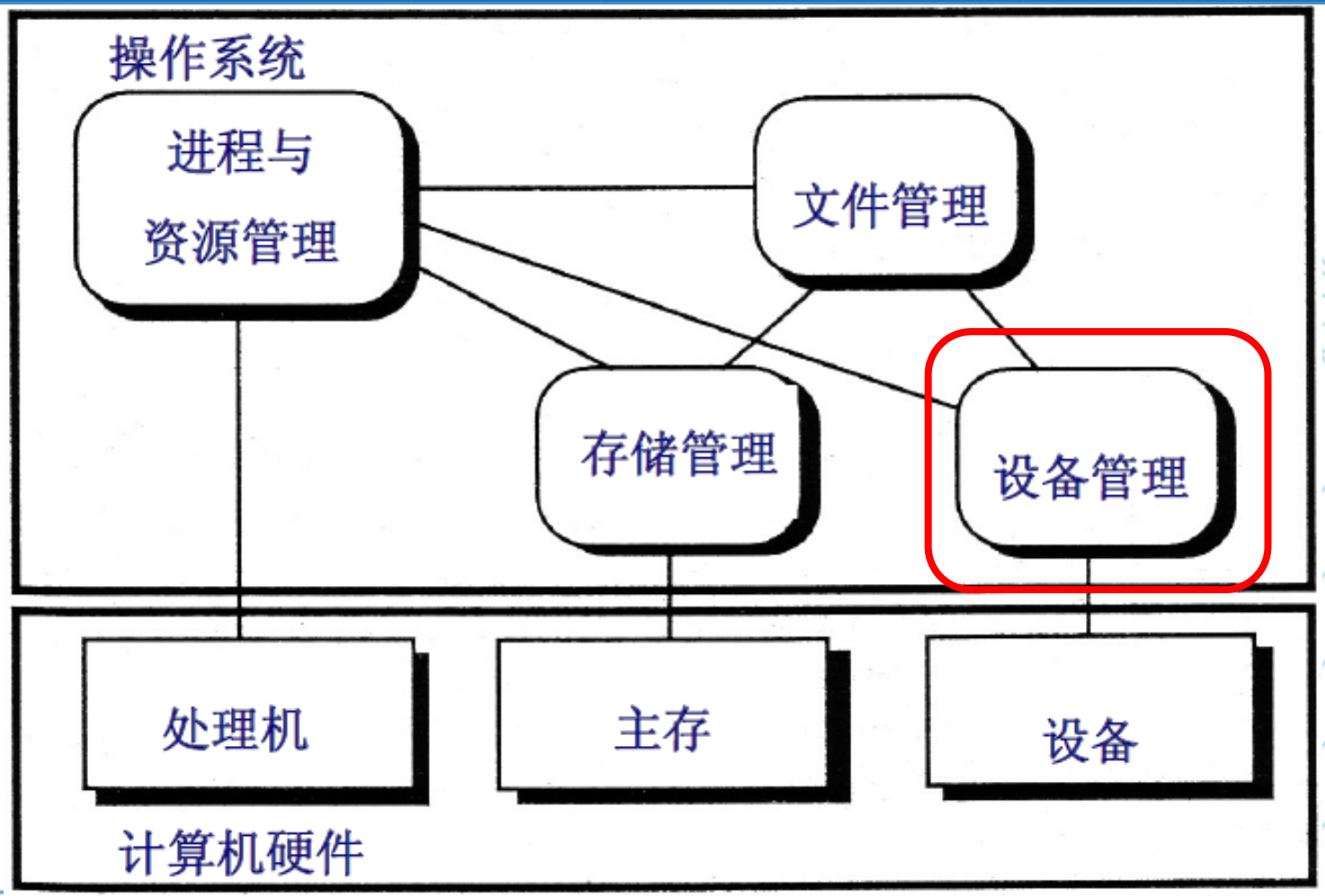
操作系统　Operating System

第五章 输入输出系统

沃天宇
woty@buaa.edu.cn
2024年5月8日

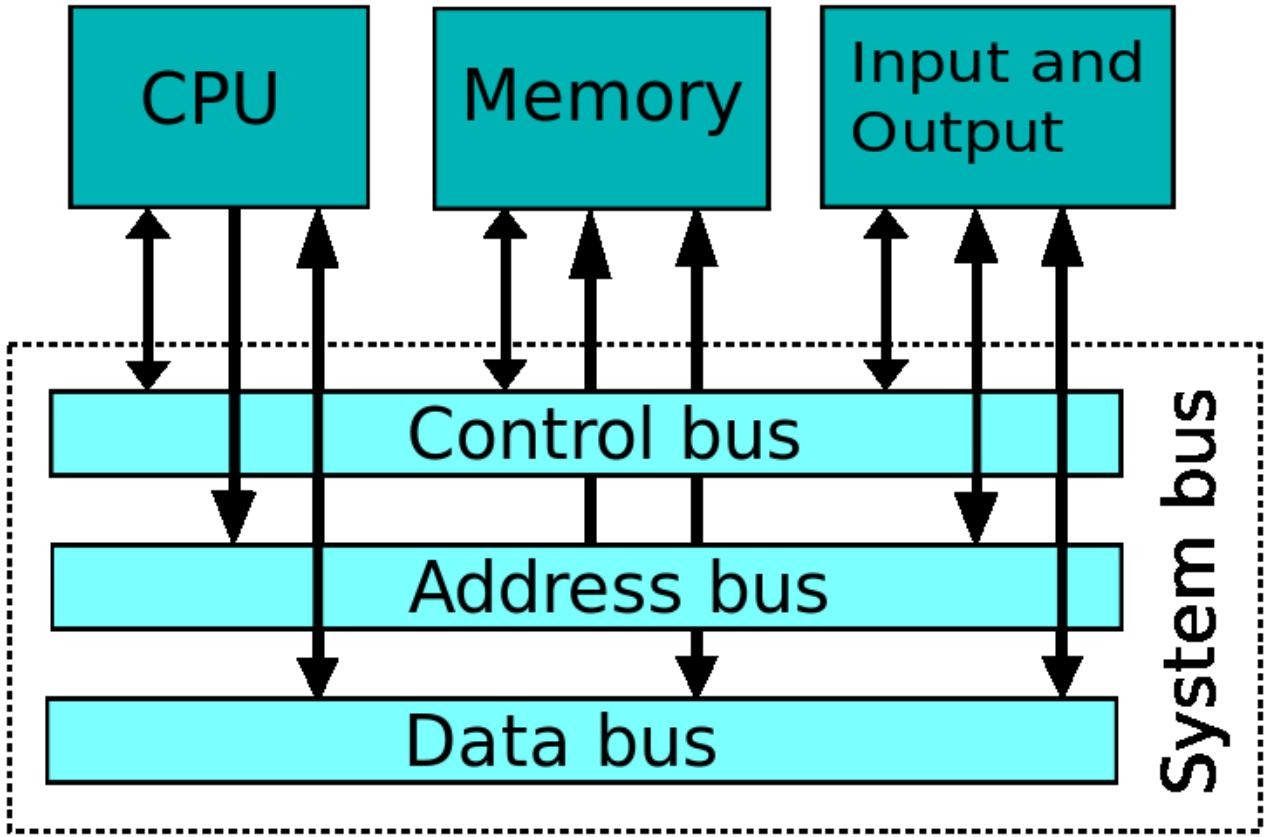
操作系统与设备管理

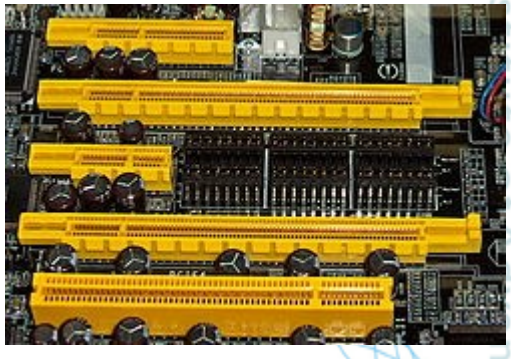


内容提要

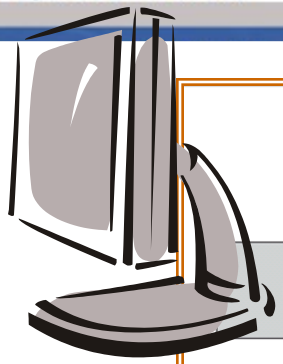
- I/O硬件基本原理
- I/O软件基本原理
- OS设备管理实例

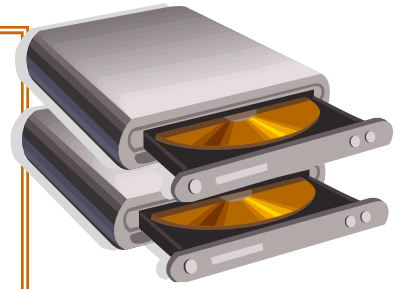
从总线(Bus)说起

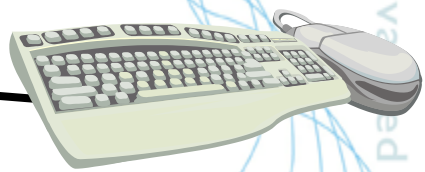


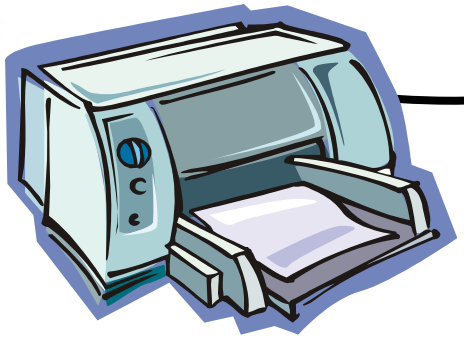


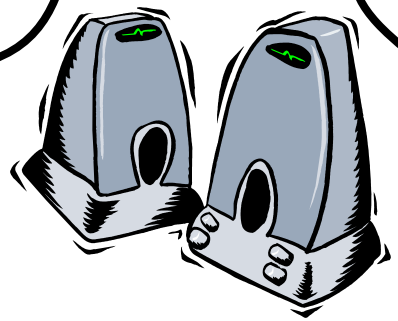
总线带宽 = 频率 x 宽度（Bytes/sec）。







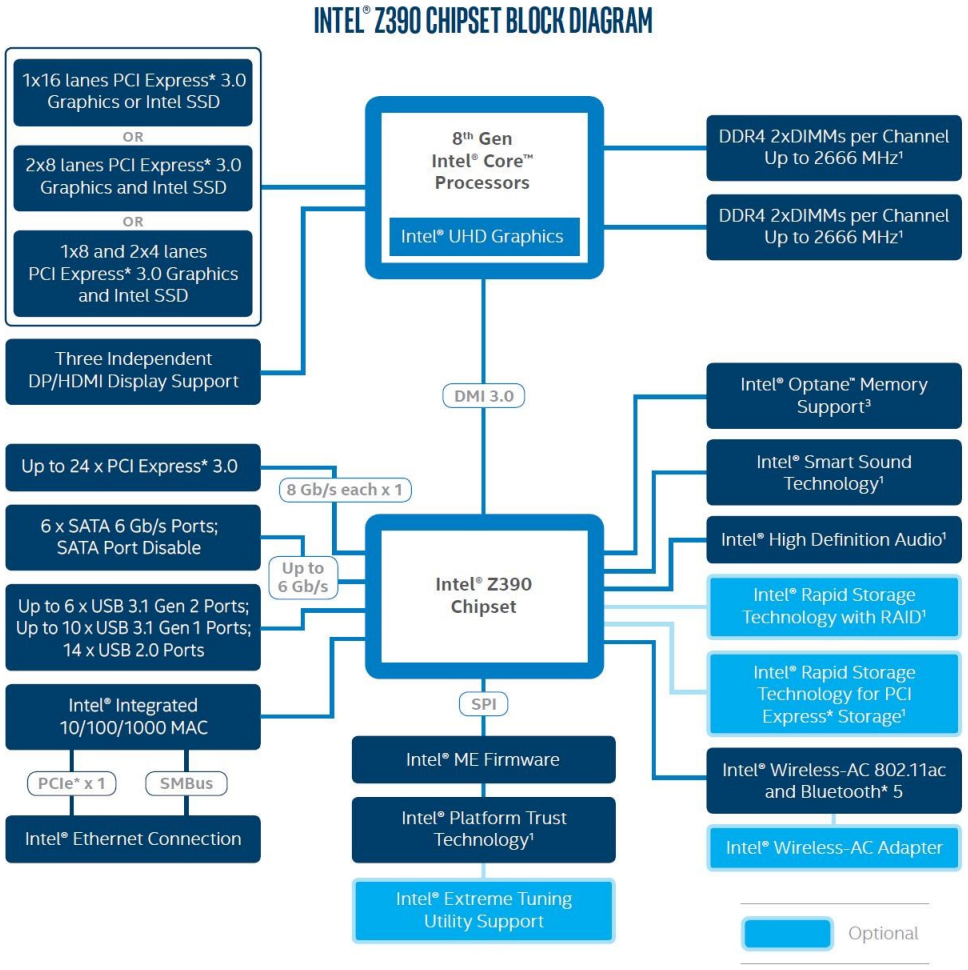




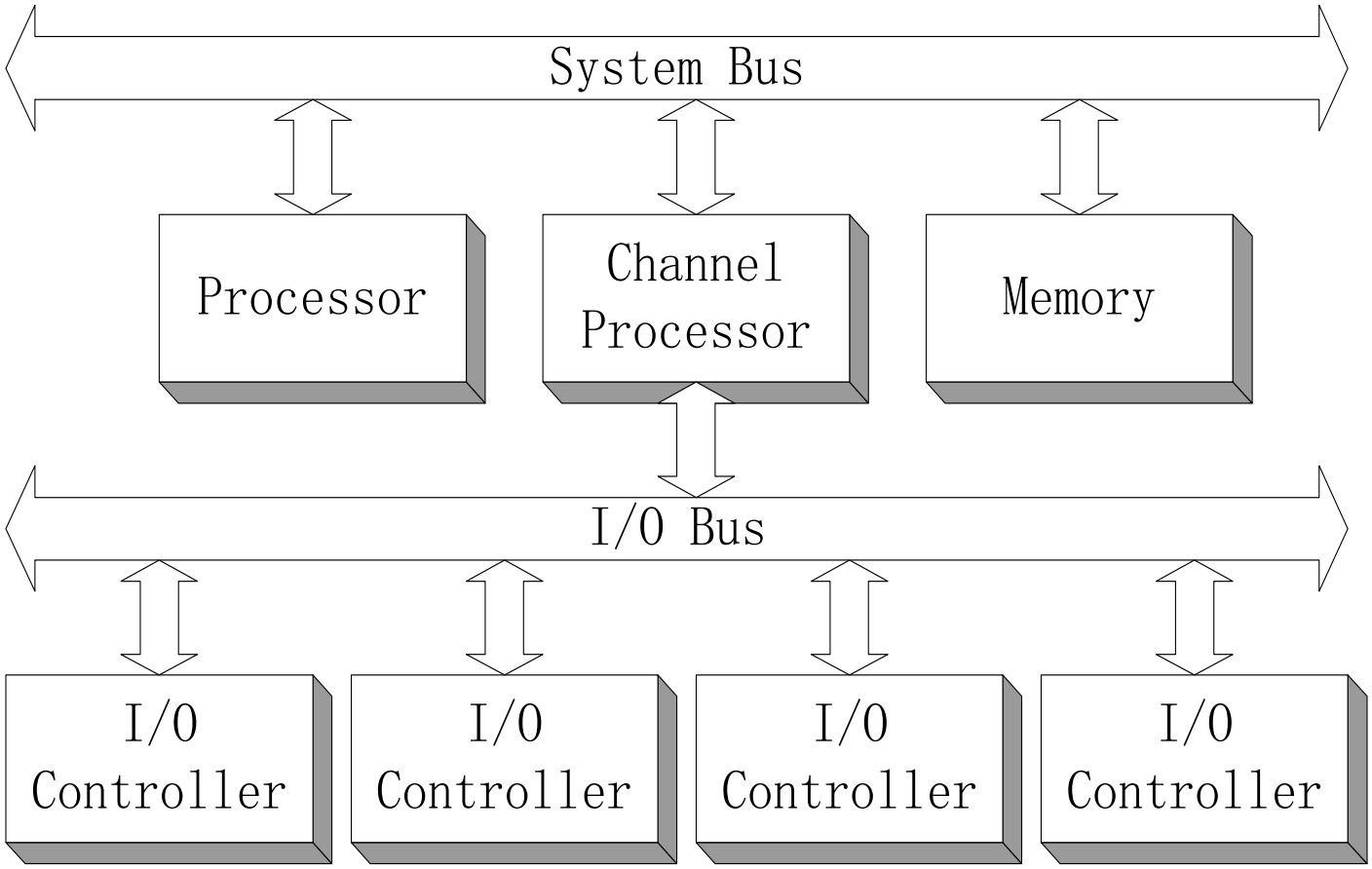
PC系统结构与PCI总线

芯片组

- CPU
  - Handles memory
  - Graphics
- 芯片组
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers



