I/O 虚拟化

学号： N/A

姓名： N/A

专业： 计算机科学与技术

1 调研部分

1.1 x86 中 MMIO 与 PIO 的概念

MMIO 和 PIO 是 x86 架构中常见的两种 I/O 访问方式。MMIO 是指内存映射 I/O，把外设的端口映射到系统的物理内存地址空间，操作系统可以通过通用的内存读写指令来访问这些寄存器。而 PIO 是指端口 I/O，通过指定的 I/O 端口与外设进行数据交换，使用特殊的 I/O 指令 IN、OUT、INS、OUTS 来访问设备。

PIO 的设计简单直接，但是需要特殊指令完成，增加了 ISA 设计复杂性，而且端口资源有限，容易造成冲突。MMIO 则是通过内存映射的方式，可以利用现有的内存管理机制，但是会占用内存地址空间，增加了内存管理的复杂性。

在虚拟化场景中，PIO 和 MMIO 的工作方式也有所不同。对于 PIO，Hypervisor 只需要将 IN、OUT 等 I/O 指令设为敏感指令，就可以在虚拟机尝试 I/O 时触发 VM-Exit，交由 Hypervisor 进行处理。相较之下，在指令上 MMIO 的行为无法与普通的内存访问区分，因此需要特殊设置 MMIO 所映射的内存区域为始终缺页，如此 HyperVisor 可以通过缺页异常来拦截虚拟机对设备的访问。

1.2 设备枚举过程

设备枚举过程的核心是扫描 PCI 总线，识别并配置已安装的硬件。每个 PCI 设备通过一个唯一的配置空间进行标识，该配置空间包含设备的厂商 ID、设备 ID、类代码等信息。系统通过读取这些信息来确认设备的存在。在扫描过程中，操作系统或 BIOS 遍历每个可能的设备位置，读取该位置的配置空间来获取设备的基本信息，确认这个位置上是否存在有效的设备。

扫描从根桥开始，根桥负责管理总线上的所有设备。每个设备在配置空间中包含一个设备标识符，如果设备存在，根桥会返回设备的配置空间数据。设备可能有桥接功能，如果遇到桥接设备，系统会继续递归扫描桥接设备的子总线。

设备被发现后，操作系统会根据已识别的硬件信息分配资源，加载适当的驱动程序，解决端口冲突等问题，使设备能够正常工作。

2 实验目的

- 了解 I/O 虚拟化的基本概念
- 认识虚拟设备驱动的原理
- 在虚拟环境中访问虚拟设备

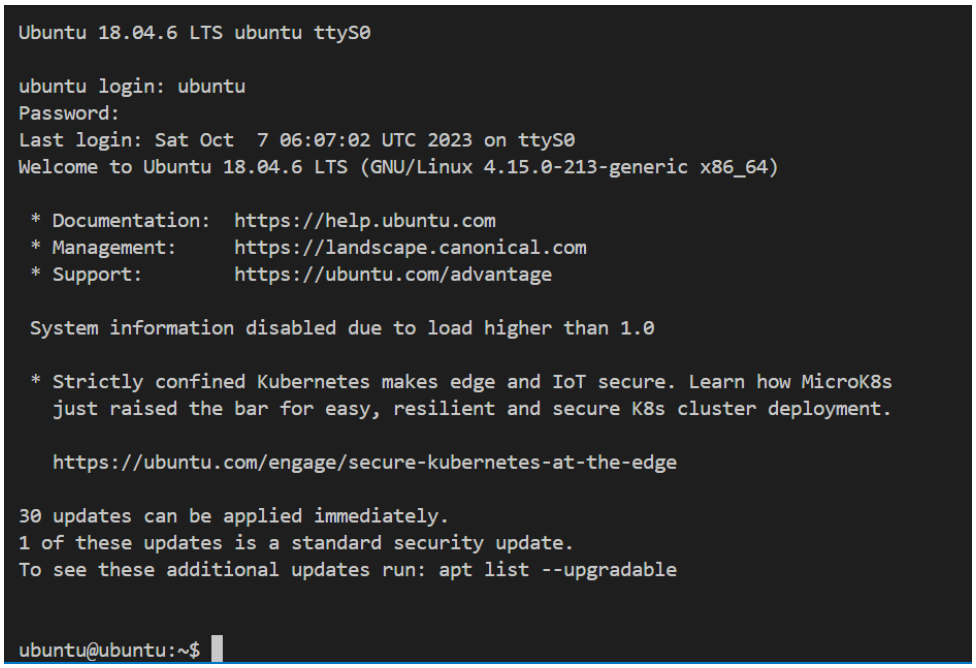
3 实验步骤

3.1 实验环境

- 操作系统：Ubuntu 22.04.3 LTS
- 内核版本：5.15.0-86-generic x86_64
- 编译环境：gcc version 11.4.0
- 编辑环境：Visual Studio Code

