目 录

1 半虚拟化驱动

1.1 QEMU 模拟 I/O 设备的基本原理

模拟 I/O 设备的过程如下:

1. 客户机中的设备驱动程序发起 I/O 操作请求，KVM 模块中的 I/O 操作捕获代码会拦截这次 I/O 请求
2. I/O 操作捕获代码对 I/O 请求的信息处理后，将其放到 I/O 共享页，并通知用户控件的 QEMU 程序
3. QEMU 模拟程序获得 I/O 操作的具体信息后，交由硬件模拟代码来模拟出本次的 I/O 操作
4. 硬件模拟代码的模拟操作完成后，把结果放回到 I/O 共享页，并通知 KVM 模块的 I/O 操作捕获代码
5. 由 KVM 模块中的 I/O 操作捕获代码读取 I/O 共享页中的操作结果，并把结果返回到客户机中

1.2 virtio 的介绍

KVM 实现半虚拟化驱动的方式是采用 virtio 这个 Linux 上的设备驱动的那个标准框架。

virtio 由四层组成，为前端驱动层、virtio 层、transport 层和后端处理层。前端驱动层是客户机中的驱动程序模块，后端处理层是 QEMU 中的后端处理程序。而 virtio 层和 transport 层用于支持客户机和 QEMU 之间的通信。

1.2.1 virtio_balloon 的介绍

首先介绍一下 ballooning 技术。ballooning 技术可以在客户机运行时动态地调整它所占用的宿主机的内存资源，而不需要关闭客户机。这个技术实现了，当宿主机内存紧张时，可以请求客户机的部分内存，从而客户机就会释放其空闲内存。如果此时客户机空闲内存不足，可能还会回收部分使用中的内存。

KVM 中 ballooning 的工作过程如下:

1. KVM 发送请求到客户机操作系统，让其归还部分内存给宿主机。
2. 客户机操作系统中的 virtio_balloon 驱动接收到 KVM 的请求，然后使客户机中的内存气球膨胀，气球中的内存不能被客户机访问。

3. 客户机操作系统将气球中的内存还给 KVM，KVM 可以把气球中的内存分配到任何需要的地方。

使用如下命令即可使用 ballooning 技术：

```
-balloon virtio
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -balloon virtio
```

可以在 qemu monitor 中查看和设置客户机内存的大小，命令如下：

```
info balloon // 查看客户机内存占用量
balloon num // 设置客户机内存占用量为numMB
```

通过如下命令，可以在客户机中看到 balloon 技术的使用，如下图所示：

```
psd@scholes:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
```

1.2.2 virtio_net 的介绍

选择 KVM 网络设备时，使用 virtio_net 半虚拟化驱动可以提高网络吞吐量和降低网络延迟。

通过以下命令即可将客户机的网卡设备指定为 virtio 类型：

```
-net nic,model=virtio
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -net nic,model=virtio
```

以下命令可以将 virtio_net 的后端处理任务放到内核空间中执行，从而提高效率。如下所示：

```
-net tap,vhost=on
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -net nic,model=virtio -net tap,
    vhost=on
```

1.2.3 virtio_blk 的介绍

使用 virtio_blk 半虚拟化驱动可以提高访问块设备 I/O 的方法。

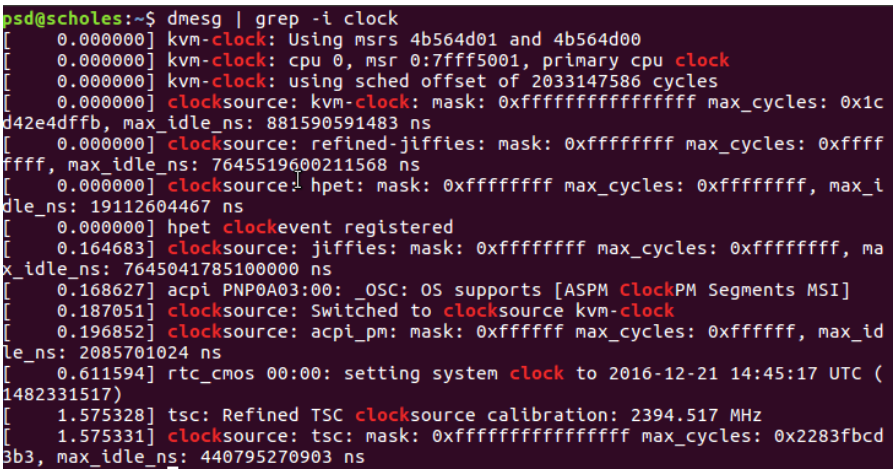
使用如下命令可以启用 virtio_blk 驱动：

```
file=filename,if=virtio
// 如，qemu-system-x86_64 -m 2048 -net nic file=ubuntu1604.img,if=virtio
```

