Linux ftrace Function Counting

13 Jul 2014

Here's another capability I didn't think was possible. What kernel bio functions are being called?

```
# ./funccount 'bio_*'
Tracing "bio_*"... Ctrl-C to end.
^C
FUNC                              COUNT
bio_attempt_back_merge               26
bio_get_nr_vecs                     361
bio_alloc                           536
bio_alloc_bioset                    536
bio_endio                           536
bio_free                            536
bio_fs_destructor                   536
bio_init                            536
bio_integrity_enabled               536
bio_put                             729
bio_add_page                       1004
```

Great! How about the top 5 functions beginning with "tcp_" every second?

```
# ./funccount -i 1 -t 5 'tcp_*'
Tracing "tcp_*". Top 5 only... Ctrl-C to end.

FUNC                              COUNT
tcp_cleanup_rbuf                    386
tcp_service_net_dma                 386
tcp_established_options             549
tcp_v4_md5_lookup                   560
tcp_v4_md5_do_lookup                890

FUNC                              COUNT
tcp_service_net_dma                 498
tcp_cleanup_rbuf                    499
tcp_established_options             664
tcp_v4_md5_lookup                   672
tcp_v4_md5_do_lookup               1071

[...]
```

Awesome!

Like my previous post on perf Hacktograms, this capability is also bread-and-butter for advanced tracers like SystemTap, ktap, and DTrace. But I'm not using those. I'm not even using perf_events.

This is using **dynamic tracing** and **by-CPU in-kernel aggregations** on a standard Linux 3.2 kernel.

It does this using ftrace, automated by my funccount script.

ftrace is part of the Linux kernel, and is included at compile time by various FTRACE CONFIG options (including CONFIG_DYNAMIC_FTRACE, which on my systems is already turned on). You can operate it via control files under /sys/kernel/debug/tracing. It's a little tricky to use, but gets many jobs done. See the kernel source documentation trace/ftrace.txt for details.

Why

Understanding function call rates can be a useful tool for debugging and performance analysis. It's also part of a regular procedure I use when tracing subsystems, especially unfamiliar ones.

Dynamic tracing is amazing, but it can be hard to know where to start, especially when faced with hundreds of functions in a subsystem of interest. By counting which functions are actually in use, you can narrow down the potential targets. So you can use funccount to find active functions, and then other tracers (including perf_events dynamic tracing) to probe them in more detail.

Another approach for identifying active kernel functions is to create a perf_events CPU Flame Graph based on stack trace samples. I'll usually start with this, since it costs lower (fixed) overhead, relative to the sample rate, and can also solve numerous problems right off the bat. I'd move onto function counts when I'm digging deeper into a subsystem.

funccount

This is a simple script to automate ftrace. It does one thing: kernel function counting.

```
# ./funccount -h
USAGE: funccount [-hT] [-i secs] [-d secs] [-t top] funcstring
                 -d seconds      # total duration of trace
                 -h              # this usage message
                 -i seconds      # interval summary
                 -t top          # show top num entries only
                 -T              # include timestamp (for -i)
  eg,
       funccount 'vfs*'          # trace all funcs that match "vfs*"
       funccount -d 5 'tcp*'     # trace "tcp*" funcs for 5 seconds
       funccount -t 10 'ext3*'   # show top 10 "ext3*" funcs
       funccount -i 1 'ext3*'    # summary every 1 second
       funccount -i 1 -d 5 'ext3*' # 5 x 1 second summaries
```

I added it to my perf-tools collection on github.

It works by enabling the ftrace function profiler. It creates per-CPU summaries (which are efficient – no synchronization overheads when updating counts), which funccount combines for the report. ftrace already provides the ability to match wildcards for its function filter.

Not all functions are visible from ftrace and funccount. If you think a function should be included in the output, but is missing, check for it in /proc/kallsyms (kernel symbols) and /sys/kernel/debug/tracing/available_filter_functions (what ftrace can trace).

Dynamic tracing of kernel functions has been problematic in the past on Linux, with the risk of kernel panics. I haven't experienced them (I've run this script on 3.2 and 3.16), but wanted to warn you of this anyway: I'd use a test machine (with load) to try this out on first.

Thanks to Steven Rostedt and others who have been busy adding great capabilities to ftrace.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).