第 3 讲：Virtual Machine Monitor

第二节：How VMM works

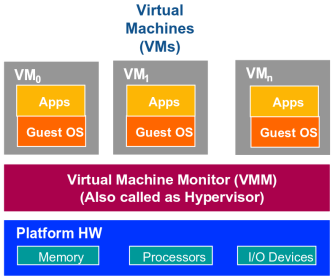
陈渝

清华大学计算机系

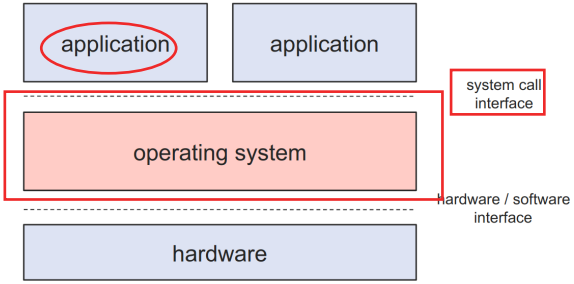
yuchen@tsinghua.edu.cn

2020 年 2 月 29 日

How VMM works

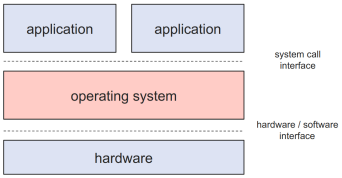


Start with a "simpler" question: how do (regular) machines work?



- What is computer hardware? (CPU, MEM, IO)
- What is an OS?
- What 's an application? (relies on the system call interface)

How VMM works

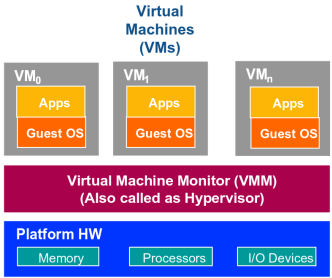


Start with a "simpler" question: how do (regular) machines work?

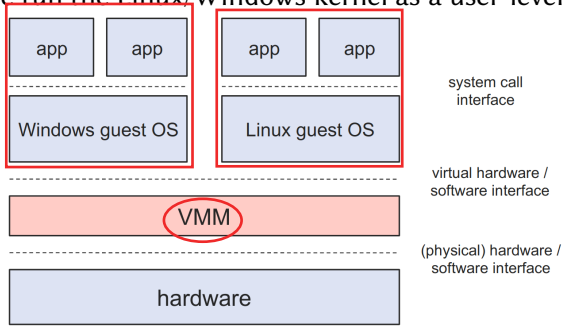
Process	Hardware	Operating System
1. Execute instructions (add, load, etc.)		
2. System call: Trap to OS		
	3. Switch to kernel mode; Jump to trap handler	
		4. In kernel mode; Handle system call; Return from trap
	5. Switch to user mode; Return to user code	
6. Resume execution (@PC after trap)		

Executing a System Call

How VMM works – CPU

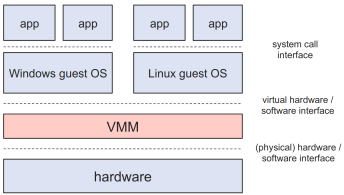


What if we run the Linux/Windows kernel as a user-level program?



- What happens when Linux issues a sensitive instruction?
- What (virtual) hardware devices should Windows see?
- How do you prevent apps running on Linux from hurting Linux?

How VMM works – CPU

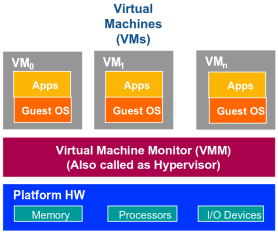


What if we run the Linux/Windows kernel as a user-level program?

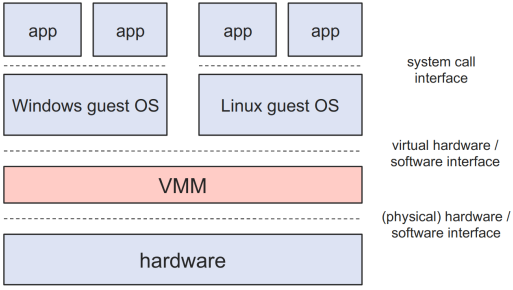
Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after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

System Call Flow with Virtualization

How VMM works – CPU



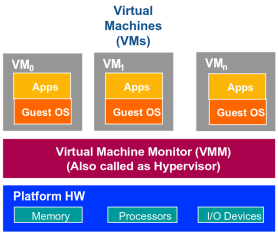
What if we run the Linux/Windows kernel as a user-level program?



