



操作系统 Operation System

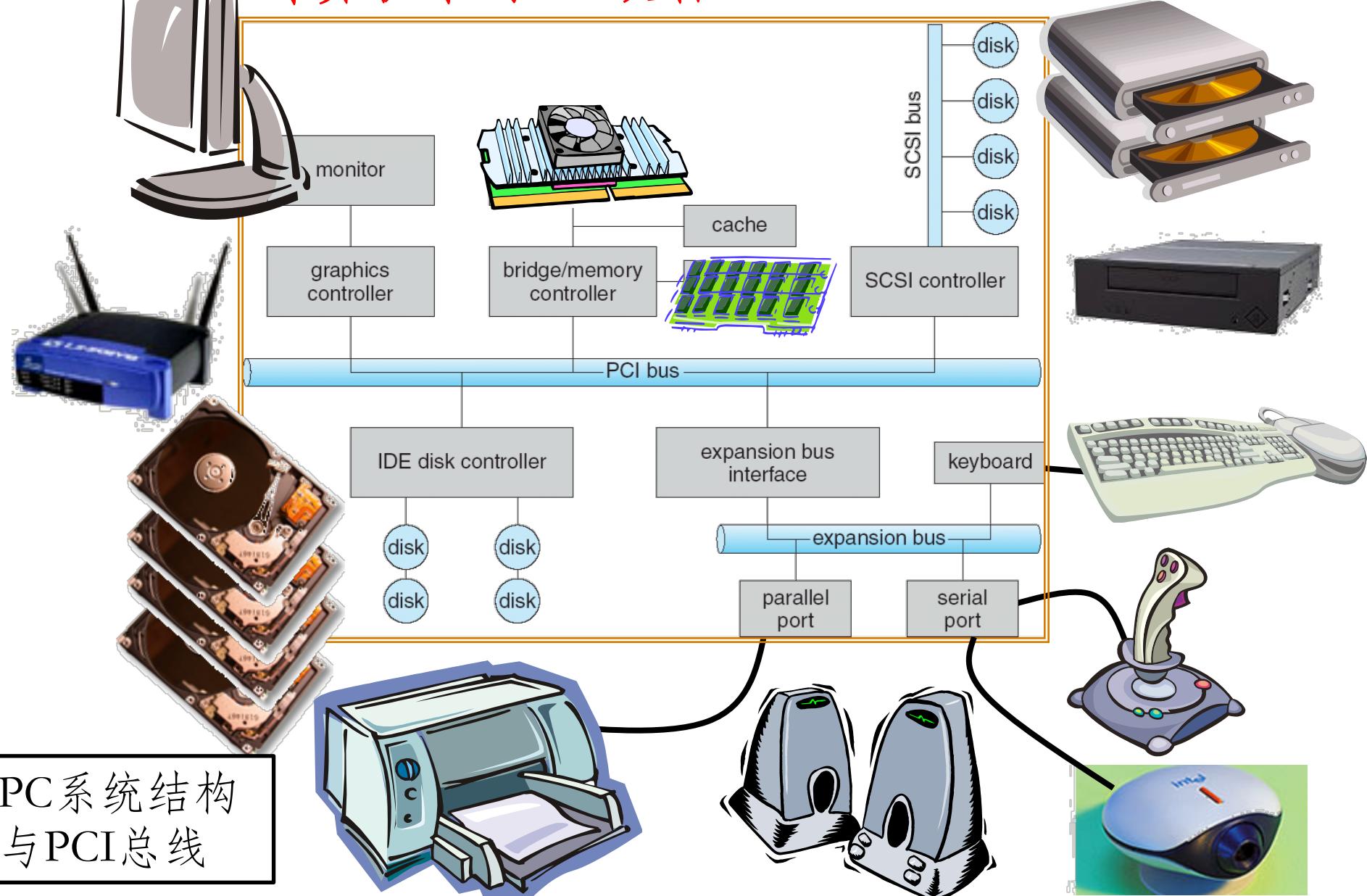
五、设备管理

2018年5月7日

内容提要

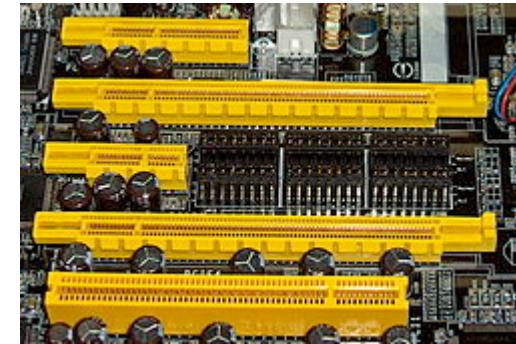
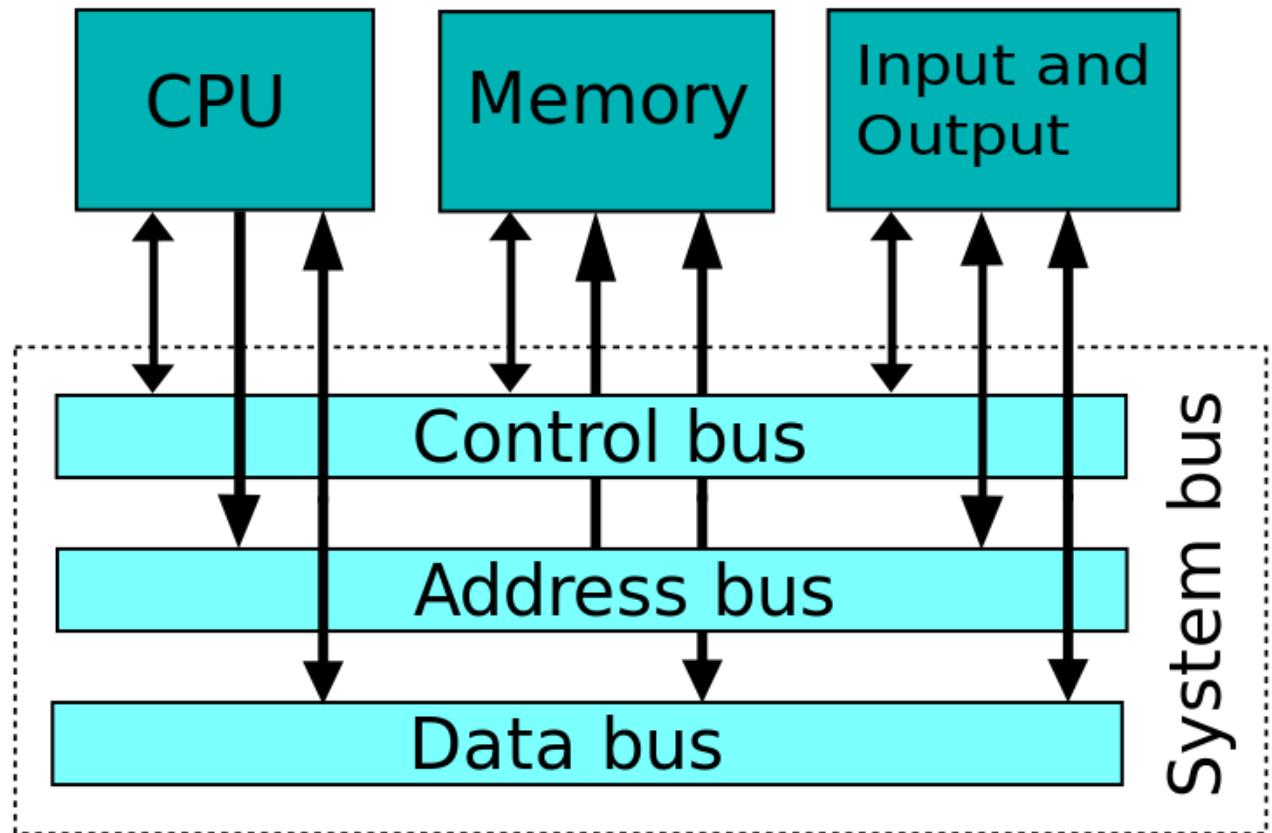
- 设备管理概述
- I/O硬件基础
- I/O软件原理
- I/O软件概述
- 设备管理实例

计算机中的I/O设备



PC系统结构
与PCI总线

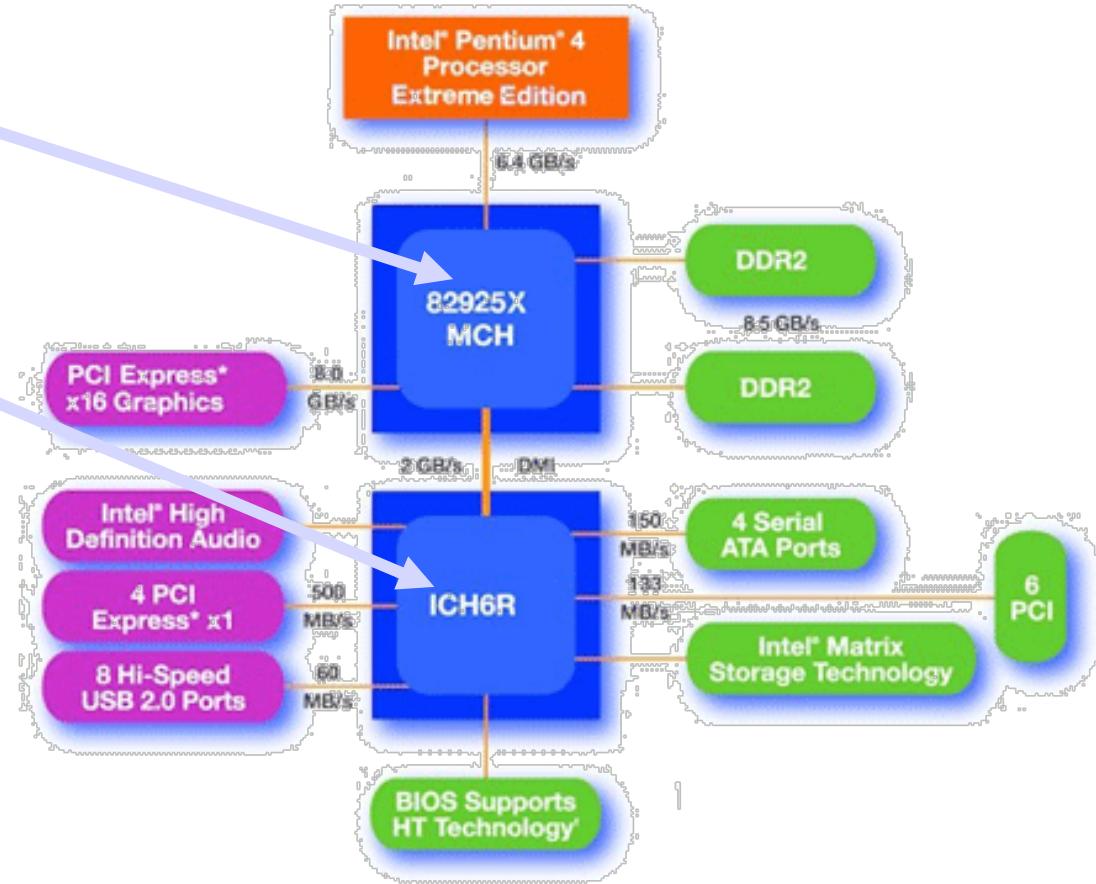
总线(Bus): 接入I/O设备的主要方式



CPU读入一个字，需要将地址放在地址总线上，然后在控制线上发一个读信号，最终数据总线会返回数据。

例如：Intel Pentium 4 芯片组的主要组件

- Northbridge:
 - Handles memory
 - Graphics
- Southbridge: I/O
 - PCI bus
 - Disk controllers
 - USB controllers
 - Audio
 - Serial I/O
 - Interrupt controller
 - Timers



I/O设备：种类多、差异大

- 传输速度：低速、中速、高速
- 信息交换单位：**块设备和字符设备**
 - 块设备：以块为传输单位的设备，每个块可以独立于其他块进行读写。例如：硬盘、CD-ROM、USB盘等。
 - 字符设备：以字符为单位发送或者接收字符流的设备，不可寻址。例如：打印机、网卡、鼠标等。
- 共享属性：独占设备、共享设备

例如：数据传输速率的差异可达若干数量级！

系统需要保障支持多个外设的情况下依然性能优良。

设备	数据率
键盘	10B/s
鼠标	100B/s
56K调制解调器	7KB/s
扫描仪	400 KB/s
数字便携式摄像机	3.5 MB/s
802.11g无线网络	6.75MB/s
52倍速CD-ROM	7.8MB/s
快速以太网	12.5 MB/s
袖珍闪存卡	40MB/s
火线 (IEEE 1394)	50MB/s
USB 2.0	60MS/s
SONET OC-12网络	78MB/s
SCSI Ultra 2磁盘	80MB/s
千兆以太网	125MB/s
SATA磁盘驱动器	300MB/s
Ultrium磁带	320MB/s
PCI总线	528MB/s

设备管理的目标和功能

■ 设备管理的目标

- 方便使用：方便用户使用，屏蔽外设的差异，对不同类型的设备统一使用方法，协调对设备的并发使用
- 提高效率：提高I/O访问效率，匹配CPU和多种不同处理速度的外设
- 方便控制：方便OS对设备的控制。例如：增加和删除设备，适应新的设备类型

设备管理的目标和功能

- 设备管理的功能
 - 提供使用设备的用户接口：命令接口和编程接口。
 - 设备分配和释放：使用设备前，需要分配设备和相应的通道、控制器。
 - 设备的访问和控制：包括并发访问和差错处理。
 - I/O缓冲和调度：目标是提高I/O访问效率。

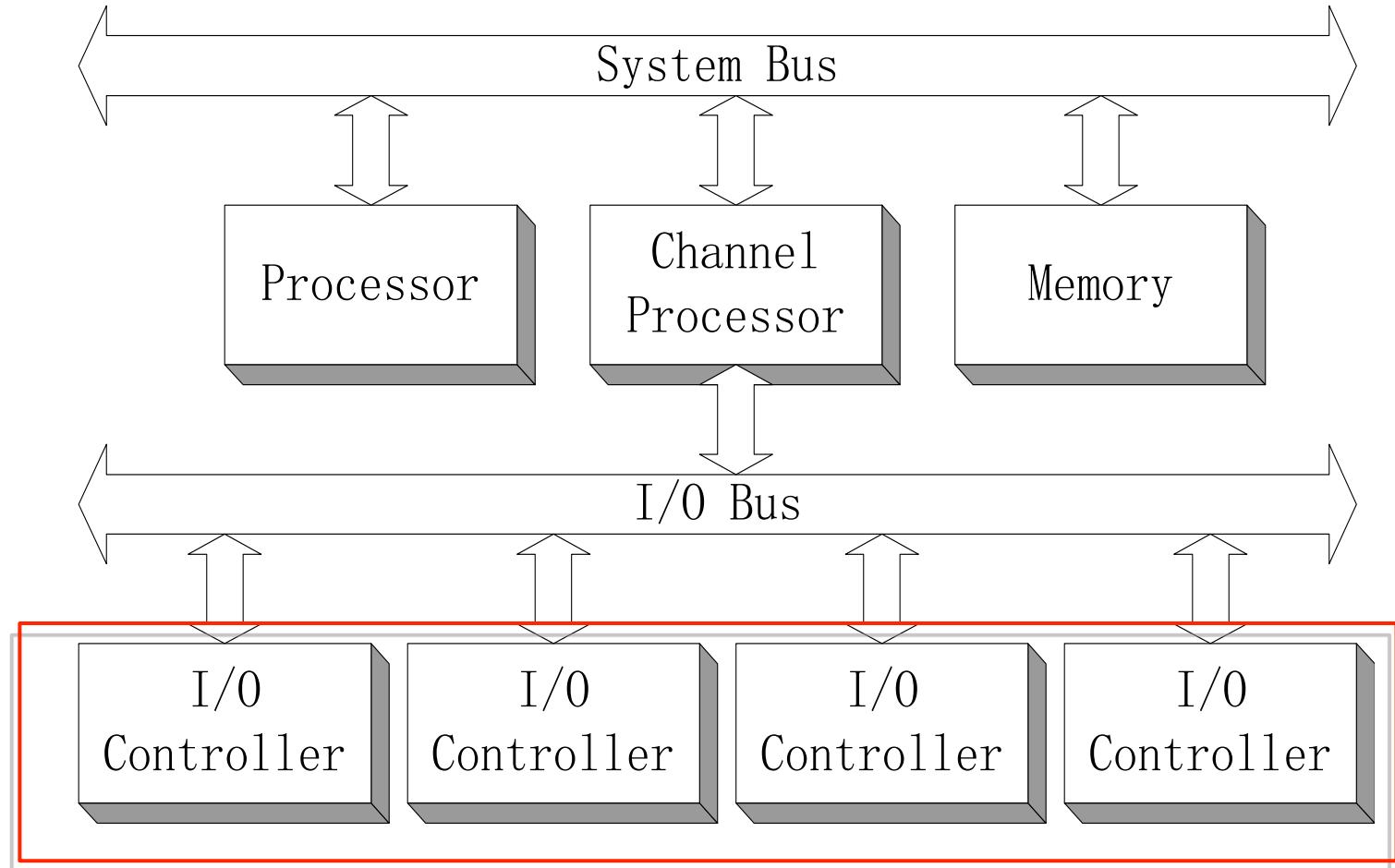
访问I/O设备的统一接口：举例

- ```
FILE fd = fopen("/dev/something", "rw");
for (int i = 0; i < 10; i++) {
 fprintf(fd, "Count %d\n", i);
}
close(fd);
```
- 实现向多种设备的输出操作
  - 设备驱动程序实现了标准的接口
- 常见的读写接口
  - 块设备：read(), write(), seek()
  - 字符设备：get(), put()
  - 网络设备（以太网、无线、蓝牙等）：独立于通信协议的socket接口

# 内容提要

- I/O管理概述
- I/O硬件基础
- I/O软件原理
- 设备管理实例

# I/O系统的抽象结构



设备控制器。典型的形式是  
插在PCIe插槽的一个板卡。

# 设备控制器

- I/O设备的电子组件
- 控制设备的功能
  - 一方面通过低层次的接口和硬件通讯
  - 一方面提供高层次的接口和CPU通讯
  - 通过一层抽象，屏蔽了底层硬件接口的复杂度
- 组成
  - 控制器与CPU接口：数据寄存器、控制寄存器、状态寄存器
    - 专门的I/O指令
    - 内存映射I/O
  - 控制器与设备接口：数据信号、控制信号、状态信号

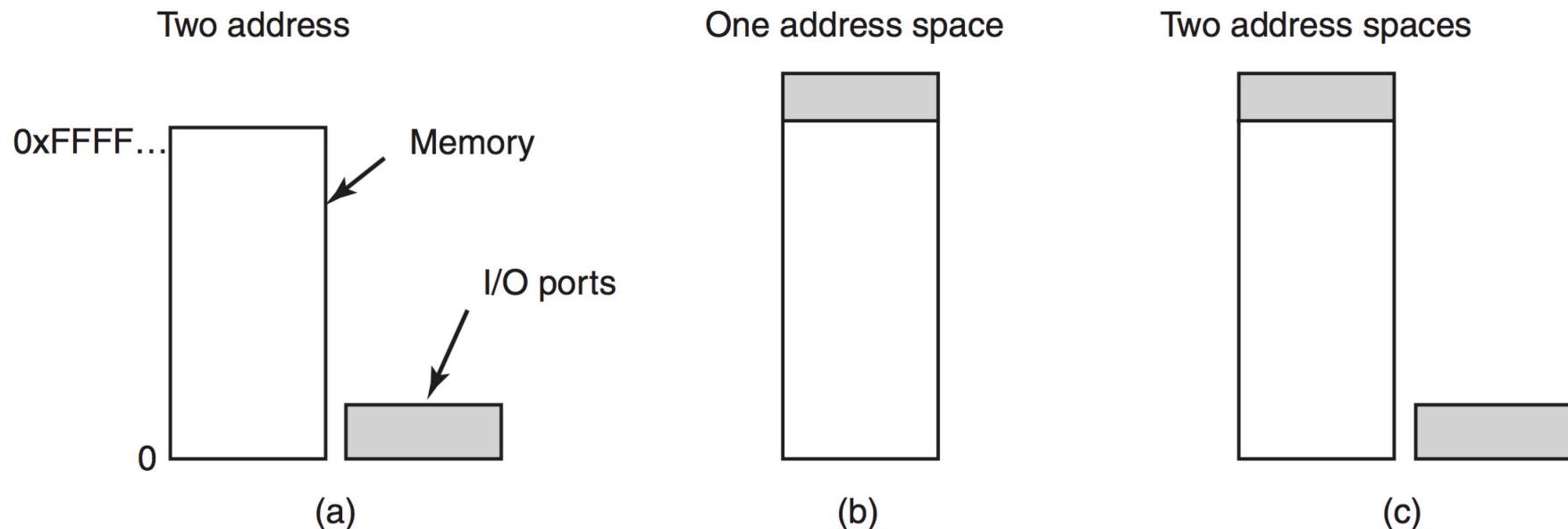
# CPU与I/O设备的通信过程

- CPU 主要和设备控制器交互
  - 控制器包括一组可以读写的寄存器
  - 也可能包括可以寻址的内存缓冲
- 每个寄存器被分配一个I/O端口号
  - 所有I/O端口构成了I/O地址空间
- CPU以访问控制器的方式：
  - I/O 指令: in/out 指令
    - 独立的I/O和内存空间
    - Intel 体系架构: out 0x21,AL
  - 内存映射 I/O: load/store指令 (MIPS)
    - 控制器的内存/寄存器作为物理内存空间的一部分，可以寻址访问
    - 对I/O的操作通过load和store指令来实现