3.2 实验过程

首先启动 L2 虚拟机：



编译并加载 pci.ko 内核模块，使用 `lspci` 命令查看 PCI 设备信息：

```
ubuntu@ubuntu:~/Book/Chapter4$ 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: Device 1234:1111 (rev 02)
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 03)
00:04.0 Unclassified device [00ff]: Device 1234:11e8 (rev 10)
00:05.0 SCSI storage controller: Red Hat, Inc. Virtio block device
00:06.0 SCSI storage controller: Red Hat, Inc. Virtio block device
ubuntu@ubuntu:~/Book/Chapter4$
```

```
ubuntu@ubuntu:~/Book/Chapter4$ lspci -s 00:04.0 -vvv -xxxx
00:04.0 Unclassified device [00ff]: Device 1234:11e8 (rev 10)
        Subsystem: Red Hat, Inc. Device 1100
        Physical Slot: 4
        Control: I/O+ Mem+ BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Interrupt: pin A routed to IRQ 10
        Region 0: Memory at fea00000 (32-bit, non-prefetchable) [size=1M]
        Capabilities: <access denied>
        Kernel driver in use: edu_pci
00: 34 12 e8 11 03 01 10 00 10 00 ff 00 00 00 00 00
10: 00 00 a0 fe 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 f4 1a 00 11
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 01 00 00
```

查询/proc/devices 查看 edu 设备的主设备号，并创建设备节点：

```
ubuntu@ubuntu:~/Book/Chapter4$ cat /proc/devices | grep edu
245 pci_edu
ubuntu@ubuntu:~/Book/Chapter4$ sudo mknod /dev/edu c 245 0
ubuntu@ubuntu:~/Book/Chapter4$ ls /dev/edu
/dev/edu
```

运行测试程序，得到输出。输出内容见实验结果，随后 poweroff 关闭 L2 虚拟机。

3.3 实验结果

运行测试程序后，可以得到内核日志输出，这里省略了无关的 config 和 io 寄存器内容。

```
[  610.825700] length 100000
[  610.825817] config 0 34
[  610.825872] config 1 12
[  610.825928] config 2 e8
[  610.825983] config 3 11
[  610.829101] dev->irq a
[  610.829189] io 8 375f00
[  610.829349] io 20 80
[  610.829375] io 24 0
[  610.829790] io 60 ffffffff
[  610.829815] io 64 ffffffff
[  610.829975] io 80 40b00104
```

```
[  610.830024] io 88 40000
[  610.830078] io 90 4
[  610.832165] irq_handler irq = 10 dev = 245 irq_status =
12345678
[  611.840333] receive a FACTORIAL interrupter!
[  611.840336] irq_handler irq = 10 dev = 245 irq_status = 1
[  612.864167] computing result 375f00
[  612.964059] receive a DMA read interrupter!
[  612.964063] irq_handler irq = 10 dev = 245 irq_status = 100
[  614.988143] receive a DMA read interrupter!
[  614.988146] irq_handler irq = 10 dev = 245 irq_status = 100
```

4 实验分析

在 edu 设备的控制和数据寄存器中，只有如下一些地址是与本次实验有关的：

地址	名称	解释
config 0x00-0x03		记录了供应商 ID 和设备 ID
io 0x08	FACTORIA_VAL	用于写入阶乘计算的值或者读取计算结果
io 0x20	IO_FACTORIA_IRQ	写入以通知设备准备接受计算任务
io 0x24	IO_IRQ_STATUS	记录设备被中断的原因
io 0x60	IO_IRQ_RAISE	写入后设备主动发起中断
io 0x64	IO_IRQ_ACK	保存中断的回复
io 0x80	IO_DMA_SRC	用于写入 DMA 读取的源地址
io 0x88	IO_DMA_DST	用于写入 DMA 写入的目的地址
io 0x90	IO_DMA_CNT	用于写入 DMA 传输的字节数
io 0x98	IO_DMA_CMD	用于写入 DMA 命令的控制选项

在实验中按时间顺序梳理如下：

- 首先当 edu 模块被发现时，pci_probe 函数会被调用，此时模块会做注册设备、设置 MMIO 内存映射等工作，并注册了中断处理函数 irq_handler。
- 测试程序 test 运行后，首先通过 ioctl 调用了 PRINT_EDUINFO_CMD，打印了 edu 设备的内存信息。此处可以从 config 0x00-0x03 看到供应商 ID 0x1234，设备 ID 0x11e8。

- test 随后使用 SEND_INTERRUPT_CMD 让 edu 设备发起中断，此时 irq_handler 会被调用，打印出中断的原因，即 irq_status = 12345678。
- test 继续使用 FACTORIAL_CMD 计算阶乘，驱动会将计算数 0x0a 写入 FACTORIA_VAL，并写入 IO_FACTORIA_IRQ 通知设备开始计算。edu 设备计算完成后，驱动主动读取 FACTORIA_VAL 得到计算结果 0x375f00。
- test 最后调用了 DMA_WRITE_CMD 和 DMA_READ_CMD，驱动将相应的源地址、目的地址、字节数和控制字写入对应的地址，edu 设备会进行 DMA 传输，传输完成后设备发起中断，通过 irq_handler 打印出中断原因。

