Ftrace

Important Notice : Courseware - Legal

This courseware is both the product of the author and of freely available opensource and/or public domain materials. Wherever external material has been shown, it's source and ownership have been clearly attributed. We acknowledge all copyrights and trademarks of the respective owners.

The contents of the **courseware PDFs are considered proprietary** and thus cannot be copied or reproduced in any form whatsoever without the explicit written consent of the author.

Only the programs - **source code** and binaries (where applicable) - that form part of this courseware, and that are made available to the participant, are released under the terms of the permissive **MIT license**.
Under the terms of the MIT License, you can certainly use the source code provided here; you must just attribute the original source (author of this courseware and/or other copyright/trademark holders).

VERY IMPORTANT :: Before using this source(s) in your project(s), you *MUST* check with your organization's legal staff that it is appropriate to do so.

The courseware PDFs are *not* under the MIT License, they are to be kept confidential, non-distributable without consent, for your private internal use only.

The duration, contents, content matter, programs, etc. contained in this courseware and companion participant VM are subject to change at any point in time without prior notice to individual participants.

Care has been taken in the preparation of this material, but there is no warranty, expressed or implied of any kind, and we can assume no responsibility for any errors or omisions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

2000-2024 Kaiwan N Billimoria
kaiwanTECH, Bangalore, India.

Resources

- **https://www.kernel.org/doc/html/latest/trace/ftrace.html#the-file-system**
  An explanatory note on *all* (pseudo)files within the tracefs filesystem appears here!

- [LWN, Steven Rostedt]
  - Debugging the kernel using Ftrace - part 1
  - Debugging the kernel using Ftrace – part 2
  - Secrets of the Ftrace function tracer

- Ftrace Favorites Cheat Sheet - Fun Commands to Try with Ftrace
- Kernel Tracing Cheat Sheet
- https://github.com/goldshtn/linux-tracing-workshop [tutorial: try various tracing tools]

Configure in kernel 'make menuconfig':

```
CONFIG_FTRACE : Kernel Hacking / Tracers / ...
```

kernel/trace/Kconfig

```
menuconfig FTRACE
        bool "Tracers"
        depends on TRACING_SUPPORT
        default y if DEBUG_KERNEL
        help
          Enable the kernel tracing infrastructure.
…
```

Points

- to see ftrace on a particular CPU core # 'N', look under
  `/sys/kernel/debug/tracing/per_cpu/cpuN/trace`

<< Mostly extracted from Documentation/tracing/ftrace.txt >>

Introduction

Ftrace is an internal tracer designed to help out developers and designers of systems to find what is going on inside the kernel.
It can be used for debugging or analyzing latencies and performance issues that take place outside of user-space.

Although ftrace is the function tracer, it also includes an infrastructure that allows for other types of tracing. Some of the tracers that are currently in ftrace include a tracer to trace context switches, the time it takes for a high priority task to run after it was woken up, the time interrupts are disabled, and more (ftrace allows for tracer plugins, which means that the list of tracers can always grow).

The File System

Ftrace uses the debugfs file system to hold the control files as well as the files to display output.

When debugfs is configured into the kernel (which selecting any ftrace option will do) the directory */sys/kernel/debug* will be created. To mount this directory, you can add to your */etc/fstab* file:

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

Or you can mount it at run time with:

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

For quicker access to that directory you may want to make a soft link to
it:

```
sudo ln –s /sys/kernel/debug /debug
```

Any selected ftrace option will also create a directory called tracing within the debugfs. The rest of the document will assume that you are in the ftrace directory (cd /sys/kernel/debug/tracing) and will only concentrate on the files within that directory and not distract from the content with the extended "/sys/kernel/debug/tracing" path name.

That's it! (assuming that you have ftrace configured into your kernel)

After mounting the debugfs, you can see a directory called "**tracing**". This directory contains the control and output files of ftrace.

```
$ uname -r
4.2.0-42-generic
$ mount |grep debugfs
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
$ sudo ls /sys/kernel/debug/tracing
[sudo] password for seawolf: xxxxxxxxxx
available_events      free_buffer              README               snapshot
trace_stat
available_filter_functions  function_profile_enabled  saved_cmdlines
stack_max_size      tracing_cpumask
available_tracers     instances                saved_cmdlines_size  stack_trace
             tracing_max_latency
buffer_size_kb              kprobe_events      set_event
stack_trace_filter  tracing_on
buffer_total_size_kb        kprobe_profile           set_ftrace_filter    trace
             tracing_thresh
current_tracer              max_graph_depth          set_ftrace_notrace
trace_clock        uprobe_events
dyn_ftrace_total_info       options            set_ftrace_pid       trace_marker
uprobe_profile
enabled_functions     per_cpu            set_graph_function   trace_options
events                      printk_formats           set_graph_notrace
trace_pipe
$
```

<<

A first tracing session with (raw) Ftrace

```
# cd /sys/kernel/tracing
# cat tracing_on
1
# cat current_tracer
nop

# echo 0 > tracing_on
# cat available_tracers
blk function_graph wakeup_dl wakeup_rt wakeup preemptirqsoff preemptoff irqsoff
function nop
#
# echo function_graph > current_tracer
# echo 1 > options/funcgraph-proc
# echo 1 > options/latency-format
#
# echo 1 > tracing_on ; sleep 1; echo 0 > tracing_on
<tracing everyhting in kernel-space ... for 1 s>

# cp trace /tmp/trc
# ls -lh /tmp/trc
-rw-r----- 1 root root 13M Dec 17 07:32 /tmp/trc
# vi /tmp/trc
...
```

>>

Here is a list of some of the key files:

Note: all time values are in microseconds.

current_tracer:

This is used to set or display the current tracer that is configured.

```
# cat current_tracer
nop
#
```

available_tracers:

This holds the different types of tracers that have been compiled into the kernel. The tracers listed here can be configured by echoing their name into current_tracer.

```
# cat available_tracers
blk function_graph mmiotrace wakeup_rt wakeup function sched_switch nop
#
```

tracing_enabled:

This sets or displays whether the current_tracer is activated and tracing or not. Echo 0 into this file to disable the tracer or 1 to enable it.

```
# cat tracing_enabled
1
#
```

trace:

This file holds the output of the trace in a human readable format (described below).

…

trace_options:

This file lets the user control the amount of data that is displayed in one of the above output files.

…

set_ftrace_filter:

When dynamic ftrace is configured in (see the section below "dynamic ftrace"), the code is dynamically modified (code text rewrite) to disable calling of the function profiler (mcount). This lets tracing be configured in with practically no overhead in performance. This also has a side effect of enabling or disabling specific functions to be traced. Echoing names of functions into this file will limit the trace to only those functions.

This interface also allows for commands to be used. See the “Filter commands" section for more details.

```
# cat set_ftrace_filter
#### all functions enabled ####
#
```

set_ftrace_notrace:

This has an effect opposite to that of set_ftrace_filter. Any function that is added here will not be traced. If a function exists in both set_ftrace_filter and set_ftrace_notrace, the function will _not_ be traced.

```
# cat set_ftrace_notrace
#
```

set_ftrace_pid:

Have the function tracer only trace a single thread.

```
# cat set_ftrace_pid
no pid
#
```

set_graph_function:

Set a "trigger" function where tracing should start with the function graph tracer (See the section “dynamic ftrace" for more details).

```
# cat set_graph_function
#### all functions enabled ####
#
```

available_filter_functions:

This lists the functions that ftrace has processed and can trace. These are the function names that you can pass to "set_ftrace_filter" or "set_ftrace_notrace". (See the section "dynamic
ftrace" below for more details.)

```
# cat available_filter_functions
_stext
do_one_initcall
cpumask_weight
run_init_process
init_post
name_to_dev_t
create_dev
T.763
create_dev
trace_kmalloc

--snip--

rfcomm_tty_open
rfcomm_get_dev_info
rfcomm_wfree
rfcomm_dev_del
rfcomm_tty_hangup
rfcomm_dev_state_change
rfcomm_release_dev
rfcomm_tty_close
rfcomm_get_dev_list
rfcomm_dev_ioctl
#
```

The Tracers

Here is the list of current tracers that may be configured.
Echo one of these values into the file 'current_tracer' for it to take effect.

"function"

Function call tracer to trace all kernel functions.

"function_graph"

Similar to the function tracer except that the function tracer probes the functions on their entry whereas the function graph tracer traces on both entry and exit of the functions. It then provides the ability to draw a graph of function calls similar to C code source.

"sched_switch"

Traces the context switches and wakeups between tasks.

"irqsoff"

Traces the areas that disable interrupts and saves the trace with the longest max latency. See tracing_max_latency. When a new max is recorded, it replaces the old trace. It is best to view this trace with the latency-format option enabled.

"preemptoff"

Similar to irqsoff but traces and records the amount of time for which preemption is disabled.

"preemptirqsoff"

Similar to irqsoff and preemptoff, but traces and records the largest time for which irqs and/or preemption is disabled.

"wakeup"

Traces and records the max latency that it takes for the highest priority task to get scheduled after it has been woken up.

"hw-branch-tracer"

Uses the BTS CPU feature on x86 CPUs to traces all branches executed.

"nop"

This is the "trace nothing" tracer. To remove all tracers from tracing simply echo "nop" into current_tracer.

Ftrace Mini-HOWTO

<<

```
0 /sys/kernel/debug/tracing> cat README
tracing mini-HOWTO:

# mount -t debugfs nodev /sys/kernel/debug
# cat /sys/kernel/debug/tracing/available_tracers
wakeup preemptirqsoff preemptoff irqsoff function sched_switch nop

# cat /sys/kernel/debug/tracing/current_tracer
nop
# echo sched_switch > /sys/kernel/debug/tracing/current_tracer
# cat /sys/kernel/debug/tracing/current_tracer
sched_switch
# cat /sys/kernel/debug/tracing/trace_options
noprint-parent nosym-offset nosym-addr noverbose
# echo print-parent > /sys/kernel/debug/tracing/trace_options
# echo 1 > /sys/kernel/debug/tracing/tracing_enabled
# cat /sys/kernel/debug/tracing/trace > /tmp/trace.txt
# echo 0 > /sys/kernel/debug/tracing/tracing_enabled
0 /sys/kernel/debug/tracing>
```

>>

Examples of using the tracer

Here are typical examples of using the tracers when controlling them only with the debugfs interface (without using any user-land utilities).

