Open Source in Industry:
Linux tracing and debugging

Technical Heidelberg OSADL Talks, April 29, 2020, Online Session 2a

Debug and trace interface of the Linux kernel
Function tracing
Event tracing
Latency tracing

Some information on today's sessions

- Please provide feedback on Legal HOT using the online form
  - Use the quick link **osadl.org/FB** (FeedBack), same as osadl.org/?id=3325
- You may ask questions during the session to be answered online, if possible
  - The quick link URL is **osadl.org/AQ** (AskQuestion), same as osadl.org/?id=3321
- You may join an online discussion on all topics of today at 4 pm
  - The quick link URL is **osadl.org/OD** (OnlineDiscussion), same as jitsi.osadl.org
  - Meeting name **OSADLTechnicalHOT**
  - Username and password will be displayed here after the last presentation

(We will show this slide again at the end of this session)

What is „ftrace“?

Initially, “ftrace” was a function tracer, i.e. a log system that could be enabled to record every time a function was called and returned along with names of the calling and the called function and a time stamp.

Today, the term “ftrace” is history, a better word is „tracing“ or „kernel tracing“. It includes a variety of method that are used to understand kernel failures and help fixing them.

The important common functionality is a FIFO that is optimized for speed and combines all tracing messages into a single data stream.

How do we communicate with the tracing interface?

The virtual file system to access kernel tracing is the same as for all other debug subsystems of the kernel (usually automatically mounted):

```
# mount -t debugfs nodev /sys/kernel/debug
```

The interface to the tracing system is localized in the

```
/sys/kernel/debug/tracing
```

directory.

Main functionality of kernel tracing

1. “Classical” function tracer with dynamic function selection
2. Event tracer
3. Tracer of certain critical sections
4. Printk tracer
5. Hardware latency tracer

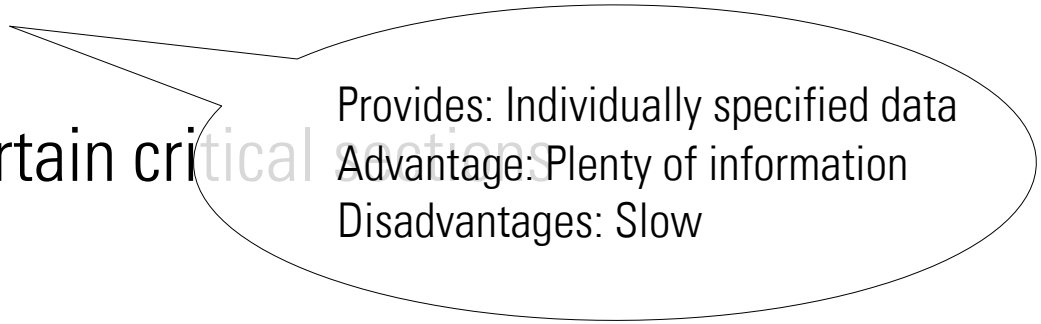
Main functionality of kernel tracing

1. “Classical” function tracer with dynamic function selection
2. Event tracer
3. Tracer of certain critical sections
4. Printk tracer
5. Hardware latency tracer

Provides: Function name and timing
Advantage: Fast
Disadvantage: No arguments

Main functionality of kernel tracing

1. “Classical” function tracer with dynamic function selection
2. Event tracer
3. Tracer of certain critical sections
4. Printk tracer
5. Hardware latency tracer



Main functionality of kernel tracing

1. “Classical” function tracer with dynamic function selection
2. Event tracer
3. Tracer of certain critical sections
4. Printk tracer
5. Hardware latency tracer

Advantage over Syslog:
Much faster and independent
from user-space program

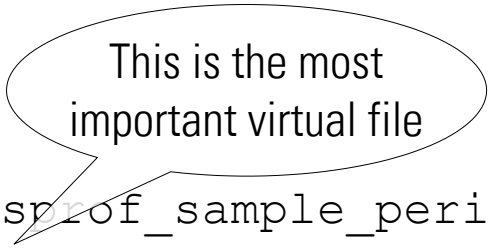
Data exchange with the tracers

```
ls /sys/kernel/debug/tracing
```

available_events	ksym_trace_filter	sysprof_sample_period
available_filter_functions	latency_hist	**trace**
available_tracers	options	trace_clock
buffer_size_kb	**per_cpu**	trace_marker
current_tracer	printk_formats	trace_options
dyn_ftrace_total_info	README	trace_pipe
events	saved_cmdlines	tracing_cpumask
failures	set_event	**tracing_enabled**
kprobe_events	**set_ftrace_filter**	**tracing_max_latency**
kprobe_profile	set_ftrace_notrace	tracing_on
ksym_profile	set_ftrace_pid	tracing_thresh

Data exchange with the tracers

ls /sys/kernel/debug/tracing



available_events	ksym_trace_filter	sysprof_sample_period
available_filter_functions	latency_hist	**trace**
available_tracers	options	trace_clock
buffer_size_kb	**per_cpu**	trace_marker
current_tracer	printk_formats	trace_options
dyn_ftrace_total_info	README	trace_pipe
events	saved_cmdlines	tracing_cpumask
failures	set_event	**tracing_enabled**
kprobe_events	**set_ftrace_filter**	**tracing_max_latency**
kprobe_profile	set_ftrace_notrace	tracing_on
ksym_profile	set_ftrace_pid	tracing_thresh

Read the tracing FIFO

```
cd /sys/kernel/debug/tracing
```

All CPUs:

```
cat trace >/tmp/trace.txt
```

A defined CPU only, e.g. core #0 :

```
cat per_cpu/cpu0/trace >/tmp/trace-cpu0.txt
```

Function tracer

Test whether function tracer is available:

```
grep function >available_tracers
```

Enable function tracer:

```
echo function >current_tracer
```

Enable tracer only for selected functions:

```
echo <[*]function[*]> >set_ftrace_filter
```

For example: `echo sys_* >set_ftrace_filter`

Stop tracing:

```
echo 0 >tracing_enabled
```

Function tracer

Test whether function tracer is available:

```
# grep function available_tracers
hwlat blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
```

