Virtualization services

Virtual machines
references

- Intel Virtualization technology
  - IEEE xplorer, May 2005
- Comparison of software and hardware techniques for x86 virtualization
  - ASPLOS 2006
- Memory resource management in VMware ESX server
  - OSDI 2002
- Virtualizing I/O devices on Vmware Workstation
  - USENIX 2002
- Xen and the art of virtualization
  - SOSP 2003
Topics

1. What is a VM?
2. Process vs System VMs
3. Virtualizing the Processor
4. Virtualizing Memory
5. Virtualizing I/O
6. VM Performance Issues
7. Intel VT-x Technology
8. Paravirtualization
Virtualization

- We have seen Vdisk, VIP, Vnode
  - An abstraction mapped to real resources
- Traditionally, a single OS virtualizes system resources among processes
  - Process Virtualization
- What if we ran multiple OSes on the same physical hardware?
  - System Virtualization
- Virtual Machine Monitor: A new layer of software (VMM) multiplexes the OSes
System Models

- Processes
  - Kernel
  - Hardware

- Programming Interface

- Virtual Machines (VMs): VM1, VM2, VM3
  - Kernel
  - Virtual Machine Implementation
  - Hardware
Process Virtualization

- Multiprogramming
  - Multiple programs resident in memory
  - Same ISA (Instruction Set Architecture)
- High level language (HLL) VM
  - Different ISA
  - Emulators
    - Interpreter or Dynamic BT
  - Java VM
System Virtualization

- Support multiple guest operating systems simultaneously
- Why?
Applications on older versions of the OS does not affect applications on a new OS.
Workload consolidation

- Instead of separate, maybe underutilized, vertical stacks (H/W, OS, APP) several OSes on the same H/W

![Diagram showing hardware and software components]

- Mail server: 20%
- Web server: 40%
- Db server: 20%
Workload migration

- A guest OS can migrate to new H/W running a VMM
Virtualization: Where is it going really!?
Mobile Virtualization: WHAT?

Mobile Platform Virtualization

Blackberry OS and Apple OS on the same smart phone!!!
System Virtualization

- Support multiple guest operating systems simultaneously on the same H/W
- Why is it hard?
Even dilbert knows
System virtualization challenges

- VMM is the new kernel
- Just run the OS as a user application
- But OS accesses privileged state?
- Trap into VMM
- Some instructions behave differently in user mode
- Kernel was written assuming it runs in privilege 0
- Guest OS must feel it is running on a real machine
Not every x86 instruction traps uniformly

- Canonical 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

- Could use additional rings
- Ring 0, VMM, ring 1 or 2 for guest OS, ring 3 for app
- Today only two rings used (0 for kernel, 3 for user mode)
System virtualization challenges

- VMM is the new kernel
- How to virtualize memory?
- Page tables within each OS.
- Traditionally OS context switches Processes
- PTE entries point to distinct physical frame numbers (unless shared)
- VMM needs to context switch entire VM
- Must support all features of memory management
  - Segments, pages, combined etc
- Allocation, replacement?
System virtualization challenges

- VMM is the new kernel
- How to virtualize I/O?
- OS schedules I/O and CPU tasks
- Multiplexing and demultiplexing I/O data
- Need to support real I/O devices
- In-bound and out-bound endpoints
- What about device drivers?
- Think about virtualizing nintendo or wii players
System VM models

Native VMM Type I
- Guest Apps
- Guest OS
- VMM
- Hardware
  - Vmware

Hosted VMM Type II
- Guest Apps
- Guest OS
- VMM
- Host OS
- Hardware
  - Vmware

Para Virtualization
- Guest Apps
- Modified Guest OS
- VMM
- Hardware
  - XEN
Role of VMM

- Virtual Machine Monitor (VMM)
- A new layer of software
  - Provides illusion of multiple isolated machines.
  - Arbitrate access to hardware resources for multiple guest OSes.
  - Layer between hardware and guest OS
- VMM tasks
  - Manage state (privileged)
  - Manage resources
Characteristics of VMM

- Popek and Goldberg criteria
- Fidelity: software on the VMM executes identically to its execution on hardware, barring timing effects
- Performance: An overwhelming majority of guest instructions are executed by the h/w without the intervention of the VMM
- Safety: VMM manages all h/w resources
CPU scheduling

- VMM schedules each guest OS
- Round robin or weighted round robin
- Need to keep state of each guest OS when timer interrupts
- Meta level scheduler
- VMM schedules guest OS
- Guest OS schedules processes
- Issues?
VMM Execution cycle

1. Meta Timer Interrupt in running VM.
2. Context switch to VMM.
3. VMM saves state of running VM.
4. VMM determines next VM to execute.
5. VMM sets meta timer interrupt.
6. VMM restores state of next VM.
7. VMM sets PC to timer interrupt handler of next VM.
8. Next VM active.
Resource Virtualization

1. Processor
2. Memory
3. I/O
Virtualizing Processor

- Guest OS should not access privilege state
- Any trap should be handled by the trap handler of the VMM
- VMM should not execute the trap but just handle

<table>
<thead>
<tr>
<th>Process (user mode) trap</th>
<th>Operating System (user mode)</th>
<th>VMM (kernel mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN: OS trap handler</td>
<td>decode trap and execute</td>
<td>trap from guest OS</td>
</tr>
<tr>
<td></td>
<td>appropriate syscall routine</td>
<td>JMP to guest OS</td>
</tr>
<tr>
<td></td>
<td>RTT from Trap</td>
<td>OS trap handler</td>
</tr>
</tbody>
</table>

intercept return from trap
do a real return from trap to user process

start running again
Visibility of privileged state
  - Can read current privilege level
  - Cpl read on %cs low bits indicate PL

