Mondrix: Memory Isolation for Linux using Mondriaan Memory Protection

Emmett Witchel   Junghwan Rhee   Krste Asanovic

Sreeram Ramalingam
10/26/2005

Roadmap

- Introduction
- Mondria(a)n Memory Protection – Overview
- Kernel Partitioning
- MMP Structure and Features
- Memory Supervisor
- Experimental Evaluation
- Conclusion and Comments

Introduction

- Motivation
- Memory Protection
  - prevents one process from corrupting the memory of another process
  - involves hardware and software
- Methods:
  - Segmentation
  - Paging
  - Protection Keys
- Fine-grained vs. Coarse-grained

Mondria Memory Protection

- Pieter Cornelis Mondriaan, Jr. (1872 – 1944) – the first modern painter
  - Each column represents a protection domain (PD)
  - Each row represents a range of memory addresses
  - Every thread associated with only one PD
  - Allows arbitrary sized memory regions

Linux Kernel Partitioning

- Mondrix isolates each kernel module in its own domain
- Each module in Disk and Network Device drivers has its own domain
- Basic Domains in Mondrix
  - PD 0 – Memory Supervisor (Bottom)
  - PD 1 – Memory Supervisor (Top)
  - PD 2 – Kernel
  - PD 3 – String Functions
- Domain creation occurs when modules are loaded in the kernel

Module Loading / Domain Creation

- insmod program is called by user to load a new kernel module
- Kernel then calls memory supervisor to set memory permissions
  - Length of program sections
  - Start address of every function
  - Address of the return instruction
Domain Creation

PD1 owns the entire address space.
PD2 owns its static code and data.
PD1 owns the rest of the address space.

Device Drivers Partitioning

- **Disk Driver**
  - Device dependent bottom half
  - Device independent top half
  - Mondrix detects improper programming of device registers

- **Network Driver**
  - Chip-specific portion (coordinates reception and transmission)
  - Board-specific portion (moves data on and off n/w card)

Other Domain Partitions

- **Device Interrupts**
  - Jumps to interrupt stubs marked executable in the global group protection domain

- **Inlined Functions**
  - Export permissions on data
  - Uninline the function

- **Slab Allocator**

Memory Supervisor

- Split into two layers – ‘top’ and ‘bottom’
- Bottom layer’s functionality is to just write the permissions table in memory
- Top layer functionalities:
  - Permissions and Memory allocation (uses API calls perm_alloc and perm_free)
  - Thread-local stack permissions
  - Permissions policy
  - Group Protection domains
    - Used in management of inodes

Permissions Policy

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Call</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

MMP System Structure
MMP Features in Mondrix(1)

- **Memory Protection**
  - Address space is the kernel virtual address space
  - Permissions information stored in permissions table in main memory
  - Protection Lookaside Buffer (PLB) is used as a cache by the MMP hardware and is similar to the TLB
  - Preserves the user/kernel mode distinction (uses high bit of PD-ID register)

MMP Features (2)

- **Cross Domain Calling**
  - Provides two way guarantee
  - Thread enters a callee’s domain at specified points called Switch Gates.
  - Thread returns from a cross-domain call at specified points called Return Gates.
  - Information about Gates are stored in a separate Gate Table and is cached with a Gate Lookaside Buffer (GLB)

Format of Gate Table Entries

- Address (32b)
- Switch/Return (1b)
- Accessed (16b)
- Resolution PD-ID (16b)

Experimental Evaluations

- **Functional Evaluation**
  - Expose a Linux Error!
  - Fault Injection experiments (used in Nooks)

- **Performance Evaluation**
  - Used SimICS and Bochs system emulators
  - CPU and Memory overheads were less than 15%
  - Four benchmarks were chosen
    - config-xemacs
    - httpd
    - find
    - MySQL

Linux Error!

- `free_task_struct()` used to free the task structure if the task structure reference count is zero
- `proc_pid_lookup()` and `proc_pid_delete_inode()` call `free_task_struct()`
- During kernel initialization task structure count is zero causing kernel stack memory to be freed

Fault Injection Experiments

- **Fault Types**
  - Bit flips
  - Low-level software faults
  - High-level software faults

- **Corruption Detection**
  - Indirect Corruption
  - Direct Corruption
    - Checksum
    - memTest
    - File copies

Source: The Rio File Cache: Surviving Operating System Crashes

Performance Evaluation - `httpd`

- Percentage of execution time
- Time (cycles)
- `httpd`
- `kuser` (3.1%)
- `other` (4.6%)
- `httpd` (4.2%)
- `httpd` (13.5%)
- `httpd` (14.4%)
- `httpd` (14.6%)
- `httpd` (14.8%)
- `httpd` (15.2%)
Performance Evaluation

- Cross Domain Calling
  - Cross domain calls accounted for less than 1% of total execution time
  - Cross domain call stack never grew deeper than 64 entries
- Memory Use
  - Sum of Active and Inactive memory in kernel was within 1% for Mondrix and Linux
- PLB Refill Traffic
  - Less than 4% execution time spent in PLB refilling

Related Work

- Nooks uses conventional hardware to isolate modules in different addressing contexts.
  - Language-based protection
    - Use safe languages for OS implementation
  - Hardware-based protection
    - Use of NX bit
- OS structure-based

Conclusion

- Provides Fine-grained memory protection
- Backward compatibility for operating systems, ISAs and programming models
- Additional hardware not on processor’s critical path
- Fits naturally with how modern software is designed and written

MMP Features (3)

- Stack Permissions
  - Registers designate stack frames in the current domain as readable or writable
  - Earlier frames are designated as read only
  - Stack write permissions table is used to decide whether a given stack address is writable by the thread (This is also cached in the PLB)