Lab5-part2：内存虚拟化

问题

1. 请阐述 x86 中 MMIO 与 PIO 两种 I/O 方式的特点（参考教材 4.2.2 节）

   MMIO：

   MMIO将I/O设备映射到物理地址空间的高地址部分，CPU可以通过内存访问指令进行MMIO，不需要IN/OUT等设备访问指令。

   在虚拟化时，Hypervisor不会将虚拟机中MMIO所在的物理地址范围映射到主机的物理地址空间，这样当客户机发起MMIO时，会触发缺页异常，产生VM-Exit，之后便可交由相关处理函数模拟；与PIO不同，MMIO相关的处理函数能够处理一片内存区域发起的MMIO访问，大大减少了处理函数的数量，降低了对内存的占用。

   PIO：

   在x86架构中，PIO需要通过IN/OUT、INS/OUTS等指令访问与设备寄存器相关的I/O端口。

   在虚拟化时，IN/OUT、INS/OUTS被设定为敏感指令，会触发VM-Exit，陷入Hypervisor；设备模型在初始化阶段会将客户机涉及的PIO处理函数在Hypervisor中进行注册，并以数组的形式保存函数的指针。Hypervisor根据截获信息匹配PIO处理函数，并交由相关函数进一步模拟。

2. 请简要描述 virtio 设备在进行 I/O 操作时的工作原理，这样的半虚拟化架构有什么优点？（参考教材 4.2.3 节）

   工作原理：

   半虚拟化情况下，虚拟机使用专门的前端驱动程序，通过特定数据传输接口与特权虚拟机或Hypervisor中的后端设备交互，后端设备则通过物理驱动程序完成对外设的访问。virtio则为半虚拟化提供了统一的前后端设备接口标准，这里一PCI设备为例，将virtio的本虚拟化大致分为virtio前端初始化和virtio前后端交互。

   virtio前端初始化过程可以分为virtio-pci设备探测和驱动加载、virtio设备初始化和驱动加载两个阶段。在第一个阶段，虚拟机启动时，虚拟BIOS和客户机OS会探测virtio-pci设备并注册到PCI总线，之后，根据一个由pci_device_id组成的id_table数组匹配PCI设备和驱动。在第二个阶段，初始化相应的virtio设备和virtio总线，将virtio设备注册到virtio总线，匹配virtio设备与virtio驱动，读取virtio-pci配置空间中关于virtio设备配置的内容，初始化virtqueue和对应的物理设备。

   virtio前后端交互主要由virtqueue机制实现。前端驱动根据传输协议将封装后的I/O请求放入virtqueue，并向后端发送通知；后端设备从virtqueue中接收前端驱动发出的I/O请求的基本信息，然后调用相关函数完成I/O操作，最后向客户机中的前端驱动发起中断。virtqueue则是由vring实现，关于vring的描述符表、可用的描述符环和已用描述符环的作用不过多展开，详见4.2.3节。

   优点：

   1. 半虚拟化架构解决了设备模拟中由于频繁的上下文切换导致的I/O性能大幅下降问题，vring在虚拟机和QEMU之间引入一段共享的环形缓冲区作为前后端数据通信的载体使得可以一次处理多个I/O请求，提高了每次I/O传递的数据量，并且显著减少了上下文切换的次数；

2. virtio为半虚拟化提供一个统一的前后端设备接口标准；
3. virtio作为一个通用半虚拟化框架，可以在不同类型设备总线之上实现。

3. 请调研设备枚举过程并根据你的理解回答设备是如何被发现的（参考书籍的 4.2 节，以及 UEFI 相关内容）

   设备枚举过程与客户机发现virtio设备的过程类似，这里也以PCI设备为例。

   首先，在设备启动时，BIOS/UEFI会检测PCI总线上的设备，并在OS启动时传递这些信息，OS随后会对这些设备进行枚举；

   之后，OS通过每个PCI设备的配置空间识别和访问PCI设备，该过程主要会确定设备号、厂商号、主机访问PCI设备的方式等信息，同时，由于PCI配置空间的初始值是由厂商预设在设备中的，可能造成不同PCI设备所映射地址空间之间的冲突，OS会在发现全部PCI设备后，重新为每个PCI BAR分配地址，然后写入PCI配置空间；

