CS3281 / CS5281

Virtual Machines

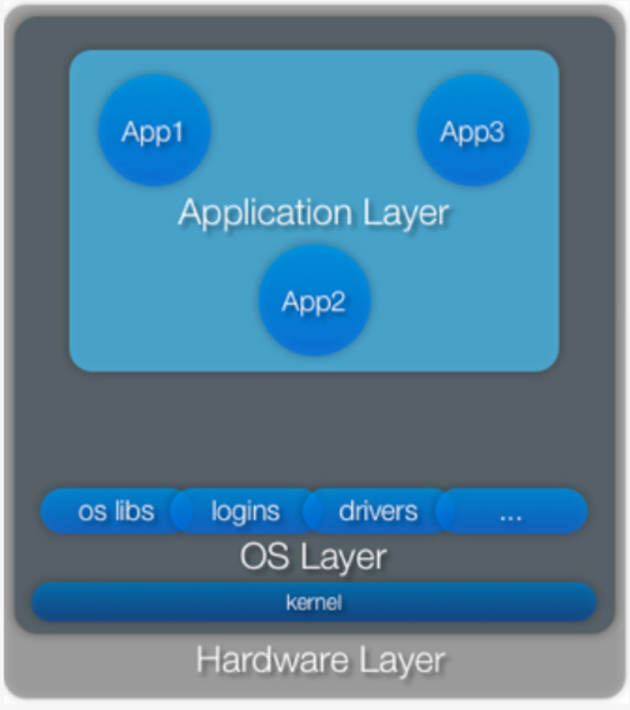
Will Hedgecock
Sandeep Neema
Bryan Ward

ISIS
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

VANDERBILT UNIVERSITY

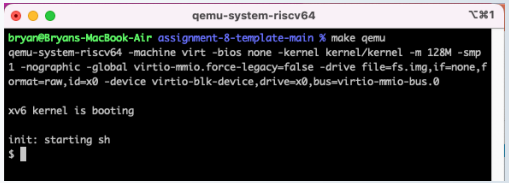
Overview

- We've discussed how operating systems abstract hardware
  - For example, the OS isolates processes from one another so that each thinks it has the whole CPU (via context switching) and all of memory (via virtual memory) to itself
- But what if we want to run different OSes on the same hardware simultaneously?
  - Solution: a virtual machine monitor (VMM); also called a hypervisor
- The VMM needs to provide similar illusions
  - Each "guest" OS i.e., Virtual Machine (VM), must think it's interacting with the physical hardware when the VMM is actually in control



Virtualization vs. Emulation

- In virtualization, the VMM allows limited direct execution on the physical processor
  - The hypervisor must intervene to make sure the VM can't affect other VMs on the same platform
- In emulation, software is used to translate from one platform to another
  - Emulated platform may not be the same as physical platform
    - Our VMs run x86 instructions, but "make qemu" creates an emulated risc-v platform that xv6 runs on
  - Emulation is slower than virtualization because every instruction must be translated
- We focus on virtualization



Why Use VMs?

- Compatibility
  - Example: this class! You need to run Linux but your computer may be running a different OS
- Cost/Consolidation
  - Run multiple OS instances on the same hardware
  - Migrate VMs among multiple servers
  - Cloud providers allow you to “rent” time on a virtual machine running on their hardware
    - Have your own OS environment separate from other VMs sharing hardware
    - E.g., VM on Amazon EC2
  - Reduce hardware and management costs
    - Netflix runs almost all of its compute and storage through Amazon Virtual Machines!
- Debugging/Introspection
  - Test software on different operating systems
  - Forensic analysis for security

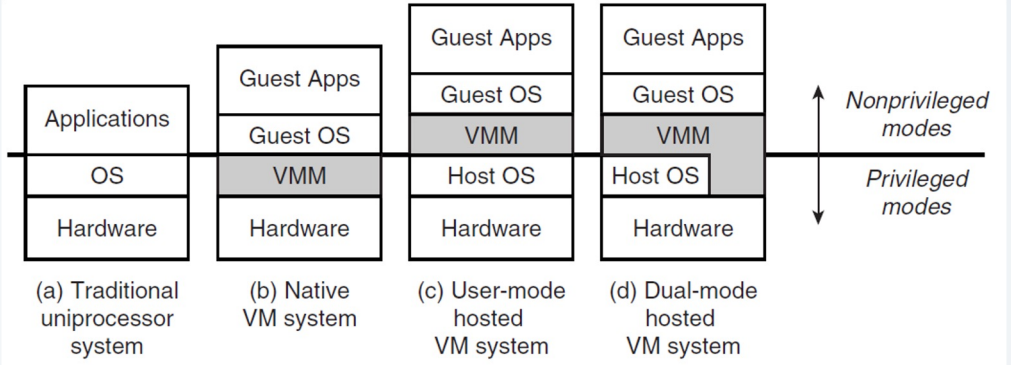
ISIS
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu
VANDERBILT UNIVERSITY

Goals and Challenges?

