redhat.

Device Assignment with Nested Guest and DPDK

Peter Xu <peterx@redhat.com>
Red Hat Virtualization Team

Agenda

- Problems
  - Unsafe userspace device drivers
  - Device assignment for nested guests
- Solution
- Status update

BACKGROUNDS

Backgrounds

- What the talk is about?
  - DMA of assigned devices (no PCI configrations, IRQs, MMIOs...)
  - vIOMMU (QEMU, x86_64/Intel)
- These two features cannot work together (before)...
  - Guest IOMMU page table is only visible to the guest
  - An assigned hardware cannot see the guest IOMMU page table
- Will we need it?

PROBLEMS

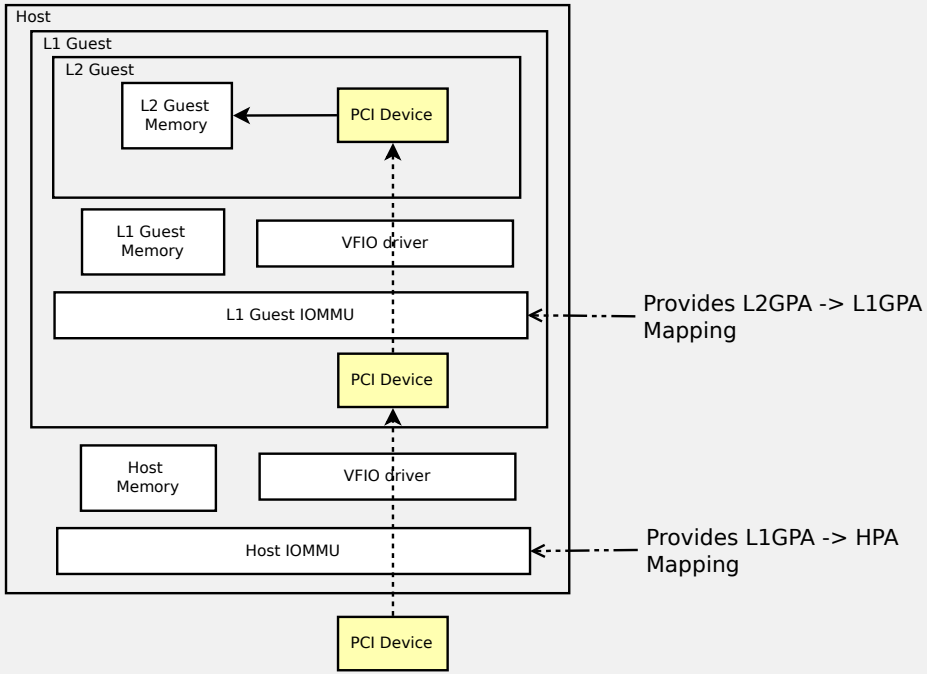
Problem 1: Userspace Drivers

- More userspace drivers!
  - VFIO/UIO driver can pass though a device to userspace
  - DPDK/SPDK uses PMDs to drive devices
- However, userspace drivers are not trusted
  - MMU protects CPU accesses (CPU instructions)
  - IOMMU protects device accesses (DMA)
- What if we want to “assign” an assigned device to DPDK in the guest?
  - No vIOMMU, means no device DMA protection
  - Guest kernel is at risk: as long as userspace driver used, kernel tainted!

Problem 2: Device Assignment for Nested Guests

- How device assignment works for L_1 guest?
  - Device seen by the L_1 guest
  - Guest uses L_1GPA as DMA addresses
  - Host IOMMU maps L_1GPA → HPA before guest starts
- What if we assign a hardware twice to a nested guest?
  - Device seen by both L_1 & L_2 guest
  - L_2 guest uses L_2GPA as DMA address
  - We need host IOMMU to map L_2GPA → HPA… but how?

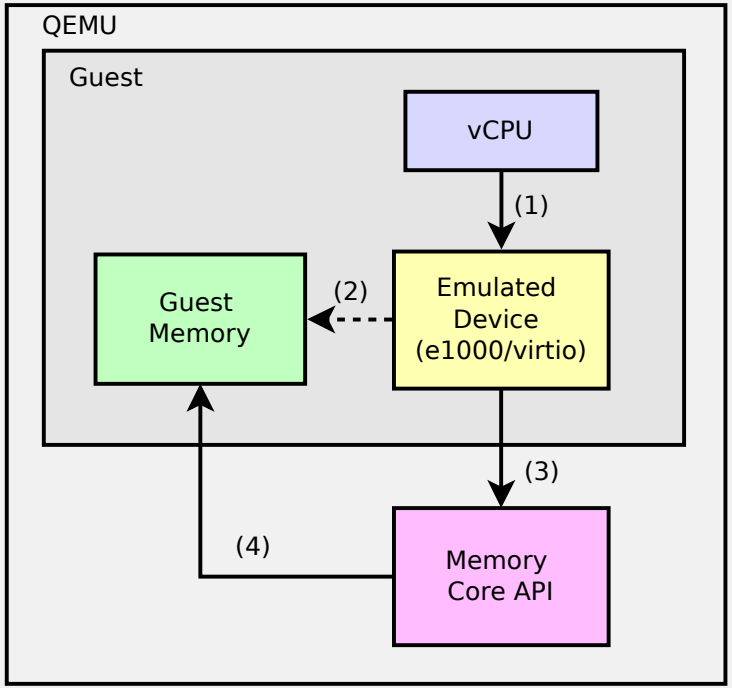
Problem 2: Device Assignment for Nested Guests (cont.)



