Goals for Today

• **Learning Objective:**
  • Define a taxonomy for virtualization architectures

• **Announcements, etc:**
  • C4 Submission for 3/9, 3/15 open (sorry for the delay)
  • Request for “Informal Early Feedback” forthcoming, probably on Monday
  • Midterm debrief forthcoming, probably on Wednesday

**Reminder:** Please put away devices at the start of class
CS 423
Operating System Design: Virtual Machines

Professor Adam Bates
Spring 2017
Virtual Machines

• What is a virtual machine?
  • Examples?
• Benefits?
• Creation of an isomorphism that maps a virtual guest system to a real host:
  – Maps guest state $S$ to host state $V(S)$
  – For any sequence of operations on the guest that changes guest state $S_1$ to $S_2$, there is a sequence of operations on the host that maps state $V(S_1)$ to $V(S_2)$
Important Interfaces

- **Application programmer interface (API):**
  - High-level language library such as *c* **lib**

- **Application binary interface (ABI):**
  - User instructions (User ISA)
  - System calls

- **Hardware-software interface:**
  - Instruction set architecture (ISA)
What’s a machine?

• Machine is an entity that provides an interface
  – Language view:
    • Machine = Entity that provides the API
  – Process view:
    • Machine = Entity that provides the ABI
  – Operating system view:
    • Machine = Entity that provides the ISA
Virtual machine is an entity that emulates a guest interface on top of a host machine

- Language view:
  - Virtual machine = Entity that emulates an API (e.g., JAVA) on top of another
  - Virtualizing software = compiler/interpreter

- Process view:
  - Machine = Entity that emulates an ABI on top of another
  - Virtualizing software = runtime

- Operating system view:
  - Machine = Entity that emulates an ISA
  - Virtualizing software = virtual machine monitor (VMM)
Purpose of a VM

• Emulation
  – Create the illusion of having one type of machine on top of another

• Replication (Multiplexing)
  – Create the illusion of multiple independent smaller guest machines on top of one host machine (e.g., for security/isolation, or scalability/sharing)

• Optimization
  – Optimize a generic guest interface for one type of host
• Emulate (ISA/ABI/API) for purposes of (Emulation/Replication/Optimization) on top of (the same/different) one.
Types of VMs

• Emulate (ISA/ABI/API) for purposes of (Emulation/Replication/Optimization) on top of (the same/different) one.
  – Process/language virtual machines (emulate ABI/API)
  – System virtual machines (emulate ISA)
Types of VMs

• Emulate (ISA/ABI/API) for purposes of (Emulation/Replication/Optimization) on top of (the same/different) one.
  – Process/language virtual machines (emulate ABI/API)
  – System virtual machines (emulate ISA)
Ex 1: Multiprogramming

- Emulate what interface?
- For what purpose?
- On top of what?
Ex1: Emulation

• Emulate one ABI on top of another
  – Emulate a Intel IA-32 running Windows on top of PowerPC running MacOS (i.e., run a process compiled for IA-32/Windows on PowerPC/MacOS)
  • Interpreters: Pick one guest instruction at a time, update (simulated) host state using a set of host instructions
  • Binary translation: Do the translation in one step, not one line at a time. Run the translated binary
• Create a simulator data structure to represent:
  – Guest memory
    • Guest stack
    • Guest heap
  – Guest registers

• Inspect each binary instruction (machine instruction or system call)
  – Update the data structures to reflect the effect of the instruction
Ex2: Binary Optimization

• Emulate one ABI on top of itself for purposes of optimization
  – Run the process binary, collect profiling data, then implement it more efficiently on top of the same machine/OS interface.
Ex3: Language VMs

• Emulate one API on top of a set of different ABIs
  – Compile guest API to intermediate form (e.g., JAVA source to JAVA bytecode)
  – Interpret the bytecode on top of different host ABIs

• Examples:
  – JAVA
  – Microsoft Common Language Infrastructure (CLI), the foundation of .NET
Types of VMs

• Emulate (ISA/ABI/API) for purposes of (Emulation/Replication/Optimization) on top of (the same/different) one.
  – Process/language virtual machines (emulate ABI/API)
  – System virtual machines (emulate ISA)
Types of VMs

• Emulate (ISA/ABI/API) for purposes of (Emulation/Replication/Optimization) on top of (the same/different) one.
  – Process/language virtual machines (emulate ABI/API)
  – System virtual machines (emulate ISA)
System VMs

- Implement VMM (ISA emulation) on bare hardware
  - Efficient
  - Must wipe out current operating system to install
  - Must support drivers for VMM

- Implement VMM on top of a host OS (Hosted VM)
  - Less efficient
  - Easy to install on top of host OS
  - Leverages host OS drivers
• Implement VMM (ISA emulation) on bare hardware
  – Efficient
  – Must wipe out current operating system to install
  – Must support drivers for VMM

• Implement VMM on top of a host OS (Hosted VM)
  – Less efficient
  – Easy to install on top of host OS
  – Leverages host OS drivers
Whole System VMs

• Emulate one ISA on top of another
  – Typically runs on top of host OS (e.g., install Windows compiled for IA-32 on top of MacOS running on PowerPC)
  – Note: this is different from a process virtual machine that emulates the Windows interface and user IA-32 instructions on top of MacOS running on PowerPC
Taxonomy

• Process VMs
  – Same ISA
    • Multiprogrammed systems
  – Different ISA
    • Dynamic translators (Java), emulators

• System VMs
  – Same ISA
    • Classic VMs
    • Hosted VMs (KVM, VirtualBox, other VMWare)
  – Different ISA
    • Whole system VMs
• Problem: Emulate guest ISA on host ISA
• Problem: Emulate guest ISA on host ISA
• Solution: Basic Interpretation

inst = code (PC)
opcode = extract_opcode (inst)
switch (opcode) {
    case opcode1 : call emulate_opcode1 ()
    case opcode2 : call emulate_opcode2 ()
    ...
}

Emulation
**Emulation**

- **Problem:** Emulate guest ISA on host ISA
- **Solution:** Basic Interpretation

new

\[
\text{inst} = \text{code (PC)}
\]

\[
\text{opcode} = \text{extract_opcode (inst)}
\]

\[
\text{routineCase} = \text{dispatch (opcode)}
\]

\[
\text{jump routineCase}
\]

\[
\ldots
\]

\[
\text{routineCase} \quad \text{call routine_address}
\]

\[
\text{jump new}
\]
Threaded Implementation

[ body of emulate_opcode1 ]
inst = code (PC)
opcode = extract_opcode (inst)
routine_address = dispatch (opcode)
jump routine_address

[ body of emulate_opcode2]
inst = code (PC)
opcode = extract_opcode (inst)
routine_address = dispatch (opcode)
jump routine_address