- Goals:
  - Support multiple OSes concurrently on the same hardware
  - Each OS “thinks” it controls hardware
  - Each OS is isolated and can’t interfere with other OSes

- Challenges?
  - Isolation
    - Prevent one OS from reading or writing other OS resources
  - Sharing hardware
    - Devices, like USB drives or disks
  - Scheduling
    - When does one OS run vs. another?
  - Privileged instructions
    - What runs in ring 0?
    - System calls
    - Interrupts

Types of VMs?



Native and Hosted VM Systems. *(a) The operating system of a traditional uniprocessor executes in the privileged mode. (b) In a native VM system, the VMM operates in the privileged mode. (c–d) A trusted host operating system executes in the privileged mode in a hosted VM system. In both systems, the guest operating system resides in the virtual machine and operates at a less privileged level. The VMM of a hosted system may work entirely in the user mode (c) or in dual mode (d) when parts of the VMM operate in the privileged mode.*

*Figure from "Virtual Machines - Versatile Platforms for Systems and Processes" by Smith and Nair

Lots of different architectures

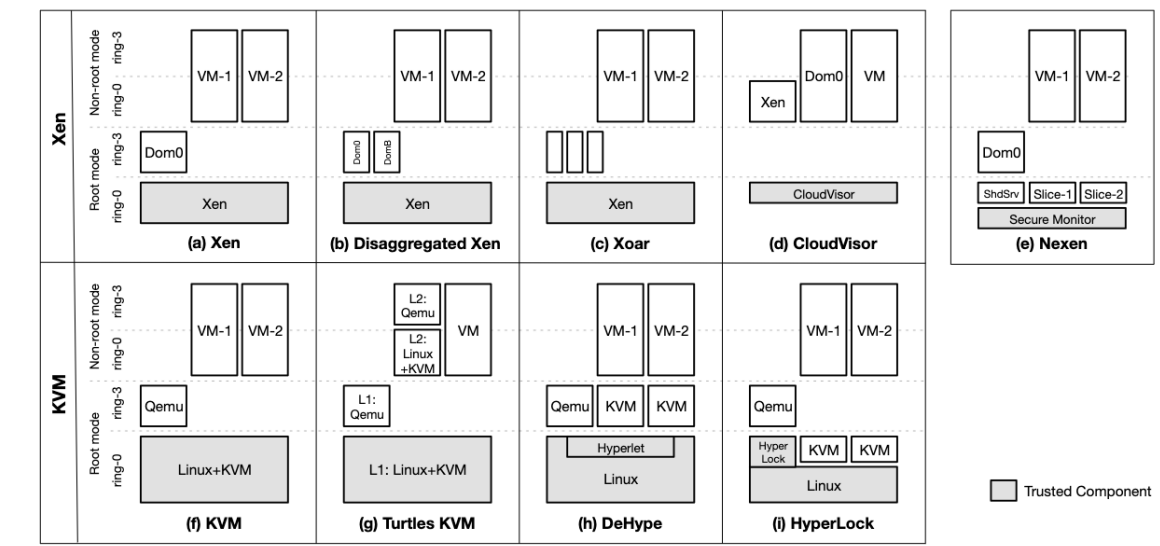


Fig. 1. Comparison between hypervisor reorganization approaches for both Xen and KVM.

There is ongoing research into different virtualization architectures to tradeoff performance and security objectives

How Does a VM Virtualize the CPU?

- Recall how a context switch works with regular processes
  - The OS saves the state of one process and restores the state of another
- With a VMM switching between OSes, a *machine switch* is performed
  - Save the entire state of one OS and restore the entire state of another
- Recall some terminology:
  - A *trap* is an intentional exception that occurs as the result of executing an instruction
- System call implementation
  - Normally: an `ecall` instruction results in the OS's trap handler being invoked
    - And this changes the processor's mode from user mode to kernel mode
- But what should happen when an `ecall` is invoked by a guest OS?
  - We can't let it change the processor's mode! Otherwise the guest is in control of everything

Handling System Calls

- The guest OS makes a system call as usual
  - But the VMM will first get control instead of the guest OS's kernel
  - It then jumps to the actual system-call handler in the guest OS

Process	Operating System
1. System call: Trap to OS	
	2. OS trap handler: Decode trap and execute appropriate syscall routine; When done: return from trap
3. Resume execution (@PC after trap)	