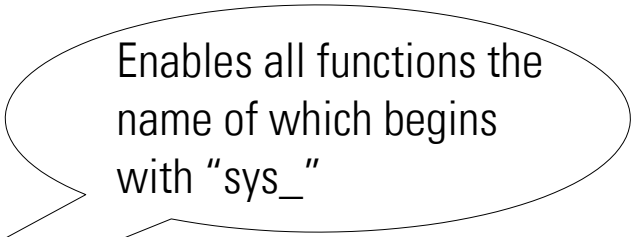
Enable function tracer:

```
# echo function >current_tracer
```

Enable tracer only for selected functions:

```
# echo <[*]function[*]> >set_ftrace_filter
```

For example: `echo sys_* >set_ftrace_filter`



Stop tracing:

```
# echo 0 >tracing_enabled
```

Function tracer example (nop=disabled)

```
# cat trace
# tracer: nop
#
# entries-in-buffer/entries-written: 0/0   #P:32
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
```

Function tracer example (enabled)

```
# echo function >current_tracer; cat trace | head -13; echo nop >current_tracer
# tracer: function
#
# entries-in-buffer/entries-written: 1004455/1004455   #P:32
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          Timer-4636  [028] d... 139138.257037: do_syscall_64 <-entry_SYSCALL_64_after_hwframe
          <idle>-0    [029] d... 139138.257037: pm_qos_read_value <-cpuidle_governor_latency_req
          <idle>-0    [013] .... 139138.257037: sched_idle_set_state <-cpuidle_enter_state
          <idle>-0    [010] d... 139138.257038: tick_nohz_get_sleep_length <-menu_select
          <idle>-0    [009] d... 139138.257038: tick_check_broadcast_expired <-do_idle
          <idle>-0    [025] .... 139138.257038: sched_idle_set_state <-cpuidle_enter_state
          <idle>-0    [026] .... 139138.257038: sched_idle_set_state <-cpuidle_enter_state
```

Event tracer

Enable individual events:

```
# echo 1 >events/sched/sched_wakeup/enable
# echo 1 >events/sched/sched_wakeup_new/enable
```

Enable event group:

```
# echo 1 >events/sched/enable
```

Example:

```
# tracer: nop
#
# entries-in-buffer/entries-written: 1040906/12523426   #P:32
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [018] dNh. 139991.935902: sched_wakeup: comm=cyclictest pid=44772 prio=0
          <idle>-0     [018] dNh. 139991.936101: sched_wakeup: comm=cyclictest pid=44772 prio=0
          <idle>-0     [018] dNh. 139991.936302: sched_wakeup: comm=cyclictest pid=44772 prio=0
```

Printk tracer

Insert into kernel code:

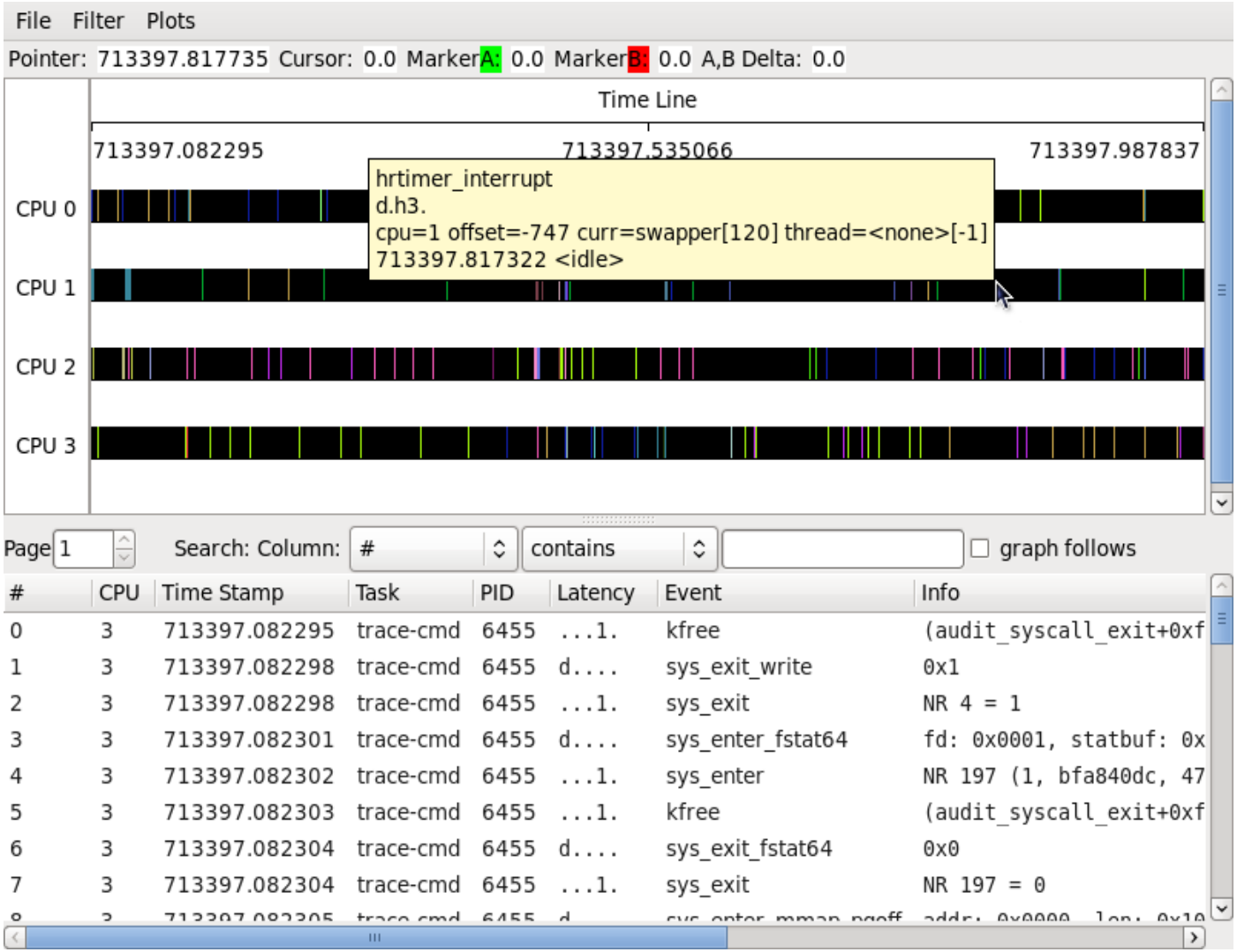
```
trace_printk(...);
```

Command line interface trace-cmd

For example: Enable all events:

```
trace-cmd record -e all
disable all
enable all
Hit Ctrl^C to stop recording
^Coffset=2aa000
offset=4d8000
offset=69c000
offset=8be000
Kernel buffer statistics: [..]
```

