Group 8:
Rob Stanford
Maria Imperatrice
Jorge Vazquez

# Userspace Bypass: Accelerating Syscall-intensive Applications - Summary

*What is the problem being solved by the paper?*

Context switches between user mode and kernel mode often creates significant overhead, creating a slowdown in applications that utilize system calls (syscalls). This paper aims to address some of the discussed issues with syscalls slowing down performance due to the overhead of context switches through an approach called userspace bypass (UB). Previous studies have shown that there are both indirect and direct costs resulting from this overhead. The direct costs include extra procedures that need to be executed in order to save registers, change protection domains, and handle the registered exceptions. The indirect costs of this can be seen when the syscalls disrupt the state of the CPU by “polluting” cache data and translation lookaside buffers (TLB). User-mode instructions per cycle (IPC) are decreased as a result of this overhead.

Additionally, several techniques also contribute to the slowdown of syscalls. This includes kernel page-table isolation (KPTI), where two sets of page tables for user space and kernel space are utilized by the OS kernel, resulting in the CPU switching to the kernel page when a syscall is invoked. Virtualization can also contribute to this overhead as an indirect cost. The UB mechanism proposed by the authors aims to reduce the overhead caused by syscall related I/O.

*Why is this problem important?*

The goal of reducing the overhead cost of switching between user mode and kernel mode is not something new. The motivation that drives to increase the performance of syscalls is due to programs that require high I/O requests per second (IOPS). They noted that there have been recent attempts to reduce this overhead cost. One was by moving the data processing into the kernel space, or by moving the drivers responsible for I/O into the user mode. This directly reduces the overhead by reducing the number of times a context switch needs to occur. The other noted method was by batching syscalls and to queue up multiple I/O requests so that the number of syscalls is also reduced. The main issue with these approaches is that it requires the developers to apply changes to their code, which often is not an easy feat. Their approach is motivated by the idea to not require the developers to change their application code (known as binary compatibility) and reduce overhead by introducing userspace bypass.

*What are the authors proposing?*

The authors are proposing an approach called userspace bypass (UB) that not only reduces overhead introduced by syscall related I/O, but that is also binary compatible. This paper proposes three major design goals: the first is to minimize the manual efforts of developers by optimizing the syscall at the execution time, the second is to minimizing changes

to system architecture, and the third is to provide a comparable performance to syscall-refactoring approaches by reducing the direct and indirect costs of syscalls. UB is accomplished through a mechanism that can translate userspace instructions into kernel-safe sanitized Binary Translation Cache (BTC). The authors implement a prototype that is then evaluated against several high input/output operations per second (IOPS) apps to show its effectiveness.

*How do the authors test their idea, and what do they find?*

The group evaluated their design and implementation of Userspace Bypass by measuring the performance across a set of different benchmarks. For each of the benchmarks their testing methodology was to evaluate results on both physical hardware and from within a virtual machine and to run with KPTI (Kernel Page Table Isolation) enabled and disabled. Ten rounds of each benchmark were conducted and the results averaged.

They tested file I/O performance via an I/O micro-benchmark of read syscall, as well as comparing IOPS from an NVMe SSD. The group also compared socket performance using raw socket packet processing (with their changes and without) against other packet processing software like eBPF (extended Berkeley Packet Filter) packet processing.

To understand the impact of the Userspace Bypass at a higher system level they used Redis and Nginx to serve as macro-benchmarks. They set up a typical server configuration for each and ran their included benchmark tools.

Overall, the group found noted performance improvements when using their Userspace Bypass. Depending on the benchmark, anywhere from -6.4% to 112.9% performance gains. Some solutions like eBPF provide even faster possible performance, but come with some restrictions in where and how it can be used. The great benefit of the Userspace Bypass solution is the performance improvements are all gained without any development effort; by an operating system adopting Userspace Bypass all programs and processes stand to benefit with no source changes.