SPDK VFIO-USER虚拟化方案实现介绍

原创 Liu Changpeng DPDK与SPDK开源社区 2022-09-14 12:00 发表于云南

◇ 前言

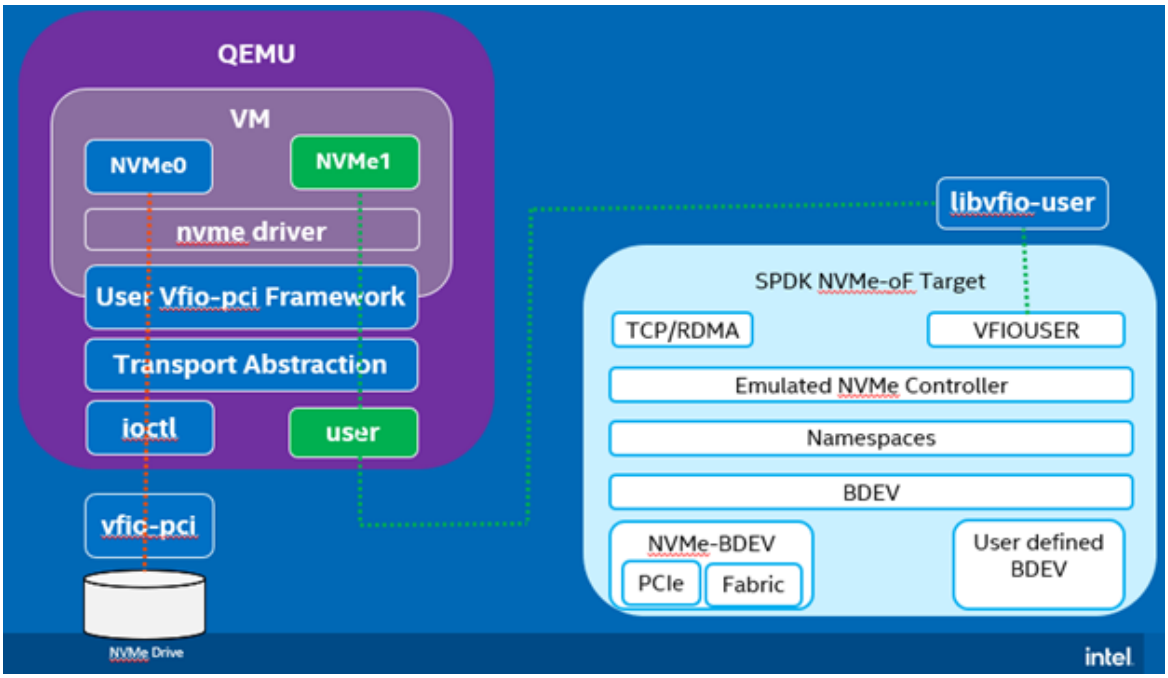
我们大家可能对SPDK VHOST-USER方案已经非常熟悉了，SPDK VHOST-USER给虚拟机提供virtio blk和scsi块设备，并且通过VM和SPDK VHOST进程间的共享内存机制高效的处理数据，今天我们介绍一种新的VFIO-USER虚拟化方案，通过该方案我们可以给虚拟机提供NVMe以及virtio blk和scsi接口，并且和VHOST-USER类似，通过共享内存机制提供高效数据处理。下面我们通过这篇文件来看下VFIO-USER的介绍。

◇ 正文

Virtual Function I/O(VFIO) 是Linux基于IOMMU实现的软件框架，VFIO可以直接使用Virtualization Technology for Directed I/O (VT-d)提供的DMA和中断重映射功能，对用户态应用程序提供简洁的ioctl接口，典型的一个应用是VMM (Virtual-Machine Monitor，如QEMU, Cloud-Hypervisor)，VMM的vfio-pci驱动可以直接利用VFIO ioctl接口安全的透传物理I/O设备给虚拟机 (Virtual Machine)；另外一个场景是可以用VFIO提供的接口实现用户态设备驱动，SPDK/DPDK都是利用该接口实现用户态的设备驱动，如NVMe和网卡的用户态驱动。

