Lecture 19: Virtual Machine Monitors

Prof. Ryan Huang
• Last lab is out
  - Start early

• Lab 4 overview session
  - Monday 11/18 5pm in Malone G33
So Far...

• We’ve covered the three fundamental concepts in OS
  - Concurrency
  - Virtualization
  - Persistency

A major milestone of the course

• Remaining lectures are slightly advanced (but important) OS topics
Review: What Is An OS

- **OS is software between applications and hardware**
  - Abstracts hardware to make applications portable
  - Makes finite resources (memory, # CPU cores) appear much larger
  - Protects processes and users from one another
What If…

- The process abstraction looked just like hardware?
How Do Process Abstraction & H/W Differ

**Process**
- Non-privileged registers and instructions
- Virtual memory
- Errors and signals
- File systems, directories, files, raw devices

**Hardware**
- All registers and instructions
- Both virtual and physical memory, MMU functions, TLB/page tables,..
- Trap, interrupts
- I/O devices accessed through programmed I/O, DMA, interrupts
**Virtual Machine Monitor**

- **Thin layer of software that virtualizes the hardware**
  -Exports a virtual machine abstraction that looks like the hardware
  -Provides the *illusion* that software has full control over the hardware
    -Run multiple instances of an OS or different OSes simultaneously on the same physical machine
Old Idea from The 1970s

- **IBM VM/370** – A VMM for IBM mainframe
  - Multiplex multiple OS environments on expensive hardware
  - Desirable when few machines around

- **Interest died out in the 1980s and 1990s**
  - Hardware got cheap
  - Compare Windows NT vs. N DOS machines

- **Revived by the Disco [SOSP ’97] work**
  - Led by Mendel Rosenblum, later lead to the foundation of VMware

- **Another important work Xen [SOSP ’03]**
• VMs are used everywhere
  - Popularized by cloud computing
  - Used to solve different problems

• VMMs are a hot topic in industry and academia
  - Industry commitment
    • Software: VMware, Xen,…
    • Hardware: Intel VT, AMD-V
      • If Intel and AMD add it to their chips, you know it’s serious…
  - Academia: lots of related projects and papers
Why Would You Do Such a Crazy Thing?

- **Software compatibility**
  - VMMs can run pretty much all software

- **Resource utilization**
  - Machines today are powerful, want to multiplex their hardware

- **Isolation**
  - Seemingly total data isolation between virtual machines
  - Leverage hardware memory protection mechanisms

- **Encapsulation**
  - Virtual machines are not tied to physical machines
  - Checkpoint/migration

- **Many other cool applications**
  - Debugging, emulation, security, speculation, fault tolerance…
Why Would You Do Such a Crazy Thing?

• **Software compatibility**
  - VMMs can run pretty much all software

• **Resource utilization**
  - Machines today are powerful, want to multiplex their hardware

• **Isolation**
  - Seemingly total data isolation between virtual machines
  - Leverage hardware memory protection mechanisms

• **Encapsulation**
  - Virtual machines are not tied to physical machines
  - Checkpoint/migration

• **Many other cool applications**
  - Debugging, emulation, security, speculation, fault tolerance…
• **Backward compatibility is bane of new Oses**
  - Huge effort require to innovate but not break

• **Security considerations may make it impossible**
  - Choice: Close security hole and break apps or be insecure

• **Example: Windows XP is end of life**
  - Eventually hardware running WinXP will die
  - What to do with legacy WinXP applications?
  - Not all applications will run on later Windows
  - Given the # of WinXP applications, practically any OS change will break something

    ```
    if (OS == WinXP) ...
    ```

• **Solution: Use a VMM to run both WinXP and Win10**
Logical Partitioning of Servers

• Run multiple servers on same box (e.g., Amazon EC2)
  - Modern CPUs more powerful than most services need: e.g., only 10% utilization
  - VMs let you give away less than one machine for running a service
  - Server consolidation: \( N \text{ machines} \rightarrow 1 \text{ real machine} \)
  - Consolidation leads to cost savings (less power, cooling, management, etc.)

• Isolation of environments
  - Printer server doesn’t take down Exchange server
  - Compromise of one VM can’t get at data of others

• Resource management
  - Provide service-level agreements

• Heterogeneous environments
  - Linux, FreeBSD, Windows, etc.
Implementing VMMs - Requirements

• **Fidelity**
  - OSes and applications work the same without modification
    • (although we may modify the OS a bit)

• **Isolation**
  - VMM protects resources and VMs from each other

• **Performance**
  - VMM is another layer of software…and therefore overhead
    • As with OS, want to minimize this overhead
  - VMware (early):
    • CPU-intensive apps: 2-10% overhead
    • I/O-intensive apps: 25-60% overhead (much better today)
VMM Case Study 1: Xen

• Early versions use “paravirtualization”
  - Fancy word for “we have to modify & recompile the OS”
  - Since you’re modifying the OS, make life easy for yourself
  - Create a VMM interface to minimize porting and overhead

• Xen hypervisor (VMM) implements interface
  - VMM runs at privilege, VMs (domains) run unprivileged
  - Trusted OS (Linux) runs in own domain (Domain0)
    • Use Domain0 to manage system, operate devices, etc.

