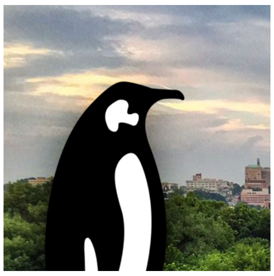

Contribution ID: **14** Type: **not specified**

Linux Kernel Debugging Microconference

When things go wrong, we need to debug the kernel. There are about as many ways to do that as you can imagine: printk, kdb/kgdb over serial, tracing, attaching debuggers to /proc/kcore, and post-mortem debugging using core dumps, just to name a few. Frequently, tools and approaches used by userspace debuggers aren't enough for the requirements of the kernel, so special tools are created to handle them: crash, drgn, makedumpfile, libkdumpfile, and many, many others.

With the variety of tools and approaches available, it's important to collaborate on whatever shared problems we may have. This microconference is an opportunity to discuss these problems and come up with shared approaches to resolve them. Some examples of potential topic areas:

- Many debuggers understand core subsystems such as tasks, slab caches, mm_structs, etc, and provide information about them. But as the kernel evolves, code changes can break these tools, which need to contain decades of cruft to handle a variety of versions. How can we improve processes and tools so that the future decades of evolution can be handled without crippling our debuggers with more technical debt?

How can we share logic between debuggers to reduce duplicate effort in interpreting core kernel data structures?

Please see Philipp Rudo's excellent talk from LPC 2022 regarding this very topic.

- Kernel core dumps can come from a variety of sources: some are generated via kexec and /proc/vmcore, then makedumpfile. Others may be created by a variety of hypervisors including Qemu, Xen, and Hyper-V. The core dumps can use ELF, or more commonly, the compressed diskdump family of formats. With the variety of core dump producers and consumers, along with the variation in formats, it's not uncommon to encounter "broken" core dumps which need tweaks or additional tools to be read. How can we build tools to handle the diversity of core dumps, more easily fix broken ones, and guide the community to a better documented standard?
- Kernel debuggers rely on debuginfo such as DWARF, which can be bulky and is not commonly distributed alongside the kernel. How can we enable lightweight debugging options that run everywhere?
- When debugging kernel-related issues on live systems, stack unwinding of both kernel and userspace tasks is important. As it is, stack unwinding in the kernel can be done via frame pointers and ORC on x86_64, but userspace stack unwinding is more difficult, since many applications and libraries are compiled without frame pointers, and the kernel lacks a DWARF-based unwinder. What can the kernel debugging and tracing community do to improve this situation?

Below are key people who may be interested in joining the discussions and/or presenting on relevant debugging topics:

- Omar Sandoval
- Petr Tesarik
- Philipp Rudo

Primary author: BRENNAN, Stephen (Oracle)

Track Classification: LPC Microconference Proposals