# CPU与I/O设备的通信过程

- CPU访问控制器的方式：

- I/O 指令: in/out 指令
  - 独立的I/O和内存空间
  - Intel 体系架构: out 0x21,AL
- 内存映射 I/O: load/store指令 (MIPS)
  - 控制器的内存/寄存器作为物理内存空间的一部分，可以寻址访问
  - 对I/O的操作通过load和store指令来实现



**Figure 5-2.** (a) Separate I/O and memory space. (b) Memory-mapped I/O.  
(c) Hybrid.

# PC上I/O设备端口

| I/O address range (hexadecimal) | device                    |
|---------------------------------|---------------------------|
| 000-00F                         | DMA controller            |
| 020-021                         | interrupt controller      |
| 040-043                         | timer                     |
| 200-20F                         | game controller           |
| 2F8-2FF                         | serial port (secondary)   |
| 320-32F                         | hard-disk controller      |
| 378-37F                         | parallel port             |
| 3D0-3DF                         | graphics controller       |
| 3F0-3F7                         | diskette-drive controller |
| 3F8-3FF                         | serial port (primary)     |

# 内容提要

- I/O管理概述
- I/O硬件基础
- I/O软件原理
- I/O软件概述
- 设备管理实例

# I/O软件的目标

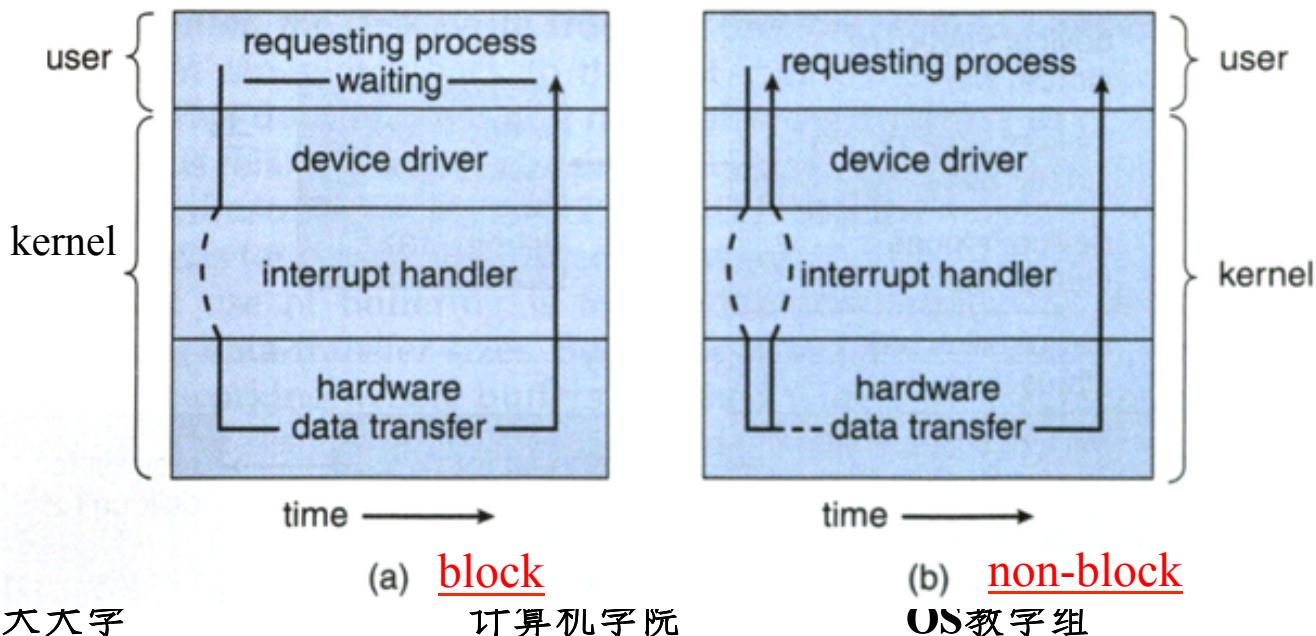
- 设备无关性
  - 应用程序对输入输出的访问和特定设备无关
  - 例如，读取文件的方式和底层具体的设备无关：  
硬盘、DVD、USB都可以读取文件
- 统一命名
  - 使用字符串或者整数统一命名设备
  - UNIX：设备集成到文件系统层次中，路径名统一寻址
- 错误处理
  - 尽可能靠近硬件的层次处理
  - 控制器→驱动→设备无关软件

# I/O软件的目标

- 同步和异步传输
  - 大多数物理I/O操作实际是异步的
  - 操作系统封装为用户实现了同步
  - 挂起进程，等待数据，唤醒进程
- 缓冲
  - 缓冲设备数据用以进行检查
  - 协调数据的生产和消费
  - 性能优化

# 用户访问I/O设备的方式

- 从I/O功能实现的角度：阻塞 v.s. 非阻塞
  - 阻塞：进程对设备发出某种操作（如读/写）后被阻塞
    - `read()`: 在数据准备好之前，进程进入睡眠状态
    - `write()`: 设备准备好之前，进程进入睡眠状态
  - 非阻塞：进程对设备发出某种操作后立即返回
    - 当数据或者设备准备好之后，需要一定的机制来通知进程。
    - 例如：`select()`



# 用户访问I/O设备的方式

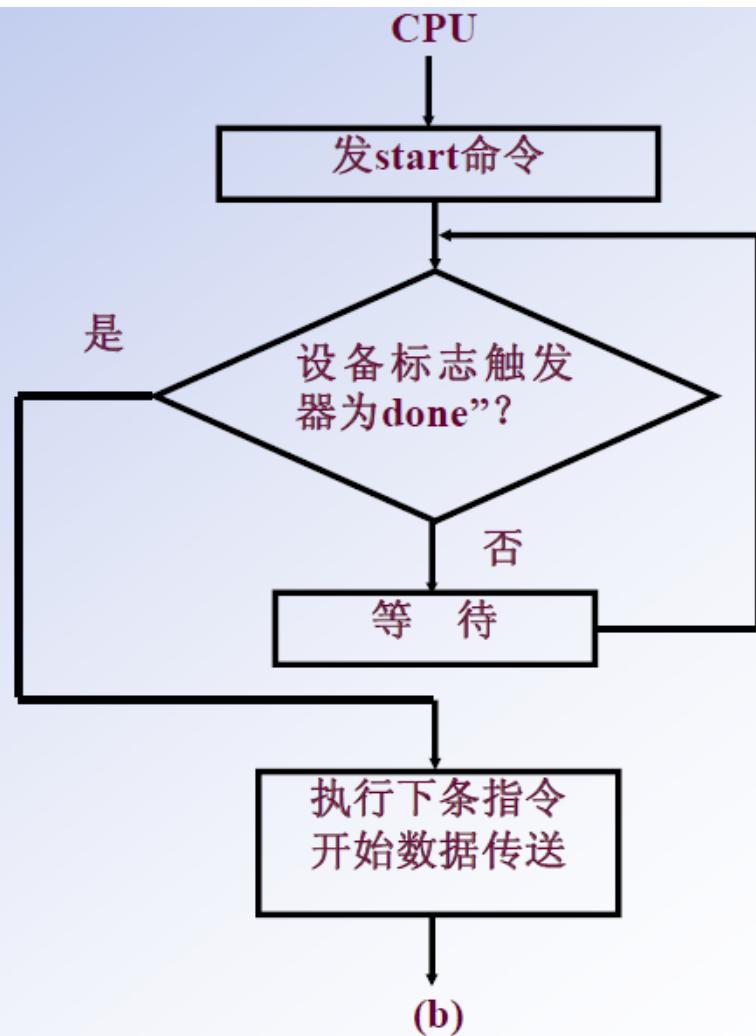
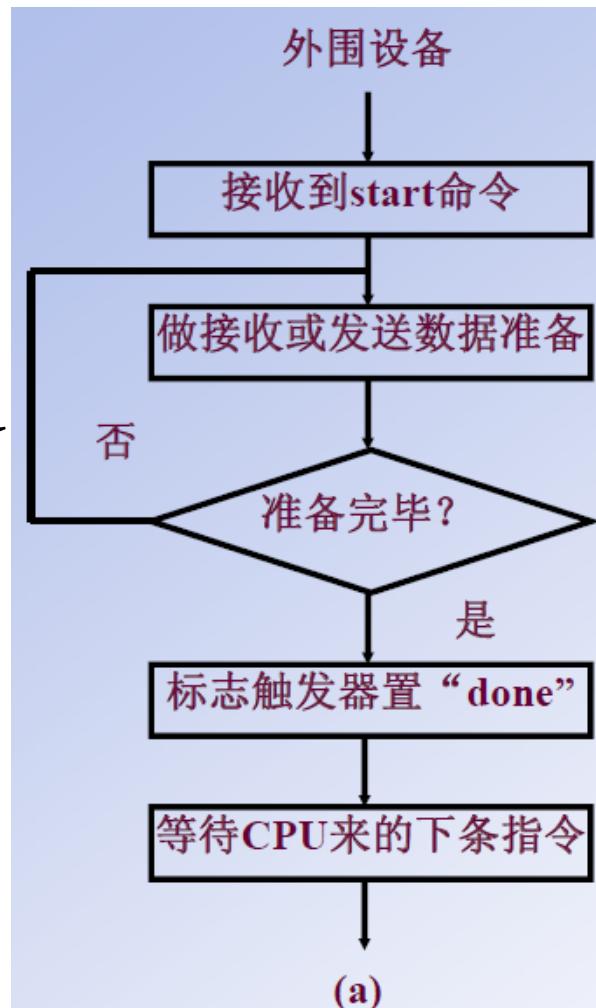
- 从用户进程的角度：同步 v.s. 异步
  - 异步
    - 用户进程发出读数据请求后，立即返回；内核将数据填充到用户的缓冲区（或从缓冲区取走数据），然后通知用户进程。
    - 窗口事件的响应大都是异步的回调函数
    - 大量网络数据的发送常常异步
  - 同步：用户进程发出请求之后，一直等待返回结果。
    - 快速的控制命令发送

# I/O控制技术

- 程序控制I/O(PIO,Programmed I/O)
- 中断驱动方式(Interrupt-driven I/O)
- 直接存储访问方式(DMA, Direct Memory Access)
- 通道技术 (Channel)

# 程序控制I/O(Programmed I/O)

I/O操作由程序发起，并等待操作完成。数据的每次读写通过CPU。



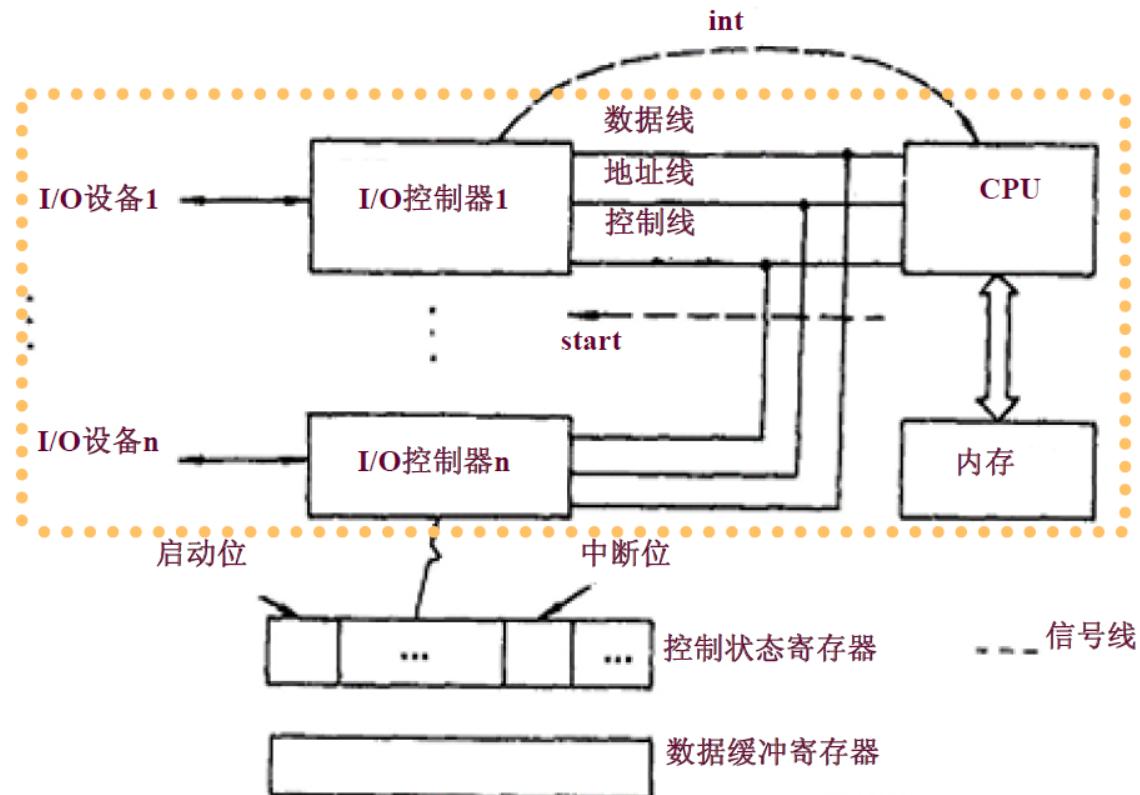
轮询或者忙等

优点：实现简单

缺点：CPU利用率低，在外设进行数据处理时，CPU只能等待，导致CPU资源浪费。

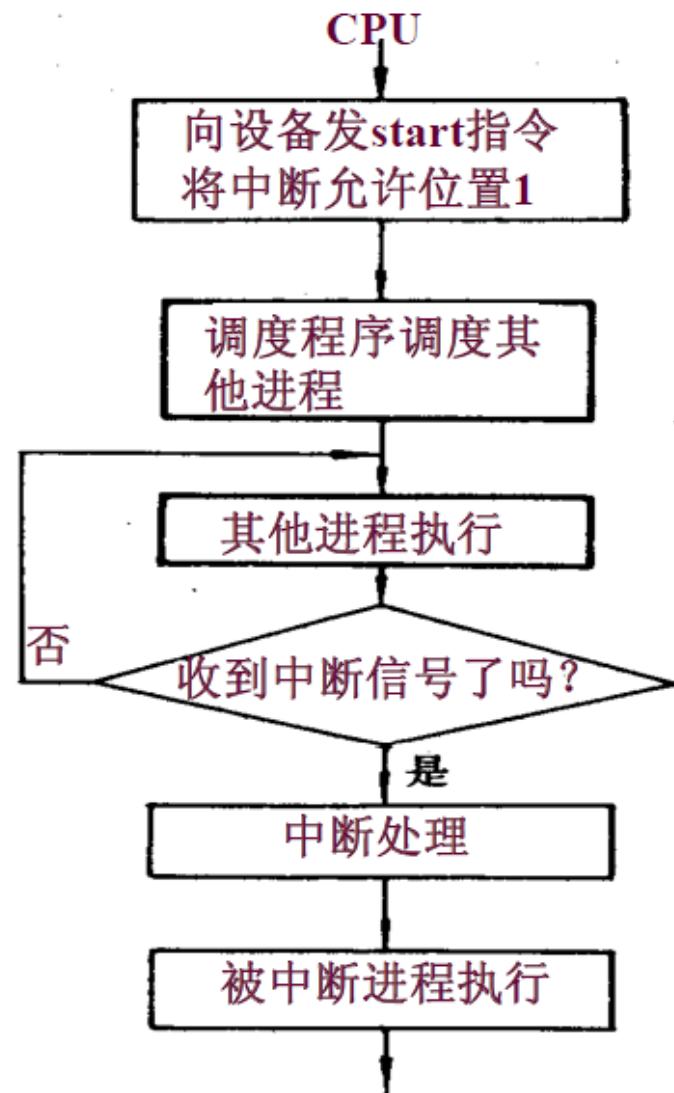
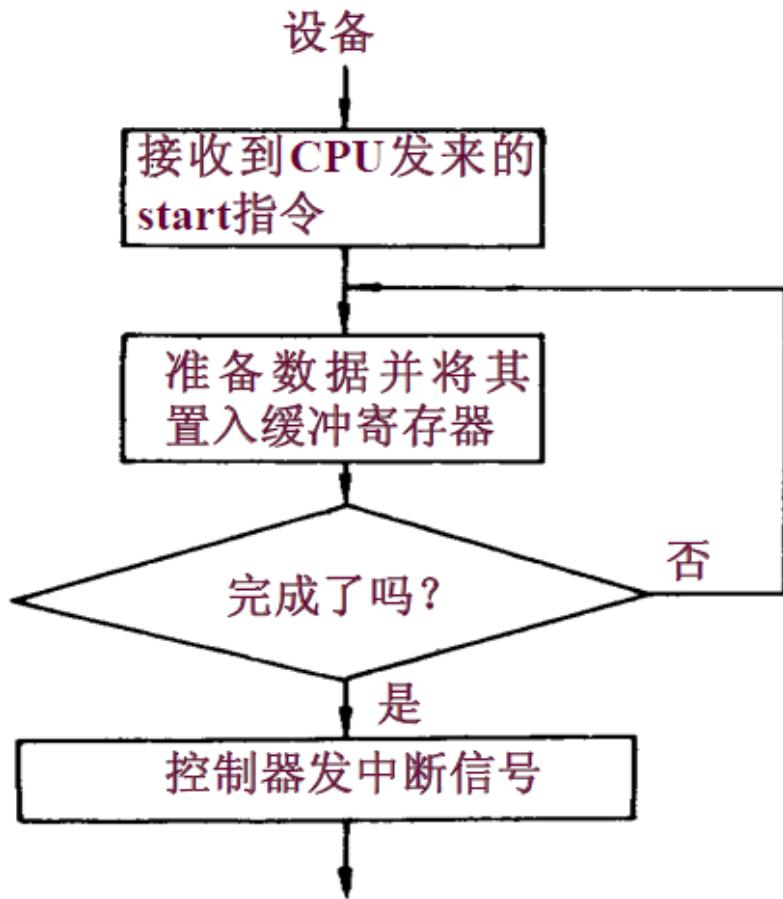
# 中断驱动方式

I/O操作由程序发起，在操作完成时（如数据可读或已经写入），由外设向CPU发出中断，通知该程序。数据的每次读写通过CPU。



- 优点：在外设进行数据处理时，CPU不必等待，可以继续执行该程序或其他程序，提高了CPU利用率；可以处理不确定事件。
- 缺点：每次输入/输出数据都要中断CPU，多次中断浪费CPU时间，只适于数据传输率较低的设备。

# 中断方式的处理过程



# 直接存储访问方式 (DMA, Direct Memory Access)

1. 由程序设置DMA控制器中的若干寄存器值（如内存始址，传送字节数），然后发起I/O操作；
2. DMA控制器完成内存与外设的成批数据交换；
3. 在操作完成时由DMA控制器向CPU发出中断。

