An overview of Linux tracing features

(present and near-future)

Jonathan Corbet
LWN.net
corbet@lwn.net

The agenda

Ftrace
Perf events
Tracepoints
Briefly:
- SystemTap
- Audit
- Linux security modules
- fanotify

Ftrace

An offshoot of the realtime preemption tree

Initial goals:

- Obtaining log information from the kernel
- Identifying latency problems

debugfs

Ftrace uses the debugfs filesystem

- Must be configured into the kernel
- (Most distributors do this)

mount -t debugfs none /sys/kernel/debug

Ftrace in debugfs

Everything is in the `tracing` directory

- `tracing_enabled`: tracing is enabled
- `tracing_on`: trace data being recorded
- `available_tracers`: options you have
- `current_tracer`: what is being traced now

Example

```
cd /sys/kernel/debug/tracing
echo function > current_tracer
echo 1 > tracing_enabled
head trace
```

```
# tracer: function
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
      audacious-4529  [001] 75183.937736: __might_sleep <-
                                              slab_pre_alloc_hook.clone.36
      audacious-4529  [001] 75183.937737: arch_local_save_flags <-__might_sleep
      audacious-4529  [001] 75183.937737: _cond_resched <-
                                              slab_pre_alloc_hook.clone.36
      audacious-4529  [001] 75183.937737: should_resched <-_cond_resched
      audacious-4529  [001] 75183.937737: need_resched <-should_resched
      audacious-4529  [001] 75183.937738: test_ti_thread_flag <-need_resched
      audacious-4529  [001] 75183.937738: arch_local_irq_save <-
                                                  __kmalloc_track_caller
      audacious-4529  [001] 75183.937738: trace_kmalloc <-
                                                  __kmalloc_track_caller
      audacious-4529  [001] 75183.937739: prefetchw <-__alloc_skb
      audacious-4529  [001] 75183.937739: atomic_add <-sock_alloc_send_pskb
      audacious-4529  [001] 75183.937739: unix_scm_to_skb <-unix_stream_sendmsg
      audacious-4529  [001] 75183.937739: atomic_inc <-unix_scm_to_skb
      audacious-4529  [001] 75183.937740: atomic_inc <-unix_scm_to_skb
      audacious-4529  [001] 75183.937740: skb_put <-unix_stream_sendmsg
      audacious-4529  [001] 75183.937740: memcpy_fromiovec <-
                                                    unix_stream_sendmsg
      audacious-4529  [001] 75183.937741: copy_from_user <-memcpy_fromiovec
```

How this works

Kernel is compiled with -pg option
Compiler inserts mcount() calls
Kernel's special mcount() provides data

If DYNAMIC_FTRACE is enabled
- mcount() calls are patched out at boot
- only enabled when tracing is turned on

The function graph tracer

echo function_graph > current_tracer

Yields a full call-chain profile

```
# tracer: function_graph
#
 1)               |  sys_poll() {
 1)               |    poll_select_set_timeout() {
 1)               |      ktime_get_ts() {
 1)   0.386 us    |        timekeeping_get_ns();
 1)   0.330 us    |        set_normalized_timespec();
 1)   1.753 us    |      }
 1)               |      timespec_add_safe() {
 1)   0.346 us    |        set_normalized_timespec();
 1)   1.017 us    |      }
 1)   3.807 us    |    }
 1)               |    do_sys_poll() {
 1)               |      copy_from_user() {
 1)               |        might_fault() {
 1)               |          __might_sleep() {
 1)   0.335 us    |            arch_local_save_flags();
 1)   1.022 us    |          }
 1)               |          _cond_resched() {
 1)               |            should_resched() {
 1)               |              need_resched() {
 1)   0.325 us    |                test_ti_thread_flag();
 1)   1.022 us    |              }
 1)   1.704 us    |            }
 1)   2.374 us    |          }
 1)   4.429 us    |        }
 1)   5.175 us    |      }
```

Function filtering

set_ftrace_filter
- Names of functions to trace
- can also restrict to specific modules

set_ftrace_notrace
- Names of functions to exclude

set_graph_function
- Limit base of function graph traces

Filtering example

```
echo function_graph > current_tracer
echo scheduler_tick > set_graph_function
echo 1  > tracing_enabled
cat trace
```

```
0)               |  scheduler_tick() {
0)               |    ktime_get() {
0)   0.732 us    |      timekeeping_get_ns();
0)   1.984 us    |    }
0)   0.926 us    |    _raw_spin_lock();
0)   0.746 us    |    update_rq_clock();
0)   0.631 us    |    update_cpu_load();
0)   0.591 us    |    task_tick_idle();
0)   0.711 us    |    _raw_spin_unlock();
0)   0.872 us    |    _raw_spin_lock();
0)   0.706 us    |    _raw_spin_unlock();
0)   0.591 us    |    idle_cpu();
0) + 14.748 us   |  }
0)               |  scheduler_tick() {
0)               |    ktime_get() {
0)   0.737 us    |      timekeeping_get_ns();
0)   1.984 us    |    }
0)   1.006 us    |    _raw_spin_lock();
0)   0.746 us    |    update_rq_clock();
0)   0.627 us    |    update_cpu_load();
0)   0.601 us    |    task_tick_idle();
0)   0.716 us    |    _raw_spin_unlock();
0)   0.972 us    |    _raw_spin_lock();
0)   0.707 us    |    _raw_spin_unlock();
0)   0.592 us    |    idle_cpu();
0) + 14.909 us   |  }
```

