Virtual Machine Technology

Wei Wang
Why Virtual Machine for this course?

• To protect you from your own attacks
  – Your VM is your victim

• We use Linux as our main system
  – You probably use a different one

• Tool of the trade for computer security research
  – A controlled environment to study viruses
  – Widely adopted for virus detection
  – Isolation and encapsulation, more secure
  – Also used by virus to avoid detection and defy analysis
Outline

• History
• Key concepts
• Terminology
• Virtual machine basics
• Virtual machine taxonomy
History of Virtual Machines
1960s and 1970s – Hardware utilization

- IBM VM/370 operating system (1972)
  - Companies and other organizations required to run multiple OSes.
  - Hypervisor (control program) creates virtual machine environment
    - Each user has own virtual machine with guest OS, own address space, virtual devices, etc.

- Official Name: Virtual Machine Monitor or Hypervisor
1980s and 1990s – VMs decline in popularity

- Poor performance
- Client-server applications
- Inexpensive PCs
- Distributed computing
Late 1990s – Renewed Interest

• Improved CPU performance renewed interest in virtual machine
• Improve Infrastructure Utilization
  – Multiple VMs share hardware
  – improve utilization
  – Reduce installation and operational cost
  – Old business: web hosting
  – New business: Cloud computing
• Failover, Security and Disaster Protection
  – Isolated Environments
  – Encapsulation
• Flexibility
  – We are in an era of mixed environments: SW and HW, need the ability to go cross-platform
  – Easy for system testing
Key Concepts

aka. The Source of Virtual Machine’s Power
Key Concepts

- Levels of abstraction
- Well-defined interfaces
- Virtualization
Levels of Abstraction

- Allow implementation details at lower levels of a design to be ignored or simplified
- Arranged in a hierarchy
  - Lower levels – Hardware
    - Physical components with real properties
  - Higher levels – Software
    - Logical components
Levels of Abstraction: HW/SW Interface

Application

Algorithms

High Level Language (HLL)

Assembly Language

Operating System

Instruction Set Architecture (ISA)

Digital Logic

Electronics – (transistors etc.)
Levels of Abstraction: HW/SW Interface

- Application
- Algorithms
- High Level Language (HLL)
- Assembly Language
- Operating System
- Instruction Set Architecture (ISA)
- Digital Logic
- Electronics – (transistors etc.)

- Data Structures/Operations
- HLL Specifications
- Assembly Specifications
- System Calls / Linking Spec
- Instructions Set
- Digital Signals
- 0/1, Voltage etc.
Well-defined interfaces: Challenges

- Reduced interoperability
  - Processors support limited instruction sets
    - IA-32 vs. PowerPC
  - Different operating systems
    - Windows vs. Linux
  - Application binaries
    - Dependent on OS and instruction set

- Hardware resource dependency
  - OS
  - Applications
Virtualization

• Formally:
  – Virtualization involves the construction of an isomorphism that maps a virtual guest system to a real host. (Popek and Goldberg, 1974)
Virtualization cont’d

- Function $V$ maps guest state to the host state.
- For a sequence of operations $e$ that modify the guest state, a corresponding $e'$ in the host modify the host state.
Virtualization cont’d

● Abstractions and Well-defined Interfaces allows the introduction of a new virtualization layer

● Virtualization helps mitigate the reduced interoperability and hardware dependency problems of layers and well-defined interfaces
Virtualization cont’d

• Mapping of virtual resources or state to real resources on underlying machine
  – E.g., registers, memory, files

• Emulation of the virtual machine ABI or ISA
  – Use of real machine instructions and/or system calls to carry out actions specified by virtual machine instructions and/or system calls
Terminology
Terminology

- **Computer architecture**
  - Functionality and appearance of a computer or subsystem

- **Implementation**
  - Actual embodiment of an architecture
  - Architectures may have several implementations
    - E.g., high-performance or low-power
Terminology (cont’d)

- Implementation layers
  - Correspond to levels of abstraction in a computer system – each layers is essentially a state-machine.
ISA, ABI and API