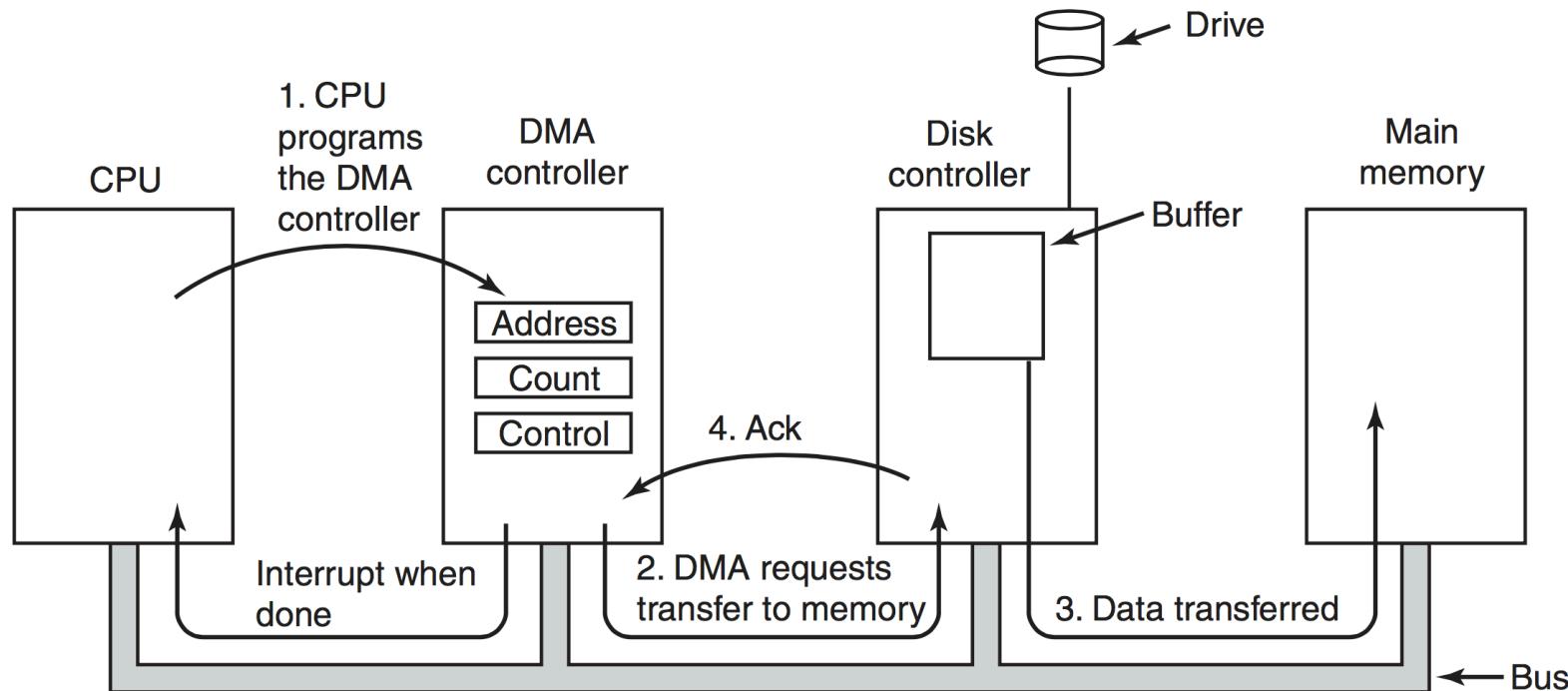
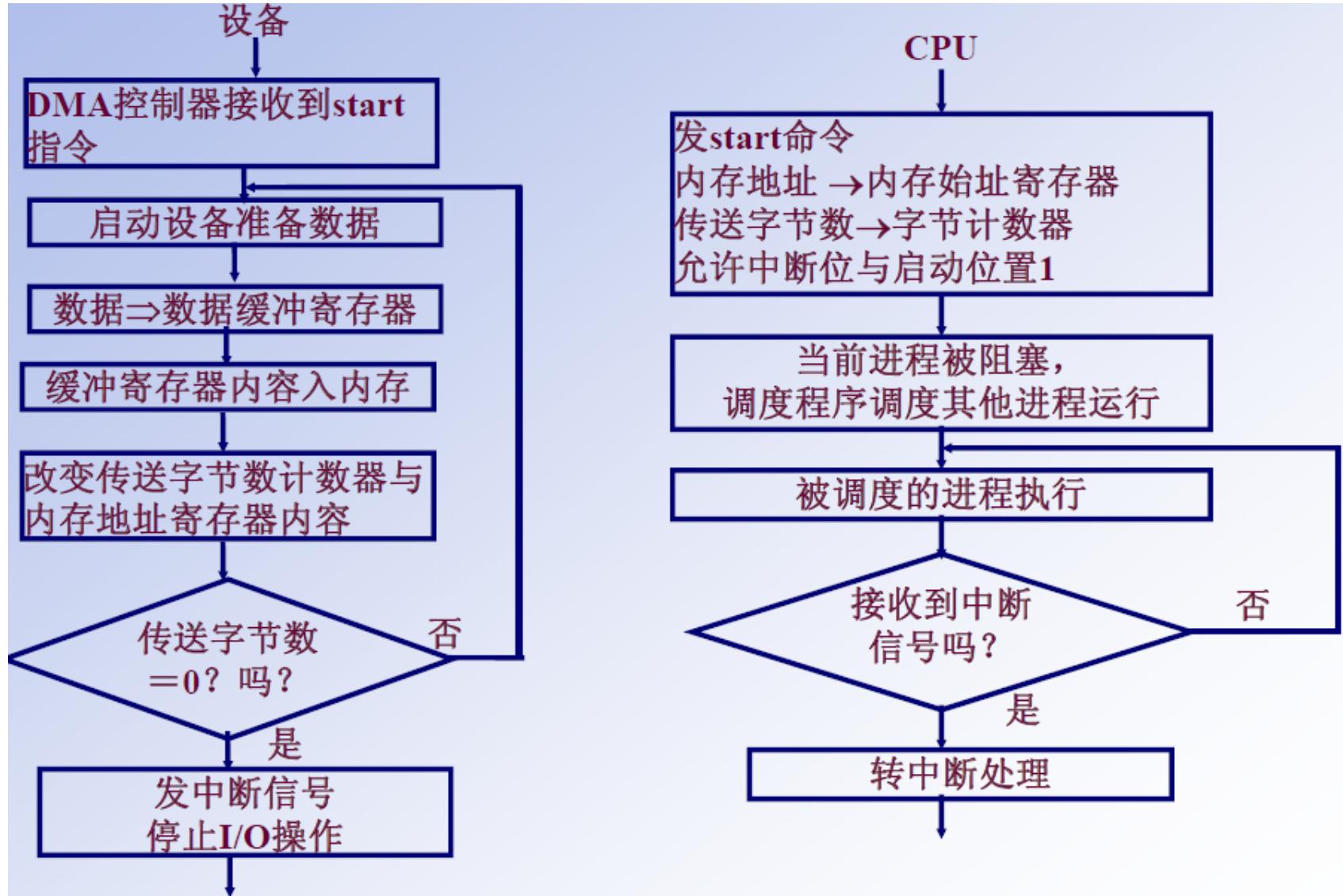


Figure 5-4. Operation of a DMA transfer.

# 直接存储访问方式 (DMA, Direct Memory Access)

- 优点：CPU只需干预I/O操作的开始和结束，而后续成批的数据读写则无需CPU控制，适于高速设备。
- 缺点：
  - 数据传送的方向、存放数据的内存地址及传送数据的长度等都由CPU控制，占用了CPU时间。
  - 每个设备占用一个DMA控制器，当设备增加时，需要增加新的DMA控制器。

# DMA处理基本流程



# 复制字符串-程序控制IO

```
copy_from_user(buffer, p, count); /* p is the kernel buffer */
for (i = 0; i < count; i++) { /* loop on every character */
 while (*printer_status_reg != READY) ; /* loop until ready */
 printer_data_register = p[i]; / output one character */
}
return_to_user();
```

**Figure 5-8.** Writing a string to the printer using programmed I/O.

对于每个字符的打印  
CPU需要不断查询打印机确认它可以接受字符

# 复制字符串-中断

```
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();
```

(a)

```
if (count == 0) {
 unblock_user();
} else {
 *printer_data_register = p[i];
 count = count - 1;
 i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

(b)

**Figure 5-9.** Writing a string to the printer using interrupt-driven I/O. (a) Code executed at the time the print system call is made. (b) Interrupt service procedure for the printer.

中断处理程序多次响应中断，每次一个字符

# 复制字符串-DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();
```

(a)

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

(b)

**Figure 5-10.** Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt-service procedure.

CPU设置DMA后调度其他程序，  
DMA打印所有字符后一次中断通知

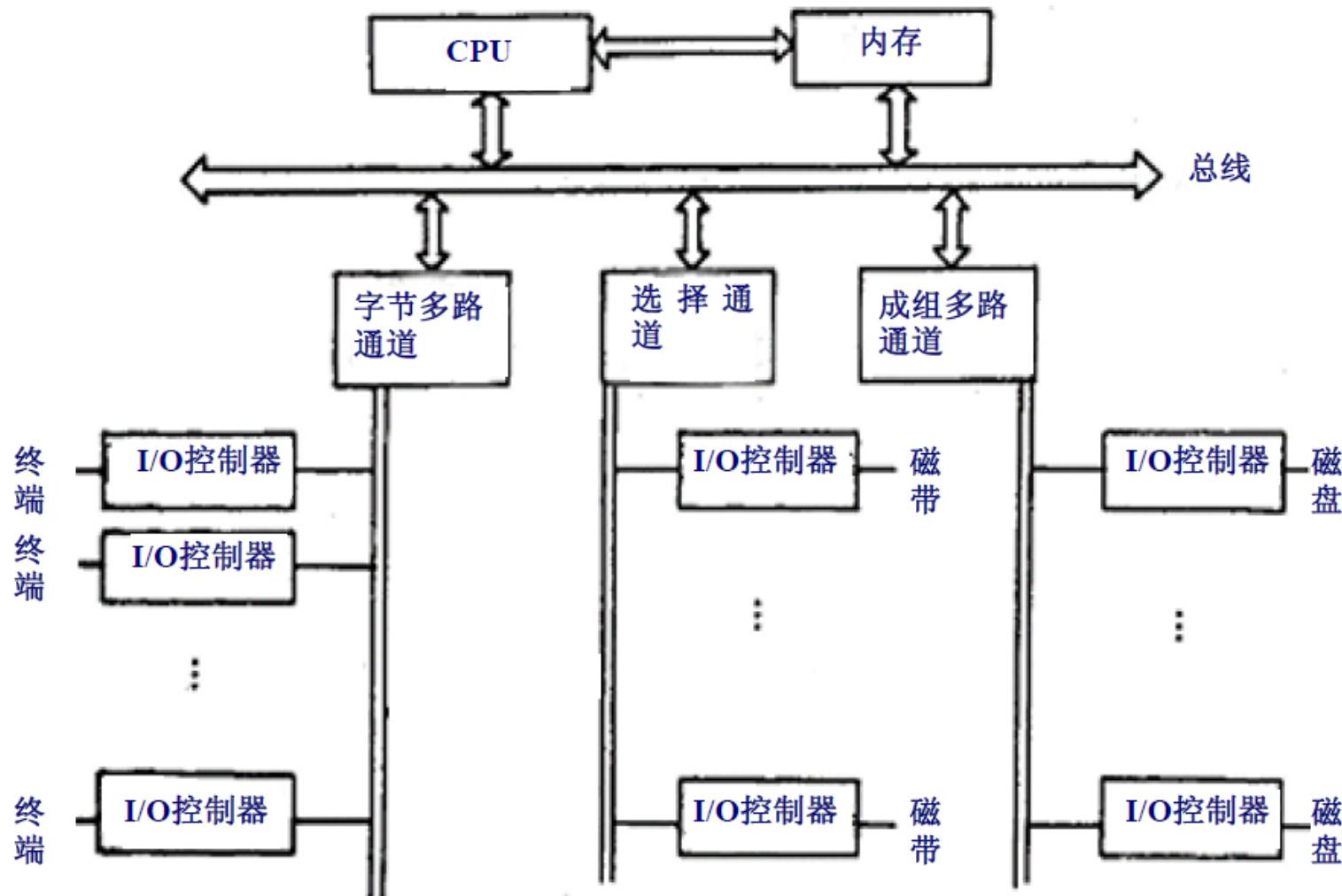
# DMA与中断方式的区别

- 中断控制方式在每个数据传送完成后中断CPU；DMA控制方式是在要求传送的一批数据完成之后中断CPU。
- 中断控制方式的数据传送是在中断处理时由CPU控制完成的，由于是程序切换，需要保护和恢复现场。
- DMA方式下是由DMA控制器控制完成的，在传输过程中不需要CPU干预，DMA控制器直接在主存和I/O设备之间传送数据，只有开始和结束才需要CPU干预。

# I/O通道控制方式(channel control)

- 基本思想：进一步减少CPU的干预。
- I/O通道是专门负责输入输出的处理器，独立于CPU，有自己的指令体系。可执行由通道指令组成的通道程序，因此可以进行较为复杂的I/O控制。通道程序通常由操作系统所构造，放在内存里。
- 优点：执行一个通道程序可以完成几组I/O操作，与DMA相比，减少了CPU干预。
- 缺点：费用较高。

# I/O通道分类



字节多路通道、数组选择通道、数组多路通道 教材P214

# 通道种类

- 字节多路通道
  - 以字节为单位分时交叉工作：当为一台设备传送一个字节后，立即转去为另一台设备传送一个字节；
  - 适用于连接打印机、终端等低速或中速的I/O设备。
- 数组选择通道
  - 以“组方式”工作，每次传送一批数据，传送速率很高，但在一段时间只能为一台设备服务。每当一个I/O请求处理完之后，就选择另一台设备并为其服务；
  - 适用于连接磁盘、磁带等高速设备。
- 数组多路通道
  - 综合了字节多路通道分时工作和选择通道传输速率高的特点；
  - 其实质是：对通道程序采用多道程序设计技术，使得与通道连接的设备可以并行工作。

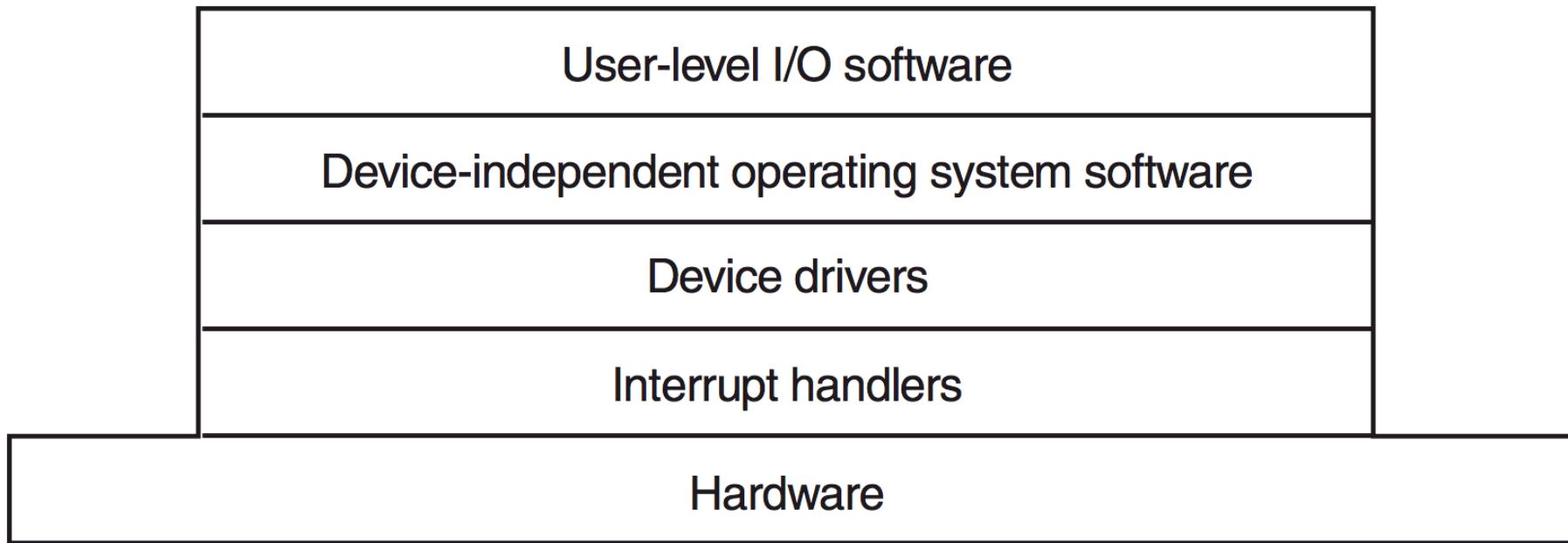
# I/O通道与DMA的区别

- DMA方式下，数据的传送方向、存放数据的内存起始地址和数据块长度都由CPU控制；而通道是一个特殊的处理器，有自己的指令和程序，通过执行通道程序实现对数据传输的控制，所以通道具有更强的独立处理I/O的功能。
- DMA控制器通常只能控制一台或者少数几台同类设备；而一个通道可同时控制多种设备。

# 内容提要

- I/O管理概述
- I/O硬件基础
- I/O软件原理
- I/O软件概述
- 设备管理实例

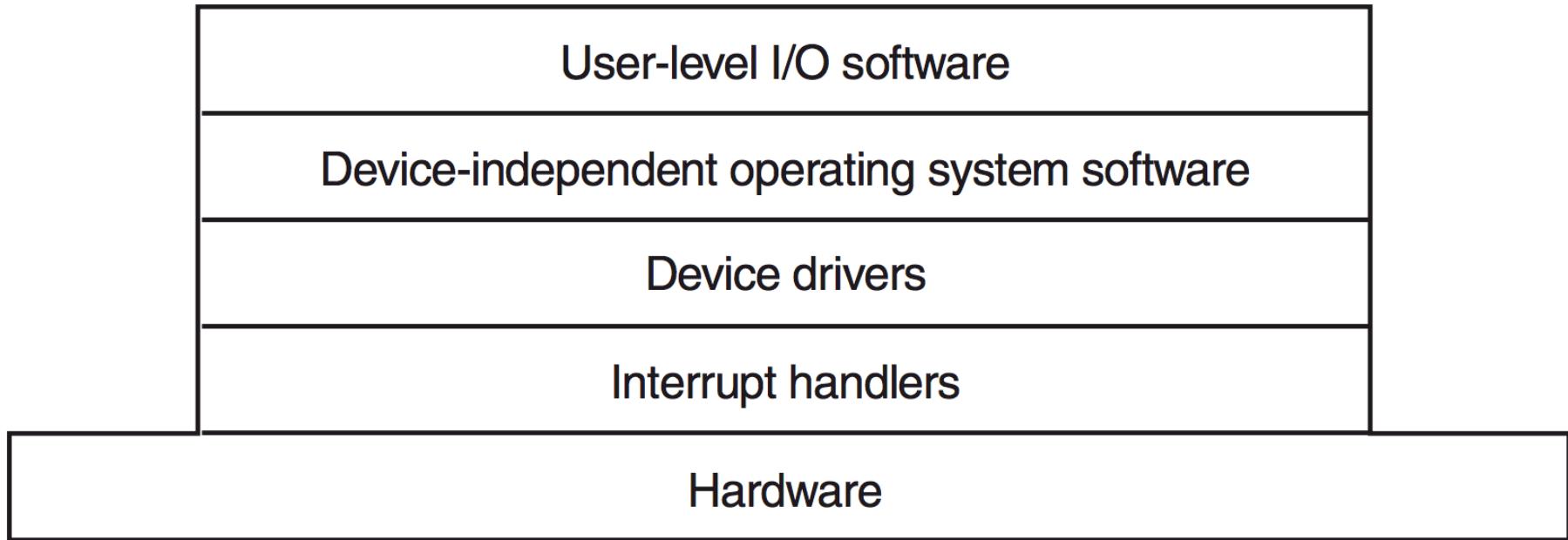
# I/O软件的层次结构



**Figure 5-11.** Layers of the I/O software system.

通过分层架构，每层完成明确的功能，实现约定的接口

# 中断处理程序-屏蔽中断处理的复杂性



**Figure 5-11.** Layers of the I/O software system.

中断的复杂性需要隐藏在系统底部，  
设备驱动程序启动I/O后，阻塞自己直到中断完成

# 中断处理过程

- 关中断
- 保存现场（各种寄存器和PSW）
- 为中断处理程序设置上下文（TLB，MMU，页表）和堆栈
- 运行设备中断处理程序，进行中断处理
- 恢复被中断进程的现场
- 开中断
- 设置MMU，PSW以执行下一个进程

# 中断处理过程

- 关中断
- 保存现场（各种寄存器和PSW）
- 为中断处理程序设置上下文（TLB，MMU，页表）和堆栈
- 运行设备中断处理程序，进行中断处理
- 恢复被中断进程的现场
- 开中断
- 设置MMU，PSW以执行下一个进程

中断处理需要花费相当多的CPU周期

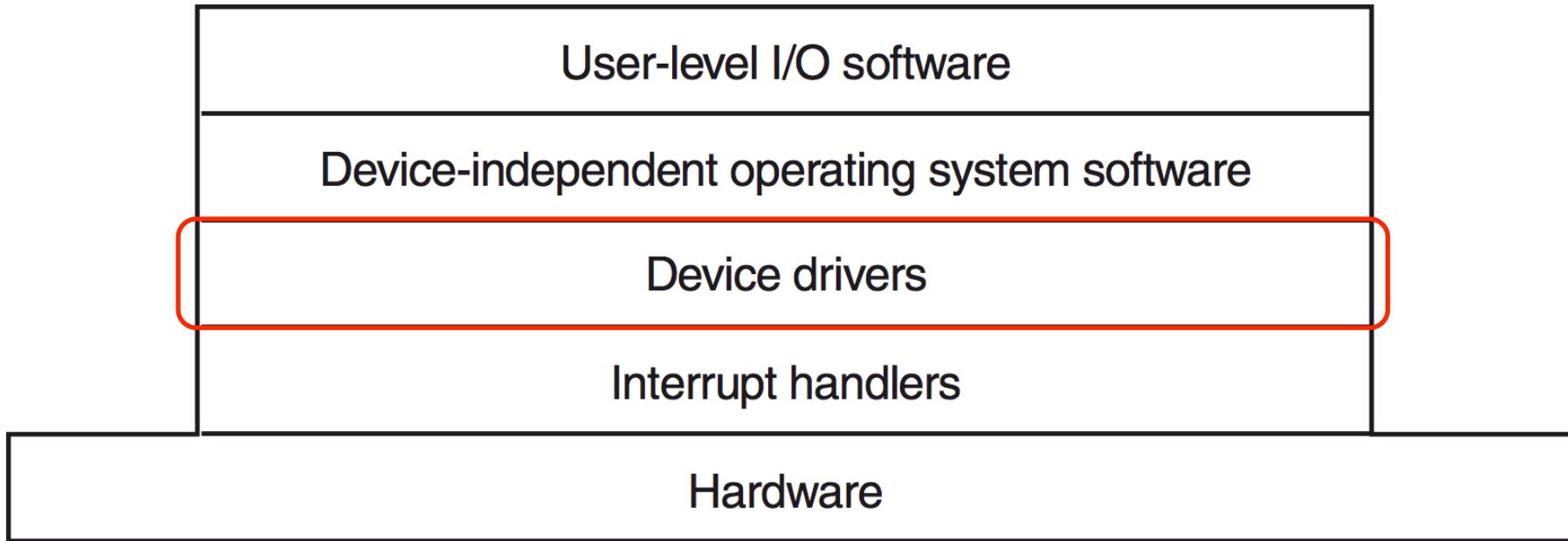
# 中断相关的几个术语

- 中断服务程序 (ISR: Interrupt Service Routine)
  - 也称为Interrupt Handler
  - 发生中断时所应执行的处理代码。
- 中断向量 (IV: Interrupt Vector)
  - 一个中断服务程序的入口地址
- 中断向量表 (IVT: Interrupt Vector Table)
  - 关联中断请求（索引）和中断服务程序的数据结构
- 中断描述符表 (IDT: Interrupt Descriptor Table)
  - x86架构下的中断向量表实现