“irqsoff” tracer

```
# latency: 65921 us, #39993/17231355, CPU#0 | (M:desktop VP:0, KP:0, SP:0
HP:0 #P:2)
#    -----------------
#    | task: Xorg-3029 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: scsi_dispatch_cmd
#  => ended at:   scsi_dispatch_cmd
#
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /_--=> lock-depth
#                |||||/     delay
#  cmd     pid   |||||| time  |   caller
#     \   /      ||||||   \   |   /
    Xorg-3029    0dNs.. 55856us : __kernel_text_address <-
                                                  print_context_stack
    Xorg-3029    0dNs.. 55856us : core_kernel_text <-__kernel_text_address
    Xorg-3029    0dNs.. 55856us : is_module_text_address <-
                                                 __kernel_text_address
```

“sched_switch” tracer

```
# tracer: sched_switch
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
     ksoftirqd/1-10    [001] 82902.395620:     10:120:S ==> [001]  7545:120:R <...>
           <...>-7545  [001] 82902.398241:   7545:120:R   + [001]  7530:120:R <...>
           <...>-7545  [001] 82902.398245:   7545:120:x ==> [001]  7530:120:R <...>
           <...>-7530  [001] 82902.398575:   7530:120:R   + [000]  7549:120:R <...>
           <...>-7530  [001] 82902.398679:   7530:120:S ==> [001]  7539:120:R <...>
           <...>-7539  [001] 82902.398774:   7539:120:R ==> [001]  7549:120:R <...>
           <...>-7549  [001] 82902.398973:   7549:120:R ==> [001]  3543:109:R pulseaudio
      pulseaudio-3543  [001] 82902.399062:   3543:109:R   + [001]  4529:120:R audacious
      pulseaudio-3543  [001] 82902.399462:   3543:109:S ==> [001]  4529:120:R audacious
       audacious-4529  [001] 82902.399488:   4529:120:R   + [001]  3543:109:R pulseaudio
       audacious-4529  [001] 82902.399717:   4529:120:S ==> [001]  3543:109:R pulseaudio
      pulseaudio-3543  [001] 82902.399729:   3543:109:S ==> [001]  7549:120:R <...>
           <...>-7549  [001] 82902.400725:   7549:120:R   + [001]  7530:120:R <...>
           <...>-7549  [001] 82902.400729:   7549:120:x ==> [001]  7530:120:R <...>
           <...>-7530  [001] 82902.401047:   7530:120:R   + [001]  7550:120:R <...>
           <...>-7530  [001] 82902.401131:   7530:120:S ==> [001]  7539:120:R <...>
           <...>-7539  [001] 82902.408923:   7539:120:R   + [001]    10:120:R ksoftirqd/1
           <...>-7539  [001] 82902.408929:   7539:120:R ==> [001]    10:120:R ksoftirqd/1
     ksoftirqd/1-10    [001] 82902.408941:     10:120:S ==> [001]  7550:120:R <...>
           <...>-7550  [001] 82902.409160:   7550:120:R   + [001]     8:  0:R migration/1
           <...>-7550  [001] 82902.409164:   7550:120:R ==> [001]     8:  0:R migration/1
     migration/1-8     [001] 82902.409175:      8:  0:S ==> [001]  7539:120:R <...>
           <...>-7539  [001] 82902.412342:   7539:120:R ==> [001]  7530:120:R <...>
           <...>-7530  [001] 82902.412650:   7530:120:R   + [001]  5331:120:R <...>
           <...>-7530  [001] 82902.412654:   7530:120:x ==> [001]  5331:120:R <...>
           <...>-5331  [001] 82902.413048:   5331:120:R   + [000]  7552:120:R <...>
```

“blk” tracer

```
echo 1 > /sys/block/sda/trace/enable

# tracer: blk
#
     jbd2/sda1-8-1161  [000] 83611.578281:   8,0    A  WS 30042695 + 8 <- (8,1) 30042632
     jbd2/sda1-8-1161  [000] 83611.578396:   8,0    Q  WS 30042695 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578403:   8,0    G  WS 30042695 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578408:   8,0    P   N [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578410:   8,0    I   W 30042695 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578413:   8,0    m   N cfq1161 insert_request
     jbd2/sda1-8-1161  [000] 83611.578415:   8,0    m   N cfq1161 add_to_rr
     jbd2/sda1-8-1161  [000] 83611.578429:   8,0    A  WS 26344903 + 8 <- (8,1) 26344840
     jbd2/sda1-8-1161  [000] 83611.578430:   8,0    Q  WS 26344903 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578434:   8,0    G  WS 26344903 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578435:   8,0    I   W 26344903 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578437:   8,0    m   N cfq1161 insert_request
     jbd2/sda1-8-1161  [000] 83611.578443:   8,0    A  WS 30042895 + 8 <- (8,1) 30042832
     jbd2/sda1-8-1161  [000] 83611.578444:   8,0    Q  WS 30042895 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578447:   8,0    G  WS 30042895 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578448:   8,0    I   W 30042895 + 8 [jbd2/sda1-8]
     jbd2/sda1-8-1161  [000] 83611.578449:   8,0    m   N cfq1161 insert_request
```

