Virtual Memory, FlashVM & SuperPage

Haibo Chen
Institute of Parallel and Distributed Systems (IPADS)
Shanghai Jiao Tong University
http://ipads.se.sjtu.edu.cn/haibo_chen
Recap: Microkernel

- LPC
- HAL
- Memory Mgmt.
- I/O System
- File Systems
- Device Drivers
- Network Stack

Applications:
- APP1
- APPn
- APP2
- APP3
- APP4

Operating Systems:
- QNX
- MINIX
- L4
Recap: Kernel Structures

Monolithic Kernel based Operating System
- Application
- VFS, System call
- IPC, File System
- Scheduler, Virtual Memory
- Device Drivers, Dispatcher, ...
- Hardware

Microkernel based Operating System
- Application
- UNIX Server
- Device Driver
- File Server
- Basic IPC, Virtual Memory, Scheduling
- Hardware

"Hybrid kernel" based Operating System
- Application
- File Server
- UNIX Server
- Application IPC
- Device Driver
- Basic IPC, Virtual Memory, Scheduling
- Hardware
Recap: Exokernel Example 2

Exokernel + Library OS

Apache

Library OS
chosen from available

Abstractions
Interface

SQL Server

Library OS
customized for SQL Server

Abstractions
Interface

Exokernel

Hardware
Recap: Exokernel Design Challenge

Kernel’s new role
- Tracking ownership of resources
- Ensuring resource protection
- Revoking resource access

Three techniques
- Secure binding
- Visible revocation
- Abort protocol
Outline

A Review of Virtual Memory

FlashVM: revisiting VM for flash
VIRTUAL MEMORY
Byte-Oriented Memory Organization

Programs Refer to Virtual Memory Addresses
- Conceptually very large array of bytes
- Actually implemented with hierarchy of different memory types
- System provides address space private to particular “process”
  - Program being executed
  - Program can clobber its own data, but not that of others

Compiler + Run-Time System Control Allocation
- Where different program objects should be stored
- All allocation within single virtual address space
How does everything fit?
- 32-bit addresses: \(~4,000,000,000\) (4 billion) bytes
- 64-bit addresses: \(~16,000,000,000,000,000,000\) (16 quintillion) bytes

How to decide which memory to use in your program?
- How about after a fork()?

What if another process stores data into your memory?
- How could you debug your program?
A Design Principle

"Any problem in computer science can be solved with another level of indirection."

–David Wheeler in Butler Lampson’s 1992 ACM Turing Award speech
So, we add a level of indirection

One simple trick solves all three problems

– Each process gets its own private image of memory
  • appears to be a full-sized private memory range

– This fixes “how to choose” and “others shouldn’t mess w/your’s”
  • surprisingly, it also fixes “making everything fit”

– Implementation: translate addresses transparently
  • add a mapping function
    – to map private addresses to physical addresses
  • do the mapping on every load or store

This mapping trick is the heart of virtual memory
Virtual Addressing

Address translation

Converting a virtual address to a physical address
Requires close cooperation between the CPU hardware and the operating system

HW: the memory management unit (MMU)
  Dedicated hardware on the CPU chip to translate virtual addresses on the fly

SW: A look-up table
  Stored in main memory
  Contents are managed by the operating system
A System Using Virtual Addressing

Used in all modern servers, desktops, and laptops
One of the great ideas in computer science
Why Virtual Memory?

(1) Allows efficient use of limited main memory (RAM)
   – Use RAM as a cache for the parts of a virtual address space
     • some non-cached parts stored on disk
     • some (unallocated) non-cached parts stored nowhere
   – Keep only active areas of virtual address space in memory
     • transfer data back and forth as needed

(2) Simplifies memory management for programmers
   – Each process gets a full, private linear address space

(3) Isolates address spaces
   – One process can’t interfere with another’s memory
     • because they operate in different address spaces
   – User process cannot access privileged information
     • different sections of address spaces have different permissions
Speeding up Translation with a TLB

“Translation Lookaside Buffer” (TLB)
Small hardware cache in MMU
Maps virtual page numbers to physical page numbers
A TLB hit eliminates a memory access
A TLB miss incurs an additional memory access (PTE)
Fortunately, TLB misses are rare. Why?
Summary: Protected-mode Address Translation

1. CPU
2. Selector
   Offset
3. Logical Address
4. Segment Translation
5. Linear Address
6. Page Translation
7. Physical Address
8. x GB
9. RAM
Summary: Protected Mode Address Translation

- **Logical Address**: Selector + Offset
- **Linear Address**: Dir + Table + Offset
- **Physical Address**: PPN + Offset

- **GDT/LDT**: Base, Limit, Flags
- **Page Table**: PPN, Flags
- **Page Directory**: PPN, Flags
- **CR3**: Page Directory Pointer
FLASHVM: REVISITING VM FOR FLASH
Motivation

There is never enough DRAM
  Price, power and DIMM slots limit amount
  Application memory footprints are ever-increasing

VM is no longer DRAM+Disk
  New memory technologies: Flash, PCM, Memristor…
Flash and Virtual Memory

DRAM is expensive

Disk is slow

Flash is cheap and fast

Flash for Virtual Memory
FlashVM

Flash for Virtual Memory
Does it improve system price/performance?
What OS changes are required?

