

Using the Linux Tracing Infrastructure

Jan Altenberg

Linutronix GmbH

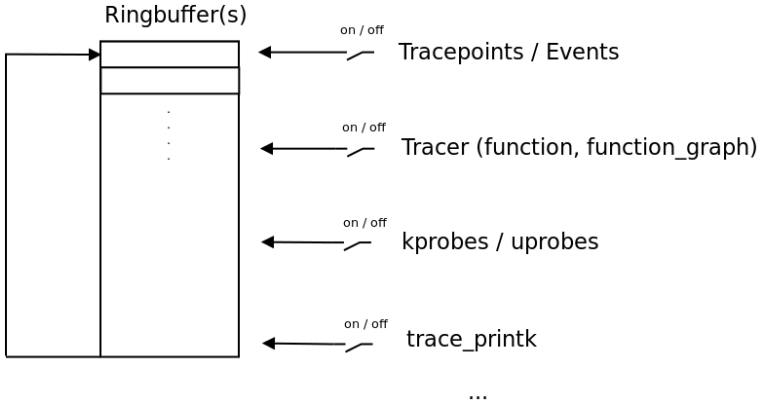
Overview

- 1 Event tracing
- 2 Tracers
- 3 trace_marker
- 4 trace_printk
- 5 kprobes
- 6 uprobes
- 7 trace-cmd
- 8 Kernelshark
- 9 Tracecompass

Kerneltracing: Overview

- ❑ DebugFS / TraceFS interface
- ❑ Event Tracing
- ❑ Custom trace events
- ❑ Different tracers: function, function_graph, wakeup, wakeup_rt, ...
- ❑ Graphical frontend(s) available

Kerneltracing: Overview



Kernel-Tracing: DebugFS

```
$ mount -t debugfs debugfs /sys/kernel/debug  
$ cd /sys/kernel/debug/tracing  
$ cat available_tracers  
blk function_graph mmioTRACE wakeup_rt wakeup
```

Event tracing

- ❑ **Pre-defined Events in the kernel**
- ❑ **Event groups**
- ❑ **Each event comes with several options**
- ❑ **Filtering based on event options**

Event tracing

```
$ cd /sys/kernel/debug/tracing
$ ls -1 events/sched/
enable
filter
sched_kthread_stop
sched_kthread_stop_ret
sched_migrate_task
sched_pi_setprio
[...]
```

Event tracing: Enable events

```
$ cd /sys/kernel/debug/tracing  
# Enable ALL events of the group 'sched'  
$ echo 1 > events/sched/enable
```


Record a trace

After enabling the events you want to see, do:

```
$ cd /sys/kernel/debug/tracing
# Start recording to the ringbuffer
$ echo 1 > tracing_on
# Stop recording to the ringbuffer
$ echo 0 > tracing_on
```

Analyze a trace

You can even do this while recording!

```
$ cd /sys/kernel/debug/tracing
# Just print the current content of the ring buffer
$ cat trace
# or: do a consuming read on the ring buffer
$ cat trace_pipe
```

Trace event format and filters

Each trace event has a specific format and parameters. You can put a filter on those parameters for recording a trace:

```
$ cat events/sched/sched_switch/format
[...]
```

```
field:__u8 prev_comm[15];
field:pid_t prev_pid;
field:int prev_prio;
field:long prev_state;
[...]
```

```
$ echo 'next_comm == bash' \
    > events/sched/sched_switch/filter
$ echo 1 > events/sched/sched_switch/enable
$ echo 1 > tracing_on
$ cat trace
```

Tracing on multicore

- ❏ One ringbuffer per cpu
- ❏ trace contains ALL events
- ❏ the `per_cpu` directory contains a trace for each cpu
- ❏ `tracing_cpumask` can limit tracing to specific cores

Tracers

- ☐ Already have some special logic
- ☐ Latency hunting
- ☐ Callgraphs
- ☐ Kernel profiling
- ☐ ...

Tracers

`available_tracers` contains the tracers which are enabled in the kernel configuration. The tracer is set by the `current_tracer` file:

- ☐ `function`: Can turn all functions into trace events
- ☐ `function_graph`: Similar to `function`, but contains a call graph
- ☐ `wakeup / wakeup_rt`: Measure the wakeup time for tasks / rt tasks
- ☐ `irqsoff`: useful for latency hunting. Identifies long sections with IRQs turned off
- ☐ ...

