Contribution ID: **10** Type: **not specified**

Tracing Microconference

Tracing in the Linux kernel is constantly improving. Since 2008 tracing has been add to Linux, and more tooling is constantly being added to help out with visibility. The work is still ongoing, with Perf, ftrace, Lttng, and eBPF. User space tooling is expanding and as the kernel gets more complex, so does the need for facilitating seeing what is going on under the hood.

The last tracing microconference that happened two years ago was extremely productive:

- The final design of bootconfig[1] came out, which enables kernel command lines be attached to the init ramdisk.
- Discussion on how to simplify the interface to ftrace histograms[2] from userspace resulted in a SQL like utility (still being worked on, but almost finished)[3]. This came from the help of the database folks.
- After several rounds of trying to have perf share PMUs (beyond the hardware limit)[4], another approach was taken to use a BPF based solution that does not need any kernel changes. Now perf can use BPF to aggregate counters[5].

[1] https://www.kernel.org/doc/html/latest/admin-guide/bootconfig.html
[2] https://www.kernel.org/doc/html/latest/trace/histogram.html
[3] https://github.com/rostedt/sqlhist
[4] https://lore.kernel.org/lkml/168F3761-98CF-4E91-B911-ECB9FCD68F0C@fb.com/T/
[5] https://lore.kernel.org/lkml/20210425214333.1090950-5-song@kernel.org/

Possible topics for this year:

- Tracepoints that allow faults[1]. It may be necessary to read user space address, but currently because tracepoints disable preemption, it can not sleep, nor fault. And then there's the possibilities of causing locking issues.
- Function parameter parsing[2]. Now that on x86 function tracing has full access to the arguments of a function, it is possible to record them as they are being traced. But knowing how to read the parameters may be difficult, because it is necessary to know the prototype of the function to do so. Having some kind of mapping between functions and how to read their parameters would be useful. Using BTF[3] is a likely candidate.
- Consolidating tracing of return of a function. Currently there's three use cases that hook to the return of a function, and they all do it differently. kretprobes[4], function graph tracer[5], and eBPF[6].
- User space libraries. Now that libtraceevent[7], libtracefs[8], and libtracecmd[9] have been released, what tooling can be built around them. Also, improving the libtraceevent API to be more intuitive.
- Improving the libtracefs API to handle kprobes[10] and uprobes[11] easier
- Python interface. Working on getting the libraries a python interface to allow full tracing from within python scripts.
- Tracing containers. What would be useful to expose on creating and running containers.
- And more.

Key people:

- Steven Rostedt
- Mathieu Desnoyers
- Peter Zijlstra
- Alexei Starovoitov
- Thomas Gleixner
- Masami Hiromatsu
- Yordan Karadzhov
- Tzvetomir Stoyanov
- Dario Faggioli

[1] https://lwn.net/Articles/835426/
[2] https://lwn.net/Articles/835571/
[3] https://www.kernel.org/doc/html/latest/bpf/btf.html
[4] https://embeddedguruji.blogspot.com/2018/12/debugging-linux-kernel-using-kretprobes.html
[5] https://www.kernel.org/doc/html/v4.18/trace/ftrace.html#function-graph-tracer
[6] https://lwn.net/Articles/803803/
[7] https://trace-cmd.org/Documentation/libtraceevent/libtraceevent.html
[8] https://trace-cmd.org/Documentation/libtracefs/libtracefs.html
[9] https://manpages.debian.org/experimental/trace-cmd/libtracecmd.3.en.html
[10] https://www.kernel.org/doc/Documentation/kprobes.txt
[11] http://www.brendangregg.com/blog/2015-06-28/linux-ftrace-uprobe.html

I agree to abide by the anti-harassment policy

I agree

Primary authors: ROSTEDT, Steven; KARADZHOV, Yordan (VMware)

Session Classification: Tracing MC

Track Classification: Tracing MC