FlashVM
System architecture using dedicated flash for VM
Extension to core VM subsystem in the Linux kernel
Improved performance, reliability and garbage collection
Background

Flash is **not disk**
- Faster random access performance: 0.1 vs. 2-3 ms
- No in-place modification

Flash blocks **wear out**
- Erasures limited to 10,000-100,000 per block
- **Reliability** dropping with increasing MLC flash density

Flash devices **age**
- Performance drops by up to 85% on some SSDs
- Requires **garbage collection** of free blocks
Virtual Memory

- Reduced DRAM
- Same performance
- Lower system price

- Faster execution
- No additional DRAM
- Similar price

Execution Time $T$

Memory Size $M$

DiskVM

Unused Memory

FlashVM

$\Delta M$

$\Delta T$
Design Goals

Performance

Reliability

Garbage Collection
FlashVM Architecture

- Page Swapping
- VM Memory Manager
- Block Layer
- Disk Scheduler
- Block Device Driver
- FlashVM Manager: Performance, Reliability and Garbage Collection
- Dedicated Flash
  - Cost-effective for VM
  - Reduced FS interference
VM Performance

Challenge
VM systems optimized for disk performance
Slow random reads, high access and seek costs, symmetric read/write performance

FlashVM:
Page write back
Page scanning
Disk scheduling
Page prefetching

Parameter Tuning
Page Prefetching

Linux Prefetching
Minimize costly disk seeks
Delimited by free and bad blocks

FlashVM Prefetching
Exploit fast flash random reads and spatial locality
Seek over free and bad blocks
Stride prefetching

FlashVM uses stride prefetching

- Exploit temporal locality in the reference pattern
- Exploit cheap seeks for fast random access
- Fetch two extra blocks in the stride
The Reliability Problem

Challenge: Reduce the number of writes
   Flash chips lose durability after 10,000-100,000 writes
   Actual write-lifetime can be two orders of magnitude less

FlashVM uses knowledge of page content and state
   Dirty page sampling
   Zero page sharing
Page Sampling

- **free_pages**

- Inactive LRU Page List → s_R → Dirty? → Dirty
  - FlashVM: Prioritize *young clean* over *old dirty* pages
  - Sample: 1-s_R → Disk Flash

- Clean

- Free Page List
Adaptive Sampling

Challenge: reference pattern variations
  Write-mostly: Many dirty pages
  Read-mostly: Many clean pages

FlashVM adapts sampling rate
  Maintain a moving average for the write rate
  Low write rate -> Increase skip probability
    Aggressively skip dirty pages
  High write rate -> Converge to native linux
    Evict dirty pages to relieve memory pressure
Garbage Collection

Flash Cleaning
   All writes to flash go to a new location
   **Discard command** notifies SSD that blocks are unused

Discard Benefits:
   More free blocks for writing
   Avoids copying data for partial overwrites
Discard is Expensive

- Read 4KB: < 0.5 ms
- Write 4KB: 2 ms
- Erase 128KB: 55 ms
- Discard 4KB: 417 ms
Discard and VM

Native Linux VM has limited discard support
  Invokes discard before reusing free page clusters
  Pays high fixed cost for small sets of pages

FlashVM optimizes to reduce discard cost
  Avoid unnecessary discards: dummy discard
  Discard larger sizes to amortize cost: merged discard
Dummy Discard

- **Observation**: overwriting a block
  - Notifies SSD it is empty
  - After discarding it, uses the free space made available by discard

- **FlashVM implements dummy discard**
  - Monitors rate of allocation
  - Virtualize discard by reusing blocks likely to be overwritten soon
Merged Discard

- Native Linux invokes discard once per page cluster
  - Result: 55ms latency for freeing 128K

- FlashVM batch many free pages
  - Defer discard until 100MB of free pages available
  - Page discarded may be non-contiguous
Design Summary

Performance improvements
  Parameter Tuning: page write back, page scanning, disk scheduling
  Improved/stride prefetching

Reliability improvements
  Reduce writes: page sampling and sharing

Garbage collection improvements
  Merged and Dummy discards
Evaluation Methodology

System and Devices
2.5G Intel Core 2 Quad, Linux 2.6.28 kernel
IBM, Intel X-25M, OCS-Vertex trim-capable SSDs

Application Workloads
ImageMagick: resizing a large JPEG image by 500%
Spin: model checking for 10 million states
SpecJBB: 16 concurrent warehouses
Memcached server: key-value store for 1 million keys
Application Performance and Memory Saving

Const Performance
84% memory savings

Const Memory
94% less execution time

Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Runtime</th>
<th>Memory Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageMagick</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpecJBB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>memcached-store</td>
<td></td>
<td></td>
</tr>
<tr>
<td>memcached-lookup</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Write Reduction

- **Uniform Page Sampling**: 7% overhead, 12% reduction
- **ImageMagick**: 14% reduction
- **Adaptive Page Sampling**: 93% reduction
- **Page Sharing**: Minimal reduction

Write Reduction Technique

Performance/Writes
Garbage Collection

- **ImageMagick**
  - Linux/Discard: 10X faster
  - FlashVM: 15% slower
  - Linux/No Discard

- **Spin**
  - Linux/Discard: 15% slower
  - FlashVM: 15% slower
  - Linux/No Discard

- **memcached**
  - Linux/Discard: 15% slower
  - FlashVM: 15% slower
  - Linux/No Discard
Conclusions

FlashVM: Virtual Memory Management on Flash
  Dedicated flash for paging
  Improved performance, reliability and garbage collection

More opportunities and challenges for OS design
  Scaling FlashVM to massive memory capacities
  Future memory technologies: PCM and Memristors