Tracer: function

```
# tracer: function
#
# TASK-PID   CPU#      TIMESTAMP  FUNCTION
#   |   |   |   |   |
wnck-2022   [003]    5766.659915:  skb_release
wnck-2022   [003]    5766.659916:  sock_wfree
wnck-2022   [003]    5766.659917:  unix_write_free
wnck-2022   [003]    5766.659917:  skb_releasee_skb
wnck-2022   [003]    5766.659918:  kfree <-skb
```

Tracer: function_graph

```

$ echo function_graph > current_tracer
$ echo 1 > tracing_on
$ sleep 1
$ echo 0 > tracing_on
$ less trace
# tracer: function_graph
# CPU    DURATION    FUNCTION CALLS
# |      |      |          |
1)      |      |          enqueue_entity() {
1)      |      |          update_curr() {
1)    0.336 us |          task_of();
1)    1.524 us |          }
1)    0.420 us |          place_entity();

```


function_graph: Set a trigger function

You can set a trigger function for the `function_graph` tracer if you just want to record specific functions and their child's:

```
echo do_IRQ > set_graph_function
# Additional triggers can be set with
echo another_function >> set_graph_function
```

Tracer: function / latency_format

```

$ echo 1 > options/latency_format
# tracer: function
#
# function latency trace v1.1.5 on 3.9.4-x1-00124-g0bfd8ff
#-----
# latency: 0 us, #204955/25306195, CPU#0 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:4)
#
# | task: -0 (uid:0 nice:0 policy:0 rt_prio:0)
#-----
#
#          _-----> CPU#
#         /_-----> irqsoft
#        ||/_-----> need-resched
#       |||/_-----> hardirq/softirq
#      ||||/_-----> preempt-depth
#     |||||/_-----> delay
#  cmd   pid  ||||| time | caller
#  \    /    ||||| \   /
terminol-11964 1.... 11639243us : ep_read_events_proc <-ep_scan_ready_list.isra.8

```

Custom application tracepoints: "simple method"

```

$ echo 1 > tracing_on
$ echo "MARK" > trace_marker
$ echo 0 > tracing_on
$ less trace
...
bash-4328 [003] 5603.687935: get_slab
bash-4328 [003] 5603.687935: _cond_re
bash-4328 [003] 5603.687936: _cond_re
bash-4328 [003] 5603.687939: 0: MARK
bash-4328 [003] 5603.687939: kfree <-
...

```

trace_printk()

- ❏ `trace_printk()` can be used to write messages to the tracing ring buffer
- ❏ Usage is similar to `printk()`

Tracing related kernel parameters

`ftrace=`

Set and start specified tracer as early as possible.

`ftrace_dump_on_oops[=orig_cpu]`

Dump the tracing ring buffer if an Oops occurs. Using `orig_cpu` it will only dump the buffer of the CPU which triggered the Oops.

`ftrace_filter=`

Only trace specific functions.

`ftrace_notrace=`

Don't trace specific functions.

`trace_event=`

Just enable trace events (comma separated list)

Dump trace buffer

The trace buffer can also be dumped by:
SysRQ-z
OR

```
echo z > /proc/sysrq-trigger
```

Trace instances

You can have separate trace instances with their own buffers and events:

```
$ cd /sys/kernel/debug/tracing
$ mkdir instances/my_inst1
$ cd instances/my_inst1
$ echo 1 > events/sched/enable
$ cat trace
[...]
```

Dynamic kernel tracepoints: KPROBES

- ❑ Similar to Tracepoints
- ❑ Can be added / removed dynamically

Dynamic kernel tracepoints: KPROBES

```

$ echo 'p:my_k_event do_IRQ' > kprobe_events
$ echo 1 > events/kprobes/my_k_event/enabled
$ echo 1 > tracing_on
$ cat trace
<idle>-0 [000] d... 545.173709: my_k_event: (do_IRQ+0x0/0xc0)
<idle>-0 [000] d... 545.331051: my_k_event: (do_IRQ+0x0/0xc0)
<idle>-0 [000] d... 545.331490: my_k_event: (do_IRQ+0x0/0xc0)
<idle>-0 [000] d... 545.490730: my_k_event: (do_IRQ+0x0/0xc0)

```

Dynamic kernel tracepoints: KPROBES for custom modules

Let's assume we want to have a tracepoint for the function `hello_init` in the module `hello.ko`

```
# Note: >> will append a new event
$ echo 'p:my_mod_event hello:hello_init' \
    >> kprobe_events
$ echo 1 > events/kprobes/my_mod_event/enable
$ insmod hello.ko
$ cat trace
insmod-9586 [000] d... 13278.003468: my_mod_event: (0xf878d080)
```

KPROBES statistics

```
$ cat kprobe_profile
my_mod_event_ret      2      0
my_mod_event          2      0
```

Dynamic Userspace Tracepoints: uprobes

- ❏ Similar to kprobes
- ❏ For userspace applications
- ❏ A uprobe event is set on a specific offset in a userland process
- ❏ Powerful method to correlate your kernel and userland events!

