Hardware Works, Software Doesn’t: Enforcing Modularity with Mondriaan Memory Protection

Emmett Witchel
Krste Asanović

MIT Lab for Computer Science
HW Works, SW Doesn’t — Negative

- Hardware has a bozo cousin named Software.
HW Works, SW Doesn’t — Positive

- Hardware cooperates with software. Each has their strengths.

![Hardware](image1.png) ![Software](image2.png)
HW Works, SW Doesn’t — Positive

- Hardware cooperates with software. Each has their strengths.
Software is Growing, Becoming Modular

• Software complexity growing quickly.
  ▪ Faster processors, larger memories allow more complicated software.
  ▪ Linux kernel growing 200,000 lines/yr.

• Debian Linux supports 253 different kernel modules.
  ▪ A module is code + data, possibly loaded at runtime, to provide functionality.

• Modules have narrow interfaces.
  ▪ Not usually as narrow as an API, some internals are exposed.
  ▪ Enforced by programming convention.
Modular Software is Failing

- Big, complex software fails too often.
  - Device drivers are a big problem.
- Big, complex software is hard to maintain.
  - Dependencies are tough to track.
Safe Languages (More SW) Not Answer

• Safe languages are slow and use lots of memory.
  ▪ Restricts implementation to a single language.
  ▪ Ignores a large installed base of code.
  ▪ Can require analysis that is difficult to scale.

• Safe language compiler and run-time system is hard to verify.
  ▪ Especially as more performance is demanded from safe language.

• Doing it all in SW as dumb as doing it all in HW.
Both Hardware and Software Needed

- Modules have narrow, but irregular interfaces.
  - HW should enforce SW convention without getting in the way.

- Module execution is finely interleaved.
  - Protection hardware should be efficient and support a general programming model.

- New hardware is needed to support software to make fast, robust systems.
Current Hardware Broken

• Page based memory protection.
  ▪ A reasonable design point, but we need more.

• Capabilities have problems.
  ▪ Revocation difficult [System/38, M-machine].
  ▪ Tagged pointers complicate machine.
  ▪ Requires new instructions.
  ▪ Different protection values for different domains via shared capability is hard.

• x86 segment facilities are broken capabilities.
  ▪ HW that does not nourish SW.
Mondriaan Memory Protection

• Efficient word-level protection HW.
  ▪ <0.7% space overhead, <0.6% extra memory references for coarse-grained use.
  ▪ <9% space overhead, <8% extra memory references for fine-grained use. [Witchel ASPLOS '02]

• Compatible with conventional ISAs and binaries.
  ▪ HW can change, if it's backwards compatible.
  ▪ Let's put those transistors to good use.

• [Engler '01] studied linux kernel bugs.
  ▪ Page protection can catch 45% (e.g., null).
  ▪ Fine-grained protection could catch 64% (e.g., range checking).
Kernel loader establishes initial permission regions

Kernel calls
mprotect(buf0, RO, 2);
mprotect(buf1, RW, 2);
mprotect(printk, EX, 2);

ide.o calls
mprotect(req_q, RW, 1);
mprotect(mod_init, EX, 1);

Multiple protection domains
How Much Work to Use MMP?

• Do nothing.
  ▪ Your application will still work.

• Change the malloc library (any dynamic lib).
  ▪ You can add electric fences.

• Change the dynamic loader.
  ▪ You can have module isolation.

• Add vmware/dynamo-like runtime system.
  ▪ Many possibilities for fine-grained sharing.

• Change the program source.
  ▪ You can have and control fine-grained sharing.
Trusted Computing Base of MMP

- MMP hardware checks every load, store and instruction fetch.

- MMP memory supervisor (software) writes the permissions tables read by the hardware.
  - Provides additional functionality and semantic guarantees.

MMP TCB smaller than safe language.
Memory Supervisor

- One protection domain (PD) to rule them all.
  - Writes MMP tables for other domains.
  - Handles memory protection faults.
  - Provides basic memory management for domain creation.
  - Enforces some memory use policies.

- Memory supervisor is part of kernel.
  - User/kernel distinction still exists.