Table B.2: **System Call Flow Without Virtualization**

Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after trap)		

Table B.3: **System Call Flow with Virtualization**

Hardware vs. Software Virtualization

- VMMs try to let software in a VM run directly on the processor
  - Limited direct execution
  - But, the VMM needs to intercept operations that would interfere with the host
- Examples:
  - When the guest OS (1) invokes a system call, (2) modifies its page tables, or (3) accesses a hard disk
- Intercepting these operations is difficult on x86
  - x86 was not originally designed to be virtualized!
- You can detect and intercept these operations in two ways:
  - Hardware virtualization: Intel calls this VT-x; AMD calls it AMD-V (or AMD SVM)
  - Software virtualization: employs binary translation and additional “tricks”

Software Virtualization of x86

- Brief history: original x86 (1970s), big changes in 1980s (protected mode with 286), 32-bit in 1980s (386), 64-bit 2000s (x86_64)
  - Take-away: x86 is an old architecture that didn't originally include virtualization support
- Recall: x86 has 4 privilege levels or rings
  - Most OSes (like Windows and Linux) only use two levels: 0 (privileged) and 3 (unprivileged)
- Virtualization also needs to distinguish between host and guest context
- Host context: no VMM; can be in either ring 0 or ring 3
- Guest context: a VM is active; usually easy in ring 3
  - Problems arise in intercepting guest context in ring 0 (kernel mode)

Intercepting Ring-0 Code

- One solution: use binary translation; analyze and transform all guest code so that it neither sees or modifies true state of the CPU
  - This is the approach of QEMU
  - Downside: expensive
- VirtualBox (VB) uses a ring-0 kernel driver so that it can take over when needed
  - i.e., part of VirtualBox is running in privileged mode
- VB is in one of three states
  - Guest code in ring 3: runs unmodified at full speed as much as possible
  - Guest code in ring 0: VB reconfigures guest so that it's actually running in ring 1
    - VB takes over when guest does something privileged
  - VB (the VMM) is active

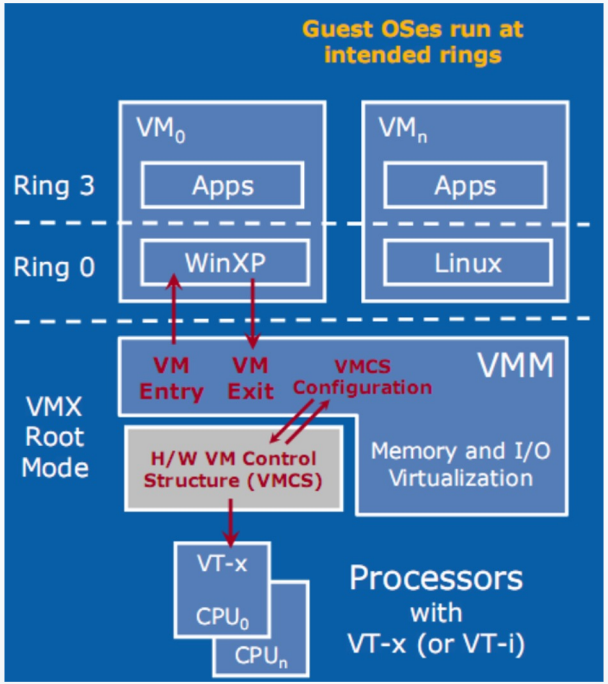
Intercepting Ring-0 Code (cont.)

- There are several low-level issues with the approach above
- So, VB uses a Code Scanning and Analysis Manager (CSAM) and Patch manager (PATM)
- Before executing ring-0 guest code
  - CSAM scans it to discover problematic instructions
  - PATM performs in-situ (in-place) patching of instructions
    - Jumps to VMM memory where a code generator has placed a "suitable" implementation
- This is complex!

Hardware Virtualization

- Intel's VT-x and AMD's AMD-V introduce new CPU operation modes:
  - VMX root mode: four privilege levels (rings), same instruction set plus new virtualization specific instructions. Used by host OS without virtualization, and also used by VMM when virtualization is active
  - VMX non-root mode: still four rings and same instruction set, but new Virtual Machine Control Structure (VMCS) controls the CPU operation and determines how certain instructions behave. Guest OSes run in non-root mode
- In a nutshell: VT-x lets guest code run in the ring it expects!
  - The VMM uses the VMCS to control how certain situations are handled
- Example: the VMM can set up the VMCS so that the guest can write certain control registers but not others