GUI kernelshark



Performance tool „perf“

A single tool (*perf*) incorporates a number of various different funktions (similar to *git*):

```
cd tools/perf
make
make install

perf
The most commonly used perf commands are:
   annotate    Read perf.data (created by perf record) and display annotated code
   list        List all symbolic event types
   record      Run a command and record its profile into perf.data
   report      Read perf.data (created by perf record) and display the profile
   stat        Run a command and gather performance counter statistics
   top         System profiling tool.
```

perf top

```
Samples: 1M of event 'cycles', 4000 Hz, Event count (approx.): 94984473493 lost: 0/0 drop: 0/0
Overhead  Shared Object                     Symbol
  16.93%  [kernel]                          [k] profile_graph_entry
  10.40%  [kernel]                          [k] native_sched_clock
  10.28%  [kernel]                          [k] profile_graph_return
   7.58%  [kernel]                          [k] queued_spin_lock_slowpath
   4.79%  [kernel]                          [k] update_blocked_averages
   4.12%  [kernel]                          [k] try_to_wake_up
   4.12%  [kernel]                          [k] __update_load_avg_cfs_rq
   3.83%  [kernel]                          [k] __x86_indirect_thunk_rax
   3.80%  [kernel]                          [k] return_to_handler
   2.72%  [kernel]                          [k] function_graph_enter
   2.54%  [kernel]                          [k] ftrace_return_to_handler
   1.66%  [kernel]                          [k] ftrace_graph_caller
   1.55%  [kernel]                          [k] __list_del_entry_valid
   1.37%  [kernel]                          [k] acpi_idle_do_entry
   1.03%  [kernel]                          [k] prepare_ftrace_return
   0.66%  [ttm]                             [k] ttm_bo_add_to_lru
   0.48%  [kernel]                          [k] update_sd_lb_stats
   0.46%  [amdgpu]                          [k] amdgpu_vm_move_to_lru_tail
   0.38%  perf                              [.] hpp__sort_overhead
   0.36%  perf                              [.] rb_next
   0.33%  libc-2.28.so                      [.] __strcmp_avx2
   0.32%  [kernel]                          [k] ftrace_graph_get_ret_stack
   0.28%  [kernel]                          [k] update_nohz_stats
   0.25%  [kernel]                          [k] smp_call_function_single
   0.24%  [kernel]                          [k] ftrace_graph_is_dead
```

perf stat

Run a program and inspect the performance counters:

```
# perf stat sleep 1

 Performance counter stats for 'sleep 1':

              3.82 msec task-clock                #    0.004 CPUs utilized
                25      context-switches          #    0.007 M/sec
                 0      cpu-migrations            #    0.000 K/sec
                63      page-faults               #    0.016 M/sec
        13,252,477      cycles                    #    3.465 GHz                      (78.71%)
         2,779,375      stalled-cycles-frontend   #   20.97% frontend cycles idle     (78.60%)
         1,109,240      stalled-cycles-backend    #    8.37% backend cycles idle      (80.42%)
         7,626,038      instructions              #    0.58  insn per cycle
                                                  #    0.36  stalled cycles per insn  (78.44%)
         1,721,232      branches                  #  449.999 M/sec                    (99.24%)
            54,843      branch-misses             #    3.19% of all branches          (84.60%)

       1.007599675 seconds time elapsed

       0.000764000 seconds user
       0.005242000 seconds sys
```

perf stat (example busy loop)

Run a busy loop and inspect the performance counters under various real-time conditions:

```
int main(int argc, char *argv[])
{
  int cycles = 100000000;
  while (cycles--) ;
}
```

perf stat (example busy loop): no real-time

Run cyclictest in background, run a busy loop and inspect the performance counters (no real-time):

```
# perf stat ./busyloop

 Performance counter stats for './busyloop':

            213.97 msec task-clock                #    0.931 CPUs utilized
             1,381      context-switches          # 6483.568 M/sec
                 0      cpu-migrations            #    0.000 K/sec
                49      page-faults               #  230.047 M/sec
       728,795,223      cycles                    # 3421573.817 GHz                    (83.01%)
       521,429,641      stalled-cycles-frontend   #   71.55% frontend cycles idle     (83.35%)
       235,074,710      stalled-cycles-backend    #   32.26% backend cycles idle      (66.86%)
       512,846,836      instructions              #    0.70  insn per cycle
                                                  #    1.02  stalled cycles per insn  (83.41%)
       103,153,124      branches                  # 484286967.136 M/sec                (83.45%)
           144,844      branch-misses             #    0.14% of all branches          (83.33%)

       0.229856332 seconds time elapsed

       0.218521000 seconds user
       0.000954000 seconds sys
```

perf stat (real-time, no affinity)

Run cyclictest in background, run a busy loop and inspect the performance counters (real-time, no affinity):

```
# perf stat chrt -f 90 ./busyloop

 Performance counter stats for 'chrt -f 90 ./busyloop':

            213.82 msec task-clock                #    0.889 CPUs utilized
             2,397      context-switches          # 11253.521 M/sec
             2,392      cpu-migrations            # 11230.047 M/sec
               112      page-faults               #  525.822 M/sec
       723,177,255      cycles                    # 3395198.380 GHz                    (83.34%)
       508,838,060      stalled-cycles-frontend   #   70.36% frontend cycles idle      (83.36%)
       226,652,203      stalled-cycles-backend    #   31.34% backend cycles idle       (66.69%)
       524,733,993      instructions              #    0.73  insn per cycle
                                                  #    0.97  stalled cycles per insn  (83.33%)
       106,182,724      branches                  # 498510441.315 M/sec                (83.36%)
           287,257      branch-misses             #    0.27% of all branches         (83.25%)

       0.240628157 seconds time elapsed

       0.223934000 seconds user
       0.000000000 seconds sys
```

perf stat (real-time, affinity)

