操作系统 Operation System

设备管理

王雷

wanglei@buaa.edu.cn

TEL：82316284

内容提要

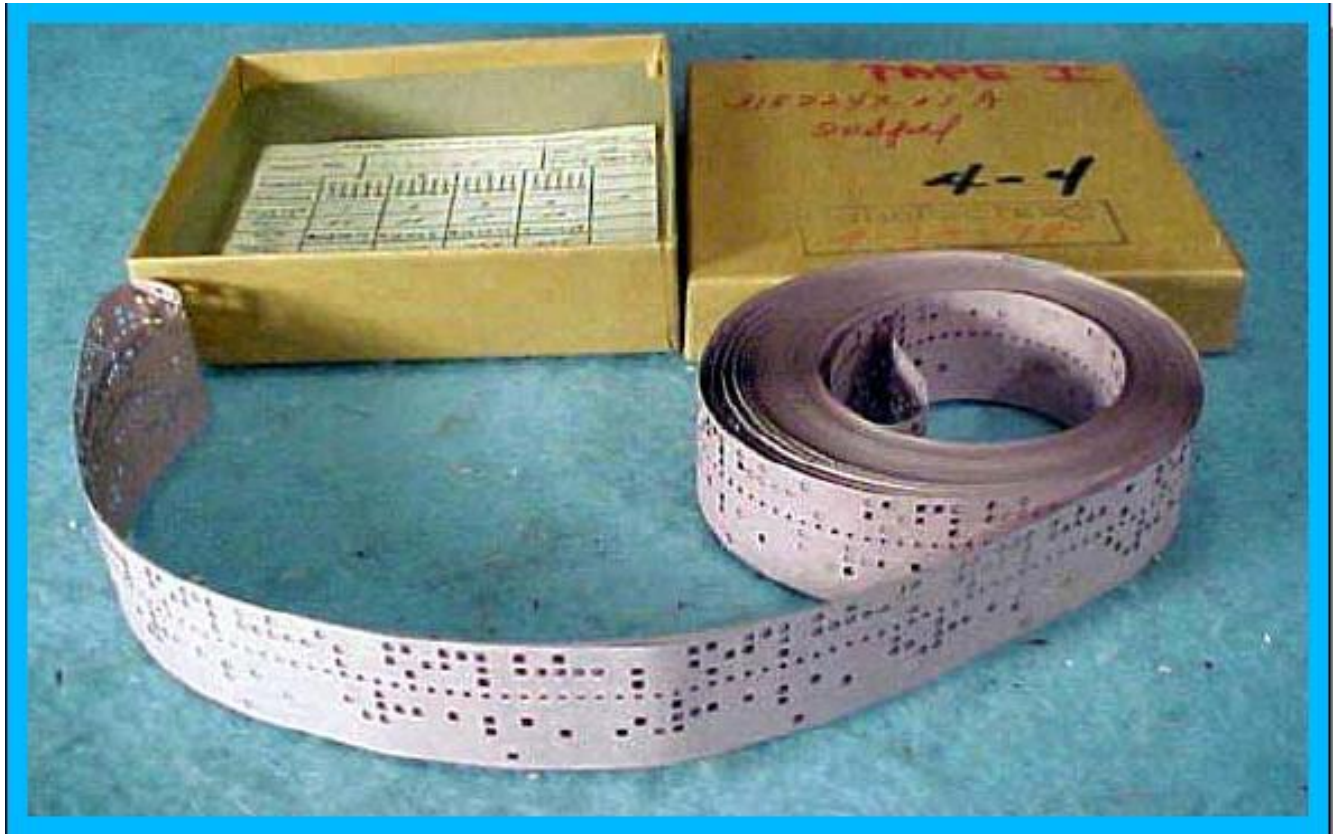
- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

