VFIO on sPAPR/POWER

Task: provide isolated access to multiple PCI devices for multiple KVM guests on POWER8 box.

SR-IOV is the target.

aik@ozlabs.ru
aik@au1.ibm.com
What is what

**VFIO:** provides userspace access to PCI hardware
>99.9% users: **PCI Passthrough in QEMU**
Supports KVM and fully emulated guests
not virtio, not ibm-vio

**sPAPR:** Server POWER Architecture platform requirements
Defines para-virtual interface

**IOMMU:** hardware mapping of bus addresses to RAM. Allows DMA for guests.
**sPAPR:** TCE table - one entry per page.
`uint64_t tce_table[window_size >> 3];`
IOMMU Group
Group: set of PCI functions which can be isolated
Exposed as:
/sys/kernel/iommu_groups/0
/sys/kernel/iommu_groups/1
...
Platform code assigns devices to groups.
Do not trust devices and do not split functions between groups. Except SR-IOV.

VFIO Container (not LXC)
Container represents an IOMMU
It handles map/unmap, not a device

x86: container is h/w IOMMU domain (>=1 group)
sPAPR: container always has one group
## PCI support in QEMU

<table>
<thead>
<tr>
<th></th>
<th>x86</th>
<th>sPAPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg space</td>
<td>piix</td>
<td>hypercalls</td>
</tr>
<tr>
<td>interrupts</td>
<td>apic</td>
<td>hypercalls</td>
</tr>
<tr>
<td>BARs</td>
<td>mmap/emu</td>
<td>mmap/emu</td>
</tr>
<tr>
<td>DMA</td>
<td>???</td>
<td>???</td>
</tr>
</tbody>
</table>

Note: endianness for host/guest - Big/Little
DMA – here is the difference

**x86:** no guest visible IOMMU.
Entire guest RAM is mapped (pinned)

**sPAPR:** guest-visible IOMMU

- map/unmap done via hypercalls:
  \[ H\_PUT\_TCE(liobn, ioaddr, gpa) \]
  \[ H\_STUFF\_TCE(liobn, ioaddr, num) \]
  \[ H\_PUT\_TCE\_INDIRECT(liobn, ioaddr, gpa, num) \]

- DMA windows in device tree
  - LIOBN – Logical IO Bus number
  - backed by IOMMU (a.k.a. TCE) table
  - default **32bit** DMA window, **2GB**, **4K** pages
  - guest maps RAM page to window
VFIO drivers

- VFIO PCI stub driver: `vfio_pci` module

- VFIO driver: `vfio` module
  - containers: `/dev/vfio/vfio`
  - groups: `/dev/vfio/0`, `/dev/vfio/1`, ...

- VFIO IOMMU driver:
  - default driver (PCI bus iommu_ops)
    `vfio_iommu_type1` for x86/arm/...
  - SPAPR TCE driver
    `vfio_iommu_spapr_tce` module
Run QEMU

x86:
-device vfio-pci,host=0000:01:2.3

sPAPR:
-device spapr-pci-vfio-host-bridge,\
  id=mybus,iommu=5\
  -device vfio-pci,host=0000:01:2.3,\
  bus=mybus.0

sPAPR: container always has one group
container also corresponds to a PHB
DMA on sPAPR

Typical DMA map/unmap operation:

GUEST:  
- `dma_alloc()`
- `hypercall (H_PUT_TCE)`

KVM: 
- real mode (MMU off) handler
- virtual mode (MMU on) handler

QEMU: 
- GPA -> UA, LIOBN -> VFIO container
- `ioctl(VFIO_container_fd, (un)map)`

HOST-VFIO: `vfio_spapr_tce` driver

HOST-ARCH: 
- UA -> HPA + update IOMMU table

To do:
- pin pages
- locked_vm counter

ALLOC > HCALL > RM > VM > USER > VFIO > PLATF
Performance

180MB/s on 10Gbit card (Chelsio CXGB3, 1050MB/s)

_H_PUT_TCE_INDIRECT/H_STUFF_TCE help -> 320MB/s

Why:
- Device drivers call `dma_alloc()` a lot
- Real mode -> Virtual mode (enabling MMU)
- Virtual mode -> User mode
- User mode -> ioctl -> Virtual mode

ALLOC > HCALL > RM > VM > USER > VFIO > PLATF
Performance - use platform API in Virtual mode

To do (in addition to QEMU):
- GPA -> UA - use KVM memslots
- LIOBN -> VFIO container - VFIO KVM device

Performance -> H_PUT_TCE 180 -> 750MB/s
..._INDIRECT 320 -> 850MB/s

ALLOC > HCALL > RM > VM > user > vfio > PLATF
Performance - use platform API in Real mode (pHyp)

All the same as in virtual mode except... pinning!

Page struct lookup: \texttt{F000.0000.0000.0000} vs. \texttt{C000.0000.0000.0000}

\begin{itemize}
  \item no \texttt{pfm\_to\_page()}/\texttt{page\_to\_pfm()}
    \rightarrow Added \texttt{realmode\_pfm\_to\_page()} API
    (fails if page struct is split)
  \item no \texttt{get\_user\_pages\_fast()}
    \rightarrow reimplement GPA -> HPA
  \item pin pages (adjust page use counter):
    \rightarrow Corner cases: counter ==0 (map), ==1 (unmap)
    \rightarrow Huge pages are even worse
\end{itemize}

Performance -> $\texttt{H\_PUT\_TCE}$ 750 -> 1050MB/s
\hspace{1cm} ...\texttt{INDIRECT} 850 -> 1050MB/s (win?)

\texttt{ALLOC} > \texttt{HCALL} > \texttt{RM} > \texttt{vm} > \texttt{user} > \texttt{vfio} > \texttt{PLATF}
Dynamic DMA windows (DDW) – POWER7/8

More than 1 window -> more than 1 table
API: query, create, remove, reset

<table>
<thead>
<tr>
<th>32bit</th>
<th>64bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>@0</td>
</tr>
<tr>
<td>size</td>
<td>2GB</td>
</tr>
<tr>
<td>pagesize</td>
<td>4KB</td>
</tr>
<tr>
<td>hcalls</td>
<td>a lot</td>
</tr>
</tbody>
</table>

DDW TCE tables are small. For 64GB guest:
- 64K pages – TCE table is 8MB
- 16M pages – TCE table is 32KB

Note: start addresses are fixed in hardware
Dynamic DMA windows (DDW) - Problems

- Guest cannot explicitly choose the window start address.
  0800.0000.0000.0000 too big for many devices.
  Works only for 64-bit drivers.

- Duplicated content for multiple groups.

- No idea what the other operating system does.
Locked memory counter

```
ulimit -S -l 1000000000
```
Sets max size that may be locked in RAM

```
mm::locked_vm protected by a semaphore
-> not for real mode
```

Workaround:
update when IOMMU is started/stopped being used.

One IOMMU is 2GB default window and optional huge window (guest RAM)

```
4GB guest + 3 IOMMUs:
3 * (2GB+4GB) = 18GB (way too much)
```
Conclusion/Plan:

1. ask audience what to do with locked_vm

2. push DDW to QEMU and VFIO!

3. push KVM acceleration
   or
   drop it as it does not support hugepages

QUESTIONS?

aik@ozlabs.ru
aik@au1.ibm.com