   最后，设备会根据PCI BAR建立端口和物理地址与自身设备接口之间的映射，这样，CPU便能通过端口号或者物理地址访问外设；OS会匹配相应的驱动程序，并将其加载到内存中以供设备使用。

实验

1. 在 QEMU 中添加模拟的 edu 设备

```
[  369.899308] length 100000
[  369.899327] config 0 34
[  369.899337] config 1 12
[  369.899349] config 2 e8
[  369.899359] config 3 11
[  369.899370] config 4 3
[  369.899383] config 5 1
[  369.899393] config 6 10
[  369.899404] config 7 0
[  369.899414] config 8 10
[  369.899425] config 9 0
[  369.899434] config a ff
[  369.899446] config b 0
[  369.899457] config c 0
[  369.899469] config d 0
[  369.899479] config e 0
[  369.899490] config f 0
[  369.899503] config 10 0
[  369.899513] config 11 0
[  369.899523] config 12 a0
[  369.899535] config 13 fe
[  369.899546] config 14 0
[  369.899558] config 15 0
[  369.899569] config 16 0
[  369.899579] config 17 0
[  369.899591] config 18 0
[  369.899602] config 19 0
[  369.899614] config 1a 0
[  369.899625] config 1b 0
```

```
[  369.899635] config 1c 0
[  369.899648] config 1d 0
[  369.899659] config 1e 0
[  369.899668] config 1f 0
[  369.899680] config 20 0
[  369.899690] config 21 0
[  369.899702] config 22 0
[  369.899713] config 23 0
[  369.899724] config 24 0
[  369.899736] config 25 0
[  369.899747] config 26 0
[  369.899759] config 27 0
[  369.899769] config 28 0
[  369.899779] config 29 0
[  369.899791] config 2a 0
[  369.899801] config 2b 0
[  369.899813] config 2c f4
[  369.899827] config 2d 1a
[  369.899844] config 2e 0
[  369.899857] config 2f 11
[  369.899867] config 30 0
[  369.899879] config 31 0
[  369.899890] config 32 0
[  369.899901] config 33 0
[  369.899913] config 34 40
[  369.899924] config 35 0
[  369.899933] config 36 0
[  369.899945] config 37 0
[  369.899956] config 38 0
[  369.899965] config 39 0
[  369.899977] config 3a 0
[  369.899988] config 3b 0
[  369.900000] config 3c b
[  369.900010] config 3d 1
[  369.900021] config 3e 0
[  369.900033] config 3f 0
[  369.900034] dev->irq a
[  369.900049] io 0 10000ed
[  369.900057] io 4 0
[  369.900064] io 8 0
[  369.900071] io c ffffffff
[  369.900078] io 10 ffffffff
[  369.900085] io 14 ffffffff
[  369.900093] io 18 ffffffff
[  369.900100] io 1c ffffffff
[  369.900107] io 20 0
[  369.900115] io 24 0
[  369.900122] io 28 ffffffff
[  369.900129] io 2c ffffffff
[  369.900136] io 30 ffffffff
[  369.900144] io 34 ffffffff
[  369.900151] io 38 ffffffff
[  369.900158] io 3c ffffffff
[  369.900166] io 40 ffffffff
```

```
[  369.900173] io 44 ffffffff
[  369.900180] io 48 ffffffff
[  369.900187] io 4c ffffffff
[  369.900195] io 50 ffffffff
[  369.900202] io 54 ffffffff
[  369.900208] io 58 ffffffff
[  369.900216] io 5c ffffffff
[  369.900224] io 60 ffffffff
[  369.900231] io 64 ffffffff
[  369.900239] io 68 ffffffff
[  369.900246] io 6c ffffffff
[  369.900254] io 70 ffffffff
[  369.900263] io 74 ffffffff
[  369.900271] io 78 ffffffff
[  369.900279] io 7c ffffffff
[  369.900287] io 80 0
[  369.900295] io 84 ffffffff
[  369.900303] io 88 0
[  369.900311] io 8c ffffffff
[  369.900319] io 90 0
[  369.900327] io 94 ffffffff
[  369.900408] irq_handler irq = 10 dev = 243 irq_status = 12345678
[  370.926634] receive a FACTORIAL interrupter!
[  370.926651] irq_handler irq = 10 dev = 243 irq_status = 1
[  371.949200] computing result 375f00
[  372.048749] receive a DMA read interrupter!
[  372.048755] irq_handler irq = 10 dev = 243 irq_status = 100
[  374.071849] receive a DMA read interrupter!
[  374.071869] irq_handler irq = 10 dev = 243 irq_status = 100
```