The event tracer

Hook into any kernel tracepoint

- Often use “nop” tracer

`available_events`

- A list of all events known to the kernel

`set_event`

- Use to enable specific events

Event tracing

```
echo nop > current_tracer
echo syscalls:sys_enter_poll > set_event
head trace
# tracer: nop
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
           cupsd-2821  [000] 84472.305890:
sys_poll(ufds: 7fff1577e240, nfds: 1, timeout_msecs: 2710)
      claws-mail-4020  [001] 84472.306028:
sys_poll(ufds: 491f280, nfds: c, timeout_msecs: c8)
      claws-mail-4020  [001] 84472.306078:
sys_poll(ufds: 491f280, nfds: c, timeout_msecs: c8)
  gnome-terminal-3761  [000] 84472.306091:
sys_poll(ufds: 194f1a0, nfds: d, timeout_msecs: a)
      claws-mail-4020  [001] 84472.306109:
sys_poll(ufds: 491f280, nfds: c, timeout_msecs: c8)
```

Event parameters

Every tracepoint appears in:

.../tracing/events/subsys/name

Examples:

.../tracing/events/sched/sched_process_fork

.../tracing/events/syscalls/sys_enter_poll

Per-event attributes

Each event has these attributes

- `enable`: Turn the tracepoint on or off
- `filter`: filter events from this tracepoint
- `format`: describe the tracepoint format
- `id`: a numerical ID number

To enable `write()` tracing:

```
echo 1 > .../tracing/events/syscalls/sys_enter_write
```

filter and format

filter can be used to reduce output

- Simple boolean expressions
- Can only use the tracepoint parameters

format describes the tracepoint

- Which parameters
- Their types
- The associated format string

Dynamic event addition

Tracepoints can be placed at runtime

```
echo 'p:myprobe do_sys_open dfd=%ax filename=%dx \
     flags=%cx mode=+4(stack)' > kprobe_events
```

trace_printk()

A way to write to the event stream:

```
trace_printk("Kilroy was here\n");
```

Like printk(), but:

- Only active when tracing is enabled
- Far more efficient
- Mixes with other trace data

function_profile

Function	Hit	Time	Avg
mwait_idle	10132	246835458 us	24361.96 us
tg_shares_up	154467	389883.5 us	2.524 us
_raw_spin_lock_irqsave	343012	263504.3 us	0.768 us
_raw_spin_unlock_irqrestore	351269	175205.6 us	0.498 us
walk_tg_tree	14087	126078.4 us	8.949 us
__set_se_shares	274937	88436.65 us	0.321 us
_raw_spin_lock	100715	82692.61 us	0.821 us
kstat_irqs_cpu	257500	80124.96 us	0.311 us

Other tracers

wakeup

Track the longest wakeup latencies

stack

Tracks maximum kernel stack depth

mmio

Memory-mapped I/O operations

branch

Profile if-statement outcomes

Check likely/unlikely declarations

Tracing options

set_ftrace_pid
- limit tracing to a single process

tracing_cpu_mask
- limit the CPUs on which tracing is done
- see also the per_cpu hierarchy