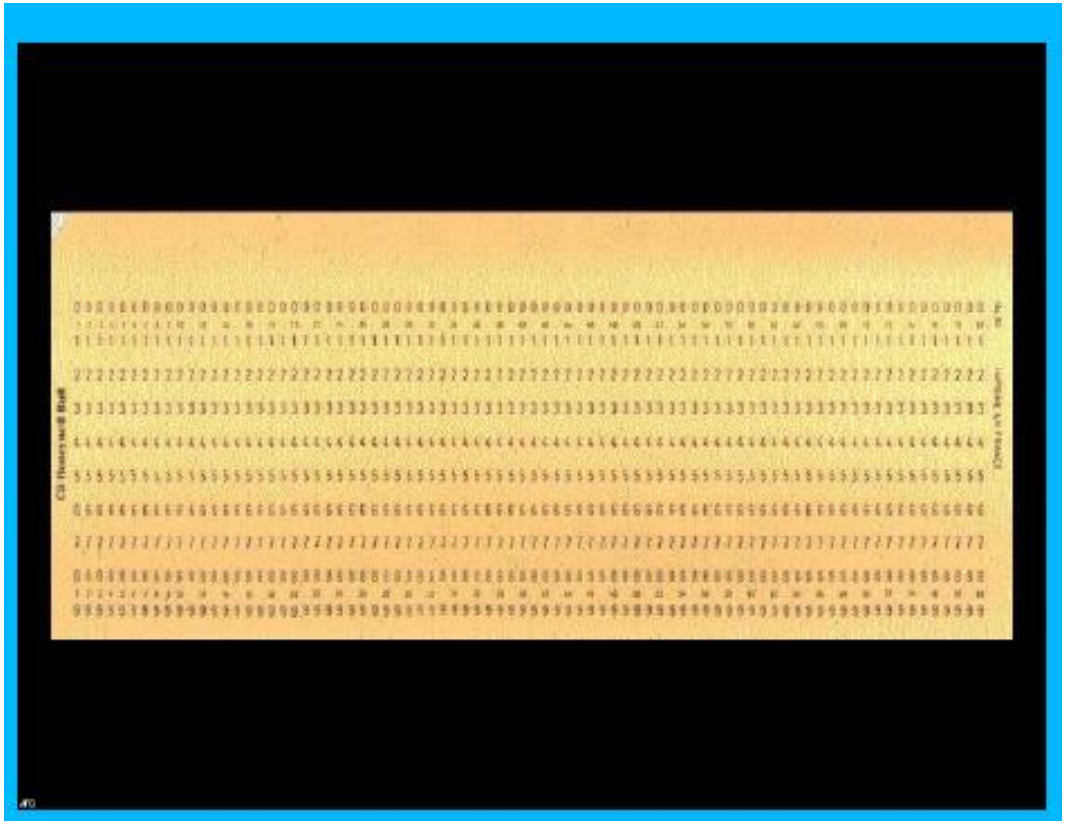
问题

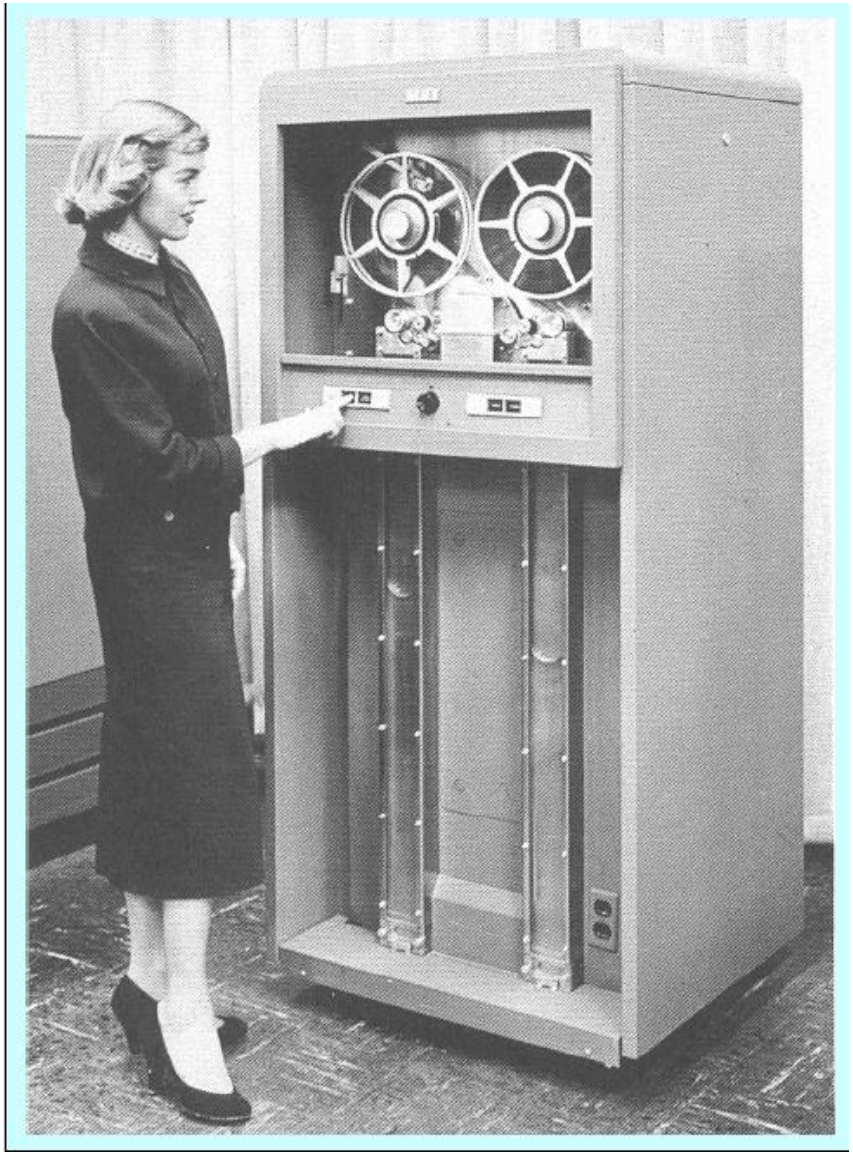
- 性能
- 复杂

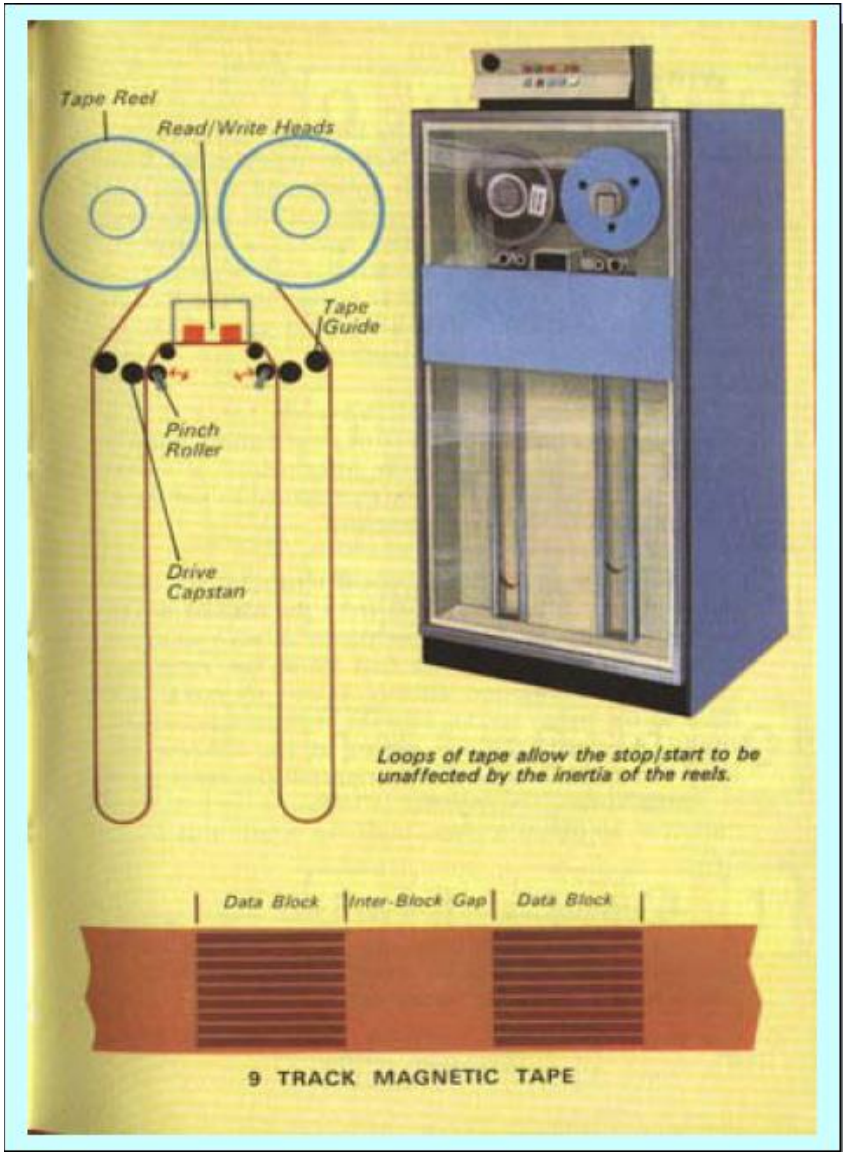
设备的管理的目的和功能

- 外设管理目的
  - 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：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。

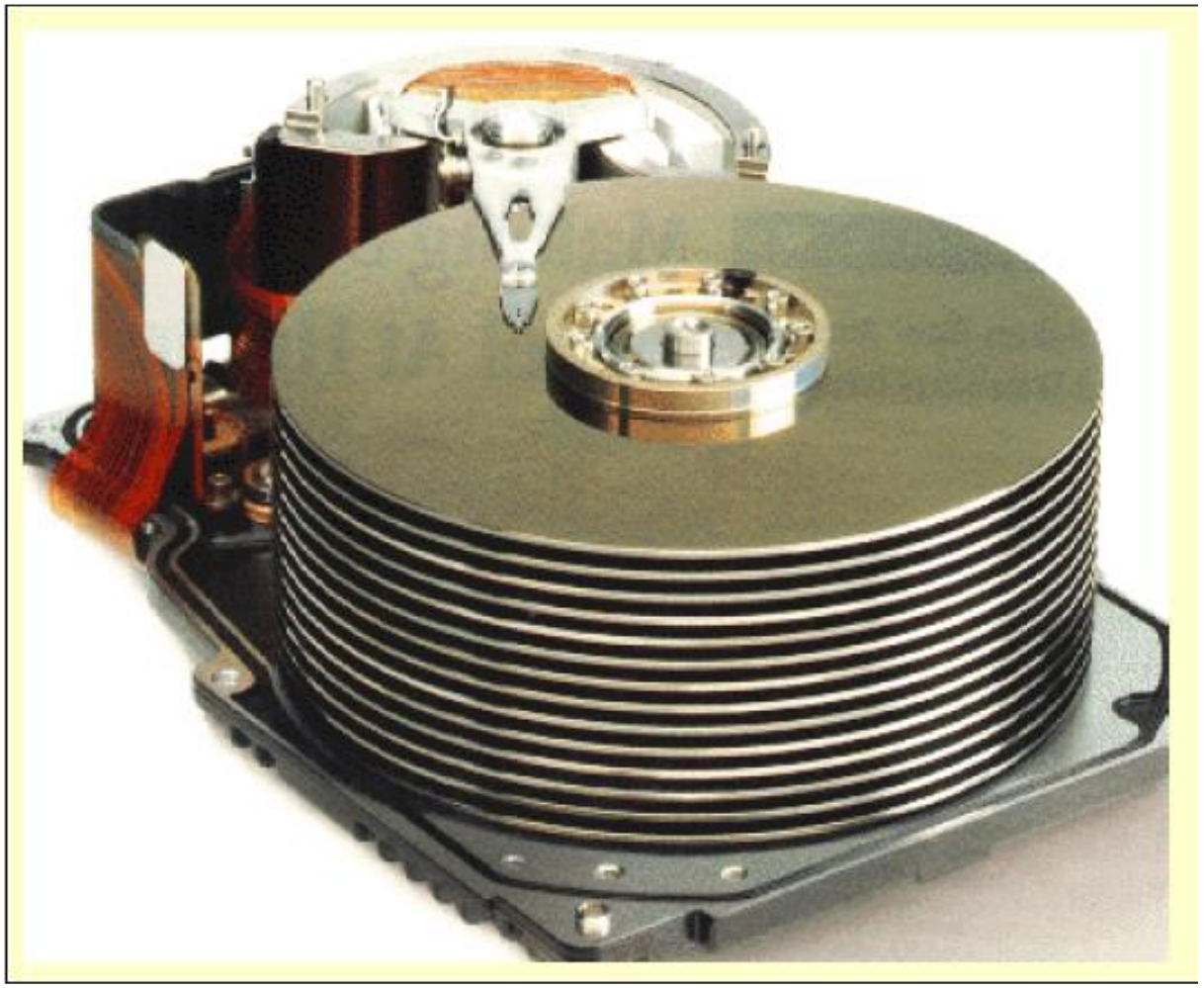

4-4



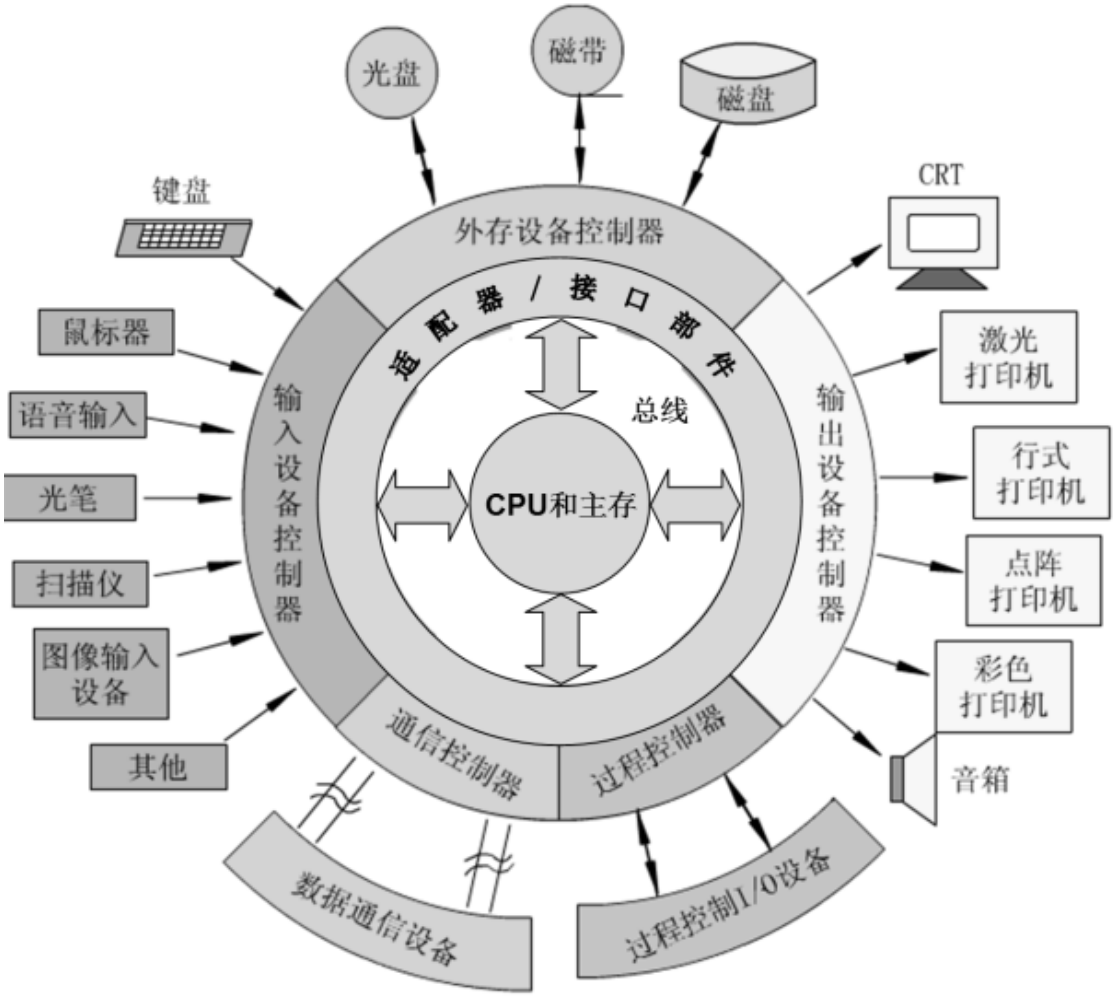


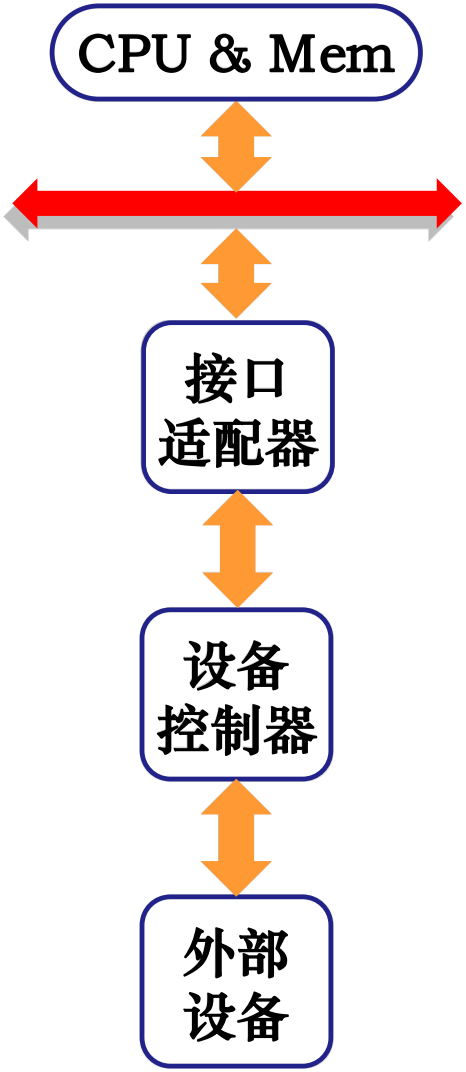

Tape Reel
Read/Write Heads
Tape Guide
Pinch Roller
Drive Capstan
Loops of tape allow the stop/start to be unaffected by the inertia of the reels.
Data Block
Inter-Block Gap
Data Block
9 TRACK MAGNETIC TAPE


