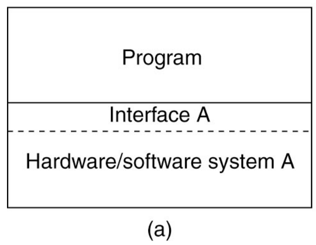
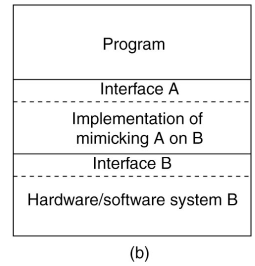
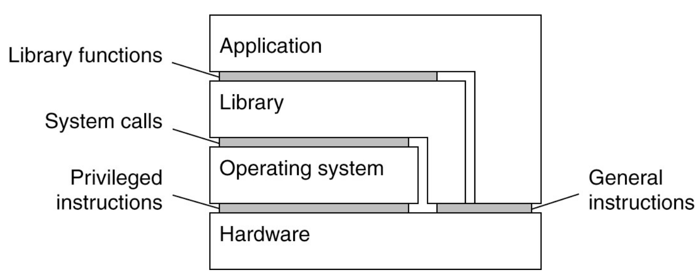
Module 1: Virtualization





- Virtualization: extend or replace an existing interface to mimic the behavior of another system.
  - Introduced in 1970s: run legacy software on newer mainframe hardware
- Handle platform diversity by running apps in VMs
  - Portability and flexibility

Types of Interfaces



- Different types of interfaces
  - Assembly instructions
  - System calls
  - APIs
- Depending on what is replaced /mimiced, we obtain different forms of virtualization

Types of Virtualization

- Emulation
  - VM emulates/simulates complete hardware
  - Unmodified guest OS for a different PC can be run
    - Bochs, VirtualPC for Mac, QEMU
- Full/native Virtualization
  - VM simulates "enough" hardware to allow an unmodified guest OS to be run in isolation
    - Same hardware CPU
  - IBM VM family, VMWare Workstation, Parallels, VirtualBox

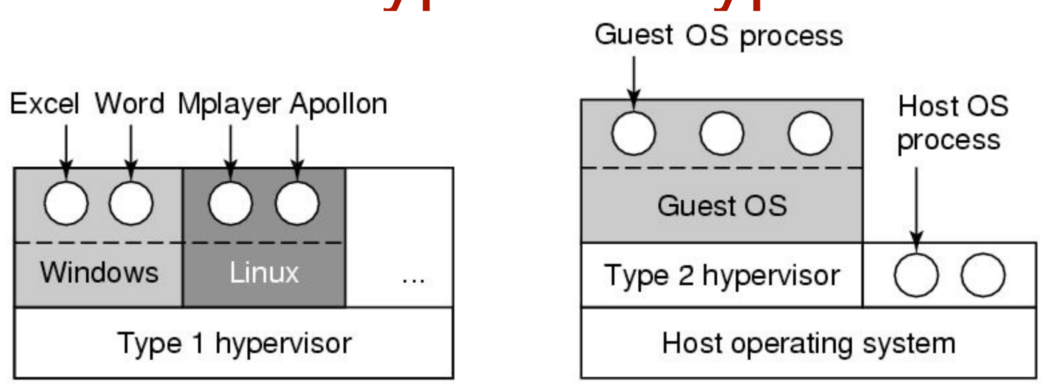
Types of virtualization

- Para-virtualization
  - VM does not simulate hardware
  - Use special API that a modified guest OS must use
  - Hypercalls trapped by the Hypervisor and serviced
  - Xen, VMWare ESX Server
- OS-level virtualization
  - OS allows multiple secure virtual servers to be run
  - Guest OS is the same as the host OS, but appears isolated
    - apps see an isolated OS
  - Solaris Containers, BSD Jails, Linux Vserver, Linux containers, Docker
- Application level virtualization
  - Application is gives its own copy of components that are not shared
    - (E.g., own registry files, global objects) - VE prevents conflicts
  - JVM, Rosetta on Mac (also emulation), WINE

Computing Parable

- Lion and the Rabbit

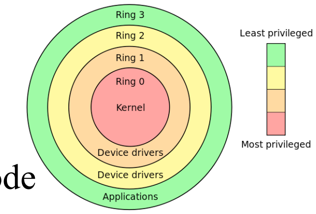
Module 2: Types of Hypervisors



(a) (b)