Many, many others

Trace output

`trace`

- Holds the first N bytes of the trace output
- Rereading yields the same data

`trace_pipe`

- Continuous stream of trace output
- Data is consumed as read

`trace_pipe_raw`

- Binary-formatted trace data
- Per-CPU only

Tracing front-ends

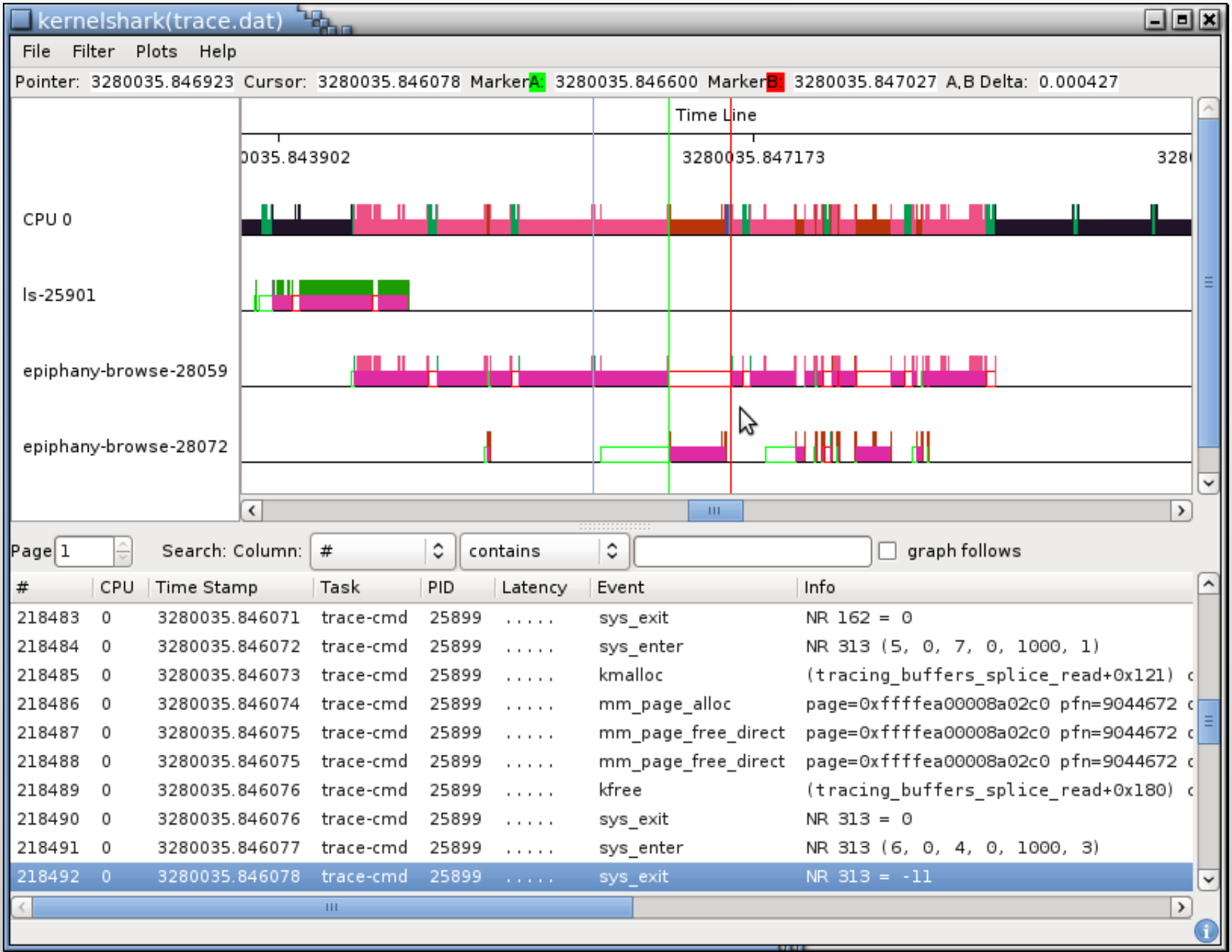
trace-cmd

- Simple command-line access to most functionality
- `http://lwn.net/Articles/410200/`

trace

- strace-like application tracing
- 2.6.38
- `http://lwn.net/Articles/415728/`

kernelshark



Questions about ftrace?

Perf events

Perf is:

- Access to hardware performance counters
- A high-volume event delivery mechanism
- A statistical event analysis tool

...where a lot of development has been happening

Simple example

```
# perf stat -e cycles ls
<...>
 Performance counter stats for 'ls':

            4011990  cycles

      0.002607205  seconds time elapsed
```

Simplest example

```
# perf stat ls
<...>
 Performance counter stats for 'ls':

          2.118830  task-clock-msecs         #      0.778 CPUs
                 0  context-switches         #      0.000 M/sec
                 0  CPU-migrations           #      0.000 M/sec
               274  page-faults              #      0.129 M/sec
           3443245  cycles                   #   1625.069 M/sec
           3013227  instructions             #      0.875 IPC
            604499  branches                 #    285.298 M/sec
             17312  branch-misses            #      2.864 %      (scaled
from 17.82%)
      <not counted>  cache-references
      <not counted>  cache-misses

        0.002723457  seconds time elapsed
```

Underneath it all

The core system call:

```
int perf_event_open(struct perf_event_attr *event,
                    pid_t pid,
                    int cpu,
                    int group_fd,
                    unsigned long flags);
```

Notes:

- One call per event of interest
- group_fd allows multiple counters to be managed as a unit
- flags should be zero

Event descriptions

There are various event types

- Hardware events
- Hardware breakpoints
- Software events (page faults, context switches..)
- Tracepoints

Other event information

- Extra info (IP, call chain, time, ...)
- Overflow threshold
- ...

perf event file descriptors

read() to get the current counter value

- Plus maybe some ancillary information

ioctl() to enable or disable

mmap() to get a ring buffer

- Fast access to detailed event data

Basic perf commands

perf record

- Records perf data for later analysis
- Lots of options

perf report

- Generate a report from recorded data

```
perf record -e kmem:kmalloc -c 1 -a
perf report --sort comm,dso,symbol --stdio
# Events: 16K cycles
#
# Overhead          Command      Shared Object                  Symbol
# ........  ...............  .................  ......................
#
    35.10%          swapper  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
    24.97%             Xorg  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
    13.81%        audacious  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     7.98%       pulseaudio  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     5.86%             perf  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     3.48%       claws-mail  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     2.63%          firefox  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     1.21%  gnome-settings-  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     1.01%            cupsd  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.92%          metacity  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.75%             emacs  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.69%      wnck-applet  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.47%   gnome-terminal  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.46%        irqbalance  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.45%      dbus-daemon  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
     0.09%    udisks-daemon  [kernel.kallsyms]  [k] perf_trace_kmem_alloc
```

```
perf record -e kmem:mm_page_alloc -c 1 -g xv image.jpg
perf report --sort symbol
   100.00%  [k] perf_trace_mm_page_alloc
            |
            --- __alloc_pages_nodemask
                |
                |--84.26%-- alloc_zeroed_user_highpage_movable.clone.54
                |           |
                |           |--99.94%-- handle_mm_fault
                |           |           do_page_fault
                |           |           page_fault
                |           |           |
                |           |           |--63.56%-- Pic24ToXImage
                |           |           |           CreateXImage
                |           |           |           |
                |           |           |           |--49.54%-- Resize
                |           |           |           |           HandleEvent
                |           |           |           |           EventLoop
                |           |           |           |           mainLoop
                |           |           |           |           main
                |           |           |           |           __libc_start_main
                |           |           |           |
                |           |           |           |--46.27%-- DrawEpic
                |           |           |           |           SelectDispMB
                |           |           |           |           handleKeyEvent
                |           |           |           |           HandleEvent
                |           |           |           |           EventLoop
                |           |           |           |           mainLoop
                |           |           |           |           main
                |           |           |           |           __libc_start_main
```

