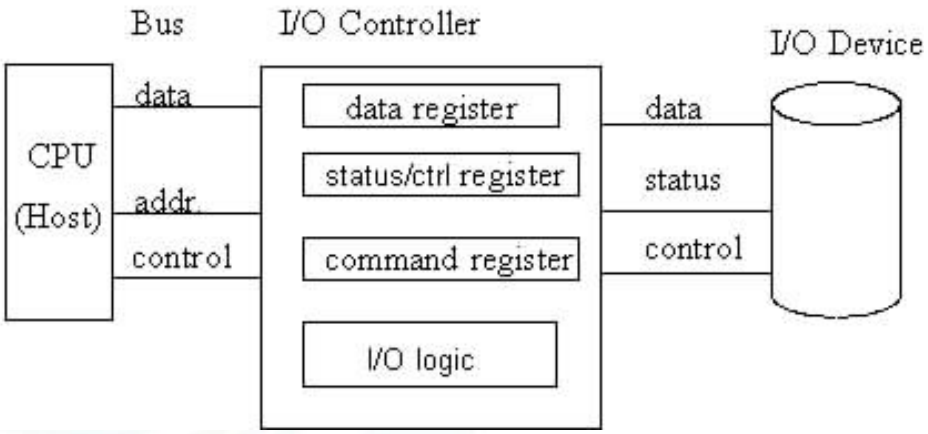
9.2 I/O空间的管理

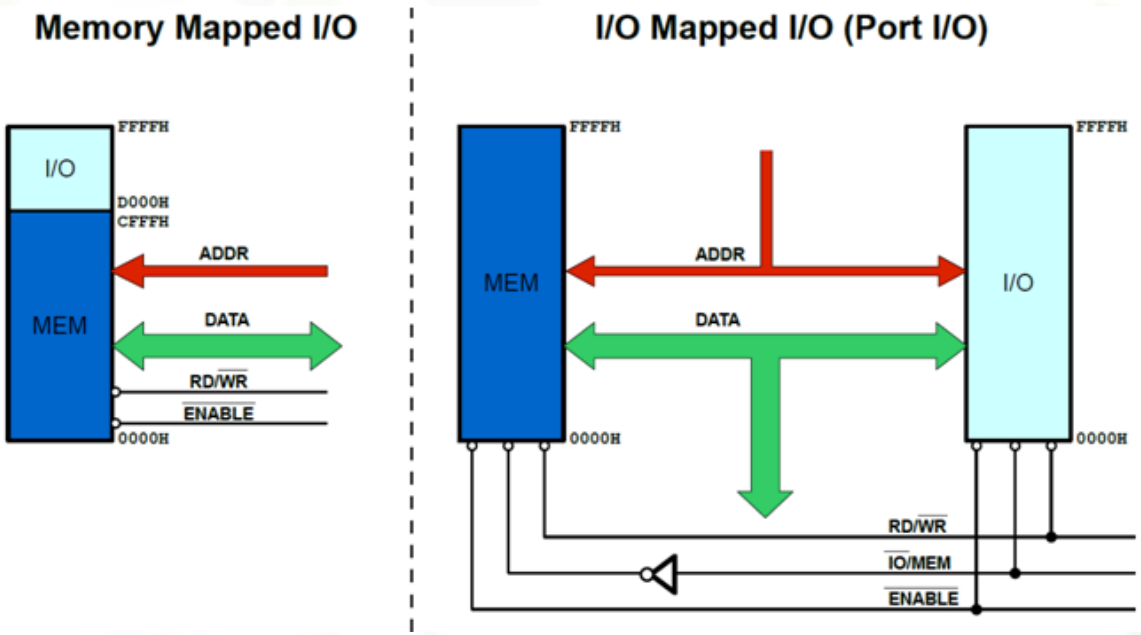
西安邮电大学

设备控制器



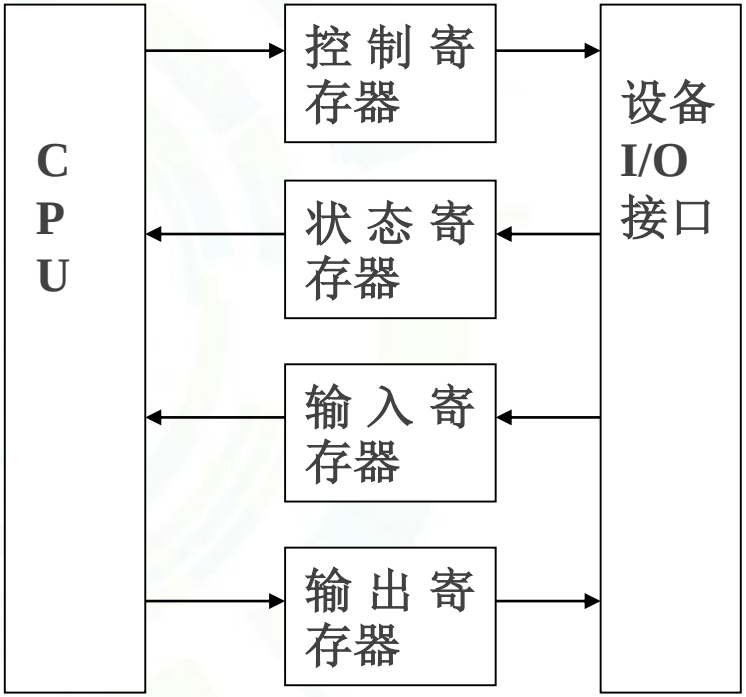
设备控制器是计算机中的一个实体，其主要职责是控制一个或多个I/O设备，以实现I/O设备和处理器之间的数据交换。它是CPU与I/O设备之间的接口，它通过控制总线接收从CPU发来的命令，并去控制I/O设备工作，控制器相当于CPU与外设打交道的助理，以使处理机从繁杂的设备控制事务中解脱出来。

I/O 空间的管理



设备通常会提供一组寄存器来控制设备、读写设备以及获取设备的状态。这些寄存器就是控制寄存器、数据寄存器和状态寄存器，它们位于控制器中。从编址方式来说，如果I/O空间与内存一起编址，对应的内存空间被称为I/O内存。如果I/O空间单独编址，就位于I/O空间，通常被称为I/O端口。

I/O 端口



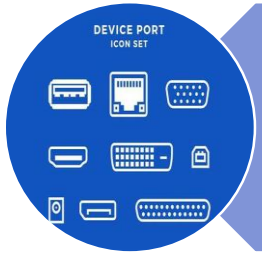
专用I/O端口

设备驱动程序要直接访问外设或其接口卡上的物理电路，通常以寄存器的形式出现访问；

外设寄存器也称为I/O端口，通常包括控制寄存器、状态寄存器和数据寄存器三类。

如何访问I/O 内存和I/O 端口

I/O内存



I/O端口

◆**访问“I/O 内存”方式：**

寄存器参与内存统一编址，访问寄存器，通过访问一般的内存指令来访问寄存器。

◆ **访问“I/O 端口”方式：**

将外设的寄存器看成一个独立的地址空间，对外设寄存器的读/写设置专用指令。

访问I/O内存资源

访问I/O内存

```
void *
ioremap(unsigned
long offset,
unsigned long size);
```

offset：I/O设备上的一块物理内存的起始地址；

size：要映射的空间的大小；

