Mondrix: Memory Isolation for Linux using Mondriaan Memory Protection

Emmett Witchel
Junghwan Rhee
Krste Asanović

University of Texas at Austin
Purdue University
MIT CSAIL
Uniprocessor Performance Not Scaling

- OS can help HW designers keep their job

Graph by Dave Patterson
Lightweight HW Protection Domains

- Divisions within address space
  - Backwards compatible with binaries, OS, ISA
  - Linear addressing – one datum per address
- HW complexity about same as TLB
- Switching protection contexts faster than addressing contexts
  - Protection check off load critical path
  - No pipeline flush on cross-domain call

<table>
<thead>
<tr>
<th>User</th>
<th>thttpd</th>
<th>MySQL</th>
<th>find</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>ide-mod</td>
<td>ide-disk</td>
<td>ne</td>
</tr>
</tbody>
</table>
Problems With Modern Modules

- Modules in a single address space
  - Simple
  - Inter-module calls are fast
  - Data sharing is easy (no marshalling)
- No isolation
  - Bugs lead to bad memory accesses
  - One bad access crashes system

Single Address Space

- Red: Read-write
- Yellow: Read-only
- Blue: Execute
- White: No access
Current Hardware Broken

• Page based memory protection
  ▪ Came with virtual memory, not designed for protection
  ▪ A reasonable design point, but not for safe modules
  ▪ Modules are not clean abstractions

• Hardware capabilities have problems
  ▪ Different programming model
  ▪ Revocation difficult [System/38, M-machine]
  ▪ Tagged pointers complicate machine

• x86 segment facilities are broken capabilities
  ▪ HW that does not nourish SW
Mondriaan + Linux = Mondrix

• Each kernel module in different protection domain to increase memory isolation
  ▪ Failure indicated before data corruption
  ▪ Failures localized, damage bounded

• Mondriaan Memory Protection (MMP) makes legacy software memory safe
  ▪ Verify HW design by building software (OS)

• ASPLOS ’02, the MMP permission table
  ▪ Nine months

• SOSP ’05, Linux support + MMP redesign
  ▪ Two years
Mondrix In Action

Memory Addresses

Kernel loader establishes initial permission regions

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

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

No perm
Read-write
Read-only
Execute-read

Multiple protection domains
Challenges for Mondrix

- Memory supervisor
  - Manage permissions, enforce sharing policy

- Memory allocators
  - Keep semantics of \texttt{kfree} even with memory sharing

- Cross-domain calling (lightweight, local RPC)
  - e.g., kernel calls \texttt{start_recv} in network driver

- Group domains
  - Permissions for groups of memory locations whose members change with time

- Device drivers (disk and net)

- Evaluation (safety and performance)
Memory Supervisor

• Kernel subsystem to manage memory permissions (Mtop). Not trust kernel.
  • Exports device independent protection API
    • `mprot_export(ptr, len, prot, domain-ID)`
  • Tracks memory owned by each domain
  • Enforces memory isolation policy
    • Non-owner can not increase permissions
    • Regulates domains joining a group domain

• Writes protection tables (Mbot)
  • All-powerful. Small.
Memory Allocation

• Memory allocators kept out of supervisor
  ▪ Allocator finds block of proper length
  ▪ Supervisor grants permissions

• Supervisor tracks sharing relationships
  ▪ \texttt{kfree} applies to all domains & groups
  ▪ No modifications to kernel to track sharing

• Slab allocator made MMP aware
  ▪ Allows some writes to uninitialized memory
Cross-Domain Calling

- Mondrix guarantees:
  - Module only entered at switch gate
  - Return gate returns to instruction after call, to calling domain
  - “Marshalling” = Giving permissions
  - Stack allocated parameters are OK

- HW writes cross-domain call stack
MMP Hardware

CPU
- Domain ID
- Program Counter
- Gate Lookaside Buffer
- Stack Regs
- Protection Lookaside Buffer (PLB)

Memory
- Gate Table
- Stack Permissions Table
- Permissions Table

Refill
- Only permissions table is large
Group Protection Domains

- Domains need permission on group of related memory objects.

- Group domain virtual until a regular domain joins.