CDC 6600

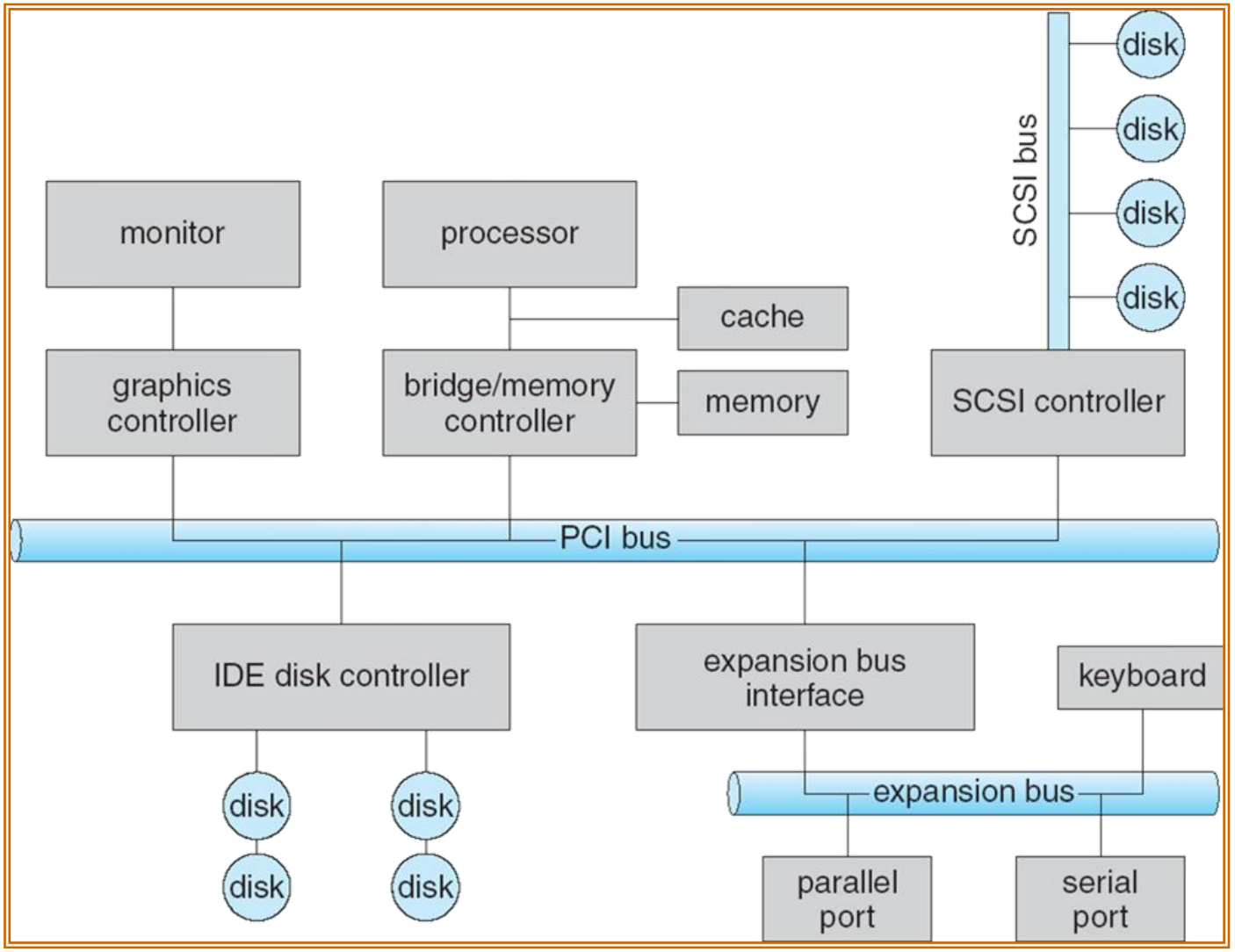


计算机I/O系统

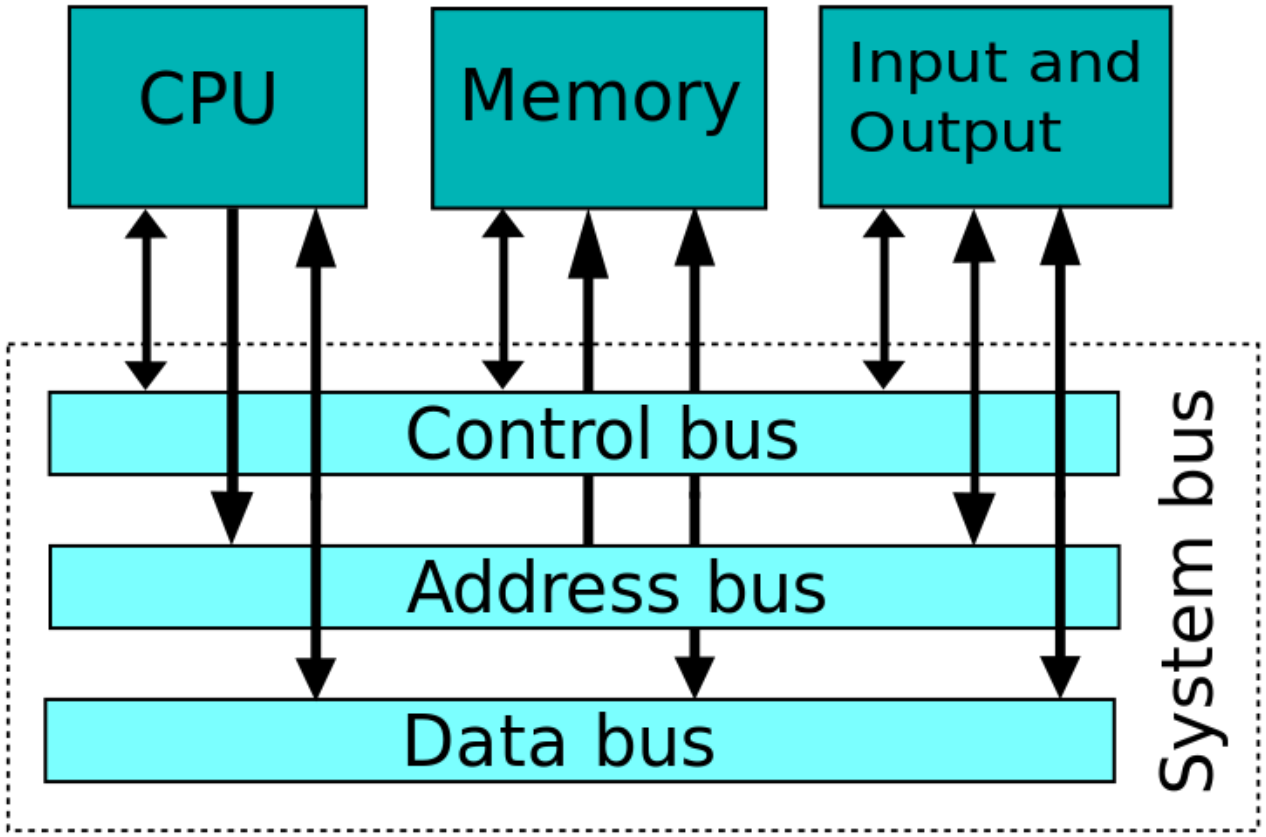

光盘
磁带
磁盘
键盘
CRT
外存设备控制器
适配器/接口部件
鼠标器
语音输入
光笔
扫描仪
图像输入设备
其他
输入设备控制器
总线
CPU和主存
输出设备控制器
激光打印机
行式打印机
点阵打印机
彩色打印机
音箱
通信控制器
过程控制器
数据通信设备
过程控制I/O设备