# Intel Pentium 处理器的中断描述符表

| INT_NUM | Short Description PM                                       |
|---------|------------------------------------------------------------|
| 0x00    | <a href="#">Division by zero</a>                           |
| 0x01    | Debugger                                                   |
| 0x02    | <a href="#">NMI</a>                                        |
| 0x03    | Breakpoint                                                 |
| 0x04    | Overflow                                                   |
| 0x05    | Bounds                                                     |
| 0x06    | Invalid Opcode                                             |
| 0x07    | Coprocessor not available                                  |
| 0x08    | <a href="#">Double fault</a>                               |
| 0x09    | Coprocessor Segment Overrun ( <i>386 or earlier only</i> ) |
| 0x0A    | Invalid Task State Segment                                 |
| 0x0B    | Segment not present                                        |
| 0x0C    | Stack Fault                                                |
| 0x0D    | <a href="#">General protection fault</a>                   |
| 0x0E    | <a href="#">Page fault</a>                                 |
| 0x0F    | <i>reserved</i>                                            |
| 0x10    | Math Fault                                                 |
| 0x11    | Alignment Check                                            |
| 0x12    | Machine Check                                              |
| 0x13    | <a href="#">SIMD Floating-Point Exception</a>              |
| 0x14    | Virtualization Exception                                   |
| 0x15    | Control Protection Exception                               |

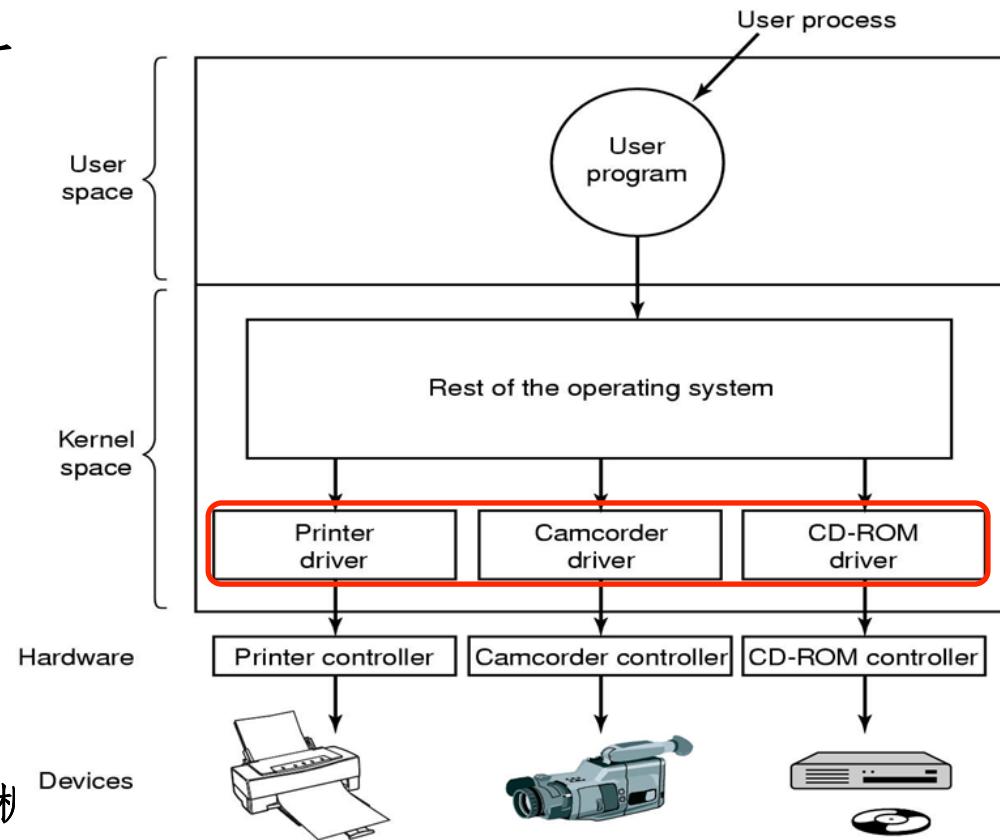
# 设备驱动程序- OS和设备能够独立开发，集成，动态加载



**Figure 5-11.** Layers of the I/O software system.

# 设备驱动程序

- 设备驱动程序：针对特定设备控制相关的代码
  - 不同设备控制器的数量和类型都很不相同
  - 一个驱动程序通常只能针对一类相似设备
- 由设备生产商开发和交付
- 通常位于OS的底部
- OS定义驱动的接口规范



# 设备驱动程序的功能

- 将抽象I/O请求转换为对物理设备的请求
- 检查I/O请求的合法性，否则报错
- 初始化设备
- 查询设备状态，对请求排队
- 启动设备，发出I/O命令
- 同步操作/异步操作
- 将数据发送给上层设备无关软件，或错误报告。

# 特点

- I/O进程与设备控制器之间的通信程序
- 驱动程序与I/O设备的特性紧密相关
- 通常是可重入的设计（例如，网络驱动）
- 处理设备的异常增加和删除

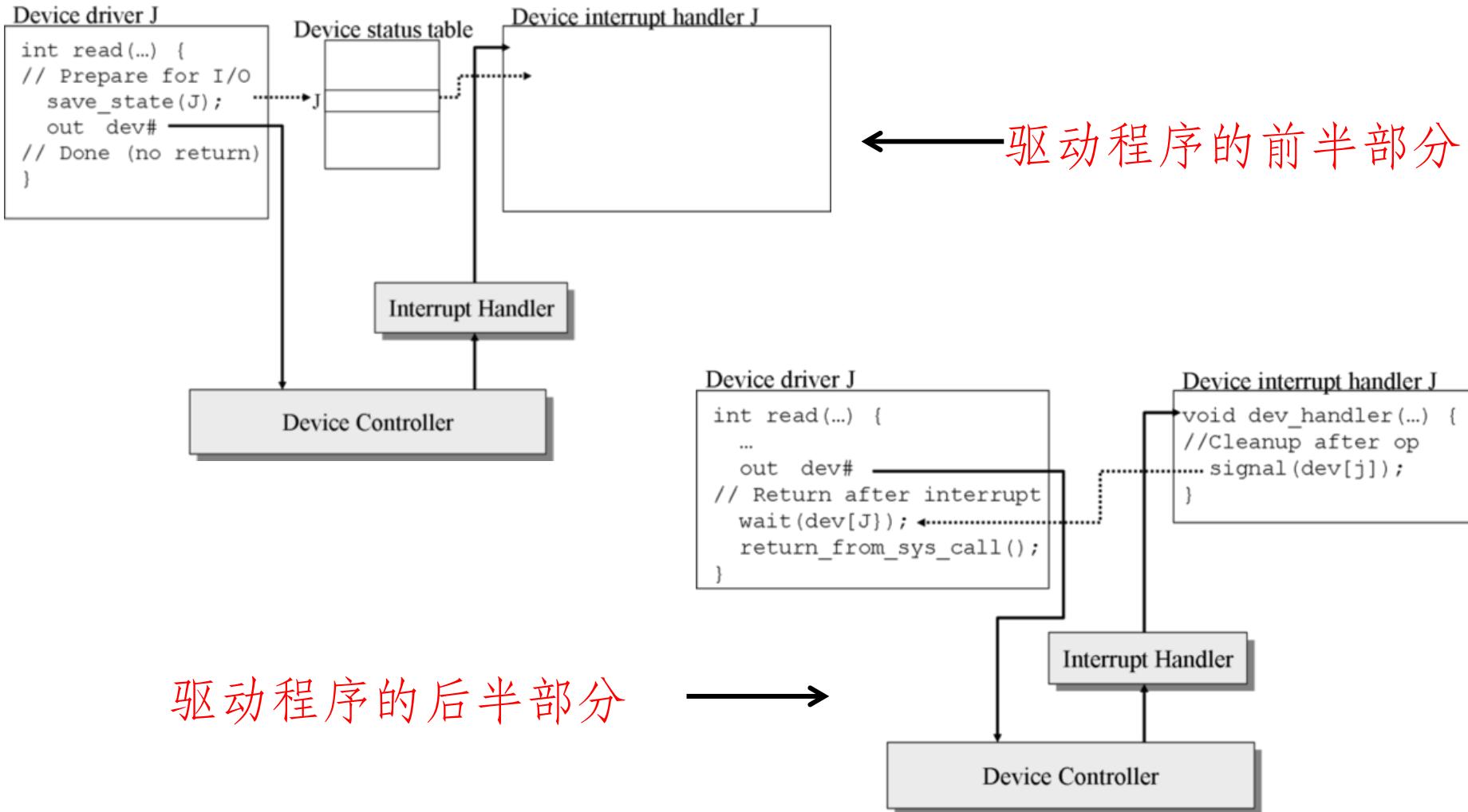
# 设备驱动程序的组成

- **自动配置和初始化子程序**: 检测所要驱动的硬件设备是否存在、是否正常。如果该设备正常，则对该设备及其相关的设备驱动程序需要的软件状态进行初始化。在初始化的时候被调用一次。
- **服务于I/O请求的子程序**: 调用该子程序是系统调用的结果。执行该部分程序时，系统仍认为是和调用进程属同一个进程，只是由用户态变成核心态，具有进行此系统调用的用户程序的运行环境，

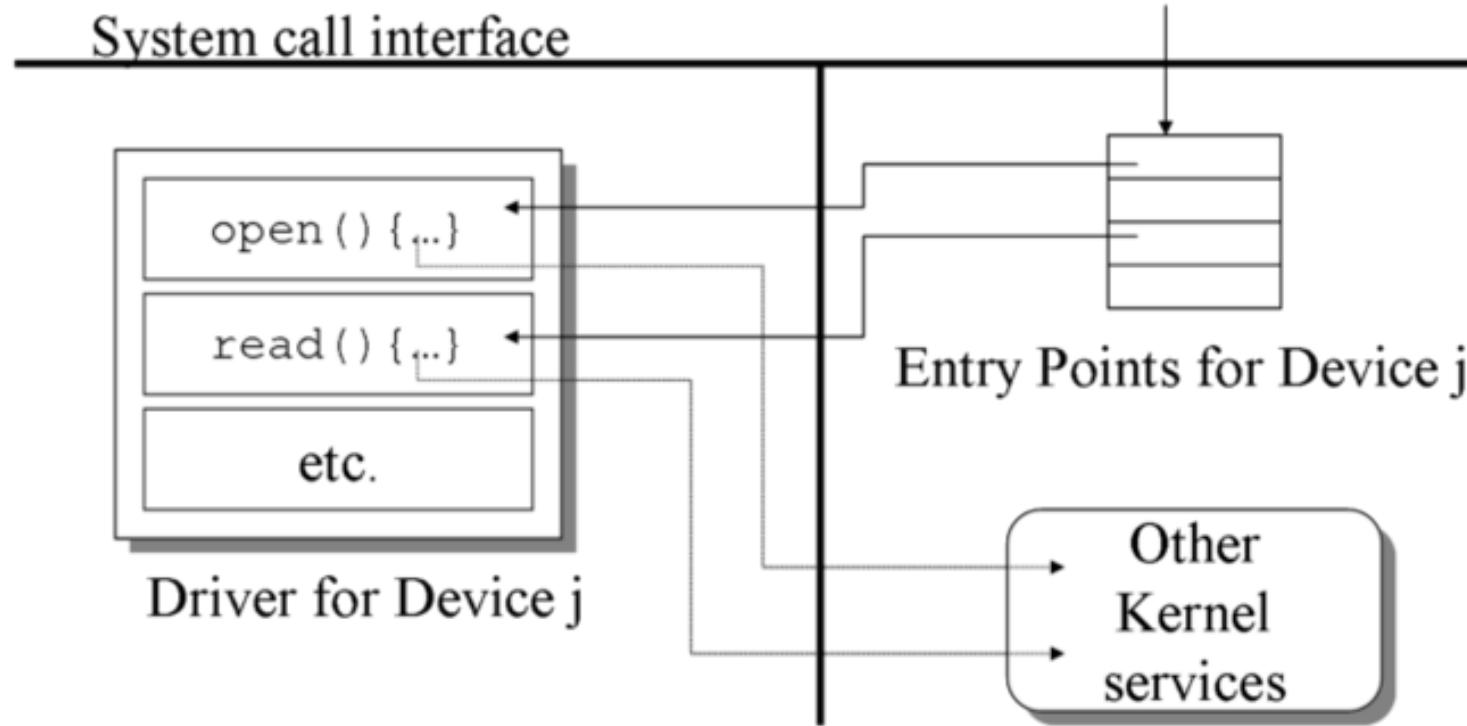
# 设备驱动具有的共性

- **核心代码**: 设备驱动是内核的一部分，出错将导致系统的严重错误。
- **与内核接口**: 设备驱动必须为内核提供一个标准接口。例如终端驱动为内核提供一个文件I/O接口
- **可使用内核的服务**: 可以使用标准的内核服务如内存分配、中断发送和等待队列等
- **动态可加载**: 在内核模块发出加载请求时加载；在不再使用时卸载，内核能有效地利用系统资源