本次实验中，edu 设备的驱动程序实现了对设备的控制和数据传输，通过中断机制和 DMA 传输机制，实现了设备与驱动的交互。

附录

参考文献

- 教材 深入浅出系统虚拟化：原理与实践
- KVM API Documentation：https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt

原始输出

```
[  610.825700] length 100000
[  610.825817] config 0 34
[  610.825872] config 1 12
[  610.825928] config 2 e8
[  610.825983] config 3 11
[  610.826039] config 4 3
[  610.826099] config 5 1
[  610.826155] config 6 10
[  610.826211] config 7 0
[  610.826266] config 8 10
[  610.826319] config 9 0
[  610.826371] config a ff
[  610.826425] config b 0
[  610.826476] config c 0
[  610.826527] config d 0
```

```
[  610.826577] config e 0
[  610.826627] config f 0
[  610.826676] config 10 0
[  610.826725] config 11 0
[  610.826774] config 12 a0
[  610.826823] config 13 fe
[  610.826873] config 14 0
[  610.826923] config 15 0
[  610.826973] config 16 0
[  610.827022] config 17 0
[  610.827078] config 18 0
[  610.827127] config 19 0
[  610.827177] config 1a 0
[  610.827227] config 1b 0
[  610.827278] config 1c 0
[  610.827326] config 1d 0
[  610.827397] config 1e 0
[  610.827447] config 1f 0
[  610.827497] config 20 0
[  610.827545] config 21 0
[  610.827593] config 22 0
[  610.827644] config 23 0
[  610.827693] config 24 0
[  610.827743] config 25 0
[  610.827793] config 26 0
[  610.827842] config 27 0
[  610.827892] config 28 0
[  610.827939] config 29 0
[  610.827990] config 2a 0
[  610.828090] config 2b 0
[  610.828140] config 2c f4
[  610.828189] config 2d 1a
[  610.828238] config 2e 0
[  610.828284] config 2f 11
[  610.828333] config 30 0
[  610.828383] config 31 0
[  610.828429] config 32 0
[  610.828480] config 33 0
[  610.828530] config 34 40
[  610.828579] config 35 0
[  610.828629] config 36 0
```

```
[  610.828678] config 37 0
[  610.828746] config 38 0
[  610.828797] config 39 0
[  610.828848] config 3a 0
[  610.828899] config 3b 0
[  610.828949] config 3c b
[  610.828999] config 3d 1
[  610.829050] config 3e 0
[  610.829100] config 3f 0
[  610.829101] dev->irq a
[  610.829142] io 0 10000ed
[  610.829166] io 4 0
[  610.829189] io 8 375f00
[  610.829212] io c ffffffff
[  610.829237] io 10 ffffffff
[  610.829264] io 14 ffffffff
[  610.829291] io 18 ffffffff
[  610.829321] io 1c ffffffff
[  610.829349] io 20 80
[  610.829375] io 24 0
[  610.829404] io 28 ffffffff
[  610.829433] io 2c ffffffff
[  610.829456] io 30 ffffffff
[  610.829484] io 34 ffffffff
[  610.829517] io 38 ffffffff
[  610.829545] io 3c ffffffff
[  610.829573] io 40 ffffffff
[  610.829602] io 44 ffffffff
[  610.829628] io 48 ffffffff
[  610.829656] io 4c ffffffff
[  610.829686] io 50 ffffffff
[  610.829710] io 54 ffffffff
[  610.829733] io 58 ffffffff
[  610.829761] io 5c ffffffff
[  610.829790] io 60 ffffffff
[  610.829815] io 64 ffffffff
[  610.829838] io 68 ffffffff
[  610.829861] io 6c ffffffff
[  610.829884] io 70 ffffffff
[  610.829907] io 74 ffffffff
[  610.829929] io 78 ffffffff
```

```
[  610.829952] io 7c ffffffff
[  610.829975] io 80 40b00104
[  610.829998] io 84 ffffffff
[  610.830024] io 88 40000
[  610.830049] io 8c ffffffff
[  610.830078] io 90 4
[  610.830104] io 94 ffffffff
[  610.832165] irq_handler irq = 10 dev = 245 irq_status =
12345678
[  611.840333] receive a FACTORIAL interrupter!
[  611.840336] irq_handler irq = 10 dev = 245 irq_status = 1
[  612.864167] computing result 375f00
[  612.964059] receive a DMA read interrupter!
[  612.964063] irq_handler irq = 10 dev = 245 irq_status = 100
[  614.988143] receive a DMA read interrupter!
[  614.988146] irq_handler irq = 10 dev = 245 irq_status = 100
```