Run cyclictest in background, run a busy loop and inspect the performance counters (no real-time, affinity):

```
# perf stat taskset -c 1 chrt -f 90 ./busyloop

 Performance counter stats for 'taskset -c 1 chrt -f 90 ./busyloop':

            209.14 msec task-clock                #    0.944 CPUs utilized
             1,108      context-switches          # 5301.435 M/sec
                 1      cpu-migrations            #    4.785 M/sec
               182      page-faults               #  870.813 M/sec
       713,360,124      cycles                    # 3413206.335 GHz                    (83.28%)
       506,002,912      stalled-cycles-frontend   #   70.93% frontend cycles idle      (83.29%)
       107,863,334      stalled-cycles-backend    #   15.12% backend cycles idle       (66.51%)
       513,113,816      instructions              #    0.72  insn per cycle
                                                  #    0.99  stalled cycles per insn  (83.24%)
       103,119,288      branches                  # 493393722.488 M/sec                (83.29%)
           137,694      branch-misses             #    0.13% of all branches          (83.63%)

       0.221539588 seconds time elapsed

       0.212750000 seconds user
       0.000967000 seconds sys
```

perf stat (high prio real-time, affinity)

Run cyclictest in background, run a busy loop and inspect the performance counters (no real-time with same priority as cyclictest, affinity):

```
# perf stat taskset -c 1 chrt -f 99 ./busyloop

 Performance counter stats for 'taskset -c 1 chrt -f 99 ./busyloop':

            215.96 msec task-clock                #    0.998 CPUs utilized
                 9      context-switches          #   41.860 M/sec
                 1      cpu-migrations            #    4.651 M/sec
               183      page-faults               #  851.163 M/sec
       742,475,874      cycles                    # 3453376.158 GHz                    (83.38%)
       539,580,837      stalled-cycles-frontend   #   72.67% frontend cycles idle      (83.34%)
       113,773,162      stalled-cycles-backend    #   15.32% backend cycles idle       (66.67%)
       504,496,302      instructions              #    0.68  insn per cycle
                                                  #    1.07  stalled cycles per insn  (83.33%)
       101,077,428      branches                  # 470127572.093 M/sec                (83.33%)
            38,413      branch-misses             #    0.04% of all branches          (83.28%)

       0.216494299 seconds time elapsed

       0.214388000 seconds user
       0.002003000 seconds sys
```

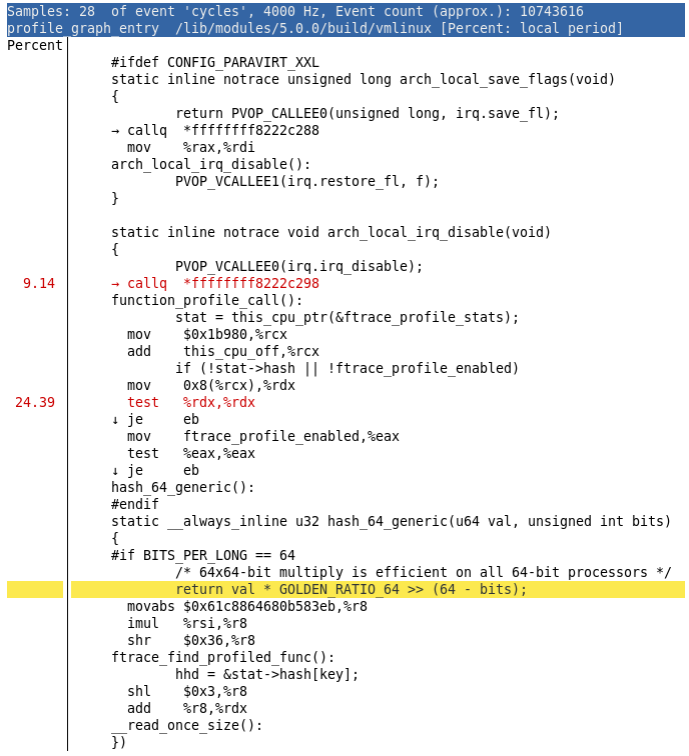
perf record/annotate

Data capturing:

```
# perf record sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.015 MB perf.data (28 samples) ]
```