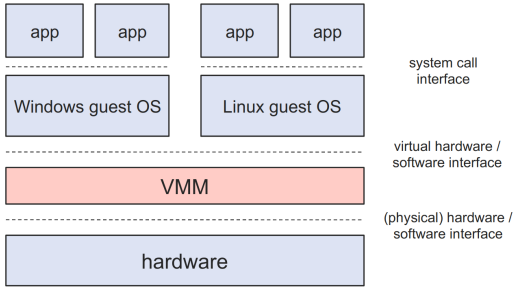
Answer: rely on CPU to trap sensitive instructions and hand off to VMM

- VMM emulates the effect of sensitive instruction on the virtual hardware that it provides to its guest OSs
- VMM provides a virtual HW/SW interface to guest OSs by trapping and emulating sensitive instructions

How VMM works – CPU



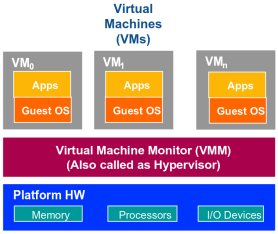
What if we run the Linux/Windows kernel as a user-level program?



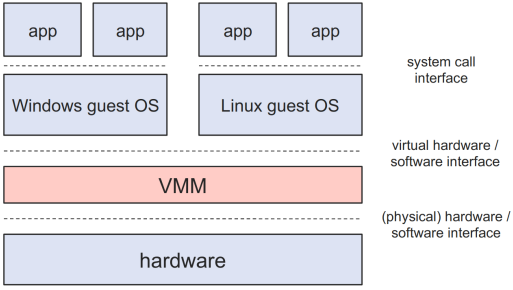
Goldberg (1974): two classes of instructions

- privileged instructions: those that trap when CPU is in user-mode
- sensitive instructions: those that modify hardware configuration or resources, and those whose behavior depends on HW configuration

How VMM works – CPU

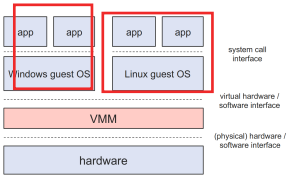


What if we run the Linux/Windows kernel as a user-level program?



A VMM can be constructed efficiently and safely if the set of ~~sensitive instructions~~ is a subset of the set of privileged instructions.

How VMM works – memory

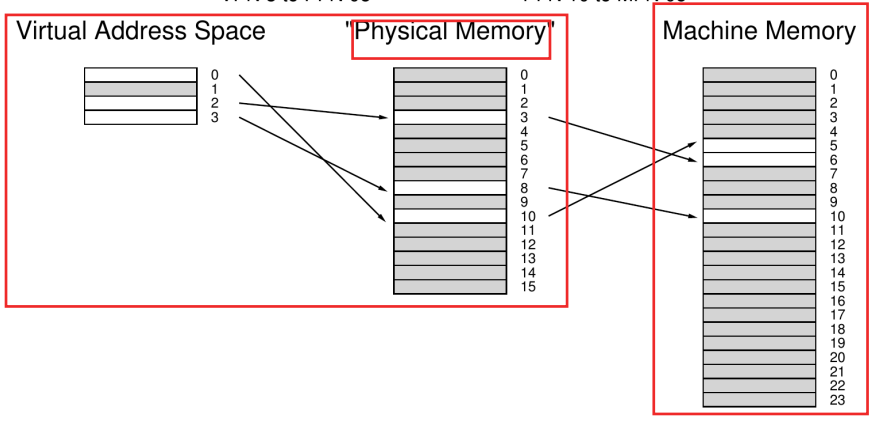


OS Page Table

VPN 0 to PFN 10
VPN 2 to PFN 03
VPN 3 to PFN 08

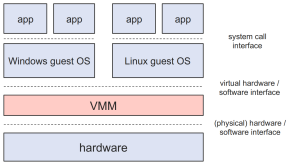
VMM Page Table

PFN 03 to MFN 06
PFN 08 to MFN 10
PFN 10 to MFN 05



VMM Memory Virtualization

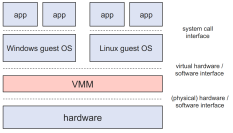
How VMM works – memory



Process	Operating System
1. Load from memory: TLB miss: Trap	
	2. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid: get PFN, update TLB; Return from trap
3. Resume execution (@PC of trapping instruction); Instruction is retried; Results in TLB hit	

TLB Miss Flow without Virtualization

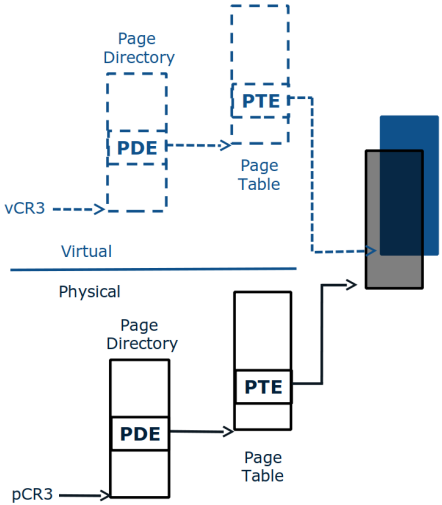
How VMM works – memory

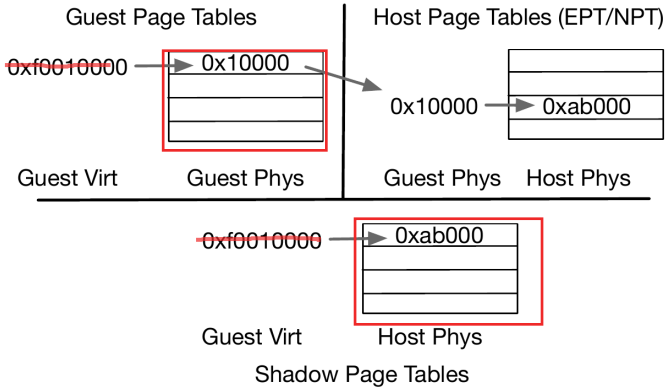


Process	Operating System	Virtual Machine Monitor
1. Load from mem TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap
7. Resume execution (@PC of instruction); Instruction is retried; Results in TLB hit		

