QUICK START GUIDE FOR LINUX FTRACE

GOPISE YUAN

AUG/2024

EXTERNAL USE

SECURE CONNECTIONS
FOR A SMARTER WORLD

Why this document:

- Ftrace is powerful tracing utility embedded in Linux kernel.
- It provides a very good method for kernel developer to get insights of the kernel behavior.

- Official document for ftrace is long and somehow complex. This document provide a simpler and faster way to quickly hands-on.

EXTERNAL USE

Ftrace:

- Ftrace is a framework designed to track the internal behavior of the Linux kernel. Two main features:
  - Event monitoring: monitor many kernel events. Such as scheduling, interrupt, timer, etc.
  - Tracing: It provides functions through different tracers: function, irqsoff, wakeup, … Different tracers designed for different purposes.
    - We will only cover function related tracers in this document.

- Scope/Limitation: Only function/tracepoint can be traced. Macro/inline will not be traced.

Preparation

Preparation:

- Function implemented mainly based compile time hooking. This means, some kernel configure needs to be turned ON and recompiling is required.

```
CONFIG_FTRACE=y
CONFIG_DYNAMIC_FTRACE=y
CONFIG_FUNCTION_TRACER=y
CONFIG_FUNCTION_GRAPH_TRACER=y
CONFIG_FUNCTION_PROFILER=y
CONFIG_FTRACE_SYSCALLS=y
CONFIG_TRACER_SNAPSHOT=y
CONFIG_BLK_DEV_IO_TRACE=y
CONFIG_IRQSOFF_TRACER=y
```

- Userspace interface:
  - “tracefs” will be automatically mounted after boot with FTRACE enabled:

```
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
```

  - If not automatically mounted, manual mount:

```
mount -t tracefs nodev /sys/kernel/tracing
```

Common and output interface

- **“tracing_on”:**
  - Main ON/OFF switch for ftrace. It will be default to “1” (ON) after boot. Note: it only determine if writing to trace buffer is enabled. Trace overhead will still be there even disabled.
- **“trace” / “trace_pipe”:**
  - Output of the trace buffer. Reading this file is just reading the trace buffer. Difference:
  - trace: reading this file will block the writing to trace buffer. Content in buffer will not be affected by reading (not consumed).
  - trace_pipe: reading this file will not block writing. Buffer will be “consumed” (read one line, remove one line) by reading.
- **“snapshot”:**
  - Similar to trace, but only taking a static snapshot of the trace buffer.
  - Needs manual trigger the snapshot:
    - “echo 1 > snapshot”: take a snapshot, content can be read after this “cat snapshot”.
    - “0” to clear and free snapshot buffer. “2” (or else) to clear snapshot buffer only.
- **“trace_options”, “options”, “tracing_max_latency”, “buffer_size_kb”, …**
  - Many configuration options for ftrace. Refer to official document for details.

Event

Event:

- Implemented base tracepoints, which is actually customized function hook, in some kernel modules.
- Two different ways to enable specific events monitoring:
  - Using “set_event” node.
  - Using “events” folder.

- Note:
  - Event is isolated from tracer. There’s no need for a tracer for event to work.

EXTERNAL USE

Event: method #1, “set_event”

- **“available_events”:**
  - File node contains all events currently supported. Format:

    <sub-system>:<event_name>
- To enable monitoring an event:
  - Simply echo the event name into “set_event” node:

```
echo irq:irq_handler_entry > set_event
```

- Note:
  - Specific event or all events in a sub-system can be enabled. To enable all event in a sub-system, just simply remove the event name after “:”. E.g.

```
root@imx93evk:/sys/kernel/tracing# echo irq: > set_event
root@imx93evk:/sys/kernel/tracing# cat trace
# tracer: nop
#
# entries-in-buffer/entries-written: 13808/13808   #P:2
#
#                                _-----=> irqs-off/BH-disabled
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| / _-=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
              sh-484     [000] d.h..  1779.789214: irq_handler_entry: irq=17 name=fsl-
lpuart
              sh-484     [000] d.h..  1779.789218: irq_handler_exit: irq=17 ret=handled
          <idle>-0       [001] dnh1.  1779.789926: irq_handler_entry: irq=1 name=IPI
          <idle>-0       [001] dNh1.  1779.789929: irq_handler_exit: irq=1 ret=handled
[…]
```