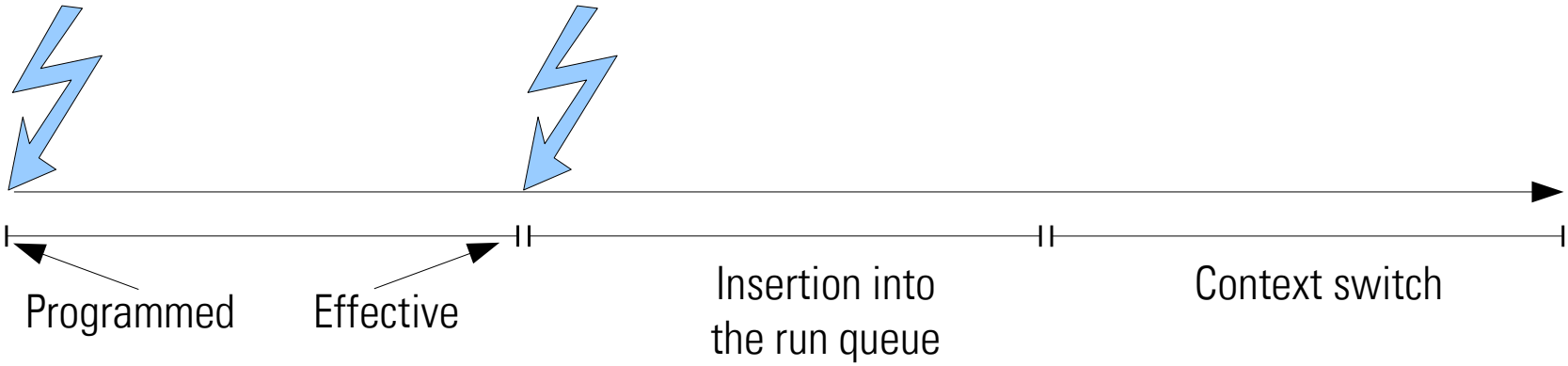
Data analysis:

```
perf annotate
```



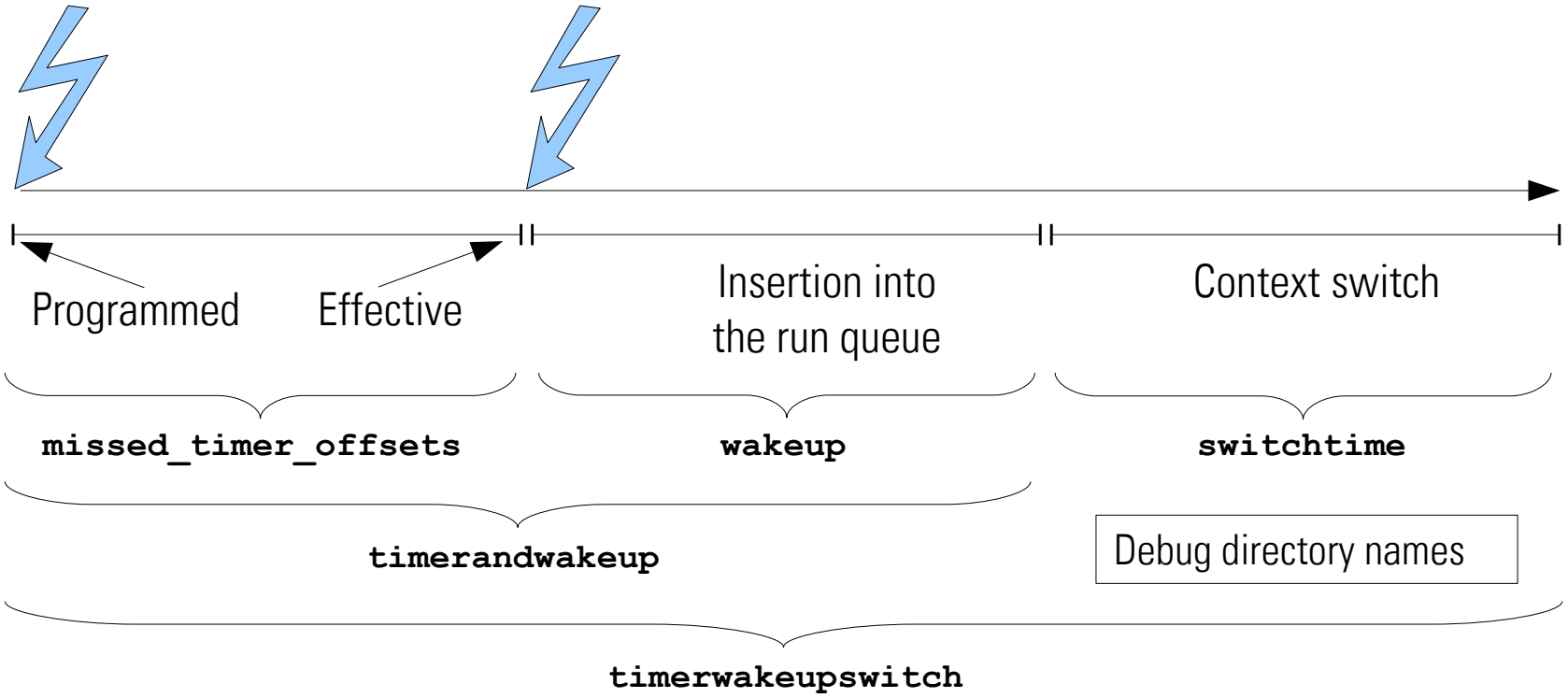
Internal recording of effective latencies, sections

Restarting a waiting application by timer expiration



Internal recording of effective latencies, variables

Restarting a waiting application by timer expiration



Internal recording of effective latencies, access

Kernel configuration

```
CONFIG_WAKEUP_LATENCY_HIST=y
CONFIG_MISSED_TIMER_OFFSET_HIST=y
CONFIG_SWITCHTIME_HIST=y
```

Directory

```
/sys/kernel/debug/latency_hist
```

Directory in earlier kernel versions

```
/sys/kernel/debug/tracing/latency_hist
```

Important subdirectories

```
/sys/kernel/debug/latency_hist/enable
/sys/kernel/debug/latency_hist/wakeup
/sys/kernel/debug/latency_hist/missed_timer_offsets
/sys/kernel/debug/latency_hist/timerandwakeup
/sys/kernel/debug/latency_hist/switchtime
/sys/kernel/debug/latency_hist/timerwakeupswitch
```

Access via virtual debug filesystem

Single command

```
mount -t debugfs nodev /sys/kernel/debug
```

Permanent configuration in `/etc/fstab`

```
nodev /sys/kernel/debug debugfs defaults 0 0
```

Internal recording of effective latencies, management

Files

Enable internal recording of effective latencies

```
echo 1 >/sys/kernel/debug/latency_hist/enable/wakeup
echo 1 >/sys/kernel/debug/latency_hist/enable/missed_timer_offsets
echo 1 >/sys/kernel/debug/latency_hist/enable/timerandwakeup
echo 1 >/sys/kernel/debug/latency_hist/enable/switchtime
echo 1 >/sys/kernel/debug/latency_hist/enable/timerwakeupswitch
```

Histograms of latency data

```
/sys/kernel/debug/latency_hist/wakeup/CPU*
/sys/kernel/debug/latency_hist/missed_timer_offsets/CPU*
/sys/kernel/debug/latency_hist/timerandwakeup/CPU*
/sys/kernel/debug/latency_hist/switchtime/CPU*
/sys/kernel/debug/latency_hist/timerwakeupswitch/CPU*
```

