LF Live MENTORSHIP SERIES

Extra Boot Configuration and Boot-time Tracing

Masami Hiramatsu,
Tech Lead, Linaro ltd.

LF live MENTORSHIP SERIES

Speaker

Masami Hiramatsu

- Work for Linaro
- Tech Lead for a Linaro member’s Landing Team
- Maintainer of Kprobes, bootconfig and related tracing features/tools

LF live MENTORSHIP SERIES

Notice

- This session is not explaining ftrace itself. If you are not familiar with ftrace, please check Steven's session.

- "Tracing with Ftrace: Critical Tooling for Linux Development" by Steven Rostedt
  - https://events.linuxfoundation.org/mentorship-session-tracefs-the-building-blocks-of-linux-kernel-tracing-by-ftrace/

Why Kernel Boot-time Tracing?

Debug and analyze boot time errors and performance issues

- Measure performance statistics of kernel boot
- Analyze driver initialization failure
- Debug boot up process
- Continuously tracing from boot time

etc.

What we can do?

LF live MENTORSHIP SERIES

Kernel Cmdline Options for Tracing

Ftrace options for kernel command line

- Setup options (trace_options=)
- Output to printk (tp_printk)
- Enable events (trace_events=)
- Enable tracers (ftrace=)
- Filtering (ftrace_filter=,ftrace_notrace=,ftrace_graph_filter=,ftrace_graph_notrace=)
- Add kprobe events (kprobe_events=)
- And other options (alloc_snapshot, traceoff_on_warning, ...)

Boot time **event** tracing

- Output events to console (`tp_printk`)
  - Write the trace event into printk buffer
  - This is important if the kernel boot-process has a bug and can not reach login
- Enable events (`trace_events=`)
  - Enable trace events
  - Wild card is available (e.g. initcall:*)
- Add kprobe events (`kprobe_events=`)
  - Add new events at everywhere by kprobes
  - Breakpoint like feature

Boot time **function-call** tracing

- Enable tracers (`ftrace=`)
  - Enable function tracer, function graph tracer, irq-off tracer etc.
  - One tracer can be enabled at once
    - Not able to run different tracers
- Function Filtering (`ftrace_filter=,ftrace_notrace=,ftrace_graph_filter=,ftrace_graph_notrace=`)
  - Set the function/function-graph tracer's filter by function name (wildcard is available)

Other **options** for boot time tracing

- Setup options (`trace_options=`*`OPTIONS`*)
  - Setup tracer options (output format, etc.)
  - E.g.
    - “stacktrace” will record stack trace for each events
    - “sym-addr” will show actual address with symbols
- And other options (`alloc_snapshot`, `traceoff_on_warning`, ...)
  - See Documentation/admin-guide/kernel-parameters.txt

Example of Kernel Cmdline Parameters

Grub configuration (grub.conf)

```
linux   /boot/vmlinuz-5.1 root=UUID=5a026bbb-6a58-4c23-9814-5b1c99b82338 ro
quiet splash tp_printk trace_options=”sym-addr” trace_clock=global
ftrace_dump_on_oops trace_buf_size=1M trace_event=initcall:*,exceptions:*
kprobe_event=”p:kprobes/myevent msleep $arg1;p:kprobes/myevent2
hrtimer_nanosleep $arg1”
```

Example output

```
[    1.179729] initcall_start: func=ipv6header_mt6_init+0x0/0x11
[    1.180493] initcall_finish: func=ipv6header_mt6_init+0x0/0x11 ret=0
[    1.181242] initcall_start: func=reject_tg6_init+0x0/0x11
[    1.181812] initcall_finish: func=reject_tg6_init+0x0/0x11 ret=0
[    1.182401] initcall_start: func=sit_init+0x0/0xcb
[    1.182893] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    1.183611] initcall_finish: func=sit_init+0x0/0xcb ret=0
[    1.184171] initcall_start: func=load_umh+0x0/0x80
[    1.185340] page_fault_kernel: address=0x4c2230 ip=__clear_user error_code=0x2
[    1.186155] page_fault_kernel: address=0x7ffe3d62b299 ip=copy_user_generic_string error_code=0x2
[    1.187044] page_fault_user: address=0x401d40 ip=0x401d40 error_code=0x14
[    1.187047] bpfilter: Loaded bpfilter_umh pid 108
[    1.187723] page_fault_user: address=0x4c16c8 ip=0x402231 error_code=0x6
[    1.188842] page_fault_user: address=0x7ffe3d62aff8 ip=0x402237 error_code=0x6
```

Kernel command line for boot-time tracing

Size and coding limitation

- kernel cmdline size is 256 bytes in minimum
  - Minimum, most arch support a larger size (4kB).
- kernel cmdline is written in a single line
  - Bootloader wrapper will allow us to write in several lines

Only partial features are supported

- It can be too complex to the single command line
- It is hard to support multiple instance and event actions
  - per-event filters/actions, instances, histograms.

