Paravirtualization

Yiying Zhang
Quiz 2 at the end of this lecture (starts at 10:40am sharp)

Quiz on Canvas

Turn on your camera for proctoring

Use Zoom chat to ask clarification questions

Open everything, but do not cheat (no communication to others)!

Project progress report due on Feb 20th!
Popek and Goldberg’s Theorem (1974)

– A machine can be virtualized (using trap-and-emulate) if every sensitive instruction is privileged.
Not all sensitive instructions are privileged with x86 for many years, i.e., non-virtualizable processor.

These instructions do not trap and behave differently in kernel and user mode.

Example: `popf`
- Pops 16 bits from top of the stack to the `%eflags` register
- Bit 9 of `%eflags` masks interrupts (i.e., enables/disables interrupts)
- `popf` is not privileged. What happens if guest OS (ring 1) runs `popf` to `%eflags`?
- In Ring 0, `popf` can set bit 9, but CPU silently ignores `popf`'s setting of system flags (bit 9) when running in Ring 1
- What should happen is a trap so that VMM can emulates interrupts (change which interrupts to forward to guest OS)

Popek and Goldberg’s Theorem (1974)

A machine can be virtualized (using trap-and-emulate) if every sensitive instruction is privileged.

Xen and the Art of Virtualization
Virtualization Approach 3: Direct Execution with Binary Translation

- VMM dynamically rewrites instructions
- So that non-virtualizable instructions can trap to VMM
- VMware’s main selling point (at least in early years)
[lec3] Virtualization Approach 5: Direct Execution with Paravirtualization

- Full virtualization (no guest OS modification)
  - Tricky and has performance overhead

- Para-virtualization: modified guest OS
  - Change (rewrite) guest OS to remove sensitive but unprivileged instructions and to use other tricks to make virtualization faster
    - Guest OS works with hypervisor (i.e., knows that it is a VM) and has some exposure to hardware
      - e.g., guest OS informs hypervisor of page table changes
      - e.g., guest OS directly calls hypervisor on system calls (hypercalls)
  - Guest applications are still unmodified
  - Pros and Cons?
• A para-virtualization hypervisor (not the only one or the first one)

• First public version released in 2003

• Backing major cloud service e.g. AWS for many years

• Natively supported by Linux after kernel version 3.0 in 2011
Xen: Overall Architecture
### Xen Para-virtualization Techniques Overview

<table>
<thead>
<tr>
<th>Memory Management</th>
<th>Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.</td>
</tr>
<tr>
<td>Paging</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>Protection</td>
<td>Guest OS must run at a lower privilege level than Xen.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.</td>
</tr>
<tr>
<td>System Calls</td>
<td>Guest OS may install a ‘fast’ handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirection through Xen on every call.</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Hardware interrupts are replaced with a lightweight event system.</td>
</tr>
<tr>
<td>Time</td>
<td>Each guest OS has a timer interface and is aware of both ‘real’ and ‘virtual’ time.</td>
</tr>
<tr>
<td>Device I/O</td>
<td>Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.</td>
</tr>
</tbody>
</table>
Control Transfer: Hypercalls and Events

- Hypercall allows guest OS to perform a software trap to hypervisor for privileged operations
  - Similar to regular system call from user process to kernel

- Async events replaces device interrupts to perform notification from Xen to a domain, e.g. new data received from network
  - Pending events stored in a per-domain bitmask
  - Benefits?
CPU Virtualization
[lec2] Protection Rings

- More privileged rings can access memory of less privileged ones
- Calling across rings can only happen with hardware enforcement
- Only Ring 0 can execute privileged instructions
- Rings 1, 2, and 3 trap when executing privileged instructions
- Usually, the OS executes in Ring 0 and applications execute in Ring 3

Image Source: https://commons.wikimedia.org/wiki/File:CPU_ring_scheme.svg
Xen Protection

