I/O Virtualization with Hardware Support

Ubaid H. and Vasia P.
Trade-offs and Motivation

- In a fully virtual I/O, we have interposition
  - Full encapsulation (store, pause, resume), portability, flexibility, live migration
  - Slow performance

- Paravirtualization
  - Benefits of interposition, with better performance
  - Special drivers and configuration required

- I/O Virtualization with Hardware Support
  - No interposition
  - Best performance (closest to bare metal)

Table 6.4: Pros and cons of the three virtual I/O models

<table>
<thead>
<tr>
<th>Virtual I/O Model</th>
<th>Emulation</th>
<th>Paravirtualization</th>
<th>Device Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable, no host-specific software</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Interposition and memory optimizations</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Performance</td>
<td>worse</td>
<td>better</td>
<td>best</td>
</tr>
</tbody>
</table>
Direct Device Assignment

Idea: dedicate an I/O device to a specific VM, and give the VM control of the device.
Direct Device Assignment

Naive implementation has serious pitfalls

- Device uses host-physical addresses - VM doesn’t know them
- Not scalable - # physical devices <<< # VMs
- Guest controls device, device can perform DMA at any physical location

Hardware comes to the rescue

- IOMMU allows for security and isolation
- SRIOV allows for scalability
Protection?
IOMMU (I/O Memory Management Unit)

Major chip vendors introduced IOMMU:
- Intel Vt-d
- AMD-Vi

Main components:
- DMA Remapper (DMAR)
- Interrupt Remapper (IR)
Figure 6.15: IOVA translation with the Intel IOMMU.
1D vs 2D IOMMU

(a) Bare metal, 1D-IOMMU
(b) Virtual, 1D-IOMMU
(c) Virtual, 2D-IOMMU
Interrupt Remapping

Device can trigger interrupt by performing a DMA to a dedicated memory range

- 0xFEE00000 - 0xFEEFFFFF on x86

VM can program device to perform DMA to this region, to perform arbitrary interrupts

Without IR, IOMMU cannot distinguish between genuine MSI from the device, and a DMA that pretends to be an interrupt.
W/O Interrupt Remapping

Figure 6.16: MSI interrupt delivery without interrupt remapping support.
With Interrupt Remapping

**Figure 6.17:** MSI interrupt delivery with interrupt remapping support. (IRindex is denoted “interrupt_index” in the VT-d specification.)
Scalability?

- Physical Constraints
- Economical Constraints
SRIOV (Single Root I/O Virtualization)

- Multiplexing Devices At Hardware Level

SRIOV Enabled Devices

- Physical Function (PF)
  - Power Management
  - Configure/Manage VFs
- Virtual Function (VF)
  - Light Weight PCIe Function
  - Very Scalable
  - Performance Benefits
Performance: SRIOV
Problem ? Exits!

- Can Devices ‘Talk’ to VM efficiently ?

Figure 6.22: Chain of events when a physical device triggers an interrupt, without hardware support for direct interrupt delivery.
Solution: Intel VT-x Hypervisor

- Assigned EOI Register
  - Exit Associated with EOI
  - Can Give Write Permissions to VM for EOI? **Old LAPIC!**
  - x2APIC - Bitmap
  - Model Specific Registers (MSR)

- Exitless Interrupts
  - Shadow IDT
  - Control Bit - **External Interrupt Exiting**
  - Trap and Emulate
  - Some Security Measures

**Caveat:**

- Assumption that all interrupts arriving at a core belong to VMs running on it
- Some Security Measures.
Performance: Intel VT-x Hypervisor

Figure 6.23: ELI interrupt delivery flow.
Are We Done? No!

- Same Core
- Increased Complexity of Hypervisor
- ELI - All or Nothing

Solution:

- Posted Interrupts - APICv
  - CPU Posted Interrupts
  - IOMMU Posted Interrupts
Intel APIC Virtualization (APICv)

- Acknowledgement and Receipt of Interrupts at guests without Hypervisor
- Virtual APIC Page
- vIRR, vISR, vEOI, vICR Registers
- Hardware Emulation
- Support for Posted Interrupts
- Exit
**CPU Posted Interrupts**

- Interrupts that are directly injected by the Hypervisor to a Guest on a Different Core

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>255:0</td>
<td>PIR</td>
<td>Post-Interrupt Requests, one bit per requested vector</td>
</tr>
<tr>
<td>256</td>
<td>ON</td>
<td>Outstanding Notification, logically, a bitwise OR of PIR</td>
</tr>
<tr>
<td>257</td>
<td>SN</td>
<td>Suppress Notification, of non-urgent interrupts</td>
</tr>
<tr>
<td>271:258</td>
<td>-</td>
<td>Reserved (must be 0)</td>
</tr>
<tr>
<td>279:272</td>
<td>NV</td>
<td>Notification Vector, doorbell to notify about pending PIR</td>
</tr>
<tr>
<td>287:280</td>
<td>-</td>
<td>Reserved (must be 0)</td>
</tr>
<tr>
<td>319:288</td>
<td>NDST</td>
<td>Notification Destination, a physical APIC ID</td>
</tr>
<tr>
<td>511:320</td>
<td>-</td>
<td>Reserved (must be 0)</td>
</tr>
</tbody>
</table>
Figure 6.25: CPU posted interrupt illustration.
IOMMU Posted Interrupts

Figure 6.26: IOMMU (VT-d) posted interrupt illustration (compare with Figure 6.17).
Final Remarks

- IOMMU and SRIOV allow for safe and scalable direct device assignment
- Exitless/Posted Interrupts enable Direct Interrupt Delivery (Bare Metal Performance)
  - Direct Device I/O gives up I/O interposition