Event: method #2

- **“events”:**
  - A folder under ftrace root contains a large directory tree. This tree is a structural reflection for “available_events”. The 1st level is the subsystem and 2nd (and 3rd) level is normally the event.
- Some nodes under these folders:
  - **“enable”:** ON/OFF switch for specific event or all events in a sub-system.
  - **“format”:** only in event folder. Read only. Contains the format information for event. Note: different event normally have different output format.
  - **“filter”:** filter the event message. More details later. Besides this filter, we can also filter event for specific PID through “set_event_pid” node.
  - **“trigger”:** can do some action when the event happens. This is the same as tracer function. More details later. A simple example:

```
# turn off trace when schedule switch happens:
echo traceoff > events/sched/sched_switch/trigger
```


EXTERNAL USE

Event: method #2, filter

- Expression based filter supported. Format:

  <field> <relational-operator> <value>

  - **<field>:** variable name in “format” file. Different event has different variable. E.g. “irq” & “name” in previous “irq_handler_*” event.
  - **<relational-operator>:** Relational operator on number & string supported: “==, !=, <, <=, >, >=, &” or “~” (string only).
  - **<value>:** value of the variable to compare.
- Multiple expression can be OR/AND to form a complex condition.

- Example: enable interrupt handler event for IRQ between 12 & 120 and belongs to UART or MMC:

```
echo "(irq > 12 && irq < 120) && (name ~ '*uart' || name ~ 'mmc*')" > events/irq/filter
```

Event: method #2, filter Cont'd

- Special notes for filter:
  - Events are arranged in leveled structure. In this case, a filter set in higher level will only applicable for all sub-level events that contains corresponding variable. Other events in same level will be kept intact.
  - Due to the same reason, the filter set in higher level can only be visible in lower "applicable" event level.

```
root@imx93evk:/sys/kernel/tracing/events/irq# echo "irq > 12 && irq < 20" > filter
root@imx93evk:/sys/kernel/tracing/events/irq# cat filter
### global filter ###
# Use this to set filters for multiple events.
# Only events with the given fields will be affected.
# If no events are modified, an error message will be displayed here
root@imx93evk:/sys/kernel/tracing/events/irq# cat irq_handler_entry/filter
irq > 12 && irq < 20
```

  - A checking will be done when setting the filter. If checking passed, the filter will appear in "filter" node. If error found, an error message will be put into "filter" as a prompt:

```
root@imx93evk:/sys/kernel/tracing/events/irq# echo "(irq > 12) && (irq < 120" > filter
root@imx93evk:/sys/kernel/tracing/events/irq# cat irq_handler_entry/filter
(irq > 12) && (irq < 120
^
parse_error: Couldn't find or set field in one of a subsystem's events
```

Trace

Trace:

- Trace is driven by tracer. Tracer are configured through kernel configure options.
- A hook will be added to every valid kernel function in compiling time. The tracers implemented different functions behind the hook.
- “Dynamic ftrace”: no tracer will be enabled in runtime by default. This allows very low overhead on performance after ftrace is enabled in compile time but kept OFF in runtime.