Solutions?

1. Use initramfs
   - Too late for kernel boot time tracing

2. Expand kernel cmdline
   - It is not easy to write down complex tracing options on bootloader
     (Single line options is too simple)

3. Introduce new boot time data
   -> **Extra Boot Configuration**

Extra Boot Configuration

Introduce a new kernel cmdline extension: Extra Boot Configuration (a.k.a. “bootconfig”)

- A plain ascii text file of structured key-value list (like sysctl.conf)

```
key.word = value
key.word2 {
     word3 = value
     nested-key {
          array-value-key = item1, item2, “arg1,arg2”
     }
}
```

- Loaded with the initrd image when boot
- In-kernel APIs for flexible option parsing

Extra Boot Configuration Syntax

- Simple key-value set

```
KEY[.WORD[...]] [+|:]= VALUE[, VALUE2[...]][;][#COMMENT]
```

  - Single value or comma-separated values (Array)

```
key = value1, value2, value3
```

  - Same key-words can be merged, e.g.

```
key.word1 = value1
key.word2 = value2
```

    This can be written as;

```
key { word1 = value1; word2 = value2 }
```

Syntax (cont.)

- Value assignment
  - ‘=’ defines the value of the key

    ```
    key = foo
    ```
  - ‘:=’ overwrites the previous assignment

    ```
    key = foo; key := bar
    ```

    Is

    ```
    key = bar
    ```
  - ‘+=’ appends a value as an array element

    ```
    key = foo; key += bar
    ```

    Is same as;

    ```
    key = foo, bar
    ```

Syntax (cont.)

- Mixing key and value on the same key (5.14)
  - A parent key can have subkey and value

```
key = value
key.subkey = foo
key.subkey.subsubkey = bar
```

  - Note that you can **NOT** merge value and subkey by brace

```
key = value
key {
    value
    subkey = foo
}
```

Syntax (cont.)

- Comment
  - Shell script like comment is available

```
key = value # comment after key-value

# comment line1
# comment line2
key.subkey = bar
```

Expand kernel command line

You can write kernel command line with bootconfig.

- The configuration keys starting with “kernel.” are passed to kernel command line.

```
kernel.root = ”UUID=12345678-9abc-...
```

- Also “init.” are passed to kernel command line but after “--”. These are passed to init process (e.g. systemd)

```
init {
    splash
    quiet
}
```

How to pass the bootconfig

- Bootconfig text file will be appended to the initrd (initramfs)
- "(ksrc)/tools/bootconfig/bootconfig" command handles it.
  - Append bootconfig (-a)

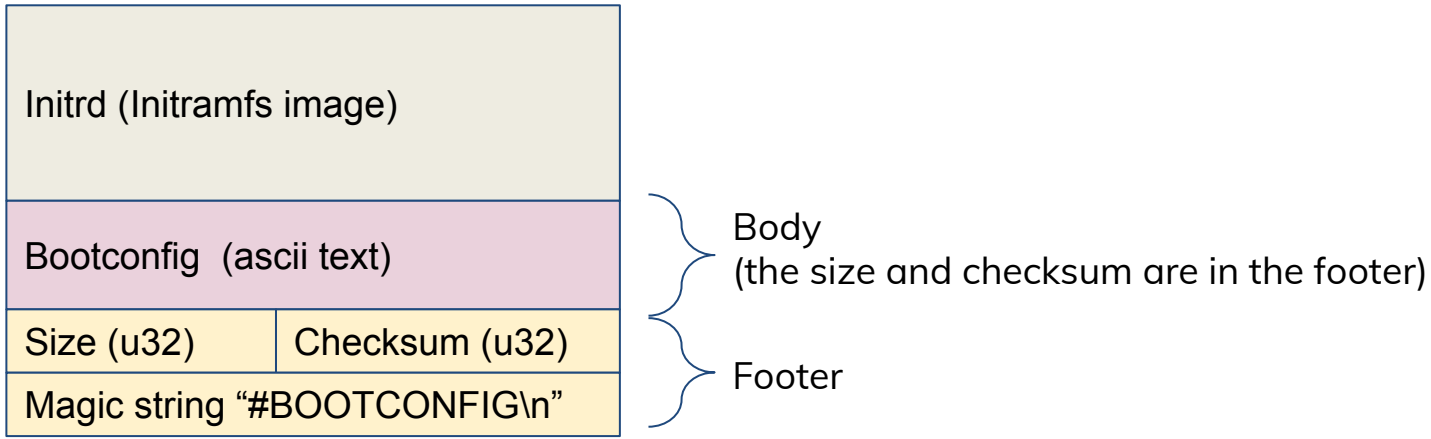
    ```
    # bootconfig -a your-boot-config system-initrd-file
    ```
  - Check current applied bootconfig