2. 为 edu 设备添加一项功能

该部分仿照阶乘的实现过程，为edu设备添加了一个非负数的自增运算功能，并可以设置在自增运算结束时触发虚拟机中断，自增线程与阶乘运算的实现基本完全一致，下面简单介绍

- 首先，我们创建了一个`edu_inc_thread`线程，用于进行自增运算，和阶乘类似，当`EDU_STATUS_INC_COMPUTING`标志位被设置时，开始进行自增运算，在代码的最后，根据`EDU_STATUS_IRQINC`标志位，判断是否需要向虚拟机发送中断

```
static void *edu_inc_thread(void *opaque)
{
    EduState *edu = opaque;

    while (1) {
        uint32_t val, ret = 0;

        qemu_mutex_lock(&edu->inc_thr_mutex);
        while ((qatomic_read(&edu->status) &
EDU_STATUS_INC_COMPUTING) == 0 &&
                        !edu->stopping) {
            qemu_cond_wait(&edu->inc_thr_cond, &edu-
```

```
>inc_thr_mutex);
        }

        if (edu->stopping) {
            qemu_mutex_unlock(&edu->inc_thr_mutex);
            break;
        }

        val = edu->inc;
        qemu_mutex_unlock(&edu->inc_thr_mutex);

        ret = ++val;

        qemu_mutex_lock(&edu->inc_thr_mutex);
        edu->inc = ret;
        qemu_mutex_unlock(&edu->inc_thr_mutex);
        qatomic_and(&edu->status, ~EDU_STATUS_INC_COMPUTING);

        smp_mb__after_rmw();

        if (qatomic_read(&edu->status) & EDU_STATUS_IRQINC) {
            qemu_mutex_lock_iothread();
            edu_raise_irq(edu, INC_IRQ);
            qemu_mutex_unlock_iothread();
        }
    }

    return NULL;
}
```

- 在`edu_mmio_write`中，我们同样在`0x20`中设置是否在计算完成时发送中断，在`0x10`中设置自增运算的值并唤起线程

```
case 0x10:
    if (qatomic_read(&edu->status) & EDU_STATUS_INC_COMPUTING) {
        break;
    }

    qemu_mutex_lock(&edu->inc_thr_mutex);
    edu->inc = val;
    qatomic_or(&edu->status, EDU_STATUS_INC_COMPUTING);
    qemu_cond_signal(&edu->inc_thr_cond);
    qemu_mutex_unlock(&edu->inc_thr_mutex);
    break;
case 0x20:
    if (val & EDU_STATUS_IRQFACT) {
        qatomic_or(&edu->status, EDU_STATUS_IRQFACT);
        /* Order check of the COMPUTING flag after setting
IRQFACT.  */
        smp_mb__after_rmw();
    } else {
```

```
            qatomic_and(&edu->status, ~EDU_STATUS_IRQFACT);
        }

        if (val & EDU_STATUS_IRQINC) {
            qatomic_or(&edu->status, EDU_STATUS_IRQINC);
            smp_mb__after_rmw();
        } else {
            qatomic_and(&edu->status, ~EDU_STATUS_IRQINC);
        }
        break;
```

- 在edu_mmio_read中，在0x10读取计算后的值

```
    case 0x10:
        qemu_mutex_lock(&edu->inc_thr_mutex);
        val = edu->inc;
        qemu_mutex_unlock(&edu->inc_thr_mutex);
        break;
```

- 最终结果如下，输入为0xa，输出为0xb

```
ubuntu@ubuntu:~/cloudos$ sudo dmesg
[  299.601810] receive a INCREMENT interrupter!
[  299.601827] irq_handler irq = 10 dev = 243 irq_status = 2
[  300.626168] computing result b
```