I/O系统结构示意



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

设备传输速度可以相差12个数量级！

I/O设备分类

- 类型
  - 传输速度：低速、中速、高速
  - 信息交换单位：块设备和字符设备
  - 共享属性：独占设备、共享设备、虚拟设备
- 设备与控制器之间的接口
  - 数据信号
  - 控制信号
  - 状态信号

设备的管理的目标和功能

- 设备管理目标
  - 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
  - 方便使用：方便用户使用，对不同类型的设备统一使用方法，协调对设备的并发使用
  - 方便控制：方便OS内部对设备的控制。例如：增加和删除设备，适应新的设备类型
- 外设管理功能
  - 提供设备使用的用户接口：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。

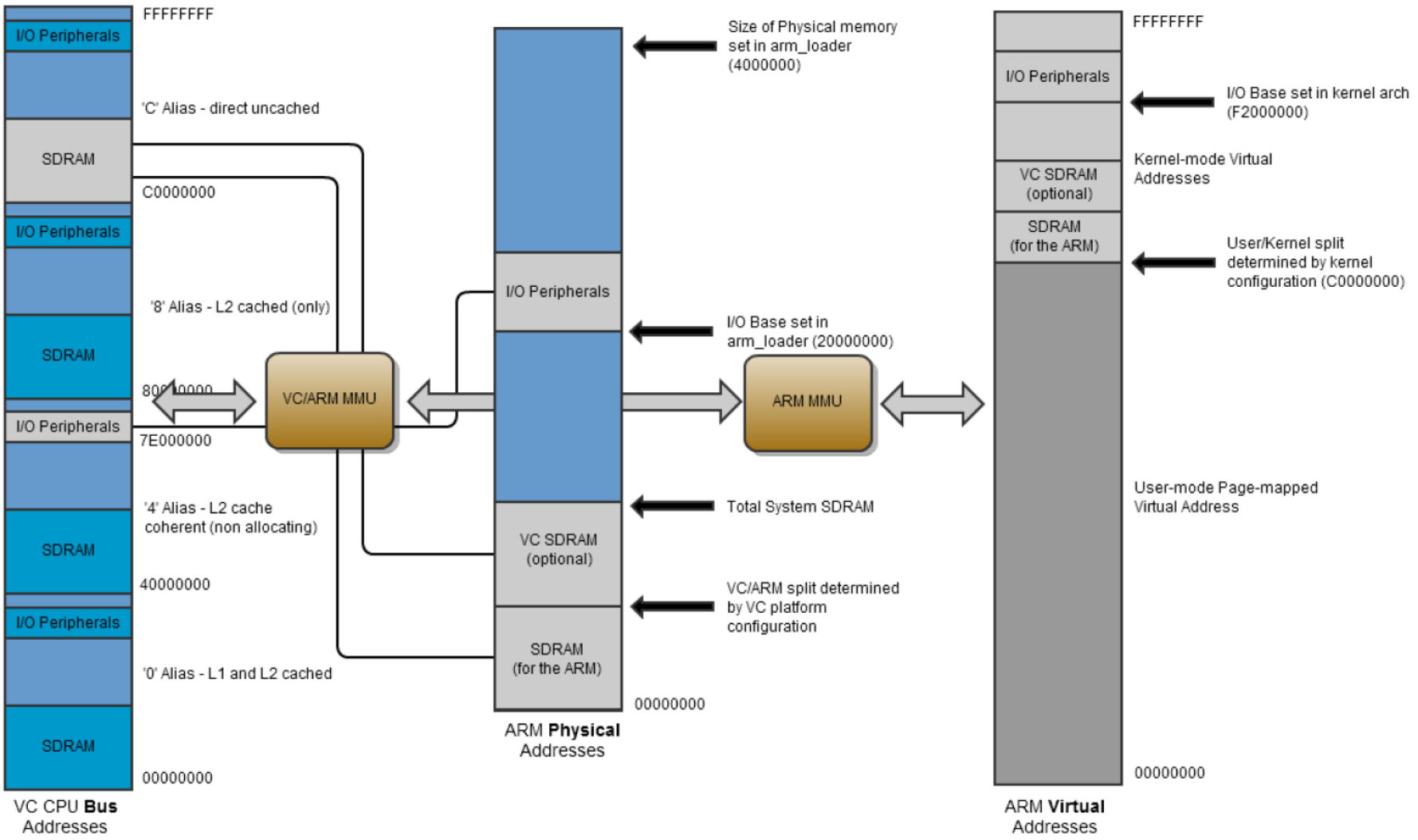
设备控制器

- 控制设备的功能
  - 接收和识别CPU命令
  - 数据交换：CPU与控制器、控制器与设备
  - 设备状态的了解和报告
  - 设备地址识别
  - 缓冲区
  - 对设备传来的数据进行差错检测
- 组成
  - 控制器与处理机接口：**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
  - 控制器与设备接口
  - I/O逻辑：用于实现CPU对I/O设备的控制

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)

IO地址映射（RISC）


FFFFFFFF
I/O Peripherals
'C' Alias - direct uncached
SDRAM
C0000000
I/O Peripherals
'8' Alias - L2 cached (only)
SDRAM
80000000
I/O Peripherals
7E000000
'4' Alias - L2 cache coherent (non allocating)
SDRAM
40000000
I/O Peripherals
'0' Alias - L1 and L2 cached
SDRAM
00000000
VC CPU Bus Addresses
VC/ARM MMU
I/O Peripherals
VC SDRAM (optional)
SDRAM (for the ARM)
00000000
ARM Physical Addresses
Size of Physical memory set in arm_loader (4000000)
I/O Base set in arm_loader (20000000)
Total System SDRAM
VC/ARM split determined by VC platform configuration
ARM MMU
FFFFFFFF
I/O Peripherals
I/O Base set in kernel arch (F2000000)
Kernel-mode Virtual Addresses
VC SDRAM (optional)
SDRAM (for the ARM)
User/Kernel split determined by kernel configuration (C0000000)
User-mode Page-mapped Virtual Address
00000000
ARM Virtual Addresses

内容提要

- I/O硬件基本原理
- I/O软件基本原理
- OS设备管理实例

The Goal of the I/O Subsystem

- Provide Uniform Interfaces, Despite Wide Range of Different Devices
  - This code works on many different devices:

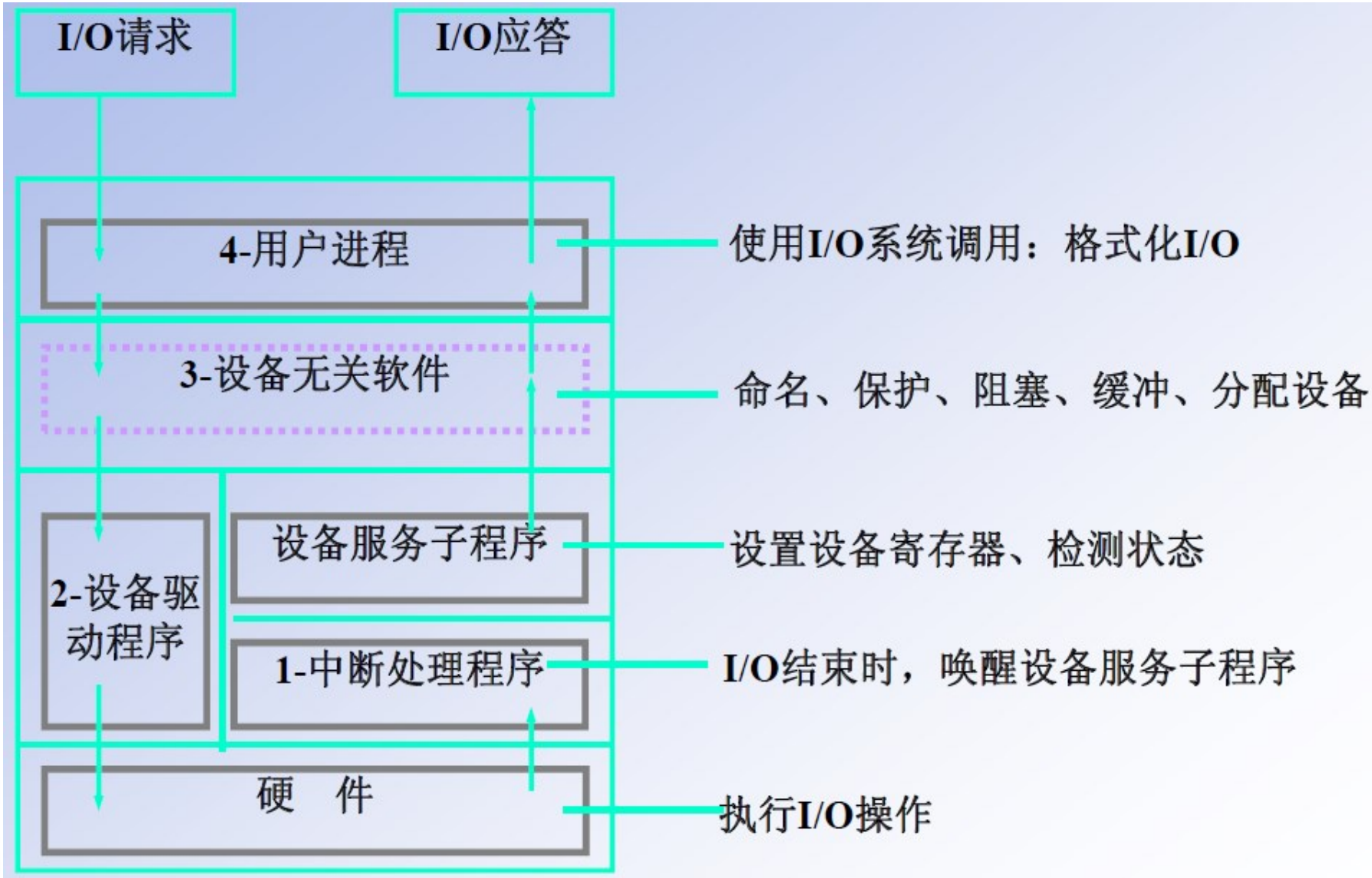
```
FILE fd = fopen("/dev/something","rw");
for (int i = 0; i < 10; i++) {
  fprintf(fd,"Count %d\n",i);
}
close(fd);
```

  - Why? Because code that controls devices ("device driver") implements standard interface.

Want Standard Interfaces to Devices

- Block Devices: *e.g.* disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Commands include `open()`, `read()`, `write()`, `seek()`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character Devices: *e.g.* keyboards, mice, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing
- Network Devices: *e.g.* Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface
  - Unix and Windows include socket interface
    - » Separates network protocol from network operation
    - » Includes `select()` functionality
  - Usage: pipes, FIFOs, streams, queues, mailboxes

