paravirtualized interfaces in HVM guests
Linux as a guests: problems

Linux PV guests have limitations:
- difficult “different” to install
- some performance issue on 64 bit
- limited set of virtual hardware

Linux HVM guests:
- install the same way as native
- very slow
Linux PV on HVM: the solution

- install the same way as native
- PC-like hardware
- access to fast paravirtualized devices
- exploit nested paging
Initial version in Linux 2.6.36:

- introduce the xen platform device driver
- add support for HVM hypercalls, xenbus and grant table
- enables blkfront, netfront and PV timers
- add support to PV suspend/resume
- the vector callback mechanism
Old style event injection
Receiving an interrupt

do_IRQ

handle_fasteoi_irq

handle_irq_event

xenEvtchnDoUpcall

ack_apic_level ← >=3 VMEXIT
The new vector callback
Receiving a vector callback

xenEvtchnDoUpcall
Later enhancements (2.6.37+):
- ballooning
- PV spinlocks
- PV IPIs
- Interrupt remapping onto event channels
- MSI remapping onto event channels
Interrupt remapping

- DomU
  - Drivers
  - events.c
- Dom0
  - Qemu-xen
    - Emulated PCI Hardware
    - Emulated E1000
- Xen
  - xc_hvm_set_pci_intx_level
- PCI Hardware
  - PCI device
  - PCI device
  - PCI device
  - PCI device
## PV spectrum

<table>
<thead>
<tr>
<th></th>
<th>HVM guests</th>
<th>Classic PV on HVM</th>
<th>Enhanced PV on HVM</th>
<th>Hybrid PV on HVM</th>
<th>PV guests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot sequence</td>
<td>emulated</td>
<td>emulated</td>
<td>emulated</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Memory</td>
<td>hardware</td>
<td>hardware</td>
<td>hardware</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Interrupts</td>
<td>emulated</td>
<td>emulated</td>
<td>paravirtualized</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Timers</td>
<td>emulated</td>
<td>emulated</td>
<td>paravirtualized</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Spinlocks</td>
<td>emulated</td>
<td>emulated</td>
<td>paravirtualized</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Disk</td>
<td>emulated</td>
<td>paravirtualized</td>
<td>paravirtualized</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Network</td>
<td>emulated</td>
<td>paravirtualized</td>
<td>paravirtualized</td>
<td></td>
<td>paravirtualized</td>
</tr>
<tr>
<td>Privileged operations</td>
<td>hardware</td>
<td>hardware</td>
<td>hardware</td>
<td></td>
<td>paravirtualized</td>
</tr>
</tbody>
</table>
Benchmarks: the setup

Hardware setup:
Dell PowerEdge R710
CPU: dual Intel Xeon E5520 quad core CPUs @ 2.27GHz
RAM: 22GB

Software setup:
Xen 4.1, 64 bit
Dom0 Linux 2.6.32, 64 bit
DomU Linux 3.0 rc4, 8GB of memory, 8 vcpus
PCI passthrough: benchmark

PCI passthrough of an Intel Gigabit NIC

CPU usage: the lower the better:

[Graph showing CPU usage for domU and dom0 with two bars for each condition: interrupt remapping and no interrupt remapping.]
Kernbench

Results: percentage of native, the lower the better

PV on HVM 64 bit  PV on HVM 32 bit  HVM 64 bit  HVM 32 bit  PV 64 bit  PV 32 bit

90  95  100  105  110  115  120  125  130  135  140
Kernbench

Results: percentage of native, the lower the better
Results: percentage of native, the lower the better
PBZIP2

Results: percentage of native, the lower the better
SPECjbb2005

Results: percentage of native, the higher the better
SPECjbb2005

Results: percentage of native, the higher the better

- PV 64 bit
- PV on HVM 64 bit

Bar chart showing performance comparison between PV 64 bit, PV on HVM 64 bit, and KVM 64 bit.
Iperf tcp

Results: gbit/sec, the higher the better

![Bar Chart]

- PV 64 bit
- PV on HVM 64 bit
- PV on HVM 32 bit
- PV 32 bit
- HVM 64 bit
- HVM 32 bit
Iperf tcp

Results: gbit/sec, the higher the better
PV on HVM guests are very close to PV guests in benchmarks that favor PV MMUs

PV on HVM guests are far ahead of PV guests in benchmarks that favor nested paging
Questions?