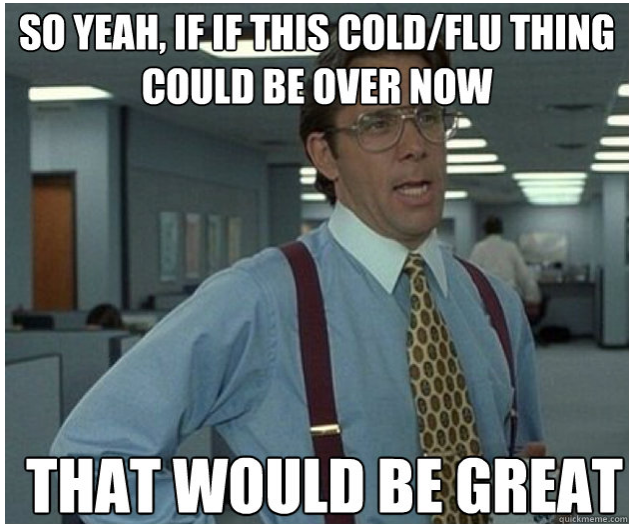
Goals for Today

- Learning Objective:
  - Define a taxonomy for virtualization architectures
- Announcements, etc:
  - Midterm debrief forthcoming (please do not post to piazza yet)
  - C4 summaries due Friday (as always)
  - MP2 due March 18th (UTC-11)



Sorry about Friday! :)

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?

Virtualization

- 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 S1 to S2, there is a sequence of operations on the host that maps state V(S1) to V(S2)

Important Interfaces

- Application programmer interface (API):
  - High-level language library such as `clib`
- 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
  - From the perspective of a language…
    - Machine = Entity that provides the API
  - From the perspective of a process…
    - Machine = Entity that provides the ABI
  - From the perspective of an operating system…
    - Machine = Entity that provides the ISA

What's a virtual machine?

- 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

Types of VMs

- 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)

Ex1: 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

Writing an Emulator

- 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

System VMs

- Implement VMM (ISA emulation) on bare hardware

  TYPE ONE HYPERVISOR

  – Efficient
  – Must wipe out current operating system to install
  – Must support drivers for VMM
- Implement VMM on top of a host OS (Hosted VM)

  TYPE TWO HYPERVISOR

  – Less efficient
  – Easy to install on top of host OS
  – Leverages host OS drivers

What is Xen?

What is VirtualBox?

What is KVM/Qemu?

Taxonomy

- Language VMs
  - Emulate same API as host (e.g., application profiling?)
  - Emulate different API than host (e.g., Java API)
- Process VMs
  - Emulate same ABI as host (e.g., multiprogramming)
  - Emulate different ABI than host (e.g., Java VM, MAME)
- System VMs
  - Emulate same ISA as host (e.g., KVM, VBox, Xen)
  - Emulate different ISA than host (e.g., MULTICS simulator)

Point of Clarification

- Emulation: General technique for performing any kind of virtualization (API/ABI/ISA)
- Not to be confused with *Emulator* in the colloquial sense (e.g., Video Game Emulator), which often refers to ABI emulation.

Writing an Emulator

- Problem: Emulate guest ISA on host ISA

Writing an Emulator

- Problem: Emulate guest ISA on host ISA
- 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

Emulation

- Problem: Emulate guest ISA on host ISA
- Solution: Basic Interpretation, switch on opcode

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

Emulation

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

```
new            inst = code (PC)
               opcode = extract_opcode (inst)
               routineCase = dispatch (opcode)
               jump routineCase
               …
routineCase                    call routine_address
                               jump new
```

Threaded Interpretation…

```
[ 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
```