virtio-iommu. This article introduces what is… | by Michael Zhao

michael2012z.medium.com/virtio-iommu-789369049443

Michael Zhao April 22, 2022

virtio-iommu

Michael Zhao

This article introduces what is `virtio-iommu`, how it works and how to test it with virtual machine monitors. At last, we will discuss an issue that was seen on Qemu and Cloud Hypervisor.

What is `virtio-iommu` ?

It is a type of virtio device. For now it has not been ready in the latest release (v1.1) of virtio spec. An on-review version can be seen at github. This online file is of the format `TeX`. To get a better view, you can follow the instructions here and generate html or PDF to read.

According to this spec:

```
The virtio-iommu device manages Direct Memory Access (DMA) from one or more
endpoints. It may act both as a proxy for physical IOMMUs managing devices
assigned to the guest, and as virtual IOMMU managing emulated and paravirtualized
devices.
```

Among the 2 major use cases: “proxy for physical IOMMUs” and “virtual IOMMU”, I will talk about the latter one only in this article.

How `virtio-iommu` works?

I would like to quote some words from the spec for a very high level explanation:

```
The driver first discovers endpoints managed by the virtio-iommu device using
platform specific mechanisms. It then sends requests to create virtual address
spaces and virtual-to-physical mappings for these endpoints.
```

To be a little bit more specific:

1. While creating the `virtio-iommu` device, the virtual machine monitor (VMM) virtio devices to `virtio-iommu`.
2. In Linux kernel, the `platform specific mechanism` detects the attached devices, sends requests to the `virtio-iommu` device to set up mappings from virtual address to guest physical address.

3. When a virtio device (one of them that have been attached to the `virtio-iommu`) performs DMA, the virtio driver is using a virtual address. The virtio device will turn to the `virtio-iommu` device for address translation. Then the `virtio-iommu` device checks the mapping information of the device (so called "endpoint") and find out the guest physical address.

How to test `virtio-iommu`?

With Qemu, you can run a VM with `virtio-iommu` with following command to boot from a specified kernel image:

```
# Boot the VM from direct kernel➜ qemu-system-aarch64 -smp 4 -m 1024 -M virt -
nographic -kernel
~/ws/src/github.com/michael2012z/myLinuxGuest/arch/arm64/boot/Image -append
"keep_bootcon console=ttyAMA0 reboot=k panic=1 root=/dev/vda1 rw" -device  -drive
if=none,id=image,file=focal-server-cloudimg-arm64-custom-20210929-0-update-
kernel.raw -netdev user,id=user0 -device virtio-net-pci,netdev=user0 -cpu host -
enable-kvm -cpu host
```

The configuration items in bold font are the critical things for `virtio-iommu` :

- "virtio-blk-pci,…,iommu_platform=true" means this is a virtio-block device with PCI transport, and it is attached to iommu.
- "virtio-iommu-pci" means a `virtio-iommu` device with PCI transport is added in the guest.

If you want to test the same thing with firmware booting, the instruction would be:

```
# Boot the VM from firmware➜ qemu-system-aarch64 -smp 4 -m 1024 -M virt -nographic
-device virtio-blk-pci,drive=image,iommu_platform=true,disable-legacy=on -drive
if=none,id=image,file=focal-server-cloudimg-arm64-custom-20210929-0-update-
kernel.raw -netdev user,id=user0 -device virtio-net-device,netdev=user0 -cpu host
-enable-kvm -cpu host -drive
file=/usr/share/AAVMF/AAVMF_CODE.fd,if=pflash,format=raw,unit=0,readonly=on -
device virtio-iommu-pci
```

With Cloud Hypervisor, you can test the identical scenario booting from direct kernel by following command:

```
➜ sudo RUST_BACKTRACE=1 cloud-hypervisor --api-socket /tmp/cloud-hypervisor.sock -
-kernel <kernel src path>/arch/arm64/boot/Image --disk path=focal-server-cloudimg-
arm64-custom-20210929-0-update-kernel.raw, path=cloudinit-20210929-0.img --cmdline
"keep_bootcon console=hvc0 console=ttyAMA0 reboot=k panic=1 root=/dev/vda1 rw" --
cpus boot=4 --memory size=1024M --serial tty --console off --log-file log.log -vvv
--net tap=,mac=12:34:56:78:90:01,ip=192.168.1.1,mask=255.255.255.0
```

On Cloud Hypervisor, appending "iommu=on" is the only thing you need to do to hide the device behind a `virtio-iommu` device.

An immediate question is: With Cloud Hypervisor how to boot a VM with `virtio-iommu` from firmware? The answer is: no way, now.

When I tried to do that with the latest Cloud Hypervisor release (v23.0), I got an error and the VMM failed:

```
# CLOUDHV_EFI.fd is the binary of EDK2➜ sudo RUST_BACKTRACE=full cloud-hypervisor
--api-socket /tmp/cloud-hypervisor.sock --kernel CLOUDHV_EFI.fd --disk path=focal-
server-cloudimg-arm64-custom-20210929-0-update-kernel.raw, --cpus boot=4 --memory
size=4096M --serial tty --console off --log-file log.log -vvv --net
tap=,mac=12:34:56:78:90:01,ip=192.168.1.1,mask=255.255.255.0
```

Reason of the failure:

1. The crash happened when a `virtio-block` endpoint that had been attached to the `virtio-iommu` device tried to translate a virtual address (GVA) into guest physical address (GPA). The mapping info of the endpoint was not found, an error was thrown.
2. Why was the mapping info not found? Because the `virtio-iommu` device had not been enabled.
3. Why was the `virtio-iommu` not enabled? Because the firmware didn't care `virtio-iommu`, and didn't drive it.

It is not a problem of the firmware. It doesn't have to work with iommu. But the VMM should improve to avoid this corner case.

How did Qemu resolve this problem?

The answer is in this commit. To be simple, during the boot stage, the `virtio-iommu` will treat any endpoint in `BYPASS` mode. This way, the devices use their virtual address as guest physical address, which is true in firmware booting stage.

The fix in Cloud Hypervisor was planned for release v24.0.

Reference