# 驱动程序框架举例：

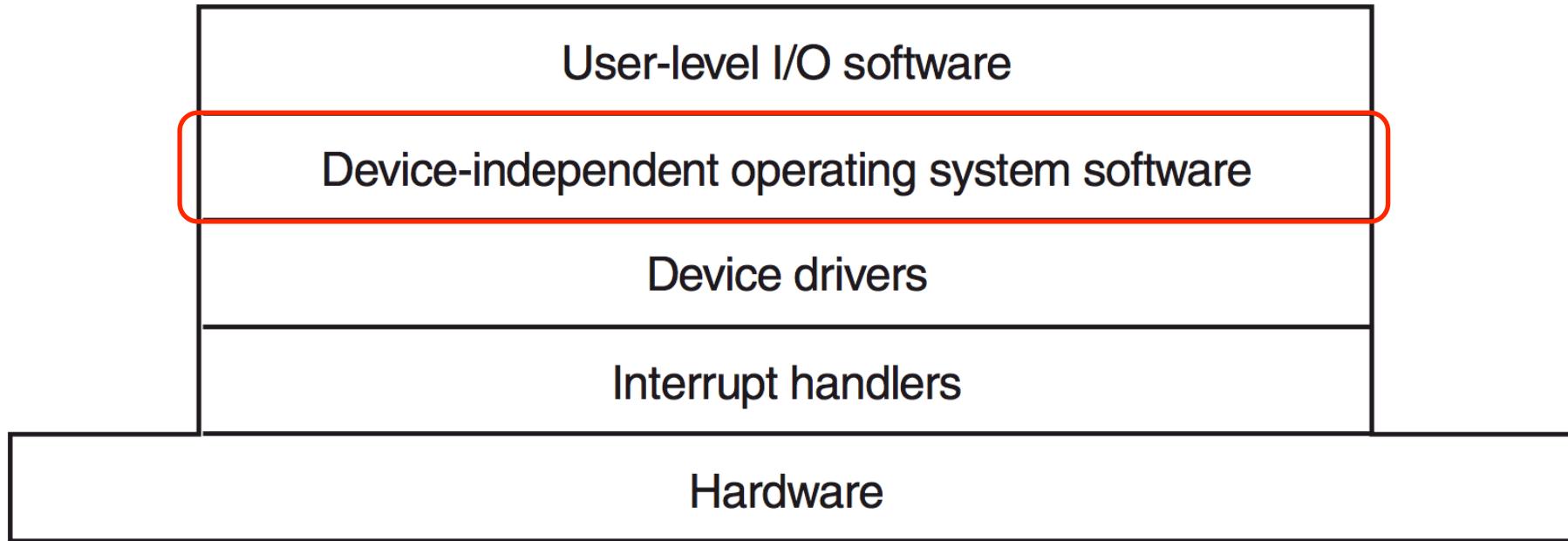


# 可动态加载的驱动程序



OS使用保存函数指针的表格，使得驱动代码  
可以动态加载，无需重新编译内核

# 设备无关软件



**Figure 5-11.** Layers of the I/O software system.

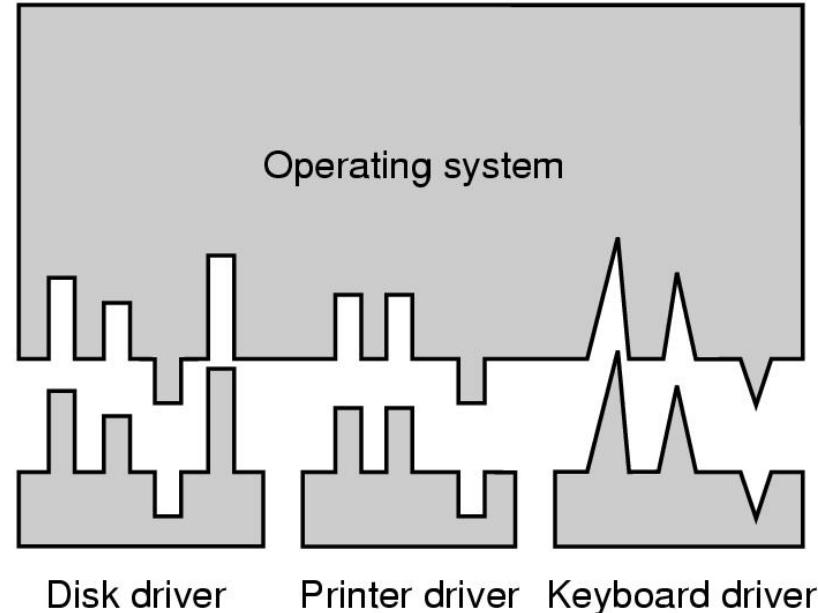
# 与设备无关的软件-提供通用IO操作

- 提供对所有设备公共的功能，向用户层软件提供统一的接口
- 设备相关功能由驱动封装

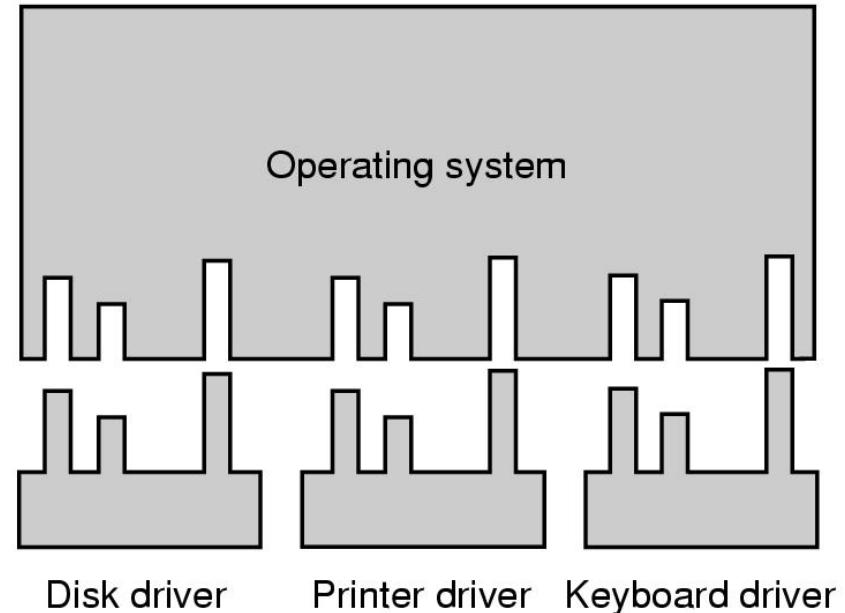
# 与设备无关的软件

- 提供和设备驱动程序的统一接口
- 缓冲
- 错误报告
- 分配与释放专用设备
- 提供与设备无关的块大小

# 与设备无关的软件(1):驱动程序的统一接口



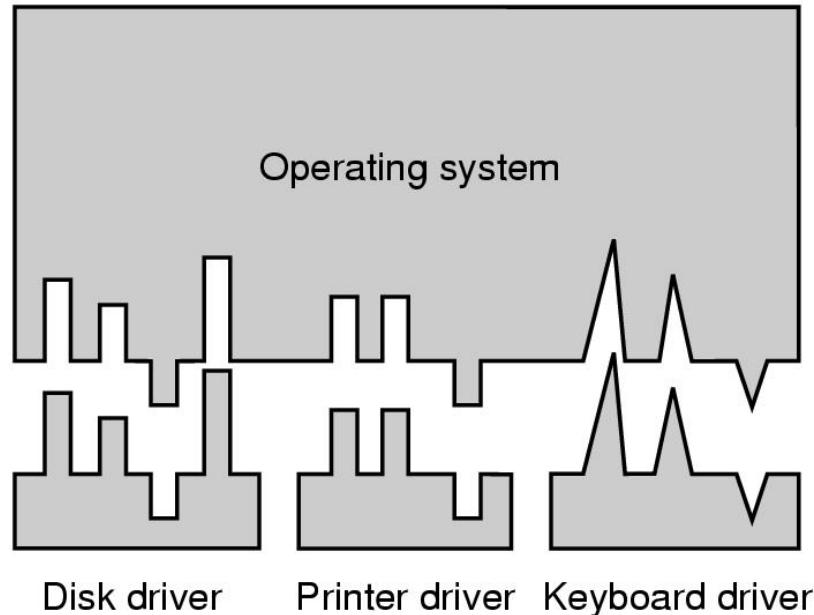
非标准的驱动程序接口



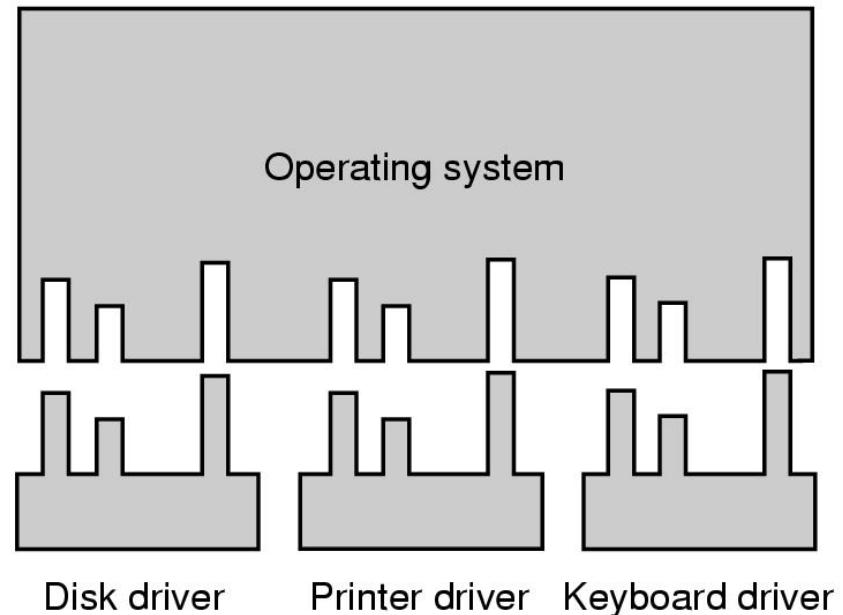
标准的驱动程序接口

接口：规定OS调用的驱动函数和驱动调用的系统函数

# 与设备无关的软件(1):驱动程序的统一接口



非标准的驱动程序接口

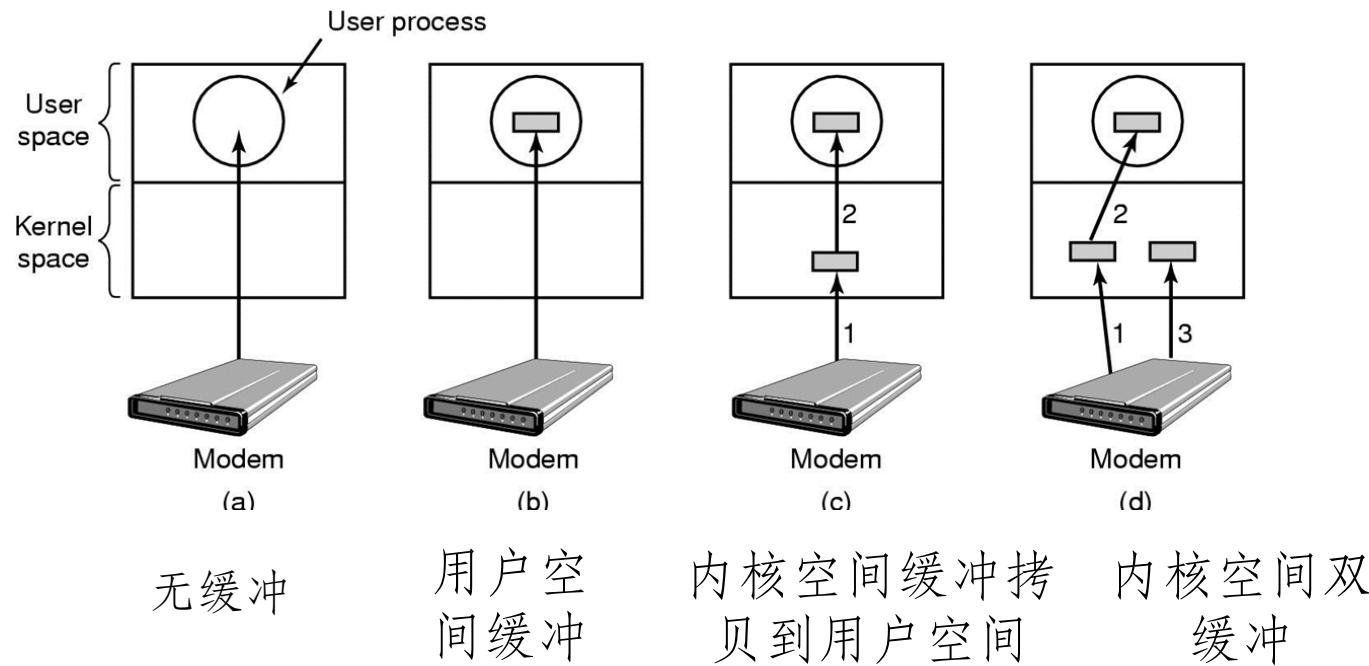


标准的驱动程序接口

驱动包含一个所有接口规定的函数指针表格，  
OS通过表格间接调用

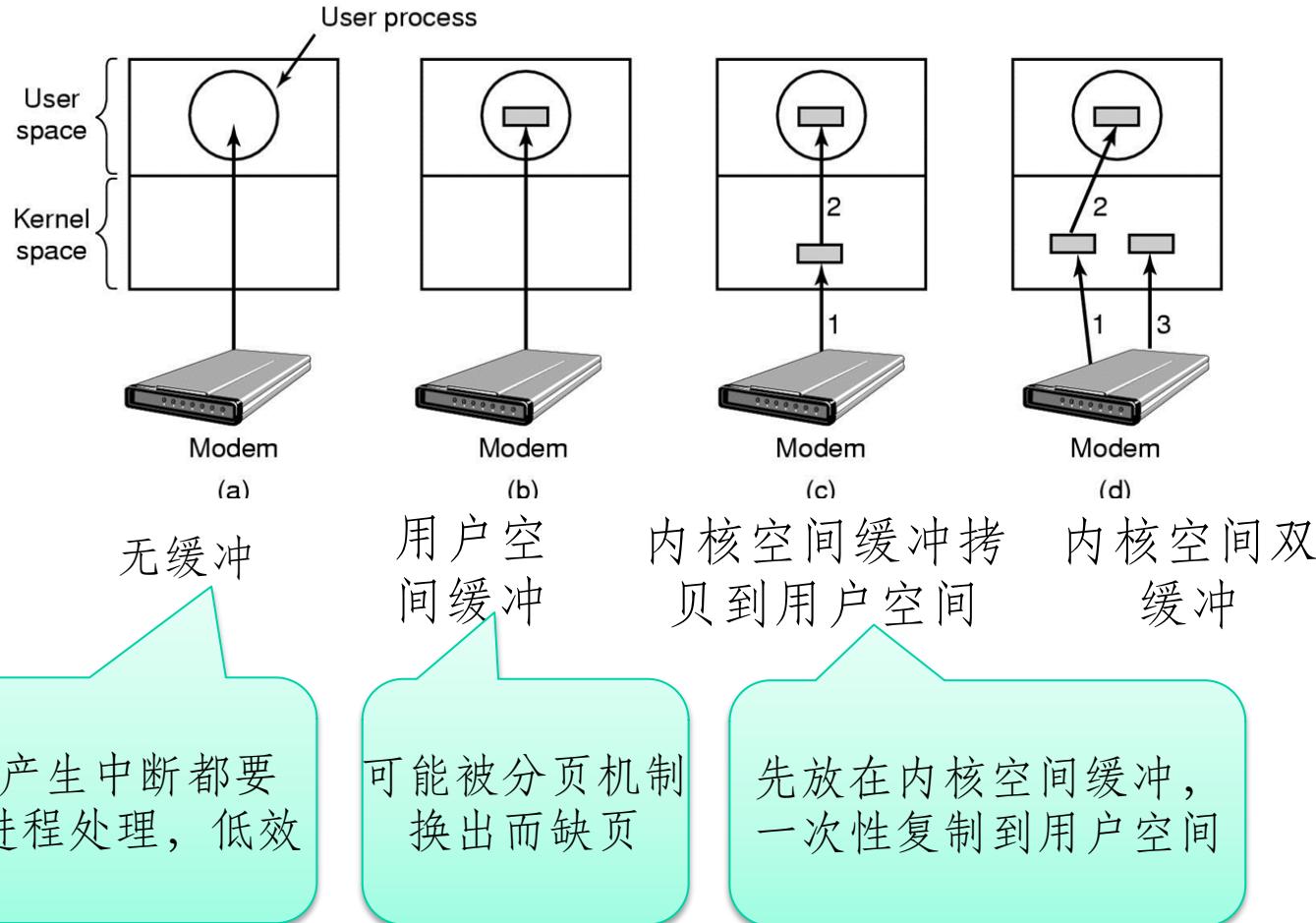
# 与设备无关的软件(2): 缓冲技术

- 缓冲技术可提高外设利用率。
  - 匹配CPU与外设的不同处理速度
  - 减少对CPU的中断次数。
  - 提高CPU和I/O设备之间的并行性。



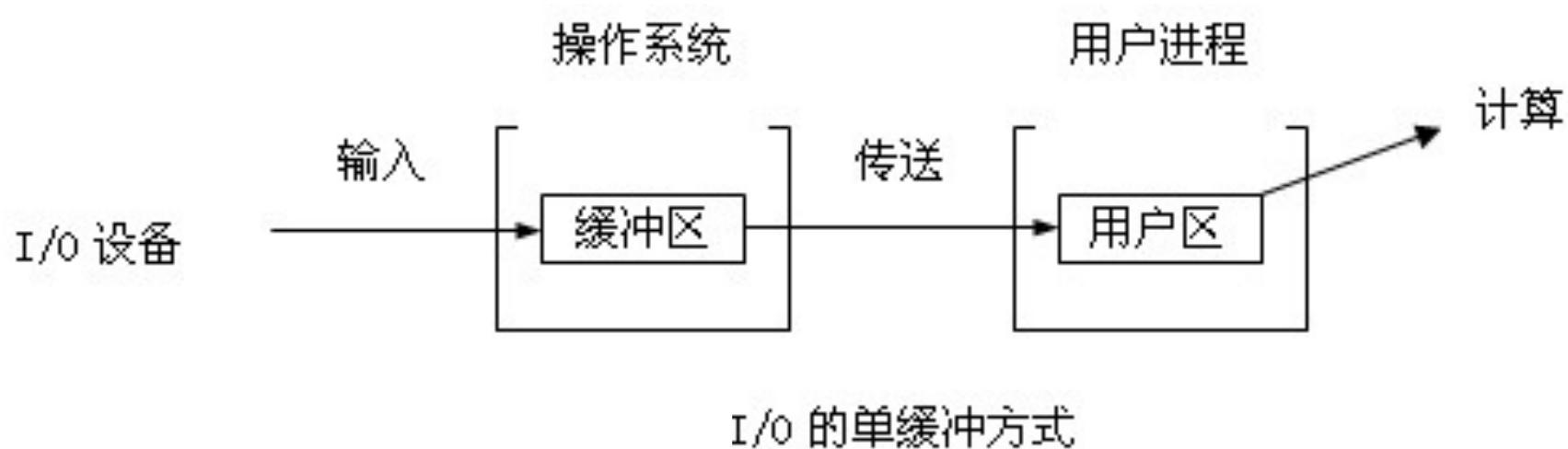
# 与设备无关的软件(2): 缓冲技术

- 缓冲技术可提高外设利用率。



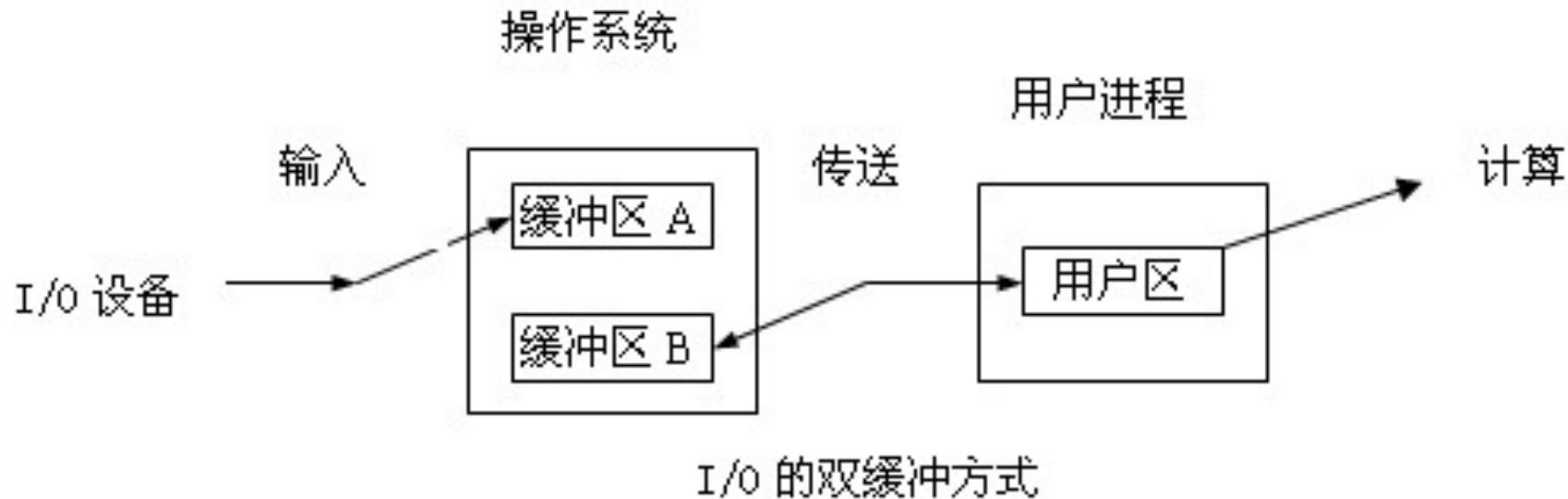
# 单缓冲(single buffer)

- 一个缓冲区，CPU和外设轮流使用，一方处理完之后接着等待对方处理。



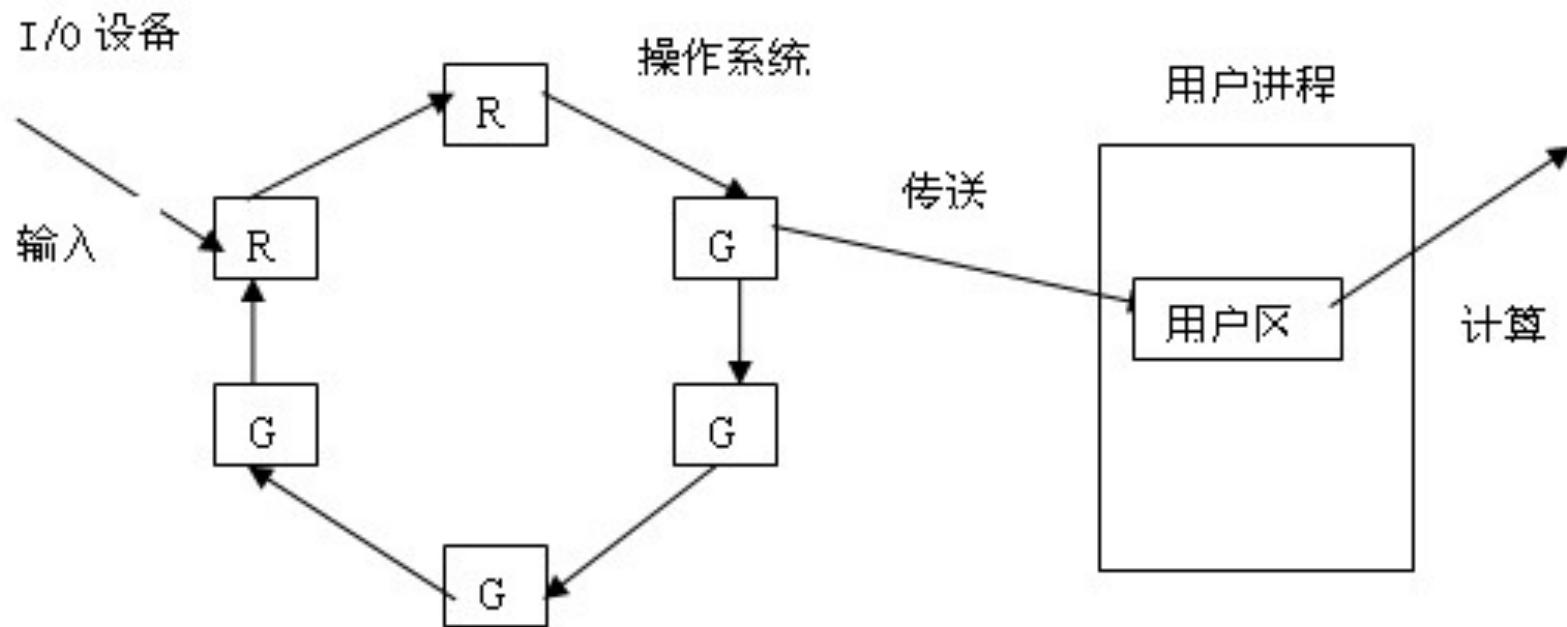
# 双缓冲(double buffer)

- 两个缓冲区，CPU和外设都可以连续处理而无需等待对方。要求CPU和外设的速度相近。



# 环形缓冲(circular buffer)

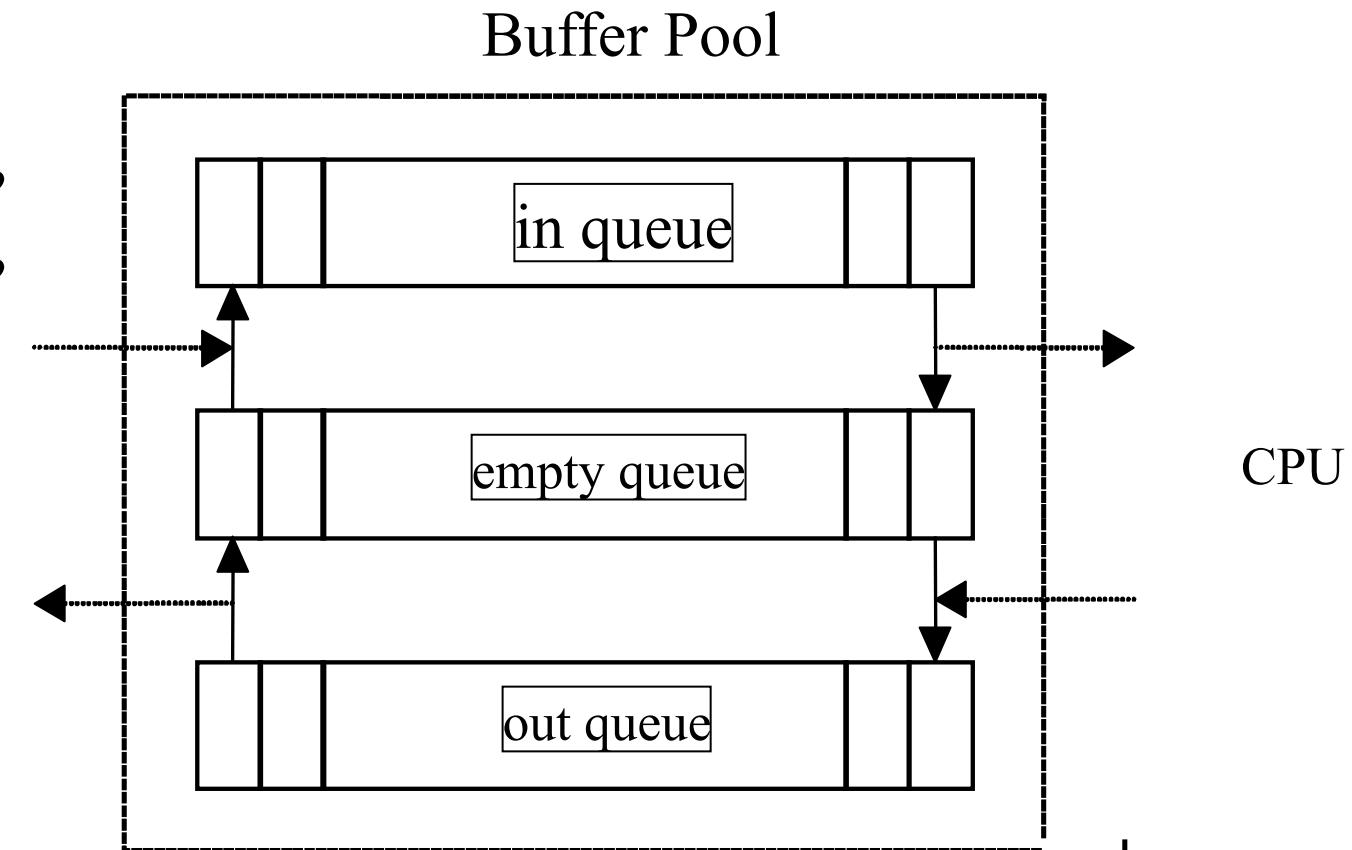
- 多个缓冲区，CPU和外设的处理速度可以相差较大。可参见“生产者—消费者问题”。



