CS343 - Operating Systems

Module-9A

Virtual Machines

Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

http://www.iitg.ac.in/johnjose/

Overview

- Benefits and Features
- Building Blocks
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components

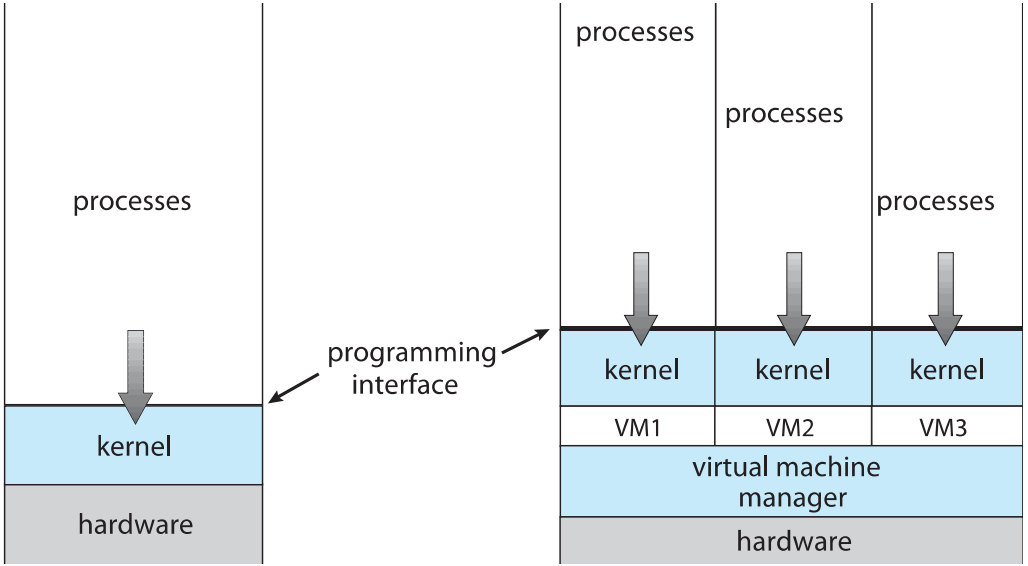
Objectives

- To explore the history and benefits of virtual machines
- To discuss the various virtual machine technologies
- To describe the methods used to implement virtualization
- To show the most common hardware features that support virtualization and explain how they are used by operating-system modules

Virtual Machine Concept

- Facilitate abstract hardware into several different execution environment
- A layer of virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run
  - **Host** – underlying hardware system
  - **Virtual machine manager** (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is **identical** to the host
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

System Models



Non-virtual machine

Virtual machine

Implementation of VMMs

- **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
  - IBM LPARs and Oracle LDOMs are examples
- **Type 1 hypervisors** – OS-like software built to provide virtualization
  - VMware ESX, Joyent SmartOS, and Citrix XenServer
- **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
  - Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
  - VMware Workstation, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs

- **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
- **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
  - Used by Oracle Java and Microsoft.Net
- **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU
- **Application containment** - Provides virtualization like features by segregating applications from the OS, making them secure, manageable
  - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

History

- IBM mainframes (1972) - allowed multiple users to share a batch system
- A VMM provides an environment for programs that is essentially identical to the original machine
- Programs running within that environment show only minor performance decreases
- The VMM is in complete control of system resources
- Intel CPUs (1990s) try virtualizing on general purpose PCs
  - **Xen** and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - A virus less likely to spread
  - Sharing is provided though via shared file system volume, network communication
- Freeze, suspend, running VM
  - Then can move or copy somewhere else and resume
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - Clone by creating copy and running both original and copy
- Run multiple, different OSes on a single machine

Benefits and Features

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- **Live migration** – move a running VM from one host to another!
  - No interruption of user access
- **Cloud computing -** Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