<table>
<thead>
<tr>
<th>Kernel Protection Domains (PD-IDs)</th>
<th>MMP Supervisor</th>
<th>Core Kernel</th>
<th>Memory Allocators</th>
<th>Kernel Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2,..,N</td>
<td>N+1,..</td>
</tr>
</tbody>
</table>
Memory Supervisor API

- Create and destroy protection domains.
  - `mmp_alloc_PD(user/kernel);`
  - `mmp_free_PD(recursive);`

- Allocate and free memory.
  - `mmp_alloc(n_bytes);`
  - `mmp_free(ptr);`

- Set permissions on memory (global PD-ID supported).
  - `mmp_set_perm(ptr, len, perm, PD-ID);`

- Control memory ownership.
  - `mmp_mem_chown(ptr, length, PD-ID);`
Managing Data

• **Heap data is owned by PD.**
  - Permissions managed with supervisor API.
  - E.g., `mmp_set_perm(&buf, 256, read-only, consumer_PD-ID);`

• **Code is owned by PD.**
  - Execute permission used within a PD.
  - Call gates are used for cross-domain calls, which cross protection domain boundaries.

• **Stack is difficult to do fast.**
Call and Return Gates

- Procedure entry is call gate, exit is return gate.

- Call gate data stored in permissions table.

- Return gate returns & restores original PD.
Architectural Support for Gates

- Architecture uses protected storage, the cross-domain call stack, to implement gates.

  On call gate execution:
  - Save current PD-ID and return address on cross-domain call stack.
  - Transfer control to PD specified in the gate.

  On return gate execution:
  - Check instruction RA = RA on top of cross-domain call stack, and fault if they are different.
  - Transfer control to RA in PD specified by popping cross-domain call stack.
Are Gate Semantics Useful?

- Returns are paired with calls.
  - Works for callbacks.
  - Works for closures.
  - Works for most implementations of exceptions (not setjmp/longjmp).

- Maybe need a call-only gate.
  - To support continuations and more exception models.
  - Allow cross-domain call stack to be paged out.
Stack Headache

- Threads cross PDs, and multiple threads allowed in one PD.
  - So no single PD can own the stack.

- MMP for stack permissions work, but it is slow.
  - Can copy stack parameters on entry/exit.
  - Can add more hardware to make it efficient.
  - Can exploit stack usage properties.
    - How prevalent are writes to stack parameters?
Finding Modularity in the OS

• Let MMP enforce module boundaries already present in software.

• Defining proper trust relations between modules is a huge task.
  ▪ Not one I want to do by hand.

• Can we get 90% of the benefit from 5% of the effort?
Using Symbol Information

• Symbol import/export gives information about trust relations.
  - Module that imports “printk” symbol will need permission to call printk.

• Data imports are trickier than code imports.
  - E.g., code can follow a pointer out of a structure imported via symbol name.
  - Do array names name the array or just one entry?
Measuring OS Modularity

• Is module interface narrow?
  ▪ Yes, according to symbol information.
  ▪ Measured the static data dependence between modules and the kernel.

• How often are module boundaries crossed?
  ▪ Often, at least in the boot.
  ▪ Measured dynamic calling pattern.
• Modules are small and mostly code.
• 4,031 named entry points in kernel.
• Kernel has 551KB of static data.
• Block devices import arrays of structures.
Measuring Cross-Domain Calls

- Instrumented bochs simulator to gather data about module interactions in Debian Linux 2.4.19.
  - Enforce module boundaries: deal with module loader, deal with module version strings in text section, etc.

- 284,822 protection domain switches in the billion instruction boot.
  - 3,353 instructions between domain switch.
  - 97.5% switches to IDE disc driver.

- This is fine-grained interleaving.
Additional Applications

- Once you have fine-grained protection, exciting possibilities for system design become possible.

- Eliminate memory copying from syscalls.

- Provide specialized kernel entry points.

- Enable optimistic compiler optimizations.

- Implement C++ const.
• Hardware should help make software more reliable.
  - Without getting in the way of the software programming model.

• MMP enables fast, robust, and extensible software systems.
  - Previously it was pick two out of three.