I/O相关软件的层次关系



I/O请求的处理过程

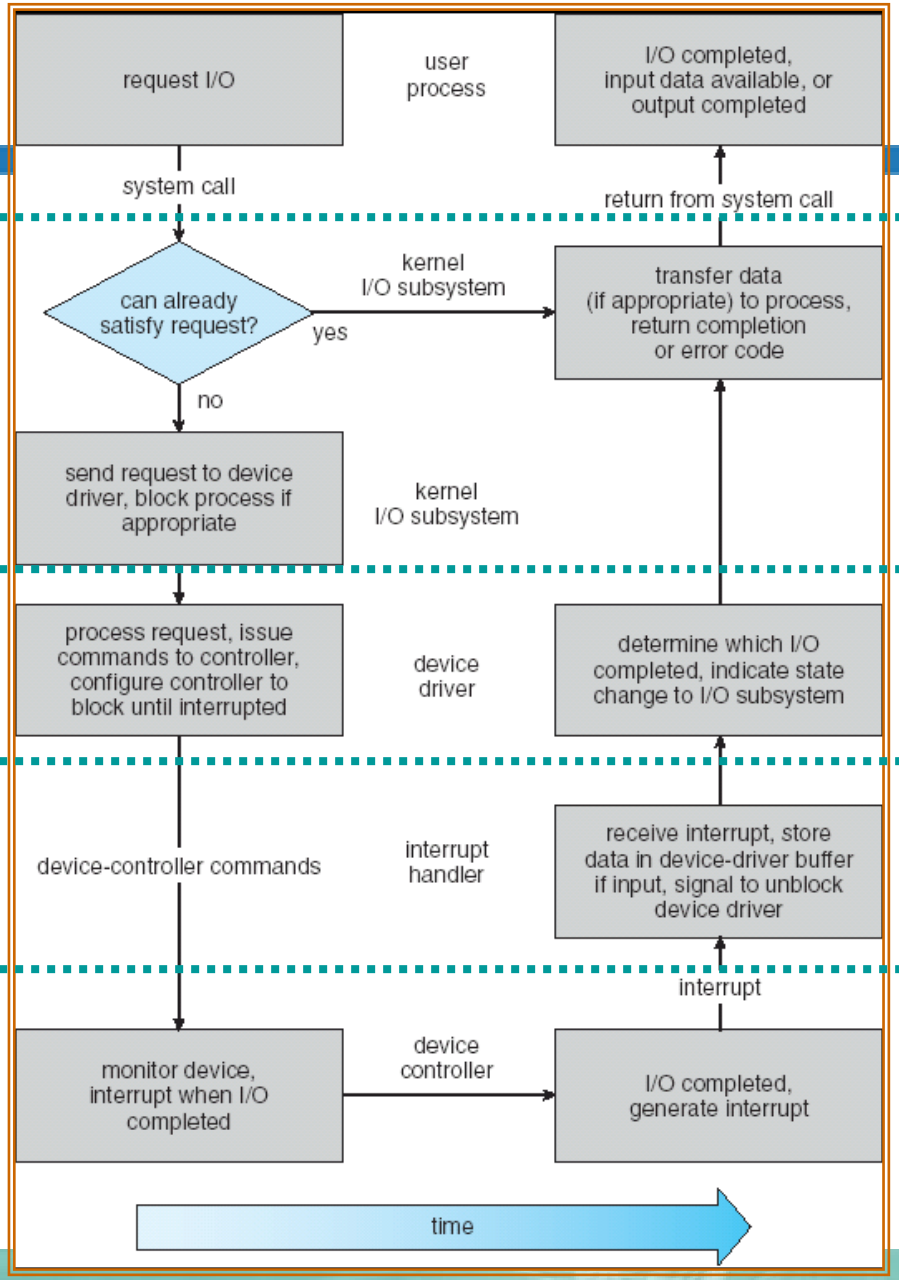
用户程序

内核I/O子系统

设备驱动程序上层部分

设备驱动程序下层部分

设备硬件



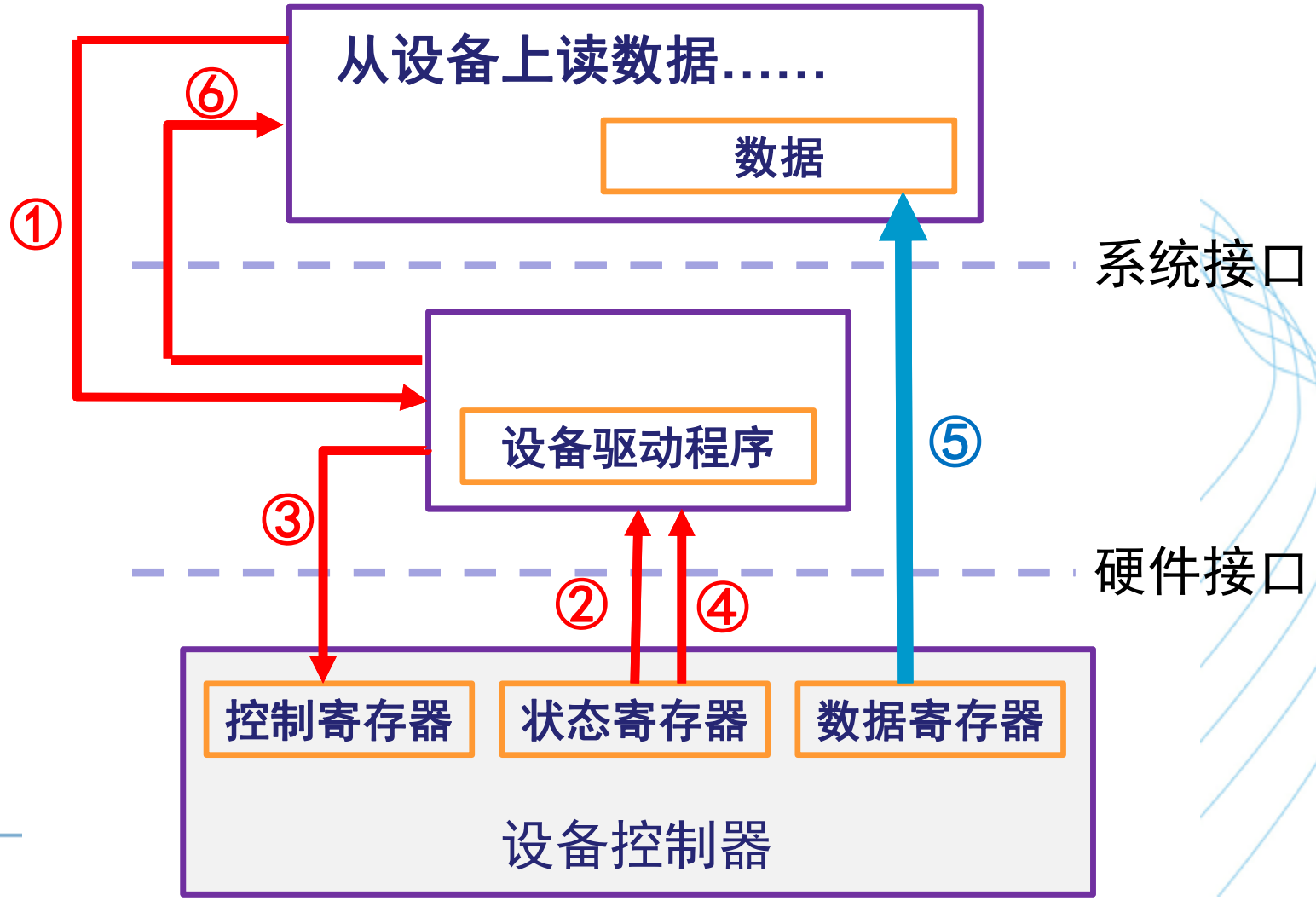
I/O控制技术

- 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）

I/O控制方式

- 程序控制I/O，也称轮询或查询方式I/O，它由CPU代表进程向I/O模块发出指令， 然后进入忙等状态， 直到操作完成之后进程才能够继续执行。
- 中断驱动， 当I/O操作结束后由设备控制器 主动地来通知设备驱动程序说这次结束， 而不是设备驱动程序不断地去轮询看看设备的状态。
- DMA，直接存储器访问方式， 是由一个专门的控制器来完成数据从内存 到设备或者是从设备到内存的传输工作。
- 通道与DMA的原理几乎是一样的，通道是一个特殊功能的处理器，它有自己的指令和程序专门负责数据输入输出的传输控制。CPU将“传输控制”的功能下放给通道后只负责“数据处理”功能。这样，通道与CPU分时使用内存，实现了CPU内部运算与I/O设备的并行工作。

轮询方式



轮询方式的工作过程

① 应用程序提出了一个读数据的请求；

② 设备驱动程序检查设备的状态；

③ 如果状态正常，就给设备发出相应的控制命令；

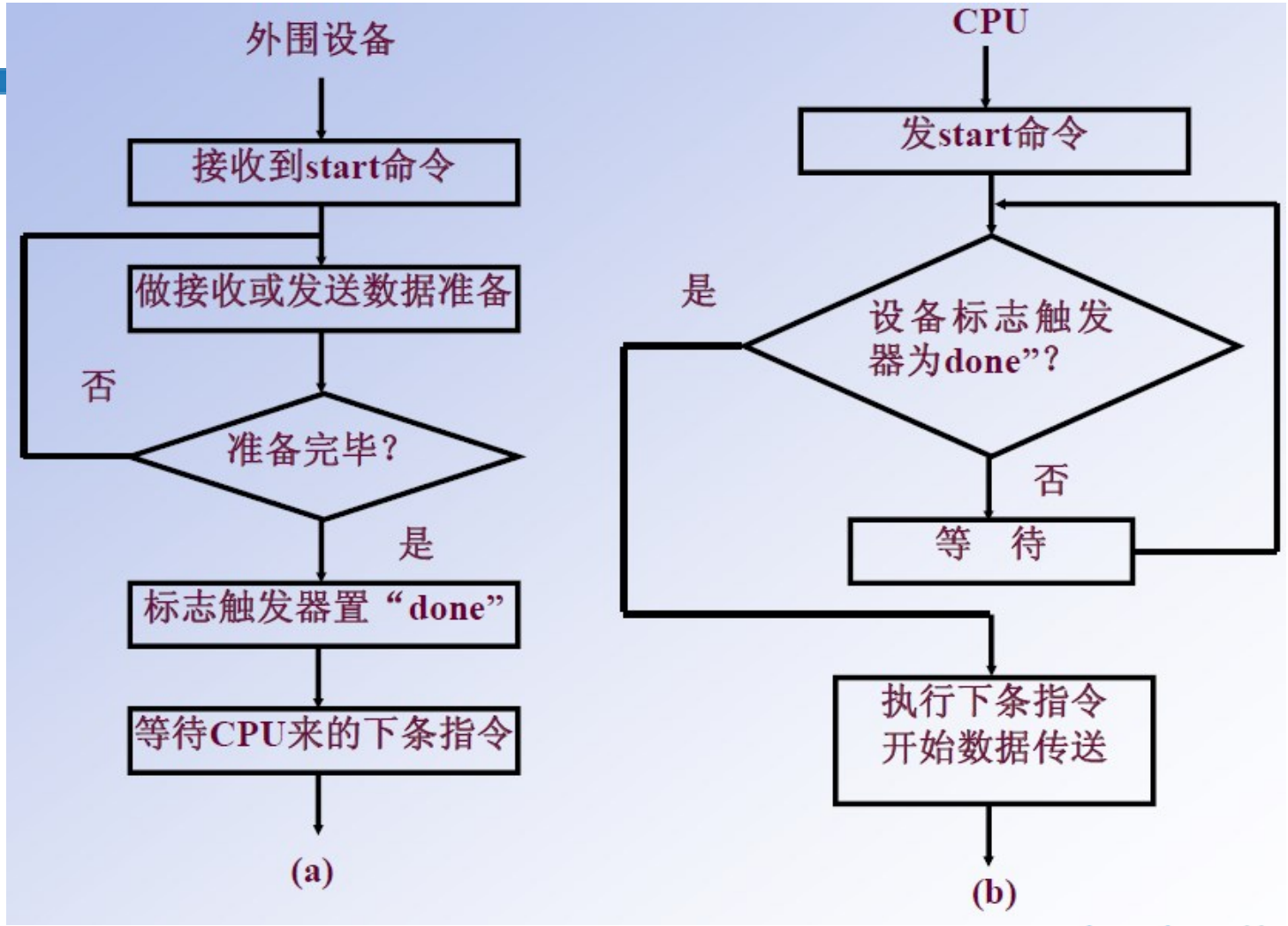
④ 不断地去测试这个设备是否完成了这次执行过程，实际上就是一个轮询；

⑤ 设备控制器完成操作，把数据送给应用程序

⑥ 应用程序继续进行相应的处理。

程序控制I/O(Programmed I/O)

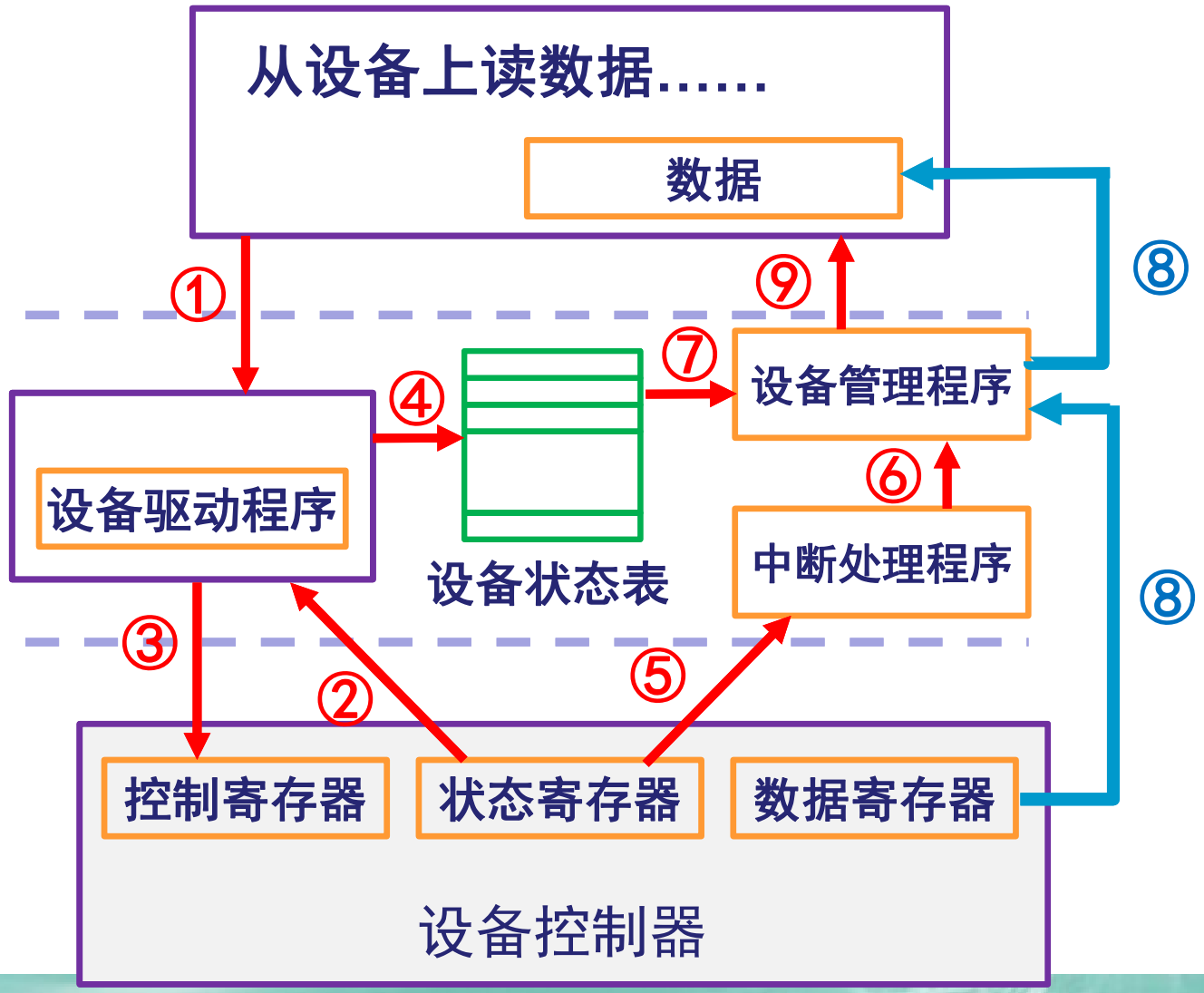
I/O操作由程序发起，并等待操作完成。数据的每次读写通过CPU。



优点：实现简单

缺点：CPU利用率相当低，在外设进行数据处理时，CPU只能等待，循环等待中浪费了大量时间。

中断驱动方式

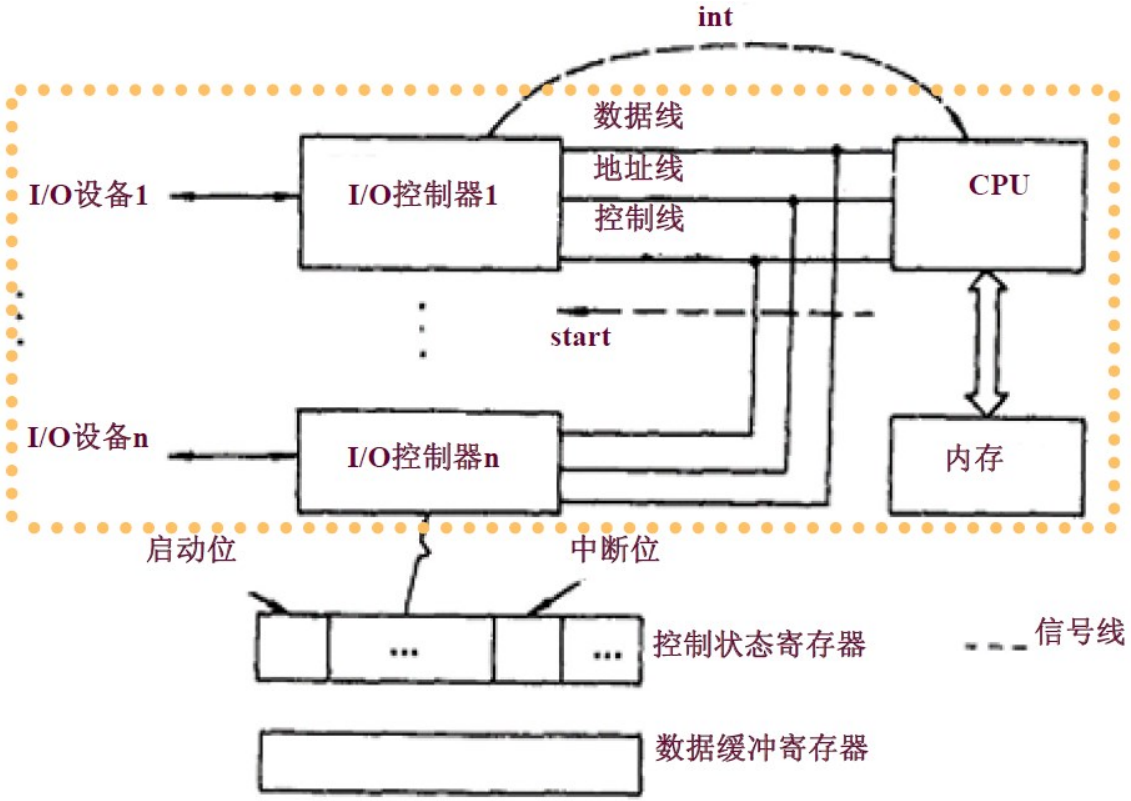


中断驱动方式的工作过程

① 用户程序提出I/O请求；
② 设备驱动程序检查设备的状态；
③ 如果设备已经准备好，那么就向设备发出控制命令；
④ 将状态记录在设备状态表中，CPU继续其它工作。
⑤ 设备完成工作后向CPU发中断信号，转入中断处理程序；
⑥ 中断处理程序发现这是一个正常地完成了控制命令的信号后，把结果提交给设备管理程序；
⑦ 设备管理程序会从设备状态表里查询是哪一个请求的完成；
⑧ 把相应的数据送到应用程序；
⑨ 通知应用程序可以继续执行。