    ```
    # bootconfig system-initrd-file
    ```
  - Delete the bootconfig (-d)

    ```
    # bootconfig -d system-initrd-file
    ```

Bootconfig and initrd

The “*bootconfig*” command appends a config file to the initrd image.



Note: you also need “**bootconfig**” kernel command line option to enable bootconfig.

LF Live MENTORSHIP SERIES

Boot-time tracing expanded with Bootconfig

- Bootconfig based boot-time tracing
  - Kernel command line + boot-time tracing in ftrace
  - Configured via bootconfig
- Dedicated top-level key “**ftrace.**” in the bootconfig
  - Some existing ftrace kernel cmdline options are started with **“kernel.”**
- Support per-event and per-instance settings
- Kprobes, synthetic events, actions and histogram are available

See Documentation/trace/boottime-trace.rst for details

Kernel cmdline and bootconfig

```
tp_printk trace_options=”sym-addr” trace_clock=global ftrace_dump_on_oops trace_buf_size=1M
trace_event=initcall:*,exceptions:* kprobe_event=”p:kprobes/myevent msleep
$arg1;p:kprobes/myevent2 hrtimer_nanosleep $arg1”
```

Kernel cmdline and bootconfig

```
tp_printk trace_options=”sym-addr” trace_clock=global ftrace_dump_on_oops trace_buf_size=1M
trace_event=initcall:*,exceptions:* kprobe_event=”p:kprobes/myevent msleep
$arg1;p:kprobes/myevent2 hrtimer_nanosleep $arg1”
```

```
ftrace {
    options = sym-addr          # Symbol + address option to showing the address precisely
    trace_clock = global        # Use the global trace clock for synchronization
    buffer_size = 1MB           # Expand buffer size to 1MB because default size is too small
    event.kprobes {
        myevent.probes = "msleep $arg1"                    # trace msleep() …
        myevent2.probes = "hrtimer_nanosleep $arg1"    # trace hrtimer_nanosleep()
        enable
    }
    events = "initcall:*, exceptions:*"
}
kernel {
    tp_printk                   # Show events in dmesg so that we can see it on console
    ftrace_dump_on_oops         # If OOPS happens, dump trace buffer for safe
}
```

Example 1) Per-instance example

Run different tracers on boot

```
ftrace {
    buffer_size = 2MB                   # Expand buffer size to 1MB
    tracer = "function_graph"           # Run function-graph tracer on default instance
}
ftrace.instance.irqsoff {
    buffer_size = 100KB                 # Expand buffer size to 100KB
    tracer = "irqsoff"                  # Run irqsoff (interrupt-off) tracer on “irqsoff” instance
}
ftrace.instance.wakeup {
    buffer_size = 80KB                  # Expand buffer size to 80KB
    tracer = "wakeup"                   # Run wakeup (Wakeup latency) tracer on “wakeup” instance
}
```

Example 1 output

```
 # ls /sys/kernel/debug/tracing/instances/
irqsoff  wakeup
 # cat /sys/kernel/debug/tracing/current_tracer
function_graph
 # cat /sys/kernel/debug/tracing/instances/irqsoff/current_tracer
irqsoff
 # cat /sys/kernel/debug/tracing/instances/wakeup/current_tracer
wakeup
# cat /sys/kernel/debug/tracing/instances/wakeup/buffer_size_kb
80
 # cat /sys/kernel/debug/tracing/instances/irqsoff/trace
[...]
# latency: 3759 us, #275/275, CPU#2 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:8)
```

LF Live MENTORSHIP SERIES

Example 2) Function graph for specific area

Trace function call graph in pci_proc_init() function

```
ftrace {
    tracing_on = 0                    # Disable tracing by default
    tracer = "function_graph"         # Set a function call graph ready
    event {
        kprobes.pci_proc_init_begin {  # Add a kprobe event to start
                probes = "pci_proc_init"
                actions = "traceon"    # Enable tracing at call
        }
        kprobes.pci_proc_init_end { # Add a kprobe event to stop
                probes = "pci_proc_init%return"
                actions = "traceoff"   # Disable tracing at return
        }
    }
}
```