SOLUTION

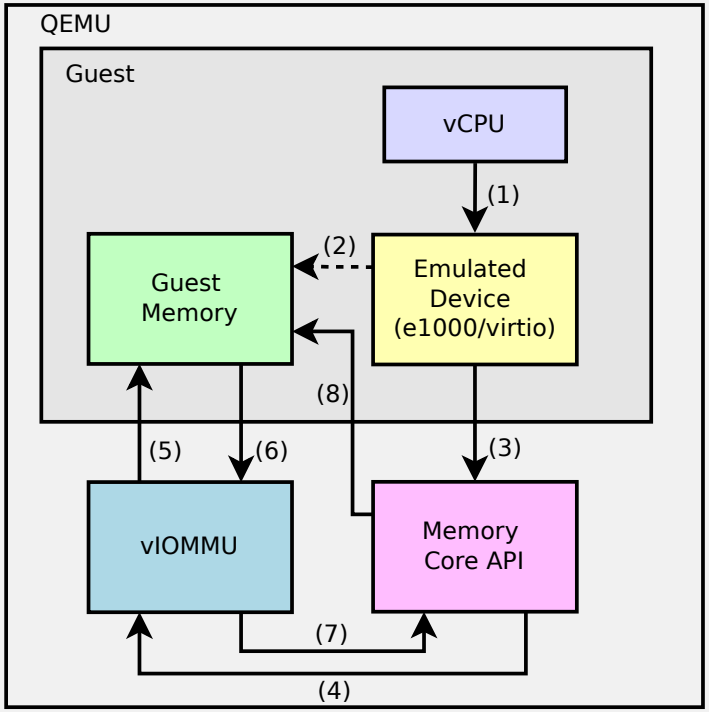
WHAT WE HAVE?

DMA for Emulated Device, w/o vIOMMU



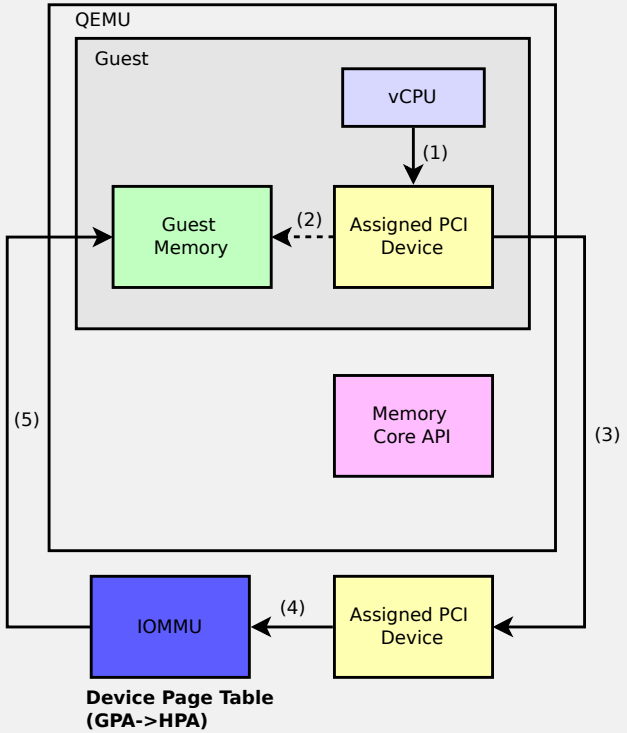
(1) IO Request
(2) Allocate DMA buffer
(3) DMA request (GPA)
(4) Memory access (GPA)

DMA for Emulated Device, w/ vIOMMU



(1) IO request
(2) Allocate DMA buffer, setup device page table (IOVA->GPA)
(3) DMA request (IOVA)
(4) Page translation request (IOVA)
(5) Lookup device page table (IOVA->GPA)
(6) Get translation result (GPA)
(7) Complete translation request (GPA)
(8) Memory access (GPA)

