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
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.
IOMMU Group
Group: set of PCI functions which can be isolated
/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
One IOMMU handles one or more groups.
VFIO drivers

- VFIO PCI stub driver: `vfio_pci` module

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

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

PCI config space
x86: emulates well known PHB (PIIX?)
sPAPR: QEMU implements sPAPR API

Interrupts
x86: emulates APIC
sPAPR: QEMU implements sPAPR API

BARs
Everything is the same
Endianness is the only worry:
    host/guest – Big/Little endian
DMA - here is the difference

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

sPAPR: guest-visible IOMMU
   ° DMA windows (LIOBN) in device tree
     ° backed by IOMMU table
     ° by default: 32bit DMA window, 2GB, 4K pages
     ° guest maps RAM page to window

   ° map/unmap done via hypercalls (LIOBN!):
     H_PUT_TCE
     H_STUFF_TCE
     H_PUT_TCE_INDIRECT
Running 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
DMA on sPAPR

Typical DMA map/unmap operation:

GUEST: \texttt{dma\_alloc()}

GUEST: hypercall (\texttt{H\_PUT\_TCE})

KVM: real mode (MMU off) handler

KVM: virtual mode (MMU on) handler

QEMU: GPA -> UA, LIOBN -> VFIO container

QEMU: \texttt{ioctl(VFIO\_container\_fd, (un)map)}

HOST-VFIO: \texttt{vfio\_spapr\_tce} driver

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

To do:

\begin{itemize}
  \item pin pages
  \item locked\_vm counter
\end{itemize}

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 -> HPA - 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: $F000.0000.0000.0000$ vs. $C000.0000.0000.0000$

- no `pfnto_page()`/`page_to_pfn()`
  - Added `realmode_pfn_to_page()` API
    - (fails if page struct is split)
- no `get_user_pages_fast()`
  - reimplement GPA -> HPA
- pin pages (adjust page use counter):
  - Corner cases: counter ==0 (map), ==1 (unmap)
  - Huge pages are even worse

Performance -> $H_{PUT\_TCE}$ 750 -> 1050MB/s
  ...间接 $850$ - $1050$MB/s (win?)

ALLOC > HCALL > RM > vm > user > vfio > 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 \textbf{64}bit drivers

- Duplicated content for multiple groups

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

ulimit -S -l 100000000
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. plan1: push DDW to QEMU and VFIO
3. plan2: push KVM acceleration or drop it as it does not support hugepages

QUESTIONS?

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