Output format:

Here is an example of the output format of the file "trace"

```
                              --------
# tracer: function
#
#           TASK-PID   CPU#    TIMESTAMP  FUNCTION
#              | |      |          |         |
            bash-4251  [01] 10152.583854: path_put <-path_walk
            bash-4251  [01] 10152.583855: dput <-path_put
            bash-4251  [01] 10152.583855: _atomic_dec_and_lock <-dput
                              --------
```

A header is printed with the tracer name that is represented by the trace. In this case the tracer is "function". Then a header showing the format. Task name "bash", the task PID "4251", the CPU that it was running on "01", the timestamp in <secs>.<usecs> format, the function name that was traced "path_put" and the parent function that called this function "path_walk". The timestamp is the time at which the function was entered.

The sched_switch tracer also includes tracing of task wakeups and context switches.

```
  ksoftirqd/1-7     [01]  1453.070013:      7:115:R   +  2916:115:S
  ksoftirqd/1-7     [01]  1453.070013:      7:115:R   +    10:115:S
  ksoftirqd/1-7     [01]  1453.070013:      7:115:R ==>    10:115:R
     events/1-10    [01]  1453.070013:     10:115:S ==>  2916:115:R
  kondemand/1-2916  [01]  1453.070013:   2916:115:S ==>     7:115:R
  ksoftirqd/1-7     [01]  1453.070013:      7:115:S ==>     0:140:R
```

Wake ups are represented by a "+" and the context switches are shown as "==>". The format is:

Context switches:

Previous task Next Task

```
<pid>:<prio>:<state>  ==>  <pid>:<prio>:<state>
```

Wake ups:

Current task Task waking up

```
<pid>:<prio>:<state>    +  <pid>:<prio>:<state>
```

The prio is the internal kernel priority, which is the inverse of the priority that is usually displayed by user-space tools.
Zero represents the highest priority (99). Prio 100 starts the "nice" priorities with 100 being equal to nice -20 and 139 being nice 19. The prio "140" is reserved for the idle task which is the lowest priority thread (pid 0).

--*snip*--

function

This tracer is the function tracer. Enabling the function tracer can be done from the debug file system. Make sure the *ftrace_enabled* is set; otherwise this tracer is a nop.

KEY POINT!

The filters can also select only those functions that belong to a specific module by using the 'mod' command in the input to the filter file:

```
   [tracing]# echo ':mod:tg3' > set_ftrace_filter
   [tracing]# cat set_ftrace_filter |head -8
   tg3_write32
   tg3_read32
   tg3_write_flush_reg32
   tw32_mailbox_flush
   tg3_write32_tx_mbox
   tg3_read32_mbox_5906
   tg3_write32_mbox_5906
   tg3_disable_ints
```

This is very useful if you are debugging a single module, and you only want to see the functions that belong to that module in the trace.

Another eg. – tracing the network driver:

```
# echo ':mod:e1000e' > set_ftrace_filter
…
```

Ftrace cheatsheet :: interpreting the symbols

```
#                         _-----=> irqs-off
#                       / _----=> need-resched
#                      | / _---=> hardirq/softirq
#                      || / _--=> preempt-depth
#                      ||| /
# CPU   TASK/PID       ||||      DURATION                   FUNCTION CALLS
# |      |    |        ||||      |   |                      |   |   |   |
  0)     ps-5268   |  .....  |      0.157 us  |               } /* seq_puts */
  0)   mycp-786    |  nd.h1  | !  175 us      |     } /* sys_write */
 ^          ^      |   ^     |       ^        |          ^
 cpu#  thread-PID    5/4 cols      time delay      function name
                    \see below/
```

Symbol in (raw) ftrace output	*Meaning*
`'+'`	a *wakeup* has occurred
`==> (or =>)`	a *context switch* has occurred
`==========>`	switch *to an interrupt* context (usually a hard irq; shows on ARM)
`<==========`	switch back *from an interrupt* context to process context; shows on ARM
Time (delay) nomenclature ; eg. `@ 175928.1 us`	
`'@'`	> 100,000 us (100 ms)
`'*'`	> 10,000 us (10 ms)
`'#'`	> 1,000 us (1 ms)
`'!'`	> 100 us (*preempt_mark_thresh)*
`'+'`	> 10 us
`' '`	<= 1 us (1 microsecond)
Latency Trace Format (the four/five columns): eg. `1d.h1`	
First column (CPU # when using *trace-cmd(1)*, else not present)	*(If not in v first col)* `n: CPU #` the thread / interrupt was running upon
Second column	`'.'` = interrupts enabled; `'d'` = interrupts disabled
Third column	need-resched; `'.'` = unset, `'N' = TIF_NEED_RESCHED` bit has been set
Fourth column	`'.'` = process ctx (context); `'h' / 'H'` = hard-irq interrupt ctx ; `'s'` = softirq interrupt ctx (`'h'` = hard irq is running ; `'H'` = hard irq occurred inside a softirq)

Fifth column	preempt-depth; 0 = no locks held, +ve = that many locks are being held

Use **trace-cmd** :

- with option switch `-l` to show the 5 char latency format
- without option switch `-p <plugin>` to show all function's parameters along with their values.

```
 # echo > /debug/tracing/trace       # clear trace buffer
 # sysctl kernel.ftrace_enabled=1
 # echo function > current_tracer
 # echo 1 > tracing_enabled
 # usleep 1
 # echo 0 > tracing_enabled
 # cat trace
# tracer: function
#
#           TASK-PID   CPU#    TIMESTAMP  FUNCTION
#              | |      |          |         |
            bash-4003  [00]   123.638713: finish_task_switch <-schedule
            bash-4003  [00]   123.638714: _spin_unlock_irq <-finish_task_switch
            bash-4003  [00]   123.638714: sub_preempt_count <-_spin_unlock_irq
            bash-4003  [00]   123.638715: hrtick_set <-schedule
            bash-4003  [00]   123.638715: _spin_lock_irqsave <-hrtick_set
            bash-4003  [00]   123.638716: add_preempt_count <-_spin_lock_irqsave
            bash-4003  [00]   123.638716: _spin_unlock_irqrestore <-hrtick_set
            bash-4003  [00]   123.638717: sub_preempt_count <-
_spin_unlock_irqrestore
            bash-4003  [00]   123.638717: hrtick_clear <-hrtick_set
            bash-4003  [00]   123.638718: sub_preempt_count <-schedule
            bash-4003  [00]   123.638718: sub_preempt_count <-preempt_schedule
            bash-4003  [00]   123.638719: wait_for_completion <-
__stop_machine_run
            bash-4003  [00]   123.638719: wait_for_common <-wait_for_completion
            bash-4003  [00]   123.638720: _spin_lock_irq <-wait_for_common
            bash-4003  [00]   123.638720: add_preempt_count <-_spin_lock_irq
[...]
```

--*snip*--

SIDEBAR :: A helper script to try ftrace

<<
Recent: try helper scripts for tracing from my *Linux Kernel Debugging* book's FOSS repo:
https://github.com/PacktPublishing/Linux-Kernel-Debugging/tree/main/ch9
>>

The code of https://github.com/PacktPublishing/Linux-Kernel-Debugging/blob/main/ch9/ftrace/ftrc_1s.sh follows:

```
#!/bin/bash
# ch9/ftrace/ftrc_1s.sh
# ****************************************************************
# This program is part of the source code released for the book
#  "Linux Kernel Debugging"
#  (c) Author: Kaiwan N Billimoria
#  Publisher:  Packt
#  GitHub repository:
#  https://github.com/PacktPublishing/Linux-Kernel-Debugging
#
# From: Ch 9: Tracing the kernel flow
#****************************************************************
# Brief Description:
# Very simple (raw) usage of kernel ftrace; traces whatever executes 
within
# the kernel for 1 second.
#
# For details, please refer the book, Ch 9.
#-------------------------------------------------------------------------
------
name=$(basename $0)
[ $(id -u) -ne 0 ] && {
  echo "${name}: needs root."
  exit 1
}
source $(dirname $0)/ftrace_common.sh || {
 echo "Couldn't source required file $(dirname $0)/ftrace_common.sh"
 exit 1
}
REPDIR=~/ftrace_reports
FTRC_REP=${REPDIR}/${name}_$(date +%Y%m%d_%H%M%S).txt

cd /sys/kernel/tracing
reset_ftrace   # here: https://github.com/PacktPublishing/Linux-Kernel-
Debugging/blob/main/ch9/ftrace/ftrace_common.sh

grep -q -w function_graph available_tracers || die "tracer specified 
function_graph unavailable"
echo function_graph > current_tracer || die "setting function_graph 
plugin failed"
echo 1 > options/funcgraph-proc
echo 1 > options/latency-format

echo "Tracing with function_graph for 1s ..."
echo 1 > tracing_on ; sleep 1 ; echo 0 > tracing_on
mkdir -p ${REPDIR} 2>/dev/null
cp trace ${FTRC_REP}
```

```
ls -lh ${FTRC_REP}
exit 0
```