用于I/O指令的“地址空间”相对来说是很小的。自从PCI总线出现后，无论CPU的设计采用I/O 端口方式，还是I/O内存方式，都必须将外设卡上的存储器映射到内存空间，实际上是采用了虚拟内存的手段，这样的映射是通过ioremap（）来建立的，该函数如图所示：

其中参数的含义如图所示：

访问I/O端口空间

访问I/O端口

inb() outb()

inw() outw()

inl() outl()

inb()的原型为：

```
unsigned char
inb(unsigned
port);
```

在驱动程序请求了I/O端口空间中的端口资源后，它就可以通过CPU的IO指令来读写这些I/O端口。在读写I/O端口时要注意的一点就是，大多数平台都区分8位、16位和32位的端口。

port参数指定I/O端口空间中的端口地址。在大多数平台上(如x86)它都是unsigned short类型的，其它的一些平台上则是unsigned int类型的。显然，端口地址的类型是由I/O端口空间的大小来决定的。

查看你机子上的I/O端口

查看你机子上的I/O端口

cat /proc/ioports

你可以通过访问 /proc/ioports

来获取设备当前的 I/O 端口号

```
[clj@localhost ~]$ cat /proc/ioports
0000-0cf7 : PCI Bus 0000:00
  0000-001f : dma1
  0020-0021 : pic1
  0040-0043 : timer0
  0050-0053 : timer1
  0060-0060 : keyboard
  0064-0064 : keyboard
  0070-0071 : rtc0
  0080-008f : dma page reg
  00a0-00a1 : pic2
  00c0-00df : dma2
  00f0-00ff : fpu
  0170-0177 : 0000:00:01.1
    0170-0177 : ata_piix
  01f0-01f7 : 0000:00:01.1
```