VT-x in Pictures



Software vs. Hardware Virtualization Recap

- VT-x and AMD-V avoid some problems faced by pure software virtualization
  - Avoid the problem of having the guest OS and the VMM share an address space
  - Guest OS kernel code runs at ring-0 (where it expects to run)
- Disadvantage of hardware virtualization vs. software:
  - The overhead of VM exits (i.e., having the VM emulate an instruction) is relatively high

Paravirtualization

- In pure virtualization the VMM intercepts and translates privileged instructions
  - "Paravirtualization" means that the guest OS is modified to run on a VMM
- In paravirtualized VM, the guest OS can issue "hypercalls" to the hypervisor
  - Hypercalls are the syscall equivalent. i.e., application:syscall :: kernel:hypercall
  - Request page-table updates directly
  - VMM is utilized more efficiently
  - etc.
  - Paravirtualization can be more efficient
    - Hypercall more efficient than intercepting and translating privileged instructions
    - VM may know that physical pages have already been zeroed by VMM
- Paravirtualization supported by Windows and Linux

Paravirtualization (cont.)

- A paravirtualization interface can be exposed by a VMM to allow the guest OS to communicate with the VMM
  - This allows guest software to be executed more efficiently
  - But it also requires that the guest OS be aware of the paravirtualization interface
- Many OSes (including Windows and Linux) support paravirtualization
  - In other words: when you run Windows or Linux as a guest, they “know” they’re being virtualized to some degree -- this is good!
- Different types of paravirtualization interfaces
  - Minimal: says that the environment is virtualized; reports TSC and APIC frequency to guest
  - KVM: support paravirtualized clocks and SMP spinlocks
  - Hyper-V: recognized by Windows 7 and newer; supports virtualized clocks, guest debugging

Memory Virtualization

- Recall: the OS provides virtual address spaces to each process
  - A virtual page is mapped to a physical page
- With VMMs, we need another level of indirection
  - Guest OS: maps virtual to “physical”
  - VMM: maps “physical” to actual machine memory
- Simplified picture on right

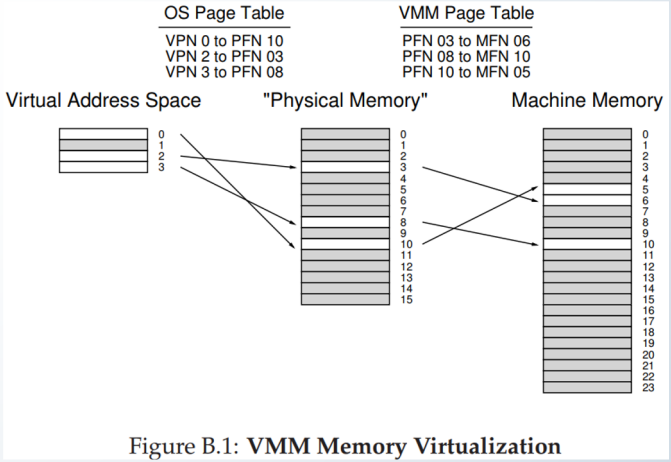


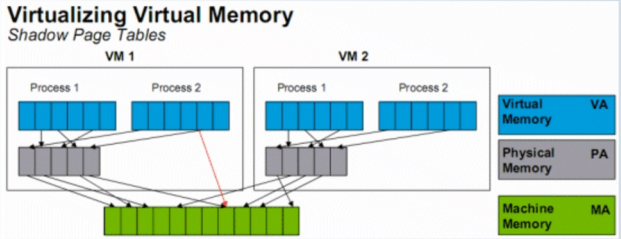
Figure B.1: **VMM Memory Virtualization**

Memory Virtualization with the IA-32

- Recall how the page table is architected
  - The TLB is maintained by hardware; when a TLB miss happens, the hardware walks the page table and inserts the correct entry in the TLB
    - If there's no entry in the page table, the OS's page fault handler takes over and either (a) installs an entry in the page table, or (b) gives a seg fault back to the offending process
- On the IA-32 architecture, the VMM can use a shadow page table for each process's page table
  - The VMM monitors for changes the OS makes to each page table and updates the shadow page table accordingly
- Guest OS: page table maps from virtual to "physical"
- VMM: shadow page table maps from virtual to actual machine memory

Shadow Page Tables

- The page table pointer for a guest process is “virtualized”
  - In other words, the page table register doesn’t hold what the guest OS thinks it holds; it holds the base address of the current shadow page table
- Whenever the guest OS accesses its page table register, the VMM is notified
  - If the guest was reading: the VMM returns the virtual CR3 value
  - If the guest was writing: the VMM updates the virtual version and then updates the real CR3 to point to the corresponding shadow table
- Newer hardware extensions enable “nested paging”
  - This implements memory management for guests in hardware