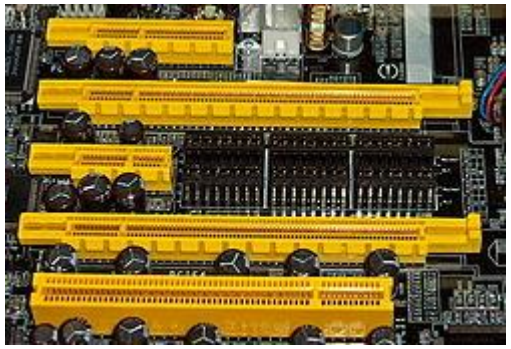

CPU & Mem
接口适配器
设备控制器
外部设备

总线(Bus)：接入I/O设备的主要方式



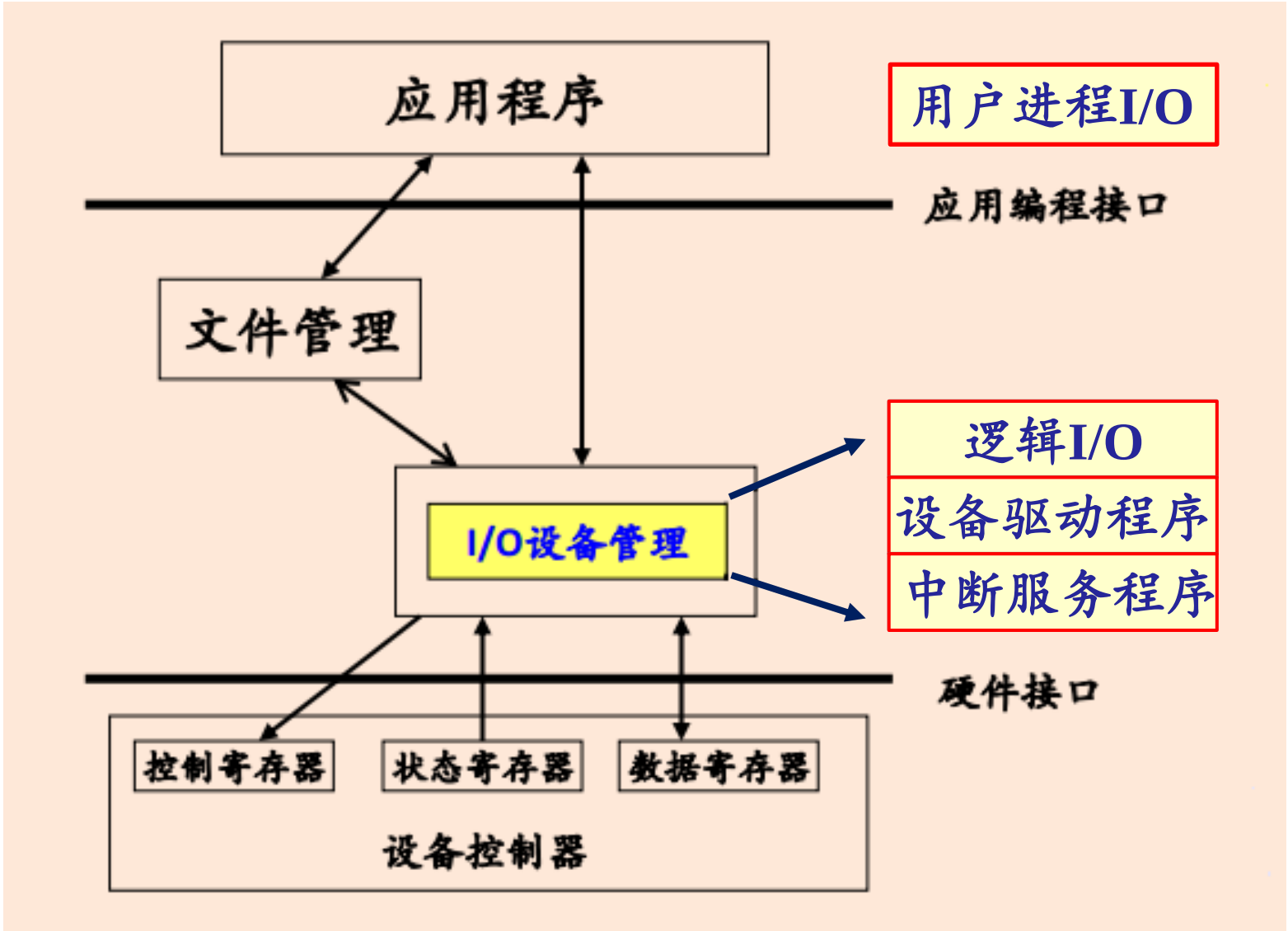
总线(Bus)：接入I/O设备的主要方式





总线带宽 = 频率 x 宽度（Bytes/sec）

I/O管理示意（软件角度）


应用程序
用户进程I/O
应用编程接口
文件管理
逻辑I/O
I/O设备管理
设备驱动程序
中断服务程序
硬件接口
控制寄存器
状态寄存器
数据寄存器
设备控制器

I/O管理示意（软件角度）

- 用户进程在运行过程中，提出I/O 请求， 一旦请求被操作系统接受，操作系统则负责完成该请求。
- 我们把硬件设备抽象为控制器。控制器中包含控制寄存器、状态寄存器，以及一些数据的寄存器。从操作系统角度，是通过对这些寄存器进行相应的控制，达到控制设备的目的。
- 在OS中，I/O 设备管理可直接从应用程序或文件系统得到请求，并负责完成这个请求。它具体包括：
  - 逻辑I/O :完成设备无关的操作，如设备分配，设备回收，数据准备等；
  - 设备驱动程序：负责对设备控制器进行控制（通过读写其中的寄存器）。
  - 中断服务程序：设备工作结束后负责向 CPU 发中断信号。

I/O管理的特点

- I/O性能经常成为系统性能的瓶颈；
- 操作系统庞大复杂的主要原因之一：资源多、杂，并发，均来自I/O；
  - 速度差异很大
  - 控制接口复杂
  - 传输单位不同
  - 数据表示各异
  - 错误条件多样
- 与其它功能联系紧密，特别是文件系统。

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

I/O设备的分类

按数据组织分类：

- 块设备：以数据块为单位存储、传输信息。传输速率较高、可寻址（随机读写）
- 字符设备：以字符为单位存储、传输信息。传输速率低、不可寻址。

按用途分类：

- 存储设备：磁盘、磁带；
- 传输设备：网卡，Modem；
- 人机交互设备：显示器、键盘、鼠标。

I/O设备的分类

从资源分配角度:

- 独占设备：在一段时间内只能由一个进程使用的设备。通常独占设备的传输速率比较慢，打印机和磁带机都属于典型的独占设备。
- 共享设备：在一段时间内允许多个进程共同使用的设备。多个进程以交叉 的方式来使用设备。资源利用率较高。硬盘是典型的共享设备。
- 虚设备：在一类设备上模拟另一类设备，常用的方法是，用共享设备模拟独占设备，用高速设备模拟低速设备。如：用Spooling技术将打印机变成共享设备。

I/O管理的目标和任务

1. 按照用户请求，控制设备操作，完成I/O设备与内存间的数据交换，最终完成用户的I/O请求。
   - 设备的分配与回收
     - 记录设备的状态；
     - 根据用户的请求和设备的类型，采用一定的分配算法，选择一条数据通路；
   - 执行设备驱动程序，实现真正的I/O操作；
   - 设备中断处理：处理外部设备的中断；
   - 缓冲区管理：管理不同的I/O缓冲区。

I/O管理的目标和任务

2. 建立方便、统一的独立于设备的接口

- 方便性：向用户提供易于使用的外部设备的接口，使用户编程时不必考虑设备复杂的物理特性；
- 统一性：对不同的设备采用统一的操作方式，即在用户程序中使用逻辑设备。逻辑设备是对物理设备的抽象，它屏蔽了物理硬件的细节（设备繁多的物理特性、错误处理、不同I/O过程的差异）。

I/O管理的目标和任务

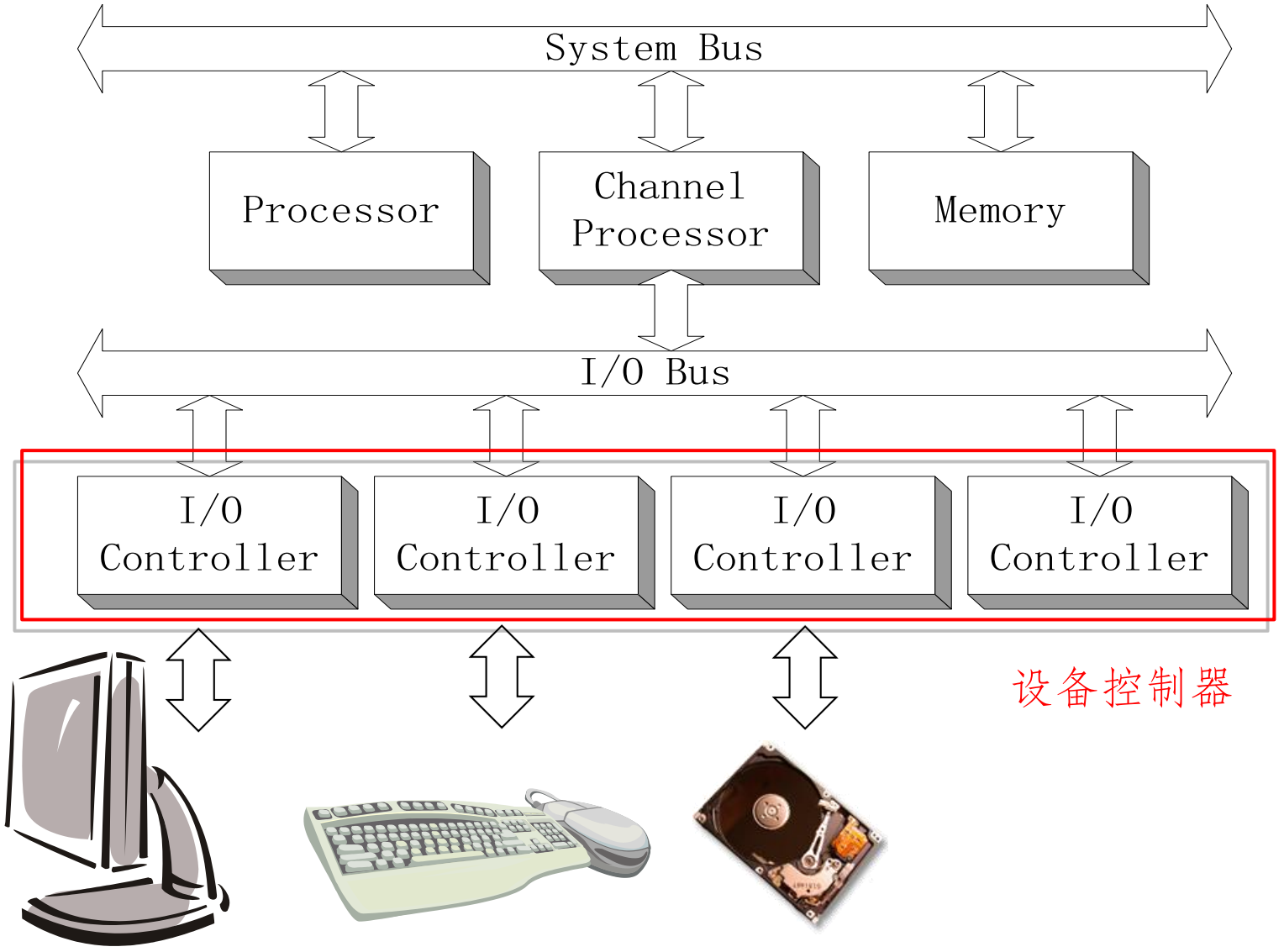
3. 充分利用各种技术提高CPU与设备、设备与设备之间的并行工作能力，充分利用资源（并行性）。由于CPU与I/O间的速度差异很大，应尽可能减少速度差异造成的整体性能开销，尽可能使CPU和设备都处于充分忙碌状态（均衡性）。