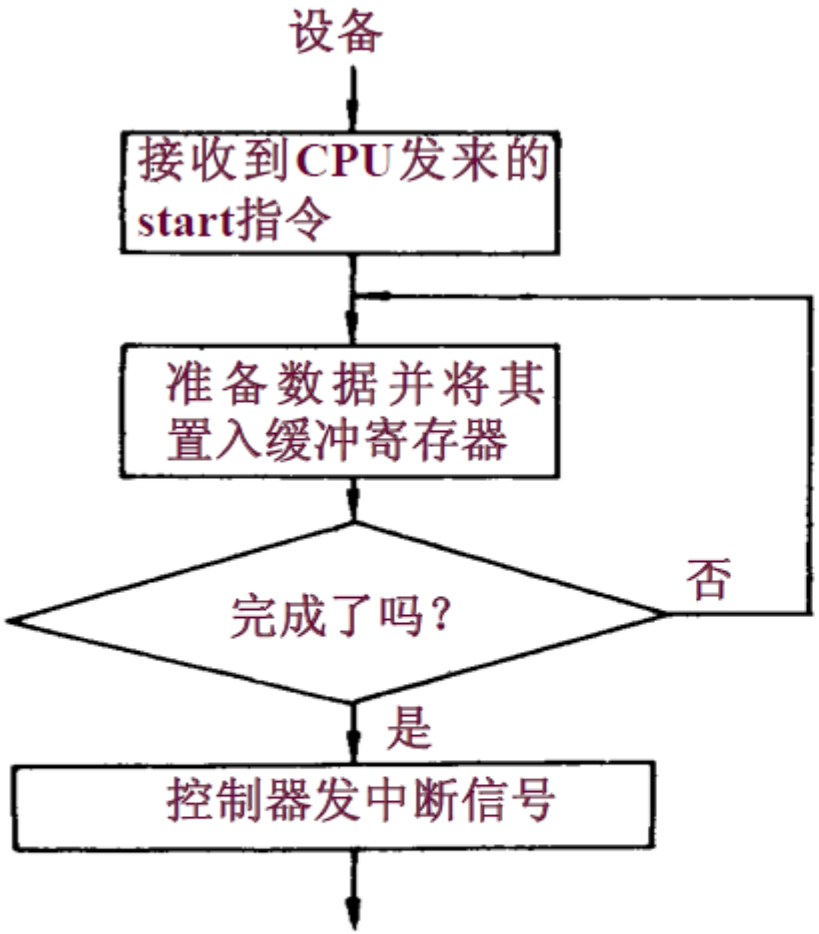
中断驱动方式

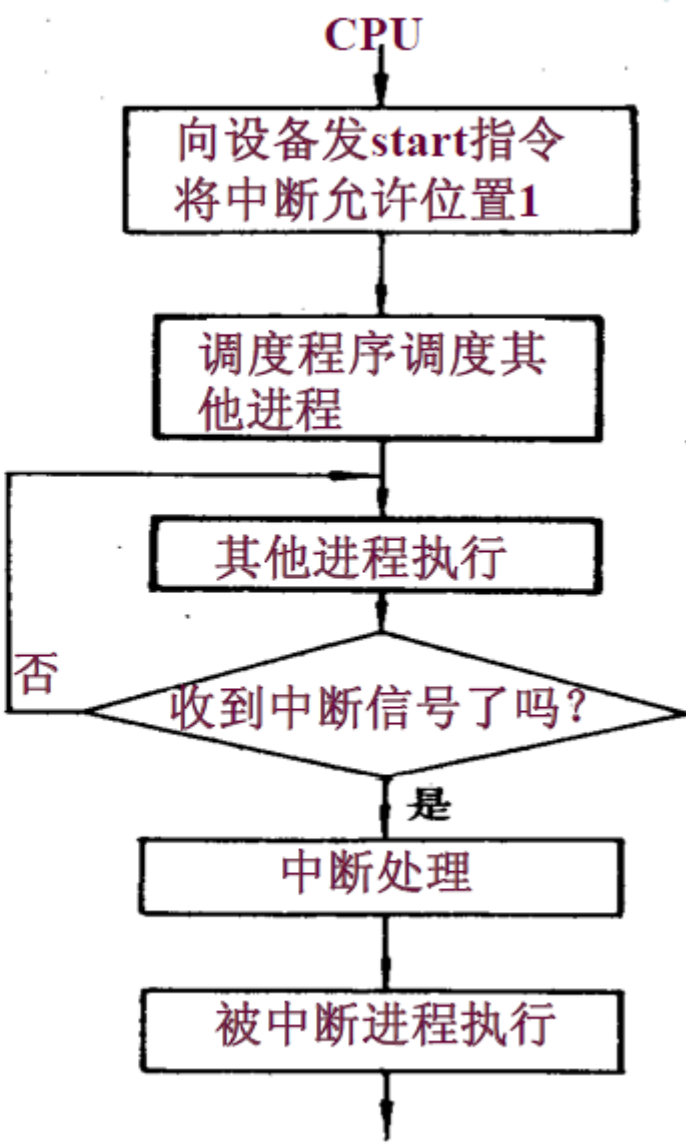
I/O操作由程序发起，在操作完成时（如数据可读或已经写入）由外设向CPU发出中断，通知该程序。数据的每次读写通过CPU。



- 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/输出一个数据都要中断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


设备
接收到CPU发来的start指令
准备数据并将其置入缓冲寄存器
完成了吗?
否
是
控制器发中断信号

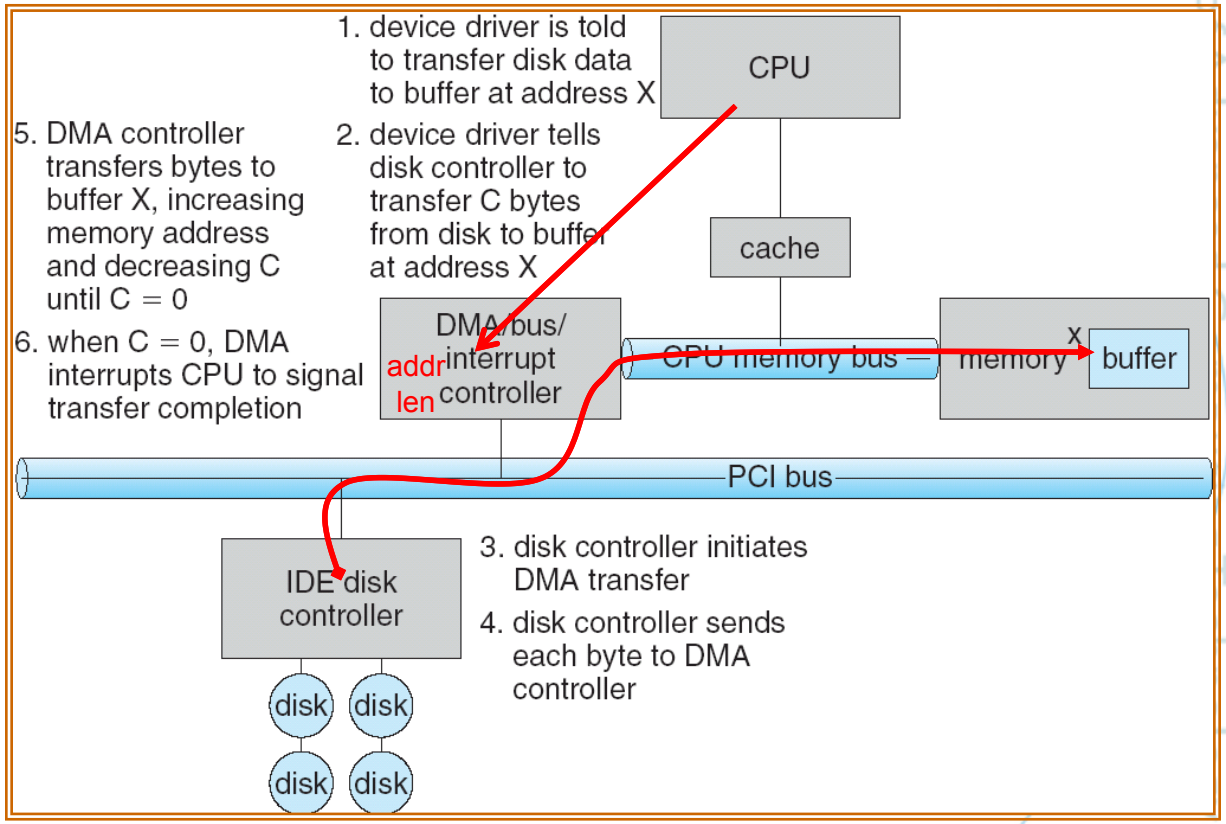

CPU
向设备发start指令
将中断允许位置1
调度程序调度其他进程
其他进程执行
否
收到中断信号了吗?
是
中断处理
被中断进程执行

Intel Pentium 处理器的中断向量表

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19Ð31	(Intel reserved, do not use)
32Ð255	maskable interrupts

直接存储访问方式 (DMA, Direct Memory Access)

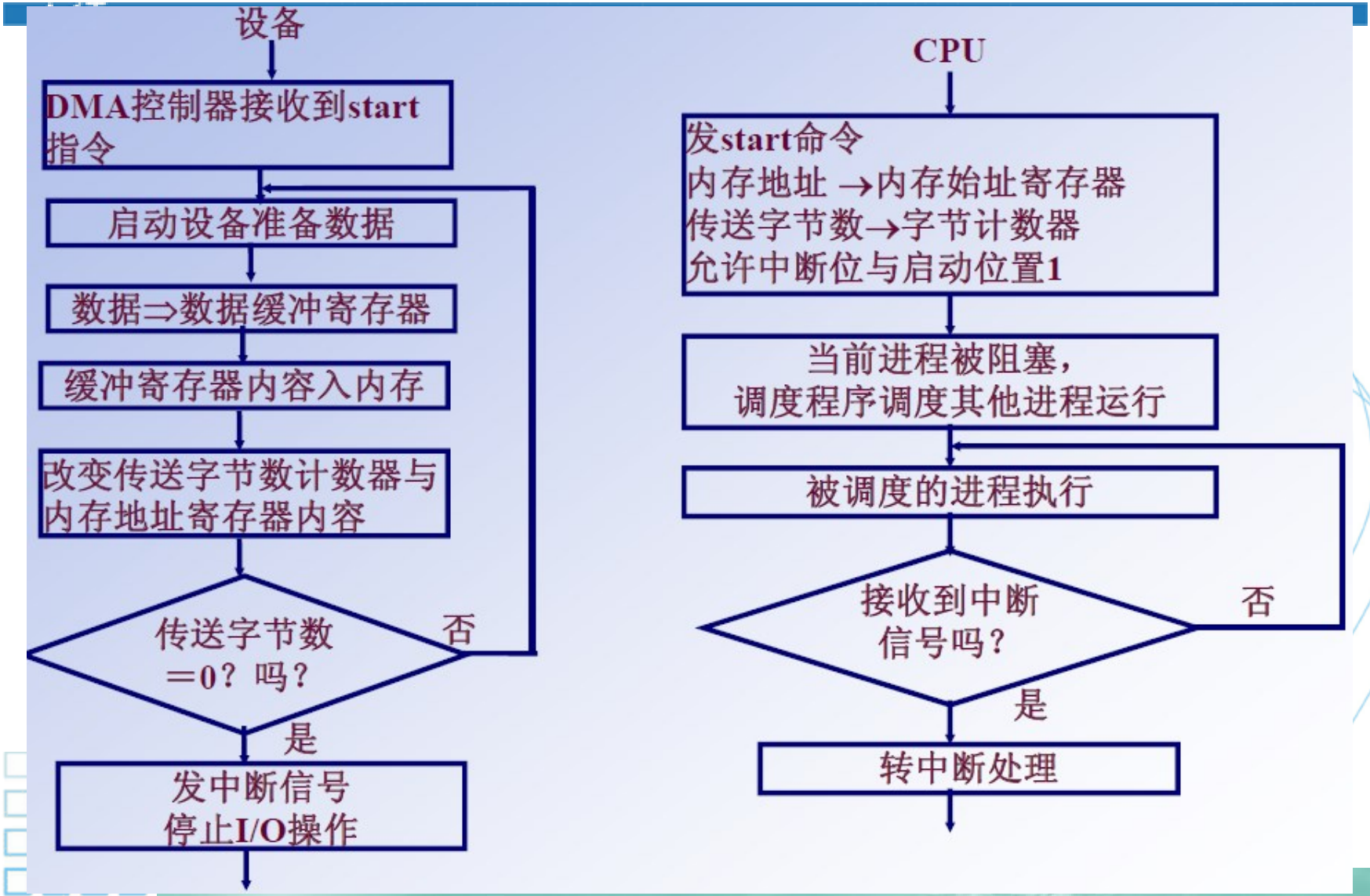
- 由程序设置DMA控制器中的若干寄存器值（如内存始址，传送字节数），然后发起I/O操作，而后者完成内存与外设的成批数据交换，在操作完成时由DMA控制器向CPU发出中断。



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，而其中的一批数据读写无需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。而且每个设备占用一个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