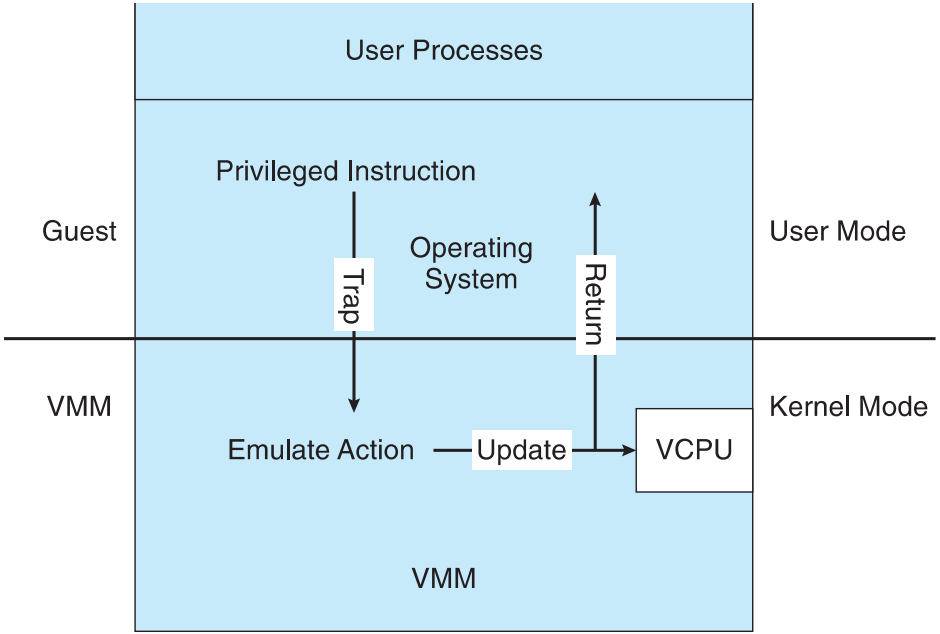
Building Blocks

- Generally difficult to provide an **exact** duplicate of underlying machine when CPU has only dual mode (user and kernel modes)
- Most VMMs implement **virtual CPU** (**VCPU**) to represent state of CPU per guest
- When guest context switched onto CPU by VMM, information from VCPU loaded and stored

Building Block – Trap and Emulate

- Dual mode CPU means guest executes in user mode
  - Kernel runs in kernel mode
  - Not safe to let guest kernel run in kernel mode too
  - So VM needs two modes – virtual user mode and virtual kernel mode
    - Both of which run in real user mode
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode

Trap-and-Emulate Virtualization Implementation



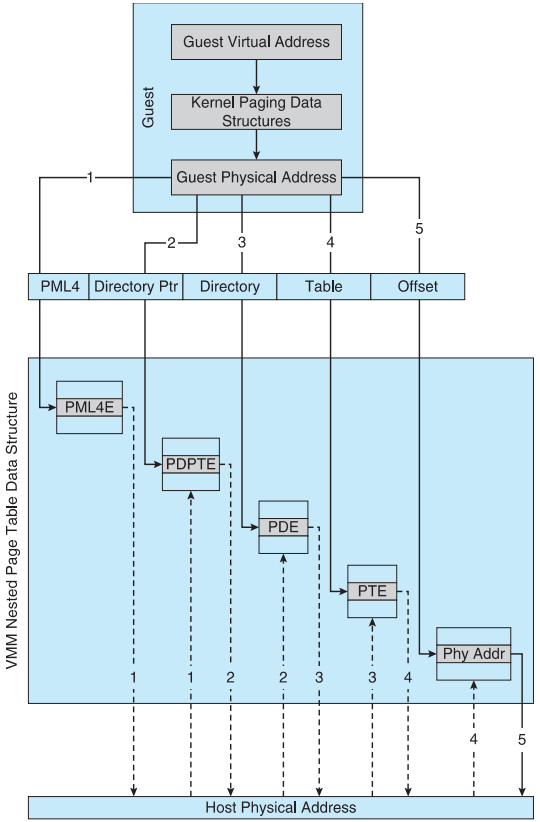
Nested Page Tables

- Memory management another general challenge to VMM implementations
- VMM keeps page-table state for both guests believing they control the page tables, but VMM control the tables.
- Common method is **nested page tables** (**NPTs**)
- Each guest maintains page tables to translate virtual to physical addresses
- VMM maintains per guest NPTs to represent guest's page-table state
- When a guest works on CPU, →VMM makes that guest's NPTs the active system page tables
- When guest tries to change page table, → VMM makes equivalent change to NPTs and its own page tables
  - Can cause many more TLB misses and slower performance