VFIO软件框架的最大特点是无关具体I/O设备的类型，VFIO-USER受到了VFIO和VHOST-USER的启发，在用户态实现了一套client/server library，client library跟现有的VFIO ioctl接口类似，主要提供访问PCI设备的接口，而在server library则提供PCI设备访问的模拟；VFIO-USER client library和server library通过Unix Domain Socket通信，VFIO通过ioctl把控制消息发送给内核VFIO驱动模块，而VFIO-USER则通过socket来发送类似的控制消息给另外一个进程。VMM可以利用client library给VM提供PCI设备访问接口，从VM的角度看，VFIO-USER提供的PCI设备直通和真实物理设备的直通并没有区别，而在实现层面，PCI设备的模拟是在另外一个进程实现的。

为了方便理解，我们先来看下QEMU VFIO PCI直通和VFIO-USER一起使用的例子，在这个例子中，本地的NVMe SSD已经绑定到内核的vfio-pci驱动模块，由SPDK模拟的NVMe SSD呈现的是一个Unix Domain socket文件，我们可以同时把这2个设备通过QEMU的vfio-pci驱动框架传递给VM，在VM中可以呈现出2个不同的NVMe控制器，一个是PCI直通过来的，另一个是通过SPDK模拟的。



这里我们不去对内核的vfio-pci驱动和原理进行分析，我们主要来看下VFIO-USER client library提供了哪些重要的接口抽象?

VFIO_USER_DEVICE_GET_INFO: 获取该设备有几个region (PCI BARs) 和支持多少个中断。

VFIO_USER_DEVICE_GET_REGION_INFO：获取某个region的信息，包括该region的能力flags，大小以及是否支持mmap访问，举个例子，我们可以在server端定义某个PCI设备的配置空间region为4K大小，不支持mmap。

VFIO_USER_REGION_READ/WRITE: 读写某个PCI region，如读写PCI配置空间。

VFIO_USER_DMA_MAP/UNMAP：map/unmap client端进程的内存空间，例如client用户是VMM的话，VMM利用map消息注册VMM本身内存文件的file descriptor到server进程，该步骤是为了实现client到server端的数据处理zero-copy特性。

其中对VFIO_USER_XX消息来讲，大部分消息都有相应的内核ioctl控制命令可以一一对应。

这里我们先稍微总结下，VFIO-USER client library是基于PCI设备提供的访问接口，它并不关心该PCI设备的类型是NVMe还是virtio，而VFIO-USER server端的实现，在做定义的时候就需要明确所模拟的PCI设备的类型是什么，然后根据该类型设备的规格实例化该设备。

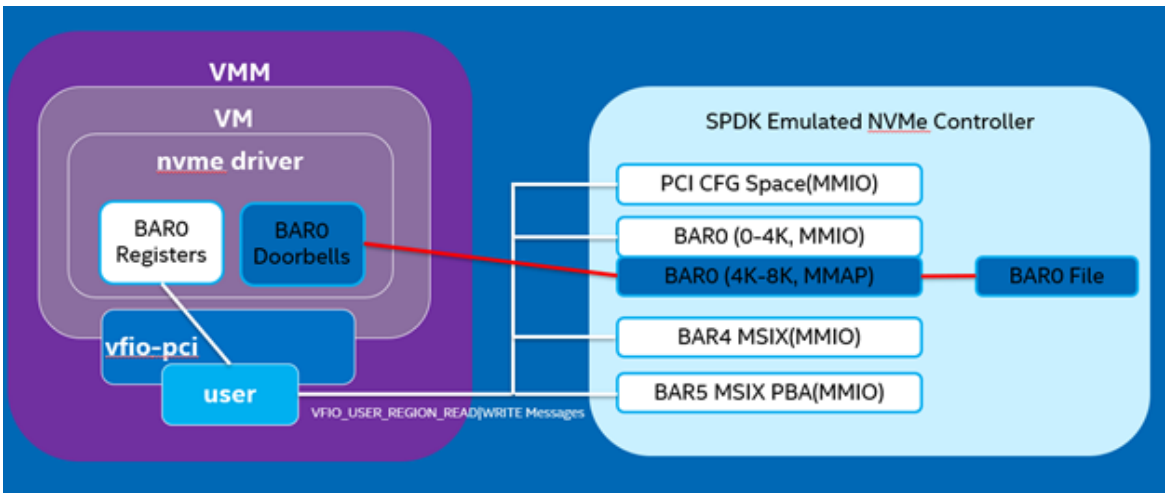
接下来我们看下**SPDK VFIO-USER server端**是如何基于VFIO-USER实例化NVMe设备的。

