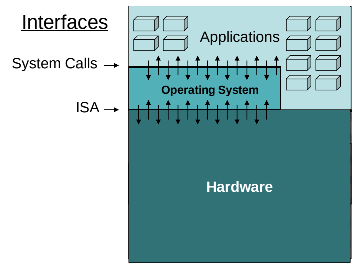
CS 537
Lecture 20
Virtual Machines

Michael Swift

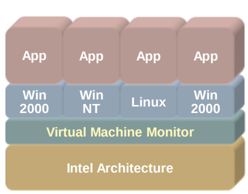
Background Information: Execution Stack



- Lots of applications
- One OS

- Why?
  - Talks to single instance of HW

Virtual Machines



A thin software layer that sits between Intel hardware and the operating system—virtualizing and managing all hardware resources

Virtual Machine Monitors

- A VMM implements the hardware interface in software
  - All instructions that reference privileged processor state refer to a software copy
  - All instructions that refer to specific physical resources (e.g., memory pages) refer to virtual resources selected by the VMM
  - All commands/instructions that refer to specific physical devices refer to software that implements/emulates that device interface
  - All interrupts from physical devices are handled by VMM
  - VMM must be at higher privilege level than guest VM, which generally runs in user mode

Virtual Machine Monitors (VMMs)

- A Virtual Machine is a software version of the hardware state of a computer system
  - An operating system running within a virtual machine is called a guest operating system
- Virtual machine monitor (VMM) or hypervisor is software that implements and supports VMs
  - VMM determines how to map virtual resources to physical ones
  - Physical resource may be time-shared, partitioned, or emulated in software
  - VMM much smaller than a traditional OS;
  - Isolation portion of a VMM is ≈ 10,000 lines of code

Why use VMMs

- Add features hard to do in an OS
  - Suspend/resume: save state to disk and reload
  - Migration: save state to network file system, reload on another machine
- Share hardware
  - Consolidate multiple services from different slow machines
- Security/isolation
  - Share a single web server with multiple customers (e.g. Amazon EC2)
- Run applications for another OS
  - Run Windows apps in a virtual machine on MacOS

Implementation Issues

- Who provides the resource management serves for the VMM?
  - another OS
  - the VMM itself
- What hardware does the VMM expose?
  - The same as a physical machine?
  - Something simpler?
- How are privileged operations performed?

Virtual Machine Types

- Type 1 / Type 2
  - Type 1 VMMs (called Hypervisors) sit just above the HW and virtualize the complete hardware
    - Example: Xen, VMware ESX server
  - Type 2 VMMs run within an OS, and rely on OS services to manage HW
    - Example: QEMU, VMware Worksation

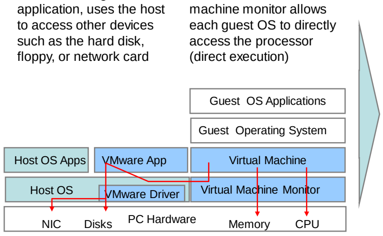
Hosted (Type 2) VMware Architecture

Host Mode

VMware, acting as an application, uses the host to access other devices such as the hard disk, floppy, or network card

VMM Mode

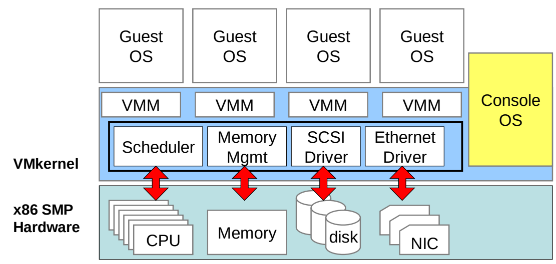
The VMware Virtual machine monitor allows each guest OS to directly access the processor (direct execution)



VMware achieves both near-native execution speed and broad device support by transparently switching* between Host Mode and VMM Mode.

*VMware typically switches modes 1000 times per second

Native (Type 1) Architecture



Comparison

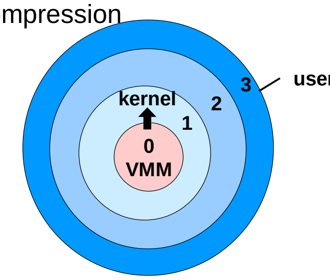
- Type 1 (native)
  - All OS's on the machine more slowly
  - All drivers run in the VMM (VMware) or a special guest OS (Xen)
  - System management is done in a guest OS
- Type 2 (hosted)
  - Host OS runs full speed, guests more slowly
  - All drivers run in host OS, leverage large code base
  - System management is done in host OS

Virtualization through Ring Compression

Virtual Machine Monitor (VMM) runs at ring 0

Kernel(s) run at ring 1

Requires that CPU is virtualizable



Virtualization Technology

- Basic approach: execute privileged software at unprivileged level
  - Privileged instructions will trap: I/O, memmgmt
  - Emulate behavior of privileged instructions in software in VMM
- VMM has complete control over the HW
  - Presents another layer of virtual memory under the OS with a separate page table
  - Presents a different set of devices to the OS
- What happens to instructions that return different results in priv. mode and normal mode?

