Syscall optimization in libvolimem

- Advisors: Jana Toljaga, Gaël Thomas
- Location: Telecom building, Palaiseau

Context

Libvolimem addresses a problem faced by the research community in operating systems: today, Linux is so complex and evolves so fast that conducting research at the operating system kernel level has become especially difficult. To illustrate, modifying the way Linux manages memory requires digging into the 194,961 lines of code in the Linux mm directory. This quickly becomes a nightmare because this code has to maintain numerous complex invariants so that the 25 million lines of Linux code remain correct. As a result, testing a new idea is becoming unrealistic within a PhD timeframe, and when it is done anyway, the research prototype quickly becomes outdated an impossible to compare to other research projects because a new major version of Linux is released every two months.

With libvolimem, our idea is that if we cannot easily modify Linux, we can complement it with a second kernel that replaces the exact feature that we want to implement. For that, we propose relying on virtualization. Thanks to virtualization, we can start a guest operating system on top of a host operating system. The guest operating system has direct access to low-level hardware features, such as a page table or an interrupt descriptor table, and can implement its own system policies to manage hardware. With libvolimem, we thus activate virtualization, but instead of running a second fully-fledged operating system inside the virtual machine, libvolimem runs a lightweight operating system that simply delegates any system call to the host Linux. That way, libvolimem acts as an interposition layer, which allows researchers to explore new system ideas easily by replacing any original Linux mechanism with their own implemented in the virtual machine. This significantly simplifies conducting experiments, as libvolimem is 8,000 lines of code instead of the 25 million lines of code of Linux.

Project

To call the operating system, libvolimem relies on sibling threads. In detail, each thread of the application runs inside the virtual machine and is associated with a sibling thread running natively. To call the system, the application thread sends a message in shared memory to its sibling thread, which performs the syscall on behalf of the application thread. This design has two main limitations. First, it wastes hardware resources since libvolimem doubles the number of threads. Second, it is inefficient because of the cost of sending the message between the application thread and the sibling thread, and then performing the system call from the sibling thread.

This project aims thus to optimize this infrastructure step-by-step: * First, by

adapting the number of sibling threads to the actual workload on the fly, * Then, by replacing the sibling threads with the io_uring infrastructure for the syscalls supported by io_uring (e.g., read, write) * Finally, by implementing a new infrastructure similar to io_uring inside the kernel to handle any syscall