那么VFIO-USER server端的主要是对PCI设备的定义以及PCI设备类型的模拟，具体到我们今天的例子，针对NVMe设备主要的PCI定义部分包括：

1. 定义PCI配置空间大小为4KB，读写访问方式为MMIO，VFIO client针对MMIO访问是通过socket message来进行的，即client library的VFIO_USER_REGION_READ/WRITE 消息。
2. 定义PCI BAR0大小为8KB，前4KB访问方式为MMIO，后4KB方式为MMAP和MMIO，取决于用户的选择，当BAR空间配置为MMAP属性时，VM内部驱动对该区域的访问不会造成VM_EXIT。
3. 然后定义NVMe的MSIX TABLE和MSIX Pending Bit Array分别在BAR4和BAR5，大小是根据中断数目计算的。
4. 最后需要配置该PCI设备CLASS ID/VENDOR ID/DEVICE ID信息。

然后我们先看下总体的软件框图：

其中对NVMe设备的模拟是基于SPDK NVMeoF实现的，VMM可以是QEMU或者Cloud-Hypervisor，其中QEMU的VFIO-USER client library还没有合入到主分支，Cloud-Hypervisor已经包含VFIO-USER client library的支持了。



Client和Server端大致的交互流程如下：

1. Server端实例化一个NVMe设备并绑定到指定的Unix Domain Socket文件
2. Server端监听该Socket文件等待有效连接
3. Client端启动VM生成一个PCI设备并建立和Socket文件的连接
4. Client端通过VFIO_USER_DEVICE_GET_INFO到Server端获取PCI设备的信息
5. Client端通过VFIO_USER_DEVICE_GET_REGION_INFO到Server端获取PCI BAR信息
6. VM内核或者BIOS根据CLASS信息给该PCI设备加载NVMe驱动
7. NVMe驱动读写BAR0 Registers会触发VM_EXIT，然后Client端基于（region, offset, len）发送VFIO_USER_REGION_READ/WRITE到server端
8. NVMe驱动写BAR0 Doorbells的值会直接反应到Server端的BAR0 File里面，并不会触发VM_EXIT

Server端第三方库libvfio-user目前已经合入到QEMU主分支，而大部分用户需要的其实是Client端的QEMU支持；SPDK VFIO-USER Server也是直接利用该库来处理大部分的通用PCI层面的语义实现，这样SPDK就可以专注于实现设备本身的规格定义，如上面所以，我们只需要实现BAR0访问的语义即可，而该语义实现是定义在NVMe Spec中有详细描述。

最后我们以**Cloud-Hypervisor**为例子展示下如何使用SPDK VFIO-USER。

选择的测试版本是cloud-hypervisor v25.0-145-g487458c9和SPDK v22.09-pre git sha1 44cbea402。

编译和准备image的过程这里就不细说了，https://github.com/cloud-hypervisor/cloud-hypervisor网页上都有详细的说明，我这里选择的自编的内核，参考”Custom kernel and disk image”即可。

SPDK 侧的命令如下：

```
scripts/rpc.py nvmf_create_transport -t VFIOUSER
scripts/rpc.py nvmf_create_subsystem -a nqn.2019-07.io.spdk:cnode0 -m 64
scripts/rpc.py bdev_null_create Null0 102400 512
scripts/rpc.py nvmf_subsystem_add_ns -n 1 nqn.2019-07.io.spdk:cnode0 Null0
scripts/rpc.py nvmf_subsystem_add_listener -t VFIOUSER -a "/var/run/nvme0"-s 0
nqn.2019-07.io.spdk:cnode0
```

Cloud-Hypervisor主要启动参数：”--kernel vmlinux.bin --disk path=focal-server-cloudimg-amd64.raw --cmdline "console=hvc0 root=/dev/vda1 rw" --cpus boot=4 --memory size=4G,shared=on,hugepages=on **--user-device socket=/var/run/muser/nvme0/cntrl**”

其中/var/run/nvme0/cntrl就是SPDK VFIO-USER server端创建的socket文件，用户也可以使用打了vfio-user client补丁的QEMU(https://github.com/oracle/qemu.git, branch vfio-user-irqmask2)来启动VM，主要参数是”**-device vfio-user-pci, x-msg-timeout=5000 ,socket=/var/run/nvme0/cntrl**”。

VFIO-USER相对于VHOST-USER最大的特点是支持任意的PCI设备类型，并且由于client library的实现相对简洁，减少用户后续的维护工作，而且由于实现简洁也相对减少了不安全的恶意攻击接口，目前该方案还在开发成熟过程中。目前SPDK主分支现在已经支持基于VFIO-