perf top

corbet@tpl:/home/corbet

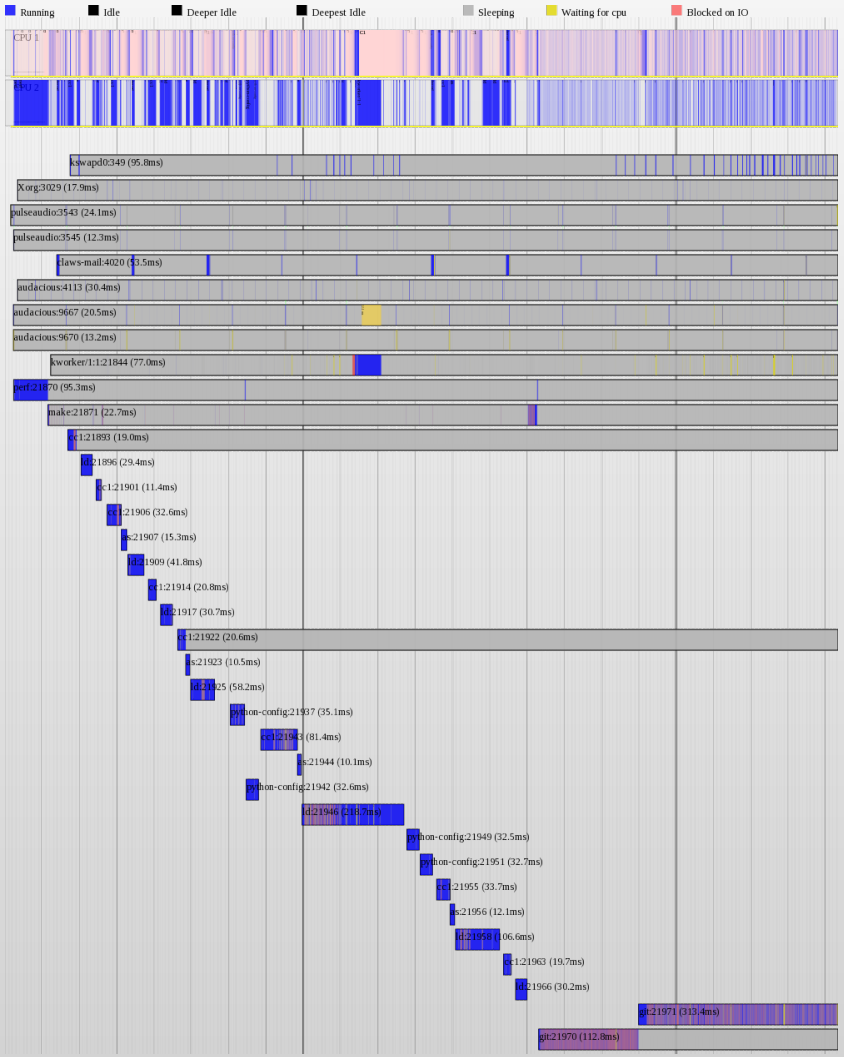
File Edit View Search Terminal Help

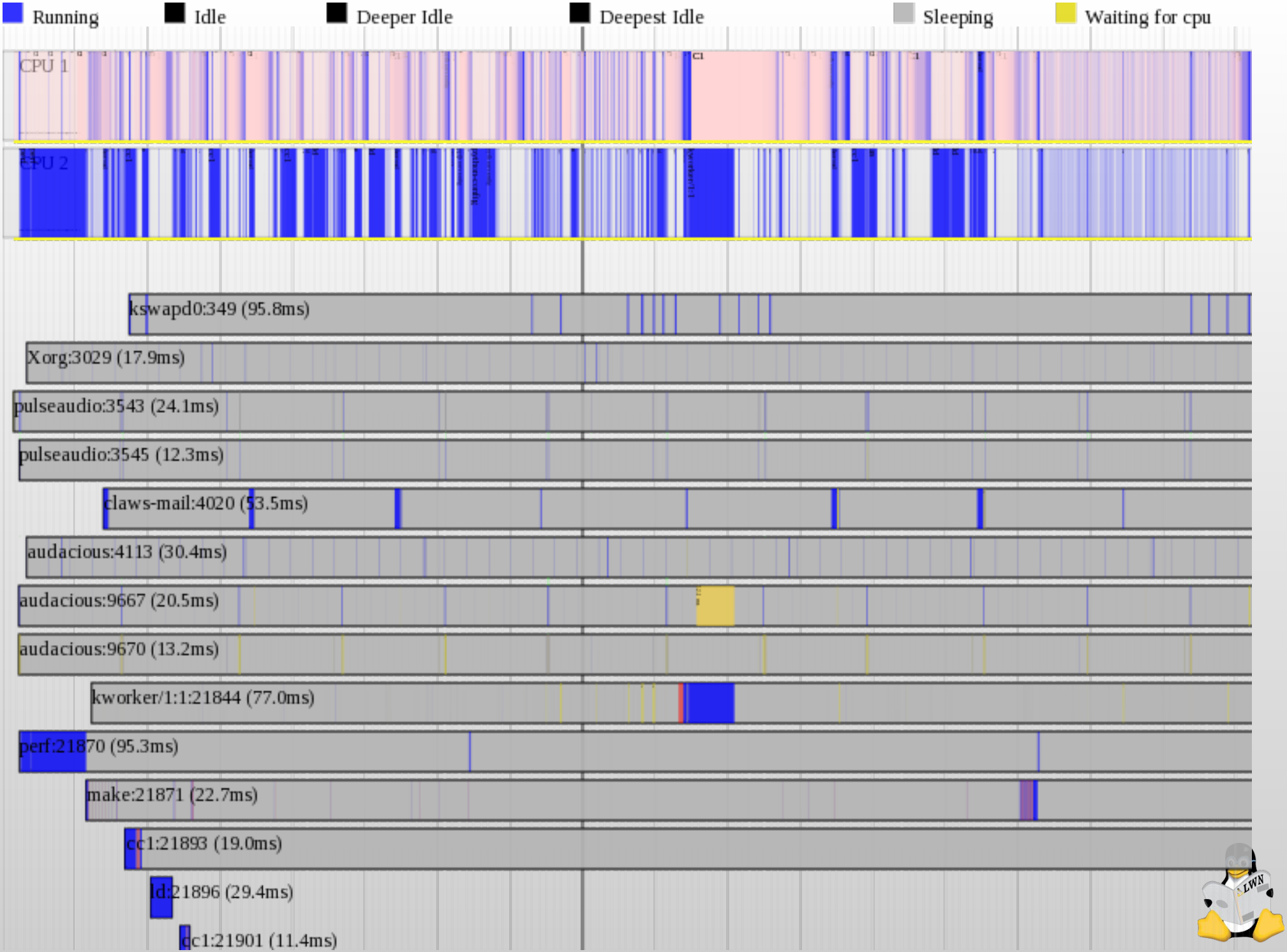
```
------------------------------------------------------------------------------
   PerfTop:     727 irqs/sec  kernel:23.1%  exact:  0.0% [1000Hz cache-misses],  (all, 2 CPUs)
------------------------------------------------------------------------------

             samples  pcnt function                    DSO
             _______ _____ ____________________________ _______________

               59.00  6.6% probe_workqueue_insertion  [kernel]
               57.00  6.4% rb_next                    [kernel]
               53.00  6.0% lock_page_cgroup           [kernel]
               48.00  5.4% i915_gem_shrink            [i915]
               44.00  4.9% inflate                    libz.so.1.2.5
               44.00  4.9% unix_poll                  [kernel]
               35.00  3.9% page_cache_get_speculative [kernel]
               29.00  3.3% list_del                   [kernel]
               27.00  3.0% __wake_up_bit              [kernel]
               27.00  3.0% find_vma                   [kernel]
               26.00  2.9% _raw_spin_lock_irqsave     [kernel]
               16.00  1.8% __memcpy_ssse3             libc-2.12.90.so
               16.00  1.8% _int_malloc                libc-2.12.90.so
               16.00  1.8% sock_poll                  [kernel]
               15.00  1.7% _int_free                  libc-2.12.90.so
               13.00  1.5% get_page_from_freelist     [kernel]
               13.00  1.5% do_raw_spin_lock           [kernel]
               11.00  1.2% unmap_vmas                 [kernel]
               11.00  1.2% do_select                  [kernel]
               10.00  1.1% inflateReset2              libz.so.1.2.5
               10.00  1.1% dso__find_symbol           /usr/bin/perf
                9.00  1.0% __cfree                    libc-2.12.90.so
```