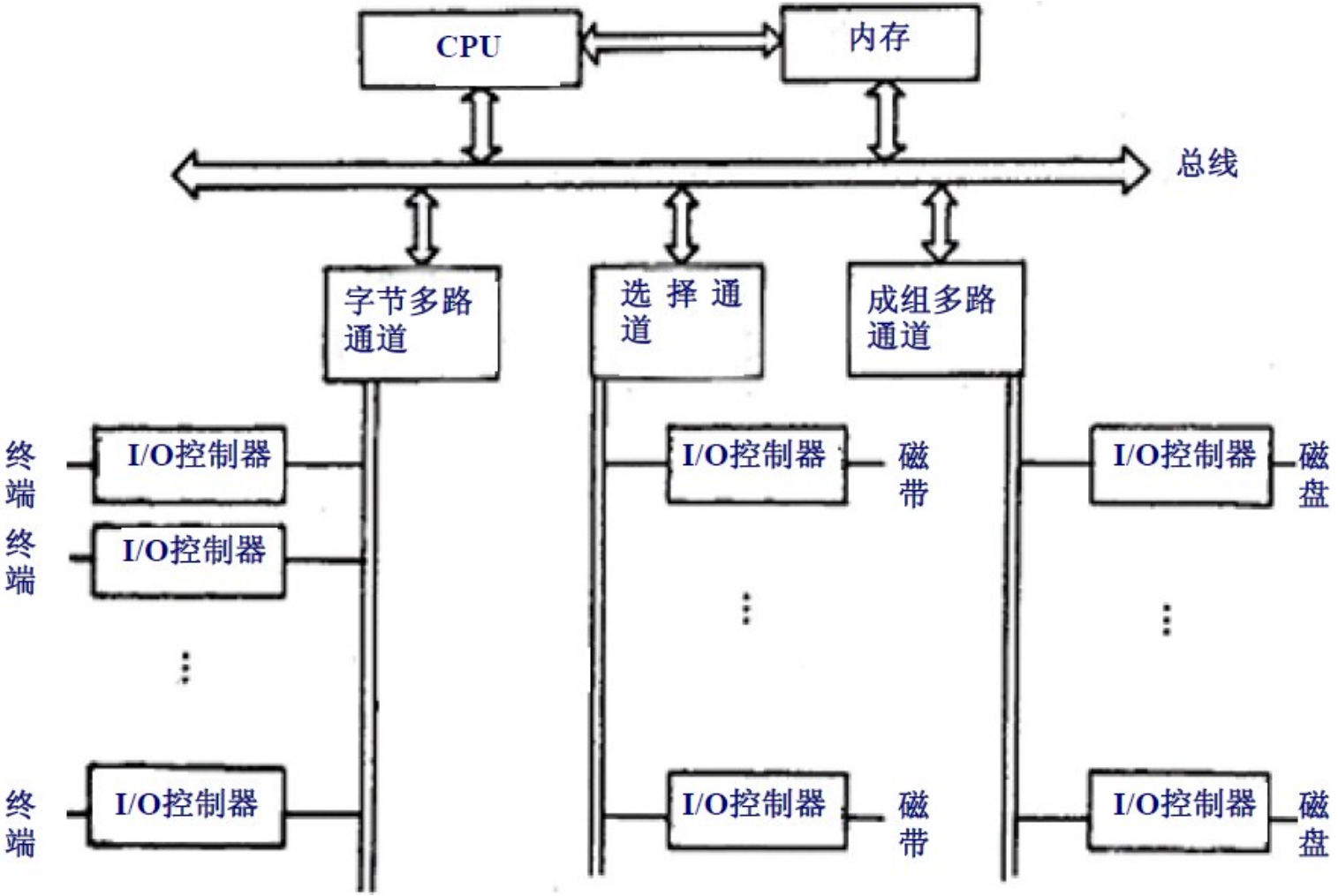
DMA与中断方式的区别

- 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，由于程序陷入内核，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。
- 程序中断方式具有对异常事件的处理能力，而DMA控制方式适用于数据块的传输。

I/O通道控制方式(channel control)

- DMA方式的发展，进一步减少CPU干预。把对一个数据块的读写干预，减少为对一组数据块读写的干预。
- 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、选择通道、成组多路通道

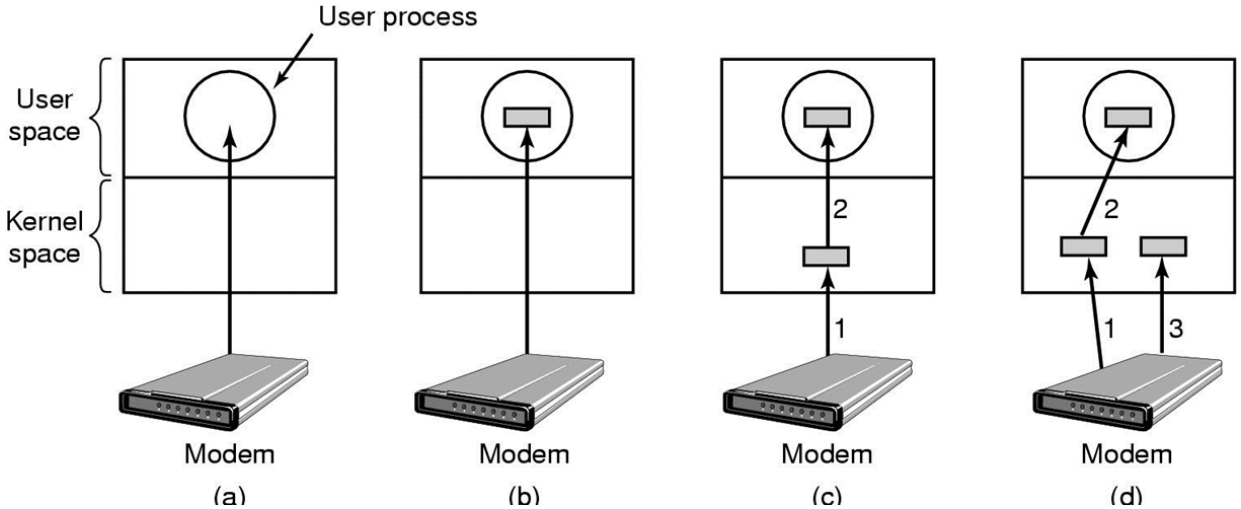
教材P214

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。

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。



无缓冲

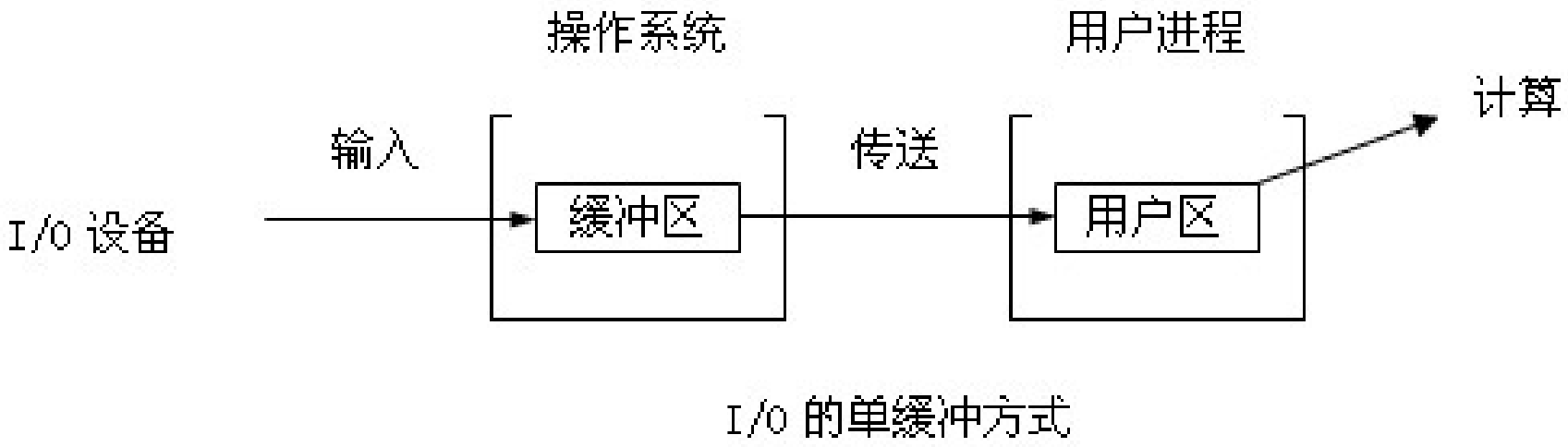
用户空间缓冲

内核空间缓冲拷贝到用户空间

内核空间双缓冲

单缓冲(single buffer)

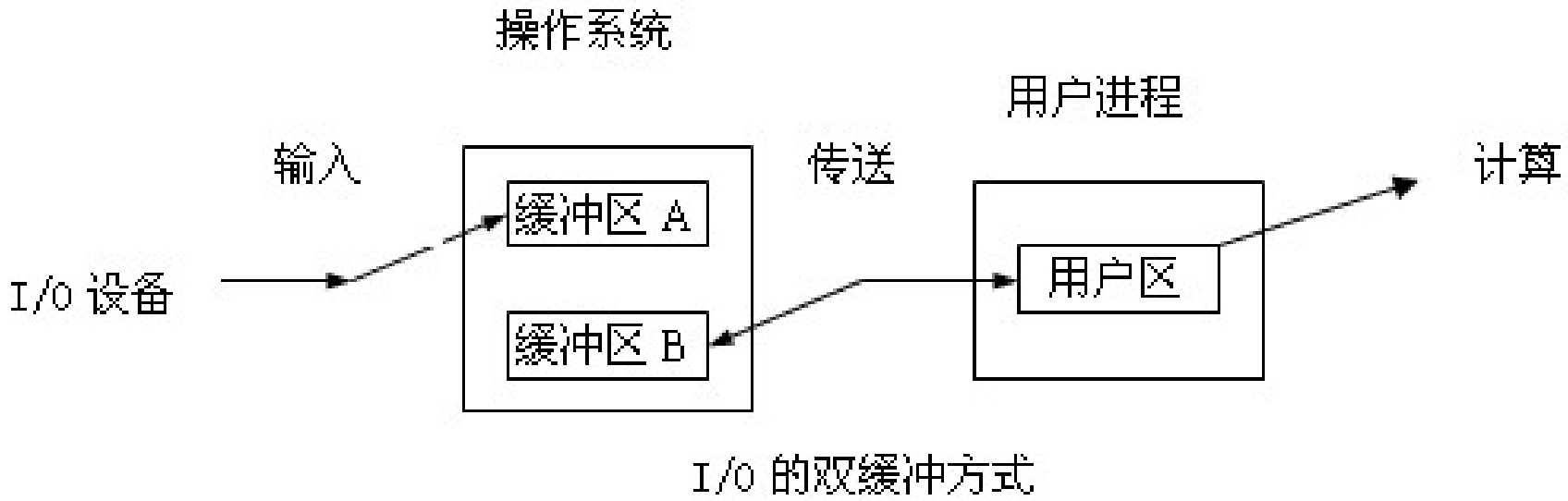
- 一个缓冲区，CPU和外设轮流使用，一方处理完之后接着等待对方处理。



I/O 的单缓冲方式

双缓冲(double buffer)

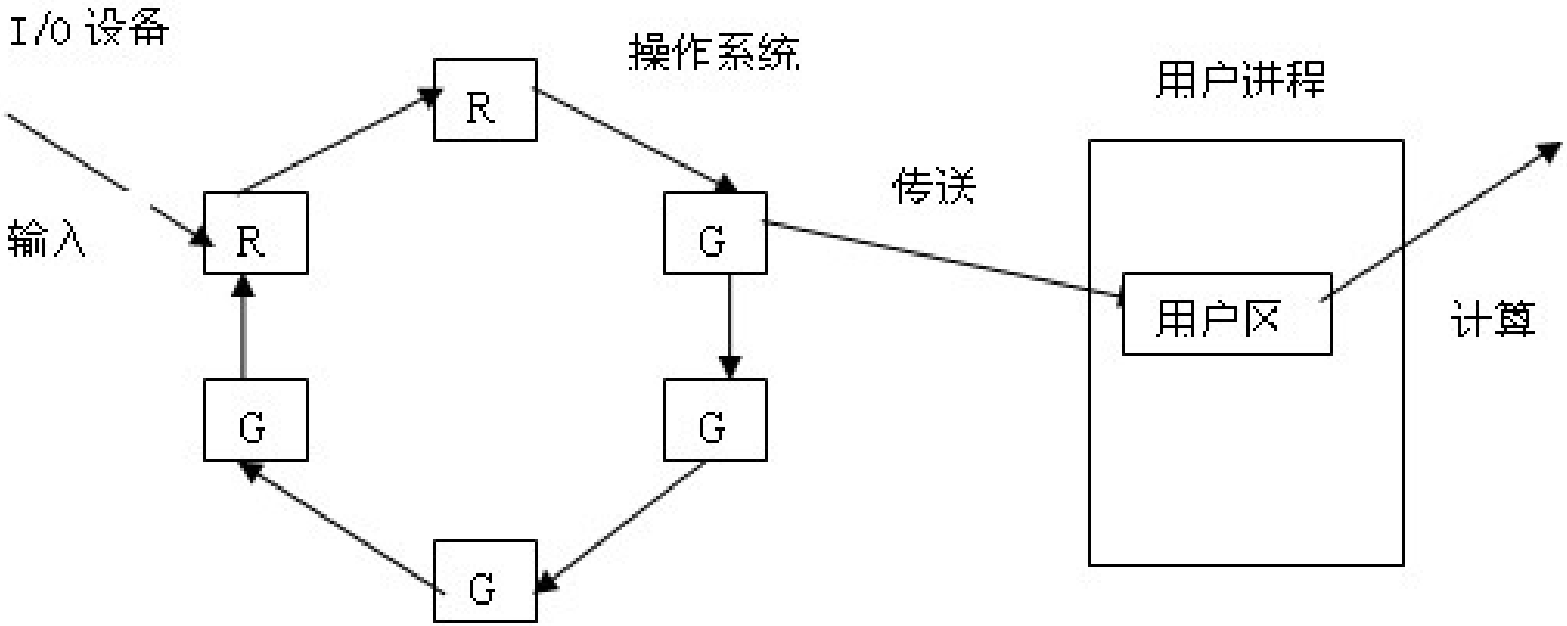
- 两个缓冲区，CPU和外设都可以连续处理而无需等待对方。要求CPU和外设的速度相近。



I/O 的双缓冲方式

环形缓冲(circular buffer)