4. 保护：设备传送或管理的数据应该是安全的、不被破坏的、保密的。

内容提要

- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

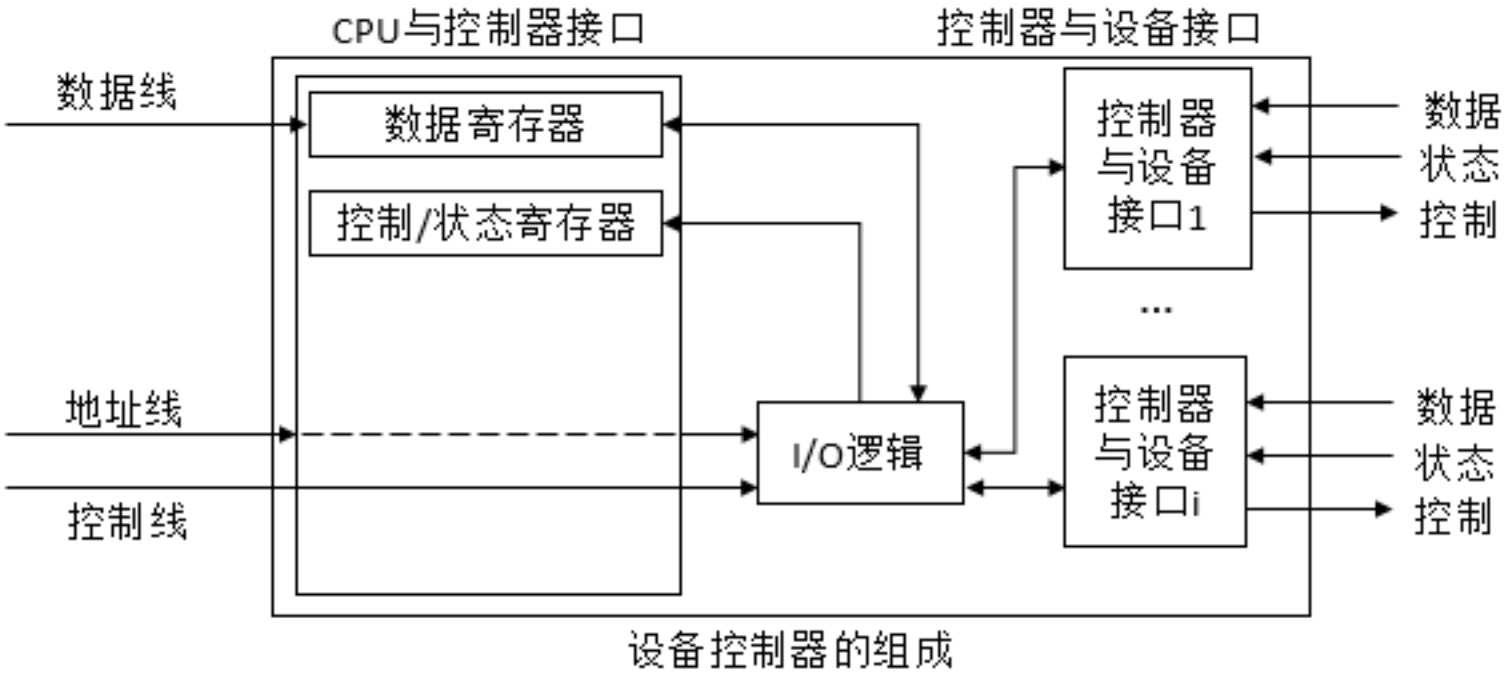
I/O系统的抽象结构


System Bus
Processor
Channel Processor
Memory
I/O Bus
I/O Controller
I/O Controller
I/O Controller
I/O Controller
设备控制器

设备控制器

- **控制器的功能**
  - 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CPU接口：数据寄存器、控制寄存器、状态寄存器，采用内存映射或专门的I/O指令
  - 控制器与设备接口：数据信号、控制信号、状态信号
  - I/O逻辑：用于实现CPU对I/O设备的控制

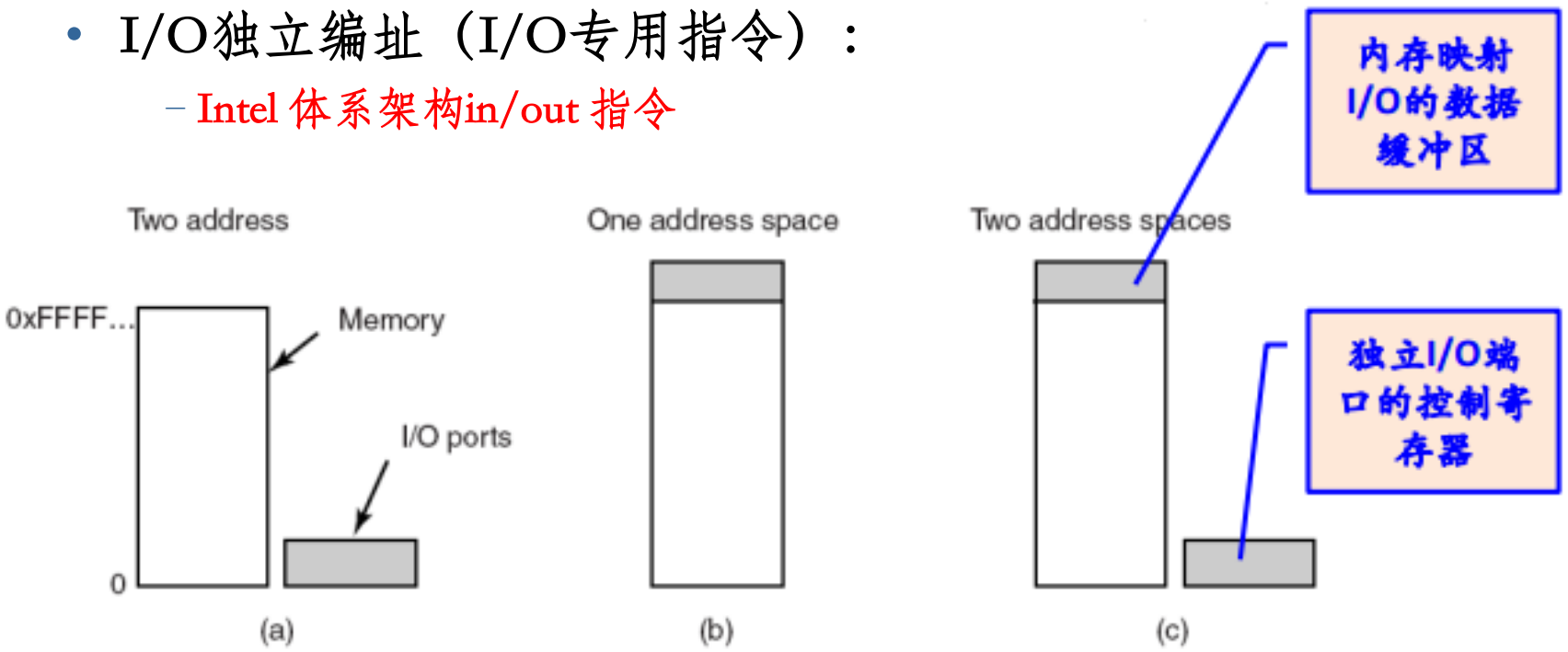
设备控制器



设备控制器的组成

I/O端口地址

- I/O端口地址：接口电路中每个寄存器具有唯一的地址
- 所有I/O端口地址形成I/O端口的地址空间（受OS保护）
- I/O指令形式与I/O地址是相互关联的，主要有以下形式：
  - 内存映像编址（内存映像I/O模式）：
    - 控制器的内存/寄存器作为物理内存空间的一部分
  - I/O独立编址（I/O专用指令）：
    - Intel 体系架构in/out 指令



内存映射I/O的特点

优点：

- 不需要特殊的保护机制来阻止用户进程进行相应的I/O 操作。操作系统要避免把包含了控制寄存器的那部分地址空间放入用户的虚拟地址空间中。
- 可以引用内存的每一条指令都可以适用于引用控制寄存器。

缺点：

- 不允许对一个控制寄存器的内容进行高速缓存。
  - 如果我们把设备控制寄存器进行了高速缓存，那么第一次引用的时候就把它 放入了高速缓存。以后再对它的引用都是从高速缓存当中取值，而不会再去对设备进行相应的检测

I/O独立编址的特点

优点：

- 外设不占用内存的地址空间
- 编程时，易于区分是对内存操作还是对I/O操作

缺点：

- I/O端口操作的指令类型少，操作不灵活

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)