Hardware Support for VMM

- All virtualization needs some HW support
- Generally define more CPU modes – guest and host
- In guest mode, guest OS thinks it is running natively, sees devices as defined by VMM for that guest

Nested Page Tables

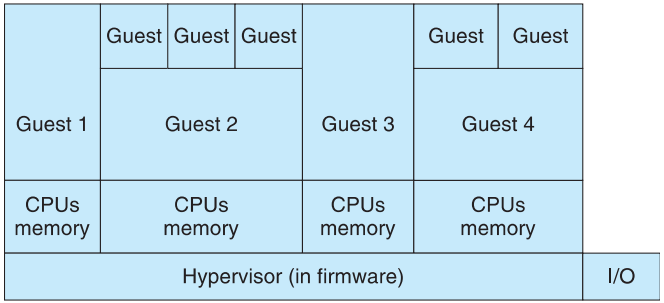


Types of Virtual Machines and Implementations

- Whatever the type, a VM has a lifecycle
  - Created by VMM
  - Resources assigned to it (number of cores, amount of memory, networking details, storage details)
  - In type 0 hypervisor, resources usually dedicated
  - Other types dedicate or share resources, or a mix
  - When no longer needed, VM can be deleted, freeing resource
- Steps simpler, faster than with a physical machine install
  - Can lead to **virtual machine sprawl** with lots of VMs, history and state difficult to track

Types of VMs – Type 0 Hypervisor

- There are partitions and domains
- A HW feature implemented by firmware
- Each guest has dedicated HW
- I/O a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons that other guests communicate with for shared I/O



Types of VMs – Type 1 Hypervisor

- Datacenter managers control OSes by controlling the Type 1 hypervisor
- Consolidation of multiple OSes and apps onto less HW
- Move guests between systems to balance performance
- Snapshots and cloning
- Special purpose operating systems that run natively on HW
  - Rather than providing system call interface, create run and manage guest OSes
  - Guests generally don’t know they are running in a VM
  - Provide other traditional OS services like CPU and memory management

Types of VMs – Type 1 Hypervisor

- Another variation is a general purpose OS that also provides VMM functionality
  - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
  - Perform normal duties as well as VMM duties
  - Typically less feature rich than dedicated Type 1 hypervisors
- Treat guests OSes as just another process

Types of VMs – Type 2 Hypervisor

- Less interesting from an OS perspective
  - Very little OS involvement in virtualization
  - VMM is simply another process, run and managed by host
    - Even the host doesn't know they are a VMM running guests
  - Tend to have poorer overall performance because can't take advantage of some HW features
  - But also a benefit because require no changes to host OS
    - Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

Types of VMs – Paravirtualization

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
- Xen, (leader in paravirtualized space) has simple device abstractions
  - Good communication between guest and VMM about device I/O
  - Each device has circular buffer shared by guest and VMM via shared memory
- Memory management does not include nested page tables
  - Each guest has own read-only tables
  - Guest uses call to hypervisor when page-table changes needed
- Guest had to be modified to use run on paravirtualized VMM

Programming Environment Virtualization

- Programming language is designed to run within custom-built virtualized environment
- For example Oracle Java has many features that depend on running in **Java Virtual Machine** (**JVM**)
- In this case virtualization is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
- JVM compiled to run on many systems (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system

Types of VMs – Emulation

- Emulation allows guest to run on different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
  - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
  - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
  - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

Types of VMs – Application Containment

- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
  - If applications compiled for the host operating system, don't need full virtualization to meet these goals
- Oracle **containers** / **zones** for example create virtual layer between OS and apps
  - Only one kernel running – host OS
  - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
  - Each zone has its own applications; networking stack, addresses, and ports; user accounts, etc

Virtualization and Operating-System Components

- Lot of design challenges at operating system aspects for implementing virtualization
  - CPU scheduling, memory management, I/O, storage, and unique VM migration feature
  - How do VMMs schedule CPU use when guests believe they have dedicated CPUs?
  - How can memory management work when many guests require large amounts of memory?

