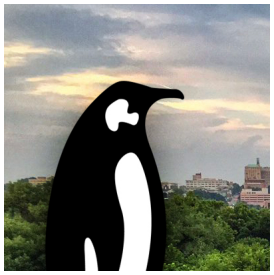

Contribution ID: **35** Type: **not specified**

Tracing Microconference

The Linux kernel has grown in complexity over the years. Complete understanding of how it works via code inspection has become virtually impossible. Today, tracing is used to follow the kernel as it performs its complex tasks. Tracing is used today for much more than simply debugging. Its framework has become the way for other parts of the Linux kernel to enhance and even make possible new features. Live kernel patching is based on the infrastructure of function tracing, as well as BPF. It is now even possible to model the behavior and correctness of the system via runtime verification which attaches to trace points. There is still much more that is happening in this space, and this microconference will be the forum to explore current and new ideas.

Results and accomplishments from the last time (2021):

- User events were introduced, and have finally made it into the kernel
- The discussion around trace events to handle user faults initiated the event probe work around to the problem. That was to add probes on existing trace events to change their types. This works on synthetic events that can pass the user space file name of the entry of a system call to the exit of the system call which would have faulted in the file and make it available to the trace event.
- Dynamically creating the events directory is currently being worked on with the eventfs patch set. This will save memory as the dentries and inodes will only be allocated when accessed.
- The discussion about function tracing with arguments has helped inspire both fprobes and function graph return value tracing.
- There's still ongoing effort in merging the return path tracers of function graph and kretprobes and fprobes.

Topics for this year:

- Use of sframes. How to get user space stack traces without requiring frame pointers.
- Updating perf and ftrace to extract user space stack frames from a schedulable context (as requested by NMI).
- Extending user events. Now that they are in the kernel, how to make them more accessible to users and applications.
- Getting more use cases with the runtime verifier. Now that the runtime verifier is in the kernel (uses tracepoints to model against), what else can it be used for.
- Wider use of ftrace_regs in fprobes and rethook from fprobes because rethook may not fill all registers in pt_regs too. How BPF handles this will also be discussed.
- Removing kretprobes from kprobes so that kprobe can focus on handling software breakpoint.
- Object tracing (following a variable throughout each function call). This has had several patches out, but has stopped due to hard issues to overcome. A live discussion could possibly come up with a proper solution.
- Hardware breakpoints and tracing memory changes. Object tracing follows a variable when it changes between function calls. But if the hardware supports it, tracing a variable when it actually changes would be more useful albeit more complex. Discussion around this may come up with a easier answer.
- MMIO tracer being used in SMP. Currently the MMIO tracer does not handle race conditions. Instead, it offlines all but one CPU when it is enabled. It would be great if this could be used in normal SMP environments. There's nothing technically preventing that from happening. It only needs some clever thinking to come up with a design to do so.

- Getting perf counters onto the ftrace ring buffer. Ftrace is designed for fast tracing, and perf is a great profiler. Over the years it has been asked to have perf counters along side ftrace trace events. Perhaps its time to finally accomplish that. It could be that each function can show the perf cache misses of that function.

Key attendees:
- Steven Rostedt
- Masami Hiramtsu
- Mathieu Desnoyers
- Alexei Starovoitov
- Peter Zijlstra
- Mark Rutland
- Beau Belgrave
- Daniel Bristot de Oliveira
- Florent Revest
- Jiri Olsa
- Tom Zanussi

Primary authors: Mr HIRAMATSU, Masami (Google); ROSTEDT, Steven

Track Classification: LPC Microconference Proposals