Example 2 output

```
/sys/kernel/debug/tracing # cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)   3.675 us    |  __rcu_read_unlock();
 4)               |  /* pci_proc_init_begin: (pci_proc_init+0x0/0x6c) */
 4)               |  pci_proc_init() {
 4)               |    proc_mkdir() {
 4)               |      _proc_mkdir() {
 4)               |        __proc_create() {
...
 4)   1.625 us    |      }
 4)   1.833 us    |    }
 0)               |          __mem_cgroup_charge.part.0() {
 4) ! 100.458 us  |  }
...
```

LF live MENTORSHIP SERIES

Example 3) Histogram

You can make a histogram on events for statistical analysis

```
ftrace.event {                                  # Possible to join key-words with “.”
        synthetic.initcall_latency {            # Add a synthetic event
                fields = "unsigned long func", "u64 lat"
                actions = "hist:keys=func.sym,lat:vals=lat:sort=lat"
        }
        initcall.initcall_start.actions = "hist:keys=func:ts0=common_timestamp.usecs"
        initcall.initcall_finish {
                actions =
"hist:keys=func:lat=common_timestamp.usecs-$ts0:onmatch(initcall.initcall_start).initcall_latenc
y(func,$lat)" # define long action command
        }
}
```

Example 3 output

```
/sys/kernel/debug/tracing # cat events/synthetic/initcall_latency/hist
...
{ func: [ffffffff827ab61b] cstate_pmu_init                    , lat:          0 } hitcount:          1  lat:      0
{ func: [ffffffff827e32db] of_platform_sync_state_init        , lat:          0 } hitcount:          1  lat:      0
{ func: [ffffffff827e6758] bpfilter_sockopt_init              , lat:          0 } hitcount:          1  lat:      0
{ func: [ffffffff827e5415] nf_nat_sip_init                    , lat:          0 } hitcount:          1  lat:      0
...
{ func: [ffffffff827daff5] virtio_pci_driver_init             , lat:      37020 } hitcount:          1  lat:  37020
{ func: [ffffffff827d79ae] acpi_init                          , lat:      58302 } hitcount:          1  lat:  58302
{ func: [ffffffff827d18a7] raid6_select_algo                  , lat:     103955 } hitcount:          1  lat: 103955

Totals:
    Hits: 462
    Entries: 462
    Dropped: 0
```

Boot-time tracing: Global options

- "**kernel.**" options: kernel cmdline options

kernel.tp_printk
Output trace-event data on printk buffer too.

kernel.dump_on_oops [= **MODE**]
Dump ftrace on Oops.

kernel.traceoff_on_warning
Stop tracing if WARN_ON() occurs.

kernel.fgraph_max_depth = **MAX_DEPTH**
Set MAX_DEPTH to maximum depth of fgraph tracer.

kernel.fgraph_filters = **FILTER**[, **FILTER2…**]
Specify function graph tracing function.

kernel.fgraph_notraces = **FILTER**[, **FILTER2…**]
Specify function graph non-tracing function.

Boot-time tracing: Instance options

- If you omit INSTANCE-NAME, default instance is used.
- If you configure a new instance, that is created automatically.

```
ftrace.instance.foo.tracing_on = 0    # create foo instance and tracing off
```

ftrace.[instance.*INSTANCE*.]options = **OPT1**[, **OPT2**[…]]

Enable given ftrace options.

ftrace.[instance.*INSTANCE*.]tracing_on = **0**|**1**

Enable/Disable tracing on this instance by default (enable dynamically by event action)

ftrace.[instance.*INSTANCE*.]trace_clock = **CLOCK**

Set given **CLOCK** to ftrace's trace_clock (local, global, mono, x86-tsc, etc.).

ftrace.[instance.*INSTANCE*.]buffer_size = **SIZE**

Configure ftrace buffer size to SIZE. You can use "KB" or "MB" for that SIZE.

Instance options (cont.)

ftrace.[instance.*INSTANCE*.]alloc_snapshot
Allocate snapshot buffer.

ftrace.[instance.*INSTANCE*.]cpumask = CPUMASK
Set CPUMASK as trace cpu-mask.(Bitmask)

ftrace.[instance.*INSTANCE*.]events = EVENT[, EVENT2[…]]
Enable given events on boot. You can use a wild card in EVENT.

ftrace.[instance.*INSTANCE*.]tracer = TRACER
Set TRACER to current tracer on boot. (e.g. function)

ftrace.[instance.*INSTANCE*.]ftrace.filters
This will take an array of tracing function filter rules.

ftrace.[instance.*INSTANCE*.]ftrace.notraces
This will take an array of NON-tracing function filter rules.

