Sara Salahi
CS 443 – Paper Summary

Title: *Mondrix: Memory Isolation for Linux using Mondriaan Memory Protection*
Authors: Emmett Witchel, Junghwan Rhee and Krste Asanović

Summary: The authors demonstrate the benefits of a fine-grained memory protection scheme (Mondriaan Memory Protection, MMP) by designing and evaluating Modrix, a version of the Linux 2.4.19 kernel enhanced with MMP. Fine-grained memory protection is used to isolate kernel modules in their own protection domains, and the beneficial effects on software robustness are evaluated. The authors particularly focus on the benefit of a fast, hardware-supported protection-domain switch.

Key Ideas

Mondrix achieves memory protection – preventing one process from corrupting the memory of another process – by employing a combination of hardware and software protection mechanisms. It overlays address space with multiple disjoint protection domains, each with a unique set of permissions. Every thread is associated with one protection domain (PD). Each kernel module has its own domain, the basic domains being PD0 – memory supervisor (bottom), PD1 – memory supervisor (top), PD2 – kernel and PD3 – string functions. Domain creation occurs when modules are loaded into the kernel, after which the kernel calls the memory supervisor to set memory permissions. The memory supervisor is partitioned into bottom and top. The bottom just writes the permissions table into memory while the top presents a hardware-independent memory protection interface to the rest of the kernel, enforcing memory protection policies, tracking memory sharing and implementing group protection domains. A Protection Lookaside Buffer (PLB) is used as a cache in MMP hardware, similar to a TLB.

Mondrix allows the processor to switch domains through cross-domain calling. Thread enters a callee's domain at specified points called Switch Gates and returns from a cross-domain call at specified points called Return Gates. Information about Gates is stored in a Gate Table which is cached with a GLB.

Registers designate stack frames in the current domain as readable or writable, with earlier frames designated as read-only. Stack write permissions table is used to decide whether a given stack's address is writable by the thread – also cached in the PLB.

Flaws

Given the very small number of PDs, the PLB working set could be smaller than PLB capacity, which means there would be no reason to expect PLB misses at all. However, the authors do not state the size of the PLB so independent confirmation of results such as "all benchmarks spend less than 4% of their execution time refilling the PLB" is impossible. There is also no evaluation of the scalability of MMP to larger numbers of PDs. A benchmark that uses a larger number of user processes would have been more representative of real-life use scenarios. This probably would have significantly increased cache contention and would likely increase Mondrix overheads as well.

The authors don't fully analyze the benefits of MMP. Fast protection-domain switching seems beneficial for enabling the use of smaller protection units inside the kernel, but the authors don't compare MMP's performance with Nooks or another existing architecture that supports fast context switches. This could have been done to establish support for fine-grained protection domains such as MMP. A more thorough cost-benefit analysis is essential to convince me that MMP is worth the architectural changes that it would require.

Relevance and Future Work

Crashes and security breaches can be reduced in severity or avoided if a fault in a single software module was caught before it propagated throughout the system. This is the primary motivation for memory isolation, which forbids one software module from reading or writing another module's memory without permissions – an essential component of a robust system.

1) Testing MMP with a larger number of user processes and 2) comparing its performance with other platforms are the easiest first two steps one could take in evaluating MMP's future potential.