-sampling mode (including basic-sampling)
        Minimum:     90 sample-data-blocks ( 364KB)
        Maximum:  49056 sample-data-blocks ( 192MB)
        Size factor:  6
```

Example output of **`lscpumf -i`**

Red Hat	**`7.X`** `s390-utils`
SUSE	**`12.X`** `s390-utils`
Red Hat	**`8/9`** `s390utils-base`
SUSE	**`15.X`** `s390-tools`

CPU Measurement Facility

Sampling visualized



Sampling

The collected sample data contains data such as PID / TID, instruction address, CPU state and so on. These samples are stored in a ring buffer that consists of several pages and will be collected at an specified period/frequency. Full pages will be marked and generate a measurement alert to consume and create perf samples. Increasing the frequency will produce lager files and more overhead. There are blind spots in the sampling like syscalls or I/O that blocks your application. Time spend on the CPU could be different than the whole execution time.

Sample frequency

Change sampling facility buffer size in Sample-Data-Blocks (SDB):

- increase frequency

```
# perf record -F<frequency> ...
```

- *allow higher frequency*

```
# echo "<frequency>" > /proc/sys/kernel/perf_event_max_sample_rate
```

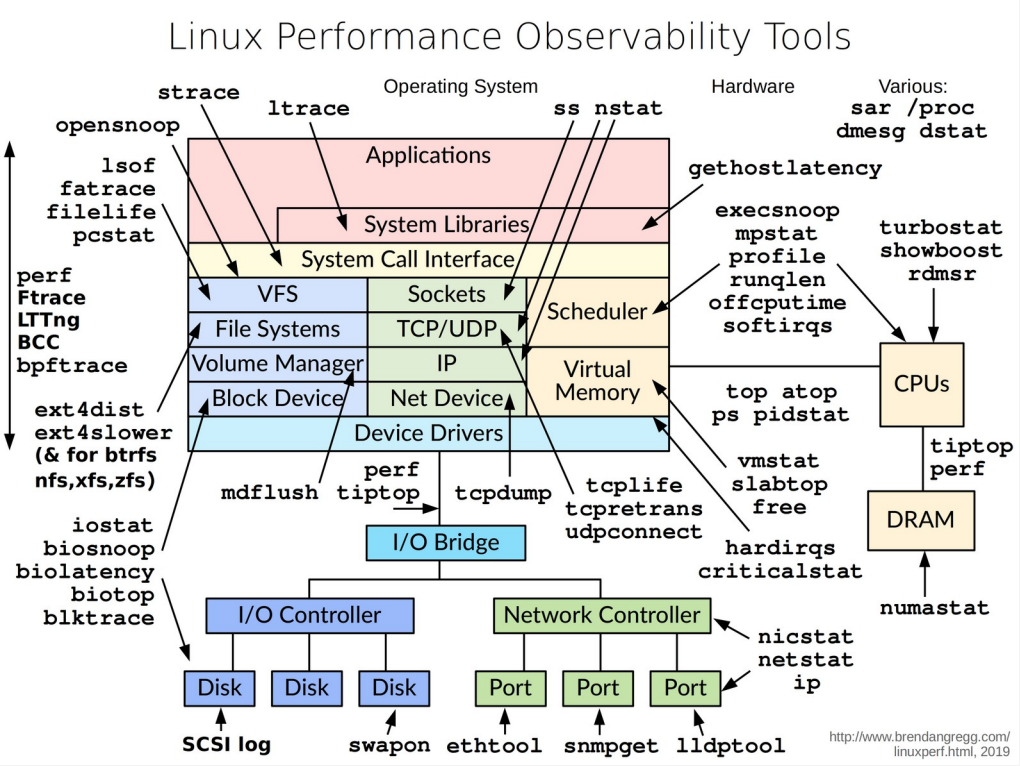
Use with caution - expert architecture knowledge required!

CPU Measurement Facility
Monitoring utilities overview

Utilize perf events

There are tons of utilities out there to monitor and analyze your system behavior under certain conditions. Perf events is an API which is developed within the kernel tree and utilized by the userspace **perf** utility. It's the most common tool which exploits the performance measuring units (PMU), that are exposed by the CPU architecture. With the perf event infrastructure it is able to monitor hardware and tracepoint events. With that ability you can look at the hardware counters but also at predefined locations in the kernel like the scheduler or the I/O controller. For CPACF you have an additional utility called **cpacfstatsd** which monitors the cryptography counter sets and is implemented as a daemon. It uses the libpfm4 library which can be used by external applications to make use of the perf events interface.

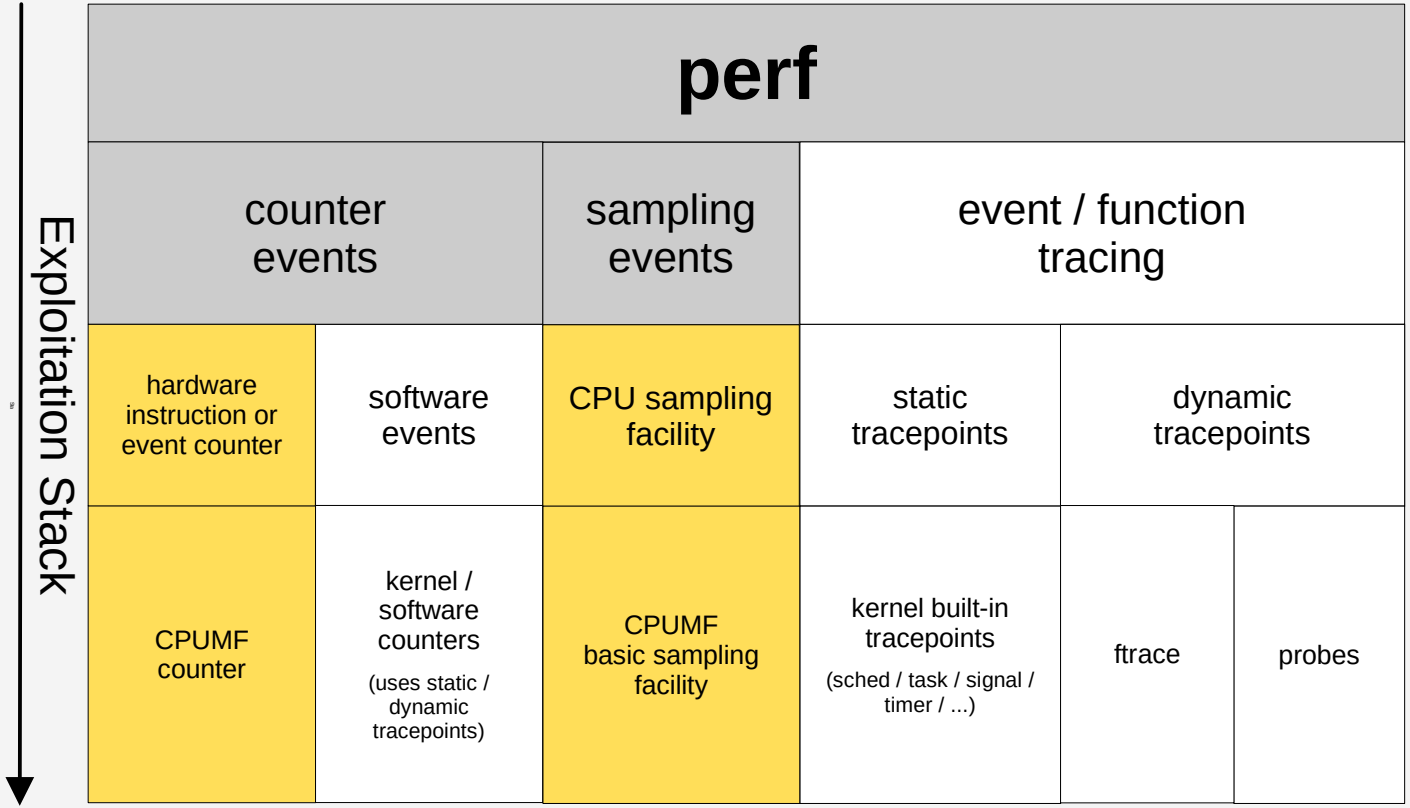
https://perf.wiki.kernel.org



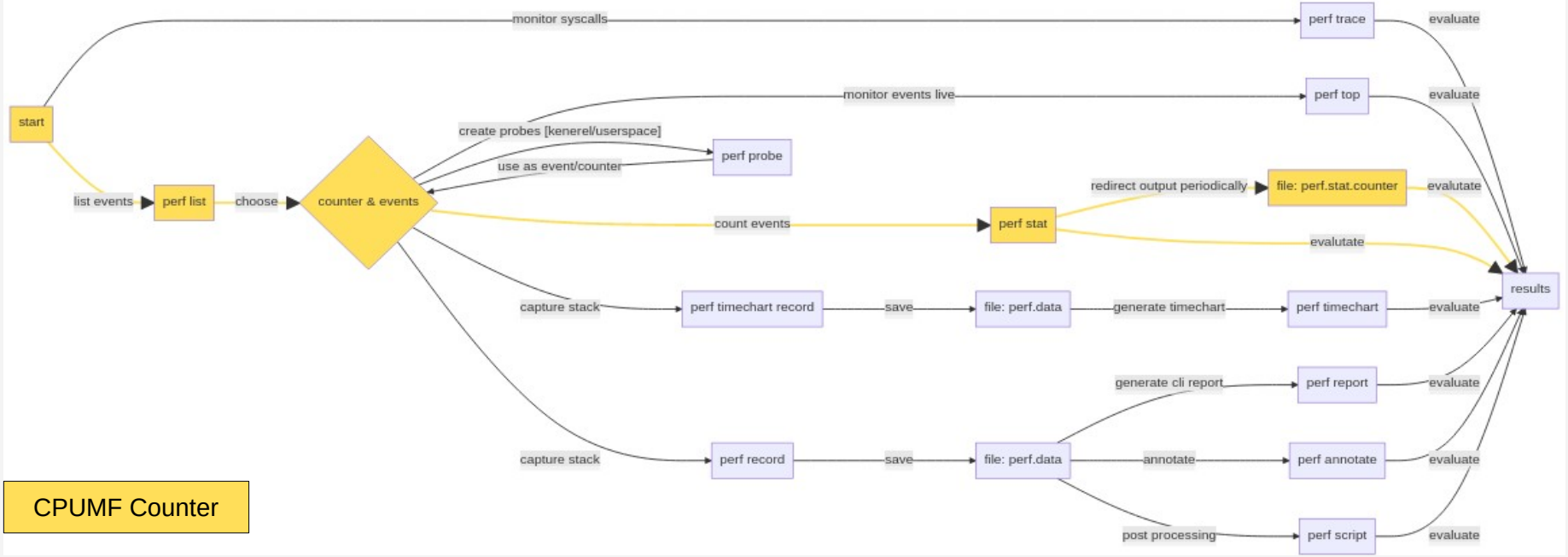
PERF
Exploitation overview

Here you can see the perf event stacks. We will only look at the highlighted CPUMF related topics. That includes the hardware counters and sampling supported by the CPU Measurement Facility.

Visit the resource page at the end to get more information related to that topic.



PERF
Basic workflow



CPUMF Counter

Notes

With this workflow diagram you can get a brief overview on how to utilize perf. It does not cover all of the functionalities provided by perf (see **`perf -h`**), but can be used as a short reference. There are some live analysis commands like top / trace / stat. Additionally it is possible for later investigations to record the system behavior with record / timechart record / trace record and more. Then it is possible to analyze the record later on a different system. Please notice that the recording with perf does not include CPUMF counters.

PERF
Collect performance statistics

Recommendation

In a situation where your system is already under high pressure you may not be able to install the utilities you need to record the current state.

Better be prepared and have your tools already installed!

Get counter subcommands [COUNTER_CMD]

Get all available counter names