1.2.4 kvm_clock 的介绍

使用 kvm_clock 半虚拟化时钟，可以为客户机提供精确的 system time 和 wall time，从而避免客户机时间不准确的问题。

使用 qemu 命令启动客户机时，已经将 kvm_clock 默认作为客户机的时钟来源。可以通过如下命令查看客户机中与时钟相关的信息，如下图所示：



2 PCI 设备直接分配

PCI 设备直接分配允许将宿主机中的物理 PCI 设备直接分配给客户机完全使用。Inte 定义的 PCI 设备直接分配技术规范称为 VT-d。

当 KVM 将宿主机的 PCI 设备附加到客户机时，客户机对该设备的 I/O 交互操作和实际的物理设备操作完全一样，不需要 KVM 的参与。

2.1 VT-d 环境配置

VT-d 环境配置包括以下几个方面：

1. 硬件支持和 BIOS 设置。需要在 BIOS 中将 VT-d 功能设置为“Enabled”状态。
2. 宿主机内核的配置。在配置内核时，需要配置如下几个 VT-d 相关的配置选项：

```
CONFIG_IOMMU_SUPPORT=y
CONFIG_DMAR_TABLE=y
CONFIG_INTEL_IOMMU=y
CONFIG_INTEL_IOMMU_DEFAULT_ON=y
CONFIG_IRQ_REMAP=y
CONFIG_PCI_STUB=m
```

可以通过以下两个命令查看宿主机是否支持 VT-d：

```
dmesg | grep DMAR -i
dmesg | grep IOMMU -i
```

3. 绑定设备到 pci_stub 驱动，从而对需要分配给客户机的设备进行隐藏，使得宿主机和其他客户机无法使用该设备。命令如下所示：

```
modprobe pci_stub // 加载pci_stub驱动
// 通过下一行命令得到设备的domain:bus:slot.function vendor_ID:device_ID
lspci -Dn -s BDF
// 绑定设备到pci_stub驱动
echo -n "vendor_ID device_ID" > /sys/bus/pci/drivers/pci-stub/new_id
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/domain:bus:slot.
    function/driver/unblind
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/pci_stub/blind
```

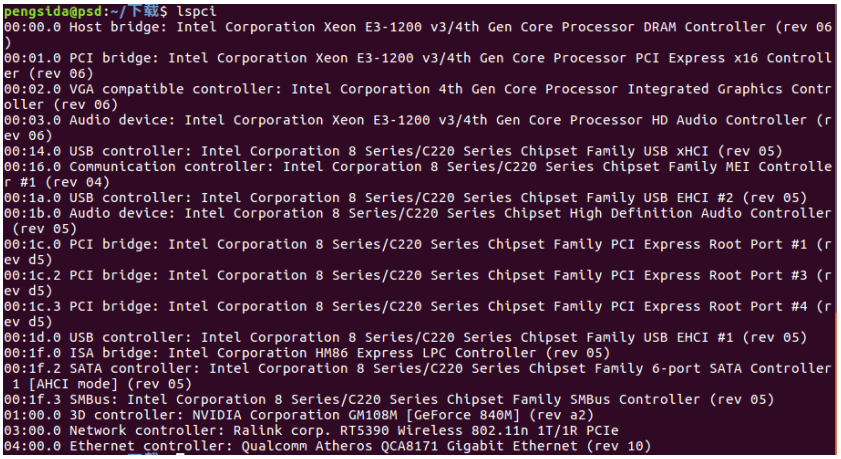
4. 使用 qemu 命令分配设备给客户机，命令如下所示：

```
-device pci-assign,host=BDF
// 如，qemu-system-x86_64 ubuntu1604.img -device pci-assign,host=08:00.0
```

