Paper 1

Summary

The paper tries to address the fundamental cost of IPC in microkernels by removing the microkernel itself from the path of IPC. It does this by allowing processes to switch their address space to a list of predefined page tables that correspond to address spaces of servers that can be called. The second-level address translation is used as a hardware-accelerated mechanism to provide user-space processes with the ability to switch page tables (the VMFUNC instruction) whilst not giving up complete control of memory management and protection (i.e. address space switching is possible only within a predefined list provisioned by the VMM.)

The motivation for this work is that the cost of IPC where the kernel is involved is necessarily bounded by the cost of kernel invocation. Kernel invocation instructions are expensive, and entering and executing kernel code pollutes architectural caches. Bypassing the kernel invocation allows this cost to be eliminated.

Their implementation of IPC inserts a virtual machine monitor called the Rootkernel beneath existing microkernels to be called the Subkernel. The Rootkernel is solely responsible for the provisioning of the EPT infrastructure, and exists because virtualisation is necessary to use the processor's EPT functionality. Most VMM traps, such as syscall traps, are disabled. To reduce the cost of second-level address translation, the first-level is configured to use hugepages as if providing the Subkernel with a contiguous memory.

To perform IPC, the Subkernel and Rootkernel collaborate to initialise a process's EPT list with all possible IPC invocation destinations, along with trampoline code mappings. A client calls the kernel-provided trampoline to execute the VMFUNC instruction and subsequent method invocation in a restricted manner. For an arbitrary binary, the VMFUNC instruction is replaced at load time to prevent unrestricted address space manipulation into other processes. A secure key is inserted to each client's trampoline to ensure it does not call unauthorised methods in a server.

Pros

Microbenchmarks and macrobenchmarks.
Measures indirect costs of context switching.

Cons

What is the CR3 register? Explain.
Mixing percentages and multiplier in the last paragraph of abstract.
What do the numbers in Table 1 mean? No unit in table nor explanation.
Does not explain in detail why the indirect costs arise. What exactly gets flushed?

Criticism

Overall, the paper seems to give a convincing argument that IPC without kernel invocation is a possible paradigm, and that it gives noticeable performance benefits. However, there are some flaws, and a major flaw in the macrobenchmarks section.

It is unclear what the costs of protection around the VMFUNC instruction is. For example, it may be possible to observe microarchitectural side effects from the previous process if the only isolation between them is a lazy page table switch. The paper has no analysis of how much of the direct context switching costs will be replicated if similar protection measures are also replicated. There is also no security analysis on this topic.

There is no analysis of worst-case performance or exploration of possible performance limitations and degenerate cases.

For a given client, the dependencies of all servers that it wishes to call are also installed into its possible list of EPTs. It seems possible that for any complex system, this list will grow to encompass every service on a system. However, it is only mentioned in the conclusion that dynamic replacement of EPT entries will be explored in future work. It would have been nice to know the performance cost of such maintenance (e.g. stale trampolines) at runtime. Due to recursive nesting and branching, it seems that a large degree of unpredictability may be possible.

The “calling-key” mechanism used to validate the integrity of IPC introduces a covert key that does not correspond with the capability for invocation (i.e. the key is only a pseudo-guarantee and can be brute-forced.) This is a problem eliminated with seL4 with the removal of threads as IPC destinations and the reply capability, however their approach re-introduces this problem.

A cursory search of VMFUNC reveals this paper. “Exit-Less Isolated Execution” Omote et al. APSys 2015. This paper mentions VMFUNC with a single address space for the purposes of a fast context switch as mentioned in the paper, however it is not referenced.

The paper’s approach of bypassing kernel invocation for IPC compromises other aspects of microkernel design, such as making kernel-guaranteed temporal isolation and time protection impossible to implement. The paper makes no mention of these compromises.

The appeal of ease of integration into existing microkernels is partially dented by the large code base and corresponding increase in TCB.

For cross-core IPC, the paper cites ridiculous numbers (e.g. 16x, 20x, 46x,) however SkyBridge is similar in concept to a passive server and thus runs in the context of the local core.

Their benchmarks make cross-core IPC appear like worst-case scenarios. They should have clarified that they are part of the analysis portion of the study, and shouldn’t be compared directly with their proposal.

Their description of multicore execution and performance in Section 6.5 is improper. IPI should have been excluded from their configuration and benchmark considering seL4’s IPC is a synchronous invocation that works exactly like SkyBridge. SkyBridge is not capable of IPC over IPI. For their ST-Server configuration, they impose a CPU-pinning topology that necessitates cross-core IPC. This is not representative of microkernel IPC performance and should be discarded.

The authors used seL4’s fastpath for their microbenchmark, but do not state that the fastpath was used for their macrobenchmarks.

Incredibly, a benchmark with cross-core IPCs over 3 cores performs similarly to a benchmark with supposedly no IPIs for the 1-thread case. This should have raised a red flag to the authors. By their own admittance, IPIs are expensive. From their microbenchmark, SkyBridge's cycle improvement is around 600 cycles. A cross-core seL4 IPC incurs around a 6000 cycle penalty. I find it difficult to believe that such copious IPIs incur only a 0.004% penalty while their improved IPC mechanism gives an 82% speedup. The authors do not mention possible reasons for this discrepancy.