Virtualize the machine?

• What is the machine?
• Machine is defined by an interface
• 3 interfaces that can be virtualized:
1. Instruction Set Architecture (ISA)
2. Application Binary Interface (ABI)
3. Application Programming Interface (API)
Interface 1: ISA

- Virtualize a complete machine, running an OS, supporting multiple processes = System VM
Interface 2: ABI

System calls

Operating System

Libraries

Application Programs

Hardware

ABI = System calls + user ISA

- Virtualize the environment of a single process (binary)
- = Process VM
Interface 3: API

- Virtualization provides same interface at programming language (source) level.

API = Libraries + user ISA
Example Virtualizing ISA

- Support a machine's complete ISA
- VM/370
- Xen*
- KVM*
Example Virtualizing ABI

- Run **binaries unmodified** on different platform
- Common: Run **Win32-x86** binaries on:
  - Unix-x86 (Wine)
  - Solaris-SPARC (Sun WABI)
  - Win32-Alpha (Digital FX!32)
Example Virtualizing API

- **Recompile** applications from source
- Runs on any platform with same API
  - E.g. Linux-x86 and Linux-ARM
- (Assuming platform-independent code)
VM Implementations

(a) System VM

(b) Process VM
What ISA? Same or different

- **Same**: Run Win32-x86 on Linux-x86
- **Diff**: Run Linux-ARM on Win32-x86
Taxonomy

Process VMs
- Same ISA
- Different ISA
  - Multiprogrammed
  - Systems
  - Emulators/
    Translators
  - High-level
    Language VMs

System VMs
- Same ISA
- Different ISA
  - Classic-System
    VMs
  - Hosted VMs
  - Whole-System
    VMs
  - Codesigned
    VMs
Example: Windows Multiprogramming

One CPU, illusion of processes running in parallel
Example: Android

High-level Language VM (HLL-VM)

Different ISA: Java vs. ARM
Example: Android Emulation
Example: Android Emulation on Xen

Java
Java VM
Linux OS
ARM VM Runtime
Win32 Native
Linux Native
Linux Native
Windows OS
Linux OS
Xen Hypervisor
X86 Hardware
Xen domain
= Hosted VM

Security & Network Engineering
Example: VMWare

- Applications
- Guest OS
- VMMonitor
- VM Driver
- Host OS
- X86 Hardware

= Hosted VM
Example: AS/400

- Application Programs
- OS
- Virtual Machine Monitor
- Hardware

Higher level ISA

Allow evolution of hardware ISA

Source ISA
Target ISA

= Co-designed VM
Taxonomy Examples

Process VMs

- Same ISA
  - UNIX
  - Multiprogrammed Systems

- Different ISA
  - FX!32, WABI
  - Emulators/Translators
  - Java VM, MS CLR
  - High-level Language VMs

System VMs

- Same ISA
  - VM/370
  - Classic-System VMs
  - VMware, Xen, Docker*
  - Hosted VMs

- Different ISA
  - ARM VM runtime
  - Whole-System VMs
  - AS/400
  - Codesigned VMs
Implementing Virtual Machines with Different ISAs
VM implementation: Emulation

- **Emulation** = implement interface of one system on another system with a different interface
- Example: x86 instruction
  - `addl %edx, 4(%eax)`
- Emulated via PowerPC instructions:
  - `lwz r4, 0(r1)`
  - `addi r5, r4, 4`
  - `lwzx r5, r2, r5`
  - `lwz r4, 12(r1)`
  - `add r5, r4, r5`
  - `stw r5, 12(r1)`
The slides on how to handle different ISAs are not part of the exam.
Executing instruction $e$ changes state of Source machine from $S_a$ to $S_b$. 
Emulation Model

- Real machine has corresponding state $S'_a$
- Performs $e$ by means of instruction(s) $e'$

Source

Target
Recall: Registers + Memory

CPU

MEM

Cache

r0
r1
r2
rN
PC
Example: Change registers

Source

Target
Example: Change registers

CPU state of Source may be kept in Target memory, not registers!
Emulation Performance + Methods

• Can be slow because of mapping Source to Target!
• Range of emulation methods:
  - Interpretation
  - Binary translation
• Interpretation:
  - Decode a single source instruction and execute using target instructions
• Binary translation:
  - Translate a block of source instructions once and reuse
Interpretation

• Source instruction is a series of bytes
• Different formats
  - **RISC**: clean and simple
  - **CISC**: complex with legacy
  - Non-hardware: *Java* bytecodes
• Complexity of format influences interpretation performance!
# Example Formats

- **x86:**

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Opcode</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4 bytes</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>0,1,2,4</td>
<td>0,1,2,4 bytes</td>
<td></td>
</tr>
</tbody>
</table>

  Software developer's manual: 3796 pages!

