Virtual Machines
- Overview and Definition
- Virtualization Techniques
- Resource Virtualization
- Hardware Virtualization

Computer System

Hardware/Software Interfaces: ISA
- Instruction Set Architecture
- Hardware instructions:
  - User-level: Directly executed by applications
  - System-level: Executed by the OS
Hardware/Software Interfaces: ABI

- Application Binary Interface
- Instructions executable by a user-level application
  - User instructions (from ISA)
  - System calls (supported by OS)

Diverse Architectures

Incompatibility

Virtual Machines

Image Source: Virtual Machines: Versatile Platforms for Systems and Processes (Book), Smith et al., 2005
Types of Virtualization

- Emulation: Replicate a hardware architecture in software
  - E.g.: Bochs
- Process VMs: Virtualization at process-level (i.e., program code + resources)
  - E.g.: JVM
- System VMs: A virtual environment for running OS + applications
  - E.g.: VMWare

Virtual Machines

- Virtualize whole system environment
  - OS, Applications
- Virtual Machine Monitor (VMM):
  - Software layer that provides virtualization support
  - Manages multiple virtual machines
- Virtual Machine: Replica of the native machine environment
- Guest OS: OS running inside a VM

Why do virtualization?

- Popek, Goldberg 1974
- A VM is an efficient, isolated duplicate of the real machine
- A VMM should have 3 characteristics:
  - Equivalence: A program running in a VM should behave identically to running in native mode (except for performance)
  - Efficiency: Programs running in a VM should show at worst only minor decreases in speed
  - Resource control: VMM should have complete control of system resources
Virtualization Theory

- Describes the requirements for an architecture to be virtualizable
- Types of instructions:
  - Sensitive: Change system state (e.g.: resource allocation, protected data, etc.)
  - Innocuous: Regular instructions
- Privileged instructions:
  - Can be executed only in kernel mode
  - Trap in user mode

Virtualization Theory (contd.)

- Theorem: An architecture can be virtualized if sensitive instructions are a subset of privileged instructions
  - Intuition: VMM can capture all sensitive instructions
  - Equivalence: A program can get same behavior
  - Efficiency: All innocuous instructions should be executed natively
  - Resource Control: VMM would have complete control of resources (via sensitive instructions)

Problems with Virtualization

- Many architectures (e.g.: Intel x86) do not satisfy this theorem
  - Some sensitive instructions are not privileged
    - Can be executed in user mode (maybe with different result)
    - E.g.: POPF instruction for setting interrupt flag
  - Cannot be easily virtualized. Why?

How to virtualize?

- Need:
  - Some way to ensure traps for sensitive instructions (for isolation)
  - Still allow innocuous instructions to execute natively (for efficiency)
Providing Virtualization

- Pure virtualization:
  - Guest OS is unmodified
  - Typically uses binary translation
  - E.g.: VMWare

- Paravirtualization:
  - Guest OS is modified to allow easier virtualization
  - VMM provides an altered instruction set
  - E.g.: Xen

Pure Virtualization

- Binary translation
  - Guest OS instructions are translated on-the-fly
  - Sensitive instructions substituted with instruction sequences causing trap to VMM
  - Common translations are cached
  - User-level instructions directly executed
    - These are innocuous
    - Provide efficiency

ParaVirtualization

- VMM exposes a modified instruction set
  - Virtualizable: Follows Popek/Goldberg Theorem
  - Guest OS modified for the new instruction set
    - Uses explicit hypercalls to access VMM
  - Can still provide unmodified application support
    - User instructions still executed directly

VMM Implementation

- Hypervisor:
  - Software layer on top of hardware
  - Controls all system resources
    - E.g.: VMWare ESX, Xen

- Host-based:
  - VMM uses services of host OS
  - VMs run as applications
    - E.g.: VMWare GSX
Resource Virtualization

- Need to virtualize:
  - CPU
  - Memory
  - I/O devices

Processor Virtualization

- VMM runs in privileged mode
- Guest OS and applications run in non-privileged mode
- VMM timeslices between different VMs
  - Similar to process timeslicing
  - Whole VM state preserved on switching
- Need to trap sensitive instructions
  - Dynamic binary translation: Patch sensitive instructions to trap into the VMM
  - Hypervisor: Allows explicit hypercalls

Interrupt Handling

- Interrupts managed by VMM
  - Guest OS disables interrupts => VMM queues up subsequent interrupts
  - Guest enables interrupts => deliver queued interrupts
  - VMM traps special instructions, e.g., POPF, so that VM sees disabled/enabled interrupts
  - Timer interrupts are handled by VMM

Processor Scheduling

- Same issues as regular CPU scheduling
- Similar scheduling algorithms used
  - E.g.: Xen uses Credit scheduler
    - Combination of weight and cap
- Multiprocessor scheduling:
  - Can use some form of space scheduling
  - E.g.: Cellular Disco divided processors into groups called Cells
Memory Virtualization

- Guest OS sees flat “physical” address space
- Page tables within guest OS:
  - Translate from virtual to physical addresses
- Second-level mapping:
  - Physical addresses to machine addresses
- VMM can swap a VM’s pages to disk transparently

Shadow page tables

- Translate from virtual addresses to machine addresses
- Used directly by MMU
- Unmapped pages transparently handled by VMM
- If guest tries to modify page table, control passed to VMM

Ballooning

- Method of reclaiming memory from a VM
- Balloon driver: Special driver inside guest OS
  - To reclaim memory: Balloon driver inflates
  - Guest OS selects pages to return
  - Returned pages passed to VMM

Translation Lookaside Buffers

- Cache recent page table lookups
- Must be cleared on VM switch
- Tagged architectures
  - Can have entries for multiple VMs
I/O Virtualization

- Problems:
  - Different devices have different characteristics
  - Large number of devices
- Techniques:
  - VMM virtualizes devices, translates into native device I/O
  - VMM gives access to I/O devices, but controls access

Host-based I/O

- VMM uses device drivers in the host OS
- Guest I/O commands converted to host I/O system calls
- E.g.: virtual disk is mapped to a file on host
  - Disk reads/writes become file reads/writes

Privileged VM

- Xen has a special VM (Domain 0)
  - Has real device drivers
- All other VMs (Dom-U) have virtual device driver
- Split driver: Divided between Dom-U and Dom0
- All I/O passes through Dom0

Virtual Network

- Each VM is assigned a separate IP address
- Communication among VMs: No physical networking required
- How do machines across the network talk to a specific VM?
  - Physical network device in promiscuous mode => Listens to all packets, picks up VM-specific packets
Hardware Virtualization

- Hardware support for VMM
  - E.g.: Intel VT, AMD Pacifica
- Provide special hardware modes, instructions, structures to allow running VMs easily
  - Eliminates/reduces need for software virtualization techniques
  - Improves performance

Architectural Support: Intel VT-x

- Two modes:
  - Root (VMM)
  - Non-root (Guest OS)
- Transition operations:
  - VM entry, VM exit
  - Transitions occur for sensitive instructions

Architectural Support: Intel VT-x

- VM control structure
  - Stores VM state (e.g., registers)
  - Similar to process control block
- Extended Page Tables
  - Hardware support for guest physical-to-machine page translation
  - No need to maintain shadow page tables