- Instruction Set Architecture (ISA)
  - User ISA + Sys ISA
- Application Binary Interface (ABI)
  - Sys call + User ISA
- Application Programming Interface
  - Defined with high level language
  - Standard libraries
Terminology: Virtual Machine (VM)

• An emulation of a particular computer system
  – Any layer is a system itself
  – Almost any layer can be emulated: hardware, OS, libraries …
  – Remember each layer is a (state-) machine itself
  – VM provides an implementation for each (state-) machine (layer)
Terminology

- **Machine**
  - Matter of perspective
    - process: OS and underlying user-level hardware
    - OS: underlying hardware with ISA as the interface

- **Host**
  - Underlying platform

- **Guest**
  - Software that runs in the VM environment
Process VM vs System VM

- **Process VM**
  - Simulates ABI (sys call & user ISA) and API

- **System VM (VMM)**
  - Simulates full ISA

---

**Diagram**: A comparison between Process VM and System VM (VMM) shows the layers of abstraction between the guest operating system, the virtual machine, and the host hardware. The Process VM simulates an application interface (E.g., JVM), while the System VM (VMM) simulates the full ISA of the host system.
Process VM Example 1

- Multiprogramming
  - Each user process given the illusion of having complete machine to itself
  - OS supporting multiple user processes
Process VM Example 2

- Emulators and Diff-ISA Dynamic Binary Translator
  - Emulate one instruction set on hardware with another instruction set
  - Example: QEMU

Diagram:
- Guest
- Proc. VM
- HOST
- Application with ARM instructions
- QEMU: Translate from ARM to x86
- OS
- X86 Processor
Process VM Example 3

- Same-ISA Dynamic Binary Translator
  - Same ISA
  - Translate code block to another code block with the same functions but,
  - Performance: Dynamo
  - Security: Strata
  - Analysis: Pin

Diagram:

- Application with x86 instructions
  - Dynamo (Performance) or Strata (Security) or Pin (Analysis)

HOST

Proc. VM

Guest

X86 Processor

OS
Process VM Example 4

- High-level language VMs
  - Cross platform portability
  - Minimize hardware-specific and OS-specific features
  - Example: Java VM
Process VM Example 5

- Libraries Emulation
  - Emulation the APIs of another operating system
  - Supporting cross-OS portability
  - Same ISA
  - Example: Wine

Guest

Windows Applications

Proc. VM

Winehq: Provides Windows APIs

HOST

Linux

Hardware
Process VM Example 6

- VM-based Obfuscator
  - Code Obfuscation: make code hard to understand to defy analysis
  - Obfuscation techniques: junk instructions, complex logic, custom ISA, etc.
  - Commonly used by virus as anti-detection and anti-analysis techniques
  - Also employed to protect private software
  - Example: VM Protect, Themida

- Hardware Obfuscated Virus Code using special ISA
  - Windows
  - VM Obfuscator: convert special ISA to x86 ISA
Process VM vs System VM

- **Process VM**
  - Simulates ABI (sys call & user ISA) and API

- **System VM (VMM)**
  - Simulates full ISA
System VM Example 1

- **Classic system**
  - VM directly on hardware, e.g., VMWare ESX, Xen, KVM

- **Hosted VM**
  - VM on host operating system, e.g., VMWare Workstation, VirtualBox

![Diagram showing the layers of a classic system and a hosted VM.]
System VM Example 2

- Codesigned VMs (Hardware optimization)
  - No native applications. VM software is part of hardware implementation
    - AMD Processors
    - Modern Intel Processors

- [Diagram showing hardware and software components]
  - x86 Applications
  - x86 OS
  - x86 to Berkeley RISC decoder
  - AMD K5 Processor: Berkeley RISC
System VM Example 3

- Whole System VMs (Emulation)
  - Complete software system (OS and applications) supported on a host system running a different ISA and OS
  - VirtualPC (Windows on Mac)
Full Virtualization vs Para-Virtualization

- **Full Virtualization**
  - Provide same functions as real hardware
  - Allows unmodified guest OS
  - Example: VMware

- **Para-Virtualization**
  - Provide similar, but no identical functions as real hardware
  - May need to modified guest OS
  - Example: Xen (both full and para)
Taxonomy

- Process vs. system
- Same ISA vs. different ISA