Histograms of latency data

Data

```
grep -v " 0$" /sys/kernel/debug/latency_hist/timerwakeupswitch/CPU0
#Minimum latency: 0 microseconds
#Average latency: 0 microseconds
#Maximum latency: 40 microseconds
#Total samples: 1457599
#There are 0 samples greater or equal than 10240 microseconds.
#usecs              samples
     0              1452538
     1                 3323
     2                 1676
     3                   11
     4                    3
     5                   17
     6                   10
     7                    5
     8                    2
```

Hints to culprit and victim

Files

Enable internal recording of effective latencies

```
echo 1 >/sys/kernel/debug/latency_hist/enable/wakeup
echo 1 >/sys/kernel/debug/latency_hist/enable/missed_timer_offsets
echo 1 >/sys/kernel/debug/latency_hist/enable/timerandwakeup
echo 1 >/sys/kernel/debug/latency_hist/enable/switchtime
echo 1 >/sys/kernel/debug/latency_hist/enable/timerwakeupswitch
```

Histograms of latency data

```
/sys/kernel/debug/latency_hist/wakeup/CPU*
/sys/kernel/debug/latency_hist/missed_timer_offsets/CPU*
/sys/kernel/debug/latency_hist/timerandwakeup/CPU*
/sys/kernel/debug/latency_hist/switchtime/CPU*
/sys/kernel/debug/latency_hist/timerwakeupswitch/CPU*
```

Hints to culprit and victim in case of a prolonged latency

```
/sys/kernel/debug/latency_hist/wakeup/max_latency-CPU*
/sys/kernel/debug/latency_hist/missed_timer_offsets/max_latency-CPU*
/sys/kernel/debug/latency_hist/timerandwakeup/max_latency-CPU*
/sys/kernel/debug/latency_hist/switchtime/max_latency-CPU*
/sys/kernel/debug/latency_hist/timerwakeupswitch/max_latency-CPU*
```

Hints to culprit and victim in case of a prolonged latency

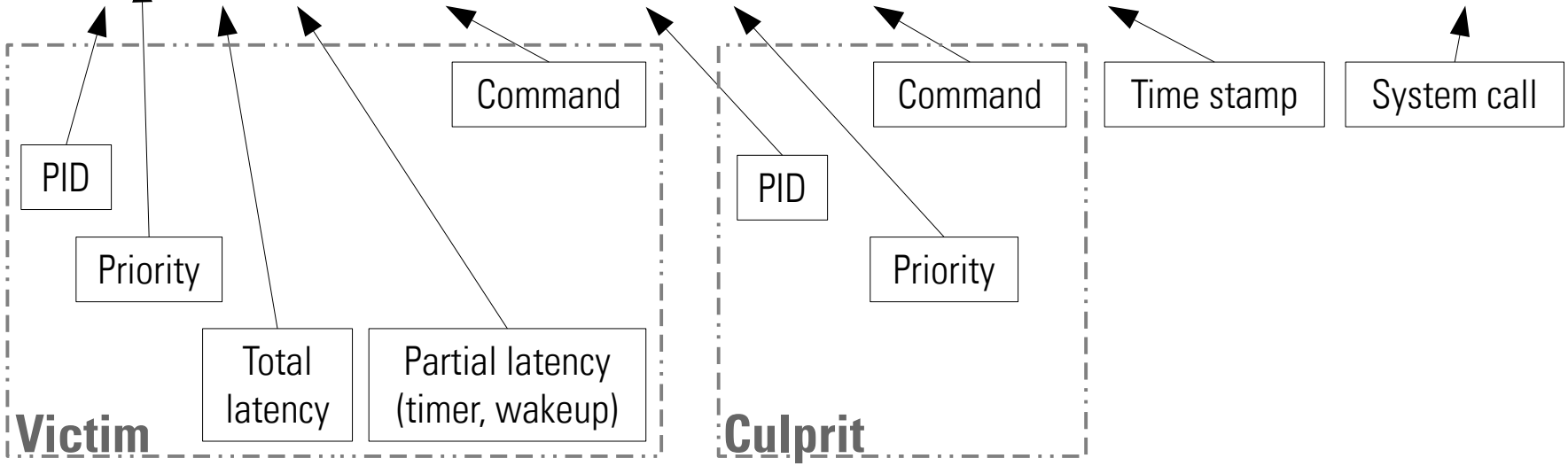
Characteristic data of the highest scheduling latency since the most recent reset (reset occurs every 5 minutes at the OSADL QA Farm):

```
# cat /sys/kernel/debug/latency_hist/timerwakeupswitch/max_latency-CPU0
23579 99 8 (7,0) cyclictest <- 22176 -19 bandwidth64 3522647.050475 __x64_sys_clock_nanosleep
```

Hints to culprit and victim in case of a prolonged latency

Characteristic data of the highest scheduling latency since the most recent reset (reset occurs every 5 minutes at the OSADL QA Farm):

```
# cat /sys/kernel/debug/latency_hist/timerwakeupswitch/max_latency-CPU0
23579 99 8 (7,0) cyclictest <- 22176 -19 bandwidth64 3522647.050475 __x64_sys_clock_nanosleep
```



Handle histograms - Reset

Reset

```
#!/bin/bash

HISTDIR=/sys/kernel/debug/latency_hist

if test -d $HISTDIR
then
  cd $HISTDIR
  for i in `find . | grep /reset$`
  do
    echo 1 >$i
  done
fi
```

Calibration of latency recording (1)

“Bad” driver (blocksys.ko)

```
local_irq_disable();
while (nops--)
  asm("nop");
local_irq_enable();
```

Using the “bad” driver (mklatency)

Command

```
./mklatency
```

Or

```
echo 1000000 >/dev/blocksys
```

Kernel log

```
# dmesg | tail -2
[231234.857241] blocksys: preemption and interrupts of CPU #6 will be blocked for 1000000 nops
[231234.876478] blocksys: preemption and interrupts of CPU #6 blocked for about 2146 us
```

Culprit/victim output

```
# cat max_latency-CPU6
4122437 99 2087 (2081,5) cyclictest <- 4122293 -21 bash 231235.023676 __x64_sys_clock_nanosleep
```

Calibration of latency recording (2)