Sample Usage:

```
$ ./ftrc_1s.sh
ftrc_1s.sh: needs root.
$ sudo ./ftrc_1s.sh
[sudo] password for osboxes:
resetting set_ftrace_filter
resetting set_ftrace_notrace
resetting set_ftrace_notrace_pid
resetting set_ftrace_pid
resetting trace_options to defaults (as of 5.10.60)
resetting options/funcgraph-*
Tracing with function_graph for 1s ...
-rw-r----- 1 root root 13M Nov 15 05:20
/root/ftrace_reports/ftrc_1s.sh_20221115_052002.txt
$
$ sudo cat <...>/ftrc_1s.sh_20221115_052002.txt
# tracer: function_graph
#
# function_graph latency trace v1.1.5 on 5.15.0-52-generic
# --------------------------------------------------------------------
# latency: 0 us, #253681/680734, CPU#0 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:6)
#    -----------------
#    | task: -0 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#
#                        _-----=> irqs-off
#                       / _----=> need-resched
#                      | / _---=> hardirq/softirq
#                      || / _--=> preempt-depth
#                      ||| /
# CPU  TASK/PID        ||||     DURATION                  FUNCTION CALLS
# |     |    |         ||||      |   |                     |   |   |   |
 0)  llvmpip-1838  |  ..... |               |  find_vma() {
 0)  llvmpip-1838  |  ..... |   0.122 us    |    vmacache_find();
 0)  llvmpip-1838  |  ..... |   0.647 us    |  }
 0)  llvmpip-1838  |  ..... |               |  handle_mm_fault() {
...
 0)    <idle>-0    |  d.... |   1.261 us    |      }
 ------------------------------------------
 0)    <idle>-0    =>  llvmpip-1838                << a context switch! >>
 ------------------------------------------

 0)  llvmpip-1838  |  d.... |               |
finish_task_switch.isra.0() {
 0)  llvmpip-1838  |  d.... |   0.139 us    |              raw_spin_rq_unlock();
 0)  llvmpip-1838  |  d.... |   0.158 us    |              irq_enter_rcu();
 0)  llvmpip-1838  |  d.h.. |               |
__sysvec_apic_timer_interrupt() {
 0)  llvmpip-1838  |  d.h.. |               |                hrtimer_interrupt()
{
 0)  llvmpip-1838  |  d.h.. |   0.153 us    |
_raw_spin_lock_irqsave();
 0)  llvmpip-1838  |  d.h.. |   0.197 us    |
ktime_get_update_offsets_now();
 0)  llvmpip-1838  |  d.h.. |               |
__hrtimer_run_queues() {
 0)  llvmpip-1838  |  d.h.. |   0.304 us    |
__remove_hrtimer();
 0)  llvmpip-1838  |  d.h.. |   0.158 us    |
_raw_spin_unlock_irqrestore();
 0)  llvmpip-1838  |  d.h.. |               |
tick_sched_timer() {
 0)  llvmpip-1838  |  d.h.. |   0.167 us    |                   ktime_get();
 0)  llvmpip-1838  |  d.h.. |               |
tick_sched_handle() {
 0)  llvmpip-1838  |  d.h.. |               |
update_process_times() {
 0)  llvmpip-1838  |  d.h.. |               |
account_process_tick() {
 0)  llvmpip-1838  |  d.h.. |               |
account_system_time() {
...
 0)  llvmpip-1838  |  d.h.. |               |           irq_exit_rcu() {
 0)  llvmpip-1838  |  d.... |               |             __do_softirq() {
 0)  llvmpip-1838  |  ..s.. |               |
run_timer_softirq() {
 0)  llvmpip-1838  |  ..s.. |   0.158 us    |
_raw_spin_lock_irq();
 0)  llvmpip-1838  |  d.s.. |   0.587 us    |
__next_timer_interrupt();
 0)  llvmpip-1838  |  ..s.. |               |                 call_timer_fn()
{
 0)  llvmpip-1838  |  ..s.. |               |
tcp_orphan_update() {
 0)  llvmpip-1838  |  ..s.. |               |                     mod_timer()
{
 0)  llvmpip-1838  |  ..s.. |               |
lock_timer_base() {
 0)  llvmpip-1838  |  ..s.. |   0.152 us    |
_raw_spin_lock_irqsave();
 0)  llvmpip-1838  |  d.s.. |   0.446 us    |                       }
 0)  llvmpip-1838  |  d.s.. |   0.141 us    |
detach_if_pending();
 0)  llvmpip-1838  |  d.s.. |   0.156 us    |
get_nohz_timer_target();
...
 0)  llvmpip-1838  |  d.... |               |  exit_to_user_mode_prepare() {
 0)  llvmpip-1838  |  d.... |   0.169 us    |
fpregs_assert_state_consistent();
 0)  llvmpip-1838  |  d.... |   0.478 us    |  }
 0)  llvmpip-1838  |  ..... |               |  __x64_sys_futex() {
 0)  llvmpip-1838  |  ..... |               |    do_futex() {
...
 0)  llvmpip-1838  |  ..... |               |        wake_up_q() {
 0)  llvmpip-1838  |  ..... |               |          try_to_wake_up() {
 0)  llvmpip-1838  |  ..... |   0.161 us    |
_raw_spin_lock_irqsave();
 0)  llvmpip-1838  |  d.... |               |            select_task_rq_fair() {
```

```
...
 1)  gnome-t-2357  |  ..... |   1.247 us    |            }
 1)  gnome-t-2357  |  ..... |               |            schedule() {
 1)  gnome-t-2357  |  d.... |               |
rcu_note_context_switch() {
 1)  gnome-t-2357  |  d.... |   0.088 us    |                rcu_qs();
 1)  gnome-t-2357  |  d.... |   0.280 us    |              }
 1)  gnome-t-2357  |  d.... |               |
raw_spin_rq_lock_nested() {
 1)  gnome-t-2357  |  d.... |   0.111 us    |                _raw_spin_lock();
 1)  gnome-t-2357  |  d.... |   0.293 us    |              }
 1)  gnome-t-2357  |  d.... |   0.119 us    |              update_rq_clock();
 1)  gnome-t-2357  |  d.... |               |              dequeue_task_fair() {
 1)  gnome-t-2357  |  d.... |               |                dequeue_entity() {
 1)  gnome-t-2357  |  d.... |               |                  update_curr() {
 1)  gnome-t-2357  |  d.... |   0.093 us    |
update_min_vruntime();
...
 1)  gnome-t-2357  |  d.... |               |              pick_next_task() {
 1)  gnome-t-2357  |  d.... |               |
pick_next_task_fair() {
 1)  gnome-t-2357  |  d.... |               |                  newidle_balance()
{
 1)  gnome-t-2357  |  d.... |   0.089 us    |
__msecs_to_jiffies();
 1)  gnome-t-2357  |  d.... |   0.087 us    |
rcu_read_unlock_strict();
 1)  gnome-t-2357  |  d.... |   0.530 us    |                  }
 1)  gnome-t-2357  |  d.... |   0.740 us    |                }
 1)  gnome-t-2357  |  d.... |               |
put_prev_task_fair() {
 1)  gnome-t-2357  |  d.... |   0.111 us    |
put_prev_entity();
 1)  gnome-t-2357  |  d.... |   0.089 us    |
put_prev_entity();
 1)  gnome-t-2357  |  d.... |   0.710 us    |                }
 1)  gnome-t-2357  |  d.... |               |
pick_next_task_idle() {
 1)  gnome-t-2357  |  d.... |               |
set_next_task_idle() {
 1)  gnome-t-2357  |  d.... |   0.087 us    |
queue_core_balance();
 1)  gnome-t-2357  |  d.... |   0.290 us    |                  }
 1)  gnome-t-2357  |  d.... |   0.480 us    |                }
 1)  gnome-t-2357  |  d.... |   2.351 us    |              }
 1)  gnome-t-2357  |  d.... |               |              psi_task_switch() {
...
 1)  gnome-t-2357  |  d.... |               |
save_fpregs_to_fpstate() {
 1)  gnome-t-2357  |  d.... |   0.090 us    |
xfd_validate_state();
 1)  gnome-t-2357  |  d.... |   0.354 us    |              }
 ------------------------------------------
 1)  gnome-t-2357  =>    <idle>-0
 ------------------------------------------

 1)    <idle>-0    |  d.... |               |      finish_task_switch.isra.0() {
 1)    <idle>-0    |  d.... |   0.091 us    |        raw_spin_rq_unlock();
```

...

and so on and on…

Great, BUT….. far too much noise!
We need to filter the ftrace raw output!

Help for interpreting the output of the ftrace latency output format

```
# ---------------------------------------------------
# Help for interpreting the output of the latency:
# from Documentation/trace/ftrace.txt
--snip--
# The next lines after the header are the trace itself. The header
# explains which is which.
#
#   cmd: The name of the process in the trace.
#
#   pid: The PID of that process.
#
#   CPU#: The CPU which the process was running on.
#
#   irqs-off: 'd' interrupts are disabled. '.' otherwise.
#           Note: If the architecture does not support a way to
#                 read the irq flags variable, an 'X' will always
#                 be printed here.
#
#  need-resched: 'N' task need_resched is set, '.' otherwise.
#
#   hardirq/softirq:
#       'H' - hard irq occurred inside a softirq.
#       'h' - hard irq is running
#       's' - soft irq is running
#       '.' - normal context.
#
#   preempt-depth: The level of preempt_disabled
#
# The above is mostly meaningful for kernel developers.
#
#   time: When the latency-format option is enabled, the trace file
#       output includes a timestamp relative to the start of the
#       trace. This differs from the output when latency-format
#       is disabled, which includes an absolute timestamp.
#
#  delay: This is just to help catch your eye a bit better. And
#        needs to be fixed to be only relative to the same CPU.
#        The marks are determined by the difference between this
#        current trace and the next trace.
#          '!' - greater than preempt_mark_thresh (default 100)
#          '+' - greater than 1 microsecond
#          ' ' - less than or equal to 1 microsecond.
#
#  The rest is the same as the 'trace' file.
#
```

```
--snip--
```