Shadow page tables enable the MMU to map directly from blue to green

Nested Paging

- Intel and AMD provide hardware extensions for virtual memory with VMMs
  - These are called nested paging or EPT
- Big idea: do memory management for VMMs in hardware
  - The guest can handle paging without intervention from the VMM
  - As opposed to shadow page tables, which require the VMM to maintain the real page tables

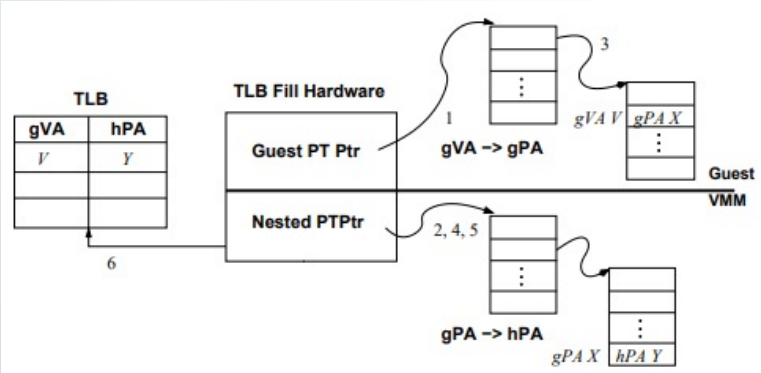


Figure from: https://www.vmware.com/pdf/asplos235_adams.pdf

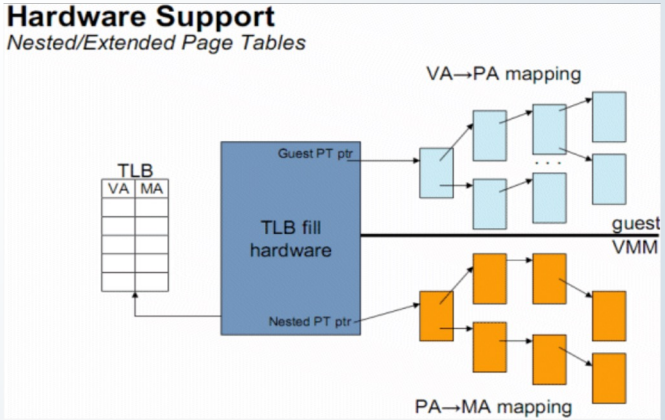
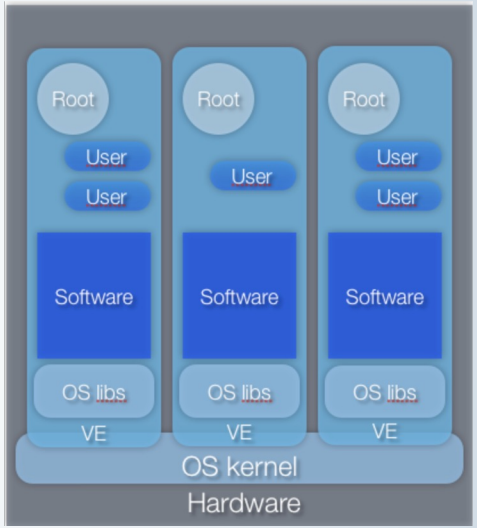


*Figure from https://www.anandtech.com/show/2480/10

Comparison to OS-Level Virtualization

- OS-level virtualization is enabled inside an individual OS
  - Examples: Docker containers, Solaris Zones, or “jails”
  - Many people refer to all of these as “containers”
- Programs inside a container can only access that container’s contents
- Under the hood: many container systems use the Linux *cgroup* facility
- The cgroup framework provides:
  - Limits on resources (e.g., a certain amount of memory)
  - Prioritization
  - Accounting (keep track of a group’s useage)
  - Control: a group of processes can be frozen or stopped and restarted



ISIS
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu
VANDERBILT UNIVERSITY

References

- Introduction: https://www.anandtech.com/show/2653
- VirtualBox documentation: https://www.virtualbox.org/manual/ch10.html
- AnandTech article: https://www.anandtech.com/show/2480/10
- https://stackoverflow.com/questions/9832140/what-exactly-do-shadow-page-tables-for-vmms-do
- https://www.anandtech.com/show/2480/10
- https://www.anandtech.com/show/3827/virtualization-ask-the-experts-1
- Control groups: https://www.linuxjournal.com/content/everything-you-need-know-about-linux-containers-part-i-linux-control-groups-and-process
- Intel architecture manual, Volume 3B, chapter 23
- Nexen: https://ipads.se.sjtu.edu.cn/_media/publications/nexen-ndss17.pdf