例：Intel8086/8088，分配给I/O端口的地址空间为64k，0000H~0FFFFH，只能用in和out指令进行读写操作

内容提要

- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

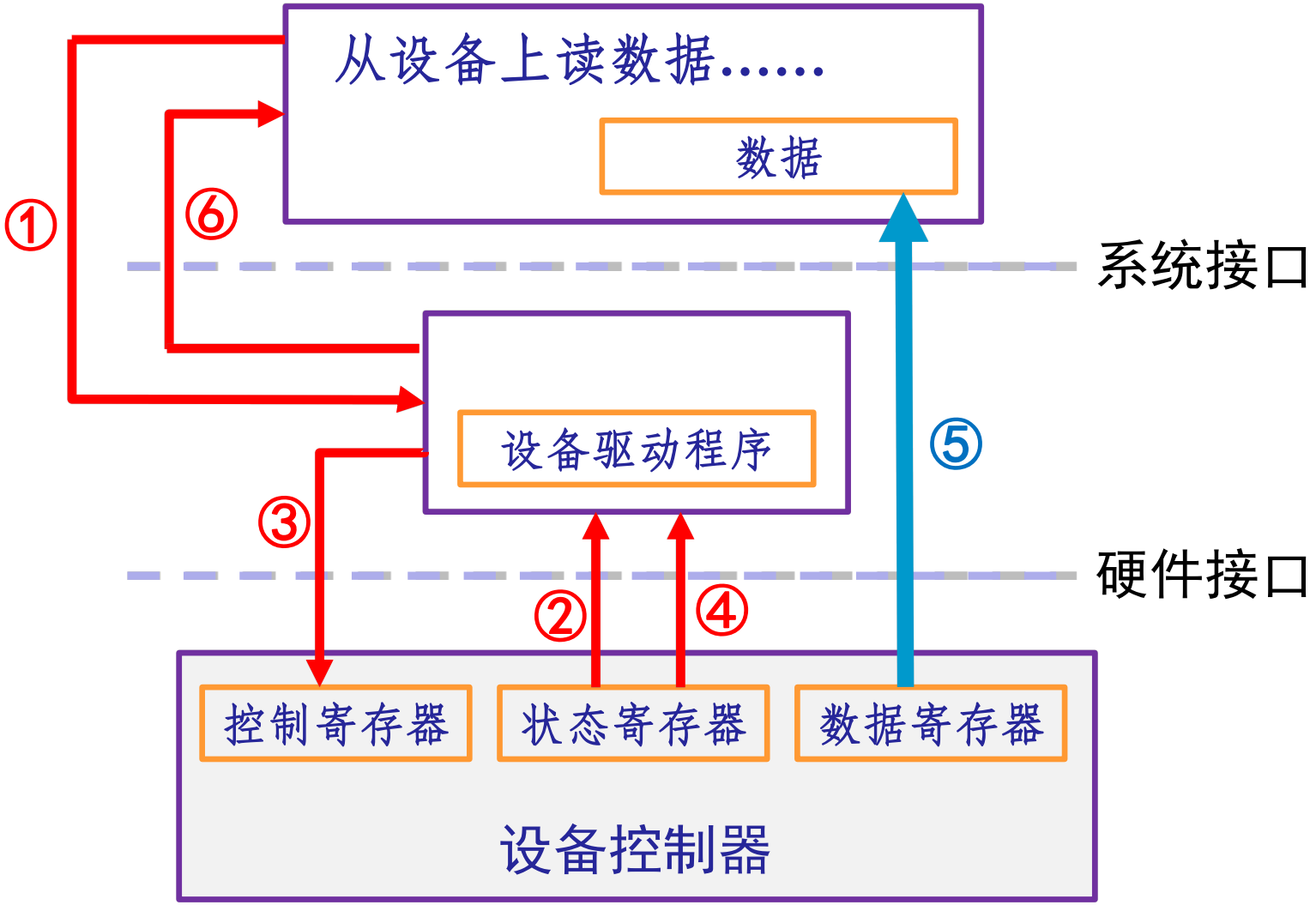
I/O控制技术

- 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。

程序控制I/O（轮询方式）



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；

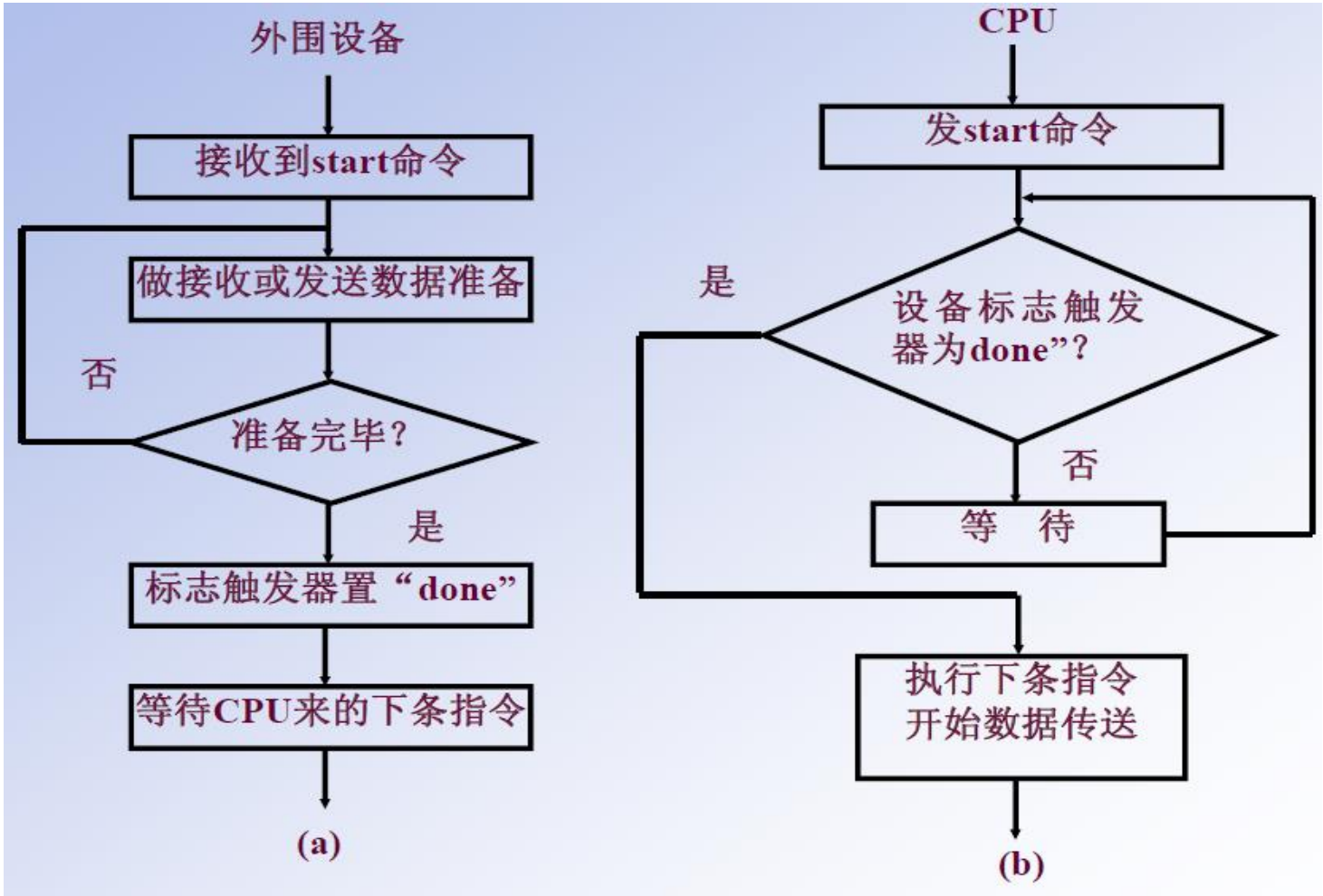
③ 如果状态正常，就给设备发出相应的控制命令；

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；

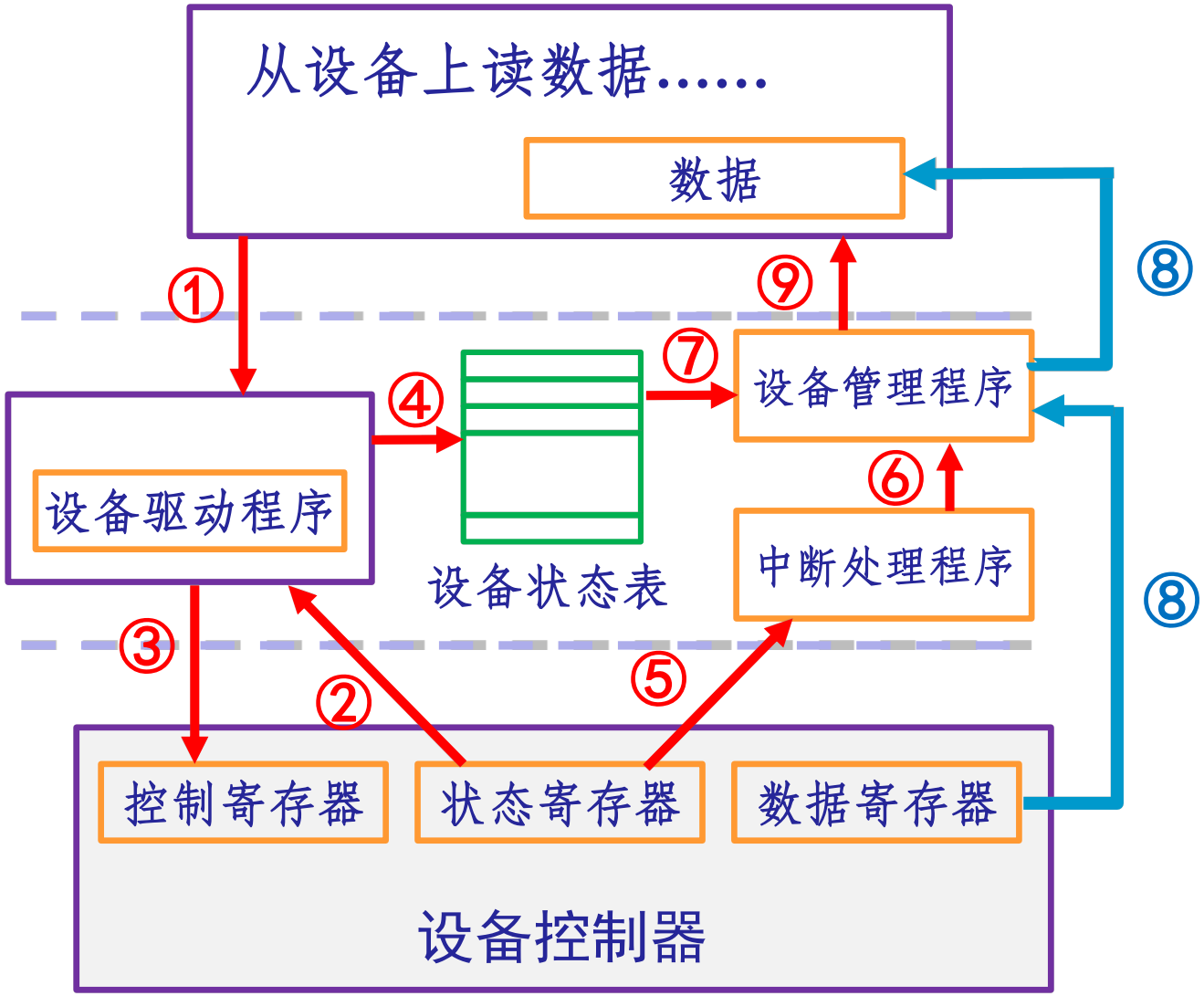
⑤ 设备控制器完成操作，把数据送给应用程序

⑥ 应用程序继续进行相应的处理。

轮询流程


外围设备
接收到start命令
做接收或发送数据准备
否
准备完毕？
是
标志触发器置“done”
等待CPU来的下条指令
(a)
CPU
发start命令
是
设备标志触发器为done”？
否
等 待
执行下条指令
开始数据传送
(b)