• Most recent version of Xen does not require OS mods
  - Because of Intel/AMD hardware support

• Commercialized via XenSource, but also open source
VMM Case Study 2: VMware

- **VMware workstation uses** **hosted** **model**
  - VMM runs unprivileged, installed on base OS (+ driver)
  - Relies upon base OS for device functionality

- **VMware ESX server uses** **hypervisor** **model**
  - Similar to Xen, but no guest domain/OS

- **VMware uses software virtualization**
  - **Dynamic binary rewriting** translates code executed in VM
  - Most instructions translated identically, e.g., `movl`
  - Rewrite privileged instructions with emulation code (may trap), e.g., `popf`
  - **Think JIT compilation for JVM, but**
    - full binary x86 → IR code → safe subset of x86
  - Incurs overhead, but can be well-tuned (small % hit)
VMware Hosted Architecture

![Hosted Architecture Diagram]

- Application
- Guest Operating System
- Virtualization Layer
- Host Operating System
- x86 Architecture
- CPU
- Memory
- NIC
- Disk
- Hosted Architecture
What Needs to Be Virtualized?

• Exactly what you would expect
  - CPU
  - Events (exceptions and interrupts)
  - Memory
  - I/O devices

• Isn’t this just duplicating OS functionality in a VMM?
  - Yes and no
  - Approaches will be similar to what we do with OSes
    • Simpler in functionality, though (VMM much smaller than OS)
  - But implements a different abstraction
    • Hardware interface vs. OS interface
Approach 1: Complete Machine Simulation

- Simplest VMM approach, used by bochs

- Build a simulation of all the hardware
  - CPU – A loop that fetches each instruction, decodes it, simulates its effect on the machine state
  - Memory – Physical memory is just an array, simulate the MMU on all memory accesses
  - I/O – Simulate I/O devices, programmed I/O, DMA, interrupts

- Problem: Too slow!
  - CPU/Memory – 100x CPU/MMU simulation
  - I/O Device – < 2x slowdown.
  - 100x slowdown makes it not too useful

- Need faster ways of emulating CPU/MMU
Virtualizing the CPU

• Observations: Most instructions are the same regardless of processor privileged level
  - Example: `incl %eax`

• Why not just give instructions to CPU to execute?
  - One issue: Safety – How to get the CPU back? Or stop it from stepping on us? How about `cli/halt`?
  - Solution: Use protection mechanisms already in CPU

• Run virtual machine’s OS directly on CPU in unprivileged user mode
  - “Trap and emulate” approach
  - Most instructions just work
  - Privileged instructions trap into monitor and run simulator on instruction
  - Makes some assumptions about architecture
Virtualizing Traps

• What happens when an interrupt or trap occurs
  - Like normal kernels: we trap into the monitor

• What if the interrupt or trap should go to guest OS?
  - Example: Page fault, illegal instruction, system call, interrupt
  - Re-start the guest OS simulating the trap

• x86 example:
  - Give CPU an IDT that vectors back to VMM
  - Look up trap vector in VM’s “virtual” IDT
    • How does VMM know this?
  - Push virtualized %cs, %eip, %eflags, on stack
  - Switch to virtualized privileged mode
Virtualizing Memory

• OSes assume they have full control over memory
  - Managing it: OS assumes it owns it all
  - Mapping it: OS assumes it can map any virtual page to any physical page

• But VMM partitions memory among VMs
  - VMM needs to assign hardware pages to VMs
  - VMM needs to control mappings for isolation
    • Cannot allow an OS to map a virtual page to any hardware page
    • OS can only map to a hardware page given to it by the VMM

• Hardware-managed TLBs make this difficult
  - When the TLB misses, the hardware automatically walks the page tables in memory
  - As a result, VMM needs to control access by OS to page tables
One Way: Direct Mapping

• VMM uses the page tables that a guest OS creates
  - These page tables are used directly by hardware MMU

• VMM validates all updates to page tables by guest OS
  - OS can read page tables without modification
  - But VMM needs to check all PTE writes to ensure that the virtual-to-physical mapping is valid
    • That the OS “owns” the physical page being used in the PTE
  - Modify OS to hypervisor call into VMM when updating PTEs

• Page tables work the same as before, but OS is constrained to only map to the physical pages it owns

• Works fine if you can modify the OS (used in Xen paravirtualization)