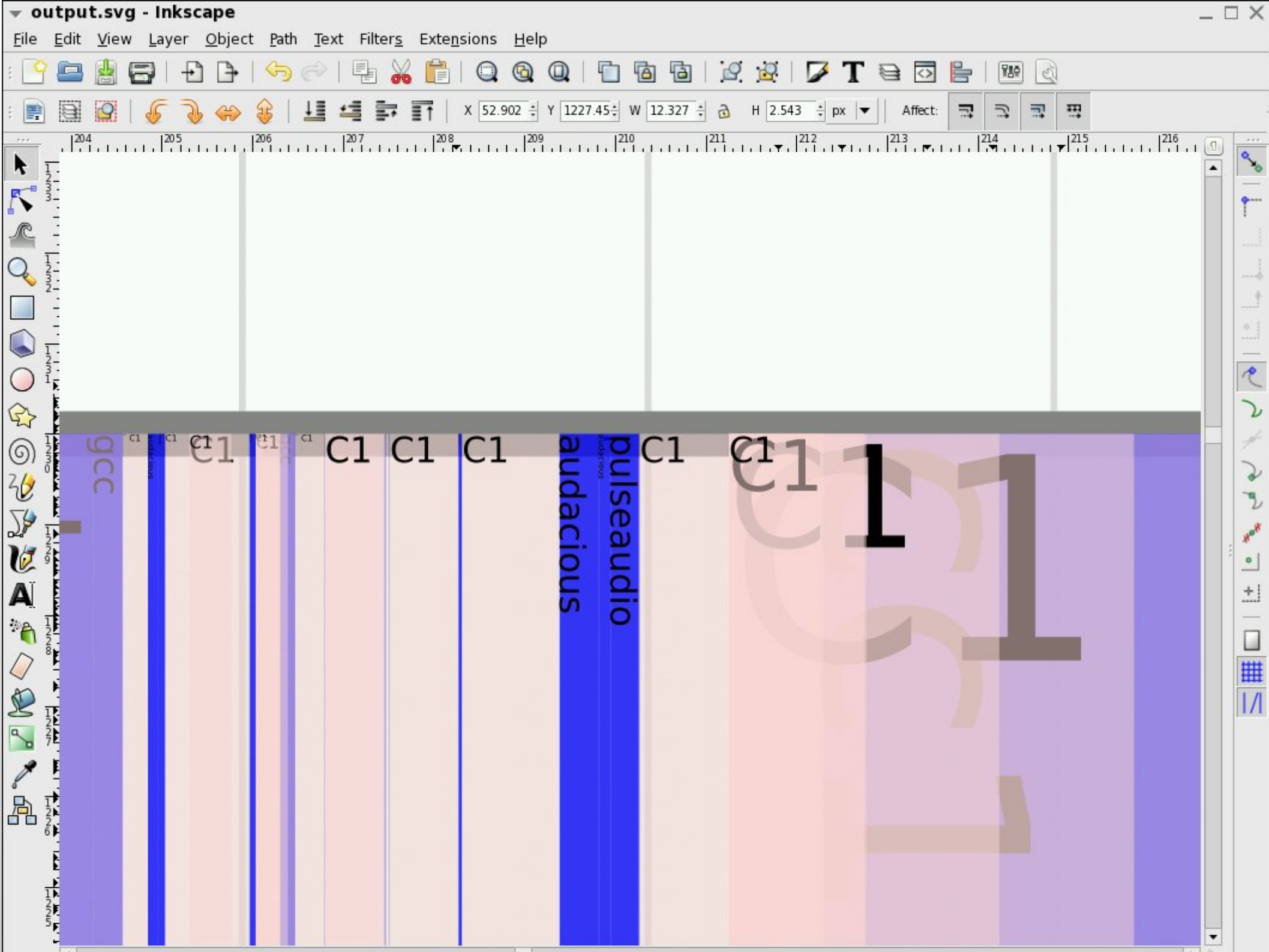
perf timechart

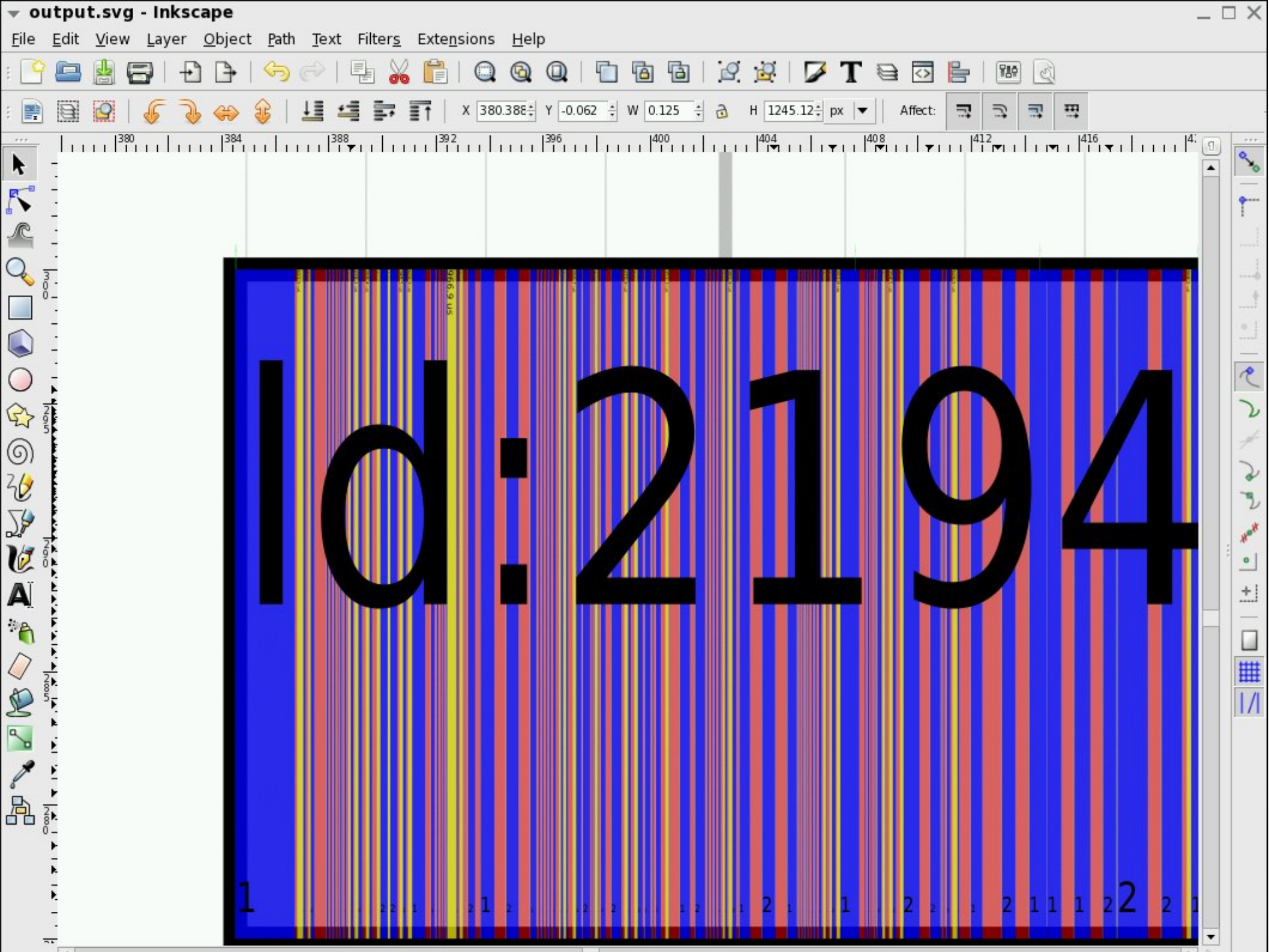
View scheduling and power behavior of the system.

```
perf timechart [record]
```




