Virtual Machines
Part 2: starting 19 years ago
## How They’re Different

<table>
<thead>
<tr>
<th>IBM 360</th>
<th>Intel x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two execution modes</td>
<td></td>
</tr>
<tr>
<td>– supervisor and problem (user)</td>
<td></td>
</tr>
<tr>
<td>– all sensitive instructions are privileged instructions</td>
<td></td>
</tr>
<tr>
<td>Memory is protectable: 2k-byte granularity</td>
<td></td>
</tr>
<tr>
<td>All interrupt vectors and the clock are in first 512 bytes of memory</td>
<td></td>
</tr>
<tr>
<td>I/O done via channel programs in memory, initiated with privileged instructions</td>
<td></td>
</tr>
<tr>
<td>Dynamic address translation (virtual memory) added for Model 67</td>
<td></td>
</tr>
<tr>
<td>Four execution modes</td>
<td></td>
</tr>
<tr>
<td>– rings 0 through 3</td>
<td></td>
</tr>
<tr>
<td>– not all sensitive instructions are privileged instructions</td>
<td></td>
</tr>
<tr>
<td>Memory is protectable: segment system + virtual memory</td>
<td></td>
</tr>
<tr>
<td>Special register points to interrupt vector</td>
<td></td>
</tr>
<tr>
<td>I/O done via memory-mapped registers</td>
<td></td>
</tr>
<tr>
<td>Virtual memory is standard</td>
<td></td>
</tr>
</tbody>
</table>
Rings

- apps
- kernel
A Sensitive x86 Instruction

• popf
  – pops word off stack, setting processor flags according to word’s content
    - sets all flags if in ring 0
      • including interrupt-disable flag
    - just some of them if in other rings
      • ignores interrupt-disable flag
What to Do?

• Binary rewriting
  – rewrite kernel binaries of guest OSes
    - replace sensitive instructions with hypercalls
    - do so dynamically
• Hardware virtualization
  – fix the hardware so it’s virtualizable
• Paravirtualization
  – virtual machine differs from real machine
    - provides more convenient interfaces for virtualization
    - hypervisor interface between virtual and real machines
    - guest OS source code is modified
Binary Rewriting

• Privilege-mode code run via binary translator
  – replaces sensitive instructions with hypercalls
  – translated code is cached
    - usually translated just once
  – VMWare
  – U.S. patent 6,397,242
  – more recently
    - KVM/QEMU
Fixing the Hardware

• Intel Vanderpool technology: VT-x
  – also known as VMX (virtual-machine extensions)
  – new processor mode
    - “ring -1”
      • root mode
      • other modes are non-root
  – certain events in non-root mode cause VM-exit to root mode
    - essentially a hypercall
    - data structure in root mode specifies which events cause VM-exits
  – non-VMM OSes must be written not to use root mode!
Virtual-Machine State

Real Execution

VM-exit

VMM

Machine state

Virtualized state
VM Control State

Virtual Machine

Guest State

Host State

Control

virtual machine control structure (VMCS)

Root Mode
VM Control State
VM-Exit

Virtual Machine
Guest State
Host State
Control
virtual machine control structure (VMCS)

Root Mode
VM Control State
VM-Entry

Virtual Machine

Guest State
Host State
Control

virtual machine control structure (VMCS)

Root Mode
Examples

• **mov instruction**
  - `mov $2, %rax`
    - no VM-exit
  - `mov $2, %CR3`
    - VM-exit

• **interrupts**
  - interrupt occurs
    - VM-exit
  - `pop %flags` in ring 0
    - affects interrupt-disable flag on guest, no effect on real machine
    - no VM-exit
  - set interrupt vector
    - VM-exit
Quiz

We’ve implemented recursive virtualization: VMM<sub>i</sub> runs on a VM supported by VMM<sub>i-1</sub>, which runs on a VM supported by VMM<sub>i-2</sub>, …, which runs on a VM supported by VMM<sub>0</sub>, which runs on the real hardware. A VM-Exit takes place on a VM running on VMM<sub>i</sub>.

a) It’s handled first on VMM<sub>i</sub>, which then VM-Exits to VMM<sub>i-1</sub>, which the VM-Exits to VMM<sub>i-2</sub>, …, which VM-exits to VMM<sub>0</sub>.

b) It’s handled first on VMM<sub>0</sub>, is then handled on VMM<sub>1</sub>, …, and finally on VMM<sub>i</sub>. 
I/O Virtualization

• Channel programs were generic
• I/O via memory-mapped registers is not
  – lots and lots and lots of device drivers
  – must VMM handle all of them?
Real-Machine OS Structure

OS

Device drivers

Devices

Processor(s)
On a Virtual Machine ...

Device drivers

Virtual devices

Virtual processor(s)

OS

Process

Device drivers

Virtual devices

Virtual processor(s)

OS

Process

Device drivers

Virtual devices

Virtual processor(s)

VMM

Devices

Processor(s)
VMware Workstation

- Process
  - Guest OS
    - Device drivers
    - Virtual devices
    - Virtual processor(s)
  - VMDriver
    - Device drivers
  - Host OS
    - Devices
    - Processor(s)
  - VMAApp
  - Process
KVM/QEMU

- **KVM**
  - kernel virtual machine monitor for Linux
  - uses VMX technology (or AMD equivalent)

- **QEMU**
  - generic and open source machine emulator and virtualizer
  - does binary rewriting and caching as does VMware
  - emulates I/O devices as well

- **KVM/QEMU**
  - code executes natively until VM-exit
  - user-space QEMU code does I/O emulation
Paravirtualization

- Sensitive instructions replaced with hypervisor calls
  - traps to VMM
- Virtual machine provides higher-level device interface
  - guest machine has no device drivers
Additional Applications

• Sandboxing
  – isolate web servers
  – isolate device drivers

• Migration
  – VM not tied to particular hardware
  – easy to move from one (real) platform to another
Process Migration
Approaches: Before
Approaches: After

Communication

System calls

local stand-in
Virtual-Machine Migration

• Virtual machines are isolated
  – by definition!
• State is well defined
  – thus easy to identify and move
  – possible exception of virtual memory
Transferring Virtual Memory

• Eager
  – all
  – dirty
    - (clean pages come from common source)

• Lazy
  – copy on reference

• Straightforward
  – flush everything to file system on source, then access file system on target

• Weird
  – precopy
Eager–Dirty

- Freeze process on source
- Transfer all dirty pages to target
- Resume process on target
Precopy

- While process still running on source
  - transfer everything to target (eager–dirty)
- While more than x pages dirty on source
  - transfer newly dirtied pages to target
- Freeze process on source
- Transfer remaining dirty pages to target
- Resume process on target