Dynamic Userspace Tracepoints: uprobes

```
$ gcc -Wall -g -o pthread_example \
  pthread_example.c -lpthread
$ objdump -F -D -S pthread_example | less
```

```
08048594 <my_test_thread> (File Offset: 0x594):
[...]
```

```
void *my_test_thread(void *x_void_ptr)
[...]
```

```
    for (i = 0; i < 10; i++) {
80485a1:  c7 45 f4 00 00 00 00    movl    $0x0,-0xc(%ebp)
80485a8:  eb 1c                    jmp     80485c6 <my_test_thread+0x32> (File Offset:
    printf("The answer is 42!\n");
80485aa:  c7 04 24 50 87 04 08    movl    $0x8048750,(%esp)
```

So, the file offset for the printf call is 0x5aa !

Dynamic Userspace Tracepoints: uprobes II

```
echo \  
'p:my_ev /home/devel/pthread/pthread_example:0x5aa' \  
> /sys/kernel/debug/tracing/uprobe_events
```

Dynamic Userspace Tracepoints: uprobes III

```

$ cd /sys/kernel/debug/tracing/
$ echo 1 > events/uprobes/my_ev/enable
$ echo 1 > tracing_on
$ /home/devel/pthread_example/pthread_example
$ echo 0 > tracing_on
$ less trace
#      TASK-PID      CPU#  ||||      TIMESTAMP  FUNCTION
#      | |          |   |  ||||      |          |
ARTHUR_DENT-5223  [000] d...  5653.154822: my_ev: (0x80485aa)
ARTHUR_DENT-5223  [000] d...  5654.155723: my_ev: (0x80485aa)

```

uprobes: statistics

```
$ cat uprobe_profile  
/home/devel/pthread/pthread_example my_ev 10
```


trace-cmd

What is trace-cmd?

trace-cmd is a commandline utility for controlling and analysing kernel traces.

trace-cmd: Usage

```

$ trace-cmd
[...]
record - record a trace into a trace.dat file
start - start tracing without recording into a file
extract - extract a trace from the kernel
stop - stop the kernel from recording trace data
reset - disable all kernel tracing / clear trace buffers
report - read out the trace stored in a trace.dat file
split - parse a trace.dat file into smaller file(s)
listen - listen on a network socket for trace clients
list - list the available events, plugins or options
restore - restore a crashed record
stack - output, enable or disable kernel stack tracing

```

trace-cmd: Usage

```
# Recording a sched_switch trace
$ trace-cmd -p sched_switch
  plugin sched_switch
Hit Ctrl^C to stop recording
Hit Ctrl^C to stop recording
offset=ae000
[...]

CPU: 0
entries: 0
overrun: 0
commit overrun: 0
```

trace-cmd: Usage

```
# Analysing a trace
$ trace-cmd report
version = 6
cpus=1
trace-cmd-29057 [000] 6901.652365: wakeup:
29057:120:1 ==+ 29057:120:1 [000]
trace-cmd-29057 [000] 6901.652388: wakeup:
29057:120:0 ==+ 323:120:0 [000]
trace-cmd-29057 [000] 6901.652393: context_switch:
29057:120:0 ==> 323:120:0 [000]
kworker/0:1-323 [000] 6901.652397: wakeup:
323:120:0 ==+ 28355:120:0 [000]
```

trace-cmd: Usage

- ❏ **trace-cmd record** generates a file called `trace.dat`. This can be overridden by the `-o` option
- ❏ **trace-cmd report** uses the `-i` option for specifying an input file

trace-cmd: Record specific events

```
trace-cmd record -e sched  
# or a specific scheduler event  
trace-cmd record -e sched_wait_task  
# List available events and options  
trace-cmd report --events
```

trace-cmd: Filters

Based on the options from "trace-cmd report --events":

```
trace-cmd record -e context_switch \  
                -f 'next_pid == 323'
```

trace-cmd: Tracing a specific command

Enable tracing while a specific command is being executed:

```
$ trace-cmd record -p function ls
```