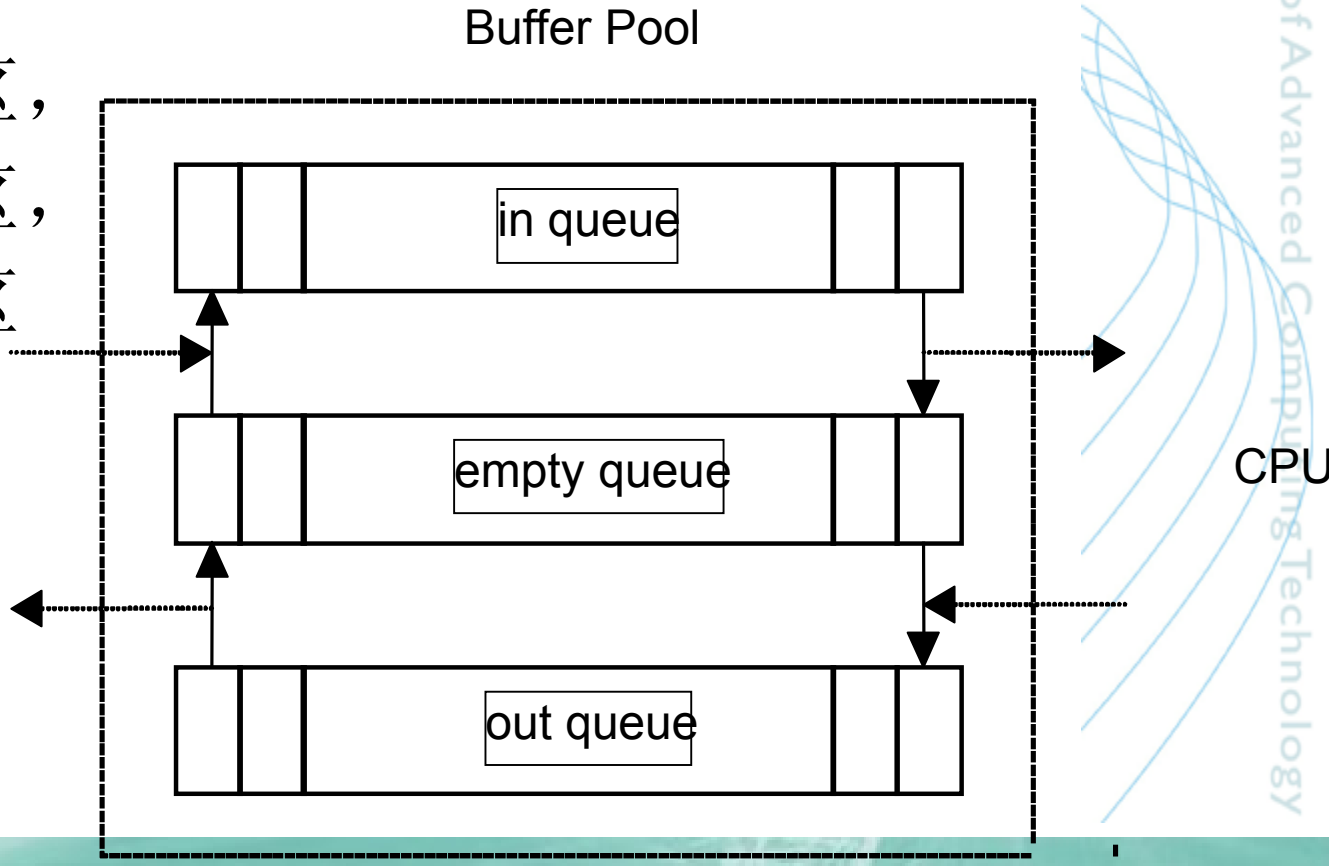
- 多个缓冲区，CPU和外设的处理速度可以相差较大。可参见“生产者－消费者问题”。



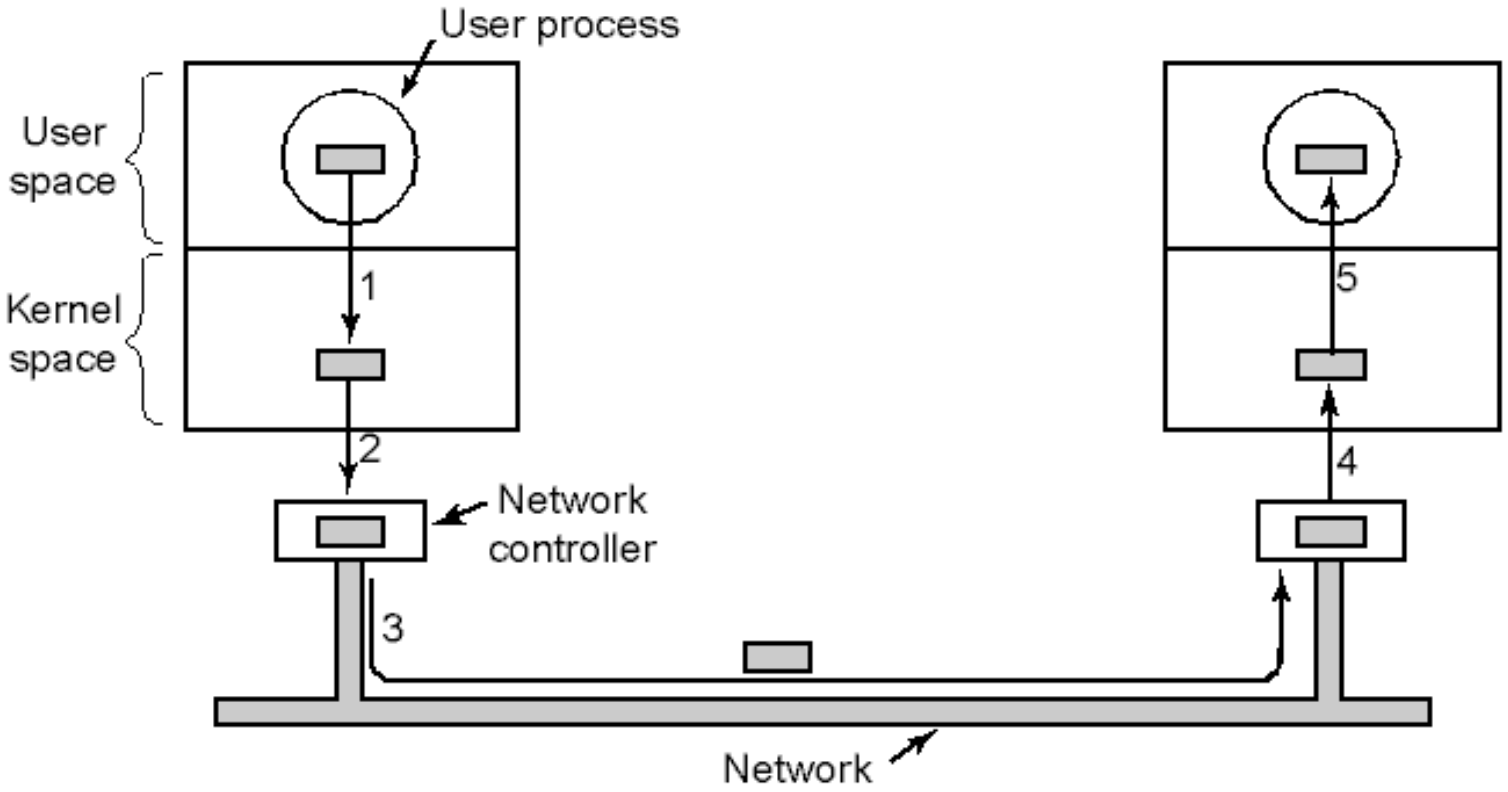
I/O 的循环缓冲方式

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



《现代操作系统》P199

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。设备分配是对进程使用外设过程的管理。
- 这里有两种作法：
  - 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。

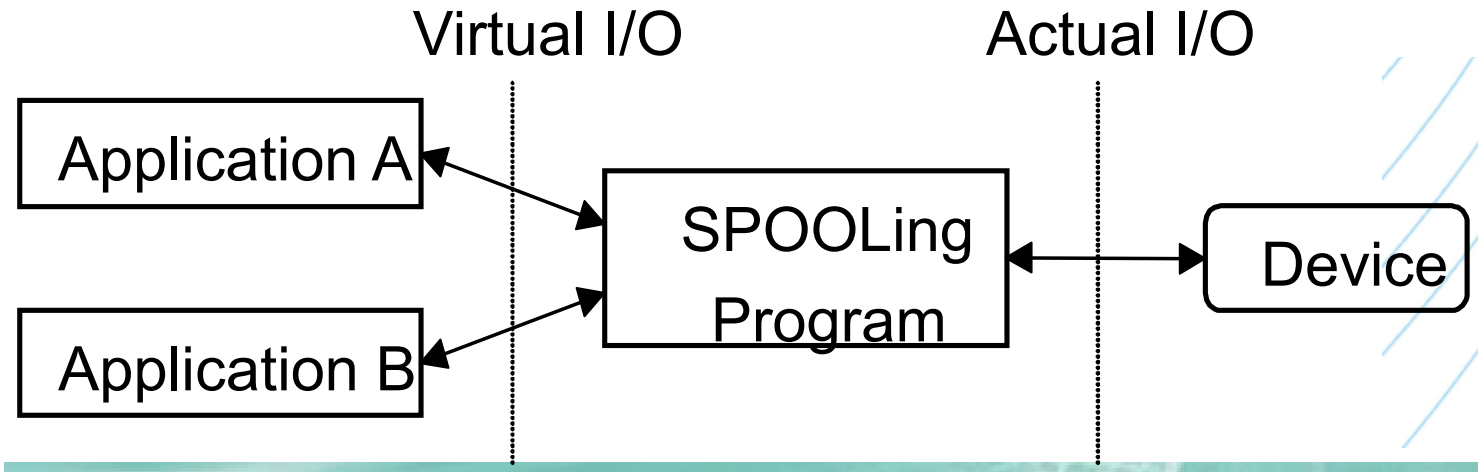
多通路I/O系统的设备分配

- 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

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。
- 引入
  - 在多道程序系统中，专门利用一道程序（SPOOLing程序）来完成对设备的I/O操作。无需使用外围I/O处理机。

- SPOOLing程序和外设进行数据交换：实际I/O
  - SPOOLing程序预先从外设输入数据并加以缓冲，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。



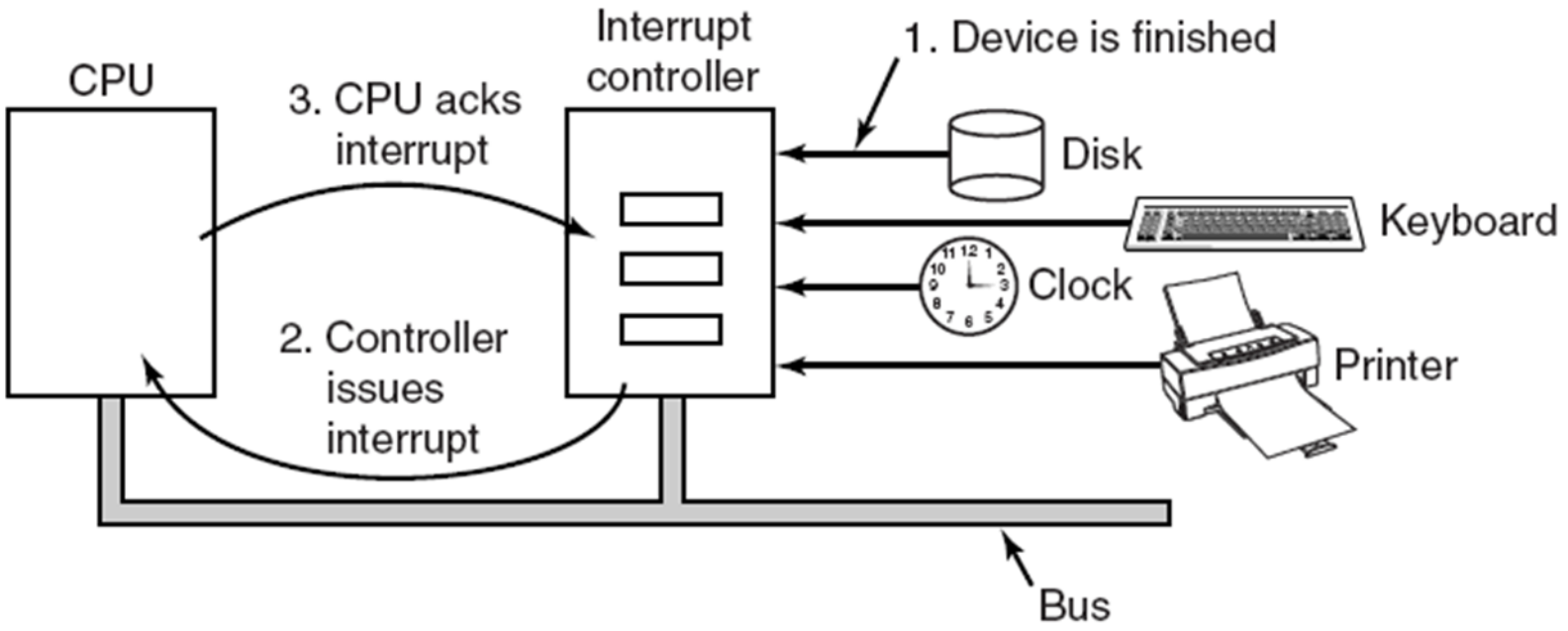
特点

- 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。
- 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。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。

中断处理过程



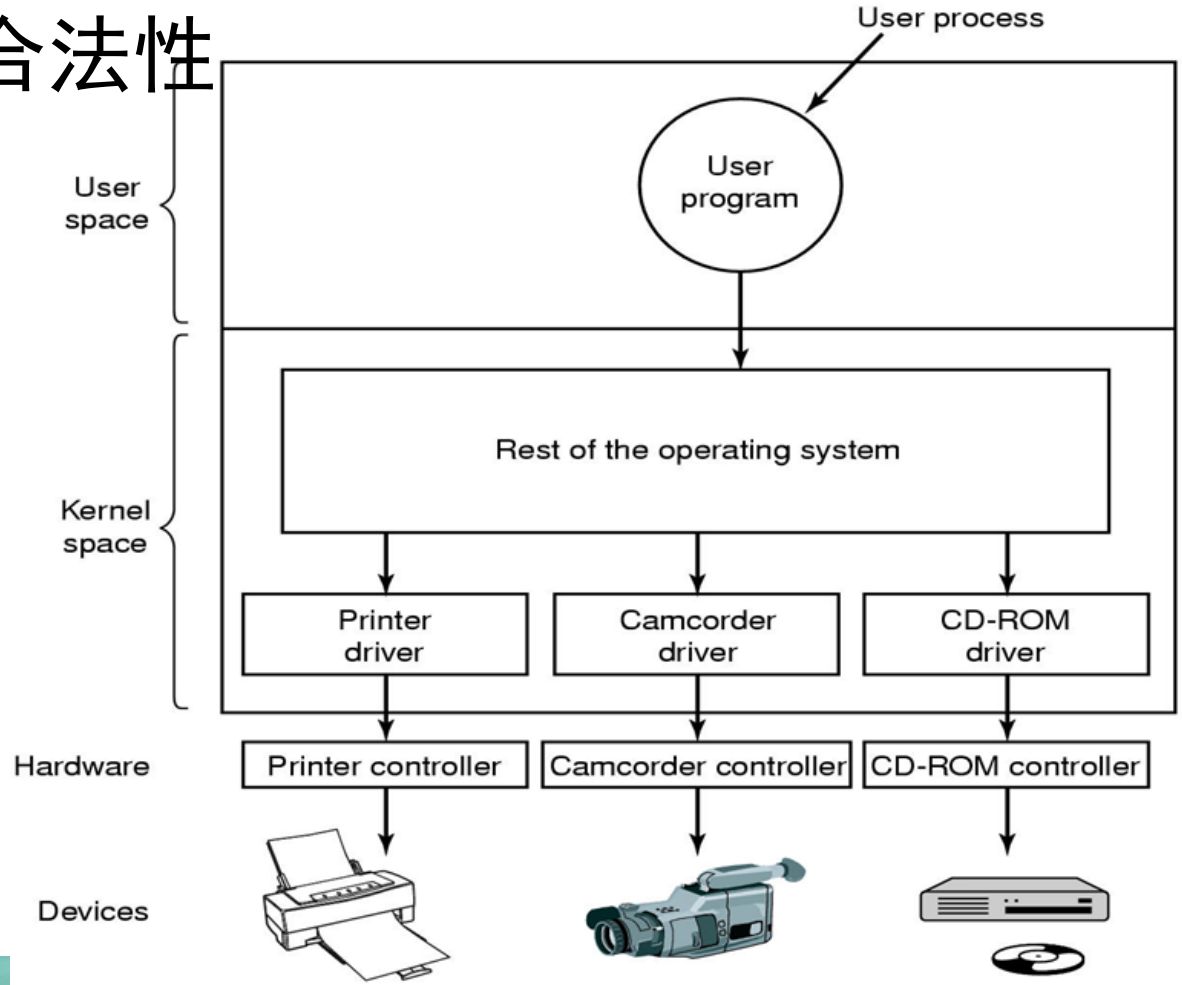
中断向量：中断服务程序的入口地址

中断处理过程

- 关中断
- 保存现场（包括断点）
- 转入设备中断处理程序
- 进行中断处理
- 恢复被中断进程的现场
- 开中断
- 返回断点（断点在哪里？）

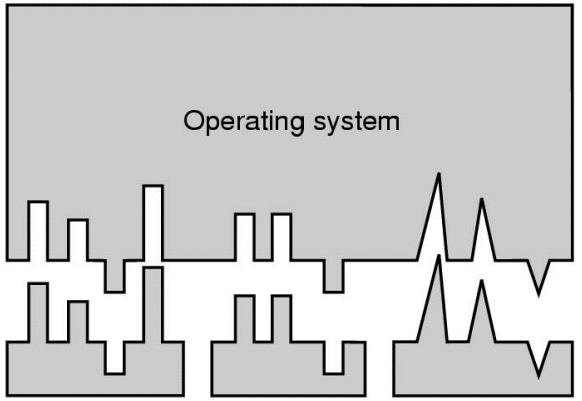
设备驱动程序的功能

- 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
- 初始化设备
- 启动设备
- 发出I/O命令
- 响应中断请求
- 构造通道程序