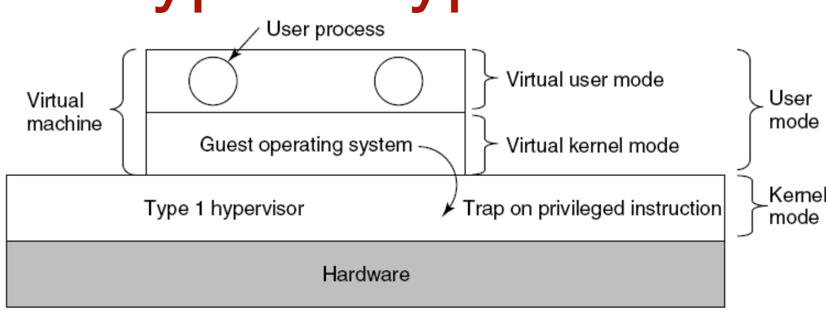
- Hypervisor/VMM: virtualization layer
  - resource management, isolation, scheduling, …
- Type 1: hypervisor runs on "bare metal"
- Type 2: hypervisor runs on a host OS
  - Guest OS runs inside hypervisor
- Both VM types act like real hardware

How Virtualization works?



- CPU supports kernel and user mode (ring0, ring3)
  - Set of instructions that can only be executed in kernel mode
    - I/O, change MMU settings etc -- *sensitive instructions*
  - Privileged instructions: cause a trap when executed in user mode
- Result: type 1 virtualization feasible if sensitive instruction subset of privileged instructions
- Intel 386: ignores sensitive instructions in user mode
  - Can not support type 1 virtualization
- Recent Intel/AMD CPUs have hardware support
  - Intel VT, AMD SVM
    - Create containers where a VM and guest can run
    - Hypervisor uses hardware bitmap to specify which inst should trap
    - Sensitive inst in guest traps to hypervisor

Type 1 hypervisor

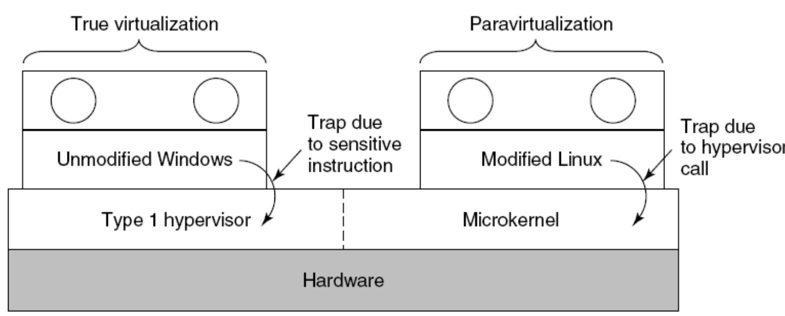


- Unmodified OS is running in user mode (or ring 1)
  - But it thinks it is running in kernel mode (*virtual kernel mode)*
  - privileged instructions trap; sensitive inst-> use VT to trap
  - Hypervisor is the “real kernel”
    - Upon trap, executes privileged operations
    - Or emulates what the hardware would do

Type 2 Hypervisor

- VMWare example
  - Upon loading program: scans code for basic blocks
  - If sensitive instructions, replace by Vmware procedure
    - Binary translation
  - Cache modified basic block in VMWare cache
    - Execute; load next basic block etc.
- Type 2 hypervisors work without VT support
  - Sensitive instructions replaced by procedures that emulate them.

Paravirtualization



- Both type 1 and 2 hypervisors work on unmodified OS
- Paravirtualization: modify OS kernel to replace all sensitive instructions with hypercalls
  - OS behaves like a user program making system calls
  - Hypervisor executes the privileged operation invoked by hypercall.