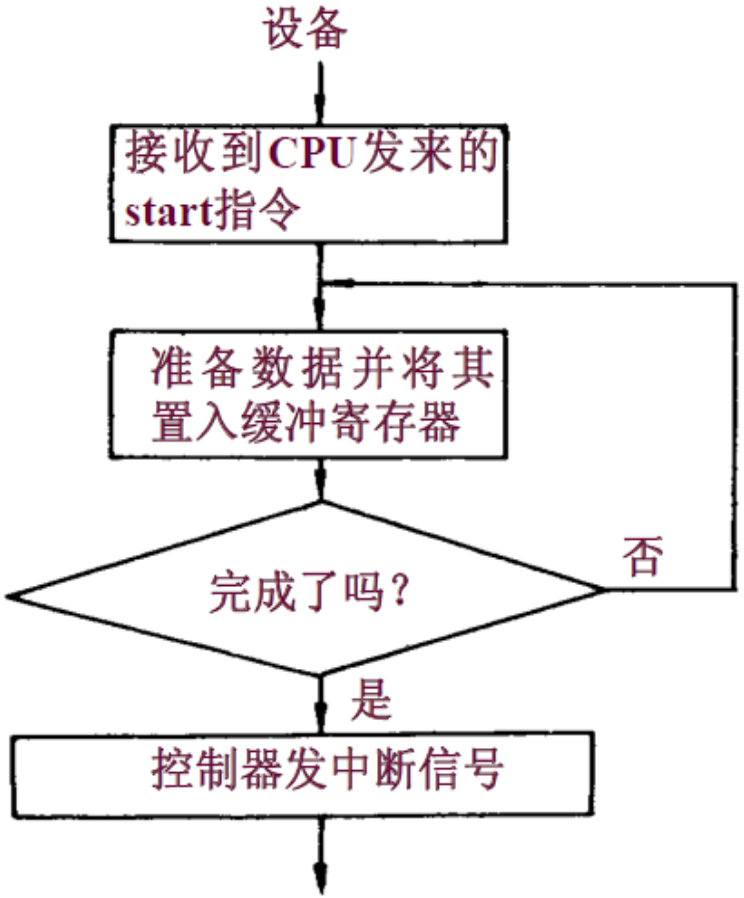
中断驱动方式

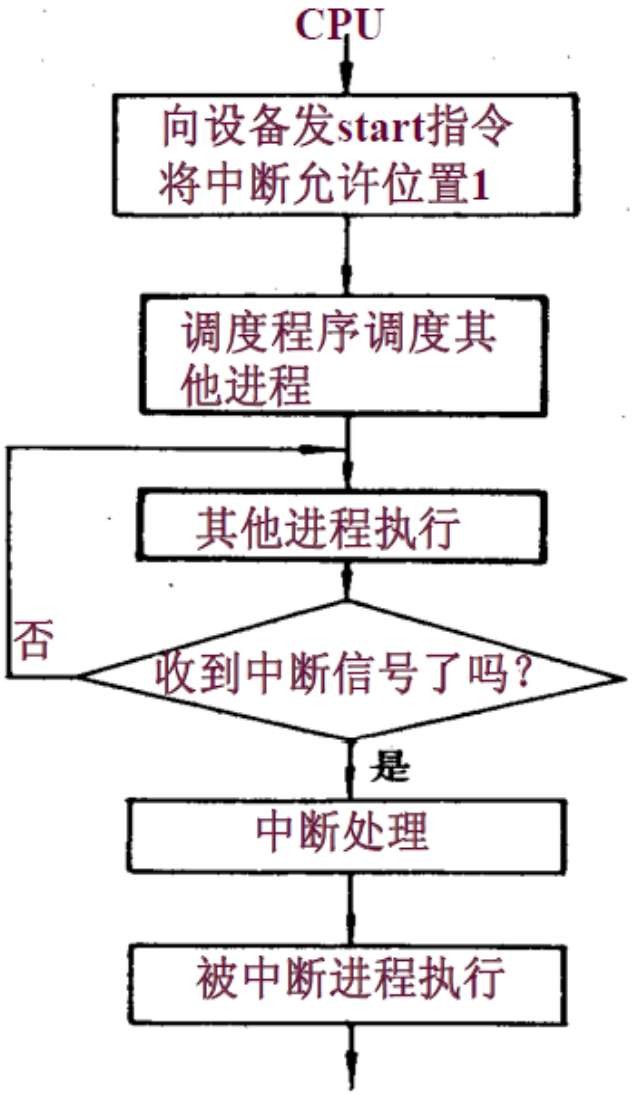

从设备上读数据……
数据
①
⑨
⑧
⑦
④
设备管理程序
设备驱动程序
⑥
中断处理程序
设备状态表
③
②
⑤
⑧
控制寄存器
状态寄存器
数据寄存器
设备控制器

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。

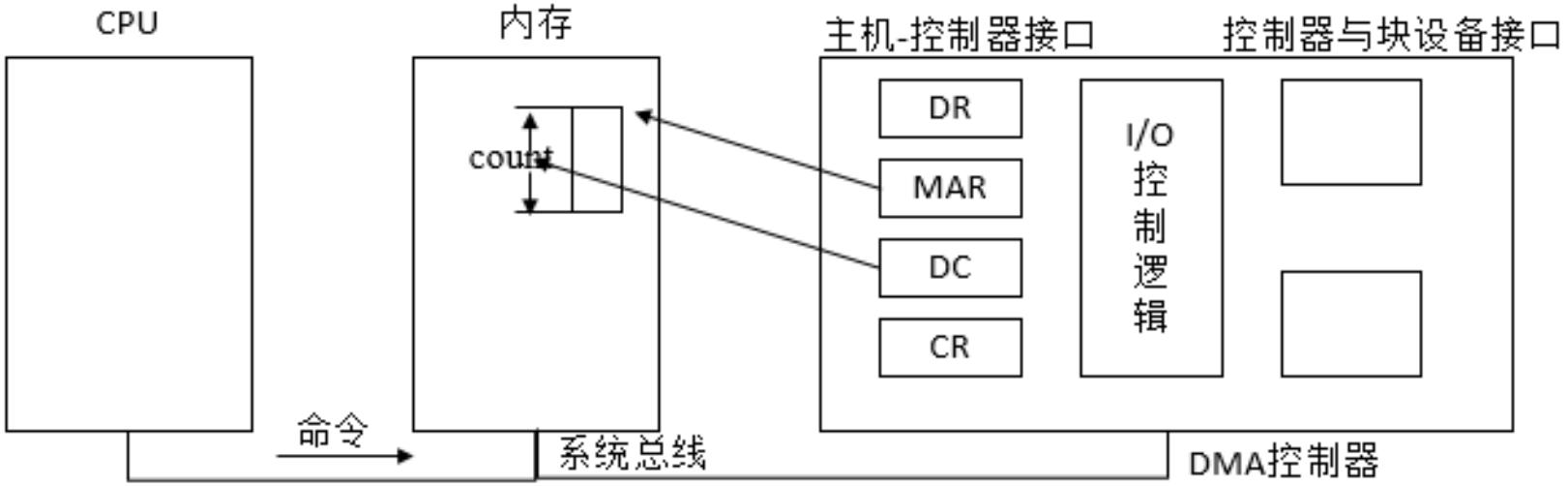
中断方式流程


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

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。



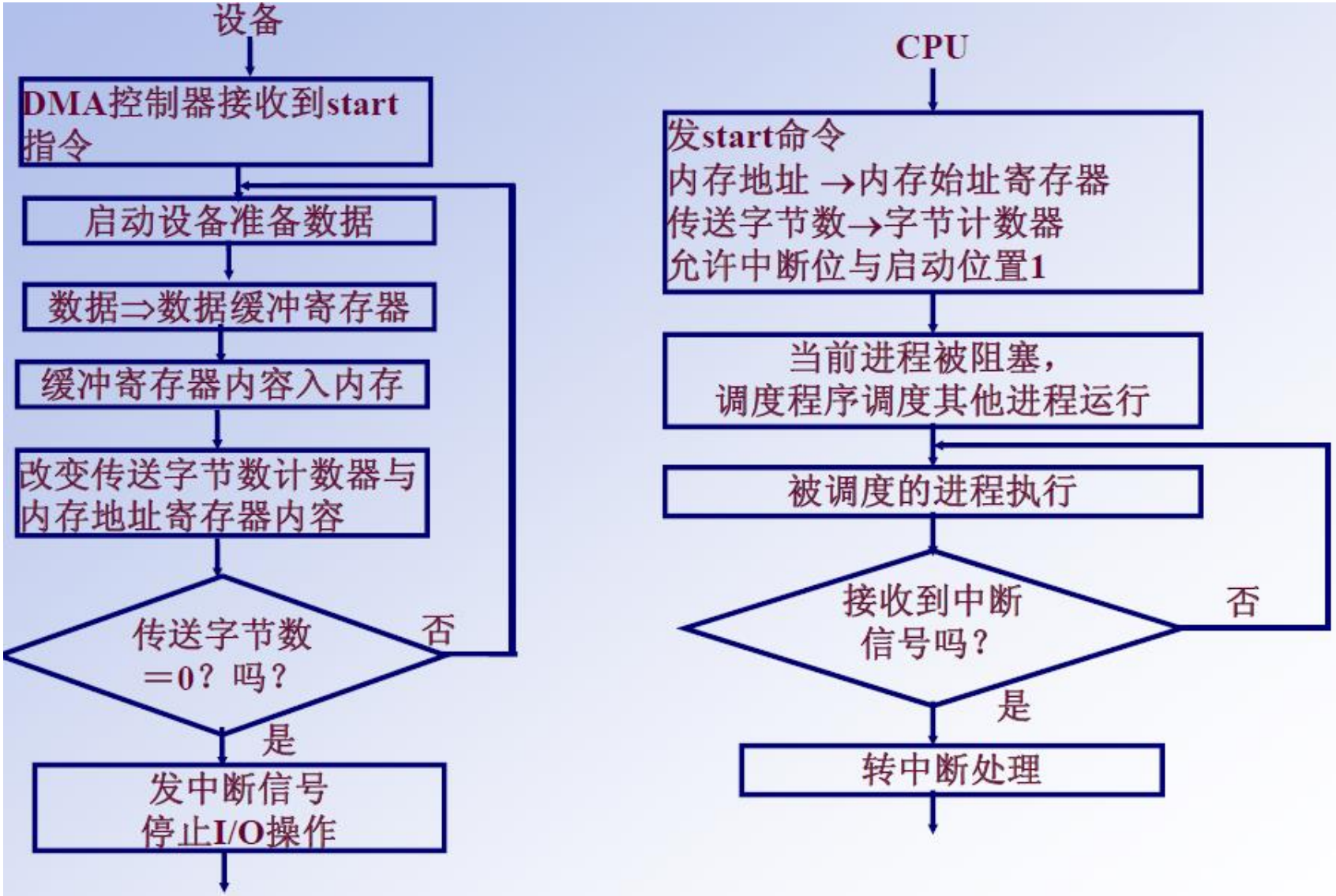
DMA控制器中的寄存器

- 命令/状态寄存器（CR），用于接收从CPU发送来的I/O命令，或有关控制信息，或设备的状态。
- 内存地址寄存器（MAR），在输入时，它存放把数据从设备传送到内存的起始目标地址，在输出时，它存放由内存到设备的内存源地址。
- 数据寄存器（DR），用于暂存从设备到内存，或从内存到设备的数据。
- 数据计数器（DC），存放本次CPU要读或写的字（节）数。