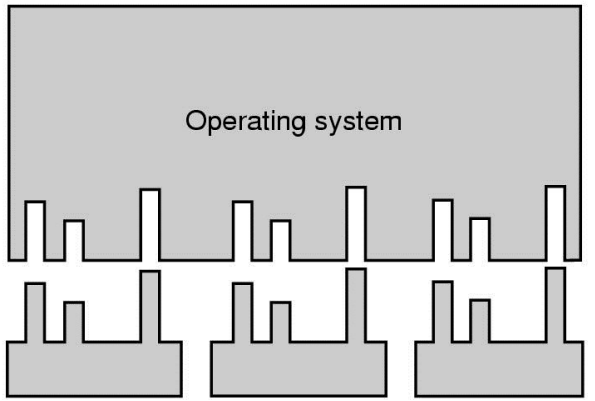


特点

- 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
- 与I/O控制方式紧密相关
- 与硬件紧密相关



非标准化



标准化

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，可在其中调用sleep()等与进程运行环境有关的函数。

- 中断服务子程序：系统来接收硬件中断，再由系统调用中断服务子程序。因为设备驱动程序一般支持同一类型的若干设备，所以一般在系统调用中断服务子程序的时候，都带有一个或多个参数，以唯一标识请求服务的设备。

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。
- 核心接口：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
- 核心机制与服务：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
- 动态性：系统启动及设备驱动初始化时将查找它所控制的硬件设备。若某个设备的驱动为一个空过程时，不会对系统造成危害，只是会占用少量系统内存

小结

- I/O管理的目标与功能
- I/O硬件管理
- I/O软件管理
  - 四种I/O控制方式：
    - 程序控制、中断、DMA、通道
  - 中断处理程序
  - 设备驱动程序
  - 缓存管理
  - 设备分配

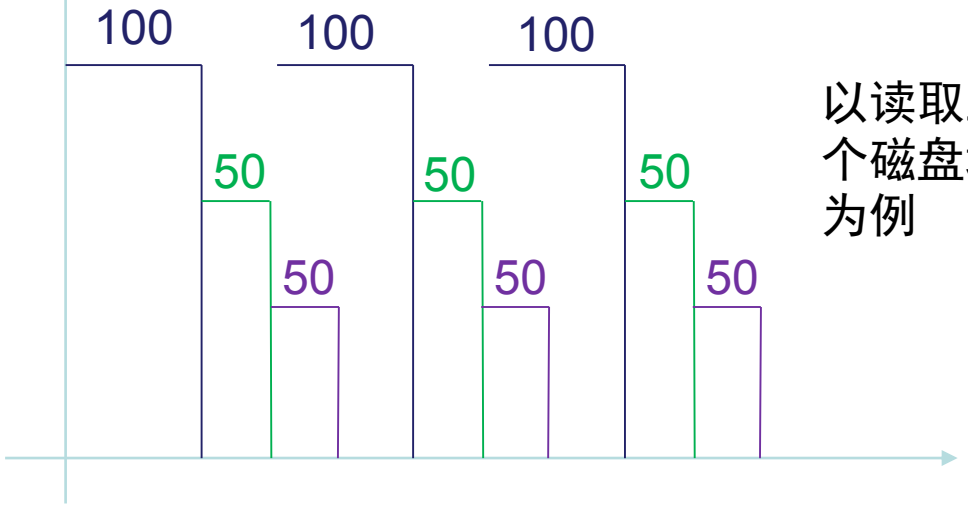
练习题1

- 某文件占10个磁盘块，现要把该文件的磁盘块逐个读入主存缓冲区，并送用户区进行分析。一个缓冲区与磁盘块大小相等。把一个磁盘块读入缓冲区的时间为100μs，缓冲区数据传送到用户区的时间是50μs，CPU对一块数据进行分析的时间为50μs。分别计算在单缓冲区和双缓冲区结构下，分析完该文件的时间是多少？

答案

- 使用单缓冲区：CPU和I/O进程轮流使用缓冲区，每处理一个磁盘块需要(100+50)微秒的I/O时间，而CPU处理数据的50微秒可以和下一次I/O进程并行，因此处理10个磁盘块的总时间为10*(100+50)+50=1550微秒
- 使用双缓冲区：CPU和I/O进程可同时分别使用两个缓冲区中的一个，I/O进程用100微秒读入一个磁盘块，CPU恰好用100(=50+50)微秒时间完成读取和处理，因此I/O进程和CPU正好可以完全并行。处理10个磁盘块需要的时间为：10*100+50+50＝1100微秒。
- 参见后页示意图。

单缓冲区



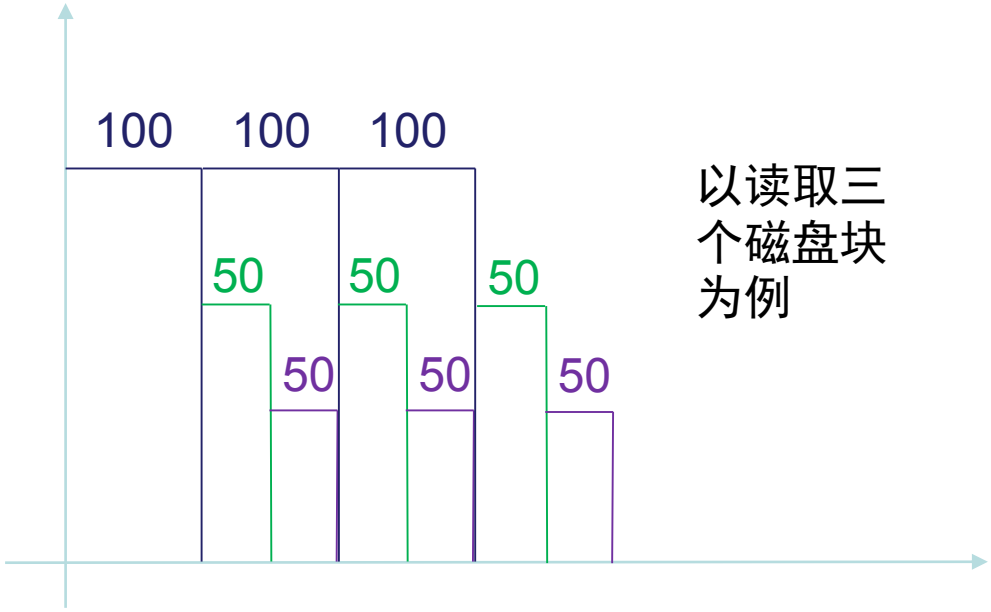
以读取三个磁盘块为例

读入缓冲区 100

读入用户区 50

CPU处理 50

双缓冲区



以读取三个磁盘块为例

缓冲改善了什么？

- 延迟
- 吞吐

练习题2

- 有一个典型的文本打印页面包括50行，每行80个字符。一台打印机每分钟可以打印6个页面，忽略将字符写到打印机输出寄存器的时间。如果每打印一个字符要请求一次中断，而进行中断服务要花费50μs时间，使用中断驱动的I/O来运行该打印机是否合理？
- 解答：打印机1分钟打印6*50*80=24000个字符，每个字符平均需要$(1*60*10^6)\div 24000=2500\mu s$。一次中断服务需要50μs，约占打印时间的2%，也就是说打印文本不需要频繁中断CPU，因此使用中断是合理的。

练习题3

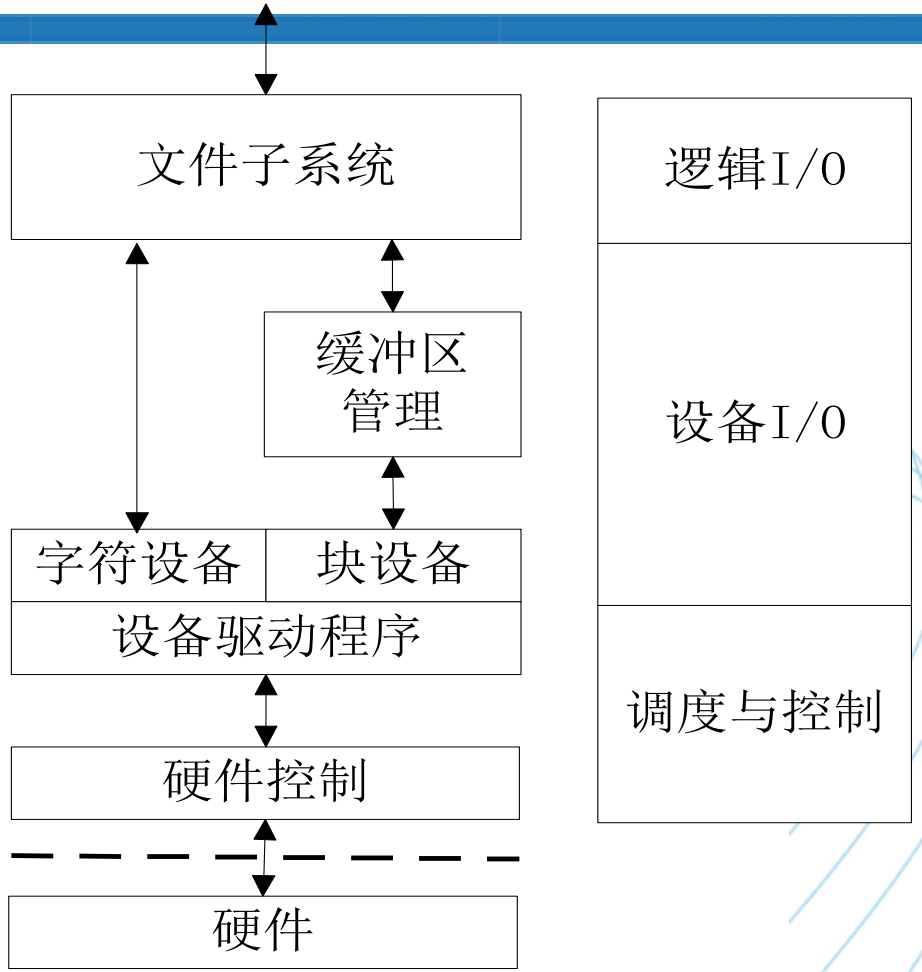
- 以下各项任务是在四个I/O软件层的哪一层完成的？
  - 为一个磁盘读操作计算磁道、扇区、磁头
  - 向设备寄存器写命令
  - 检查用户是否允许使用设备
  - 将二进制整数转换成ASCII码以便打印
- 解答
  - 设备驱动程序
  - 设备驱动程序
  - 设备无关软件
  - 用户程序
- 将系统调用参数翻译成设备操作命令的任务是由设备无关软件完成的，而不是设备驱动程序。相关细节可参见《现代操作系统》5.3节内容。

内容提要

- I/O硬件基本原理
- I/O软件基本原理
- OS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；

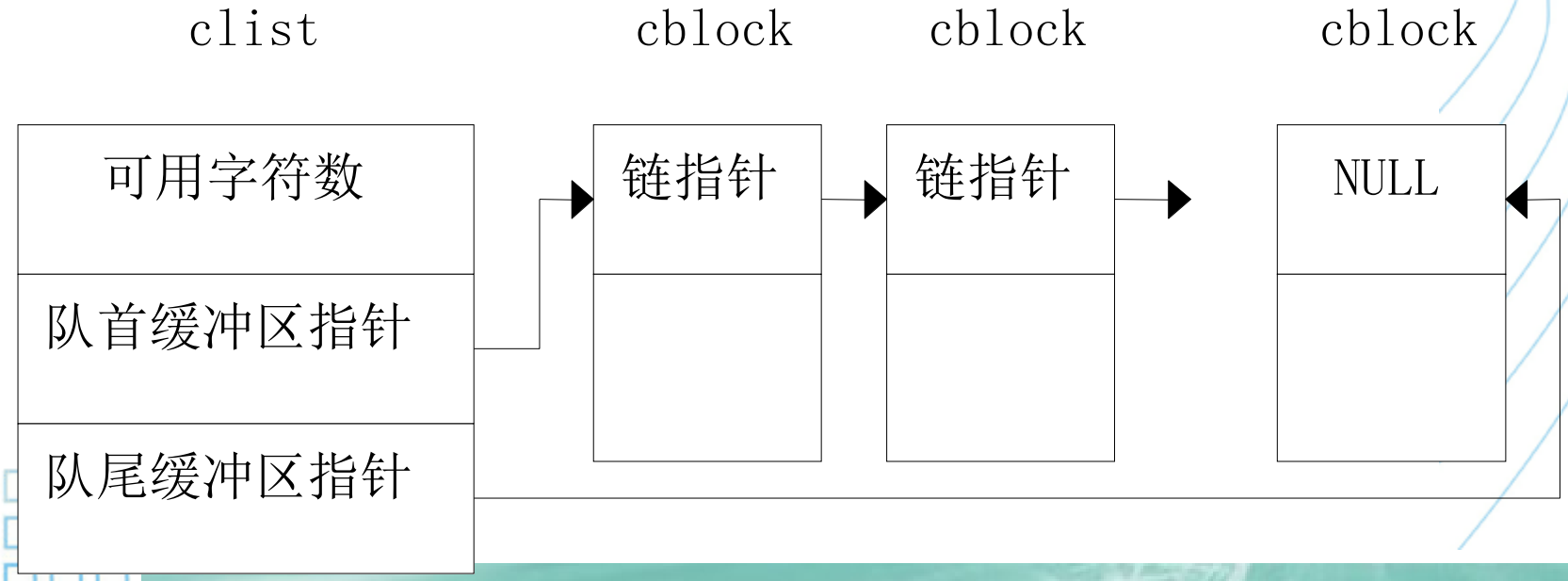
- 缓冲区检索和置换方式
  - 设备缓冲区队列为Hash队列：为了检索方便，设备缓冲区队列为一个按(逻辑设备号，物理块号)组织的Hash队列。把逻辑设备号和物理块号之和对64取模作为哈希函数值，据此建立多个哈希队列（64个队列）。
  - 缓存块可同时链入设备缓冲区队列和空闲缓冲队列：一个缓存块在分配给一个外设后，一直与该外设相关(既使该缓存块在空闲缓冲队列中)，直到分配给另一外设。即：设备释放缓冲区后，该缓冲区可处于“延迟写”状态，等待被写入到外设；该缓冲区被重新分配之前，要将其写入到外设。