Output of cyclictest

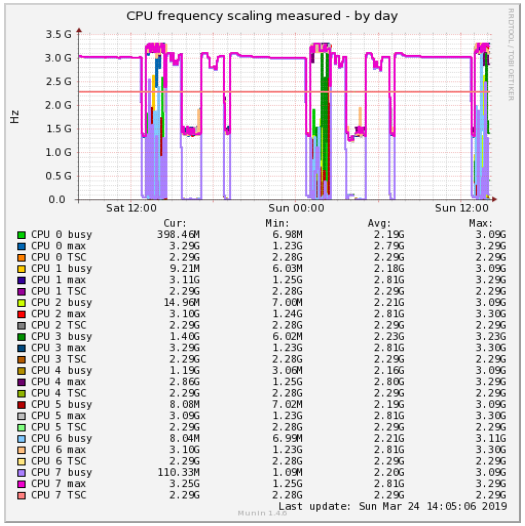
```
# cyclictest -m -n -Sp90 -i100 -d0
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 10.43 6.56 3.38 2/1454 4126098

T: 0 (4122431) P:99 I:100 C:5154828 Min:      3 Act:    4 Avg:    6 Max:       42
T: 1 (4122432) P:99 I:100 C:5154687 Min:      3 Act:    4 Avg:    5 Max:       88
T: 2 (4122433) P:99 I:100 C:5154561 Min:      3 Act:    4 Avg:    5 Max:       40
T: 3 (4122434) P:99 I:100 C:5154439 Min:      3 Act:    7 Avg:    6 Max:       40
T: 4 (4122435) P:99 I:100 C:5154318 Min:      3 Act:    4 Avg:    6 Max:       31
T: 5 (4122436) P:99 I:100 C:5154196 Min:      3 Act:    5 Avg:    5 Max:       47
T: 6 (4122437) P:99 I:100 C:5153993 Min:      3 Act:    4 Avg:    6 Max:     2091
T: 7 (4122438) P:99 I:100 C:5153936 Min:      3 Act:    4 Avg:    5 Max:       94
T: 8 (4122439) P:99 I:100 C:5153807 Min:      3 Act:    4 Avg:    5 Max:       39
T: 9 (4122440) P:99 I:100 C:5153662 Min:      3 Act:    5 Avg:    5 Max:       51
T:10 (4122441) P:99 I:100 C:5153517 Min:      3 Act:    5 Avg:    5 Max:       42
T:11 (4122442) P:99 I:100 C:5153371 Min:      3 Act:    4 Avg:    5 Max:       30
```

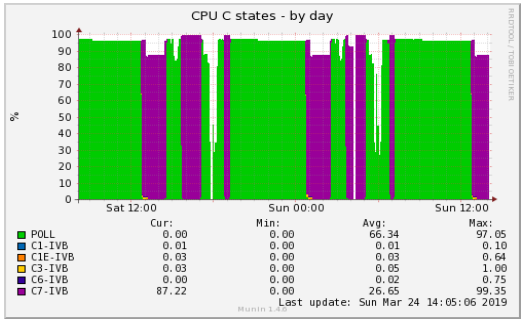
Calibration of latency recording (2)

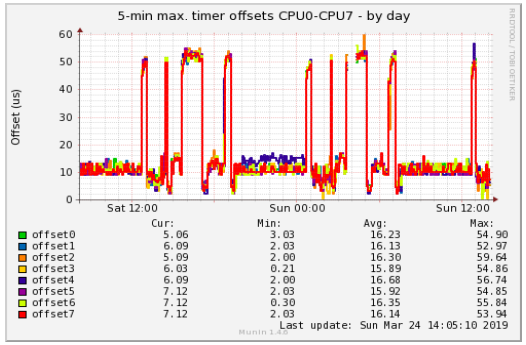
Output of cyclictest

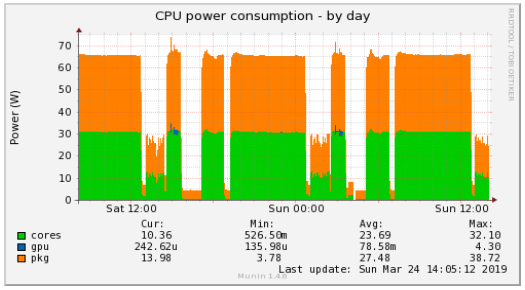
```
# cyclictest -m -n -Sp90 -i100 -d0
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 10.43 6.56 3.38 2/1454 4126098

T: 0 (4122431) P:99 I:100 C:5154828 Min:      3 Act:    4 Avg:    6 Max:       42
T: 1 (4122432) P:99 I:100 C:5154687 Min:      3 Act:    4 Avg:    5 Max:       88
T: 2 (4122433) P:99 I:100 C:5154561 Min:      3 Act:    4 Avg:    5 Max:       40
T: 3 (4122434) P:99 I:100 C:5154439 Min:      3 Act:    7 Avg:    6 Max:       40
T: 4 (4122435) P:99 I:100 C:5154318 Min:      3 Act:    4 Avg:    6 Max:       31
T: 5 (4122436) P:99 I:100 C:5154196 Min:      3 Act:    5 Avg:    5 Max:       47
T: 6 (4122437) P:99 I:100 C:5153993 Min:      3 Act:    4 Avg:    6 Max:     2091
T: 7 (4122438) P:99 I:100 C:5153936 Min:      3 Act:    4 Avg:    5 Max:       94
T: 8 (4122439) P:99 I:100 C:5153807 Min:      3 Act:    4 Avg:    5 Max:       39
T: 9 (4122440) P:99 I:100 C:5153662 Min:      3 Act:    5 Avg:    5 Max:       51
T:10 (4122441) P:99 I:100 C:5153517 Min:      3 Act:    5 Avg:    5 Max:       42
T:11 (4122442) P:99 I:100 C:5153371 Min:      3 Act:    4 Avg:    5 Max:       30
```

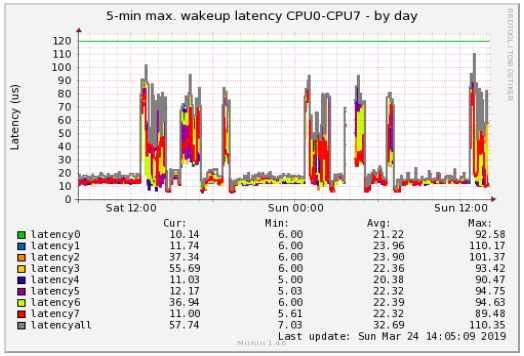
Continuous recording of real-time related system variables