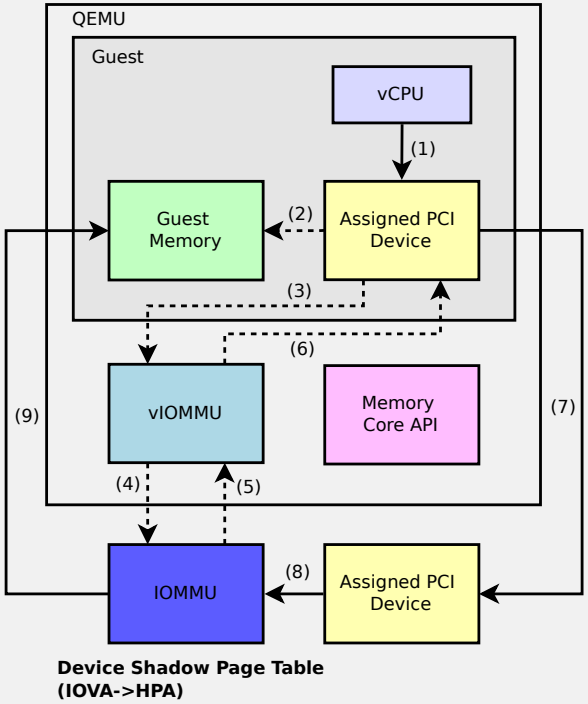
DMA of Assigned Devices, w/o vIOMMU



(1) IO request
(2) Allocate DMA buffer
(3) Virtual DMA request (using GPA)
(4) DMA request (using GPA)
(5) Memory access (using HPA)

WHAT WE NEED?

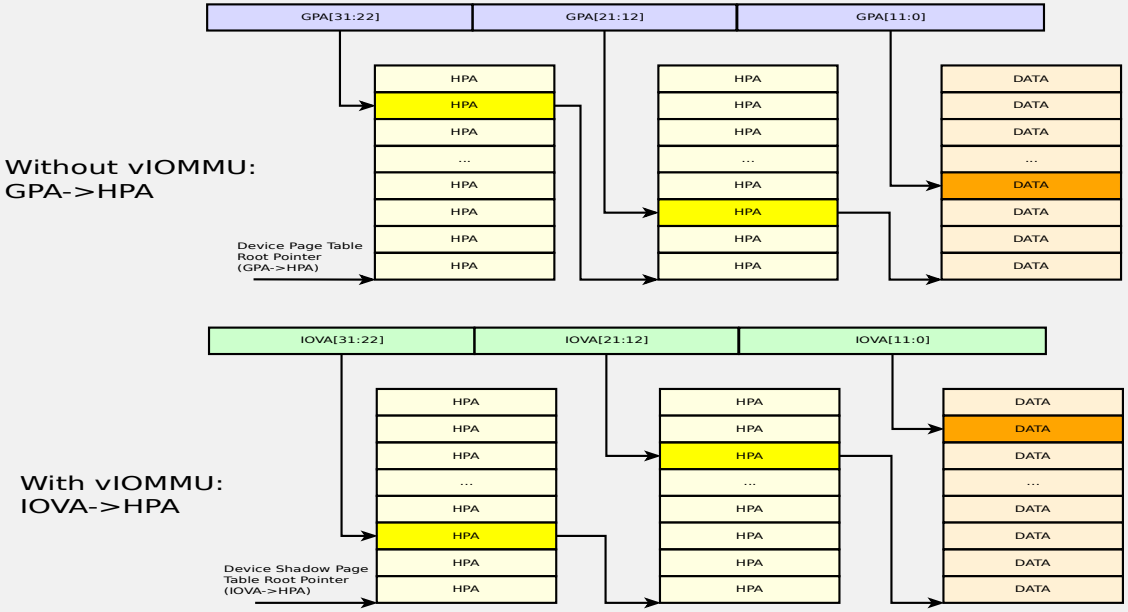
DMA of Assigned Devices, w/ vIOMMU



(1) IO request
(2) Allocate DMA buffer, setup device page table (IOVA->GPA)
(3) Send MAP notification
(4) Sync shadow page table (IOVA->HPA)
(5) Sync Complete
(6) MAP notification Complete
(7) Virtual DMA request (using IOVA)
(8) DMA request (using IOVA)
(9) Memory access (using HPA)

IOMMU Shadow Page Table

Hardware IOMMU page tables without/with a vIOMMU in the guest
(GPA→HPA is the original page table; IOVA→HPA is the shadow page table)