- 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

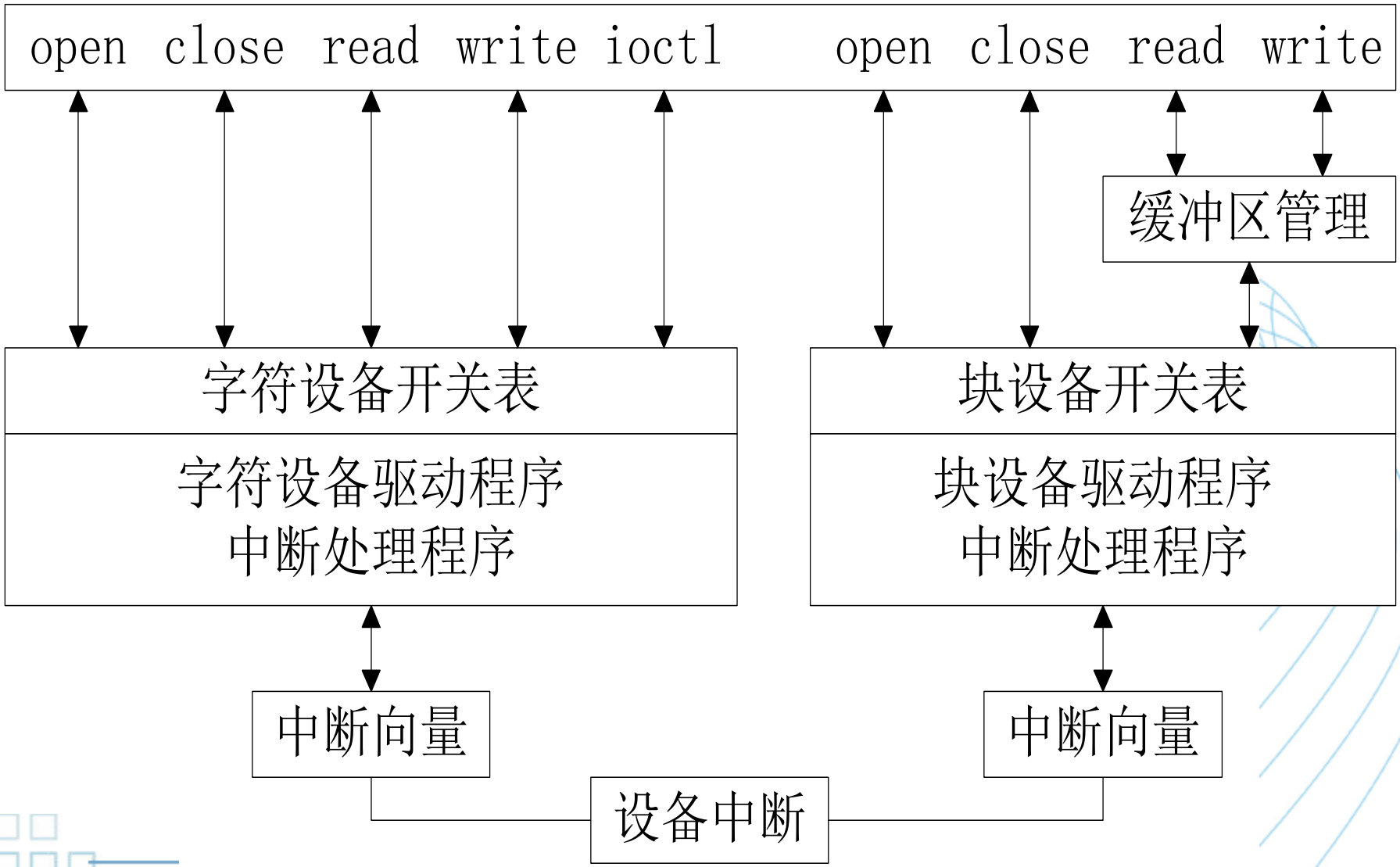


- 字符设备缓冲区的操作
  - 空闲缓冲区操作：申请空闲缓冲区、释放空闲缓冲区；
  - 字符设备缓冲队列操作：从队首读出一个字符、向队尾写入一个字符、读出队列缓冲区的所有字符、向队尾加入一个有数据的缓冲区、从队首读出n个字符、向队尾写入n个字符；

设备开关表(switch table)

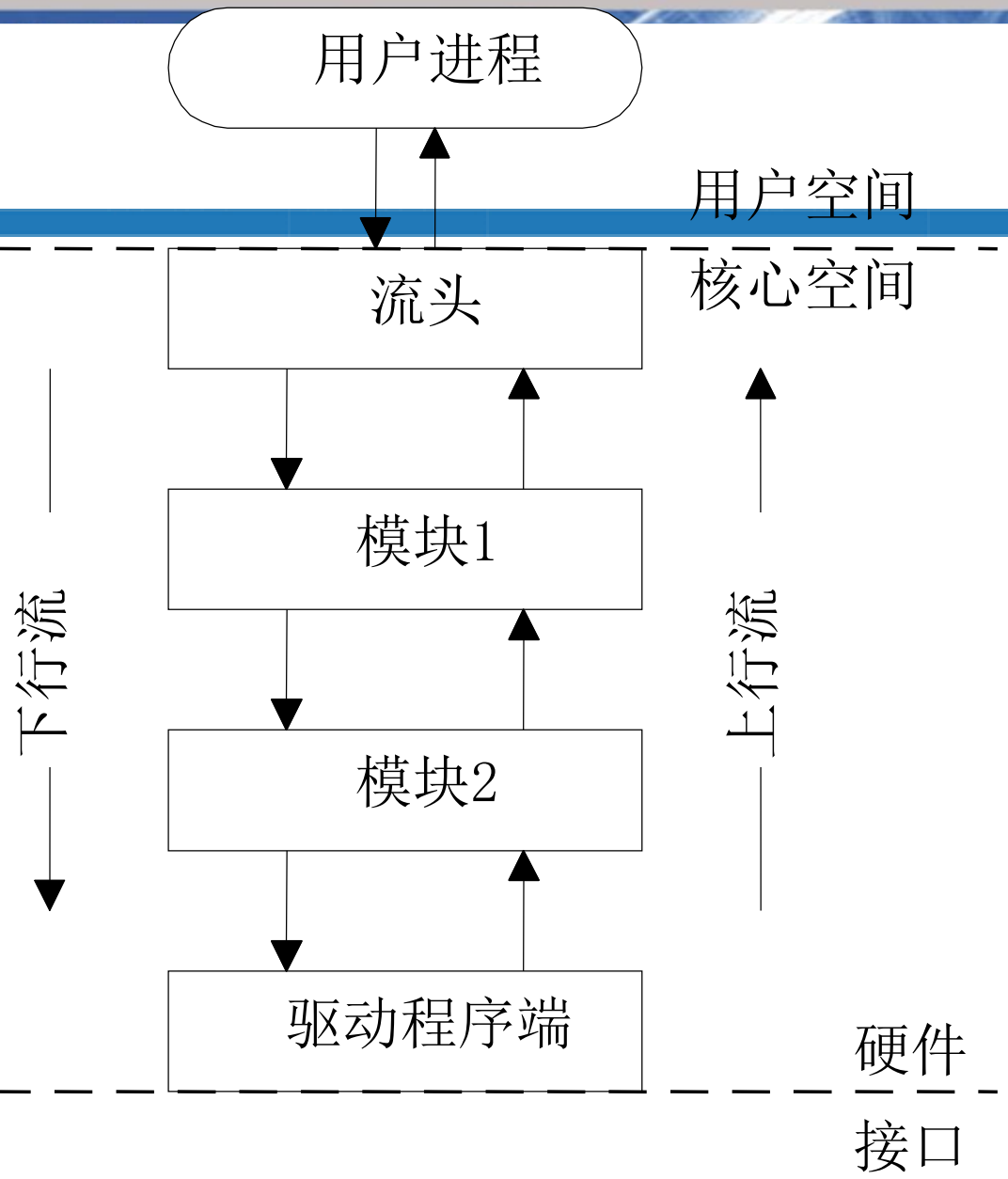
- UNIX设备驱动程序通过相应的块设备开关表和字符设备开关表描述向上与文件系统的接口。
- 开关表是每个设备驱动程序的一系列接口过程的入口表，给出了一组标准操作的驱动程序入口地址，文件系统可通过开关表中的各函数入口地址转向适当的驱动程序入口。

文件系统



流机制(streams)

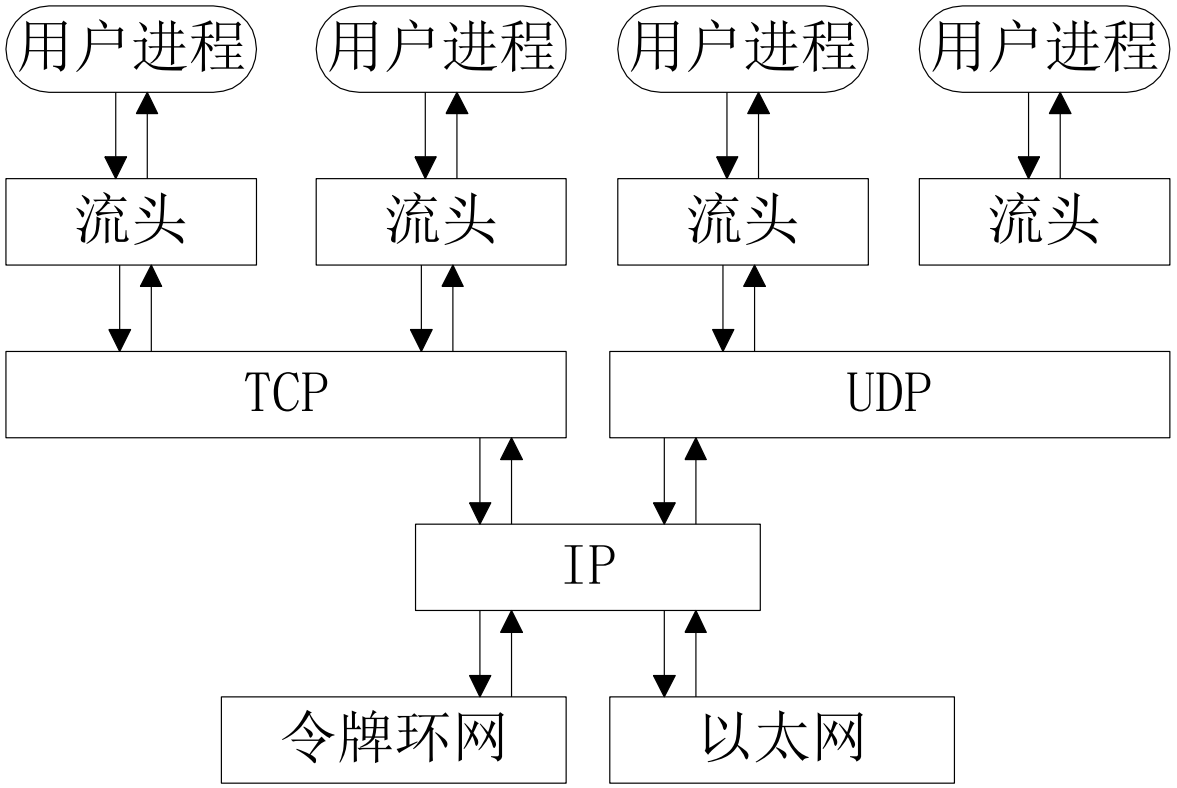
- 流的引入：流的引入是为了解决内核与驱动程序抽象层次过高，而引起的驱动程序功能大量重复。它可提供一个完全基于消息的模块化的驱动程序编写方法。
- 流的定义：流是一组系统调用、内核资源和创建、使用及拆除流的例程的集合，构成一个数据传输通道，两端为读队列和写队列。
- 流的结构：上行流(upstream)和下行流(downstream)



教材P229

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；

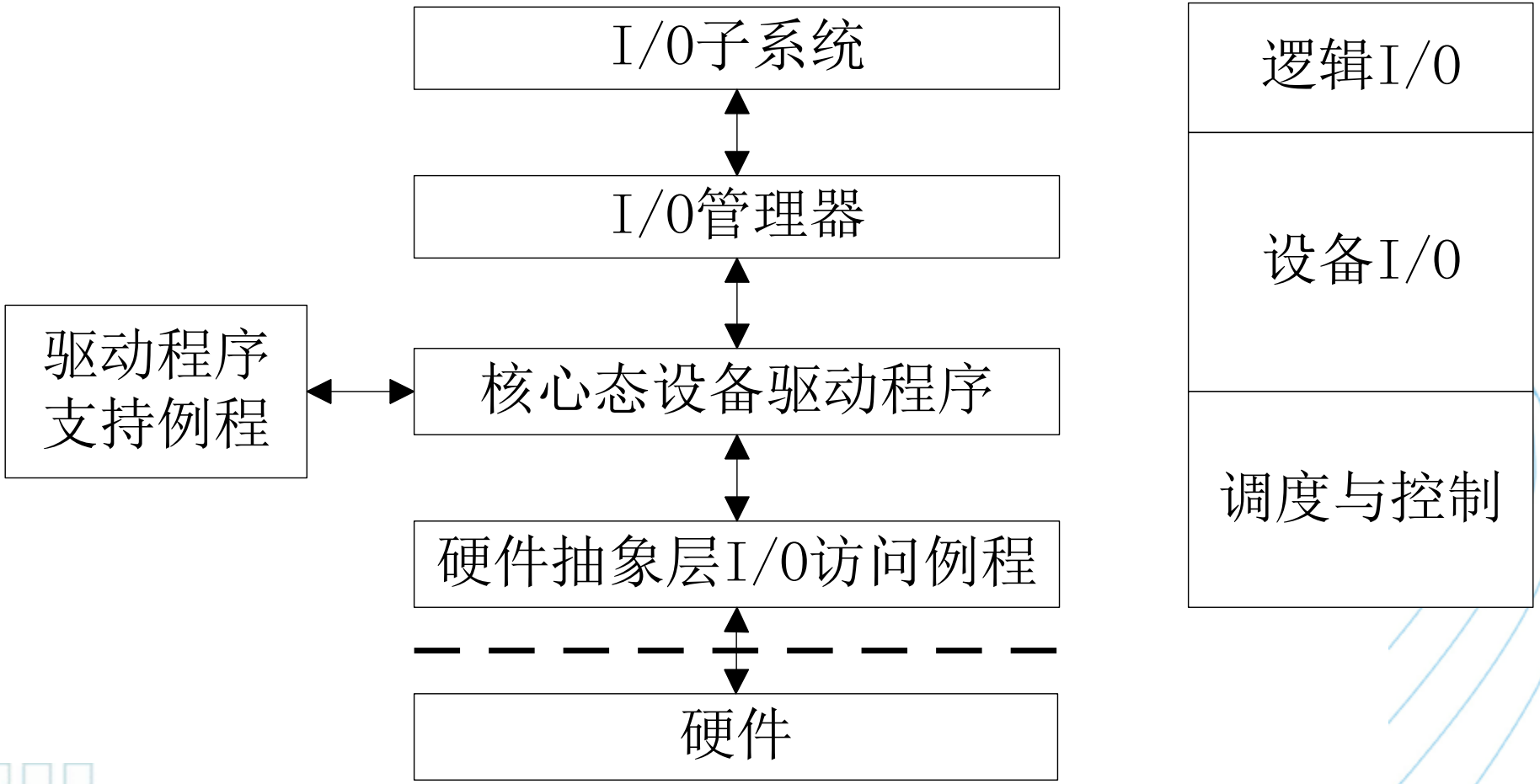


NT的I/O系统结构

- 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)；



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；

问题？