5. 当客户机不需要使用该设备后，让宿主机重新使用该设备命令如下：

```
echo -n "vendor_ID device_ID" > /sys/bus/pci/drivers/domain:bus:slot.function
    /driver/new_id
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/pci_stub/unblind
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/domain:bus:slot.
    function/driver/blind
```

在绑定设备到 pci_stub 驱动和使用 qemu 命令分配设备给客户机两个步骤，主要需要知道设备的 BDF。可以通过 lspci 查看电脑所有设备的 BDF，每行设备信息前面的 bus:slot.function 就是设备的 BDF。如下图所示：



2.2 SR-IOV 技术

SR-IOV 技术实现了多个虚拟机能够共享同一个物理设备的资源，并且达到设备直接分配的性能。SR-IOV 有两个功能，如下所示：

1. 物理功能 (PF)，放在宿主机中配置和管理虚拟功能，它本身也可以作为一个普通的 PCI-e 设备使用。
2. 虚拟功能 (VF)，轻量级 PCI-e 功能。虚拟功能通过物理功能配置后，可以分配到客户机中作为独立功能使用。

可以通过如下命令查看设备是否具备 SR-IOV 的能力：

```
lspci -v -s BDF
```

在宿主机中，当加载支持 SR-IOV 技术的 PCI 设备的驱动时，可以加上相应的参数来指定启用多少个 VF。相关命令如下所示：

```
modprobe driver max_vfs=num
```

在已知设备 domain:bus:slot.function 的情况下，可以通过以下命令查看该设备的 VF：

```
ls -l /sys/bus/pci/devices/domain:bus:slot.function/virtfn*
```

3 热拔插

热拔插指的是可以在电脑运行时插上或拔除硬件。在 KVM 虚拟化环境中，在不关闭客户机的情况下，也可以对客户机的设备进行热拔插。

3.1 PCI 设备的热拔插

PCI 设备的热拔插需要以下几个方面的支持：

1. 硬件支持。现在的 BIOS 和 PCI 总线都支持热拔插。
2. 客户机操作系统支持，内核配置文件中需要有以下配置：

```
CONFIG_HOTPLUG=y
CONFIG_HOTPLUG_PCI_PCIE=y
CONFIG_HOTPLUG_PCI=y
CONFIG_HOTPLUG_PCI_FAKE=m
CONFIG_HOTPLUG_PCI_ACPI=y
CONFIG_HOTPLUG_PCI_ACPI_IBM=m
```

可以在 qemu monitor 中完成热拔插功能，比如要将 BDF 为 02:00.0 的 PCI 设备动态添加到客户机中，在 monitor 中的命令如下：

```
device_add pci-assign,host=02:00.0,id=mydevice
```

也可以将设备从客户机中动态移除，在 monitor 中的命令如下：

```
device_del mydevice
```

需要注意的是，如果要把宿主机中的 PCI 设备给客户机作为热拔插使用，需要绑定设备到 pci_stub 驱动，从而对需要分配给客户机的设备进行隐藏，使得宿主机和其他客户机无法使用该设备。

4 动态迁移

4.1 虚拟化环境中的迁移

在虚拟化环境中的迁移分为静态迁移和动态迁移。

静态迁移有两种的实现方式:

- 一种实现方式是，关闭客户机后，将其硬盘镜像复制到另一台宿主机上然后恢复启动起来。
- 另一种实现方式是，两台宿主机共享存储系统，只需要在暂停客户机后，复制其内存镜像到另一台宿主机中恢复启动。

可以通过以下两个步骤实现静态迁移:

1. 在源宿主机上某客户机的 qemu monitor 中使用“savevm my_tag”命令来保存一个完整的客户机镜像快照。
2. 在源宿主机中关闭或暂停该客户机。
3. 将该客户机的镜像文件复制到另外一台宿主机中，在其 qemu monitor 中用“loadvm my_tag”命令来加载保存快照时的客户机快照。

动态迁移指的是在保证客户机上应用服务正常运行的同时，让客户机在不同的宿主机之间进行迁移。一个成功的动态迁移，需要保证客户机的内存、硬盘存储、网络连接在迁移到目的主机后依然保存不变，而且迁移过程的服务暂停时间较短。

4.2 动态迁移的应用场景

1. 负载均衡。当一台物理服务器的负载较高时，可以将其上运行的客户机动态迁移到负载较低的宿主机服务器中。
2. 解除硬件依赖。当系统管理员需要在宿主机上升级、添加或移除某些硬件设备时，可以将该宿主机上运行的客户机动态迁移到其他宿主机上。
3. 节约能源。可以将宿主机上的客户机动态迁移到几台服务器上，而某些宿主机上的客户机完全迁移走后，就可以将其关闭电源，从而省电。
4. 实现客户机地理位置上的远程迁移。

4.3 KVM 动态迁移原理

4.3.1 基于共享存储系统的动态迁移的原理

当源宿主机和目的宿主机共享存储系统时，只需要通过网络发送客户机的 vCPU 执行状态、内存中的内容和虚拟设备的状态到目的主机上。具体迁移过程如下所示：

1. 在客户机在源客户机运行的同时，将客户机的内存页传输到目的主机上。
2. KVM 会监控并记录下迁移过程中所有已经被传输的内存页的任何修改。
3. 当内存数据量传输完成时，KVM 会关闭源宿主机上的客户机，然后将剩余的数据量传输到目的主机上去。
4. 当所有内存内容传输到目的宿主机后，就可以在目的宿主机上恢复客户机的运行状态。

需要注意的是，如果目的主机上缺少一些配置，那么客户机就无法正常运行。比如，在原宿主机上有给客户机配置好网桥类型的网络，但是目的主机没有网桥配置，那么迁移后的客户机就会网络不通。

还有一种情况就是，如果内存中数据被修改的速度大于 KVM 能够传输的内存速度时，动态迁移就无法完成。

4.3.2 动态迁移的注意事项

- 共享存储在源宿主机和目的宿主机上的挂载位置必须完全一致。
- 为了提高动态迁移的成功率，尽量在同类型 CPU 的主机上面进行动态迁移。
- 64 位的客户机只能在 64 位宿主机之间迁移，而 32 位客户机可以在 32 位宿主机和 64 位宿主机之间迁移。
- 动态迁移的源宿主机和目的宿主机对 NX 位的设置必须相同。
- 在目的宿主机上不能有与被迁移客户机同名的客户机存在。
- 目的宿主机和源宿主机的软件配置应该尽可能相同。

4.3.3 实现基于共享存储系统的动态迁移

动态迁移的实现如下所示：

1. 在源宿主机挂载 NFS 上的客户机镜像，并启动客户机。命令如下所示：

```
// 挂载客户机镜像
mount my-nfs:/rw-images/ /mnt/
// 启动客户机
qemu-system-x86_64 /mnt/ubuntu1604.img -smp 2 -m 2048 -net nic -net tap
```

2. 在目的宿主机上挂载 NFS 上的客户机镜像，并启动一个客户机用于接受动态迁移过来的内存内容。需要注意的是共享存储在源宿主机和目的宿主机上的挂载位置必须完全一致。命令如下所示：

```
// 挂载客户机镜像
mount vt-nfs:/rw-images/ /mnt/
// 启动客户机
qemu-system-x86_64 /mnt/ubuntu1604.img -smp 2 -m 2048 -net nic -net tap -
    incoming tcp:0:6666
```

参数“-incoming tcp:0:6666”表示在 6666 端口建立一个 TCP Socket 连接，用于接受来自源主机的动态迁移的内容，其中“0”表示允许来自任何主机的连接。

3. 在源宿主机的客户机的 qemu monitor 中使用如下命令进入动态迁移的流程：

```
migrate tcp:vt-snb9:6666
```

“vt-snb9”是目的宿主机的主机名，tcp 协议和 6666 端口号需要与目的宿主机上 qemu-kvm 命令行的“-incoming”参数中的值保持一致。

4.3.4 实现使用相同后端镜像文件的动态迁移

5 嵌套虚拟化

6 KSM 技术

7 KVM 安全

8 QEMU 监控器

9 qemu-kvm 命令行参数

10 迁移到 KVM 虚拟化环境