Eg 1 : A sample ftrace session on an ARM v7 (emulated using QEMU) running Linux 3.10.24 :

```
--snip--

# tracer: function_graph
#
# function_graph latency trace v1.1.5 on 3.10.24
# --------------------------------------------------------------------
# latency: 0 us, #44942/144640, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0
#P:1)
#    -----------------
#    | task: -0 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#
#                        _-----=> irqs-off
#                       / _----=> need-resched
#                      | / _---=> hardirq/softirq
#                      || / _--=> preempt-depth
#                      ||| /
# CPU  TASK/PID        ||||  DURATION                  FUNCTION CALLS
# |     |    |         ||||   |   |                      |   |   |   |
           << unfortunately, the indented function names wrap around
sometimes >>
  0)   sleep-533    |  d.s5  3.292 us    |      << notice we're in softirq
context >>
wakeup_preempt_entity();
  0)   sleep-533    |  d.s5  4.292 us    |
resched_task();
  0)   sleep-533    |  dNs5! 169.791 us  | << notice the TIF_NEED_RESCHED flag
set
                                       implying we need to reschedule ASAP!
>>                                                    }
  0)   sleep-533    |  dNs5! 224.209 us  |
}
  0)   sleep-533    |  dNs5! 279.875 us  |
}
  0)   sleep-533    |  dNs5! 833.375 us  |
}
  0)   sleep-533    |  dNs5              |
_raw_spin_unlock() {
  0)   sleep-533    |  dNs5  2.583 us    |
sub_preempt_count();
  0)   sleep-533    |  dNs4+ 52.416 us   | }

--snip--

  0)   sleep-533    |  dN.2! 6794.292 us |                      }
  0)   sleep-533    |  dN.2! 6946.542 us |                    }
  0)   sleep-533    |   <========== |       << the arrow denotes a switch of ctx
>>
```

```
  0)   sleep-533    |  .N..                |                      __schedule() {
  0)   sleep-533    |  .N..  1.667 us     |
add_preempt_count();
  0)   sleep-533    |  .N.1  4.167 us     |
rcu_note_context_switch();
  0)   sleep-533    |  .N.1               |
_raw_spin_lock_irq() {
  0)   sleep-533    |  dN.1  2.209 us     |
add_preempt_count();
  0)   sleep-533    |  dN.2+ 55.833 us    |                        }
  0)   sleep-533    |  dN.2  1.750 us     |
update_rq_clock();
  0)   sleep-533    |  dN.2               |
put_prev_task_fair() {
  0)   sleep-533    |  dN.2  2.292 us     |
update_curr();
```

```
                        _-----=> irqs-off
                       / _----=> need-resched
                      | / _---=> hardirq/softirq
                      || / _--=> preempt-depth
                      ||| /
CPU  TASK/PID         ||||  DURATION                    FUNCTION CALLS
 |     |    |         ||||   |   |                       |   |   |   |

 0)   sleep-533    |  dN.2  2.875 us     |
__enqueue_entity();
 0)   sleep-533    |  dN.2! 104.833 us  |                        }
 0)   sleep-533    |  dN.2               |
pick_next_task_fair() {
 0)   sleep-533    |  dN.2  2.750 us     |
wakeup_preempt_entity();
 0)   sleep-533    |  dN.2  2.833 us     |
clear_buddies();
 0)   sleep-533    |  dN.2  4.583 us     |
__dequeue_entity();
 0)   sleep-533    |  dN.2! 165.083 us  |                        }
 0)   sleep-533    |  d..2               | atomic_notifier_call_chain() {
 0)   sleep-533    |  d..2               |  << TIF_NEED_RESCHED flag cleared >>
__atomic_notifier_call_chain() {
 0)   sleep-533    |  d..2  1.958 us     |
__rcu_read_lock();
 0)   sleep-533    |  d..2               |
notifier_call_chain() {
 0)   sleep-533    |  d..2  2.875 us     |
vfp_notifier();
 0)   sleep-533    |  d..2+ 49.208 us   |                            }
 0)   sleep-533    |  d..2  2.250 us     |
__rcu_read_unlock();
 0)   sleep-533    |  d..2! 189.292 us  |                          }
 0)   sleep-533    |  d..2! 233.875 us  |                        }
 ------------------------------------------
 0)   sleep-533    =>  kworker-302         << ctx switch has taken place! >>
 ------------------------------------------
 0)  kworker-302   |  d..2                 |      finish_task_switch() {
 0)  kworker-302   |  d..2                 |        _raw_spin_unlock_irq() {

--snip--
```

Example 2 of using Ftrace – within a shell script

```
...
net_ftrc()
{
# Set to 1 to use the 'eXclude these functions' approach;
# set to 0 (default) to use the 'Include these functions only' approach.
# With the 'eXclude' approach, you will probably gather more detail at the
 # cost of getting extraneous info.. With the 'Include only' approach, you
 # will probably gather only what you *think* is important, possibly missing
 # out on some functions. As usual, it's a tradeoff :)
 USE_EXCLUDE_APPROACH=0

<<
Practically and/or empirically speaking, the 'Include Only these functions'
approach seems to work better; see the output above with both approaches..
>>

 echo 20480 > buffer_size_kb
 echo function_graph > current_tracer
 # show process name/PID pair in the trace
 echo funcgraph-proc > trace_options

 # turn ON latency-format
 echo 1 > options/latency-format

 if [ ${USE_EXCLUDE_APPROACH} -eq 1 ]; then
  echo "---------> eXcluding functions, pl wait (takes a while!)... <---------"

  # eXclude these fns
  echo find_get_page 'ktime*' 'get_page*' 'vm_*' 'unlink_*' 'memblock*'
'filemap*' 'pfn*' '*wake*' 'next_zone*' 'prepare_bin*' 'copy_strings*' 'lru*'
'__get_free*' 'l2x0*' '*exit*' 'ptep*' 'anon_vma*' '__sync*' 'unlock_*' 'kmem*'
'open_exec' 'mark_page*' 'do_mmap*' 'do_mun*' 'search_binary*' unlock_page
'rcu*' 'uart_*' '__do_fault' 'page_*' do_wp_page 'zone_*' mmput 'arch_*' 'elf*'
'*execve*' handle_IRQ exit_mm 'handle_pte*' 'find_vma' '__might*' 'T*' 'unmap*'
'free_pg*' 'flush_*' '__alloc*' 'v6*' '*tty*' 'handle_mm*' 'up*' 'down*'
'*spin*' 'asm*' 'do_page*' '*irq*' '*timer*' 'sched*' '*sched*' '*tick*'
'*mall*' 'spin*' '*lock*' '*page*' '*probe*' '*trace*'  >> set_ftrace_notrace

else    # the 'Include Approach'   - Better!
 echo "---------> Including functions to ftrace, pl wait (takes a while!)...
<---------"
 echo `grep -i net /sys/kernel/debug/tracing/available_filter_functions` >>
set_ftrace_filter
 echo `grep -i tcp /sys/kernel/debug/tracing/available_filter_functions` >>
set_ftrace_filter
 echo `grep -i udp /sys/kernel/debug/tracing/available_filter_functions` >>
```

```
set_ftrace_filter
 echo `grep -i sock /sys/kernel/debug/tracing/available_filter_functions` >>
set_ftrace_filter
 echo `grep -i '^ip' /sys/kernel/debug/tracing/available_filter_functions` >>
set_ftrace_filter
 echo `grep -i xmit /sys/kernel/debug/tracing/available_filter_functions` >>
set_ftrace_filter

fi

 # trace
 echo 1 > tracing_on ; <...>/xtalker_dgram 192.168.2.10 "hello, goat" ; echo 0
> tracing_on
 cp trace /tmp/trc.txt
ls -lh /tmp/trc*
}
…
```

Sample output:

1. With the 'Include Only' Approach:

```
<< trace file size was just ~ 6.5 KB >>

# cat /tmp/trc.txt
…
#
#                       _-----=> irqs-off
#                      / _----=> need-resched
#                     | / _---=> hardirq/softirq
#                     || / _--=> preempt-depth
#                     ||| /
# CPU  TASK/PID       ||||  DURATION                  FUNCTION CALLS
# |     |    |        ||||   |   |                     |   |   |   |
 0)  xtalker-568   |  ....               | sys_socket() {
 0)  xtalker-568   |  ....               |   sock_create() {
 0)  xtalker-568   |  ....               |     __sock_create() {
 0)  xtalker-568   |  ....               |       sock_alloc() {
 0)  xtalker-568   |  ....+ 16.250 us    |         sock_alloc_inode();
 0)  xtalker-568   |  ....+ 60.166 us    |       }
 0)  xtalker-568   |  ....               |       inet_create() {
 0)  xtalker-568   |  ....  7.583 us     |         sock_update_classid();
 0)  xtalker-568   |  ....+ 21.375 us    |         sock_init_data();
 0)  xtalker-568   |  ....! 120.625 us   |       }
 0)  xtalker-568   |  ....! 231.041 us   |     }
 0)  xtalker-568   |  ....! 258.458 us   |   }
 0)  xtalker-568   |  ....               |   sock_map_fd() {
 0)  xtalker-568   |  ....+ 40.417 us    |     sock_alloc_file();
 0)  xtalker-568   |  ....+ 68.542 us    |   }
 0)  xtalker-568   |  ....! 394.541 us   | }
 0)  xtalker-568   |  ....+ 24.500 us    | sockfd_lookup_light();
 0)  xtalker-568   |  ....               | sock_sendmsg() {
 0)  xtalker-568   |  ....  7.500 us     |   sock_update_classid();
 0)  xtalker-568   |  ....               |   inet_sendmsg() {
 0)  xtalker-568   |  ....               |     inet_autobind() {
```

```
 0)  xtalker-568   |  ....+ 62.750 us   |        lock_sock_nested();
 0)  xtalker-568   |  ....              |        udp_v4_get_port() {
 0)  xtalker-568   |  ....  5.833 us    |          udp4_portaddr_hash();
 0)  xtalker-568   |  ....  4.834 us    |          udp4_portaddr_hash();
 0)  xtalker-568   |  ....              |          udp_lib_get_port() {
 0)  xtalker-568   |  ....  6.208 us    |
inet_get_local_port_range();
 0)  xtalker-568   |  ..s1  6.667 us    |            sock_prot_inuse_add();
 0)  xtalker-568   |  ....! 125.584 us  |          }
 0)  xtalker-568   |  ....! 186.541 us  |        }
 0)  xtalker-568   |  ....+ 10.208 us   |        release_sock();
 0)  xtalker-568   |  ....! 319.542 us  |      }
 0)  xtalker-568   |  ....              |      udp_sendmsg() {
 0)  xtalker-568   |  ....  5.916 us    |        sock_tx_timestamp();
 0)  xtalker-568   |  ....              |        ip_route_output_flow() {
 0)  xtalker-568   |  ....  9.333 us    |          inet_getpeer();
 0)  xtalker-568   |  ....  5.625 us    |          ipv4_mtu();
 0)  xtalker-568   |  ..s1              |          ipv4_neigh_lookup() {
 0)  xtalker-568   |  ..s1  8.250 us    |            inet_addr_type();
 0)  xtalker-568   |  ..s1! 141.042 us  |          }
 0)  xtalker-568   |  ....! 422.208 us  |        }
 0)  xtalker-568   |  ....              |        ip_make_skb() {
 0)  xtalker-568   |  ....              |          ip_setup_cork() {
 0)  xtalker-568   |  ....  4.833 us    |            ipv4_mtu();
 0)  xtalker-568   |  ....+ 15.708 us   |          }
 0)  xtalker-568   |  ....              |          sock_alloc_send_skb() {
 0)  xtalker-568   |  ....+ 70.917 us   |            sock_alloc_send_pskb();
 0)  xtalker-568   |  ....+ 93.041 us   |          }
 0)  xtalker-568   |  ....+ 28.875 us   |          ip_generic_getfrag();
 0)  xtalker-568   |  ....  5.083 us    |          ipv4_mtu();
 0)  xtalker-568   |  ....+ 22.416 us   |          ip_cork_release();
 0)  xtalker-568   |  ....! 302.834 us  |        }
 0)  xtalker-568   |  ....              |        udp_send_skb() {
 0)  xtalker-568   |  ....              |          ip_send_skb() {
 0)  xtalker-568   |  ....              |            ip_local_out() {
 0)  xtalker-568   |  ....              |              ip_output() {
 0)  xtalker-568   |  ....              |                ip_finish_output() {
 0)  xtalker-568   |  ....  4.500 us    |                  ipv4_mtu();
 0)  xtalker-568   |  ..s.  9.000 us    |                  inet_addr_type();
 0)  xtalker-568   |  ..s.              |                  arp_xmit() {
 0)  xtalker-568   |  ..s.              |                    dev_queue_xmit( {
 0)  xtalker-568   |  ..s1              |
sch_direct_xmit() {
 0)  xtalker-568   |  ..s1              |
dev_hard_start_xmit() {
 0)  xtalker-568   |  ..s1+ 24.667 us   |
netif_skb_features();
 0)  xtalker-568   |  ..s1! 186.583 us  |
smsc911x_hard_start_xmit();
 0)  xtalker-568   |  ..s1! 267.750 us  |                        }
 0)  xtalker-568   |  ..s1! 298.083 us  |                      }
 0)  xtalker-568   |  ..s.! 328.416 us  |                    }
 0)  xtalker-568   |  ..s.! 352.333 us  |                  }

--snip--

 0)  xtalker-568   |  ....! 2063.625 us |  }
 0)  xtalker-568    |  ....               |  sock_close() {
 0)  xtalker-568    |  ....               |    sock_release() {
 0)  xtalker-568    |  ....               |      inet_release() {
 0)  xtalker-568    |  ....  6.125 us     |        ip_mc_drop_socket();
 0)  xtalker-568    |  ....               |        udp_lib_close() {
 0)  xtalker-568    |  ....               |          udp_destroy_sock() {
 0)  xtalker-568    |  ....+ 51.583 us    |            lock_sock_fast();
 0)  xtalker-568    |  ..s1  5.917 us     |
udp_flush_pending_frames();
 0)  xtalker-568    |  ....+ 98.958 us    |          }
 0)  xtalker-568    |  ....               |          udp_lib_unhash() {
 0)  xtalker-568    |  ..s1  5.750 us     |            sock_prot_inuse_add();
 0)  xtalker-568    |  ....+ 25.042 us    |          }

--snip--

 0)  xtalker-568    |  ....! 308.417 us   |  }
 0)  xtalker-568    |  ....               |  iput() {
 0)  xtalker-568    |  ....+ 12.875 us    |    sock_destroy_inode();
 0)  xtalker-568    |  ....! 186.291 us   |  }
#
```

