Paravirtualization

Poojitha, Umit
Full virtualization

• Unmodified OS
• It doesn’t know about hypervisor
• Back and forth between hypervisor and MMU-visible shadow page table: inefficient
• Unprivileged instructions which are sensitive: difficult to handle (binary translation VMware ESX)
• Cannot access hardware in privileged mode
• If guest OS wants real resource information? (Timer, superpages)
Paravirtualization

• Modify Guest OS
• It knows about hypervisor
• Applications not modified
• Some exposure to hardware and real resources like time
• Improved performance (reduce redirections, allowing guest OS to use real hardware resources in a secure manner)
• It can allow us to do virtualization without hardware support
Discussion – Xen

• Memory Management
• CPU
  • Protection
  • Exception
  • System call
  • Interrupt
  • Time
• Device I/O
Protection

• Privilege of OS must be less than Xen:
• In x86, 4 levels of privilege
• 3 for applications, Zero for OS - generally
• Downgrade guest OS to level 1 or 2
• Xen will be at 0

Wikipedia
Exceptions

• System calls, Page Faults

• Register with Xen: descriptor table for exception handlers
• No back and forth between Xen and Guest OS like in full Virtualization

• Fast handlers for system call:
• When Applications execute system call, it directly goes to Guest OS handler in ring 1 – not to Xen (But not page fault handler it has to go through Xen)
• Handlers validated before installing in hardware exception table
Time

• Guest OS can see: both real and virtual time
• Real time
• Virtual time
• Wall clock time
• Why do you want to see time? e.g., need it for TCP: TCP timeouts, RTT estimates
Memory Management

- TLB flush on context switch (Guest OS – Guest OS) – Undesirable
- Software TLB – can virtualize without flushing between switches
- Hardware TLB – tag it with address space identifier.
- Want to Avoid flushing between switches (Guest OS - Xen)

- What about x86?
- Not software TLB. Hardware, but no tags
- What can we do?
Memory Management

x86 architecture perspective

• Guest OS allocate and manage own hardware page tables
• Minimal involvement of Xen
• More safety and isolation
• Avoid flush on switch (Guest OS - Xen) : Xen in top 64MB of VM address space.
• Guest OS shouldn’t access top 64MB.
• Xen never paged out
Paging

- Guest OS has its own memory reservation.
- When it needs new page table, allocate from what it has
- Registers with Xen
- Xen gives up write-privileges
- Guest OS can read directly
- Guest OS must validate with Xen for writes / updates
- No back and forth like in Full virtualization.
- No shadow table here. Life is easier.
Hypercalls and Events: Control Transfer

• So guest OS validates with Xen for every update.
• Minimize these calls: Batch these updates together. “Hypercalls” to Xen

• Hypercalls: think of them as synchronous calls TO Xen
• In xen/include/public/xen.h: ~40 hypercalls. E.g. set trap table, mmu update, etc.
• Another term: Events

• Events: async notifications FROM Xen. Like device interrupts
• Guest OS: Xen hosts this OS
• Domain: VM, inside which Guest OS executes
• Guest OS- program, Domain-process
• Domain0: separate Guest OS. Privileged. Control management
• Domain0- more access to hardware & hypervisor
• Like a “supervisor” who manages others
• It creates new domains. Work delegated to it.
• Reduces hypervisor complexity
• Memory reservation for new domains done statically. Non contiguous phys mem.

Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a XenoLinux environment.

Xen and the Art Of Virtualization. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield
I/O

- Event notifications instead of Interrupts
- Simple abstraction:
  - Asynchronous I/O rings
- Data transfer to guest from Xen and vice versa
- Using these shared memory buffers
I/O rings

- Ring is circular queue of descriptors
- Descriptors allocated by domains
- Descriptors don’t directly contain I/O data
- Two pairs of producer/consumer pointers
- Domains place request
- Domain Advances request producer pointer
- Xen removes and handles them
- Xen advances request consumer pointer
- Zero copy transfer

Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.
Disk

- Domain0, the privileged one, can access disk directly
- Other domains can not. They use Virtual Block Drivers. VBD
- VBD: contains ownership and access control information
- Translation table: Map VBD request -> physical device, sector address
- VBD, for others, is created and configured at Domain0
- Other domains access via I/O rings
- Reorder, Batch disk requests
Network

• Each domain has:
  • 1 Send I/O ring,
  • 1 receive I/O ring
• Send packet: domains place in I/O ring.
• Receive packet: Xen does pattern matching to find destination domain
• Go through Virtual Firewall. Match patterns.
• Domain0 created the rules. Pattern -> Action
## Questions?

<table>
<thead>
<tr>
<th>Memory Management</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.</td>
</tr>
<tr>
<td>Paging</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 discontinuous machine pages.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th></th>
</tr>
</thead>
<tbody>
<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 indirecting 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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device I/O</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Network, Disk, etc.</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>

**Table 1: The paravirtualized x86 interface.**