We have seen how the OS virtualizes subsystems: CPU, Memory, IO. To give applications illusions about owning the system, the OS knows all.

What about:
- Virtualizing the whole system
- Giving OSes the illusion of a system that isn’t real
- The OS doesn’t know all

Why do this?
- To enable multiple OSes to run "at the same time" on the same hardware, sharing resources without harming anyone

The Idea

Virtual Machines Monitor (VMM)
- Sits between multiples OSes and hardware (or a host OS)
- Presents a hardware interface to the OSes above
- Gives the illusion to each OS above that it controls the whole machine. Actually, the VMM does, and each OS sees a virtual machine
- The VMs (and OSes) share the actual hardware resources
- Manages (multiplexes) resources among several virtual machines (VMs)
- Isolates VMs from each other
- Similar to what an OS does: abstraction, resource mgmt
- a.k.a. Hypervisor
History

- Have been around since 1960s on mainframes
  - Used to run apps on different OSes on same (very expensive) mainframe
  - Good example – VM/370
- Computers became cheaper, people lost interest
- Have resurfaced on commodity platforms
  - Server Consolidation: save space, power; data centers
  - High-Performance Compute Clusters: run different OSes
  - Managed desktop/thin-client
    - Save desktop in a VM and bring it with you on a USB drive
    - Software development/kernel hacking
      - Crash your development kernel but don’t disable whole machine

Goals

- Manageability
  - Creation, maintenance, administration, provisioning, etc.
- Performance
  - Overhead of virtualization should be small
- Isolation, like separate physical machines
  - Activity of one VM should not impact other active VMs
  - Data of one VM is inaccessible by another
- Scalability
  - Minimize cost per VM; run more VMs on hardware
- Reliability
  - Same goals as for many subsystems

VMM Types

Type I VMM

<table>
<thead>
<tr>
<th>VM1</th>
<th>VM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Applications</td>
</tr>
<tr>
<td>Guest Operating System</td>
<td>Guest Operating System</td>
</tr>
<tr>
<td>Type-I Hypervisor</td>
<td>Type-II Hypervisor</td>
</tr>
<tr>
<td>Hardware (CPU, Memory, Interrupts)</td>
<td>Hardware (CPU, Memory, Interrupts)</td>
</tr>
</tbody>
</table>

Type II VMM

<table>
<thead>
<tr>
<th>VM</th>
<th>Guest Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Type-II Hypervisor</td>
</tr>
<tr>
<td>Native Operating System</td>
<td></td>
</tr>
</tbody>
</table>

VMM Types

<table>
<thead>
<tr>
<th>Type I VMM</th>
<th>Type II VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest application</td>
<td>Guest application</td>
</tr>
<tr>
<td>Guest application</td>
<td>Guest application</td>
</tr>
<tr>
<td>Guest application</td>
<td>Guest application</td>
</tr>
<tr>
<td>guest operating system</td>
<td>virtual machine monitor (VMM)</td>
</tr>
<tr>
<td>virtual machine monitor (VMM)</td>
<td>host operating system</td>
</tr>
<tr>
<td>host hardware</td>
<td>host hardware</td>
</tr>
</tbody>
</table>
Virtualization Styles

- **Fully virtualizing VMM**
  - Virtual machine looks exactly like a (some) physical machine
  - Not necessarily exactly like the underlying hardware itself
  - Run guest OS unchanged
  - VMM is transparent to the OS

- **Para-virtualizing VMM**
  - Guest OS is changed to cooperate with VMM
  - Sacrifice transparency for better performance
  - E.g. VMM can provide idealized view of some hardware
  - E.g. VMM can provide "hypervisor API" so guest can perform certain functions, e.g. with optimizations for performance

VMM Classification

| Type I                | Type II
|-----------------------|------------------------
| Fully-virtualized     | VMware ESX (standalone, Windows Hyper V (with OS)) |
| VMware Workstation, Fusion, Parallels Desktop, VirtualBox, Virtual PC |
| Para-virtualized      | Xen                     |
|                       | User Mode Linux         |

VMM Implementation

- Should efficiently virtualize the hardware
- Provide illusion of multiple machines
- Retain control of the physical machine

Subsystems
- Processor Virtualization
- I/O virtualization
- Memory Virtualization
Processor Virtualization

Popek and Goldberg (1974)

- Sensitive instructions: only executed in kernel mode
- Privileged instructions: trap when run in user mode
- CPU architecture is virtualizable only if sensitive instructions are subset of privileged instructions
  - i.e. sensitive instructions will always trap if run in user mode
- When guest OS, which runs in user mode, runs a sensitive instruction, must trap to VMM so it maintains control

Example: System Call (Type 1 Hypervisor)

1. System call: Trap to OS
2. Process trapped: call OS trap handler (at reduced privilege)
3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap
4. OS tried to return from trap; do real return-from-trap
5. Resume execution (@PC after trap)

What if not fully virtualizable?