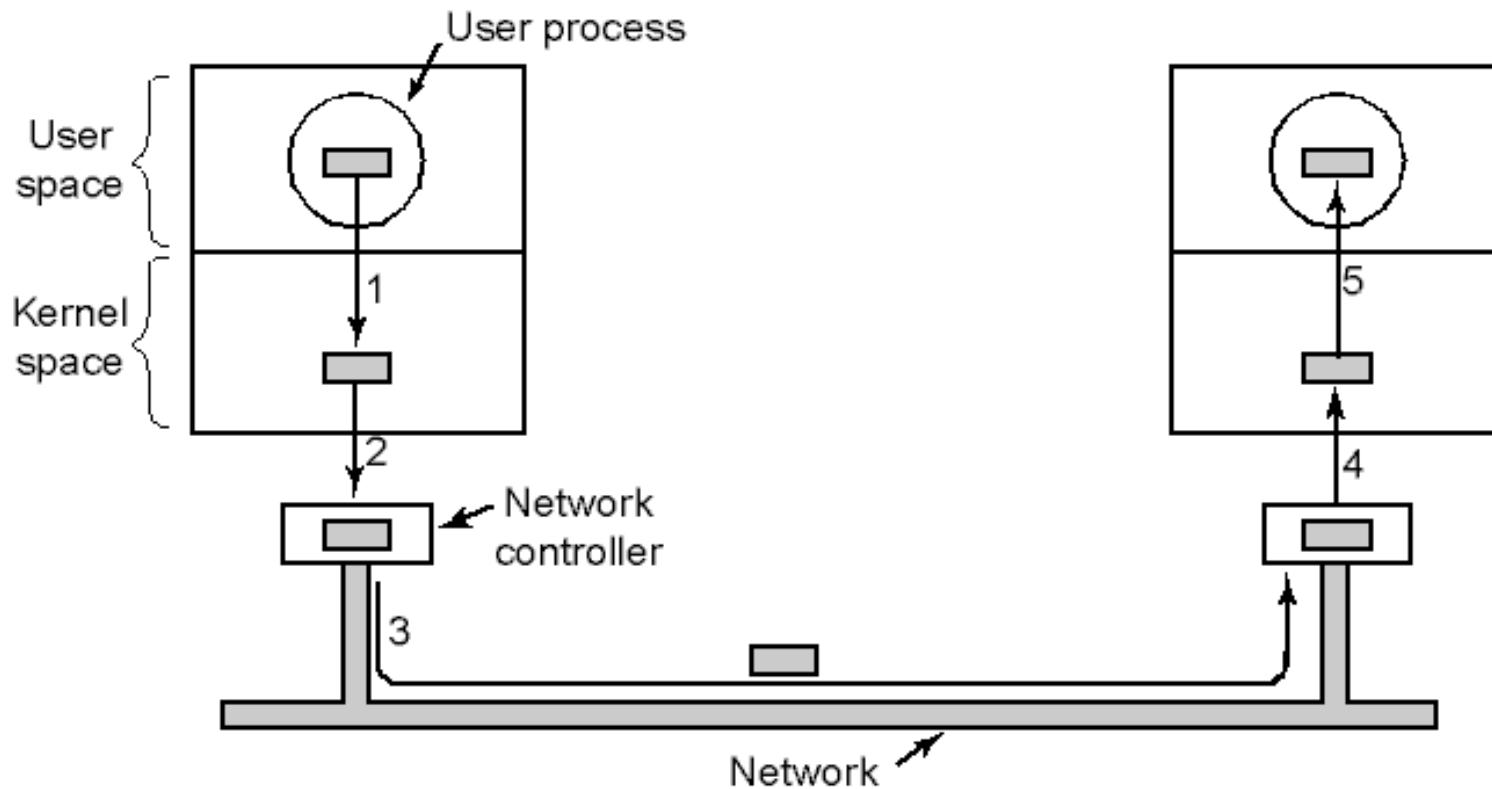
# 缓冲池(buffer pool)

- 缓冲区整体利用率高。
- 缓冲区队列

- 空闲缓冲区,
- 输入缓冲区,
- 输出缓冲区



# 缓冲的问题：多次复制一个数据包



多次缓冲方便了开发，也会带来性能的降级

《现代操作系统》P199

# 与设备无关的软件(3): 错误报告

- 编程错误
  - 错误的设备参数或者无效设备
  - 发送错误代码给调用者
- 实际的IO错误
  - 读取被破坏的磁盘块
  - 从驱动到设备无关软件，逐层传递解决或传递错误

# 与设备无关的软件(4): 设备分配

- 由于外设资源的有限，需解决进程间的外设共享问题，以提高外设资源的利用率。
- 有两种常见作法：
  - 1)在进程间切换使用外设，如键盘和鼠标；
  - 2)通过一个虚拟设备把外设与应用进程隔开，只由虚拟设备来使用设备。

# 数据结构

- 设备控制表(DCT, Device Control Table): 每个设备一张，描述设备特性和状态。反映设备的特性、设备和控制器的连接情况。
- 控制器控制表(COCT, COntroller Control Table) : 每个设备控制器一张，描述I/O控制器的配置和状态。如DMA控制器所占用的中断号、DMA数据通道的分配。
- 通道控制表(CHCT, CHannel Control Table): 每个通道一张，描述通道工作状态。

- 系统设备表(SDT, System Device Table): 系统内一张，反映系统中设备资源的状态，记录所有设备的状态及其设备控制表的入口。SDT表项的主要组成：
  - DCT指针：指向相应设备的DCT；
  - 设备使用进程标识：正在使用该设备的进程标识；
  - DCT信息：为引用方便而保存的DCT信息，如：设备标识、设备类型等；

# 设备分配时应考虑的因素

- 设备固有属性：独享、共享、虚拟设备
- 设备分配算法：先来先服务、优先级高者优先
- 设备分配中的安全性(safety)：死锁问题
  - 安全分配（同步）：在设备分配中防止死锁，进程发出I/O请求之后，进入阻塞，直到I/O完成。CPU和I/O串行工作，打破了死锁条件，但是效率低。
  - 不安全分配（异步）：设备在分配时不考虑可能产生的死锁，进程发出I/O请求后，仍然继续运行，可继续请求其他I/O设备。需要进行安全性检查，但进程执行效率相对较高
- 设备独立性
  - 用户程序的设备独立性：用户程序使用逻辑设备名，系统实际执行时，映射到物理设备名。
  - I/O软件的设备独立性：除了直接与设备打交道的低层软件外，其余部分软件不依赖于设备，可提高设备管理软件效率。

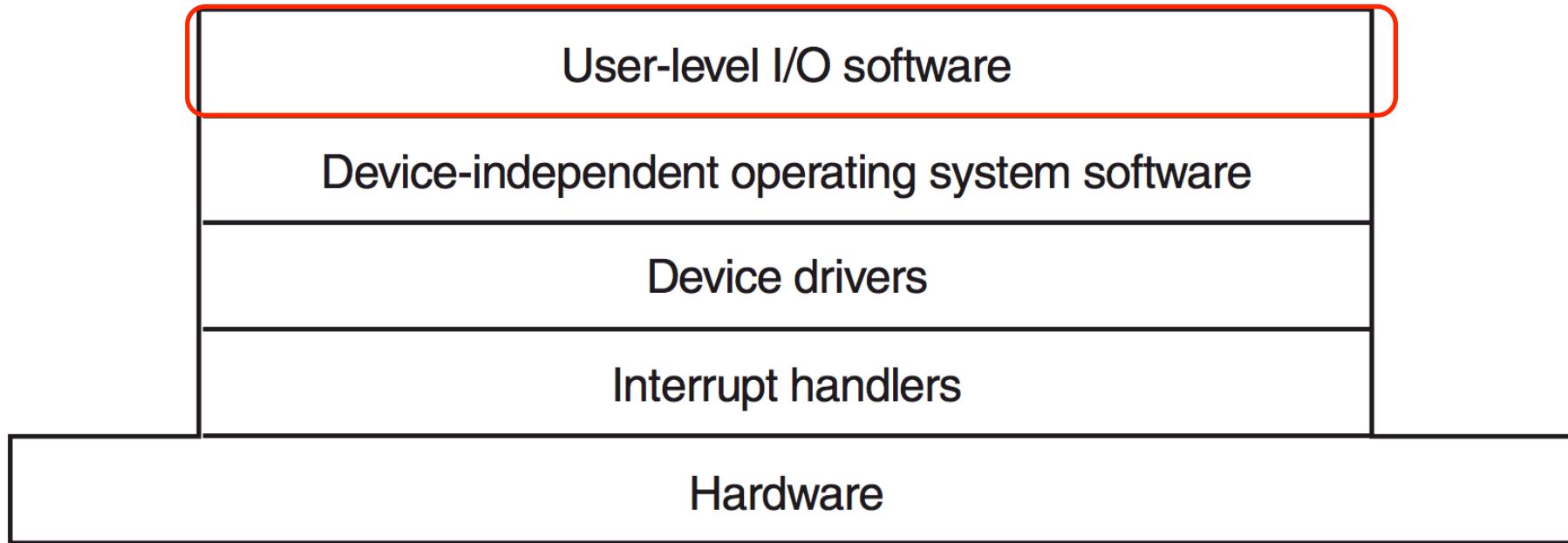
# 单通路I/O系统的设备分配

- 单通路：一个设备对应一个控制器，一个控制器对应一个通道
  - 分配设备
    - 根据物理设备名查找系统设备表SDT，从中找到设备控制器表DCT，如果设备忙，则进入等待队列；否则，计算是否产生死锁，进行分配。
  - 分配设备控制器
    - 将设备分配给进程后，在DCT中找到该设备相连的设备控制器表COCT，如果控制器空闲，则分配；否则，进入等待队列。
  - 分配通道
    - 从COCT中找到相连的通道控制表CHCT，如果通道空闲，则分配，否则，进入等待队列。

# 多通路I/O系统的设备分配

- 一个设备与几个控制器相连，一个控制器与几个通道相连
- 设备分配的过程类似单通路
  - 分配设备
  - 分配控制器
  - 分配通道

# 设备无关软件



**Figure 5-11.** Layers of the I/O software system.

# 用户空间的I/O软件：

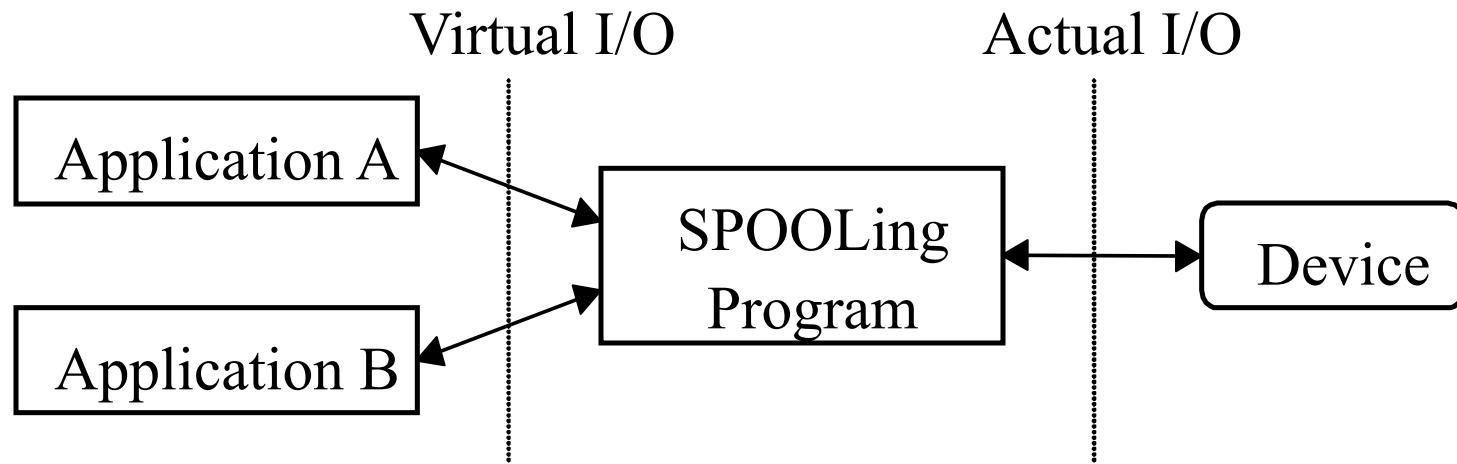
- 提供库函数以支撑系统调用
  - 库函数运行时会和应用链接在一起
  - write、printf、scanf
- 假脱机技术（spooling）
- 数据格式化

# 用户空间的I/O软件：假脱机技术

- 利用假脱机技术(SPOOLing, Simultaneous Peripheral Operation On Line, 也称为虚拟设备技术)可把独享设备转变成具有共享特征的虚拟设备，从而提高设备利用率。
- 假脱机技术的引入
  - 在多道程序系统中，专门利用一道程序（SPOOLing程序，一个daemon守护进程）来完成对设备的I/O操作。

- SPOOLing程序和外设进行数据交换：实际I/O
  - SPOOLing程序预先从外设读取数据并加以缓冲，在以后需要的时候输入到应用程序；
  - SPOOLing程序接受应用程序的输出数据并加以缓冲，在以后适当的时候输出到外设。
- 应用程序进行I/O操作时，只是和SPOOLing程序交换数据，可以称为“虚拟I/O”。应用程序实际上是从SPOOLing程序的缓冲池中读出数据或把数据送入缓冲池，而不是跟实际的外设进行I/O操作。

# SPOOLing系统组成



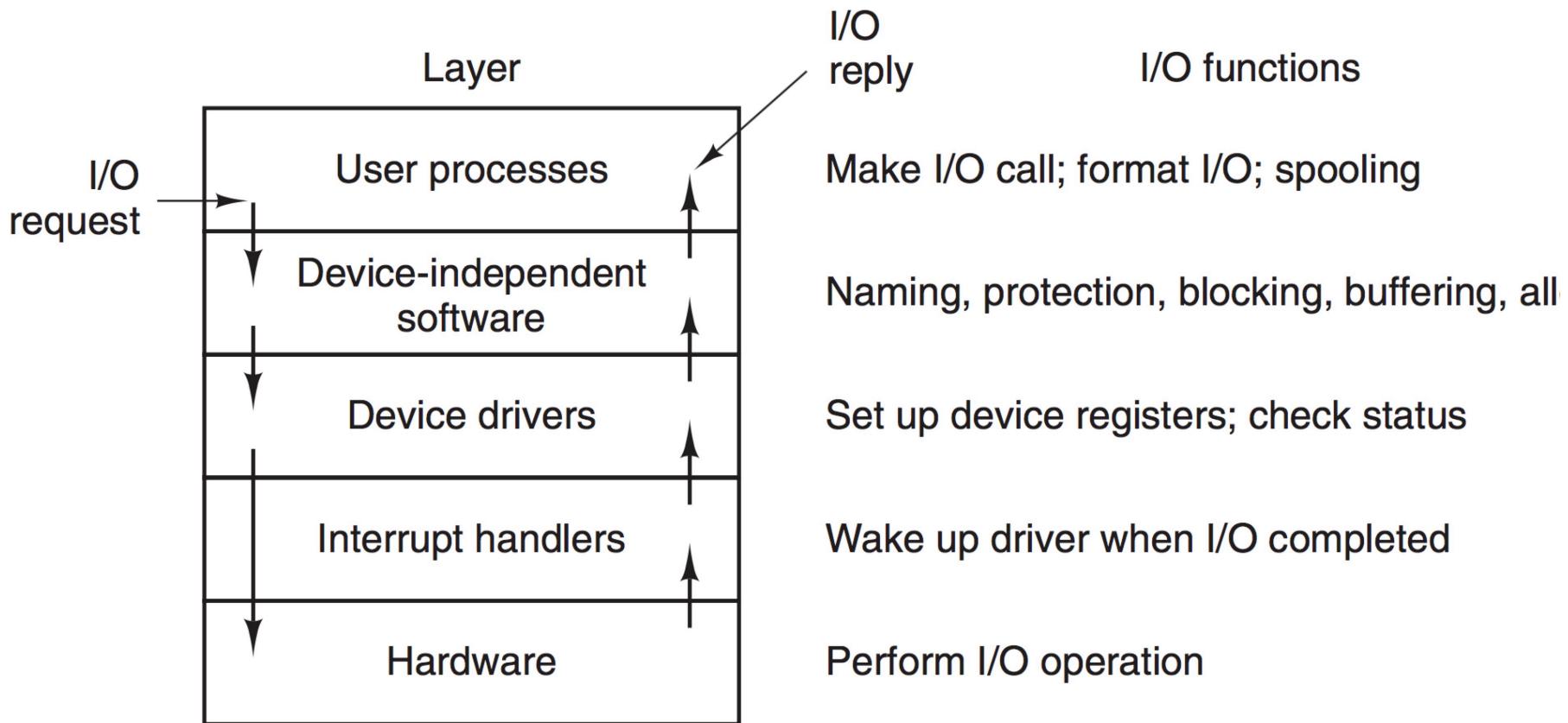
# 特点

- 高速虚拟I/O操作：应用程序的虚拟I/O比实际I/O速度提高，缩短应用程序的执行时间（尽快完成计算，并释放占用的计算机资源）。另一方面，程序的虚拟I/O操作时间和实际I/O操作时间分离开来。
- 实现对独享设备的共享：由SPOOLing程序提供虚拟设备，可以对独享设备依次共享使用。

# 举例

- 打印机设备和可由打印机管理器管理的打印作业队列。
- 例如：Windows NT中，应用程序直接向针式打印机输出需要15分钟，而向打印作业队列输出只需要1分钟，此后用户可以关闭应用程序而转入其他工作，在以后适当的时候由打印机管理器完成15分钟的打印输出而无需用户干预。

# I/O软件的总结

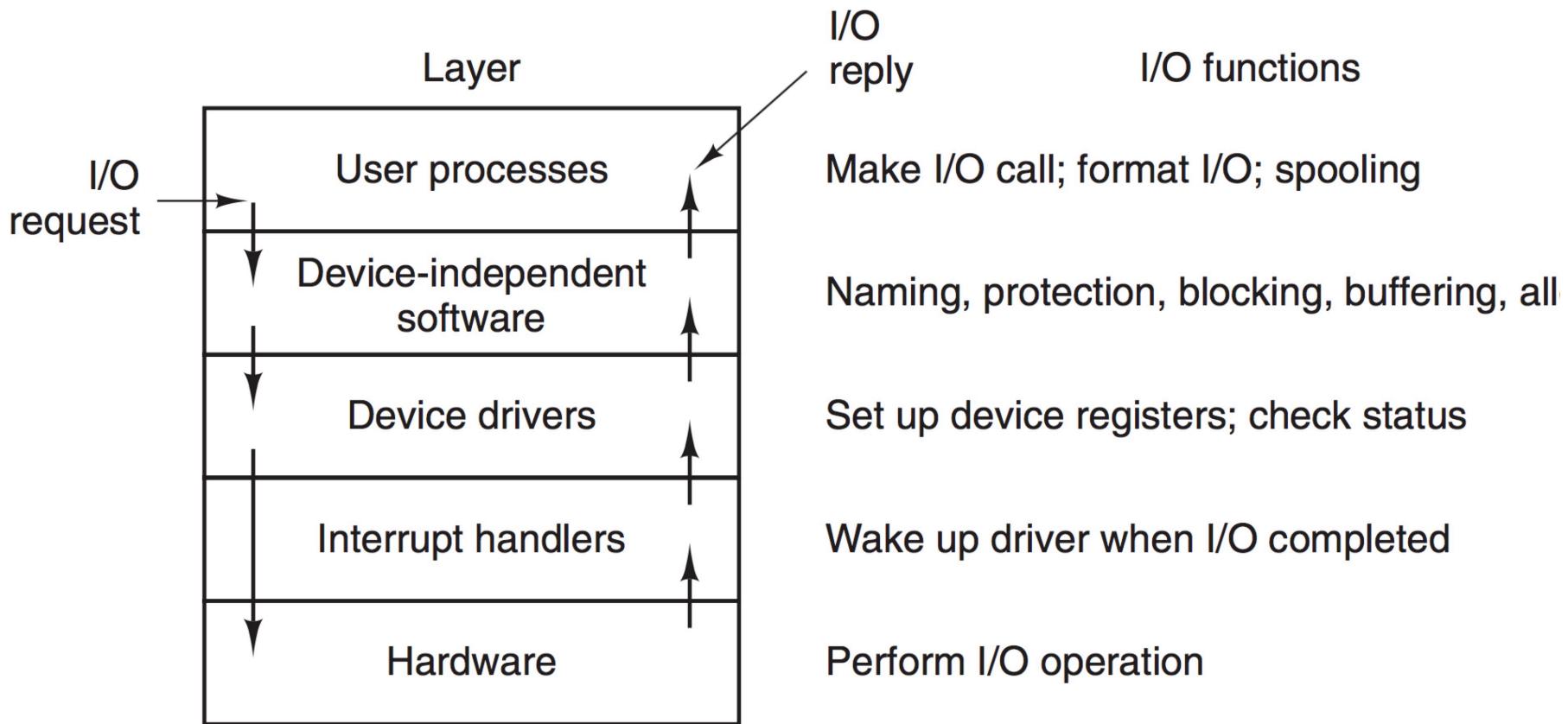


**Figure 5-17.** Layers of the I/O system and the main functions of each layer.

# 练习题

- 以下各项工作是在四个IO软件层的那一层完成的？
- 1. 为磁盘读操作计算磁道、扇区、磁头
- 2. 向设备寄存器写命令
- 3. 检查用户是否允许使用设备
- 4. 将二进制证书转换为ASCII码以便打印