- **“available_tracers”:**

  File contains all available/configured tracers in current system. Some common used ones:

  - **hwlat**: Hardware Latency, detect hardware latency.
  - **blk**: Used by blktrace application to trace block device/IO activity.
  - **wakeup*/preempt*/irqoff**: Tracers used for tracing scheduling, preemption, irq ON/OFF, etc.
  - **function/function_graph**: function tracers. The major difference is, “function” will only trace function entry, while “function_graph” will also trace exit. This enables “function_graph” to draw a full calling graph for target functions.

```
root@imx93evk:/sys/kernel/tracing# cat available_tracers
hwlat blk function_graph wakeup_dl wakeup_rt wakeup preemptirqsoff preemptoff irqsoff function nop
```

Load tracer

- Default tracer will be “nop”, which means “no tracer”, after boot.
- To load a tracer, just simply echo the name of the tracer to “current_tracer” file.
- As previously explained, the “tracing_on” will be default to “1”. So, trace will start to output after loading the tracer.
- Note:
  - Switching tracer requires locking of the trace buffer. If an error says “*** busy”, that means someone is currently reading trace (such as “cat trace_pipe”).

```
root@imx93evk:/sys/kernel/tracing# echo function > current_tracer
sh: echo: write error: Device or resource busy
root@imx93evk:/sys/kernel/tracing# fuser trace_pipe
/sys/kernel/tracing/trace_pipe:   510
root@imx93evk:/sys/kernel/tracing# kill 510
root@imx93evk:/sys/kernel/tracing# echo function > current_tracer
root@imx93evk:/sys/kernel/tracing# cat current_tracer
function
```

function & function_graph tracer:

- These two tracers are similar and use a common set of configuration interfaces/nodes. Commonly used interfaces for both tracers:
  - set_ftrace_pid, available_filter_functions, set_ftrace_filter / set_ftrace_notrace.

  Yes! They are all for filters.

function & function_graph tracer: Filter

- All functions in kernel will be traced after either one of these tracer is enabled. We normally needs to filter the trace to focus on what we need.
- Two types of filtering: “by PID” & “by name”. These two filter can work in a combination.
- **“available_filter_functions”:**
  - Contains a long list for all traceable functions in current system. Mostly a function symbol list.

- PID filter:
  - “set_ftrace_pid”: only trace functions called from target PID.

function & function_graph tracer: Filter, by name

- “set_ftrace_filter” / “set_ftrace_notrace”:
  - Configure to only trace specific functions (filter) or not trace specific functions (notrace).
  - The default setting is tracing all functions:

```
root@imx93evk:/sys/kernel/tracing# cat set_ftrace_filter
#### all functions enabled ####
```

  - This is an example to trace USB hub related functions: →

```
root@imx93evk:/sys/kernel/tracing# echo "usb_hcd_*" > set_ftrace_filter
root@imx93evk:/sys/kernel/tracing# cat set_ftrace_filter
usb_hcd_start_port_resume
usb_hcd_check_unlink_urb
usb_hcd_unlink_urb_from_ep
[…]
root@imx93evk:/sys/kernel/tracing# echo function > current_tracer
root@imx93evk:/sys/kernel/tracing# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 177/1381873621   #P:2
#
#                                _-----=> irqs-off/BH-disabled
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| / _-=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
     kworker/0:2-54      [000] d.h2.  1286.466939: usb_hcd_irq <-host_irq
     kworker/0:2-54      [000] d.h3.  1286.466947: usb_hcd_resume_root_hub <-ehci_irq
     kworker/0:2-54      [000] d.h2.  1286.466951: usb_hcd_poll_rh_status <-ehci_irq
[…]
```

function & function_graph tracer: Filter, by name Cont'd

- Some advanced usage for name filter.
- **Module filter:**
  - Besides function name, module name also allowed. Format:

    [function]:mod:<module>
  - Example: trace all functions starting with "write" in "ext3" module.

```
echo 'write*:mod:ext3' > set_ftrace_filter
```

  - Note: At least in L6.6.23, the module must be built as ".ko" (can be found from "lsmod"). Built-in modules are not supported.
- **Trigger:**
  - Similar to trigger in event monitoring, an action can be triggered by target function call.
  - Commonly used actions: traceon/traceoff (turn trace ON/OFF), enable_event/disable_event (turn event ON/OFF), snapshot/dump/cpudump (take snapshot or dump). Format:

