

Mondrix

Memory Isolation for Linux using Mondrian Memory Protection

Emmett Witchel, Junghan Rhee, Krste Asanović
SOSP'05, Oct 2005, Brighton, UK.

Goals and Motivation

- Modules usually share address space, but can be unreliable.
- Provide fine-grained memory protection to limit modules' damage
- Hardware extension with backward compatibility

Alternative solutions are all too slow:

- Safe languages, like Java and C#
- Per-module address space (more threads)

Authors' previous work

Publications:

- Witchel and Asanović. Mondrian Memory Protection. In *Architectural Support for for Programming Languages and Operating Systems*, Oct 2002.
- Witchel and Asanović. Hardware works, software doesn't: Enforcing modularity with Mondrian memory protection. In *HotOS-9*, 2003.
- Witchel Ph.D. thesis, MIT 2004.

Protection Domains

On every load, store, and instruction fetch the hardware checks the *permissions table* for access rights on that address.

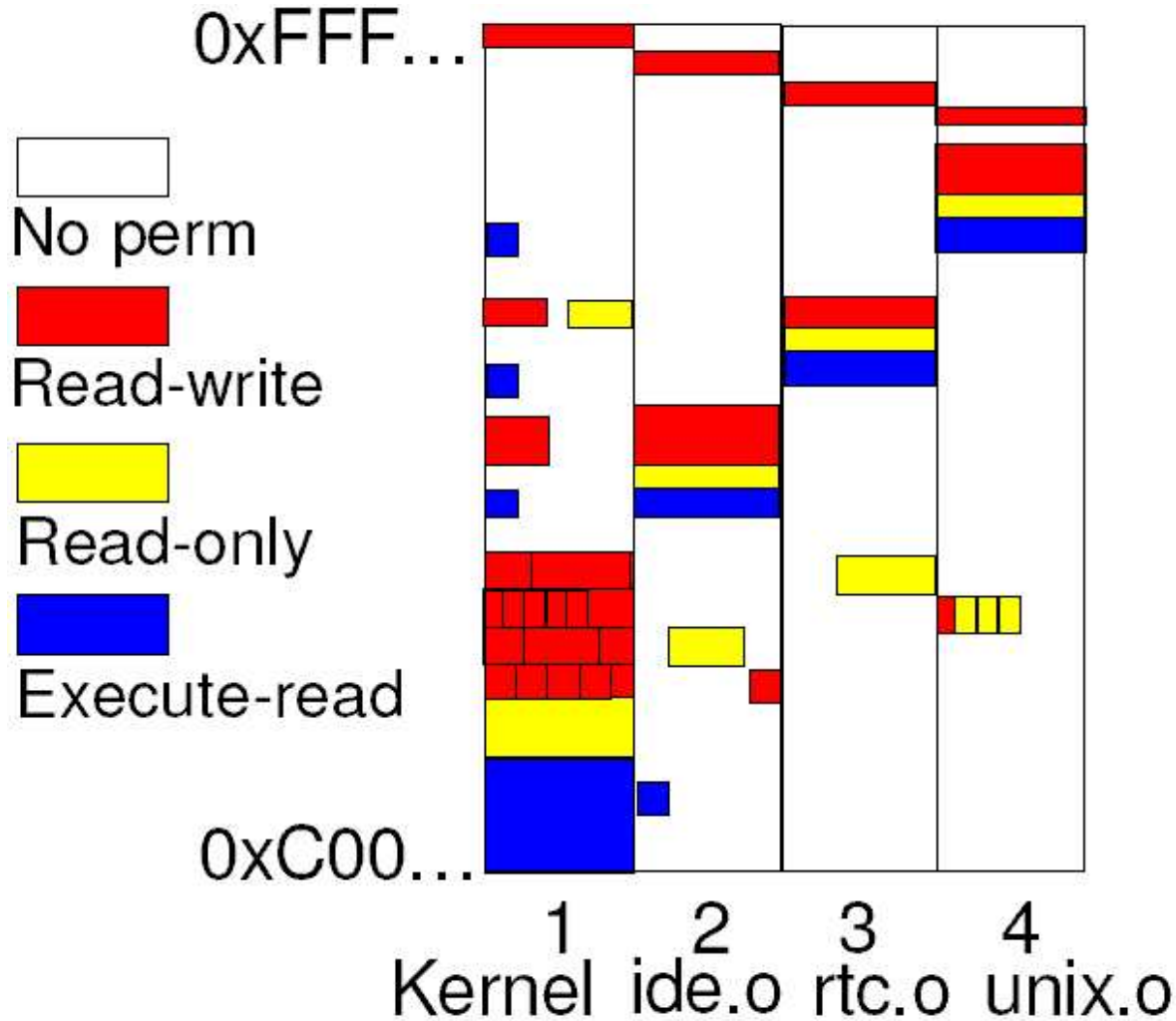
- Each PD owns a chunk of the address space.
- PD-ID zero is the *supervisor domain*
 - initially owns all mem
 - has write access to Permissions Table.
- PD is changed by *switch gates*, usually on function calls.

This paper's contribution

MMP proof-of-concept implementation and evaluation

- Linux extended to use MMP
- Linux kernel is compartmentalized
- Patch holes in MMP spec
- Simulation and performance evaluation in SimICS and Bochs
- Simple security experiments

Isolation of Linux modules



Four protection domains for four modules within the same address space.