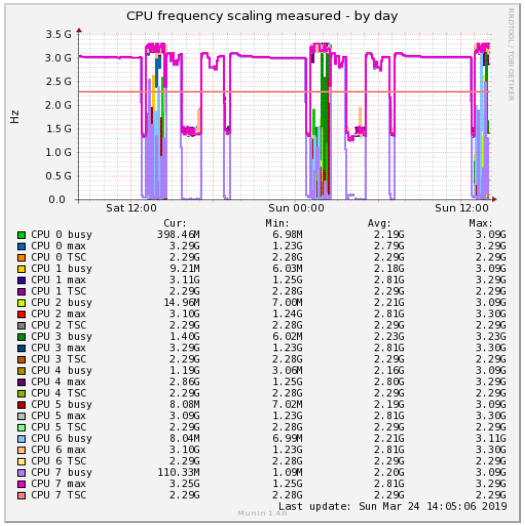


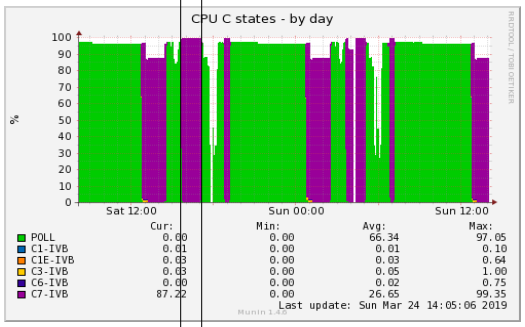


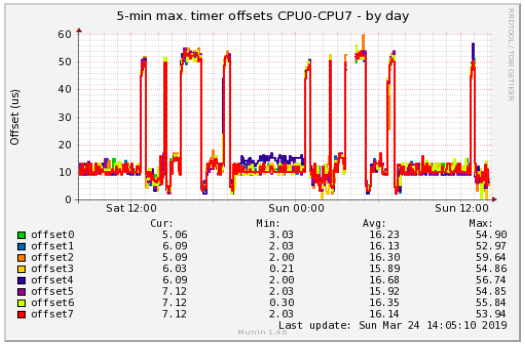


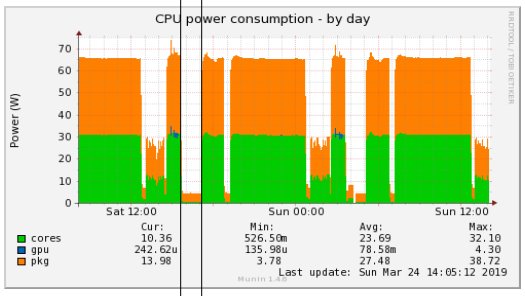
Using the *Munin* monitoring tool equipped with additional plugins

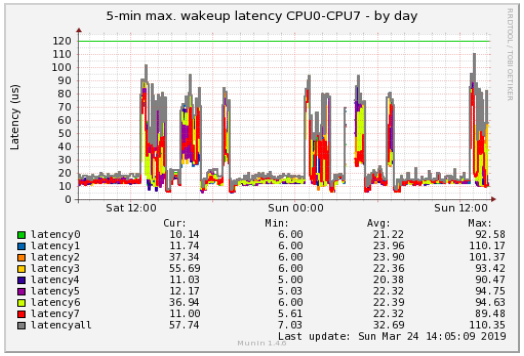
Continuous recording of real-time related system variables







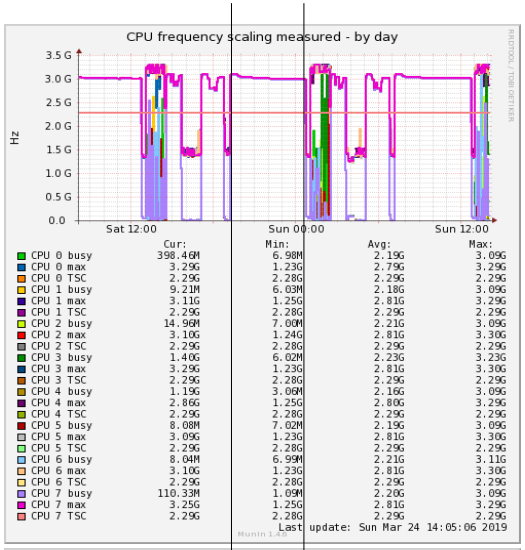




Temporal relationship between C states and power consumption

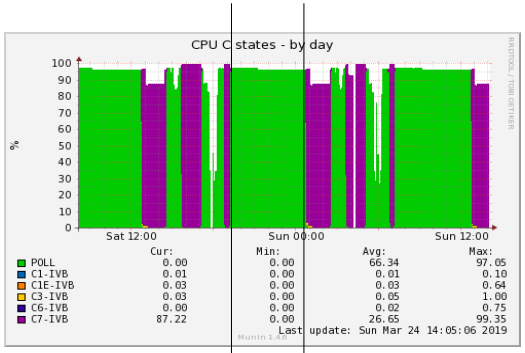
Using the *Munin* monitoring tool equipped with additional plugins

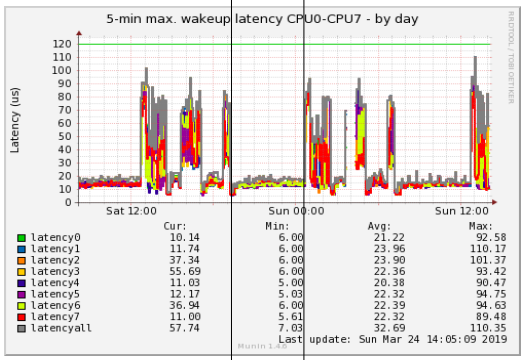
Compare real-time data to frequency modulation



Frequency modulation disabled leads to minimum latency

Compare real-time data to sleep stages





Sleep stages disabled (polling)
leads to minimum latency

Regain timing information that normally is lost in histograms