```
echo -n "$(lscpumf -c | awk '/^[[:digit:]]/ { print $2 }')" | tr '\n' ','
```

Get counter raw values (includes even undocumented counters)

```
echo {0..5} {32..37} {64..83} {128..287} {448..456} | sed -re "s/([[:digit:]]
+)/$(cat /sys/devices/cpum_cf/type):\1/g" | tr ' ' ','
```

Collect counter information

Get all CPUMF counter at once

```
perf stat -e $(COUNTER_CMD) --pid <pid>
```

Get all CPUMF counter 5 times after an interval of 1000ms

```
perf stat --interval-print 1000 --interval-count 5 -e $(COUNTER_CMD) --pid <pid>
```

Collect default set of events from a running process

```
perf stat --pid <pid>
perf stat --pid <pid> --interval-print 1000 --interval-count 5
```

CSV-style output (-x)

Export data

```
perf stat -x, -- <command-to-monitor> > output.csv
perf stat -x, --pid <pid> --interval-print 1000 --interval-count 5 > output.csv
```

Collect data from a running process

```
root@test:~# perf record -g --pid 170708 -- sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.008 MB perf.data ]
```

Show default events for a running process

```
root@test:~# perf stat --pid 170729
^C
 Performance counter stats for process id '170729':

          6,099.07 msec task-clock                #    0.499 CPUs utilized
               607      context-switches          #    0.100 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                 0      page-faults               #    0.000 K/sec
    31,704,923,453      cycles                    #    5.198 GHz
    18,477,916,454      instructions              #    0.58  insn per cycle
   <not supported>      branches
   <not supported>      branch-misses

      12.229605988 seconds time elapsed
```

Get all CPUMF counters for a running process

```
root@test:~# perf stat -e $(echo -n "$(lscpumf -c | awk '/^[[:digit:]]/ { print $2 }
')" | tr '\n' ',') --pid 170780
^C
 Performance counter stats for process id '170780':

    37,003,086,931      CPU_CYCLES
    22,494,708,774      INSTRUCTIONS
           425,797      L1I_DIR_WRITES
        28,233,017      L1I_PENALTY_CYCLES
       245,624,190      L1D_DIR_WRITES
     2,758,900,823      L1D_PENALTY_CYCLES
    36,977,393,339      PROBLEM_STATE_CPU_CYCLES
    22,489,991,537      PROBLEM_STATE_INSTRUCTIONS
```


PERF
Verify hardware exploitation using counters

Verify hardware exploitation

There are a few counters that reflect the usage of certain CPU supported or assisted functionality. For example deflate and cryptography functions have their own counters. Use them to verify that the workload or application exploits the instructions that are supported by the CPU. Below is an example which verifies the deflate instruction exploitation of gzip.

Instructions

Use perf list to get a list of available performance counter

```
perf list pmu
```

Now use the performance counter and collect statistics

```
perf stat -e DFLT_ACCESS,DFLT_CYCLES,DFLT_CC,DFLT_CCFINISH -- gzip -k <testfile>
```

Record performance counter for later inspection

```
perf stat --interval-print 1000 --interval-count 5 -e
DFLT_ACCESS,DFLT_CYCLES,DFLT_CC,DFLT_CCFINISH -- gzip -k <testfile> >
perf.stat.log
```

No deflate exploitation

```
root@test:~# export DFLTCC=0
root@test:~# perf stat -e DFLT_ACCESS,DFLT_CC,DFLT_CCFINISH,DFLT_CYCLES -- gzip -k perf.data.old

 Performance counter stats for 'gzip -k perf.data.old':