Synchronizing Shadow Page Tables

- Solution 1 (not used): Write-protect guest page table
  - Complicated; possibly need a new KVM interface to report the event
- Solution 2 (being used): VT-d caching mode
  - “Any page entry update will require explicit invalidation of caches” (VT-d spec chapter 6.1)
  - No KVM change needed
  - Have existing Linux guest driver support
  - Intel-only solution; PV-like, but also applies to hardware

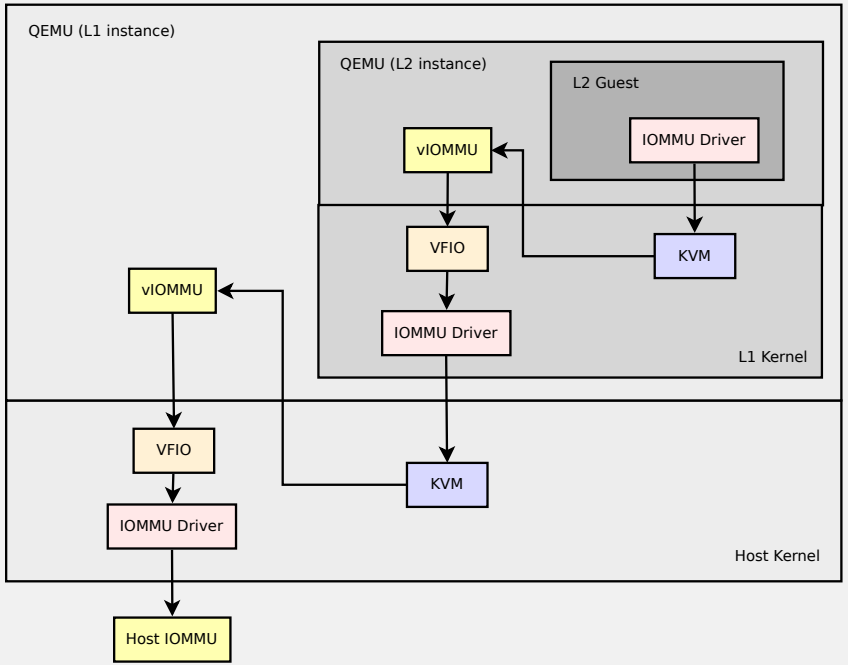
Shadow Page Tables: MMU vs IOMMU

TYPE	MMU	IOMMU
Target	Processors	Devices
Allow page faults?	Yes (of course!)	No [*]
Trigger mode (shadow sync)	Page Fault	Explicit Message (caching-mode)
Page Table Format	32bits, 64bits, PAE,...	64bits
Cost (shadow sync)	Small, relatively (KVM only)	Huge (long code path [**])
Need Previous State?	No	Yes [***]

[*]: Upstream work ongoing to enable Intel IOMMU page faults
[**/***]: Please refer to follow up slides for more information

Shadow Sync: Costly for IOMMU!

(Example: when L_2 guest maps one page)



Shadow Sync: About State Cache

- MMU shadow sync
  - Talk to page tables: PGD, PUD, PMD, PTE,...
  - Doing **set()** on page table entries
  - No need to cache previous state
- IOMMU shadow sync
  - Talk to vfio-pci driver: VFIO_IOMMU_MAP_DMA, VFIO_IOMMU_UNMAP_DMA
    (no direct access to page tables, the same even to vfio-pci driver underneath)
  - Doing **add()/remove()** on page table entries
  - We can either create a new entry (it must not exist before), or delete an old entry
  - Previous state matters, since otherwise we can’t judge what page has been mapped

STATUS UPDATE

Some Facts... and TBDs

- Emulated devices v.s. Assigned devices from IOMMU perspective
  - Emulated: fast mapping (no sync), slow IO (need guest translation)
  - Assigned: slow mapping (need sync), fast IO (no guest translation)
- Some performance numbers (Intel ixgbe, 10Gbps NIC)
  - Kernel ixgbe driver, very slow (~80% degradation on L1)
  - Userspace DPDK driver, very fast (close to line speed, both L1 & L2)
- Future works?
  - Reduce context switches when sync shadow pages? (vhost-iommu?)
  - Nested page table? (need hardware support, like EPT comparing to softmmu)
  - Sharing the state cache?
    (e.g. vfio-pci has similar state cache, see “vfio_iommu.dma_list”)

Wanna try?

- QEMU command line to try this out:

```
qemu-system-x86_64 -M q35,accel=kvm,kernel-irqchip=split -m 2G \
                   -device intel-iommu,intremap=on,caching-mode=on \
                   -device vfio-pci,host=XX:XX:XX
```

- Versions:
  - QEMU: please use v3.0 or newer
  - Linux: please use v4.18-rc1 or newer
- For more information, please visit:
  - https://wiki.qemu.org/Features/VT-d

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos