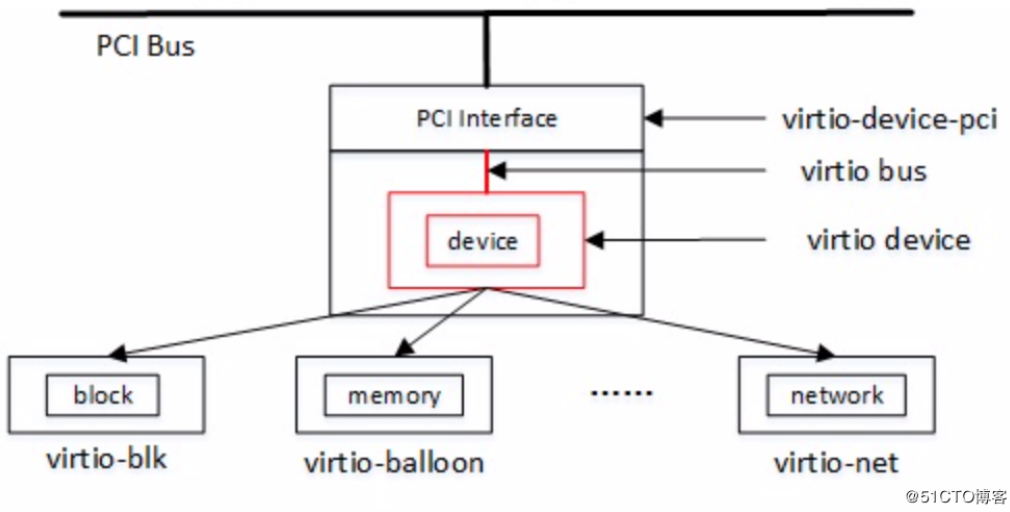
原创

virtio分析

老张的技术 关注 2018-12-28 20:09:37 8977人阅读 0人评论

virtio

virtio是一个通用的io虚拟化框架，hypervisor通过他模拟出一系列的虚拟化设备，并使得这些设备在虚拟机内部通过api调用的方式变得可用。它为客户机提供了一个高效访问块设备的方法。它包含4个部分：前端驱动、后端驱动、vring及通信间统一的接口。与其他的模拟io方式对比，virtio减少了虚拟机的退出和数据拷贝，能够极大地提高IO性能。计算机中存在不同的总线标准，而virtio采用的是pci总线（当然也可以用其他总线来实现）。每一个virtio设备就是一个pci设备。



virtio-blk的后端初始化

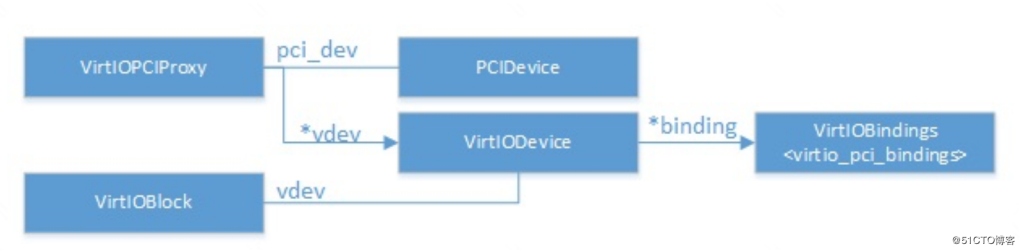
virtio-blk代码包保存在hw/virtio-pci.c和hw/virtio-blk.c中，通过如下函数对virtio_blk进行初始化。主要的初始化函数是virtio_blk_init_pci。这里定义了设备的信息。

```
static void virtio_blk_class_init(ObjectClass *klass, void *data)
{
    DeviceClass *dc = DEVICE_CLASS(klass);
    PCIDeviceClass *k = PCI_DEVICE_CLASS(klass);
    k->init = virtio_blk_init_pci;  //virtio-blk初始化函数
    k->exit = virtio_blk_exit_pci;
    k->vendor_id = PCI_VENDOR_ID_REDHAT_QUMRANET;   //设备厂商号，所有的virtio设备都为0x1af4
    k->device_id = PCI_DEVICE_ID_VIRTIO_BLOCK;      //设备号
    k->revision = VIRTIO_PCI_ABI_VERSION; //virtio ABI版本号
    k->class_id = PCI_CLASS_STORAGE_SCSI;
    dc->reset = virtio_pci_reset;
    dc->props = virtio_blk_properties;  //virtio-blk设备所支持的特征
}

static TypeInfo virtio_blk_info = {
```

```
    .class_init  = virtio_blk_class_init,    //virtio-blk设备类型初始化函数
};
```

virtio_blk后端数据结构如下



PCIDevice：表示一个pci设备

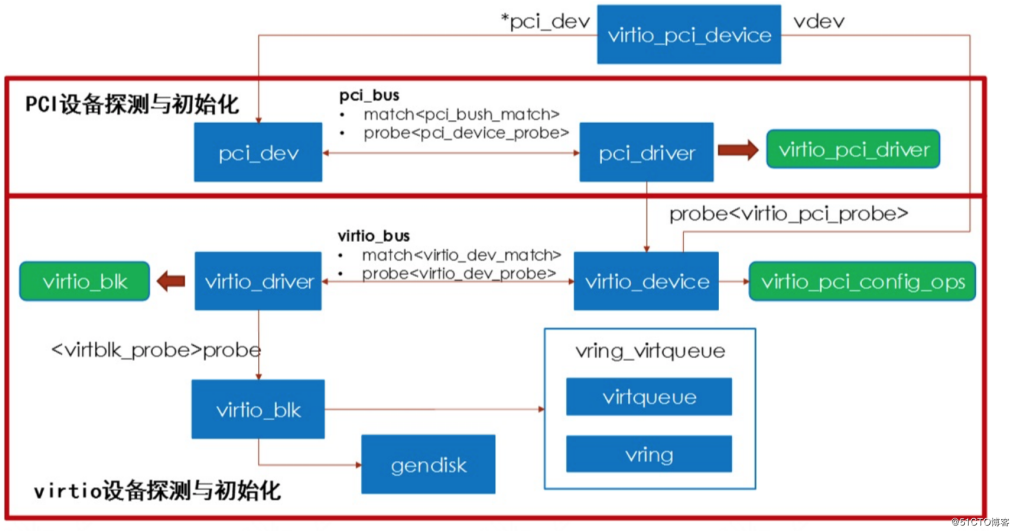
VirtIODevice：表示一个virtio设备

VirtIOBlock：表示一个virtio块设备

VirtIOBingdings：通用配置和处理函数集合

VirtIOPCIProxy：一个框架，奖virtio设备和pci设备关联起来

virtio-blk的前端初始化



virtio-blk首先是一个pci设备，初始化主要分两个阶段：pci设备初始化和设备初始化

以下是它初始化的几个阶段：

- PCI设备探测和初始化

虚拟机启动时，bios和系统会扫描pci总线，看看上面有没有挂载的pci设备。如果有，则会创建一个pci_dev结构。一个pci设备用一个pci_dev数据结构表示，创建之后会用pci设备配置空间信息来填充pci_dev，然后调用device_register来将其注册到pci总线上。PCI总线的match和probe函数根据pci_dev数据结构中的Vendor ID和Device ID将设备与注册在PCI总线上的驱动进行匹配，进而匹配到了所有virtio设备所共用的PCI驱动virtio_pci_driver。

```
static struct pci_device_id virtio_pci_id_table[] = {
    { 0x1af4, PCI_ANY_ID, PCI_ANY_ID, PCI_ANY_ID, 0, 0, 0 },
    { 0 },
```

```
    .id_table      = virtio_pci_id_table, //驱动所支持的设备ID信息
    .probe         = virtio_pci_probe,//探测函数(负责PCI设备初始化和进一步的virtio设备探测)
    .remove        = virtio_pci_remove,//设备移除时的处理函数
#ifdef CONFIG_PM
    .driver.pm = &virtio_pci_pm_ops, //电源管理函数
#endif
};
```

- virtio设备的探测和初始化

virtio_pci_driver是该阶段的关键函数，具体流程如下

virtio_pci_probe函数:

1. 分配struct virtio_pci_device数据结构（内嵌struct virtio_device）vp_dev，并初始化vp_dev相关数据
    1. vp_dev->vdev.config = &**virtio_pci_config_ops**;
    2. 设置vp_dev->pci_dev=**pci_dev**，其中pci_dev为virtio_pci_probe函数传入，而该函数是被PCI总线设备探测时调用
2. vp_dev->ioaddr = **pci_iomap**(pci_dev, 0, 0)：映射virtio设备（PCI设备）的bar0
3. 设置vp_dev->vdev.id.vendor和device为PCI设备的subsystem_vendor和subsystem_device
4. register_virtio_device：注册virtio设备
    - ida_simple_get
    - device_register：引起总线寻找一个匹配的驱动
        - device_initialize
        - device_add
            - bus_probe_device
                - device_attach
                    - bus_for_each_drv：针对设备所在总线上所注册的所有驱动，依次调用__device_attach函数
                        - __device_attach<dev,drv>：首先调用driver_match_device进行dev和drv的匹配，如果匹配成功则继续调用**driver_probe_device**函数，否则返回上层bus_for_each_drv循环，继续匹配总线上的下一个驱动
                            - driver_match_device：调用**drv->bus->match**对dev和drv进行匹配，如virtio_bus对应**virtio_dev_match**函数，如果匹配则返回1，否则返回0
                                - virtio_dev_match
                            - driver_probe_device<der,drv>
                                - really_probe
                                - 调用**drv->bus->probe函数**（如vitio设备对应virtio_bus，对应**virtio_dev_probe**）