Running
Idle
Deeper Idle
Deepest Idle
Sleeping
Waiting for cpu
CPU 1
CPU 2
kswapd0:349 (95.8ms)
Xorg:3029 (17.9ms)
pulseaudio:3543 (24.1ms)
pulseaudio:3545 (12.3ms)
claws-mail:4020 (53.5ms)
audacious:4113 (30.4ms)
audacious:9667 (20.5ms)
audacious:9670 (13.2ms)
kworker/1:1:21844 (77.0ms)
perf:21870 (95.3ms)
make:21871 (22.7ms)
cc1:21893 (19.0ms)
ld:21896 (29.4ms)
cc1:21901 (11.4ms)


output.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
X 52.902 Y 1227.45 W 12.327 H 2.543 px Affect:
gcc
pulseaudio
audacious
C1


output.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
X 380.388 Y -0.062 W 0.125 H 1245.12 px Affect:
id:2194

Other perf features

perf sched
- Trace scheduling decisions and latencies

perf lock
- Spinlock and mutex behavior

perf kvm
- Monitor virtualized guest and host together

perf probe
- Insert dynamic tracepoints

perf annotate
- Instruction-level event reporting

Scripting

Framework for analysis scripts

- Perl
- Python

perf and ftrace

The two have some overlap

Who is the fairest event reporter of them all?

"Well, the direction is that we are unifying ftrace and perf events and we are actively phasing out individual ftrace plugins as matching events become available... Most new tools use the perf syscall and tool writers have expressed the very understandable desire that all events be enumerated and accessible via a unified API/ABI."

-- Ingo Molnar

perf questions?

Tracepoints

Goals of tracepoints

- Signal events to tracing tools
- Support human-readable information
- Support high-speed binary information
- Be self-documenting
- Minimal cluttering of code
- Minimal (zero) runtime impact

Linux tracepoints do all this

- At the cost of some complexity

Tracepoint declarations

Tracepoints must be defined separately

The basic form is:

```
TRACE_EVENT(name,
            prototype,
            args,
            binary_structure,
            assignment_code,
            printk_args);
```

```
#define TRACE_SYSTEM irq

TRACE_EVENT(irq_handler_entry,
  TP_PROTO(int irq, struct irqaction *action),
  TP_ARGS(irq, action),
  TP_STRUCT__entry(
      __field( int, irq     )
      __string(   name,   action->name )
  ),
  TP_fast_assign(
      __entry->irq = irq;
      __assign_str(name, action->name);
  ),
  TP_printk("irq=%d name=%s", __entry->irq,
                              __get_str(name))
);
```

Tracepoint placement

```
irqreturn_t handle_IRQ_event(unsigned int irq,
                             struct irqaction *action)
{
        irqreturn_t ret, retval = IRQ_NONE;
        unsigned int status = 0;

        do {
            trace_irq_handler_entry(irq, action);
            ret = action->handler(irq, action->dev_id);
            trace_irq_handler_exit(irq, action, ret);
            /* ... */
```

Other tracepoint notes

Tracepoints are off by default
- (Hopefully) little or no runtime impact

Tracepoint information is found in sysfs
- `/sys/kernel/debug/tracing/events/irq/irq_handler_entry`
- Enable flag
- Format description
- Filtering - limit when the tracepoint fires
  - Simple expression based on args

Tracepoints and ABI

Are tracepoints part of the Linux ABI?

Arguments for:

- They are clearly an interface to user space
- Applications are beginning to depend on them
- Pushed for user-oriented features

Arguments against:

- They reflect a kernel state which changes
- ABI status can retard kernel development

ABI status

“Stable” tracepoints will be marked specially
- Might move to /sys/events/
- Cannot be registered from modules

Other tracepoints can be volatile
- ...within reason
- Applications should use format descriptions

User-space tracepoints

Can be supported with utrace/uprobes
- Significant opposition to these patches

Maybe a user-space problem
- Probes inserted with a gdb-like tool

Missing piece: add user-space events to the stream

```
prctl(PR_TASK_PERF_USER_TRACE, msg, len);
```

Tracepoint questions?

In brief: SystemTap

The Linux answer to DTrace (sort of)

- Dynamic probing capabilities
- Strong analysis features

The “stap” language

Place probes into the kernel
- ...or user space
- Can also hook into existing tracepoints

Respond to events on the probes
- Collect data
- Perform analysis

Generate reports

Running stap programs

Stap programs are compiled to a C module

That module is then loaded into the kernel

- Data collection/analysis done there

SystemTap status

Not supported in the mainline kernel
- Numerous out-of-tree modules needed

Well supported on enterprise distributions
- Poorly supported elsewhere

The future of this project is unclear

In brief: audit

Linux has an audit subsystem
- File and process events, mostly
- Focus on reliable information delivery

Can be used to track actions on the system

If it were done today
- ...it might be done with tracepoints

In brief: Linux security modules

There is an internal API for security modules
- Hooks provided for security-relevant operations
- Used by:
  - SELinux, TOMOYO, Smack, AppArmor, ...

Rules
- Modules are built into the kernel
- Can only tighten policy

In brief: fanotify

An API for malware scanners
- A daemon can intercept file-related syscalls
- Those calls can be denied

Other applications may exist
- Hierarchical storage

Merged for 2.6.36
- System calls disabled due to API issues

In summary

Kernel visibility has come a long way

- Most of this infrastructure is < 2 years old

Many problems remain to be solved

Stay tuned!