- **Java:**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>Index1</td>
</tr>
<tr>
<td>Opcode</td>
<td>Data1</td>
</tr>
</tbody>
</table>

  Java VM Specification: 604 pages
x86 Format

- Prefixes: Repetition for strings, overrides for address and operand sizes
- ModR/M: addressing mode and which register
- SIB: base register, index register, index scale
- Displacement: offset to be added to address
- Immediate: variable length operand

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Opcode</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4 bytes</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>0,1,2,4</td>
<td>0,1,2,4 bytes</td>
<td>0,1,2,4 bytes</td>
</tr>
</tbody>
</table>
Binary Translation

- Per-instruction interpretation slow
  - Especially when complex
- Alternative:
  - Translate blocks of source instructions once
  - Reuse
- cf. Just-in-Time compilers
- Hard
Performance Tradeoff

- $E(n) =$ time needed to execute an instruction $n$ times
- Formula: $E(n) = S + n*T$
- $S =$ startup time
- $T =$ time required per emulation of the instruction

- Interpretation:
  - $S$ low, $T$ high
- Binary translation
  - $S$ high, $T$ low
In practice: automatic profiling, often used code is binary translated
OS Emulation

- For Process VMs have to emulate whole ABI
  - User ISA
  - System call interface
- System call instructions (e.g. `sysenter`) emulated
- Translate call from Source OS to Target OS
- **Same OS**: straightforward to hard
- **Diff OS**: straightforward to impossible
  - No guarantees that Target OS has same features as Source!
  - E.g. `fsync()`
Same ISA VMs

- Emulation needed for different ISA VMs
- For same ISA: Theoretically, source instructions can be executed directly on target
- Fastest
- Does this work for all instructions?
Same ISA VMs (cont'd)

- No: System ISA instructions need to be controlled
- When? System VMs with a guest OS
- How?
  - Guest OS runs in CPU User Mode
  - System ISA instructions called in User Mode activate Kernel Mode i.e., cause a Trap
- VMM in Kernel Mode then emulates system instruction
Example: App Scheduling

- User Mode
- Kernel Mode
- Operating System
- Applications
- Hardware
- Run
- Interrupts, Traps, faults
- Privileged instructions
Example: App Scheduling

1. Set interval timer
2. Run app in User Mode
3. Interrupt

Hardware

Operating System

User Mode

Kernel Mode

Applications
Example: App Scheduling

1. Set interval timer $t$

2. Set interval timer $t'$

3. Run OS in User Mode

4. Run app in User Mode

5. Interrupt

Applications

Operating System

Kernel Mode

User Mode

Hardware
Example: App Scheduling

1. Set interval timer t

Virtual Machine Monitor

2. Set interval timer t'

Hardware

3. Run OS in User Mode

Operating System

4. Run app in User Mode

Applications

5. Interrupt

User Mode

Kernel Mode

t = requested by OS

t' = granted by VMM for fair scheduling of multiple VMs
Example: App Scheduling

1. Set interval timer $t$

2. Set interval timer $t'$

3. Run OS in User Mode

4. Run app in User Mode

5. Interrupt

- VMM schedules VMs
- Guest OS schedules Apps
x86 Same ISA Problems (1/3)

- Normal: OS + Apps
  - Ring 0: Kernel Mode

- VM: OS no longer in kernel mode!
  - Ring 3: User Mode
  - OS + Apps
  - VMM
x86 Same ISA Problems (2/3)

• In x86, not all system instructions in User Mode activate Kernel Mode!

• → When Guest OS runs in User Mode, not all system instruction calls observed by VMM

• Old solution: patch all binary code!
  - Replace these critical instructions with explicit traps to Kernel Mode
x86 Same ISA Problems (3/3)

- New solution: **Intel VT-x**
- Allows Guest OS to run in Kernel Mode (Ring 0)
- Shared resources still controlled by VMM
- Using extra mode: VMX
  - VMX Root for VMM
  - VMX Non-root for Guest OS
- “Ring -1”
- Also hardware support for VM context switch
Native and Hosted VMs

(a) Native VM
(b) Dual-Mode Hosted VM

User Mode
Kernel Mode
VMWare Workstation

• Install on top of existing host OS
  - Easy to use
  - Can use myriad of device drivers available in host OS
VMWare Architecture

- Applications
- Guest OS
- VMMonitor
- VM Driver
- Host OS
- X86 Hardware
VMWare Workstation

- Adds 3 components
- VMMonitor
  - VMM in Kernel Mode, alongside Host OS
- VMApp
  - User Process for translating VMM requests into system calls to host OS
- VMDriver
  - Extension of the host OS
  - Support switching between Host OS and VMM
  - Enable VMM↔VMApp communication
VMWare I/O

Direct support, e.g. disk access via IDE
Use host support, e.g. CD, sound, serial port
VMWare: New Capabilities

- Applications
- Guest OS
- VMMonitor
- Device Driver
- Host OS
- X86 Hardware
- VMApp
- VM Driver
- Device Driver
- COW
- Golden Image
Operating System Support for Virtualization
Native, Hosted, Paravirtualized VMs

Modify the Guest OS!
Paravirtualization

• System VMs can be faster when **Guest OS** can be **modified** for virtualization
• Showcased in Xen Project
• Modified
  - Linux
  - Windows XP
• Near native performance!
Paravirtualizing I/O

Operating System

- Real Device Driver
- Reverse Device Driver
- Network Handler

Hardware

Operating System

- PV Device Driver
- Network Handler

Hardware

I/O instructions

Hypercalls
Xen Evolution

• Problems:
  - only open-source OSes can be modified
  - Xen implementation tricks not on x86-64
• New approach: Start from Full virtualization with Hardware Support (e.g. VT-x)
• Apply Paravirtualization in areas where speed can be gained:
  1. Disk and network I/O
  2. Interrupts and timers
  3. Emulated motherboard, legacy boot
  4. Privileged instructions, page tables
Xen Mode: HVM

P = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized

http://wiki.xen.org/wiki/Virtualization_Spectrum
**Xen Mode: PV**

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Privileged Instructions, Page Tables, Disk and Network, Emulated Motherboard, Legacy Boot, Interrupts &amp; Timers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>VS</td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>VH</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized
## Xen Mode: HVM + PV Drivers

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Privileged Instructions, Page Tables</th>
<th>Emulated Motherboard, Legacy Boot</th>
<th>Disk and Network</th>
<th>Interrupts &amp; Timers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>PV Drivers</td>
<td>P</td>
<td>VS</td>
<td>VS</td>
<td>VS</td>
</tr>
</tbody>
</table>

P = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized

### Security & Network Engineering
# Xen Mode: PVHVM Drivers

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Privileged Instructions, Page Tables</th>
<th>Emulated Motherboard, Legacy Boot</th>
<th>Disk and Network</th>
<th>Interrupts &amp; Timers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td>VS, VS, VS, VH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>P, VS, VS, VH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVHVM</td>
<td>HVM</td>
<td>P, P, VS, VH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PV</td>
<td>PV</td>
<td>P</td>
<td></td>
<td></td>
<td>P, P, P, P, P</td>
<td></td>
</tr>
</tbody>
</table>
## Xen Mode: PVH

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Disk and Network</th>
<th>Emulated Motherboard</th>
<th>Privileged Instructions, Page Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>PV Drivers</td>
<td>P</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>PVHVM</td>
<td>HVM</td>
<td>PVHVM Drivers</td>
<td>P</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>PVH</td>
<td>PV</td>
<td>pvh=1</td>
<td>P</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td>PV</td>
<td>PV</td>
<td></td>
<td>P</td>
<td>VS</td>
<td>VS</td>
</tr>
</tbody>
</table>

P = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized
## KVM

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Mode</th>
<th>With</th>
<th>Poor Performance</th>
<th>Scope for Improvement</th>
<th>Optimal Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVM / Fully Virtualized</td>
<td>HVM</td>
<td></td>
<td>VS</td>
<td>VS</td>
<td>VH</td>
</tr>
<tr>
<td>HVM + PV drivers</td>
<td>HVM</td>
<td>PV Drivers</td>
<td>P</td>
<td>VS</td>
<td>VS</td>
</tr>
<tr>
<td><strong>KVM</strong></td>
<td>HVM</td>
<td></td>
<td>P</td>
<td>VS</td>
<td>VH</td>
</tr>
<tr>
<td>PVHVM</td>
<td>HVM</td>
<td>PVHVM Drivers</td>
<td>P</td>
<td>P</td>
<td>VS</td>
</tr>
<tr>
<td>PVH</td>
<td>PV</td>
<td>pvh=1</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>PV</td>
<td>PV</td>
<td></td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>