2. With the 'eXclude' Approach:

<< trace file size was ~ 1 MB ! >>

```
[...]
#                        _-----=> irqs-off
#                       / _----=> need-resched
#                      | / _---=> hardirq/softirq
#                      || / _--=> preempt-depth
#                      ||| /
# CPU  TASK/PID        ||||  DURATION                  FUNCTION CALLS
# |     |    |         ||||   |   |                     |   |   |   |
 0)  net_ftr-610    |  ....+ 19.458 us    |  __fsnotify_parent();
 0)  net_ftr-610    |  ....               |  fsnotify() {
 0)  net_ftr-610    |  ....               |    __srcu_read_lock() {
 0)  net_ftr-610    |  ....  5.500 us     |      add_preempt_count();
 0)  net_ftr-610    |  ...1  4.917 us     |      sub_preempt_count();
 0)  net_ftr-610    |  ....+ 24.542 us    |    }
 0)  net_ftr-610    |  ....               |    __srcu_read_unlock() {
 0)  net_ftr-610    |  ....  4.291 us     |      add_preempt_count();
 0)  net_ftr-610    |  ...1  4.541 us     |      sub_preempt_count();
 0)  net_ftr-610    |  ....+ 21.750 us    |    }
 0)  net_ftr-610    |  ....+ 60.125 us    |  }
 0)  net_ftr-610    |  ....               |  sys_dup2() {
 0)  net_ftr-610    |  ....               |    sys_dup3() {
 0)  net_ftr-610    |  ....  5.541 us     |      add_preempt_count();
 0)  net_ftr-610    |  ...1  4.292 us     |      expand_files();
 0)  net_ftr-610    |  ...1  4.666 us     |      sub_preempt_count();
 0)  net_ftr-610    |  ....               |      filp_close() {
 0)  net_ftr-610    |  ....  4.334 us     |        dnotify_flush();
 0)  net_ftr-610    |  ....  4.167 us     |        locks_remove_posix();
```

```
[...]

 0)  net_ftr-610   |  ...1+ 61.166 us   |            }
 0)  net_ftr-610   |  ...1+ 76.625 us   |          }
------------------------------------------
 0)  net_ftr-610   =>  xtalker-613
------------------------------------------
 0)  xtalker-613   |  ...1              |  schedule_tail() {
 0)  xtalker-613   |  ...1  6.083 us    |    finish_task_switch();
 0)  xtalker-613   |  ...1  7.041 us    |    sub_preempt_count();
 0)  xtalker-613   |  ....              |    __task_pid_nr_ns() {
 0)  xtalker-613   |  ....  4.000 us    |      __rcu_read_lock();
 0)  xtalker-613   |  ....  3.791 us    |      __rcu_read_unlock();
 0)  xtalker-613   |  ....+ 21.083 us   |    }
 0)  xtalker-613   |  d...              |    do_page_fault() {
 0)  xtalker-613   |  d...  5.333 us    |      add_preempt_count();
 0)  xtalker-613   |  d..1  4.458 us    |      sub_preempt_count();
 0)  xtalker-613   |  ....              |      down_read_trylock() {

[...]

 0)  xtalker-613   |  ....              |    down_read_trylock() {
 0)  xtalker-613   |  ....              |      add_preempt_count() {
 0)  xtalker-613   |   ==========> |   << at 5% of the trace file >>
 0)  xtalker-613   |  d..1              |        asm_do_IRQ() {
 0)  xtalker-613   |  d..1              |          irq_enter() {
 0)  xtalker-613   |  d..1+ 17.917 us   |            idle_cpu();
 0)  xtalker-613   |  d..1  4.000 us    |            add_preempt_count();

[...]

 0)  xtalker-613   |  d..1  3.625 us    |    idle_cpu();
 0)  xtalker-613   |  d..1  4.416 us    |    sub_preempt_count();
 0)  xtalker-613   |  d...! 744.000 us  |  }
 0)  xtalker-613   |   <========== |
 0)  xtalker-613   |  ....              |  sys_socket() { << at 78% of the
                                                        trace file !! >>
 0)  xtalker-613   |  ....              |    sock_create() {
 0)  xtalker-613   |  ....              |      __sock_create() {
 0)  xtalker-613   |  ....              |        sock_alloc() {
 0)  xtalker-613   |  ....              |          new_inode_pseudo() {
 0)  xtalker-613   |  ....              |            alloc_inode() {
 0)  xtalker-613   |  ....              |              sock_alloc_inode() {
 0)  xtalker-613   |  ....  4.500 us    | __init_waitqueue_head();

[...]

 0)  xtalker-613   |  ...1  7.833 us    |            __rcu_read_unlock();
 0)  xtalker-613   |  ...1+ 51.583 us   |          }
 0)  xtalker-613   |  ...1+ 65.083 us   |        }
------------------------------------------
 0)  xtalker-613   =>  net_ftr-610
------------------------------------------
 0)  net_ftr-610   |  ...1              |          finish_task_switch() {
 0)  net_ftr-610   |  ...1              |            __mmdrop() {
 0)  net_ftr-610   |  ...1              |              pgd_free() {
```

```
[...]

#
```

<<
Practically and/or empirically speaking, the 'Include Only these functions' approach seems to work better; see the output above with both approaches..
>>

<<
How to Dump Function Call Graph in Linux (using ftrace), Nov 2021
>>

Dynamic ftrace with the function graph tracer

...special features only available in the function-graph tracer.

```
# cat /sys/kernel/debug/tracing/set_graph_function
#### all functions enabled ####
#
```

If you want to trace only one function and all of its children, you just have to echo its name into `set_graph_function`:

```
echo __do_fault > set_graph_function
...
```

You can also expand several functions at once:

```
 echo sys_open > set_graph_function
 echo sys_close >> set_graph_function
...
```

Similarly, one can request an event "type" or class. Eg.

```
# echo -n "kmem" > /sys/kernel/debug/tracing/set_event
# cat /sys/kernel/debug/tracing/set_event
kmem:kmalloc
kmem:kmem_cache_alloc
kmem:kmalloc_node
kmem:kmem_cache_alloc_node
kmem:kfree
kmem:kmem_cache_free
kmem:mm_page_free_direct
kmem:mm_pagevec_free
kmem:mm_page_alloc
kmem:mm_page_alloc_zone_locked
```

```
kmem:mm_page_pcpu_drain
kmem:mm_page_alloc_extfrag
#
```

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

will get the full ftrace dump into the kernel ring buffer (dmesg).

...
Having Ftrace configured and enabling ftrace_dump_on_oops in the kernel boot parameters, or by echoing a "1" into */proc/sys/kernel/ftrace_dump_on_oops*, will enable Ftrace to dump to the console the entire trace buffer in ASCII format on oops or panic. Having the console output to a serial log makes debugging crashes much easier. You can now trace back the events that led up to the crash.
...

Trace a particular process: write PID into *set_ftrace_pid* .

<<
Caution: Make sure to reset the state of ftrace options/files/etc to their defaults once you're done (especially in a shell script), otherwise you may get unpleasant surprises the next time you try it out (without a reboot of course).
>>

Using trace_printk()

printk() is the king of all debuggers, but it has a problem. If you are debugging a high volume area such as the timer interrupt, the scheduler, or the network, printk() can lead to bogging down the system or can even create a live lock. It is also quite common to see a bug "disappear" when adding a few printk()s. This is due to the sheer overhead that printk() introduces (especially on a slow serial line – which is typically the case in embedded development environments).

Ftrace introduces a new form of printk() called trace_printk(). It can be used just like printk(), and can also be used in any context (interrupt code, NMI code, and scheduler code). What is nice about trace_printk() is that it does not output to the console. Instead it writes to the Ftrace ring buffer and can be read via the trace file.

Writing into the ring buffer with trace_printk() only takes around a tenth of a microsecond or so. But using printk(), especially when writing to the serial console, may take several milliseconds per write. The performance advantage of trace_printk() lets you record the most sensitive areas of the kernel with very little impact.