    <function>:<trigger>[:count]

    [count] is optional (count down counter). It specify how many times the action will be done if <function> been called.
  - Example: enable "gadget:usb_gadget_connect" event every time when "usb_hcd_resume_root_hub" been called.

```
echo "usb_hcd_resume_root_hub:enable_event:gadget:usb_gadget_connect" > set_ftrace_filter
```

  - *Note: at least on L6.6.23, the filter will be abnormal if trigger enabled. Trigger will work, but filter will be invalid.*

function_graph tracer:

- Several interfaces design for “function_graph” only.
- “set_graph_function” / “set_graph_notrace”:
  - Similar to “set_ftrace_filter”. Difference is, “set_ftrace_filter” filter through function names and output a flat function list, while this filter will output a nested call graph inside the target function.
- “max_graph_depth”:
  - Max level of depth to trace.
- Example: trace all functions call inside “usb_disable_endpoint”, max level 3 →

```
root@imx93evk:/sys/kernel/tracing# echo usb_disable_endpoint > set_graph_function
root@imx93evk:/sys/kernel/tracing# echo 3 > max_graph_depth
root@imx93evk:/sys/kernel/tracing# echo function_graph > current_tracer
root@imx93evk:/sys/kernel/tracing# cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)               |  usb_disable_endpoint() {
 0)               |    irq_enter_rcu() {
 0)   1.166 us    |      preempt_count_add();
 0)   0.834 us    |      irqtime_account_irq();
 0)   4.959 us    |    }
 […]
 0)               |    usb_hcd_flush_endpoint() {
 0)   2.583 us    |      _raw_spin_lock_irq();
 0)   1.708 us    |      _raw_spin_unlock_irq();
 0)   8.334 us    |    }
 […]
 0) # 5982.083 us |  }
```

- *Note: at least on L6.6.23, setting both “set_graph_function” and “set_ftrace_filter” may cause filter not working, even there’s overlap.*

Others

Usage and expression:

- Most of the configuration nodes in ftrace can accept the following usage:
  - “*” expression, e.g. : “func*”, “*name”, etc.
  - Space separated list: e.g. : “echo func1 func2 func3 > [node]”
  - “>>” to add more content to existing node.
  - “!” to revoke/remove an item from existing node.
  - “echo > [node]” to clear the content of a node. Note: Some nodes doesn’t support this, needs to use “!” to remove one by one.

Other feature:

- Instance:
  - Ftrace support multi-instance. Multiple traces can be done in parallel. The “instance” folder under the ftrace root is designed for this purpose.
  - To create a new instance, just create a new folder under this directory through “mkdir”. Many files/folders (almost the same as ftrace root folder) will be automatically created by ftrace. Usage is totally the same as single instance case, except the file nodes used are under this new folder.
  - To remove an instance, just “rmdir” the corresponding folder.

Known-how

Known pitfalls and issues:

- Newly added functions can’t be found from trace, even it’s listed in “available_filter_functions” and no filters are configured:
  - Ftrace can’t trace inline functions, which will be expanded by compiler. Under some specific compiling configurations, some short/small non-inline functions will still be expanded by the optimizer.
  - Add “**noinline**” key word to the definition of the function.
- Boot failure after enabling ftrace in kernel:
  - This might be caused by the kernel size. The kernel size will dramatically increase after enabling ftrace. In this case, the kernel loaded into RAM may be overlapped with following boot components (DTB, ramdisk, etc.).
  - Check some address settings in uboot (fdt_addr*, initrd_addr, scriptaddr).

References:

- https://www.kernel.org/doc/html/v5.0/trace/ftrace.html
- https://www.kernel.org/doc/html/v5.0/trace/events.html
- https://docs.kernel.org/trace/tracepoints.html

SECURE CONNECTIONS
FOR A SMARTER WORLD