- x86 architecture was not fully virtualizable
  - Certain privileged instructions behave differently when run in unprivileged mode, e.g. do nothing (e.g. POPF)
  - Certain unprivileged instructions can access privileged state (so guest OS would be able to see that it’s not running in kernel mode)
- Techniques to address
  - Replace non-virtualizable instructions with easily virtualized ones statically (Paravirtualization)
  - Perform Binary Translation (Full Virtualization)

I/O Virtualization

- Issue: lots of I/O devices
- Problem: Writing device drivers for all I/O device in the VMM layer is not a feasible option
- Insight: Device driver already written for popular Operating Systems
- One Solution:
  - Present virtual I/O devices to guest VMs
  - Channel I/O requests to a trusted host VM running a popular OS that has the device drivers
I/O Virtualization

(a) Virtual DD, channel to guest OS - e.g. Xen

(b) Integrate DD with VMM - e.g. VMware ESX (Linux DDs)

Memory Virtualization

- Traditional way is to have the VMM maintain a shadow of the VM’s page table
- The shadow page keeps mapping from virtual pages within a VM to real physical pages
- When VM tries to change MMU to point to a specific page table, this traps to VMM which updates MMU to point to the shadow page table
- Shadow PT has actual mappings between virtual pages in VM and real physical pages in machine
- Keeping shadow page table in sync with guest PT:
  - When guest OS updates page table, VMM updates shadow
  - E.g. pages of guest OS page table marked read-only

Case Study: VMware ESX Server

- Type I VMM - Runs on bare hardware
- Full-virtualized – Legacy OS can run unmodified on top of ESX server
- Fully controls hardware resources and provides good performance

ESX Server – CPU Virtualization

- Most user code executes in Direct Execution mode; near native performance
- For kernel code, uses runtime Binary Translation for x86 virtualization
  - Privileged mode code is run under control of a Binary Translator, which emulates problematic instructions
  - Fast compared to other binary translators as source and destination instruction sets are nearly identical
ESX Server – Memory Virtualization

- Maintains shadow page tables with virtual to machine address mappings.
- Shadow page tables are used by the physical processor.
- ESX maintains a “pmap” data structure for each VM, which holds “physical” to machine address mappings.
- Shadow page tables are kept consistent with pmap.
- With pmap, ESX can easily remap a physical to machine page mapping, without guest VM knowing the difference.

ESX Server – Memory Mgmt

- Page reclamation
  - Problem: VMM does not have as good information on page usage as guest OS, for actual page replacement algorithms.
  - Solution: Ballooning technique
    - Reclaims memory from other VMs when memory is overcommitted.
- Page sharing
  - Many VMs will use the same pages.
  - Solution: Content based sharing
  - Eliminates redundancy and saves memory pages when VMs use same operating system and applications.

ESX Server – Ballooning

- Inflates balloon (+ pressure)
- Deflates balloon (- pressure)
- Guest OS manages memory implicitly.
- May page out to virtual disk.
- May page in from virtual disk.

ESX Server – Page Sharing

- Copy-on-write for writing shared pages.
- Hash page contents.
Real World Page Sharing

<table>
<thead>
<tr>
<th>Workload</th>
<th>Guest Types</th>
<th>Total MB</th>
<th>Saved MB</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate IT</td>
<td>Windows</td>
<td>2048</td>
<td>673</td>
<td>32.9</td>
</tr>
<tr>
<td>Nonprofit Org</td>
<td>Linux</td>
<td>1846</td>
<td>345</td>
<td>18.7</td>
</tr>
<tr>
<td>VMware</td>
<td>Linux</td>
<td>1658</td>
<td>120</td>
<td>7.2</td>
</tr>
</tbody>
</table>

Corporate IT – database, web, development servers (Oracle, WebSphere, IIS, Java, etc.)
Nonprofit Org – web, mail, anti-virus, other servers (Apache, MySQL, MySQL, etc.)
VMware – web proxy, mail, remote access (Apache, Pardus, RAV, sox, etc.)

ESX Server – I/O Virtualization

- Has highly optimized storage subsystem for networking and storage devices
- Directly integrated into the VMM
- Uses device drivers from Linux kernel to talk directly to device
- Low performance devices are channeled to special “host” VM, which runs a full Linux OS

VMware Workstation

- Type II VMM - Runs on host operating system
- Full-virtualized – Legacy OS can run unmodified on top of VMware Workstation
- Appears like a process to the Host OS

Workstation - Virtualization

- CPU Virtualization and Memory Virtualization
  - Uses Similar Techniques as the VMware ESX server
- I/O Virtualization
  - Workstation relies on the Host OS for satisfying I/O requests
  - I/O incurs huge overhead as it has to switch to the Host OS on every IN/OUT instruction
  - E.g., Virtual disk maps to a file in Host OS
### Xen

- Type I VMM
- Para-virtualized
- Open-source
- Designed to run about 100 virtual machines on a single machine

### Xen – CPU Virtualization

- Privileged instructions are para-virtualized by requiring them to be validated and executed with Xen
- Processor Rings
  - Guest applications run in Ring 3
  - Guest OS runs in Ring 1 (not ring 0 as without virtualization)
  - Xen runs in Ring 0
  - So if guest OS executes privileged instruction, it traps to Xen

### Xen – Memory Virtualization (1)

- Initial memory allocation is specified and memory is statically partitioned
- A maximum allowable reservation is also specified.
- Balloon driver technique similar to ESX server used to reclaim pages
Xen – Memory Virtualization(2)

- Guest OS is responsible for allocating and managing hardware page table
- Xen involvement is limited to ensure safety and isolation
- OS maps Xen VMM into the top 64 MB section of every address space to avoid TLB flushes when entering and leaving the VMM

Xen – I/O Virtualization

- Xen exposes its own set of clean and simple device abstractions – doesn’t emulate existing devices
- I/O data is transferred to and from each domain via Xen, using shared memory, asynchronous buffer descriptor rings
- Xen supports lightweight event delivery mechanism used for sending asynchronous notifications to domains

Summary

- Classifying Virtual Machine Monitors
  - Type I vs. type II
  - Full vs. para-virtualization
- Processor virtualization
- Memory virtualization
- I/O virtualization