I/O 资源管理

```
struct resource {
        resource_size_t start; //资源范围的开始
         resource_size_t_end; //资源范围的结束
         const char * name; //资源拥有者的名字
         unsigned long flags; //各种标志
         struct resource * parent, * sibling, * child;
      //指向资源树中父、兄以及孩子的指针
}
```

Linux将基于I/O端口和I/O内存的映射方式通称为“I/O区域”(I/O region)。

Linux 设计了一个通用的数据结构 resource 来描述各种 I/O 资源。该结构定义在：

include/linux/ioport.h

I/O 资源管理

```
struct resource ioport_resource = {
        .name = "PCI IO",
        .start = 0,
        .end = IO_SPACE_LIMIT,
        .flags = IORESOURCE_IO,
};
EXPORT_SYMBOL(ioport_resource);
```

```
struct resource iomem_resource = {
        .name = "PCI mem",
        .start = 0,
        .end = -1,
        .flags = IORESOURCE_MEM,
};
```

Linux下对I/O资源主要用结构体resource来管理，管理的方法就是用resource来描述使用的I/O资源的状态，并将这些resource用如下两个resource作为表头，一个是ioport_resource，一个是iomem_resource，这个链表按地址大小的顺序链接起来。

I/O 资源管理

request_resource()	• 把一个给定范围分配给一个 I/O 设备
alloate_resource()	• 在资源树中寻找一个给定大小和排列方式可用的范围
release_resource()	• 释放以前分配给 I/O 设备的给定范围

任何设备驱动程序都可以使用下面三个函数申请、分配和释放资源，传递给它们的参数为资源树的根节点和要插入的新资源数据结构的地址。

管理I/O端口资源

request_region()	• 请求在I/O端口空间中分配指定范围的I/O端口资源。
check_region()	• 检查I/O端口空间中的指定I/O端口资源是否已被占用。
release_region()	• 释放I/O端口空间中的指定I/O端口资源。

采用I/O端口的X86处理器为外设实现了一个单独的地址空间，也即“I/O空间”或称为“I/O端口空间”，其大小是64KB(0x0000-0xffff)。Linux在其所支持的所有平台上都实现了“I/O端口空间” 这一概念。

Linux是基于“I/O 区域”这一概念来实现对I/O端口资源的管理的。Linux在头文件include/linux/ioport.h中定义了三个对I/O端口空间进行操作的接口函数：

管理I/O 区域资源

__request_region()	• I/O 区域的分配
__release_region()	• I/O 区域的释放
__check_region()	• 检查指定的I/O 区域是否已被占用

Linux将基于I/O端口和基于I/O内存的资源统称为“I/O区域”。I/O 区域仍然是一种I/O资源，因此它仍然可以用resource结构类型来描述。Linux在头文件include/linux/ioport.h中定义了三个对I/O区域进行操作的接口函数：