For example you can add something like this to the kernel or module:

```
    trace_printk("read foo %d out of bar %p\n", bar->foo, bar);
```

Then by looking at the trace file, you can see your output.

```
    [tracing]# cat trace
    # tracer: nop
    #
    #           TASK-PID    CPU#    TIMESTAMP  FUNCTION
    #              | |       |          |         |
               <...>-10690 [003] 17279.332920: : read foo 10 out of bar
ffff880013a5bef8
```

The above example was done by adding a module that actually had a `foo` and `bar` construct.

--snip--

To dump the ftrace buffer (to the kernel log), use:

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

and look it up via dmesg.

TIP: write a string to ***<tracefs>/trace_marker*** *to achieve the same on the command-line or within a script.*

The trace-cmd utility

trace-cmd is a very versatile front-end to the ftrace infrastructure.

Eg.
Use trace-cmd to trace all the 'sched'-related calls of the particular application 'myapp':

```
# trace-cmd record -e sched -F myapp
[…]
#
# trace-cmd report
[…]
#
# trace-cmd
```

```
trace-cmd version 1.0.3

usage:
  trace-cmd [COMMAND] ...

  commands:
     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
     restart - restart the kernel trace data recording
     show - show the contents of the kernel tracing buffer
     reset - disable all kernel tracing and clear the trace buffers
     clear - clear the trace buffers
     report - read out the trace stored in a trace.dat file
     stream - Start tracing and read the output directly
     profile - Start profiling and read the output directly
     hist - show a histogram of the trace.dat information
     stat - show the status of the running tracing (ftrace) system
     split - parse a trace.dat file into smaller file(s)
     options - list the plugin options available for trace-cmd report
     listen - listen on a network socket for trace clients
     list - list the available events, plugins or options
     restore - restore a crashed record
     snapshot - take snapshot of running trace
     stack - output, enable or disable kernel stack tracing
     check-events - parse trace event formats

#
```

USEFUL!

So, to see just the “event label” and not each and every function associated with each event, in abbreviated format:

```
$ sudo trace-cmd list |awk -F':' 'NF==2 {print $1}'|sort -t':' -k1 |uniq|tr '\n'
' '
alarmtimer amd_cpu avc block bpf_test_run bpf_trace bridge cgroup clk compaction
cpuhp cros_ec dev devfreq devlink dma_fence drm error_report events event
systems exceptions ext4 fib fib6 filelock filemap fs_dax gpio huge_memory hwmon
hyperv i2c initcall intel_iommu interconnect iocost iomap iommu io_uring irq
irq_matrix irq_vectors jbd2 kmem libata mce mdio migrate mmap mmap_lock mmc
module mptcp msr napi neigh net netlink nmi oom options page_isolation pagemap
page_pool percpu power printk pwm qdisc ras raw_syscalls rcu regmap regulator
resctrl rpm rseq rtc sched scsi signal skb smbus sock spi swiotlb sync_trace
syscalls task tcp thermal thermal_power_allocator timer tlb tracers udp vmscan
vsyscall wbt workqueue writeback x86_fpu xdp xen xhci-hcd $
```

Separate multiple event types with multiple `-e <event-type>` options.

Eg.
`trace-cmd record -e net -e sock -e syscalls [-F ...]`

```
# trace-cmd list
events:                                  << specify to trace-cmd record
                                             with the -e switch >>

kvmmmu:kvm_mmu_pagetable_walk
kvmmmu:kvm_mmu_paging_element
kvmmmu:kvm_mmu_set_accessed_bit
kvmmmu:kvm_mmu_set_dirty_bit
kvmmmu:kvm_mmu_walker_error
kvmmmu:kvm_mmu_get_page
...
kvm:kvm_async_pf_doublefault
kvm:kvm_async_pf_not_present
kvm:kvm_async_pf_ready
kvm:kvm_async_pf_completed
skb:kfree_skb
skb:consume_skb
skb:skb_copy_datagram_iovec
net:net_dev_xmit
net:net_dev_queue
net:netif_receive_skb
net:netif_rx
napi:napi_poll
syscalls:sys_enter_socket
syscalls:sys_exit_socket
syscalls:sys_enter_socketpair
…
syscalls:sys_exit_recvmmsg
syscalls:sys_enter_socketcall
syscalls:sys_exit_socketcall
scsi:scsi_dispatch_cmd_start
scsi:scsi_dispatch_cmd_error
scsi:scsi_dispatch_cmd_done
scsi:scsi_dispatch_cmd_timeout
scsi:scsi_eh_wakeup
regulator:regulator_enable
regulator:regulator_enable_delay
…
bkl:lock_kernel
bkl:unlock_kernel
block:block_rq_abort
block:block_rq_requeue
…
block:block_bio_remap
block:block_rq_remap
```

```
syscalls:sys_enter_add_key
syscalls:sys_exit_add_key
syscalls:sys_enter_request_key
…
jbd2:jbd2_checkpoint
jbd2:jbd2_start_commit
…
ext4:ext4_free_inode
ext4:ext4_request_inode
…
writeback:writeback_nothread
writeback:writeback_queue
…
vfs:dirty_inode
syscalls:sys_enter_getcwd
syscalls:sys_exit_getcwd
…
fs:do_sys_open
fs:open_exec
syscalls:sys_enter_truncate
syscalls:sys_exit_truncate
…
compaction:mm_compaction_isolate_migratepages
compaction:mm_compaction_isolate_freepages
compaction:mm_compaction_migratepages
syscalls:sys_enter_mbind
syscalls:sys_exit_mbind
…
kmem:kmalloc
kmem:kmem_cache_alloc
kmem:kmalloc_node
kmem:kmem_cache_alloc_node
kmem:kfree
kmem:kmem_cache_free
kmem:mm_page_free_direct
kmem:mm_pagevec_free
…
vmscan:mm_vmscan_kswapd_sleep
vmscan:mm_vmscan_kswapd_wake
vmscan:mm_vmscan_wakeup_kswapd
…
power:cpu_idle
power:cpu_frequency
power:machine_suspend
…
module:module_load
module:module_free
module:module_get
module:module_put
module:module_request
syscalls:sys_enter_delete_module
```

```
syscalls:sys_exit_delete_module
syscalls:sys_enter_init_module
syscalls:sys_exit_init_module
…
workqueue:workqueue_queue_work
workqueue:workqueue_activate_work
workqueue:workqueue_execute_start
workqueue:workqueue_execute_end
syscalls:sys_enter_setpriority
syscalls:sys_exit_setpriority
…
syscalls:sys_exit_getcpu
signal:signal_generate
signal:signal_deliver
signal:signal_overflow_fail
signal:signal_lose_info
syscalls:sys_enter_restart_syscall
syscalls:sys_exit_restart_syscall
…
timer:timer_init
timer:timer_start
timer:timer_expire_entry
…
plugins:                                    << specify with the '-p' switch >>
blk function_graph mmiotrace wakeup_rt wakeup function sched_switch nop

options:
print-parent
nosym-offset
nosym-addr
…
funcgraph-cpu
funcgraph-overhead
nofuncgraph-proc
funcgraph-duration
nofuncgraph-abstime
funcgraph-irqs
#
```

(Repeated from above): To see just the "event label" and not each and every function associated with each event, in abbreviated format:

```
$ sudo trace-cmd list |awk -F':' 'NF==2 {print $1}'|sort -t':' -k1 |uniq|
tr '\n' ' '
alarmtimer asoc avc block bpf_test_run bpf_trace bridge cfg80211 cgroup
clk compaction cpuhp cros_ec devfreq devlink dma_fence drm error_report
events exceptions ext4 fib fib6 filelock filemap fs_dax gpio gvt hda
hda_controller hda_intel huge_memory hwmon hyperv i2c i915 initcall
intel_iommu intel_ish interconnect iocost iomap iommu io_uring irq
irq_matrix irq_vectors iwlwifi iwlwifi_data iwlwifi_io iwlwifi_msg
```

```
iwlwifi_ucode jbd2 kmem kvm kvmmmu libata mac80211 mac80211_msg mce mdio
mei migrate mmap mmap_lock mmc module mptcp msr napi neigh net netlink
nmi nvme oom options page_isolation pagemap page_pool percpu power printk
pwm qdisc random ras raw_syscalls rcu regmap regulator resctrl rpm rseq
rtc sched scsi signal skb smbus sock spi swiotlb sync_trace syscalls task
tcp thermal thermal_power_allocator timer tlb tracers ucsi udp v4l2 vb2
vmscan vsyscall wbt workqueue writeback x86_fpu xdp xen xhci-hcd trccmd $
```

Separate multiple event types with multiple `-e <event-type>` options. Eg.

```
trace-cmd record -e net -e sock -e syscalls
```

Tips

Ftrace all kernel APIs concerned with kernel memory (kmem) and networking (net) for the ping command:

```
sudo trace-cmd record -e kmem -e net -F ping -c1 google.com
sudo trace-cmd report
```

trace-view: nicer GUI interrface for viewing and filtering the ASCII text report

Rudimentary GUI with trace-graph.

trace-cmd report – a few of the options:

-e
This outputs the endianess of the file. trace-cmd report is smart enough to be able to read big endian files on little endian machines, and vise versa.

-f
This outputs the list of functions that have been recorded in the file.

-P
This outputs the list of "trace_printk()" data. The raw trace data points to static pointers in the
kernel. This must be stored in the trace.dat file.

-I
Do not print events where the HARDIRQ latency flag is set. This will filter out most events that are from interrupt context. Note, it may not filter out function traced functions that are in interrupt context but were called before the kernel "in interrupt" flag was set.

-S

Do not print events where the SOFTIRQ latency flag is set. This will filter out most events that are from soft interrupt context.

--profile

With the --profile option, "trace-cmd report" will process all the events first, and then output a format showing where tasks have spent their time in the kernel, as well as where they are blocked the most, and where wake up latencies are.

See trace-cmd-profile(1) for more details and examples.

-l

This adds a "latency output" format. Information about interrupts being disabled, soft irq being disabled, the "need_resched" flag being set, preempt count, and big kernel lock are all being recorded with every event. But the default display does not show this information. This option will set display this information with 6 characters. When one of the fields is zero or N/A a '.\' is shown.

```
  <idle>-0       0d.h1. 106467.859747: function:             ktime_get <--
tick_check_idle
```

The `0d.h1.` denotes this information. The first character is never a '.' and represents what CPU the trace was recorded on (CPU 0). The 'd' denotes that interrupts were disabled. The 'h' means that this was called inside an interrupt handler. The '1' is the preemption disabled (preempt_count) was set to one. The two '.'s are "need_resched" flag and kernel lock counter. If the "need_resched" flag is set, then that character would be a 'N'.

