Kernel Boot-time Tracing

Linux Plumbers Conference 2019 - Tracing Track

Masami Hiramatsu <mhiramat@kernel.org>
Linaro, Ltd.

Speaker

Masami Hiramatsu

- Working for Linaro and Linaro members
- Tech Lead for a Landing team
- Maintainer of Kprobes and related tracing features/tools

Why Kernel Boot-time Tracing?

Debug and analyze boot time errors and performance issues

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

etc.

What We Have

There are already many ftrace options on 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, ...)

See Documentation/admin-guide/kernel-parameters.txt

Example of Kernel Cmdline Parameters

In 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:*,irq:*,exceptions:*”
kprobe_event=”p:kprobes/myevent foofunction $arg1
$arg2;p:kprobes/myevent2 barfunction %ax”
```

What Issues?

Size limitation

- kernel cmdline size is small (< 256bytes)
- A half of the cmdline is used for normal boot

Only partial features supported

- ftrace has too complex features for single command line
- 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. Reuse structured boot time data (Devicetree)
   - Well documented, structured data
   -> V1 & V2 series based on this.

Boot-time Trace: V1 and V2 series

V1 and V2 series posted at June.

These series was based on “devicetree”

- Devicetree(DT) is well structured data to be passed to the kernel at boot time.
- Stable and good user-space tools
- Well documented with YAML schema
- Bootloaders are already supported
- Some architecture requires it to boot

Discussed and Rejected

- Devicetree is standardized and documented as **hardware** description.
- Devicetree is **NOT** for configuration information.

V3: Introduce Supplemental Kernel Cmdline

Introduced a new kernel cmdline extension: Supplemental kernel cmdline (SKC)

- A plain ascii text of tree-structured key-value list

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

- Loaded by bootloaders and parsed at early stage
  - Still need some work on bootloaders and Qemu (for test)

Demo

Qemu -skc option

- Show SKC file
- /proc/sup_cmdline
- Instance based tracing
- Per-event setting
- Histogram

Current Status

What's done

- RFC patches with SKC
- Qemu -skc option implementation (for x86)
- Grub "skc" command implementation (for x86)

TODOs

- "Actions" syntax expansion for histogram
- Initialize tracing earlier
- Userspace tool for writing SKC file
    - e.g. setting emulation, perf-probe integration etc.

- SKC and kernel cmdline integration (out of scope of this session)
- More bootloader support (out of scope of this session)

Event Histogram Update

Current

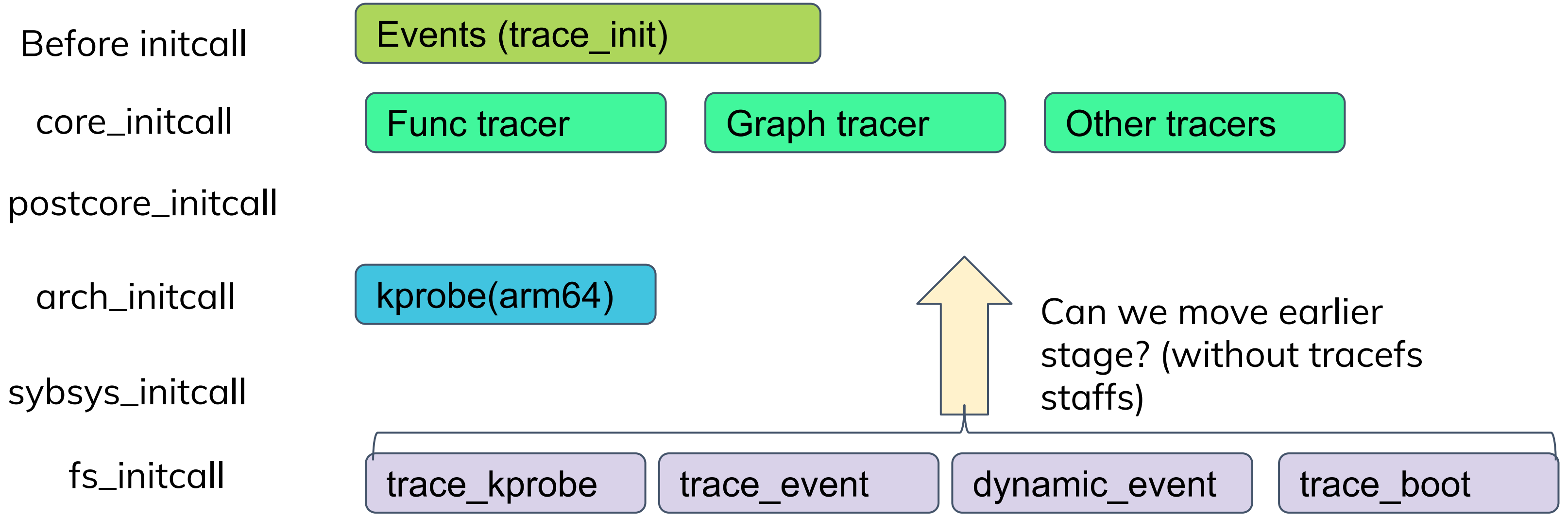
```
        initcall.initcall_finish {
                actions =
"hist:keys=func:lat=common_timestamp.usecs-$ts0:onmatch(initcall.initcall_start).initcall_latency
(func,$lat)";
        }
```

Proposal