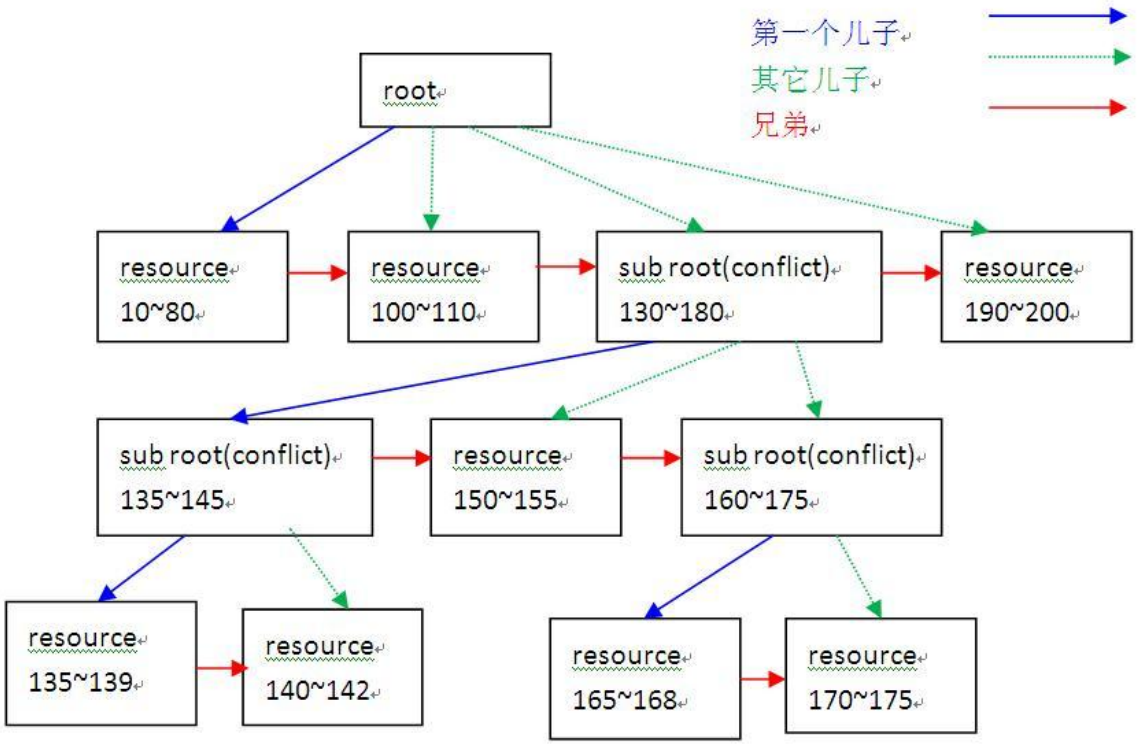
其中__request_region函数的主要功能为：查找resource链表中是否有与申请的I/O资源有冲突，如冲突则返回NULL，如不冲突则将新申请resource按resource地址从小到大的顺放插入至以ioport_resource或iomem_resource为表头（root）的单向指针链表中。

管理I/O内存资源

request_mem_region()	• 请求分配指定的I/O内存资源。
_check_mem_region()	• 检查指定的I/O内存资源是否已被占用。
release_mem_region()	• 释放指定的I/O内存资源。

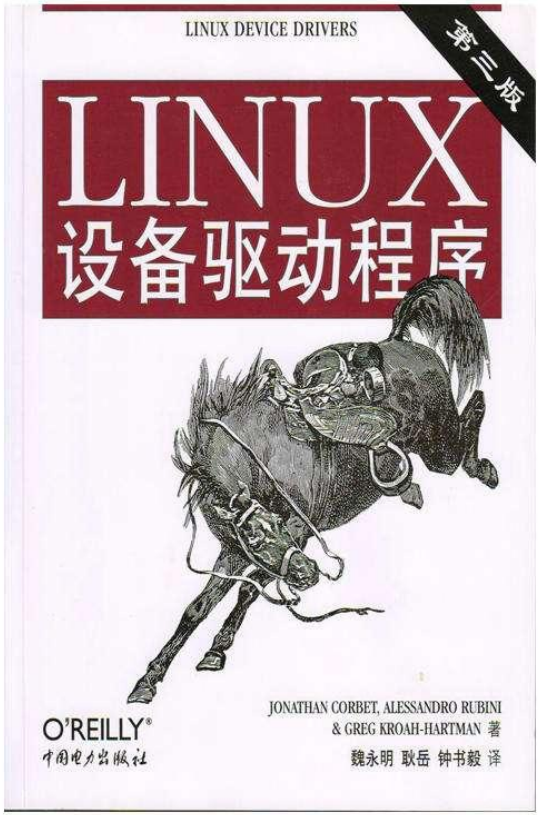
基于I/O区域的操作函数__xxx_region()，Linux在头文件include/linux/ioport.h中定义了三个对I/O内存资源进行操作的接口：

小结



Linux是以一种倒置的树形结构来管理每一类I/O资源（如：I/O端口、外设内存、DMA和IRQ）等。每一类I/O资源都对应有一颗倒置的资源树，树中的每一个节点都是一个resource结构，而树的根结点root则描述了该类资源的整个资源空间。

参考文献



1.《Linux 驱动开发》是最经典的参考书

2.网上有大量详尽的驱动开发资料，读者可自行查阅，推荐一篇基础篇：https://www.cnblogs.com/mrzhangxinjie/p/7170736.html

带着疑问上路

内核对I/O资源的管理为什么采用树结构？

谢谢大家！

THANK YOU