# I/O软件的总结



**Figure 5-17.** Layers of the I/O system and the main functions of each layer.

# 练习题

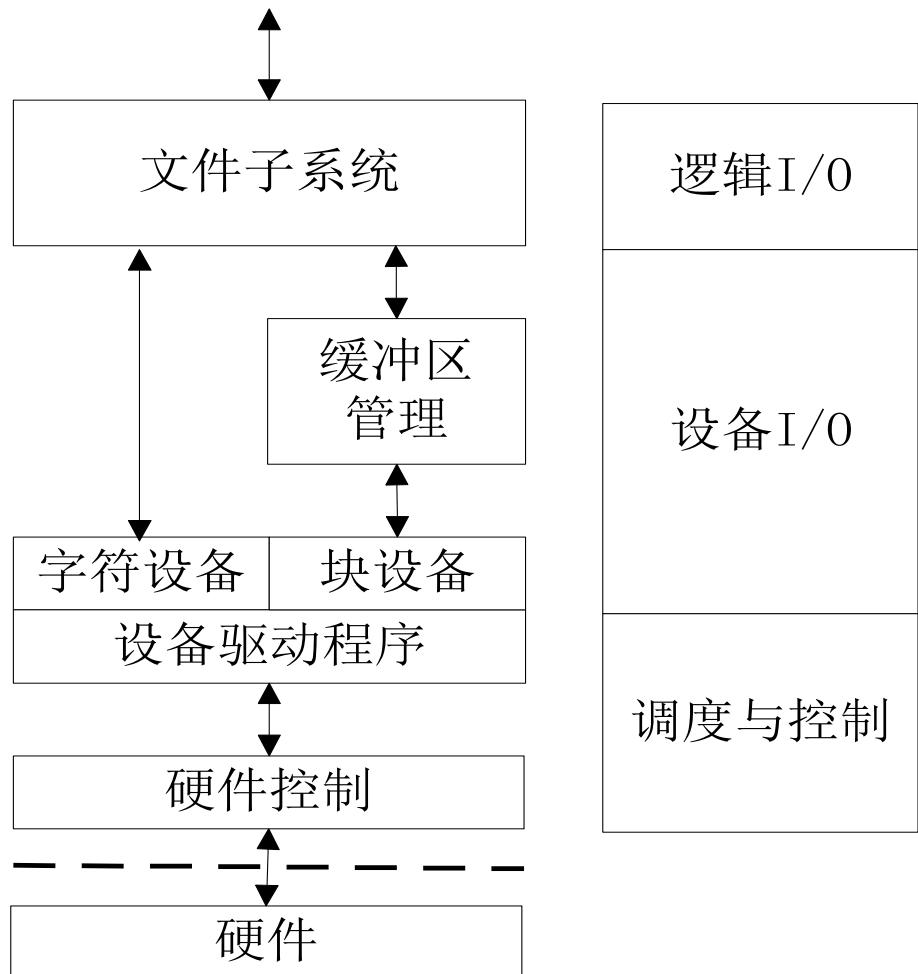
- 以下各项工作是在四个IO软件层的那一层完成的？
- 1. 为磁盘读操作计算磁道、扇区、磁头- **驱动**
- 2. 向设备寄存器写命令-**驱动**
- 3. 检查用户是否允许使用设备-**设备无关软件**
- 4. 将二进制证书转换为ASCII码以便打印-**用户层软件**

# 内容提要

- I/O管理概述
- I/O硬件基础
- I/O软件原理
- I/O软件概述
- 设备管理实例

# UNIX的设备管理

- UNIX的外设与特殊文件对应，由文件系统按文件管理方式对其进行管理，向上提供一个与文件系统统一的接口。



- 按设备I/O的不同情况，UNIX系统的I/O分成2种：
  - 无缓存I/O (UnBuffered I/O): 在进程I/O区域与系统I/O模块间直接进行数据交换；
  - 有缓存I/O (Buffered I/O): 有缓存I/O要经过系统的缓冲区管理机构，分成系统缓冲区(system buffer caches)和字符队列(character queues)两种。

# 块设备的缓冲区管理

- 采用缓冲池结构: 用于磁盘等外存的缓存。
- 块设备缓冲区结构: 缓存块是缓存使用的基本单位, 与外设数据块对应; 每个缓存块由两部分组成: 缓冲控制块和缓冲数据区。前者用于缓冲区管理, 而后者用于存放数据 (长度为512字节或1024字节)。
- 缓冲控制块: 也称为缓冲首部(buffer header)。内容包括: 逻辑设备号, 物理块号, 缓冲区状态(如空闲、延迟写、锁定等标志), 指向缓冲数据区的指针, 哈希队列的前后向指针, 空闲队列的前后向指针。

- 缓冲区管理的相关数据结构
  - 空闲缓冲队列：系统的所有空闲缓冲区列表；
  - 设备I/O请求队列：正与外设进行I/O操作的缓存块列表；一个缓存块必须处于空闲或操作状态；
  - 设备缓冲区队列：与各外设相关的缓存块列表，其中有缓存数据；

## ■ 缓冲区检索和置换方式

- 设备缓冲区队列为Hash队列：为了检索方便，设备缓冲区队列为一个按(逻辑设备号，物理块号)组织的Hash队列。把逻辑设备号和物理块号之和对64取模作为哈希函数值，据此建立多个哈希队列（64个队列）。
- 缓存块可同时链入设备缓冲区队列和空闲缓冲队列：一个缓存块在分配给一个外设后，一直与该外设相关(即使该缓存块在空闲缓冲队列中)，直到分配给另一外设。即：设备释放缓冲区后，该缓冲区可处于“延迟写”状态，等待被写入到外设；该缓冲区被重新分配之前，要将其写入到外设。

## ■ 缓冲区数据读写：

- 外设与核心缓冲区之间：

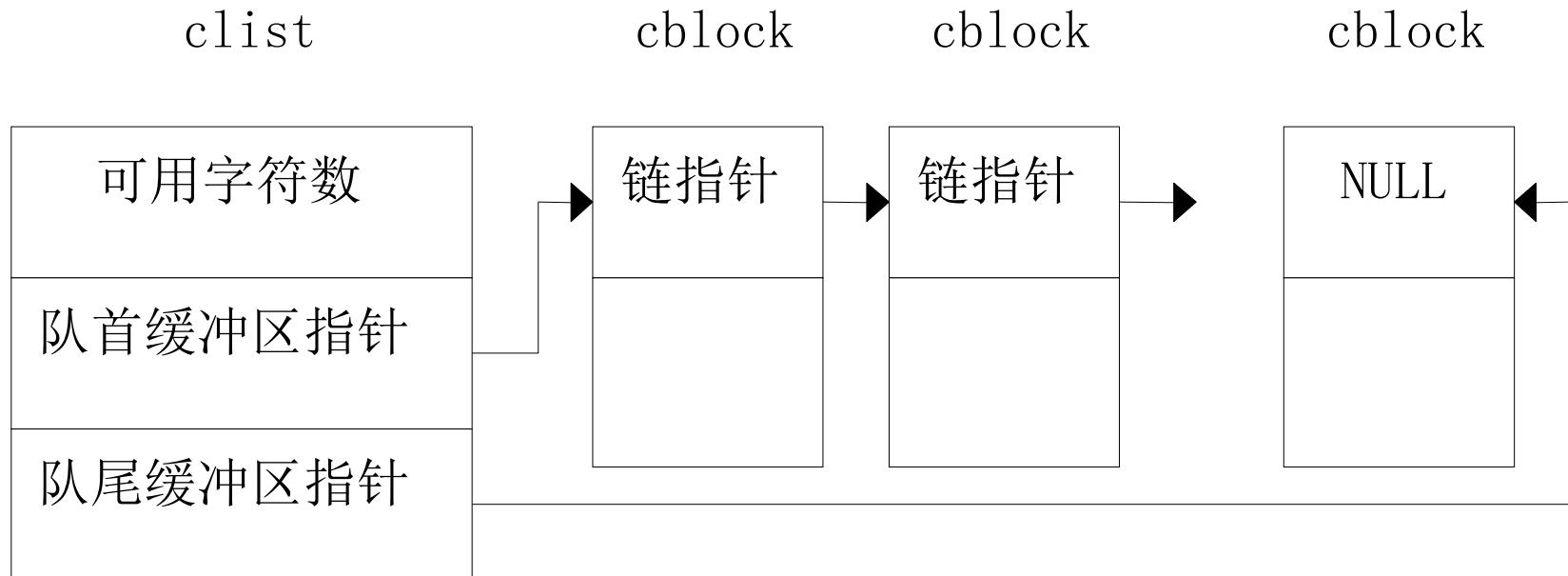
- 一般读和预先读（适用于对文件的顺序访问）：一般读是从外设读入指定的数据块；预先读是在一般读的基础上，异步读入另一块，以提高数据读取速度；
  - 一般写（立即起动I/O并等待完成）、异步写（立即起动I/O而不等待完成，以提高写速度）、延迟写（不立即起动I/O，以减少不必要的I/O操作，但系统故障时会产生数据错误）

- 核心缓冲区与进程的用户区：使用DMA方式在缓存与用户进程间进行内存到内存的数据传送，可节约CPU时间，但要占用总线。

# 字符设备的缓冲区管理

- 字符缓冲区采用缓冲池结构，构成一个字符队列 (Character Queue)，它不同于块设备缓冲区的多次读写，缓冲区中每个字符只能读一次，读后被破坏。

- 字符缓冲池的基本分配单位为字符缓冲区 cblock：供各种字符设备（的设备驱动程序）使用。
  - 每个缓冲区大小为70字节，包括：第一个字符和最后一个字符的位置（便于从开头移出字符和向末尾添加字符），指向下一缓冲区的指针c\_next，可存放64字符的数据区



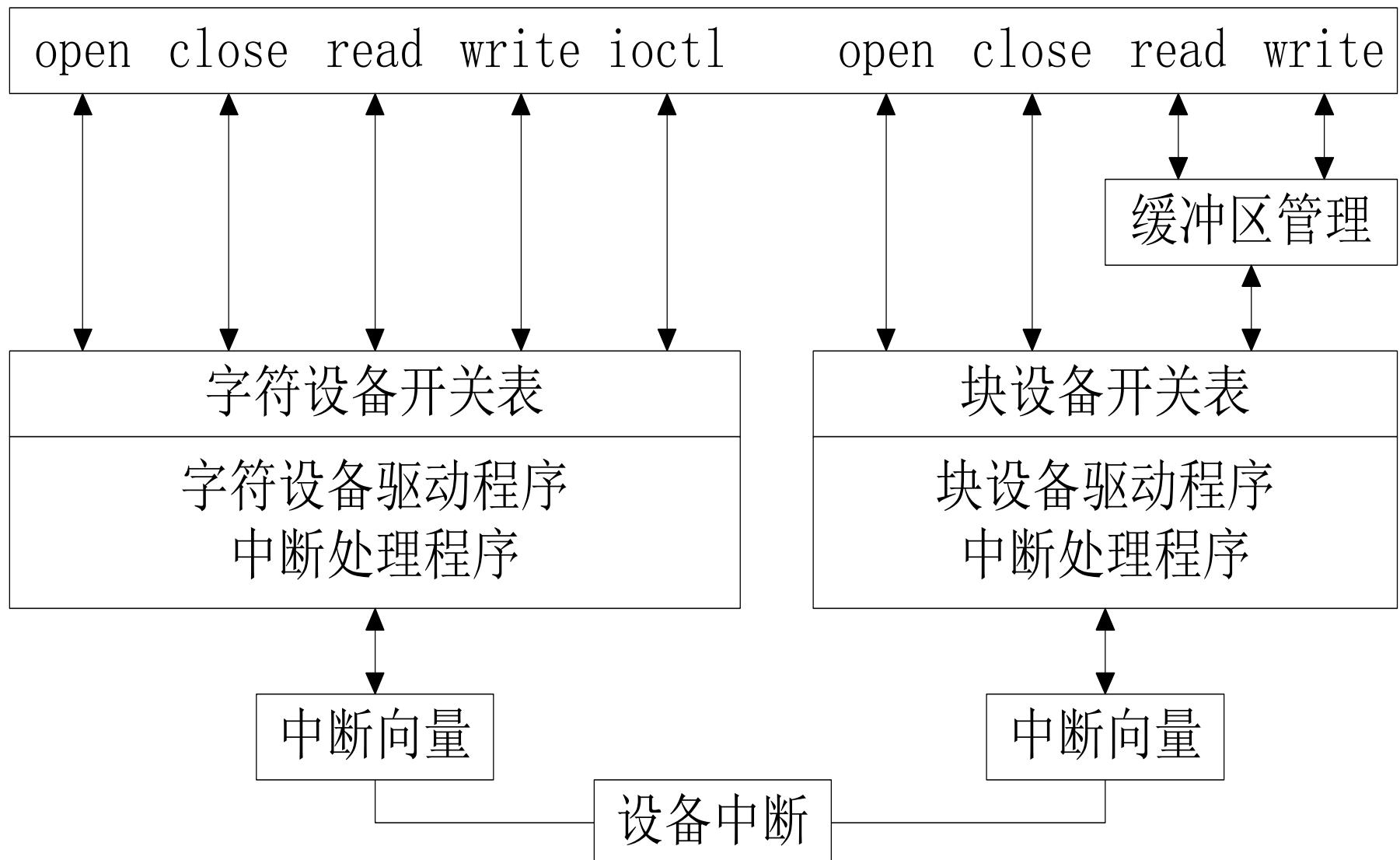
## ■ 字符设备缓冲区的操作

- 空闲缓冲区操作：申请空闲缓冲区、释放空闲缓冲区；
- 字符设备缓冲队列操作：从队首读出一个字符、向队尾写入一个字符、读出队列缓冲区的所有字符、向队尾加入一个有数据的缓冲区、从队首读出n个字符、向队尾写入n个字符；

# 设备开关表(switch table)

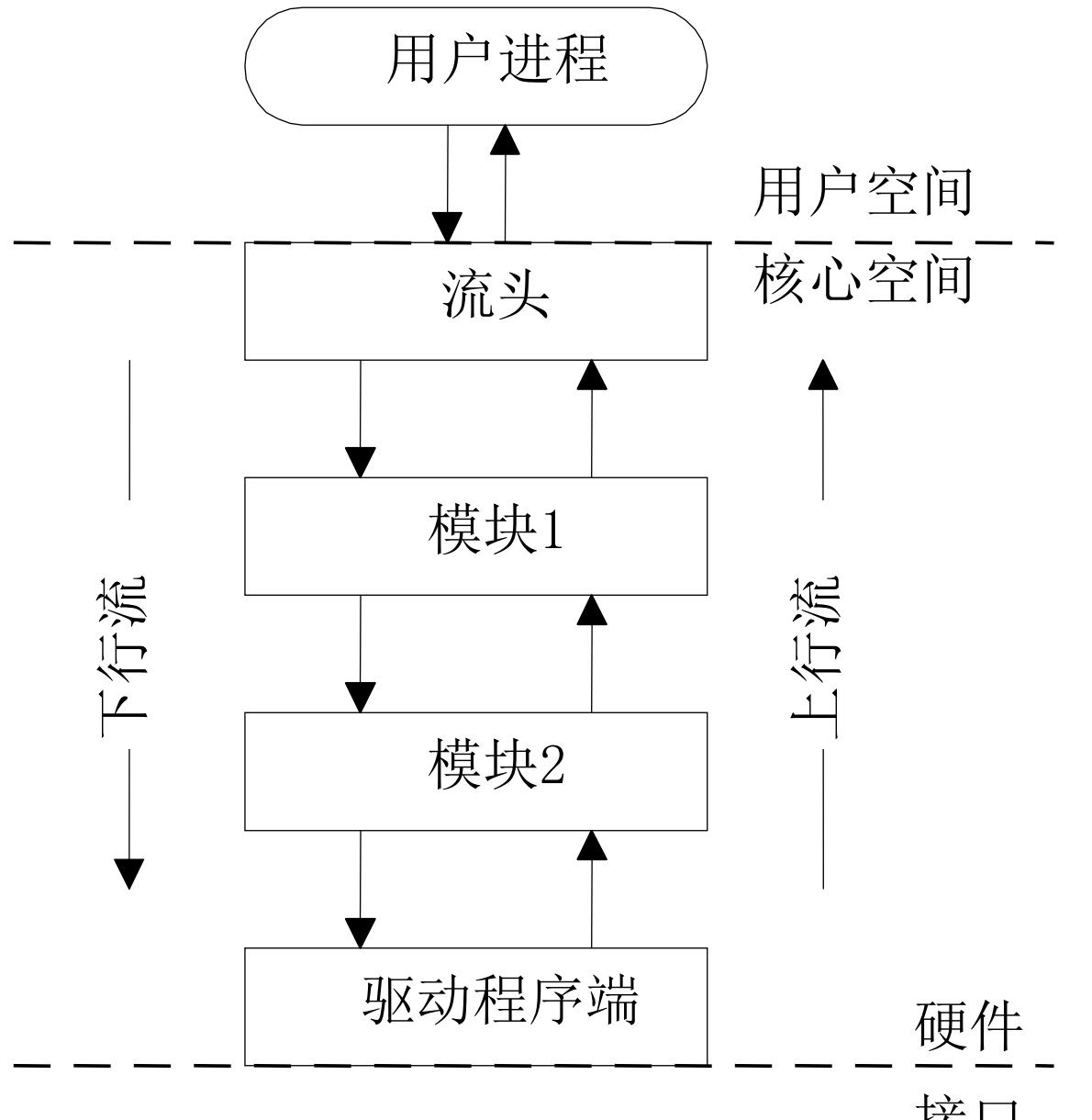
- UNIX设备驱动程序通过相应的块设备开关表和字符设备开关表描述向上与文件系统的接口。
- 开关表是每个设备驱动程序的一系列接口过程的入口表，给出了一组标准操作的驱动程序入口地址，文件系统可通过开关表中的各函数入口地址转向适当的驱动程序入口。

# 文件系统



# 流机制(streams)

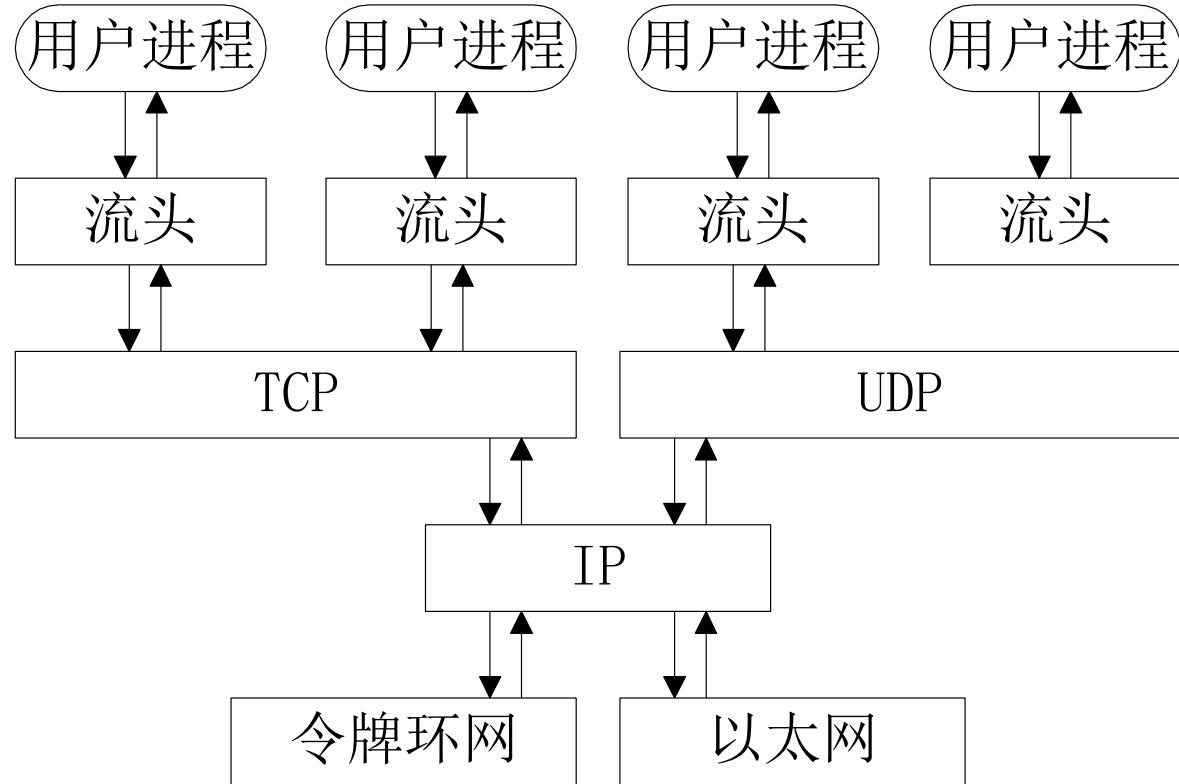
- 流的引入：流的引入是为了解决内核与驱动程序抽象层次过高，而引起的驱动程序功能大量重复。它可提供一个完全基于消息的模块化的驱动程序编写方法。
- 流的定义：流是一组系统调用、内核资源和创建、使用及拆除流的例程的集合，构成一个数据传输通道，两端为读队列和写队列。
- 流的结构：上行流(upstream)和下行流(downstream)



教材P229  
硬件  
接口

# 流的多路复用机制

- 上部多路复用器：向上连接多个流；
- 下部多路复用器：向下连接多个流；
- 双向多路复用器：同时支持向上连接的多个流和向下连接的多个流；



# NT的I/O系统结构

- I/O子系统：实现文件化的I/O函数,提供的I/O特性有：
  - 通常的打开、关闭和读写函数；
  - 异步I/O：应用进程在发出I/O请求后，不需等待I/O完成，可继续其它工作；
  - 映射文件I/O：把文件作为进程虚拟空间的一部分进行直接访问；
  - 快速I/O：不通过I/O管理器，直接向驱动程序发出I/O请求；

