Boot-time tracing

Author: Masami Hiramatsu <mhiramat@kernel.org>

Overview

Boot-time tracing allows users to trace boot-time process including device initialization with full features of ftrace including per-event filter and actions, histograms, kprobe-events and synthetic-events, and trace instances. Since kernel command line is not enough to control these complex features, this uses bootconfig file to describe tracing feature programming.

Options in the Boot Config

Here is the list of available options list for boot time tracing in boot config file [1]. All options are under "ftrace." or "kernel." prefix. See kernel parameters for the options which starts with "kernel." prefix [2].

[1] See :ref:`Documentation/admin-guide/bootconfig.rst <bootconfig>`

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\trace\[linux-master][Documentation][trace]boottime-trace.rst`, line 27); *backlink*
>
> Unknown interpreted text role "ref".

[2] See :ref:`Documentation/admin-guide/kernel-parameters.rst <kernelparameters>`

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\trace\[linux-master][Documentation][trace]boottime-trace.rst`, line 28); *backlink*
>
> Unknown interpreted text role "ref".

Ftrace Global Options

Ftrace global options have "kernel." prefix in boot config, which means these options are passed as a part of kernel legacy command line.

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

kernel.dump_on_oops [= MODE]
: Dump ftrace on Oops. If MODE = 1 or omitted, dump trace buffer on all CPUs. If MODE = 2, dump a buffer on a CPU which kicks 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...]
: Add fgraph tracing function filters.

kernel.fgraph_notraces = FILTER[, FILTER2...]
: Add fgraph non-tracing function filters.

Ftrace Per-instance Options

These options can be used for each instance including global ftrace node.

ftrace.[instance.INSTANCE.]options = OPT1[, OPT2[...]]
: Enable given ftrace options.

ftrace.[instance.INSTANCE.]tracing_on = 0|1
: Enable/Disable tracing on this instance when starting boot-time tracing. (you can enable it by the "traceon" event trigger action)

ftrace.[instance.INSTANCE.]trace_clock = CLOCK
: Set given CLOCK to ftrace's trace_clock.

ftrace.[instance.INSTANCE.]buffer_size = SIZE
: Configure ftrace buffer size to SIZE. You can use "KB" or "MB" for that SIZE.

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

ftrace.[instance.INSTANCE.]cpumask = CPUMASK
: Set CPUMASK as trace cpu-mask.

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.

Ftrace Per-Event Options

These options are setting per-event options.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.enable
: Enable GROUP:EVENT tracing.

ftrace.[instance.INSTANCE.]event.GROUP.enable
: Enable all event tracing within GROUP.

ftrace.[instance.INSTANCE.]event.enable
: Enable all event tracing.

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. It is able to define multiple probes on one event, but those must have same type of arguments. This option is available only for the event which group name is "kprobes".

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

Note that kprobe and synthetic event definitions can be written under instance node, but those are also visible from other instances. So please take care for event name conflict.

Ftrace Histogram Options

Since it is too long to write a histogram action as a string for per-event action option, there are tree-style options under per-event 'hist' subkey for the histogram actions. For the detail of the each parameter, please read the event histogram document (Documentation/trace/histogram.rst)

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]keys = KEY1[, KEY2[...]]
: Set histogram key parameters. (Mandatory) The 'N' is a digit string for the multiple histogram. You can omit it if there is one histogram on the event.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]values = VAL1[, VAL2[...]]
: Set histogram value parameters.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]sort = SORT1[, SORT2[...]]
: Set histogram sort parameter options.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]size = NR_ENTRIES
: Set histogram size (number of entries).