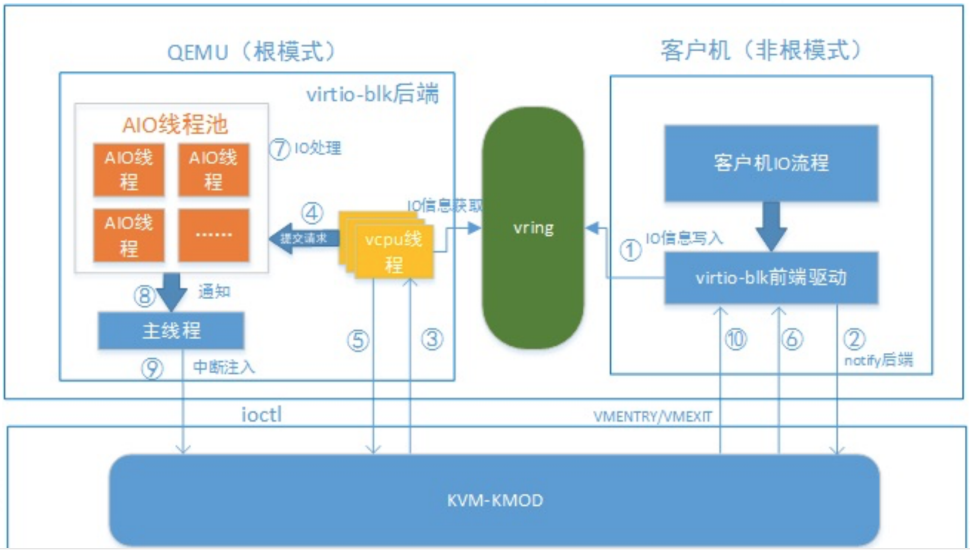
通信区域初始化

虚拟机与物理机的通信通过vring来实现数据交互，这之间存在一种io的通信机制。

- 主机通知客户机是通过注入中断来实现，虚拟设备连在模拟的中断控制器上，有自己的中断线信息，PCI设备的中断信息会被写入该设备的配置空间
- 客户机通知主机是通过virtio读写内存来实现的。