CPU Scheduling

- Even single-CPU systems act like multiprocessor ones when virtualized
  - One or more virtual CPUs per guest
- Generally VMM has one or more physical CPUs and number of threads to run on them
- When enough CPUs for all guests → VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs → CPU **overcommitment**
  - VMM can use standard scheduling algorithms to put threads on CPUs

Memory Management

- Extra management efficiency from VMM during oversubscription
- Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store
- Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests
- **Balloon** memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available

Input/Output

- Easier for VMMs to integrate with guests because I/O has lots of variation
  - Already somewhat segregated / flexible via device drivers
  - VMM can provide new devices and device drivers
- But overall I/O is complicated for VMMs
- Networking also complex as VMM and guests all need network access
  - VMM can **bridge** guest to network (allowing direct access)
  - And / or provide **network address translation** (**NAT**)
  - NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

Storage Management

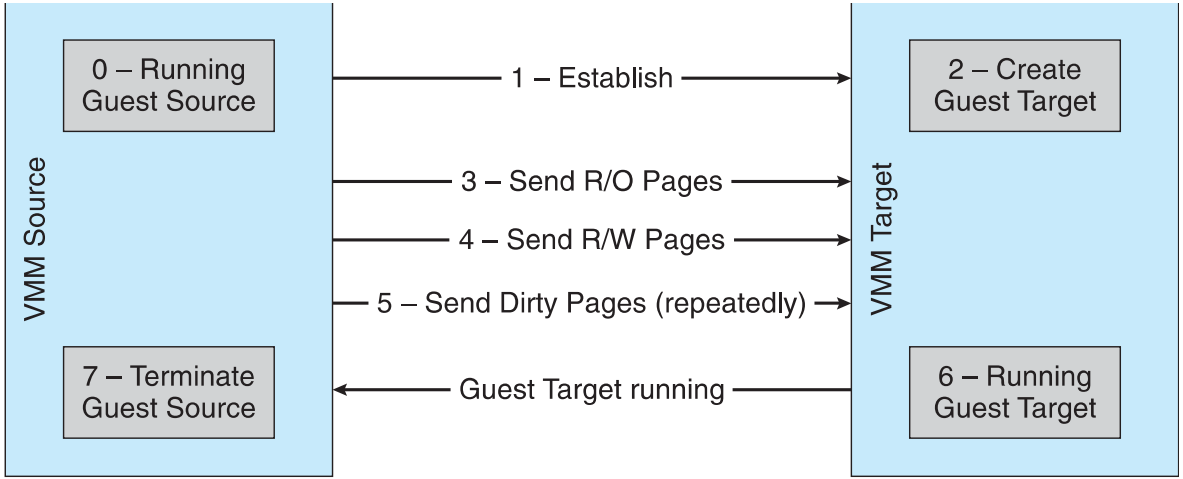
- Need to support many guests per VMM (not by standard disk partition)
- Type 1 → storage of guest root disks and config information within file system provided by VMM as a **disk image**
- Type 2 → store as files in file system provided by host OS
- Duplicate file → create new guest
- Move file to another system → move guest
- **Physical-to-virtual** (**P-to-V**) convert native disk blocks into VMM format
- **Virtual-to-physical** (**V-to-P**) convert virtual format to native disk format

Live Migration

- Running guest can be moved between systems, without interrupting user access to the guest or its apps
- Very useful for resource management, maintenance downtime, etc

1. The source VMM establishes a connection with the target VMM
2. The target creates a new guest by creating a new VCPU, etc
3. The source sends all read-only guest memory pages to the target
4. The source sends all read-write pages to the target, marking them clean
5. The source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty
6. Source VMM sends VCPU's final state details, sends final dirty pages, and tells target to start running the guest
7. Once target acknowledges that guest running, source terminates guest

Live Migration of Guest Between Servers



johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/