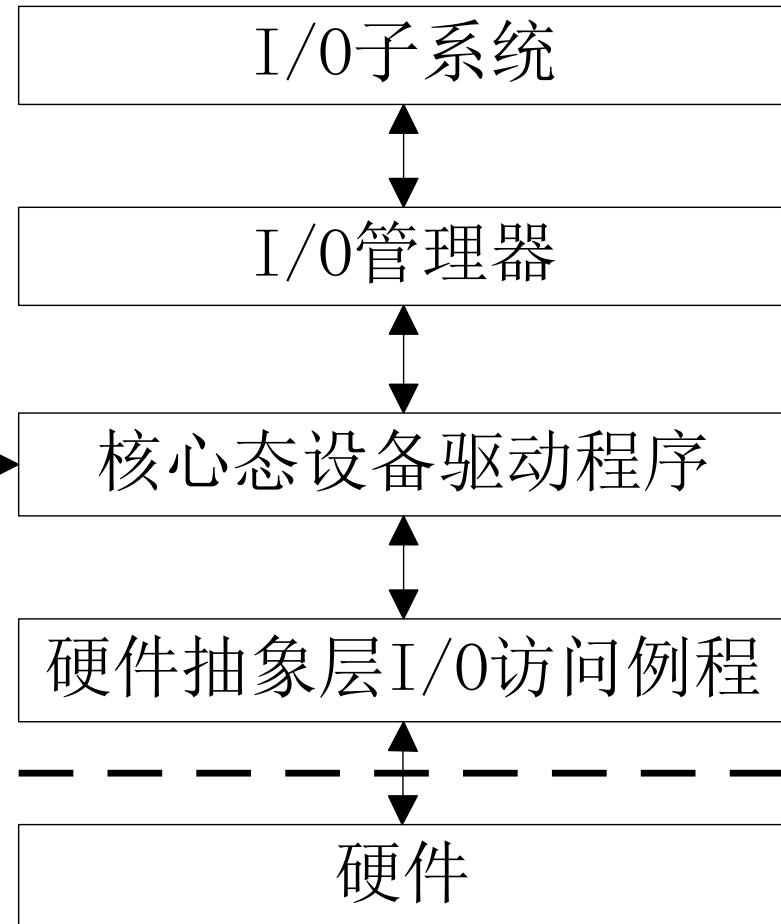
# NT的I/O系统结构

- I/O管理器：依据抽象I/O操作创建和传送I/O请求包(IRP)；
- 核心态设备驱动程序：将I/O请求包转化为对硬件设备的特定控制请求；
- 驱动程序支持例程：供设备驱动程序调用，以完成I/O请求；
- 硬件抽象层I/O访问例程：隔离驱动程序与硬件平台，以提高可移植性(相同体系结构上的二进制可移植和NT支持平台间的源代码可移植)；

逻辑I/O

设备I/O

调度与控制



# NT的设备驱动程序

- NT采用分层驱动程序的思想
  - 只有最底层的硬件设备驱动程序访问硬件设备，高层驱动程序都是进行高级I/O请求到低级I/O请求的转换工作；
  - 各层驱动程序间的I/O请求通过I/O管理器进行。

- 文件系统驱动程序：实现文件I/O请求到物理设备I/O请求的转换；
- 文件系统过滤器驱动程序：截取文件系统驱动程序产生的I/O请求，执行另外处理，并发出相应的低层I/O请求。如：容错磁盘驱动程序；
- 类驱动程序(**class driver**)：实现对特定类型设备的I/O请求处理。如：磁盘、磁带、光盘等；
- 端口驱动程序(**port driver**)：实现对特定类型I/O端口的I/O请求处理。如：SCSI接口类型；
- 小端口驱动程序：把对端口类型的一般I/O请求映射到适配器类型；
- 硬件设备驱动程序(**hardware device driver**)：直接控制和访问硬件设备；

# 设备驱动程序的组成

- 初始化例程：I/O管理器在加载驱动程序时，利用初始化例程创建系统对象；
- 调度例程集：实现设备的各种I/O操作。如：打开、关闭、读取、写入等；
- 启动I/O例程：初始化与设备间的数据传输；
- 中断服务例程(ISR)：设备(软)中断时的调用例程；要求快速简单；
- 中断服务延迟过程调用(DPC)例程：以内核线程方式执行ISR执行后的中断处理工作；

# 小结

- I/O管理的目标与功能
- I/O硬件基础
  - 控制器
- I/O软件基础
  - 四种I/O控制方式：
    - 程序控制、中断、DMA、通道
- I/O软件概述
  - 中断处理程序
  - 设备驱动程序
  - 与设备无关的软件
    - 缓冲区管理
    - 设备分配
  - 用户空间的I/O软件：假脱机技术



# 电源管理

- 能耗的降低发展缓慢
  - 和性能的摩尔定律相比，止步不前
- 降低能耗的办法
  - 提高硬件的能量效率
    - PC的能量效率85%
  - 在设备不使用的时候关闭
  - 应用程序优化，主动降低能量使用

# 电源管理 – 笔记本的电能耗费

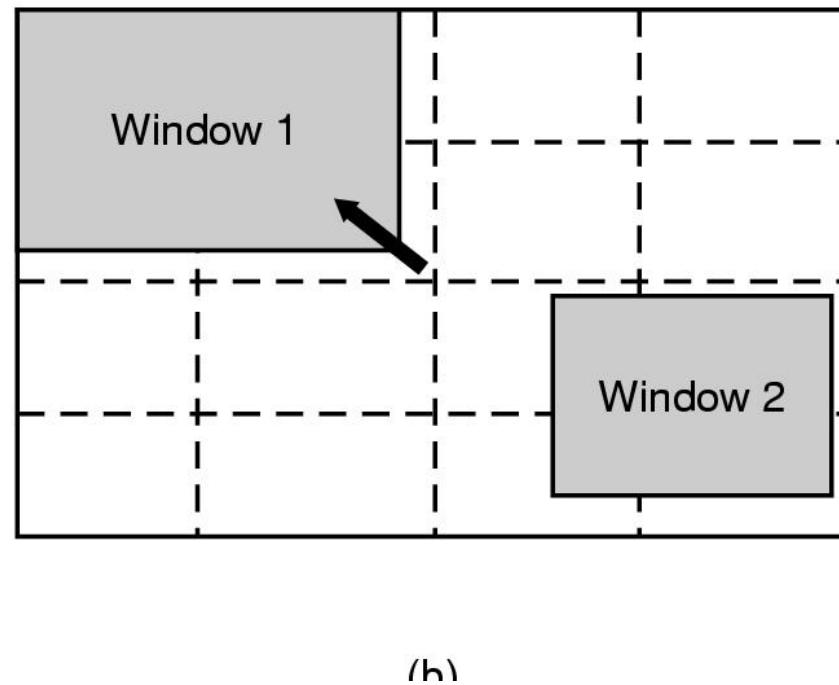
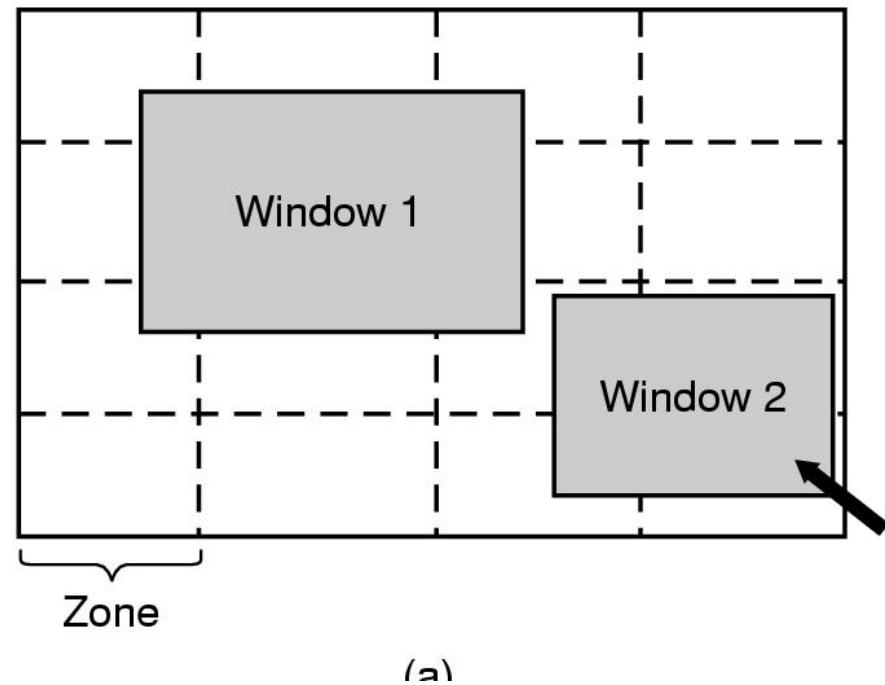
| Device    | Li et al. (1994) | Lorch and Smith (1998) |
|-----------|------------------|------------------------|
| Display   | 68%              | 39%                    |
| CPU       | 12%              | 18%                    |
| Hard disk | 20%              | 12%                    |
| Modem     |                  | 6%                     |
| Sound     |                  | 2%                     |
| Memory    | 0.5%             | 1%                     |
| Other     |                  | 22%                    |

- Li, K., Kumpf, R., Horton, P., and Anderson, T. A quantitative analysis of disk drive power management in portable computers. Proceedings of the 1994 Winter USENIX Conference, 279–291, January 1994.
- J. R. Lorch and A. J. Smith, "Software strategies for portable computer energy management," in *IEEE Personal Communications*, vol. 5, no. 3, pp. 60-73, Jun 1998.

# 电源管理

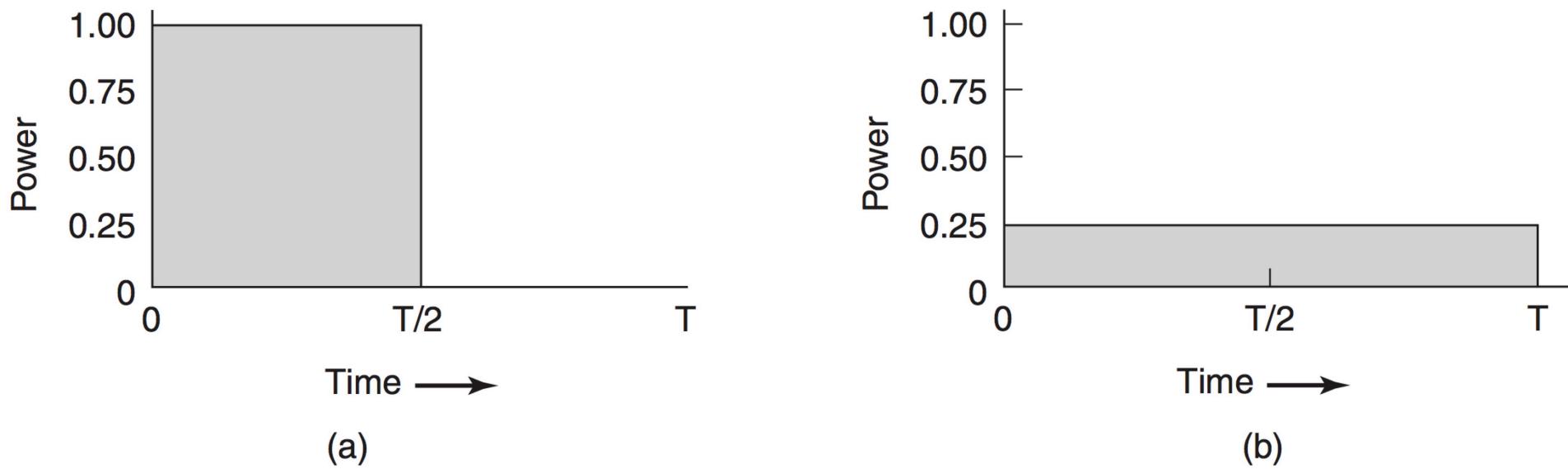
- 显示器
  - 用户不需要时关闭显示器
    - 让用户设置关闭的阈值
    - 避免应用频繁唤醒屏幕
  - 分区显示策略
- 硬盘
  - 预测磁盘使用，停止磁盘工作
  - 增加内存的缓存
- CPU
  - 休眠
  - 降频
- 无线通讯-空闲则关闭

# 分区显示 – 显示器的性能改进策略



分区显示，动态移动window1占用较少分区

# CPU – 执行时间和能耗的tradeoff



**Figure 5-42.** (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four.

电能和电压平方正比，降低电压  
1/2，能耗降低到1/4

# 移动应用下载能耗问题

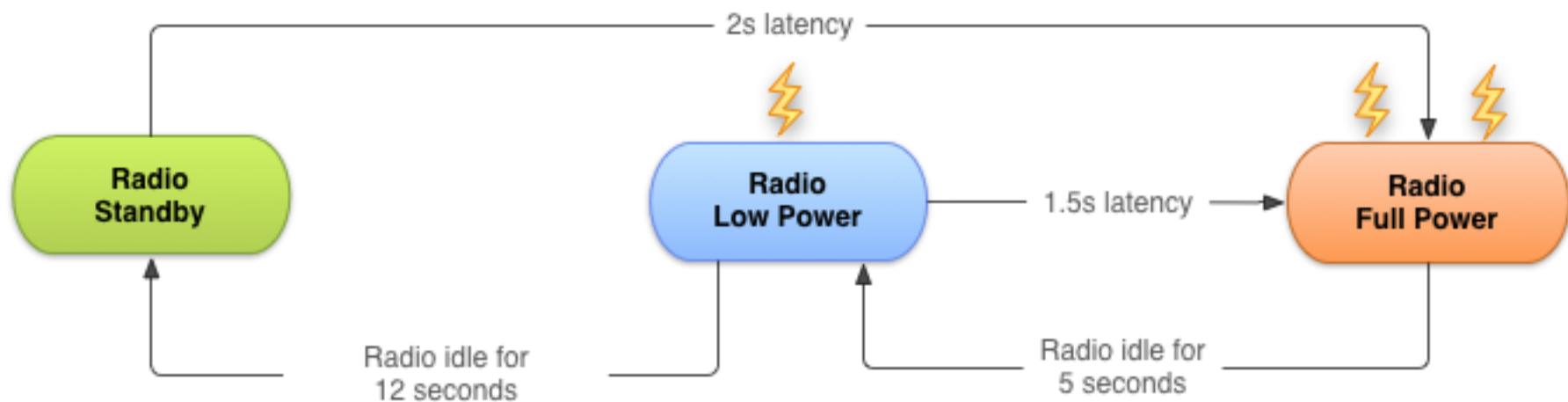
- 如何优化下载的能耗?
  - 无线传输是最耗费电量的操作之一
  - 采用预取、捆绑传输
- 无线状态机
  - 传输状态的无线设备非常耗电
  - 需要在不同能耗状态下转换
  - 降低能耗缩小启动延迟

# Android 下载能耗问题

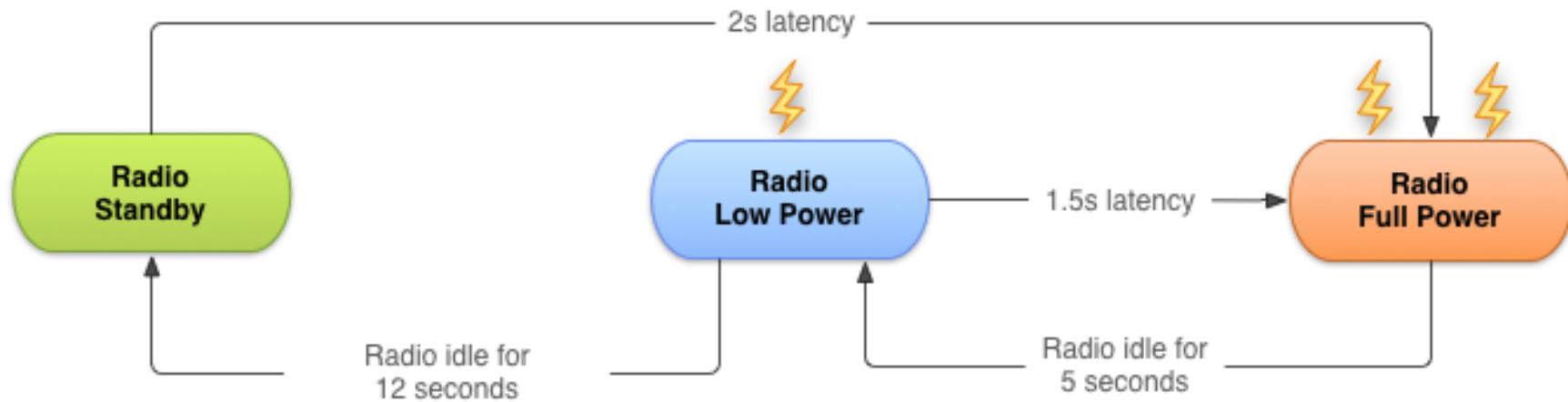
- 3G无线传输的能耗模型
  - 全能耗：全速传输状态，最耗电
  - 低能耗：中间状态，50%电耗费
  - standby：最小能量状态，无网络活动
  - 低能耗和待命状态降低能耗，但切换引入传输延迟
    - 低→全<sup>~1.5s</sup>，待命→全<sup>~2s</sup>

# Android 下载能耗问题

- 3G无线传输的能耗模型
  - 适当延迟转换
  - AT&T 3G无线状态机：适合浏览器应用

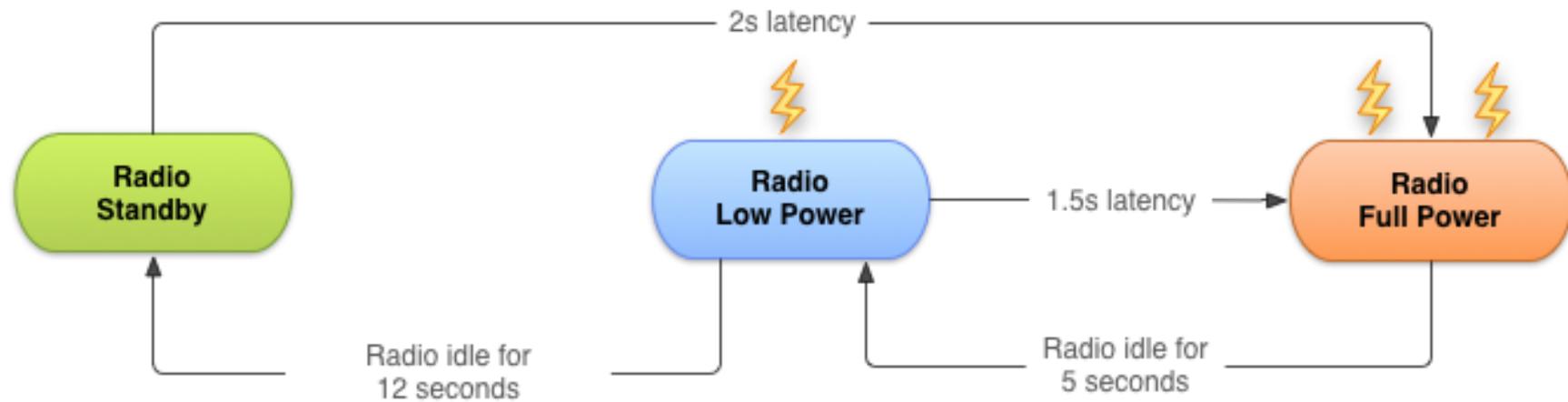


# Android 下载能耗问题



- 假设一个应用每18秒传输1秒的数据
  - 永远不进入待命，6秒全速12秒低速2秒切换
  - 1分钟：18秒全速，36秒低速
- 假设一个应用每分钟传输3秒的数据
  - 1分钟：8秒全速，12秒低速，40秒待命

# Android 下载能耗问题



- 通过预取和绑定传输进行集中传输
- 避免频繁唤醒无线设备传送小数据

# Android 下载能耗问题

- 数据预取：

- 事前一次全速取出数据
- 减少激活无线连接次数、缩短延迟，节省电量、缩短下载时间，提升用户体验
- 问题：判断不准可能增加下载量，耗费电量
  - 程序启动所需数据可以提前预取
- 原则：每2–5分钟取1–5MB数据
  - 视频应用，每2–5分钟取一部分视频流

# Android 下载能耗问题例子

- 音乐播放器-有限预取
  - 如果下载整个专辑， 用户第一首后暂停， 浪费了带宽和能耗
  - 缓冲下一首和当前播放的一首歌
  - 使用HTTP live streaming, 批量传输音频
- 新闻阅读器-按需下载
  - 选中类别才下载标题， 选中文章才下载内容，
    - » 切换类别， 查看标题和阅读文章会保持无线一直活跃
    - » 频繁切换还会增加延迟
  - 启动时下载一部分头条和缩略图， 启动后批量下载主标题列表中的缩略图和文本
    - » 缩短启动时间、批量预取
  - 后台完整下载-可能浪费流量， WiFi打开充电时使用

# Android 无线能耗

- 不同的无线连接类型耗电不同
  - 能耗:  $4G > LTE > 3G > 2G > WiFi$ 
    - WiFi高带宽低能耗, 尽量使用!
    - 无线频谱: 带宽越高, 能耗越高
    - 越高带宽, 每次预取数据和下载时间应越长
    - E. g., LTE 2倍3G带宽和能耗 $\rightarrow$ 每次4倍数据