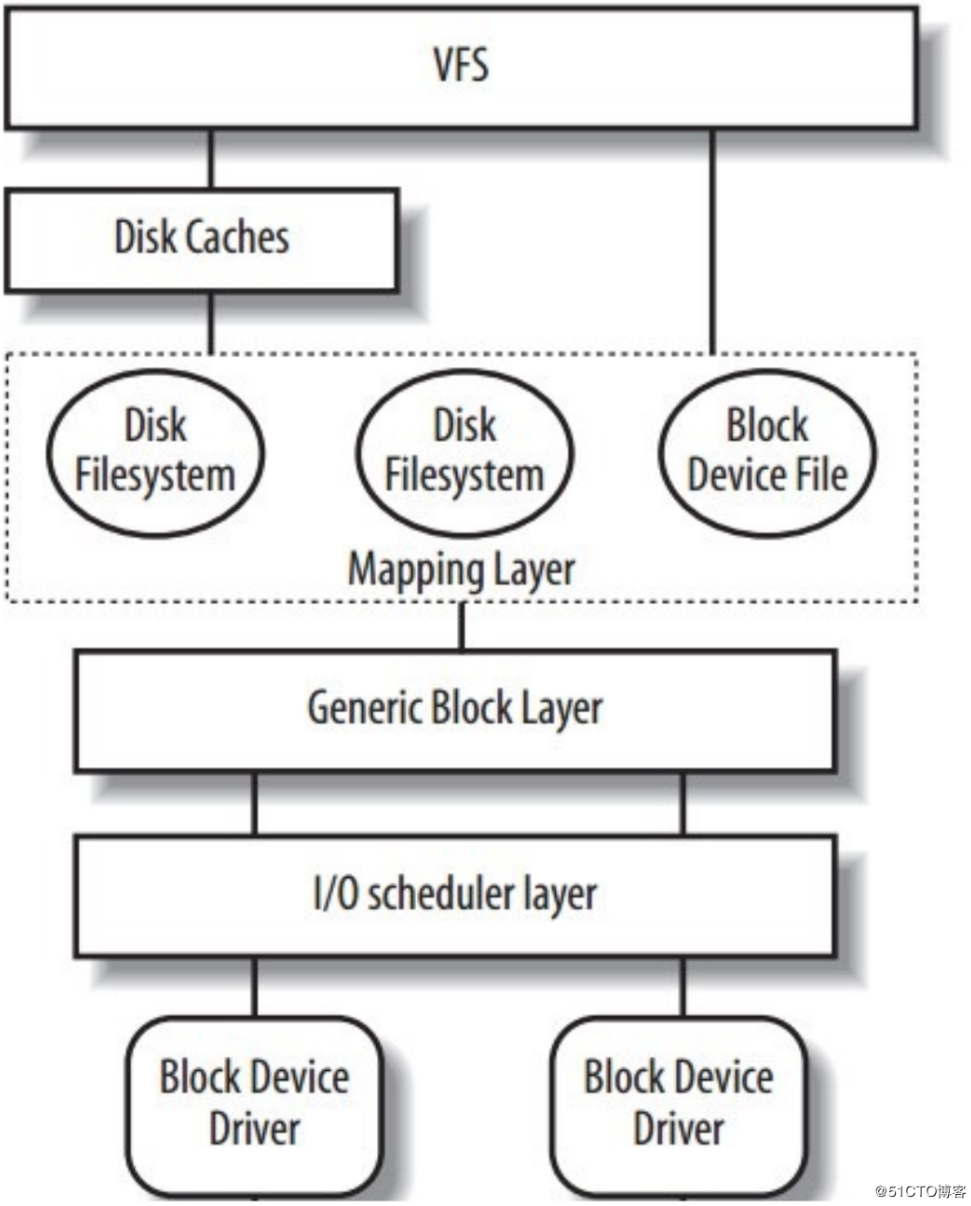
上面第二条分有两类：MMIO和PIO。MMIO是通过mmap()像写内存一样读写虚拟设备，比如内存。PIO（就是通常意义上的io端口）通过hypervisor捕获设备io来实现虚拟化。两者的区别是：MMIO是通过内存的异常来进行，PIO则是通过io动作的捕获。

virtio工作流程



1. 前端驱动读取io请求放入vring
2. 前端通过notify通知机制通知后端驱动处理io
3. notify操作使vcpu执行线程退出到qemu应用层，其从vring中获取客户机io请求信息，将请求线程放入aio线程池，然后vcpu线程的处理流程重新返回到客户机
4. aio线程处理完成后，通知主线程，并向客户机注入中断说明其已完成io操作
5. 客户机相应中断，并获取io请求结果和处理信息，接着继续向上层返回结果

客户机io请求流程



1、读写操作通过系统调用进入到操作系统内核层，首先到达VFS层

2、VFS层继续向下层传递请求，如果页高速缓存命中，而文件又不是直接读写，则IO请求在页高速缓存得到处理

3、如果没有页高速缓存或者页高速缓存MISS，则进入到Mapping Layer（映射层），在这一层主要是根据文件系统信息，解决文件偏移量与块设备中的的逻辑块号的映射，并根据映射将请求下发到Generic Block Layer（通用块层）

4、在通用块层，IO读写请求由struct bio表示，通用块层继续将请求下发到IO Scheduler Layer（IO调度

5、在IO调度层，上层传递下来的bio将根据类型、以及逻辑块是否靠近等因素进行调度，最终形成一个req（struct request类型），req将被链接到设备的request_queue中

6、IO调度层继续将请求下发，请求到达了块设备驱动，块设备驱动从request_queue中取下一个个请求进行处理。

虚拟化　内核　virtio

1　收藏　分享

上一篇：cgroup底层研究　下一篇：tcpdump原理

老张的技术

19篇文章，21W+人气，9粉丝

红帽认证架构师、OCM、学者

关注

提问和评论都可以，用心的回复会被更多人看到和认可

Ctrl+Enter 发布　取消　发布

推荐专栏　更多

VMware vSAN中小企业应用案例

掌握VMware超融合技术

共41章 | 王春海　订 阅

￥51.00　436人订阅

网工2.0晋级攻略 ——零基础入门Python/A…

网络工程师2.0进阶指南

共30章 | 姜汁啤酒　订 阅

￥51.00　1952人订阅

猜你喜欢

在线客服

nfs调优思路

运维自动化-Ansible (一)

windows10用WMware安装Linux虚拟机详细步骤

oom机制分析及对应优化策略

自动化运维工具Ansible详细部署

VMware Horizon View 7 安装部署