```
        initcall.initcall_finish {
                hist {
                      keys = func;
                      lat = common_timestamp.usecs-$ts0;
                      handler = onmatch(initcall.initcall_start);
                      action = initcall_latency(func,$lat);
                 }
        }
```

Earlier Tracer Initialization

Can we initialize tracer in earlier stage?



Questions?

Thank you

Linaro

Backup slides

How to Use SKC at Boot?

Grub

1. Write an SKC file under /boot (e.g. /boot/ftrace.skc)
2. Use “skc” command to load from grub console (or grub.cfg)
   E.g. “skc /boot/ftrace.skc”
3. Boot.

Qemu

1. Write and SKC file
2. Pass the SKC file by -skc option. (Only with -kernel option)
   E.g. “qemu-system-x86_64 -kernel vmlinux -skc ftrace.skc …”

Requirements for Boot-time Tracing

- Start before init process
- Existing kernel cmdline compatible
- Per-event filter and actions support
- Per-instance settings support
- Kprobes and synthetic dynamic events support
- Easy to read/write programming interface

-> If we can use some settings on boot, we can setup ftrace.

Boot-time Tracing with SKC

- All options start from “ftrace.”
- Existing ftrace kernel cmdline options are supported
- Per-event and per-instance settings are naturally embedded in the key.

```
“ftrace.event.GROUP.EVENT.filter = FILTER”
“ftrace.instance.INSTANCE.buffer_size = SIZE”
```

- Kprobes and synthetic events

```
“ftrace.event.kprobes.EVENT.probes = PROBE_DEFINITION[, …]”
“ftrace.event.synthetic.EVENT.fields = SYNTH_FIELDS[,...]”
```

SKC based Boot-time Tracing (1)

```
ftrace {

    options = symaddr;
    buffer_size = 1MB;
    tp_printk;

    event.kprobes.vfs_read {
                probes = "kernel_read $arg1 $arg2";
                filter = "common_pid < 200";
                enable;
    }
}
```

<- Support normal command line options.

<- User can add kprobe events
<- Per-event filter is available
<- Per-event enable/disable is controllable

SKC based Boot-time Tracing (2)

It is possible to setup histogram event actions (usually involving several events)

```
ftrace.event {
       synthetic.initcall_latency {                                  <- Define 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-$ts0:onmatch(initcall.initcall_start).initcall_latency
(func,$lat)";
        }
}
```