Virtual Machines

CMPU 334 – Operating Systems
Jason Waterman
Layers and Abstraction

Diagram:

- Process
- OS Kernel
- Hardware

Diagram: (Reversed)

- Process
- OS Kernel
- Virtual Machine Monitor
- Hardware
Virtual Machine Monitor (VMM)

• Also called a hypervisor

• Provides a virtualized version of the hardware to each running OS
  • OS thinks it has direct control of the hardware
  • An illusion provided by the VMM

• VMM is really the one in charge
  • Must multiplex the running of multiple OSes across the machine

• Think of the VMM as an Operating System for OSes
Type 1 and Type 2 Hypervisors

Type 1 Hypervisor

- Windows
- Linux

Type 2 Hypervisor

- Guest OS (e.g., Windows)
- Host OS (e.g., Linux)

Control Domain

Hardware
(CPU, disk, network, interrupts, etc.)

Guest OS process

Host OS process
Motivation for VMMs

• Server consolidation
  • Different services with different Oses
  • If these machines are lightly loaded, it makes sense to consolidate to one machine
  • Can suspend, snapshot, and migrate VMs

• Testing and debugging
  • Easily test and deploy on multiple platforms

• Isolation
  • If one VM is compromised or crashes, sandboxing will prevent it from effecting the other VMs
VMM History

• Original concepts date back to the 1970’s
  • IBM used VMs on their mainframe systems

• Did not gain traction for PCs until the 90’s
  • x86 architecture presented challenges to efficient virtualization
    • Current x86 processes have virtualization extensions
  • Research group out of Stanford came out with Disco
    • Went on to found VMware
    • First virtualization solution in 1999
VMM Requirements

- **Safety**: the hypervisor should have full control of the hardware

- **Fidelity**: the behavior of a program on a virtual machine should be identical to that of the same program running on bare metal

- **Efficiency**: much of the code in the virtual machine should run without intervention by the hypervisor
  - An extension of Limited Direct Execution used for processes
Virtualizing the CPU

• When the VVM wishes to “boot” a new OS, jump to the starting address of the OS kernel

• When the VMM wants to switch between OSes, the VMM must perform a machine switch
  • Similar to a context switch
  • Save the state of the OS (PC, registers, and any privileged hardware state)
  • Restore the state of the next OS to run
Trap and Emulate (mechanism)

• What happens when the OS tries to perform a privileged operation?
  • VMM must remain in control of hardware resources, so it can’t allow the OS run those instructions directly
  • E.g., the kernel now runs in User mode instead of Kernel mode

• A privileged instruction by the OS causes a trap into the VMM
  • The VMM can emulate that operation on behalf of the OS
  • VMM retains control but OS still thinks it is in control
Handling an OS syscall

• In a non-virtualized environment
  • On OS boot, kernel installs trap handlers
  • User process calls privileged instruction as part of the syscall
    • Int on x86
    • Hardware transitions to kernel mode and jumps to the trap handler

• In a virtualized environment
  • OS kernel is not able in kernel mode
  • User process can’t trap into the kernel
  • It actually traps into the VMM
    • But VMM doesn’t know how to handle the trap
    • So VMM transfers control to the OS trap handler
    • VMM knows the right handler because OS tried to install the handlers at OS “boot” time
VM Memory Translation

• OS builds its own page table

• Loads the hardware register to point to the top-level page table
  • This is a privileged instruction, so trap into the VMM

• VMM creates a shadow page table to map to the actual physical pages used
  • Must also keep track of any changes to the OS page table
  • Problem: the OS can change a page table just by writing memory

• One possible solution: Mark the page table memory as read-only
  • When the OS tries to modify the table, a page fault will trap into the VM
  • This is expensive, 10s of thousands of cycles
Hardware Support for Nested Page Tables

• To reduce this cost hardware support is needed
  • Nested Page Tables (AMD)
  • EPT -- Extended Page Tables (Intel)
The Information Gap

• Even though the VMM is in charge it knows very little about what the OS is doing or what it needs
  • This is an information gap between the VMM and the OS

• Examples
  • Idle loop
  • Demand zeroing of pages
  • Balloon memory
Paravirtualization

• To this point, we’ve assumed the OS does not know it not running directly on the hardware

• The VMM must go through hoops to maintain this illusion

• What modifications can we make to the OS to improve performance
  • Trap and emulate no longer needed
  • Make an explicit call on the VMM
    • Similar to how a process makes a syscall
I/O Virtualization

- OS will probe the hardware on boot to find out what I/O devices exist
  - Trap into the VMM

- VMM can report back the actual hardware that exists
  - But then OS will install device drivers for that hardware
    - When driver accesses device hardware, will trap into VMM

- Or report back generic canonical hardware
  - Paravirtualized drivers, e.g., virtio
Fun Virtualization Tricks

• Checkpointing
  • Write out state of VM as backup
  • Can boot directly to a running VM from a checkpoint

• Live Migration
  • Move a running system from one host to another
Summary

• Virtualization is here to stay
  • The basis for cloud computing

• VMMs extend the notion of limited direct execution
  • Set up the hardware to enable the VMM to interpose on key events such as traps
  • Control how machine resources are allocated while preserving the illusion that the OS in charge

• Similar to the virtualization that the OS does for processes
  • However, OSes were designed to have a clean interface (e.g., syscall)
  • VMMs interface is the hardware
  • Paravirtualization can make that interface a little cleaner (e.g., KVM)