Examples

...

To see only the kmalloc calls that were greater than 1000 bytes:

```
# trace-cmd report -F 'kmalloc: bytes_req > 1000'
```

…

Eg.

On a Raspberry Pi 3 Model B+:

Steps:

Get trace-cmd:

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git
```

Build & install:

```
make && sudo make install
```

Use it: wanted to capture the work of *platform_driver_register()* within the insmod:

First, lets make it easier by *ourselves* emitting a message into the ftrace log buffer (via the *trace_printk()*), around the area of interest; this way, we can just search the (typically huge) report for this string!

```
[...]
      trace_printk("@@@@@ %s: MARKER 1: platform_driver_register()
begin\n", DRVNAME);
      ret_val = platform_driver_register(&my_platform_driver);
      trace_printk("@@@@@ %s: MARKER 2: platform_driver_register() done\
n", DRVNAME);
```

```
trace-cmd record -p function_graph -F taskset 01 insmod
./dtdemo_platdrv.ko
trace-cmd report -I -S -l > ftrc_rep.txt
```

Notice how we use taskset(1) to ensure that the process we're interested in ftrace-ing (insmod) runs on exactly one CPU.
Voila! Done.

Tip 2

To convert raw ftrace (manual invocation) trace data to a *trace.dat* file, usually so that we can interpret it graphically with KernelShark, do:

```
# trace-cmd extract -o 2.dat
```

This will read from the current “trace” buffer and write to the file 2.dat .

```
# kernelshark 2.dat &
```

(KernelShark appears to be a neat clubbing of the trace-graph and trace-view GUIs).

Tip 3

To visualize the trace data as a histogram:

```
# trace-cmd hist
version = 6                        << This routine ran 10,500 times >>
  %50.00  (25172) qemu-system-x86                    user_stack #10500
         |
         --- *user_stack*
          kvm_pvclock_update
          kvm_arch_vcpu_ioctl_run
          kvm_vcpu_ioctl
          do_vfs_ioctl
          SyS_ioctl
          tracesys_phase2
  %11.85  (25172) qemu-system-x86         kvm_mmu_paging_element #2488
         |
         --- *kvm_mmu_paging_element*
            |
            |--%100.00-- paging64_gva_to_gpa  # 2488
                         |
                         |--%51.13-- emulator_read_write_onepage  # 1272
                         |          emulator_read_write
                         |            |
                         |            |--%90.25--
emulator_write_emulated  # 1148
                         |            |          segmented_write
                         |            |          writeback
                         |            |          x86_emulate_insn
                         |            |          x86_emulate_instruction
--snip--
```

Using trace-cmd – a decent resource:
http://lwn.net/Articles/383334/

Q. What if I don't have a serial console or JTAG?

A. If you're doing early kernel debug, you're out of luck.
If not, and your board supports networking/ethernet, check out the useful Netconsole.
Also, see the section on “Debugging Early Boot Issues” !

Do check out my *trccmd* front-end to the trace-cmd front-end :-)

https://github.com/kaiwan/trccmd

Example: trace a single ping:
./trccmd -F 'ping -c1 yahoo.com' -e 'net sock skb tcp udp'

Practicalities – 1

If you do the above, i.e.,
`./trccmd -F 'ping -c1 yahoo.com' -e 'net sock skb tcp udp'`

the output is great but over 70,000 lines!

My LKD book's trccmd_1ping.sh script allows you to pass a parameter:

```
$ ./trccmd_1ping.sh
./trccmd_1ping.sh: needs root.
Usage: ./trccmd_1ping.sh option
option = -f : show in function_graph format
option = -p : show the parameters
```

Interestingly, with the -f option, you get wonderfully indented function-graph style output but at a cost – it's very voluminous (70k – 80k lines).
With the -p option, you get *all the parameters* to each function (wow!) AND the output is now merely around 80 – 100 lines!

A sample run:

```
$ sudo ./trccmd_1ping.sh -p
trace-cmd record -e net -e sock -e skb -e tcp -e udp -F ping -c1 packtpub.com
PING packtpub.com (104.22.1.175) 56(84) bytes of data.
64 bytes from 104.22.1.175 (104.22.1.175): icmp_seq=1 ttl=63 time=22.9 ms

--- packtpub.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.893/22.893/22.893/0.000 ms
CPU0 data recorded at offset=0x6f2000
    4096 bytes in size
CPU1 data recorded at offset=0x6f3000
    0 bytes in size
CPU2 data recorded at offset=0x6f3000
    4096 bytes in size
...
$ cat ping_trccmd.txt
CPU 1 is empty
CPU 3 is empty
CPU 4 is empty
CPU 5 is empty
cpus=6
    ping-19070   2.... 17979.705983: kfree_skb:
skbaddr=0xffff90ab89200400 protocol=0 location=0xffffffff9c83af78 reason:
NOT_SPECIFIED
    ping-19070   2.... 17979.705992: kfree_skb:
skbaddr=0xffff90ab89200400 protocol=0 location=0xffffffff9c83af78 reason:
NOT_SPECIFIED
    ping-19070   2.... 17979.706149: net_dev_queue:        dev=lo
skbaddr=0xffff90ab89200400x len=83
    ping-19070   2.... 17979.706150: net_dev_start_xmit:   dev=lo
queue_mapping=0 skbaddr=0xffff90ab89200400 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 len=83 data_len=0 network_offset=14
transport_offset_valid=1 transport_offset=34 tx_flags=0 gso_size=0 gso_segs=0
gso_type=0
    ping-19070   2.... 17979.706151: netif_rx_entry:       dev=lo napi_id=0x3
queue_mapping=0 skbaddr=0xffff90ab89200400 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 hash=0xf3b04bf4 l4_hash=1 len=69
data_len=0 truesize=768 mac_header_valid=1 mac_header=-14 nr_frags=0 gso_size=0
gso_type=0
    ping-19070   2.... 17979.706152: netif_rx:             dev=lo
skbaddr=0xffff90ab89200400x len=69
```

```
    ping-19070   2.... 17979.706153: netif_rx_exit:        ret=0
    ping-19070   2.... 17979.706153: net_dev_xmit:         dev=lo
skbaddr=0xffff90ab89200400 len=83 rc=0
    ping-19070   2..s1 17979.706154: netif_receive_skb:    dev=lo
skbaddr=0xffff90ab89200400x len=69
    ping-19070   2.... 17979.706292: net_dev_queue:        dev=lo
skbaddr=0xffff90ab89200900x len=83
    ping-19070   2.... 17979.706292: net_dev_start_xmit:   dev=lo
queue_mapping=0 skbaddr=0xffff90ab89200900 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 len=83 data_len=0 network_offset=14
transport_offset_valid=1 transport_offset=34 tx_flags=0 gso_size=0 gso_segs=0
gso_type=0
    ping-19070   2.... 17979.706293: netif_rx_entry:       dev=lo napi_id=0x3
queue_mapping=0 skbaddr=0xffff90ab89200900 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 hash=0xf3b04bf4 l4_hash=1 len=69
data_len=0 truesize=768 mac_header_valid=1 mac_header=-14 nr_frags=0 gso_size=0
gso_type=0
    ping-19070   2.... 17979.706293: netif_rx:             dev=lo
skbaddr=0xffff90ab89200900x len=69
    ping-19070   2.... 17979.706293: netif_rx_exit:        ret=0
    ping-19070   2.... 17979.706293: net_dev_xmit:         dev=lo
skbaddr=0xffff90ab89200900 len=83 rc=0
    ping-19070   2..s1 17979.706293: netif_receive_skb:    dev=lo
skbaddr=0xffff90ab89200900x len=69
    ping-19070   2.... 17979.761847: consume_skb:
skbaddr=0xffff90ab89200700
    ping-19070   0.... 17979.800782: consume_skb:
skbaddr=0xffff90ab89200500
    ping-19070   0.... 17979.800858: skb_copy_datagram_iovec:
skbaddr=0xffff90ab902f0000 len=164
    ping-19070   0.... 17979.800858: consume_skb:
skbaddr=0xffff90ab902f0000
    ping-19070   0.... 17979.800865: skb_copy_datagram_iovec:
skbaddr=0xffff90ab902f0000 len=144
    ping-19070   0.... 17979.800866: consume_skb:
skbaddr=0xffff90ab902f0000
    ping-19070   0.... 17979.800867: consume_skb:
skbaddr=0xffff90ab902f0500
    ping-19070   0.... 17979.800868: skb_copy_datagram_iovec:
skbaddr=0xffff90ab902f0000 len=20
    ping-19070   0.... 17979.800868: consume_skb:
skbaddr=0xffff90ab902f0000
    ping-19070   0.... 17979.800946: net_dev_queue:        dev=enp0s3
skbaddr=0xffff90ab902f0000x len=98
    ping-19070   0.... 17979.800947: net_dev_start_xmit:   dev=enp0s3
queue_mapping=0 skbaddr=0xffff90ab902f0000 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=0 len=98 data_len=0 network_offset=14
transport_offset_valid=1 transport_offset=34 tx_flags=0 gso_size=0 gso_segs=0
gso_type=0
    ping-19070   0.... 17979.800960: net_dev_xmit:         dev=enp0s3
skbaddr=0xffff90ab902f0000 len=98 rc=0
    ping-19070   0.... 17979.823928: skb_copy_datagram_iovec:
skbaddr=0xffff90ab888aab00 len=64
    ping-19070   0.... 17979.823930: consume_skb:
skbaddr=0xffff90ab888aab00
    ping-19070   0.... 17979.824005: net_dev_queue:        dev=lo
skbaddr=0xffff90ab902f0a00x len=96
```

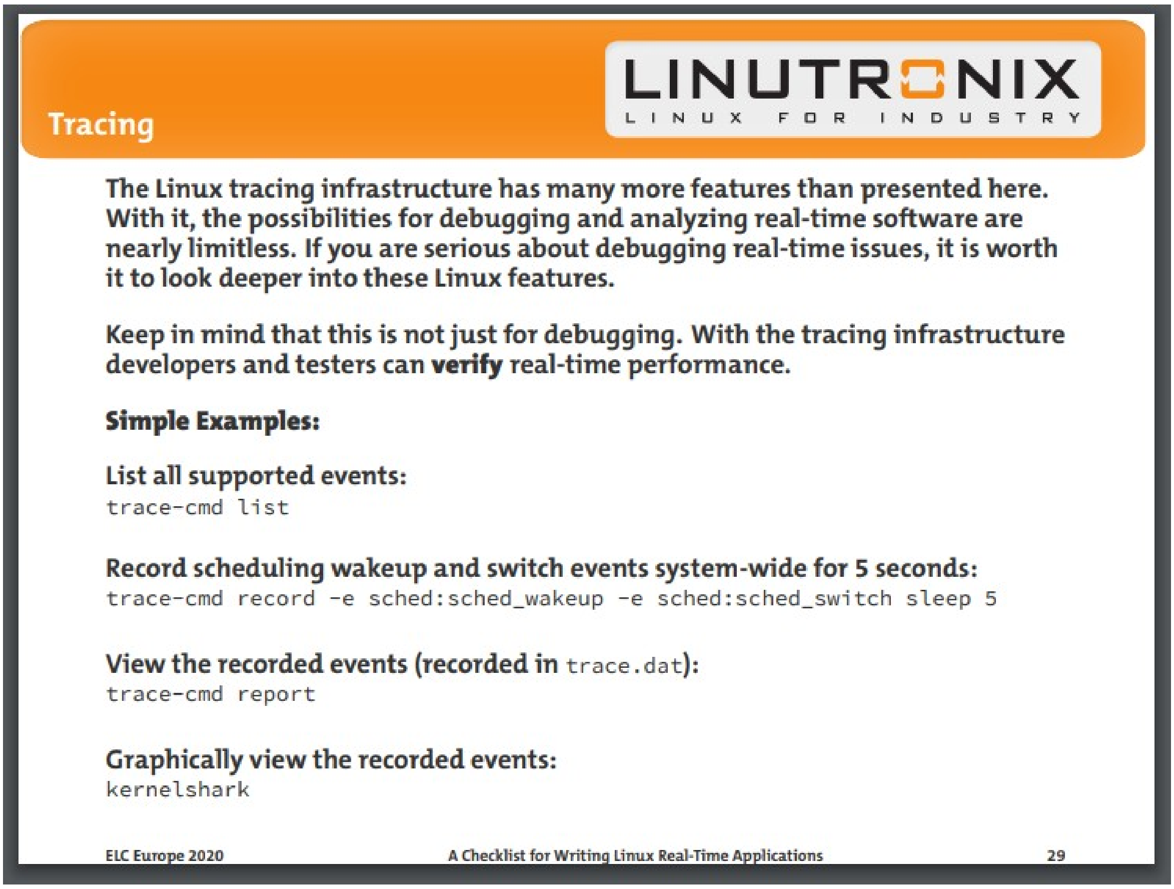
```
    ping-19070   0.... 17979.824006: net_dev_start_xmit:   dev=lo
queue_mapping=0 skbaddr=0xffff90ab902f0a00 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 len=96 data_len=0 network_offset=14
transport_offset_valid=1 transport_offset=34 tx_flags=0 gso_size=0 gso_segs=0
gso_type=0
    ping-19070   0.... 17979.824007: netif_rx_entry:       dev=lo napi_id=0x1
queue_mapping=0 skbaddr=0xffff90ab902f0a00 vlan_tagged=0 vlan_proto=0x0000
vlan_tci=0x0000 protocol=0x0800 ip_summed=3 hash=0x9e122a79 l4_hash=1 len=82
data_len=0 truesize=768 mac_header_valid=1 mac_header=-14 nr_frags=0 gso_size=0
gso_type=0
    ping-19070   0.... 17979.824008: netif_rx:             dev=lo
skbaddr=0xffff90ab902f0a00x len=82
    ping-19070   0.... 17979.824008: netif_rx_exit:        ret=0
    ping-19070   0.... 17979.824008: net_dev_xmit:         dev=lo
skbaddr=0xffff90ab902f0a00 len=96 rc=0
    ping-19070   0..s1 17979.824009: netif_receive_skb:    dev=lo
skbaddr=0xffff90ab902f0a00x len=82
    ping-19070   0.... 17979.899048: consume_skb:
skbaddr=0xffff90ab89200600
tracecmd $
```

<<

Practicalities – 3

Using tracing

Src: *A Checklist for Writing Linux Real-Time Applications; ELC, Oct 2020*



Tracing

LINUTRONIX
LINUX FOR INDUSTRY

The Linux tracing infrastructure has many more features than presented here. With it, the possibilities for debugging and analyzing real-time software are nearly limitless. If you are serious about debugging real-time issues, it is worth it to look deeper into these Linux features.

Keep in mind that this is not just for debugging. With the tracing infrastructure developers and testers can **verify** real-time performance.

Simple Examples:

List all supported events:

```
trace-cmd list
```

Record scheduling wakeup and switch events system-wide for 5 seconds:

```
trace-cmd record -e sched:sched_wakeup -e sched:sched_switch sleep 5
```

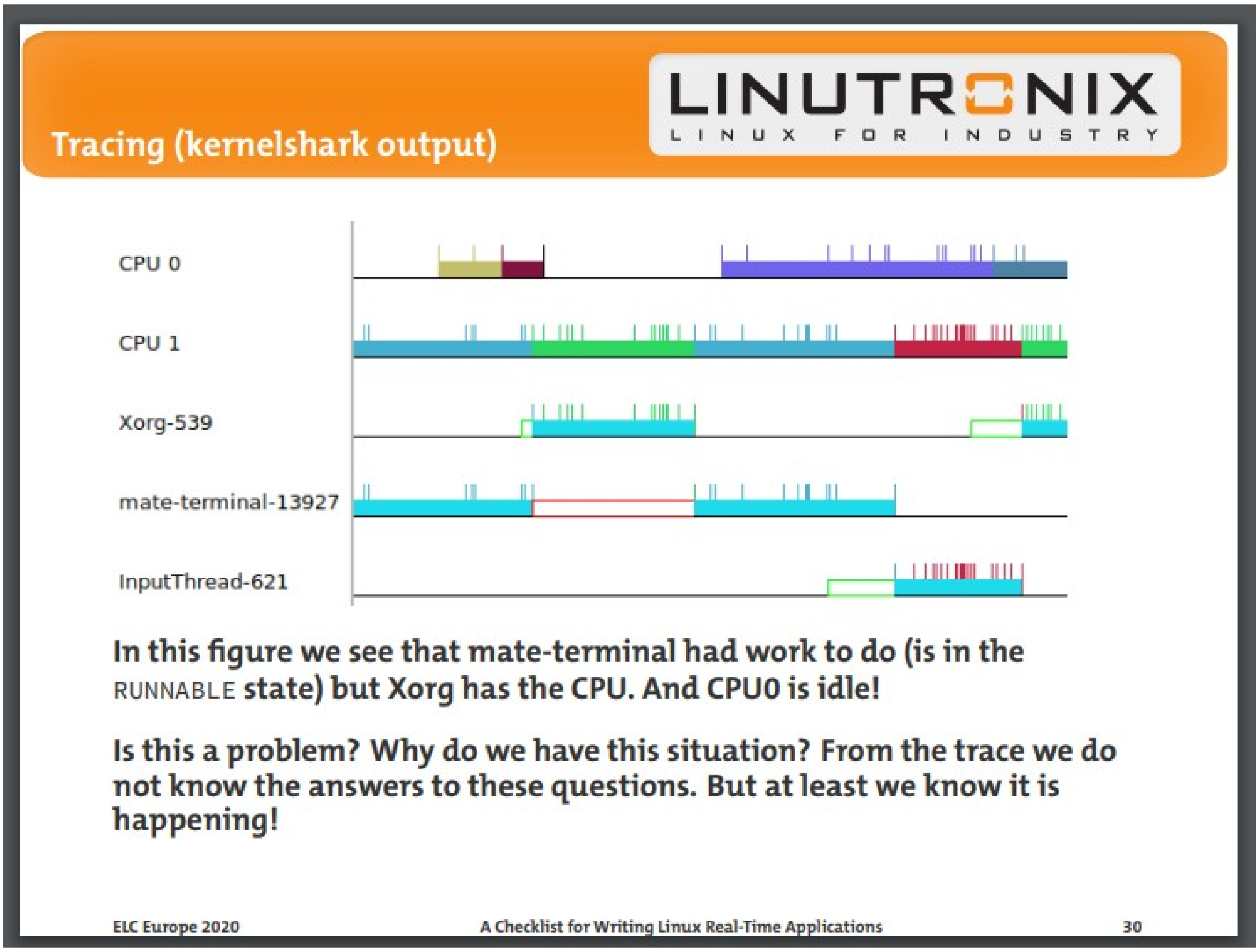
View the recorded events (recorded in `trace.dat`):

```
trace-cmd report
```

Graphically view the recorded events:

```
kernelshark
```

ELC Europe 2020 A Checklist for Writing Linux Real-Time Applications 29



Tracing (kernelshark output)

In this figure we see that mate-terminal had work to do (is in the RUNNABLE state) but Xorg has the CPU. And CPU0 is idle!

Is this a problem? Why do we have this situation? From the trace we do not know the answers to these questions. But at least we know it is happening!

ELC Europe 2020 A Checklist for Writing Linux Real-Time Applications 30

Practicalities - 3

Once (on a project, ARM64), I used my front-end *trccmd* to get a detailed kernel-level trace to see deep into an OLED display's (Python-based) write… the embedded system would invariably reset when attempting this large record/report session… I noticed the kernel log (`journalctl --since="5 min ago"`) showing messages like this:

```
systemd[1]: Stopped Serial Getty on ttyS0.
```

So, I temporarily disabled this service:

```
sudo systemctl stop serial-getty@ttyS0.service
```

and then ran *trccmd* again. This time it worked! I got the very detailed trace (1.7GB for the /tmp/trc.txt (!) and 722 MB for the binary trccmd_trc.dat data file).

Had me realize I should use more filtering rather than capture all events (the default).. So, something like

```
trccmd -F ‘</path/to/app>’ -e ‘block dev ext4 filelock filemap
fscache gpio hwmon i2c iomap io_uring ipi irq mmc module net power
random raw_syscalls regmap rtc sched signal smbus sock spi task
tcp thermal timer udp v4l2 workqueue writeback’
```

helps… With the event filters, the /tmp/trc.txt (raw function-graph style trace) became (just) 77 MB and the binary trccmd_trc.dat went down to 36 MB!

Further, analysing the binary trace file with the KernelShark GUI helps!

(Note though, that even more ‘visual’ visualizations :-) are available via the *FlameGraph* scripts!
See my wrapper to generate FlameGraph / FlameChart here:
https://github.com/kaiwan/L5_debug_trg/tree/master/flamegraph).
