PCI P2P inside VM

Oded Gabbay / ogabbay@kernel.org
The kernel doesn’t support doing PCI P2P inside a Virtual Machine
Problem Statement

To allow PCI P2P, the kernel needs to verify that two peers can perform P2P.

This is done by calling the following function:

```c
static enum pci_p2pdma_map_type
calc_map_type_and_dist(struct pci_dev *provider, struct pci_dev *client, int *dist, bool verbose)
```
Virtual Machine owns entire Host resources

Behind the same switch with ACS p2p-forwarding disabled

Behind the same switch with ACS p2p-forwarding enabled AND root complex is white-listed
PCI information for P2P

- The function requires the following information:
  - PCIe topology
  - ACS configuration of the relevant PCIe switches
  - PCI root complex type / CPU type

- The PCI information is not exposed inside the Guest OS
  - Qemu emulates only old root ports
  - The user defines the PCI topology inside the guest, without any correlation to the real PCI topology
  - ACS configuration of PCI switches is not exposed
Proposal 1 – Replicate Topology

-device pcie-root-port-p2p.id=root_port1,chassis=x,slot=y,host=0000:30:00.0
-device pcie-switch-p2p,id=upstream_port1,bus=root_port1,host=0000:31:00.0
-device pcie-switch-p2p-downstream,id=downstream_port1,bus=upstream_port1,chassis=x1,slot=y1,host=0000:32:00.0
-device pcie-switch-p2p-downstream,id=downstream_port2,bus=upstream_port1,chassis=x1,slot=y1,host=0000:32:01.0
-device vfio-pci,host=0000:33:00.0,id=hostdev0,bus=downstream_port1
-device vfio-pci,host=0000:34:00.0,id=hostdev1,bus=downstream_port2
Proposal 1 (cont’)

- Emulate P2P PCIe root port and P2P Generic PCIe switch
  - Replicate ACS configuration and expose p2p root port type (for whitelist check)
- Add the new P2P PCIe root port vendor and device id to p2pdma whitelist
- Export p2pdma whitelist to uAPI header file

- Pros:
  - Minimal changes to kernel code
- Cons:
  - Requires major modification of existing VM configurations
  - Exposes host PCIe topology to the Guest
Proposal 2 - Hypercall

- Add a hypercall that will get the guest BDF of two peers, calculate and return the map type and distance.
  - Mapping between KVM and vfio-pci devices can be identified inside the kernel

- It will be called by `calc_map_type_and_dist()`
  - Decision to call hypercall can be done based on `kvm_para_available()`
  - Can we use root port type instead? (if its red-hat it’s virtualized)

- To match guest and host BDF, the kernel vfio-pci object will hold its guest BDF. This requires a new vfio-pci `ioctl` that QEMU will use.

- Pros:
  - Zero changes to existing VM configurations

- Cons:
  - A new hypercall to maintain
Proposal 3 - VIRTIO

• Instead of a hypercall, define a new VIRTIO device for the guest to query the host. The VIRTIO device will only export a kernel API, no need for uAPI.

• The kAPI will be called by `calc_map_type_and_dist()`
  • Decision to call API can be done in case VIRTIO device exists (or `kvm_para_available()` as in hypercall)

• Expose `calc_map_type_and_dist()` as a uAPI for QEMU

• Pros (vs. hypercall):
  • Straight-forward kernel changes

• Cons (vs. hypercall):
  • Requires minor modification of existing VM configurations