Virtual Machines
Part 2: starting 20 years ago
## How They’re Different

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

- apps
- kernel
- 2
- 1
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

Virtualized state

Machine state

VMM

VM-exit
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

Root Mode

virtual machine control structure (VMCS)
Examples

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

• interrupts
  - interrupt occurs
    - VM-exit
  - popf 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: $VMM_i$ runs on a VM supported by $VMM_{i-1}$, which runs on a VM supported by $VMM_{i-2}$, …, which runs on a VM supported by $VMM_0$, which runs on the real hardware. A VM-Exit takes place on a VM running on $VMM_i$.

a) It’s handled first on $VMM_i$, which then VM-Exits to $VMM_{i-1}$, which the VM-Exits to $VMM_{i-2}$, …, which VM-exits to $VMM_0$.

b) It’s handled first on $VMM_0$, is then handled on $VMM_1$, …, and finally on $VMM_i$. 
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

- Process
- Process
- Process
- Process
- Process

OS

Device drivers

Devices

Processor(s)
On a Virtual Machine ...

Diagram:

- **Process** connected to **OS**
- **Device drivers** connected to **Virtual devices**
- **Virtual processor(s)** connected to **OS**
- **VMM** connects to **Devices** and **Processor(s)**
VMware Workstation

Guest OS

- Virtual devices
- Virtual processor(s)

Guest OS

- Virtual devices
- Virtual processor(s)

VMDriver

- Device drivers

Host OS

- Device drivers

VMApp

- Process

- Process

Processes

- Process

- Process

Devices

- Device drivers

Processor(s)
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
Xen with Isolated Driver

Domain 0
- Process
- net device driver
- net back end

Domain U1
- Process
- net device driver
- net front end
- block front end
- block back end

Domain U2
- block back end
- disk device driver

OS

VMM

Hardware
Process Migration
Approaches: Before
Approaches: After

Communication

System calls

local standby

Operating Systems In Depth
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