- Supervisor regulates membership
Disk and Network Device Drivers

• Disk driver (EIDE)
  ▪ Permission granted before device read/write
  ▪ Permission revoked after device read/write
  ▪ DMA supported

• Network driver (NE2000)
  ▪ Permissions tightly controlled
  ▪ Read-write to 32 of 144 bytes of `sk_buff`
    • Device driver does not write kernel pointers
  ▪ Device does not support DMA
Net Driver Example

mprot_export(&skb, PROT_RW, sr_pd);
dev->start_recv(skb, dev); // XD
mprot_export(&skb, PROT_NONE, sr_pd)

• Kernel loader modifications
  - start_recv becomes cross-domain call

• Also add module memory sharing policy
  - Permission grant/revoke explicit
Evaluation Methodology

• Turned x86 into x86 with MMP
  ▪ Instrumented SimICS & bochs machine simulator
  ▪ Complete system simulation, including BIOS
  ▪ 4,000 lines of hardware model of MMP

• Turned Linux into Mondrix
  ▪ 4,000 lines of memory supervisor top
  ▪ 1,720 lines of memory supervisor bottom
  ▪ 2,000 lines of kernel changes
    • Modified allocators, tough but only done once
    • Modified disk & network code easier
Fault Injection Experiments

• Ext2 file system, RIO/Nooks fault injector

<table>
<thead>
<tr>
<th>Symptom</th>
<th># runs</th>
<th>MMP catch</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>157</td>
<td>4 (2.5%)</td>
</tr>
<tr>
<td>Hang</td>
<td>23</td>
<td>9 (39%)</td>
</tr>
<tr>
<td>Panic</td>
<td>20</td>
<td>18 (90%)</td>
</tr>
</tbody>
</table>

• Mondrix prevented 3 of 5 cases where filesystem became corrupt (lost data)
  - MMP detected problems before propagation
  - 2 of 3 errors detected outside device driver
Workloads

• ./configure for xemacs-21.4.14
  ▪ Launches many processes, creates many temporary files

• thttpd
  ▪ Web server with cgi scripts

• find /usr –print | xargs grep kangaroo

• MySQL – client test subset 150 test transactions
Performance Model

• 1 instruction per cycle
• 16KB 4-way L1 I & D cache
• 2MB 8-way associative unified L2 cache
• 4 GHz processor, 50ns memory
• L1 miss = 16 cycles, L2 miss 200 cycles
• Slowdown = Total time of Mondrix workload/Total time of Linux workload
config-xemacs

Percentage of execution time

Time (cycles)

Kernel (26%) User (74%)

Other (4.4%)

sed (16%)

configure (0.62%)

as (3.4%)

ld (22%)

ccl (27%)

border between K/U

Other (18%)

mmp_bot (2.4%)

mmp_top (0.55%)

sys_call (5.2%)

disk (0.25%)
## Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Slow</th>
<th>Cyc*10^9</th>
<th>Mbot</th>
<th>Mtop</th>
<th>Kern</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf-xemacs</td>
<td>4.4%</td>
<td>16.5</td>
<td>2.4%</td>
<td>0.7%</td>
<td>1.3%</td>
</tr>
<tr>
<td>thttpd</td>
<td>14.8%</td>
<td>0.23</td>
<td>9.3%</td>
<td>2.0%</td>
<td>3.7%</td>
</tr>
<tr>
<td>find</td>
<td>3.3%</td>
<td>14.3</td>
<td>1.3%</td>
<td>1.2%</td>
<td>0.8%</td>
</tr>
<tr>
<td>MySQL</td>
<td>9.6%</td>
<td>0.21</td>
<td>4.0%</td>
<td>3.3%</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mem</th>
<th>XD</th>
<th>Cy/XD</th>
<th>PLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf-xemacs</td>
<td>10.2%</td>
<td>0.3%</td>
<td>1,286</td>
<td>0.8%</td>
</tr>
<tr>
<td>thttpd</td>
<td>1.1%</td>
<td>0.8%</td>
<td>939</td>
<td>3.8%</td>
</tr>
<tr>
<td>find</td>
<td>7.8%</td>
<td>0.2%</td>
<td>846</td>
<td>0.4%</td>
</tr>
<tr>
<td>MySQL</td>
<td>1.6%</td>
<td>0.7%</td>
<td>664</td>
<td>1.7%</td>
</tr>
</tbody>
</table>
Performance, Protection, Programming

• Incremental performance cost for incremental isolation

• Loader only (~0.1%)
  - Gates, inaccessible words between strings

• Memory allocation package (~1.0%)
  - Guard words
  - Fault on accessing uninitialized data

• Module-specific policies (~10%)
Related Work

• Safe device drivers with Nooks [Swift ’04]

• Asbestos [Efstathopoulos ’05] event processes
  ▪ Isolating user state perfect task for MMP

• Failure oblivious software [Rinard ’04]
  ▪ MMP optimizes out some memory checks

• Useful to implement safe languages?
  ▪ Unmanaged pieces/unsafe extensions
  ▪ Reduce trusted computing base
Conclusion

• Mondrix demonstrates that legacy software can be made safe (efficiently)
• MMP enables fast, robust, and extensible software systems
  - Previously it was pick two out of three
• OS should demand more of HW

Thanks to the PC, and I hope SOSP ’07 accepts ~20%