                 0      DFLT_ACCESS
                 0      DFLT_CC
                 0      DFLT_CCFINISH
                 0      DFLT_CYCLES

       0.001129712 seconds time elapsed

       0.000679000 seconds user
       0.000466000 seconds sys
```

vs. deflate exploitation

```
root@test:~# perf stat -e DFLT_ACCESS,DFLT_CC,DFLT_CCFINISH,DFLT_CYCLES -- gzip -k perf.data.old

 Performance counter stats for 'gzip -k perf.data.old':

             1,041      DFLT_ACCESS
                12      DFLT_CC
                 3      DFLT_CCFINISH
            46,398      DFLT_CYCLES

       0.000751212 seconds time elapsed

       0.000160000 seconds user
       0.000613000 seconds sys
```


PERF
Use flame-graphs

Recommendation

With tools like the flame-graph perl script from Brendan Gregg it is easier to visualize the impact of certain code paths Just keep in mind that the recording does not include or reflect CPUMF counters. It's just an really great tool to visualize where time have been spent.

Clone repository

```
# Clone repository and change directory
git clone https://github.com/brendangregg/FlameGraph
cd FlameGraph
```

Instructions to create a flame-graph

Capture perf events with call-graph (-g)

```
perf record -g -- <command-to-monitor>
```

Collapse stacks

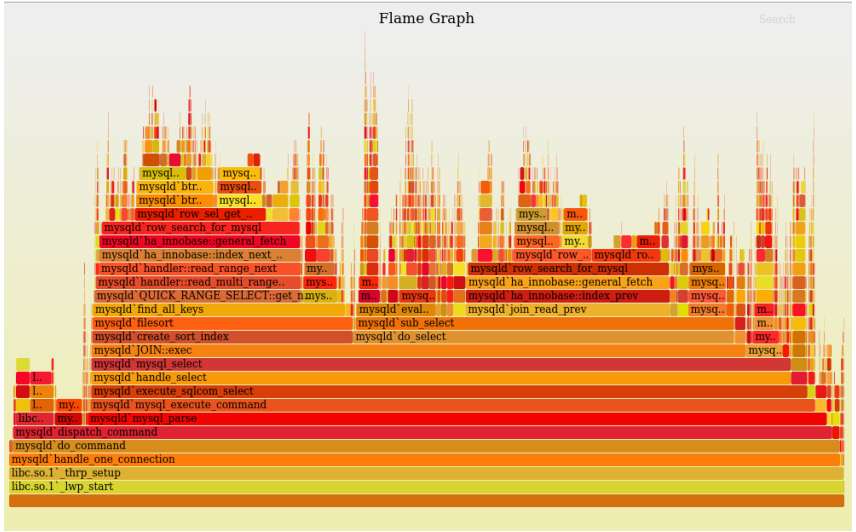
```
perf script | ./stackcollapse-perf.pl > perf.data.collapsed
```

Generate flame-graph

```
./flamegraph.pl perf.data.collapsed > fg_example.svg
```

Use another highlighting (for more see help page)

```
./flamegraph.pl --colors mem perf.data.collapsed > fg_example.svg
```



flame-graph example output from https://github.com/brendangregg/FlameGraph

Open flame-graph

Open the *.svg flame-graph file with the preferred browser (firefox / chrome / …). Within the browser navigate through the stacks and filter them by typing functions or specific names.

Resources

- Linux on IBM Z and IBM LinuxONE
  - Official homepage: http://www.ibm.com/systems/z/os/linux
  - Device Driver Feature and Commands: https://www.ibm.com/docs/linuxonibm/liaaf/lnz_r_dd.html
- Brendan Gregg
  - https://www.brendangregg.com
  - https://github.com/brendangregg/FlameGraph
- Linux Kernel
  - perf wiki: https://perf.wiki.kernel.org
- Logos & Icons
  - https://www.redhat.com/de/about/brand/standards/logo
  - https://brand.suse.com