ISA Impact on Virtual Machines

- Consider x86 PUSHF/POPF instructions
  - Push flags register on stack or pop it back
  - Flags contains condition codes (good to be able to save/restore) but also interrupt enable flag (IF)
- Pushing flags isn't privileged
  - Thus, guest OS can read IF and discover it's not the way it was set
    - VMM isn't invisible any more
- Popping flags in user mode ignores IF
  - VMM now doesn't know what guest wants IF to be
  - Should trap to VMM
- Possible solution: modify code, replacing pushf/popf with special interrupting instructions
  - But now guest can read own code and detect VMM

Classification of processor architectures

- Strictly virtualizable processor architectures
  - Can build a VMM based on trap emulation exclusively
    - No software running inside the VM cannot determine the presence of the VMM (short of timing attacks)
  - Examples: IBM S/390, DEC Compaq Intel Alpha, PowerPC
- (Non-strictly) virtualizable processor architectures
  - Trap emulation alone is not sufficient and/or not complete
    - E.g. instructions have different semantics at various levels (sufficient)
    - E.g Some software sequences can determine the presence of the VMM (complete)
  - Examples: IA-32, IA-64
- Non virtualizable processor architectures
  - Basic component missing (e.g. MMU, …)

Virtualizing x86

- Pure approaches:
  - systems present the interface of real, existing HW and can run unmodified operating systems
  - Binary translation
    - Convert kernel code into a new binary that calls into VMM for all privileged instructions / instructions that do something different between kernel/user mode (VMware)
  - Emulation
    - Emulate all instructions in kernel mode (VirtualPC)
- New hardware
  - Intel VT, AMD Pacifica adds new ring (-1) that traps correctly

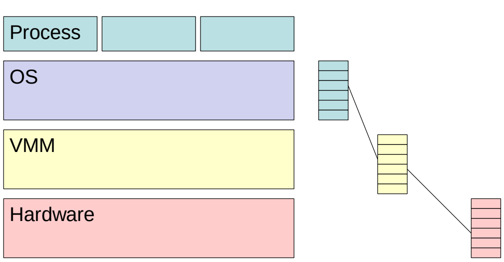
Para-Virtualization

- Para-virtualize side steps the problem
  - present a new, simpler interface but require OS modifications
  - Change kernel code to avoid all privileged instructions
  - Issue explicit **HyperCalls** into VMM to provide these services
- Made possible when:
  - Operating system source is available
    - Open source: Linux and Xen)
    - OS vendor writes VMM: Microsoft Windows/HyperV, Sun Solaris LDOM, IBM AIX/LPAR

Virtualizing Memory

- VMMs present virtual memory to an OS as physical memory
  - Allows the VMM to reclaim pages, swap, give to another VM
- use 3 layer translation: virtual, real, physical
  - OS manages Virtual -> real translation with existing page tables
  - VMM manages real -> physical translation
- How?
  - Trap-on-write to OS page table
  - Shadow page table given to hardware that maps virtual -> physical directly

Virtrual/real/physical memroy



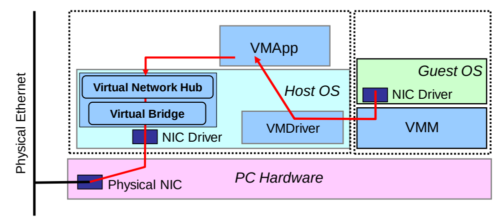
Translating an address

- Given virtual address V:
  1. Lookup V in guest OS page table to find P
  2. Lookup P in VMM page table to find R
  3. use R for memory reference
- Making this fast: Shadow page tables
  1. Create a second page table in VMM containing V -> R mapping, give to hardware
  2. On miss to this table, look at guest OS page table to find P, look at VMM page table to compute R, ad to shadow page table
  3. When guest OS changes PT, remove from shadow
     1. But don't change P->R mapping

Virtualizing Devices

- Virtualization by Emulation
  - Trap on read/write of device registers
  - Emulate device action in VMM
- Virtualization by Replacement
  - Write a new driver for the class of device (e.g., network)
  - Network driver explicitly calls into VMM to perform work

Virtualizing a Network Interface



Virtualizing Disks

- Sharing
  - Networking shared a single device through time multiplexing
  - Disks share through space multiplexing
  - Some device might not be shared, but just assigned to a single VMM, which can run the driver itself
    - USB flash drive
- VMM makes a file in the FS act like a disk to the VMM
  - Can grow incrementally as disk is used
  - Can be copied between systems
- Done by implementing a SCSI or IDE device that talks to the FS