Module 3: Memory virtualization

- OS manages page tables
  - Create new pagetable is sensitive -> traps to hypervisor
- hypervisor manages multiple OS
  - Need a second shadow page table
  - OS: VM virtual pages to VM's physical pages
  - Hypervisor maps to actual page in shadow page table
  - Two level mapping
  - Need to catch changes to page table (not privileged)
    - Change PT to read-only - page fault
    - Paravirtualized - use hypercalls to inform

I/O Virtualization

- Each guest OS thinks it "owns" the disk
- Hypervisor creates "virtual disks"
  - Large empty files on the physical disk that appear as "disks" to the guest OS
    - Hypervisor converts block # to file offset for I/O
  - DMA need physical addresses
    - Hypervisor needs to translate
- NIC Virtualization

Virtual Appliances & Multi-Core

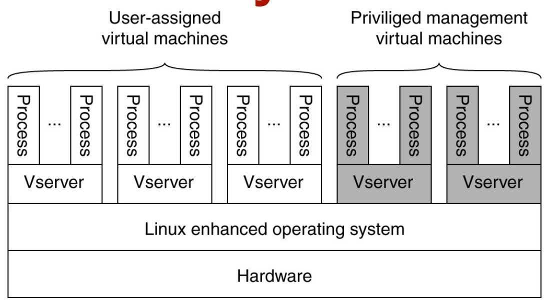
- Virtual appliance: pre-configured VM with OS/ apps pre-installed
  - Just download and run (no need to install/configure)
  - Software distribution using appliances
- Multi-core CPUs
  - Run multiple VMs on multi-core systems
  - Each VM assigned one or more vCPU
  - Mapping from vCPUs to physical CPUs

- Today: Virtual appliances have evolved into docker containers

Use of Virtualization Today

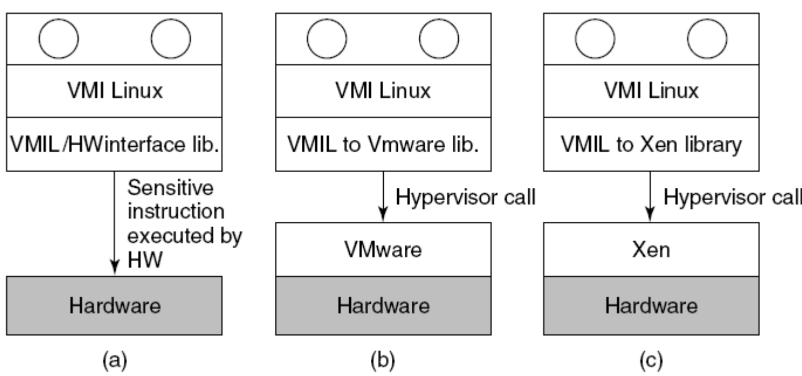
- Data centers:
  - server consolidation: pack multiple virtual servers onto a smaller number of physical server
    - saves hardware costs, power and cooling costs
- Cloud computing: rent virtual servers
  - cloud provider controls physical machines and mapping of virtual servers to physical hosts
  - User gets root access on virtual server
- Desktop computing:
  - Multi-platform software development
  - Testing machines
  - Run apps from another platform

Case Study: PlanetLab



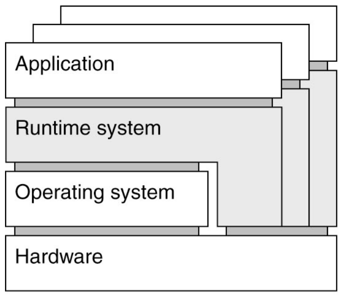
- Distributed cluster across universities
  – Used for experimental research by students and faculty in networking and distributed systems
- Uses a virtualized architecture
  – Linux Vservers
  – Node manager per machine
  – Obtain a "slice" for an experiment: slice creation service

Virtual machine Interface

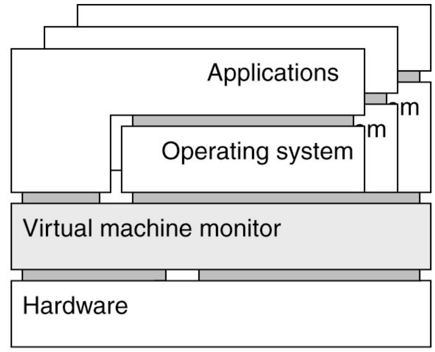


- Standardize the VM interface so kernel can run on bare hardware or any hypervisor

Examples



(a)



(b)

- Application-level virtualization: “process virtual machine”
- VMM /hypervisor