• If you can’t…
Second Approach: Level of Indirection

- Three abstractions of memory
  - Machine: actual hardware memory
    - 16 GB of DRAM
  - Physical: abstraction of hardware memory managed by OS
    - If a VMM allocates 512 MB to a VM, the OS thinks the computer has 512 MB of contiguous physical memory
    - (Underlying machine memory may be discontiguous)
  - Virtual: virtual address spaces you know and love
    - Standard $2^{32}$ or $2^{64}$ address space

- Translation: VM’s Guest VA → VM’s Guest PA → Host PA

- In each VM, OS creates and manages page tables for its virtual address spaces without modification
  - But these page tables are not used by the MMU hardware
Shadow Page Tables

- VMM creates and manages page tables that map virtual pages directly to machine pages
  - These tables are loaded into the MMU on a context switch
  - VMM page tables are the shadow page tables

- VMM needs to keep its $V \to M$ tables consistent with changes made by OS to its $V \to P$ tables
  - VMM maps OS page tables as read-only (i.e., write-protected)
  - When OS writes to page tables, trap to VMM
  - VMM applies write to shadow table and OS table, returns
  - Also known as memory tracing
  - Memory-mapped devices must be protected for both read- and write- protected
Memory Mapping Summary

 physical machine

 Host Virtual Address Host PT Host Physical Address

 Guest Virtual Address

 Guest PT Guest Physical Address VMM map Host Physical Address

 Guest Virtual Address

 Shadow Page Table Host Physical Address

 virtual machine
Shadow Page Table Example
Memory Allocation

- **VMMs tend to have simple hardware memory allocation policies**
  - Static: VM gets 512 MB of hardware memory for life
  - No dynamic adjustment based on load
    - OSes not designed to handle changes in physical memory...
  - No swapping to disk

- **More sophistication: Overcommit with balloon driver**
  - Balloon driver runs inside OS to consume hardware pages
    - Steals from virtual memory and file buffer cache (balloon grows)
  - Gives hardware pages to other VMs (those balloons shrink)

- **Identify identical physical pages (e.g., all zeroes)**
  - Map those pages copy-on-write across VMs
Virtualizing I/O

• OSes can no longer interact directly with I/O devices

• Types of communication
  - Special instruction – in/out
  - Memory-mapped I/O
  - Interrupts
  - DMA

• Make in/out trap into VMM

• Use tracing for memory-mapped I/O

• Run simulation of I/O device
  - Interrupt – Tell CPU simulator to generate interrupt
  - DMA – Copy data to/from physical memory of virtual machine
Virtualizing I/O: Three Models

• **Xen**: modify OS to use low-level I/O interface *(hybrid)*
  - Define generic devices with simple interface
    - Virtual disk, virtual NIC, etc.
  - Ring buffer of control descriptors, pass pages back and forth
  - Handoff to trusted domain running OS with real drivers

• **VMware**: VMM supports generic devices *(hosted)*
  - E.g., AMD Lance chipset/PCNet Ethernet device
  - Load driver into OS in VM, OS uses it normally
  - Driver knows about VMM, cooperates to pass the buck to a real device driver (e.g., on underlying host OS)

• **VMware ESX Server**: drivers run in VMM *(hypervisor)*
Virtualized I/O Models

Abramson et al., “Intel Virtualization Technology for Directed I/O”, Intel Technology Journal, 10(3) 2006
Hardware Support

• Intel and AMD implement virtualization support in their recent x86 chips (Intel VT-x, AMD-V)
  - Goal is to fully virtualize architecture
  - Transparent trap-and-emulate approach now feasible
  - Echoes hardware support originally implemented by IBM

• Execution model
  - New execution mode: guest mode
    • Direct execution of guest OS code, including privileged insts
  - Virtual machine control block (VMCB)
    • Controls what operations trap, records info to handle traps in VMM
  - New instruction `vmenter` enters guest mode, runs VM code
  - When VM traps, CPU executes new `vmexit` instruction
  - Enters VMM, which emulates operation
Hardware Support (2)

**Memory**
- Intel extended page tables (EPT), AMD nested page tables (NPT)
- Original page tables map virtual to (guest) physical pages
  - Managed by OS in VM, backwards-compatible
- New tables map physical to machine pages
  - Managed by VMM
- Tagged TLB w/ virtual process identifiers (VPIDs)
  - Tag VMs with VPID, no need to flush TLB on VM/VMM switch

**I/O**
- Constrain DMA operations only to page owned by specific VM
- AMD DEV: exclude pages (c.f. Xen memory paravirtualization)
- Intel VT-d: IOMMU – address translation support for DMA
Summary

• VMMs multiplex virtual machines on hardware
  - Export the hardware interface
  - Run OSes in VMs, apps in OSes unmodified
  - Run different versions, kinds of OSes simultaneously

• Implementing VMMs
  - Virtualize CPU, Memory, I/O

• Lesson: Never underestimate the power of indirection