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]name = NAME
: Set histogram name.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]var.VARIABLE = EXPR
: Define a new VARIABLE by EXPR expression.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]<pause|continue|clear>
: Set histogram control parameter. You can set one of them.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]onmatch.[M.]event = GROUP.EVENT
: Set histogram 'onmatch' handler matching event parameter. The 'M' is a digit string for the multiple 'onmatch' handler. You can omit it if there is one 'onmatch' handler on this histogram.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]onmatch.[M.]trace = EVENT[, ARG1[...]]
: Set histogram 'trace' action for 'onmatch'. EVENT must be a synthetic event name, and ARG1... are parameters for that event. Mandatory if 'onmatch.event' option is set.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]onmax.[M.]var = VAR
: Set histogram 'onmax' handler variable parameter.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]onchange.[M.]var = VAR
: Set histogram 'onchange' handler variable parameter.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]<onmax|onchange>.[M.]save = ARG1[, ARG2[...]]
: Set histogram 'save' action parameters for 'onmax' or 'onchange' handler. This option or below 'snapshot' option is mandatory if 'onmax.var' or 'onchange.var' option is set.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.[N.]<onmax|onchange>.[M.]snapshot
: Set histogram 'snapshot' action for 'onmax' or 'onchange' handler. This option or above 'save' option is mandatory if 'onmax.var' or 'onchange.var' option is set.

ftrace.[instance.INSTANCE.]event.GROUP.EVENT.hist.filter = FILTER_EXPR
: Set histogram filter expression. You don't need 'if' in the FILTER_EXPR.

Note that this 'hist' option can conflict with the per-event 'actions' option if the 'actions' option has a histogram action.

When to Start

All boot-time tracing options starting with `ftrace` will be enabled at the end of core_initcall. This means you can trace the events from postcore_initcall. Most of the subsystems and architecture dependent drivers will be initialized after that (arch_initcall or subsys_initcall). Thus, you can trace those with boot-time tracing. If you want to trace events before core_initcall, you can use the options starting with `kernel`. Some of them will be enabled eariler than the initcall processing (for example,. `kernel.ftrace=function` and `kernel.trace_event` will start before the initcall.)

Examples

For example, to add filter and actions for each event, define kprobe events, and synthetic events with histogram, write a boot config like below:

```
ftrace.event {
      task.task_newtask {
              filter = "pid < 128"
              enable
      }
      kprobes.vfs_read {
              probes = "vfs_read $arg1 $arg2"
              filter = "common_pid < 200"
              enable
      }
      synthetic.initcall_latency {
              fields = "unsigned long func", "u64 lat"
              hist {
                      keys = func.sym, lat
                      values = lat
                      sort = lat
              }
      }
      initcall.initcall_start.hist {
              keys = func
              var.ts0 = common_timestamp.usecs
      }
      initcall.initcall_finish.hist {
              keys = func
              var.lat = common_timestamp.usecs - $ts0
              onmatch {
                      event = initcall.initcall_start
                      trace = initcall_latency, func, $lat
              }
      }
}
```

Also, boot-time tracing supports "instance" node, which allows us to run several tracers for different purpose at once. For example, one tracer is for tracing functions starting with "user_", and others tracing "kernel_" functions, you can write boot config as below:

```
ftrace.instance {
      foo {
              tracer = "function"
              ftrace.filters = "user_*"
      }
      bar {
              tracer = "function"
              ftrace.filters = "kernel_*"
      }
}
```

The instance node also accepts event nodes so that each instance can customize its event tracing.

With the trigger action and kprobes, you can trace function-graph while a function is called. For example, this will trace all function calls in the pci_proc_init():

```
ftrace {
      tracing_on = 0
      tracer = function_graph
      event.kprobes {
              start_event {
                      probes = "pci_proc_init"
                      actions = "traceon"
              }
              end_event {
                      probes = "pci_proc_init%return"
                      actions = "traceoff"
              }
      }
}
```

This boot-time tracing also supports ftrace kernel parameters via boot config. For example, following kernel parameters:

```
trace_options=sym-addr trace_event=initcall:* tp_printk trace_buf_size=1M ftrace=function ftrace_filter="v
```

This can be written in boot config like below:

```
kernel {
        trace_options = sym-addr
        trace_event = "initcall:*"
        tp_printk
        trace_buf_size = 1M
        ftrace = function
        ftrace_filter = "vfs*"
    }
```

Note that parameters start with "kernel" prefix instead of "ftrace".