Per-event options

Per-event filter/actions and dynamic events are available

ftrace.[instance.*INSTANCE*.]event.[*GROUP*.[*EVENT*.]]enable
Enable GROUP:EVENT tracing. (per-group/all-events enablement is also available)

ftrace.[instance.*INSTANCE*.]event.*GROUP*.*EVENT*.filter = FILTER
Set FILTER rule to the GROUP:EVENT.

ftrace.[instance.*INSTANCE*.]event.*GROUP*.*EVENT*.actions = ACTION[, ACTION2[…]]
Set ACTIONs to the GROUP:EVENT.

ftrace.[instance.*INSTANCE*.]event.kprobes.*EVENT*.probes = PROBE[, PROBE2[…]]
Defines new kprobe event based on PROBEs.

ftrace.[instance.*INSTANCE*.]event.synthetic.*EVENT*.fields = FIELD[, FIELD2[…]]
Defines new synthetic event with FIELDs. Each field should be “type varname”.

LF Live MENTORSHIP SERIES

Boot-time tracing options in bootconfig: summary

- Most “**ftrace.**” options are corresponding to the tracefs (ftrace) interface (some subkeys changes singular/plural and are simplified).

  E.g.

  ftrace.instance.foo.options -> (tracefs)/instances/foo/trace_options

- Some options which tracefs does not expose, use kernel command line with “**kernel.**” keys.

  E.g.

  kernel.tp_printk, kernel.dump_on_oops

LF Live MENTORSHIP SERIES

Boot time tracing support in perf-probe

The latest kernel's perf tool support bootconfig format.

```
$ perf probe -h
...
      --bootconfig      Output probe definition with bootconfig format
...
```

This will help you to add a kprobe event by source-level information

E.g. probe the 55th line of cgroups_init() with 'ssid'

```
$ perf probe --bootconfig --definition "cgroup_init:55 ssid"

ftrace.event.kprobes.cgroup_init_L55.probe = 'cgroup_init+415 ssid=%r12:s32'
```

LF Live MENTORSHIP SERIES

Ftrace2bconf/bconf2ftrace

Shell scripts for the boot-time tracing under tools/bootconfig/scripts/

- `ftrace2bconf.sh`
  - Convert ftrace current setting to bootconfig [ftrace -> bootconfig]
  - For making a proto-type bootconfig from current ftrace settings (some options are not supported, you must check it)
- `bconf2ftrace.sh`
  - Convert given bootconfig to shell script for ftrace [bootconfig -> ftrace]
  - **--apply** option tries to apply the generated commands to ftrace
  - For checking your bootconfig before reboot

LF Live MENTORSHIP SERIES

Start Timing of Boot-time Tracing

Before Initcall	Bootconfig, Events (no tracefs), Cmdline options
Early initcall	kprobes
Core initcall	Func tracer, Graph tracer, Other tracers, trace_kprobe, trace_event, Boot time tracing
Postcore/arch/subsys initcall	Traceable
Fs initcall	Traceable, tracefs
Rootfs, device and late initcall	Traceable, Device drivers
After init	Traceable

Configuration setup order

Boot-time tracing has the setup order

1) Setup kernel cmdline options
2) Setup Global (default) instance
   a) Set instance options (buffer size etc.)
   b) Set per-event options (add kprobes, filters etc.)
   c) Enable events & tracers
3) Setup each instance
   a) Make instance, and set it up

E.g. you can not enable events before setting buffer size

Summary of usage

With following steps, you can run the boottime tracing on your kernel.

(1) Build and install kernel with CONFIG_BOOT_CONFIG=y and CONFIG_BOOTTIME_TRACING=y. And build `tools/bootconfig/bootconfig`.

(2) Setup ftrace as you like.

(3) Generate a bootconfig by `tools/bootconfig/scripts/ftrace2bconf.sh`

(4) Apply the config file to the initrd image as follows.

```
# bootconfig -a <configfile> <initrd>
```

(5) Reboot machine with “`bootconfig`” on the kernel command line

LF Live MENTORSHIP SERIES

Questions?

LF live MENTORSHIP SERIES

Resources

- Linux kernel command line options
  https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html
- Linux kernel Extra Boot Configuration documentation
  https://www.kernel.org/doc/html/latest/admin-guide/bootconfig.html
- Linux kernel Boot-time tracing documentation
  https://www.kernel.org/doc/html/latest/trace/boottime-trace.html

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.

LF *live*

MENTORSHIP SERIES