- Xen runs guest OS in ring 1, guest application in ring 3, hypervisor in ring 0
- Privileged instructions trap to hypervisor
- Modify guest OS to change sensitive instructions into hyper calls

| CPU Protection | Guest OS must run at a lower privilege level than Xen. |
## Exceptions and SysCalls

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Calls</td>
<td>Guest OS may install a ‘fast’ handler for system calls, allowing direct calls from an application into its guest OS and avoiding indrecting through Xen on every call.</td>
</tr>
</tbody>
</table>

- Guest OS registers an exception handler table with Xen for validation
- Most handlers do not need modification (page fault handler needs to be changed to not use CR2 for getting faulting page info)
- Guest OS can register a “fast” handler for system calls for the hardware to directly jump to it
Memory Virtualization
[lec4] Difficulty in Virtualizing Hardware-Managed TLB

- Hardware-managed TLB
  - Hardware does page table walk on each TLB miss
  - and fills TLB with the found PTE
- Hypervisor doesn’t have chance to intercept on TLB misses
difficulty in virtualizing hardware-managed TLB

- Hardware-managed TLB
  - Hardware does page table walk on each TLB miss
  - and fills TLB with the found PTE
- Hypervisor doesn’t have chance to intercept on TLB misses
[lec4] Difficulty in Virtualizing Hardware-Managed TLB

- Hardware-managed TLB
  - Hardware does page table walk on each TLB miss
  - and fills TLB with the found PTE
- Hypervisor doesn’t have chance to intercept on TLB misses

- Solution-1: shadow paging
- Solution-2: direct paging (para-virtualization) (later this quarter if have time)
- Solution-3: new hardware
[lec4] Shadow Paging

- GPT
- PPN
- pmap
- SPT
- MPN
- VPN
- PPN
- MPN
- CR3

Guest VM

Diagram showing the relationship between different components in Shadow Paging.
[lec4] Hardware-Assisted Memory Virtualization

- **VPN**
- **PPN**
- **EPT**
- **MPN**

**Guest VM**

**VMM**

- **PT**
- **PPN**
- **EPT**
- **MPN**

**Non-root mode**

**Root mode**

- **VA**
- **PA**
- **PT**
Paravirtualization: Direct Paging

- Register VPN->MPN PTs with MMU
  - These PTs are read-only to guest OS
- Guest OS manages PPN <-> MPN mapping
- PT updates are through hypercalls to hypervisor for validation before they are applied
  - e.g., can’t map to other guest’ machine memory, no writable PT
Memory Allocation

• Initial memory partition of a VM is specified at the time of its creation

• Uses a balloon driver to pass memory between VM and hypervisor for adjusting memory size
I/O Virtualization
Xen I/O Virtualization

- Xen hypervisor does not emulate hardware devices
- Exposes a set of simple and clean device abstractions
- Lightweight event delivery mechanism for virtualized interrupts
- I/O transfers between VM and hypervisor are through shared-memory
Data Transfer: I/O Rings

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

- Request queue - Descriptors queued by the VM but not yet accepted by Xen
- Outstanding descriptors - Descriptor slots awaiting a response from Xen
- Response queue - Descriptors returned by Xen in response to serviced requests
- Unused descriptors
Overall Performance

Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).
Transparency vs. Optimization

• Butler Lampson once gave a set of principles for system design. Among these, he gave two conflicting pieces of advice on the nature of implementations. He said,

• “Keep secrets of the implementation. Secrets are assumptions about an implementation that client programs are not allowed to make... Obviously, it is easier to program and modify a system if its parts make fewer assumptions about each other.”

• “One way to improve performance is to increase the number of assumptions that one part of a system makes about another; the additional assumptions often allow less work to be done, sometimes a lot less.”

• That is, on the one hand we should hide an implementation for ease of development (transparency), and, on the other, we should expose our implementations for speed (optimization). (cited from CSE221 homework)
Xen chooses optimization while VMware chooses transparency
What goals of Xen are not valid or less valid in today's cloud environments?
New use cases for para-virtualization now?