TLB Miss Flow with Virtualization

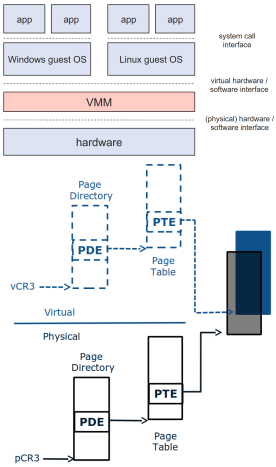
How VMM works – memory





How does the virtual machine monitor get involved with a hardware-managed TLB?

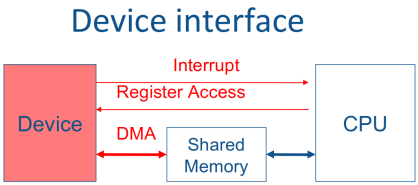
How VMM works – memory



How does the virtual machine monitor get involved with a hardware-managed TLB?

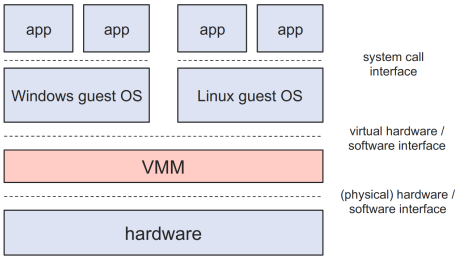
- VMM doesn't have a chance to run on each TLB miss to sneak its translation into the system.
- VMM must closely monitor changes the OS makes to each page table
- VMM must keep a **shadow page table** that maps the virtual addresses of each process to the VMM's desired machine pages
- VMM installs a process's **shadow page table** whenever the OS tries to install the process's OS-level page table

How VMM works – IO

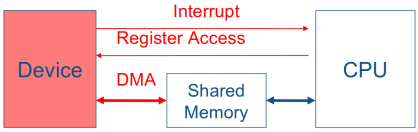


- Interaction between device and driver:
  - Driver programs device through register access
  - Device notifies driver through interrupt
  - Device could DMA for massive data movement

How VMM works – IO

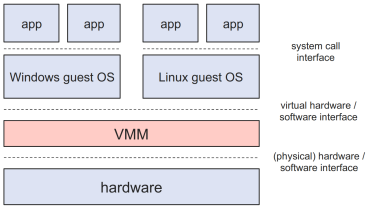


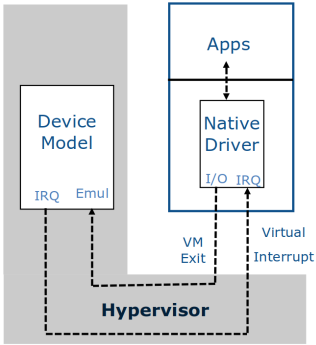
Device interface



- I/O Virtualization requires the hypervisor to present guest a complete device interface
  - Presenting an existing interface: Software Emulation
  - Presenting an existing interface: Direct assignment
- Presenting a brand new interface
  - Paravirtualization

How VMM works – IO

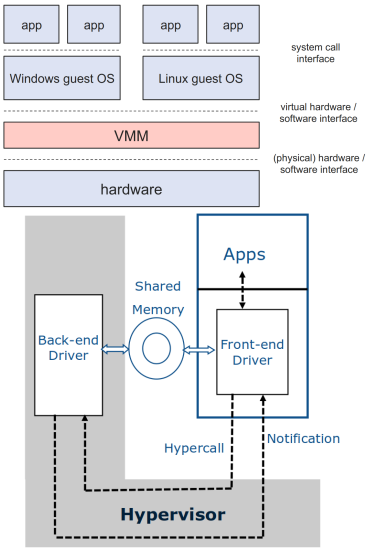




Software Emulation

- Guest runs native device driver, e.g. NE2000
  - I/O access is trap-and-emulated by device model in hypervisor
  - Translation for MMIO is zapped
  - Virtual Interrupt is signaled by device model per semantics
- Excessive trap and emulation

How VMM works – IO



Paravirtualization

- A new front-end driver (FE driver) is run in guest
  - Optimized request through hypercall
- Hypervisor runs a back-end driver (BE driver) to service request from FE driver
  - Notify guest for processing
- Shared memory is used for massive data communication
  - To reduce guest/hypervisor boundary crossin
  - e.g. Xen VNIF, KVM Virtio-Net