Virtual Virtual Memory

Jason Power

3/20/2015

With contributions from Jayneel Gandhi and Lena Olson
Virtual Machine History

• 1970’s: VMMs
• 1997: Disco
• 1999: VMWare (binary translation)
• 2003: Xen (para-virtualization)
• 2006(ish): Hardware support
VM Origins

1974

Survey of Virtual Machine Research

Robert P. Goldberg
Honeywell Information Systems
and Harvard University
Virtual Machine Monitor (VMM)

VMM also called a hypervisor
Disco

- Trap and emulate
What about x86?

- x86 can’t use trap and emulate

- Classic Example: *popf* instruction
  - Same instruction behaves differently depending on execution mode
  - User Mode: changes ALU flags
  - Kernel Mode: changes ALU and system flags
  - Does not generate a trap in user mode
VMWare

- **Solution:** binary translation
- Only need to translate OS code
  - Makes SPEC run fast by default
- Most instruction sequences don’t change
- Instructions that **do change:**
  - Indirect control flow: call/ret, jmp
  - PC-relative addressing
  -Privileged instructions
Overheads

- Traps are heavy weight
- Binary translation
  - Bad for OS-heavy workloads (many server apps)
- What if you’re allowed to change OS a little?
Paravirtualization and Xen

- Use **hypercalls** to bypass VMM
- Still emulate for corner cases & safety reasons
- Commonly used!
  - Amazon EC2
- Not “full virtualization”
Hardware Support

- Another ring
  - `int` moves from user-mode to kernel-mode
  - `vmrun` moves from kernel-mode to vmm-mode

- Many other instructions

- What about virtual memory?
Let’s recall virtual memory
Virtualize the OS’s memory?

OS 1

OS 2

Hypervisor

2^{34} 0 2^{34} 0

2^{34} 0 2^{34} 0
Two dimensions of translation

Virtual addresses

Guest physical addresses

Hypervisor

Physical addresses

P1

P2

OS 1

OS 2

2^{34}

0
Two *dimensions* of translation

- **Guest Virtual Address (gVA)**
- **Guest Physical Address (gPA)**
- **Host Physical Address (hPA)**

1. **Guest Page Table**
2. **Nested Page Table**
What about the TLB?

- Want to cache virtual → machine in TLB
- (Relatively) Easy with software-loaded TLBs
  - TLB miss is a trap (virtual → guest physical)
  - Guest OS loads TLB (VMM trap)
  - Translates guest physical → machine physical
  - VMM actually does the TLB insert
- Problem?
Hardware walked pagetable

- Page table walker walks nested pagetable
- Need a “fake” page table
Keeping shadow page table coherent introduces overheads.
Hardware support

- Today’s hardware is aware of nested pagetable
- Nested page table walk
  - For each level, must do a full pagetable walk
  - Can be very high overhead
Support for Virtualizing Memory
Tradeoffs

Nested Paging

- Up to 24 memory references
- Updates to either page tables without VMM intervention
- Beneficial with
  - Low TLB miss rate
  - High page table updates

Shadow Paging

- Up to 4 memory references
- Updates to either page tables requires costly VMM intervention
- Beneficial with
  - High TLB miss rate
  - Low page table updates
Cost of virtualization

Overheads (native)

Execution time overhead

0% 20% 40% 60% 80% 100% 1200% 1000% 800% 600% 400% 200% 0%

4K 4K+4K 4K+2M 2M+2M graph500
4K 4K+4K 4K+2M 2M+2M memcached
4K 4K+4K 4K+2M 2M+2M NPB:CG
4K 4K+4K 4K+2M 2M+2M gups
Cost of virtualization

![Graph showing execution time overhead for various configurations and applications.](chart.png)
Cost of virtualization

![Bar chart showing execution time overhead for different workloads: graph500, memcached, NPB:CG, and gups. Overheads (native) and Overheads (virtualized) are compared.](chart.png)
Cost of virtualization

Execution time overhead

Overheads (native)  Overheads (virtualized)

graph500

memcached

NPB:CG

gups
Reducing translation overhead

Bhargava et. al: page walk cache

- Opportunity?
  - PTE reuse (10% of entries cover 90% of accesses)
    - Why?
  - Nested translations are redundant
Reducing translation overhead

Bhargava et. al: page walk cache

- Page walk cache
  - Why not cache L1 entries?

- What is the NTLB?
  - Caches guest physical to system physical
  - Skips the 2nd dimension walk
That was fun, let’s make it more complicated...

DEVICES AND VIRTUAL MEMORY
No IOMMU: No Virtualization

Diagram showing the separation between the Proc, Kernel, Physical, and Device layers.
No IOMMU (virtualized)
Virtualization

- Devices accessed by physical addresses
  - Emulation of IO devices is too expensive!

- Approach 1: VMM driver (paravirtualization)
  - Protection domains: IOMMU checks permissions for the memory location; use physical address
  - Need to rewrite drivers!

- Approach 2: Guest driver (true virtualization)
  - Direct Assignment: driver uses guest physical address
  - IOMMU translates to machine physical address
IOMMU Overview

Want

Isolation and Safety

Memory

IOMMU

Device table lookup

Address translation service

Interrupt remapper

IOTLB

Read memory

VM

proc

CPU

proc

proc

VM

GPU

I/O Device
History

- Initially combination of
  - GART (graphics aperture remapping table) and
  - DEV (device exclusion vector)
- GART
  - Physical-to-physical translation so graphics addresses appear contiguous
  - IOMMU is a generalization
- DEV
  - Devices classified into domains
  - Each domain is allowed to access a set of physical addresses
Laundry list of features

- I/O page tables for I/O devices to access memory
  - permission checking
  - virtual address translation
- Interrupt remapping for I/O interrupts
- Service page faults from I/O devices
- Legacy I/O
- User mode device access
- VM guest device access
- Virtualized user mode device access
- Two-level address translation
- Interrupt virtualization
- ...

IOMMU data structures
I/O page tables

- **CPU register CR3**
- **6th level page table** (used only by IOMMU; only table entries 0 and 127 are valid)
- **5th level page table** (used only by IOMMU)
- **4th level page tables** (CPU and IOMMU are separate)
- **3rd level page tables** (shared by CPU and IOMMU)

Shared page tables for “non-negative” virtual addresses

Shared page tables for “negative” virtual addresses
Page faults (before)

- Generated if the I/O device accesses unallowed memory
- Fatal error
- Written to a log
- Requires pinned memory
Page faults (now)

- Generated if the I/O device accesses unallowed memory
- Written to a buffer
- Interrupt raised on CPU core
  - (Kernel) driver handles the fault
- No support to notify the device it should retry
  - Device keeps on executing/waiting for the TLB miss