trace-cmd: Recording traces via network

On the host:

```
trace-cmd listen -p 1234 -o trace_remote
```

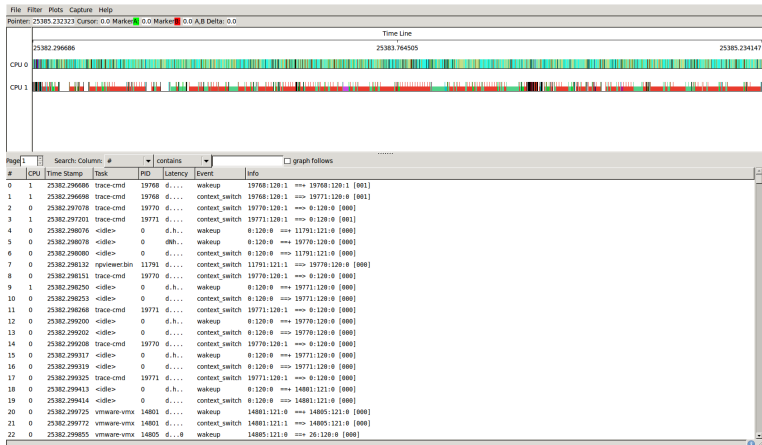
On the target:

```
trace-cmd record -p sched_switch \  
-N 192.168.0.182:1234 /bin/ls
```

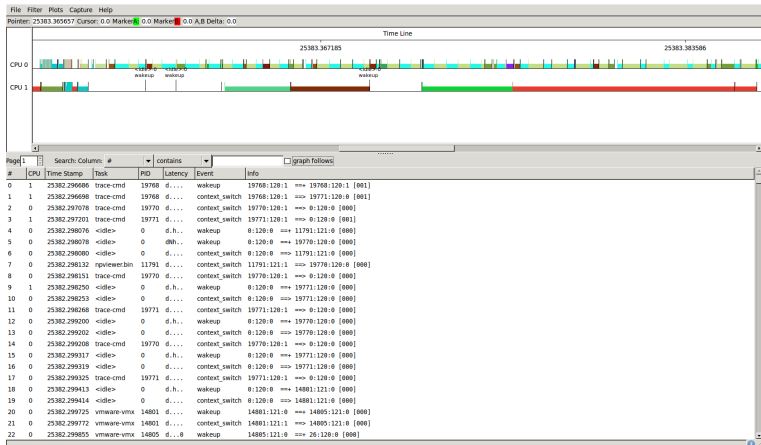
Kernelshark: A graphical front-end

```
$ kernelshark  
# or  
$ kernelshark -i mytrace.dat
```

Kernelshark



Kernelshark



Tracecompass

- ❑ Uses the C ommon T race F ormat
- ❑ perf can convert traces to CTF
- ❑ perf uses libbabeltrace for the conversion
- ❑ A recent version of libbabeltrace is needed

Build perf for your Target

```
cd kernel_source/tools/perf
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
# Now copy the perf executable to the target
```

Setup the tools on your host: libbabeltrace

```
git clone https://github.com/efficios/babeltrace.git
cd babeltrace
# This is a known working commit.
# Recent commits seem to be broken for perf-ctf
git checkout 9aac8f729c091ddddb688038f5d417a7b1ce4259
./bootstrap
./configure
make
sudo make install
```

Setup the tools on your host: perf

```
cd kernel_source/tools/perf  
make LIBBABELTRACE=1 LIBBABELTRACE_DIR=/usr/local
```

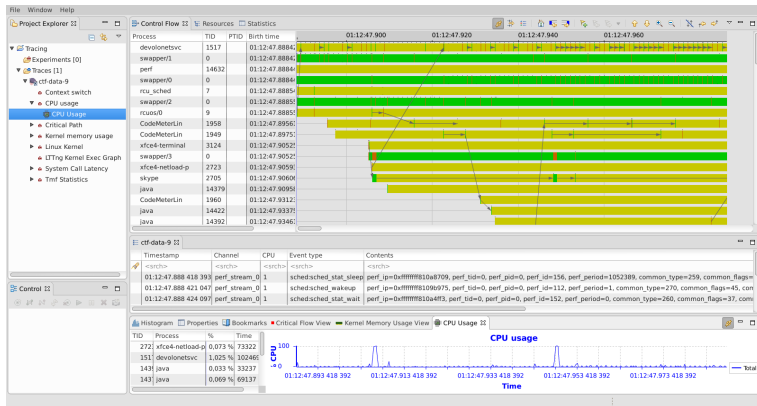

Record a trace on the target

```
./perf record -e 'sched:*' -a  
# (stop with Ctrl-C)  
# Copy perf.data to the host
```

On the host: Convert perf.data to the proper format

```
LD_LIBRARY_PATH=/usr/local/lib ./perf data convert --to-ctf ./ctf-data  
# Now the trace data should be available in ctf-data/  
# You can import this directory with Eclipse / Tracecompass
```

Tracecompass



sources



<http://lwn.net/Articles/365835/>



<http://lwn.net/Articles/366796/>