The Memory Supervisor:

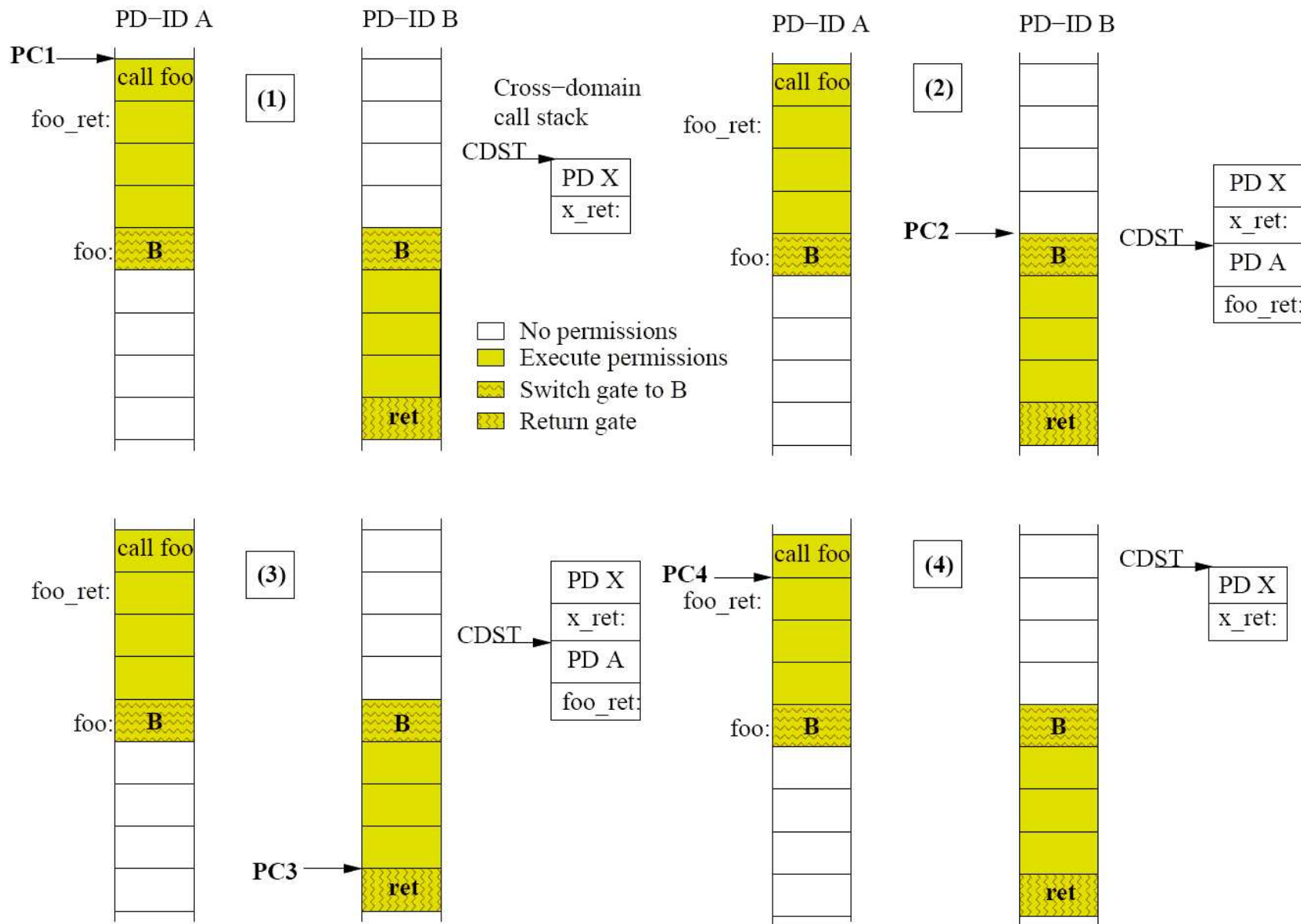
A kernel within a kernel

- Bottom layer writes permission tables in memory
- Top layer is kernel interface and bottom-controlling logic
- Module loading
 - Done via a modified `insmod` command.
 - A new protection domain is created.
 - Switch gates are inserted at function entry and return.

Special cases

- Group protection domains
- Export permissions
- Stack permissions
- Cross-domain calling

Cross-domain calling: example



Evaluation methodology

- Four benchmark apps on SimICS and Bochs simulators.
 - MySQL and `find` – disk intensive
 - `thttpd` – network intensive
 - `config-xemacs` – process intensive
- Measured:
 - percentage of execution time in MMP modules
 - execution time compared to non-MMP system

Experimental results

- Runtime increases up to 15%.
- Main performance hits are on:
 - process creation
 - network buffer control
- In disk-intensive apps, MMP overhead is swallowed in I/O wait time.
- Hardware cost is ignored.

Related Work

- Nooks – software approach to device driver safety
 - Coarse grained
 - OS only, not general purpose
 - large trusted computing base (22k lines code)
- Safe languages
 - Require complete OS rewrite – long timeframe
 - SPIN – efficient but device drivers written in C

Related Work (cont.)

- Hardware approaches
 - x86 NX bit and segments provide coarse protection
 - Multics, circa 1970
- OS structure
 - single-address space w/page-based thread permissions
 - microkernel
 - virtual machine

Problems/Questions

- Do we really want more kernel complexity?
- Who is asking for better mem protection?
- Can an exokernel solve the same problem?
- Why not use different logical address space for different domains (microkernel)?
- How should we deal with memory protection violations at runtime?
- Would it make sense to enable MMP only for testing and debugging?
- Could extra hardware instead be used to speed up context switches?
- Can DMA be protected?