Sensitive instructions

Certain instructions behave differently based on the CPL

This disrupts x86 binary distribution
Para virtualization

For each critical instruction in the ISA (guest OS) replace with modified critical instruction

Define a replacement sequence

E.g., POPF{} to POPF’{}

Rebuild OS to use POPF’ and VMM will support POPF’

Need access to source code
CPU virtualization

- Interpreter
- Run OS code inside a binary translator/Interpreter
- Every instruction designed to behave appropriately
- Too much overhead
- Does not satisfy P&G criterion
CPU virtualization

- Binary translation
- Translate critical instructions at runtime
- Code is translated only when it is about to execute
- Replace critical instructions with acceptable instructions
- Code cache to improve performance
- Lots of details
CPU virtualization

- H/W support; new processors support virtualization
- Intel and AMD VT-x architecture
- VT-x has a new operating mode; enabled with VMXON/VMXOFF
- Provides two new forms of operation: root operation (fully privileged, intended for VMM) and non-root operation (not fully privileged, intended for guest OS)
- guest OS runs at ring 0; apps running on guest OS run at ring 3 and VMM runs in new higher mode (VMX) -- equivalently -1 privilege
Virtual memory

- Each Process has its own page table that maps to physical frame that the VM is running in
- Guest OS should not map any Virtual page to any physical page
- No isolation
- On the other hand need PTE to translate virtual addresses to physical addresses
- More than one PTBR (one for each VM)
Shadow Page Tables

Guest OS maintains its own page tables.
  - Virtual to real memory mapping.

VMM maintains shadow page tables
  - Virtual to physical memory mapping.
  - Used by hardware to translate virtual addresses.
  - VMM validates guest page table updates.
  - Replicates guest changes in shadow page table.

Virtualize page table pointer register.
  - VMM manages real page table pointer.
  - Updates page table ptr when switching VMs.
Shadow page table

Process → Shadow page table (read only) → Real page table

Trap to VMM → Write real PTE

VMM

Physical memory
Shadow Page Tables

- Guest reads
- Guest writes
- Accessed & dirty bits
- Updates

Guest Page Table
Guest OS

- Updates

Shadow Page Table
VMM

- MMU

Hardware
Reclaiming memory

- In non-virtualized system, OS uses page replacement algorithms to decide which page to evict
- What should the VMM do?
  - Swap an entire VM
  - Swap pages belonging to individual VMs
  - VMM needs to decide the page as well as the page guest Oss page
- What policy to use?
- What if it conflicts with guest OS page replacement policy
Double paging

- Swap lists, free lists are maintained by each VM also
- VMM pages out, the guest OS selects the same page
- VMM needs to bring back the page into memory and write back into virtual paging device
- So the guest OS needs to decide what pages to swap
- We need to induce the guest OS to start swapping
Ballooning

- Create a balloon device (a driver) inside each guest OS
- Driver can ask for physical pages
- VMM inflates balloon when in need of pages
- VMM deflates balloon when it does not need

Physical memory

Linux

Windows XP

Free BSD
Ballooning

- VMM instructs balloon to inflate
- Driver can ask for physical pages
- Causes memory pressure on guest OS
- Guest OS kicks in page swap process
- Freed pages can be allocated to another guest OS

Linux  Windows XP  Free BSD

Physical memory

SWAP
Ballooning limitations

- Driver may be uninstalled, disabled
- Upper bounds on balloon sizes may need to be imposed
- Should not penalize one guest OS over another
Page sharing

- What if many guest OSes are identical copies
- Unlike a guest OS, which knows text segments of processes (on fork, e.g.,), VMM has no idea
- Nice to find shared read-only pages, at least
- Idea: scan physical pages and seek matches
- So identical pages point to the same PFN
- A Copy-on-write bit is set and if modified make a copy and make the pages different
Virtualizing I/O

- All I/O operations are privileged
- VMM must intercept all guest OS I/O operations
- VMM must implement all device drivers
- Many devices are chatty
- Type II VM may be better
- Uses device drivers in the host OS
- Need to switch between VMM world and host world
Hosted vs. Native

World switching is expensive
Virtualizing a network card

- Each Virtual NIC appears as a full fledged PCI controller with its own MAC address.
- All Virtual NICs form a bridge with physical NIC.
- The physical NIC in promiscuous mode picks up all packets and forwards it to all virtual NICs.
- For virtual networks, ethernet interface is not required.
- How does IP networking work?
Virtualizing a Network Card

**Network Packet Send**
- Guest OS
  - OUT to I/O port
- VMM
  - Context switch
- VM Driver
  - Return to VM App
- VM App
  - Syscall
- VM Net Driver
  - Bridge code
- Host Ethernet Driver
  - OUT to I/O port
- Ethernet H/W
  - packet launch

**Network Packet Receive**
- Ethernet H/W
  - Device Interrupt
- Host Ethernet Driver
  - Bridge code
- VM Net Driver
  - return from select()
- VM App
  - memcpy to VM memory
  - ask VMM to raise IRQ
- VMM
  - raise IRQ
- Guest OS
  - IN/OUT to I/O port
- VMM
  - Context switch
- VM Driver
  - Return from IOCTL
- VM App
  - packet receive completion