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


设备
DMA控制器接收到start指令
启动设备准备数据
数据⇒数据缓冲寄存器
缓冲寄存器内容入内存
改变传送字节数计数器与内存地址寄存器内容
传送字节数=0？吗？
否
是
发中断信号
停止I/O操作
CPU
发start命令
内存地址→内存始址寄存器
传送字节数→字节计数器
允许中断位与启动位置1
当前进程被阻塞，
调度程序调度其他进程运行
被调度的进程执行
接收到中断信号吗？
否
是
转中断处理

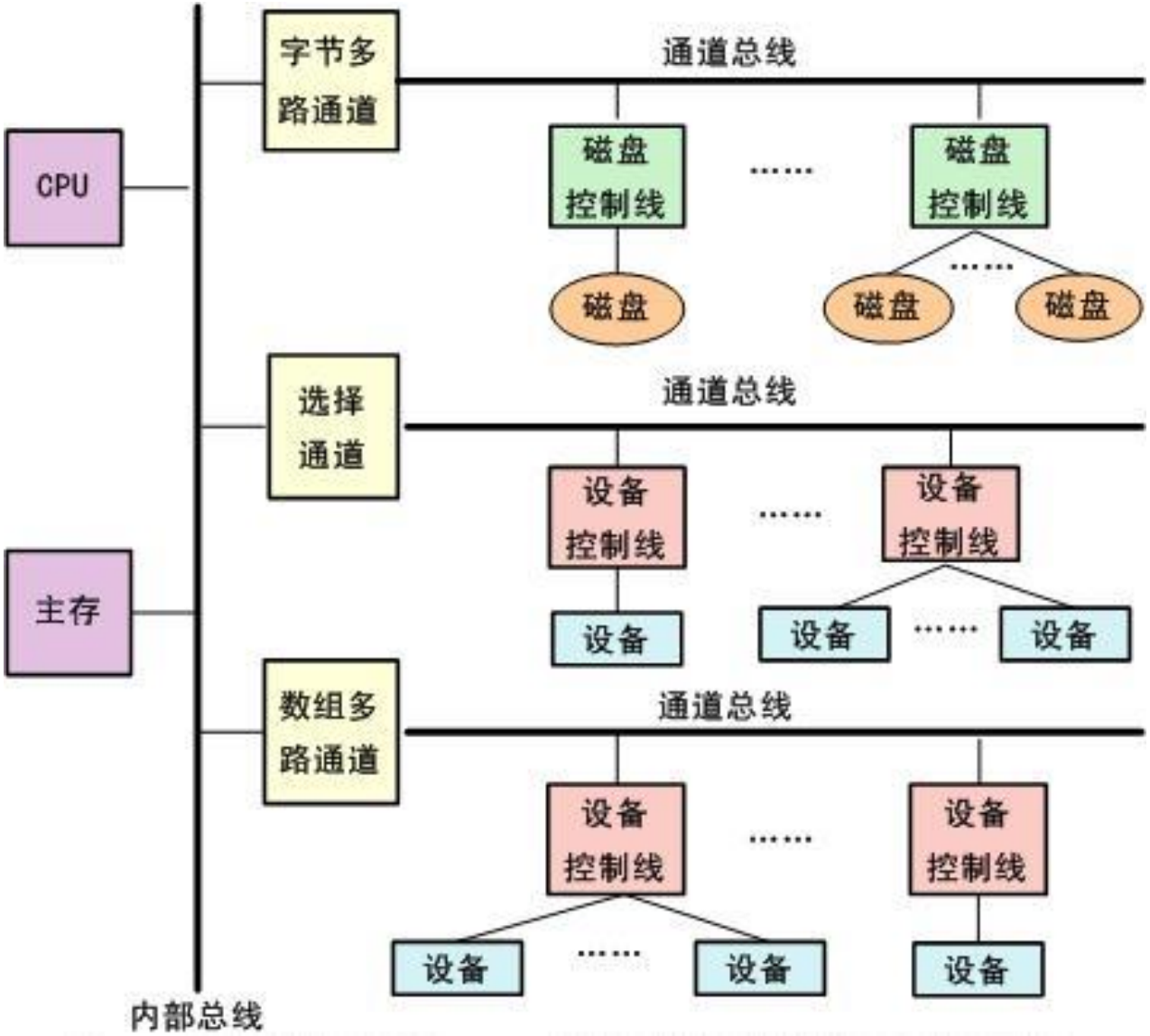
DMA与中断方式的区别

- 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。
- 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)

- 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。

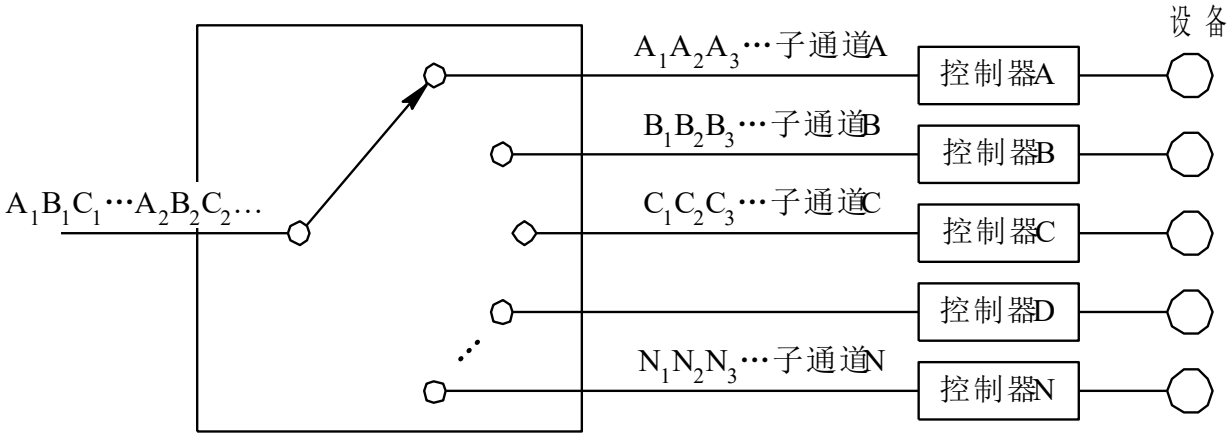
I/O通道分类

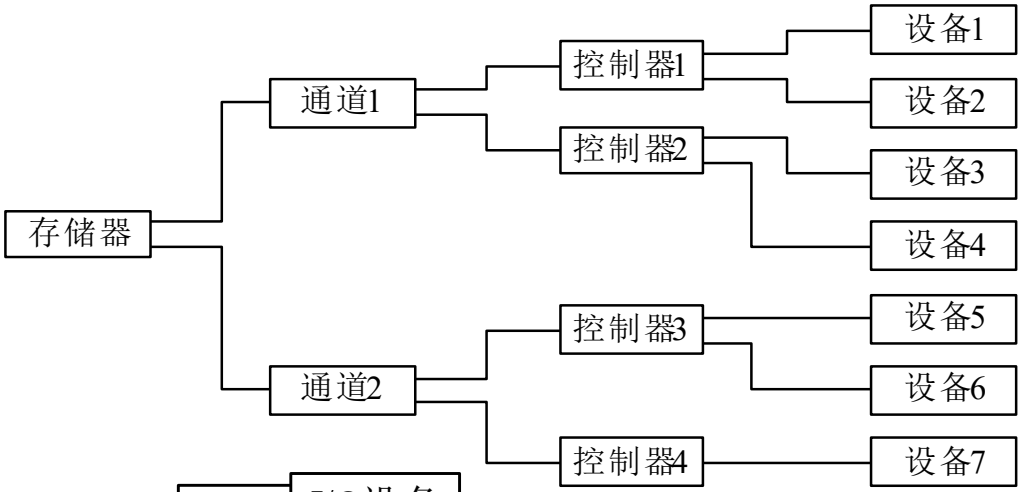

CPU
主存
内部总线
字节多路通道
选择通道
数组多路通道
通道总线
通道总线
通道总线
磁盘控制线
磁盘控制线
磁盘
磁盘
磁盘
设备控制线
设备控制线
设备
设备
设备
设备控制线
设备控制线
设备
设备
设备

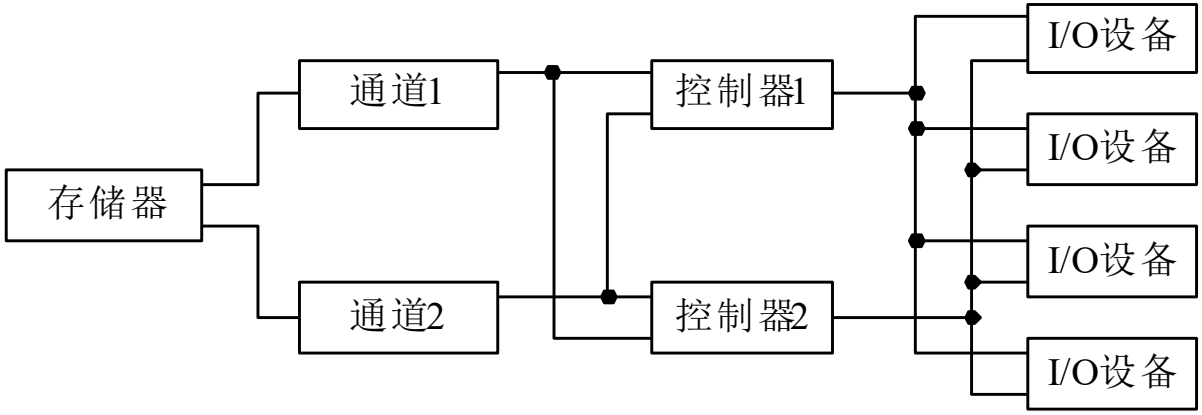
通道种类

- 字节多路通道
  - 以字节为单位交叉工作：当为一台设备传送一个字节后，立即转去为另一它设备传送一个字节；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


设 备
$A_1A_2A_3$…子通道A
控制器A
$B_1B_2B_3$…子通道B
控制器B
$A_1B_1C_1$…$A_2B_2C_2$…
$C_1C_2C_3$…子通道C
控制器C
控制器D
$N_1N_2N_3$…子通道N
控制器N


存储器
通道1
通道2
控制器1
控制器2
控制器3
控制器4
设备1
设备2
设备3
设备4
设备5
设备6
设备7


存储器
通道1
通道2
控制器1
控制器2
I/O设备
I/O设备
I/O设备
I/O设备

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

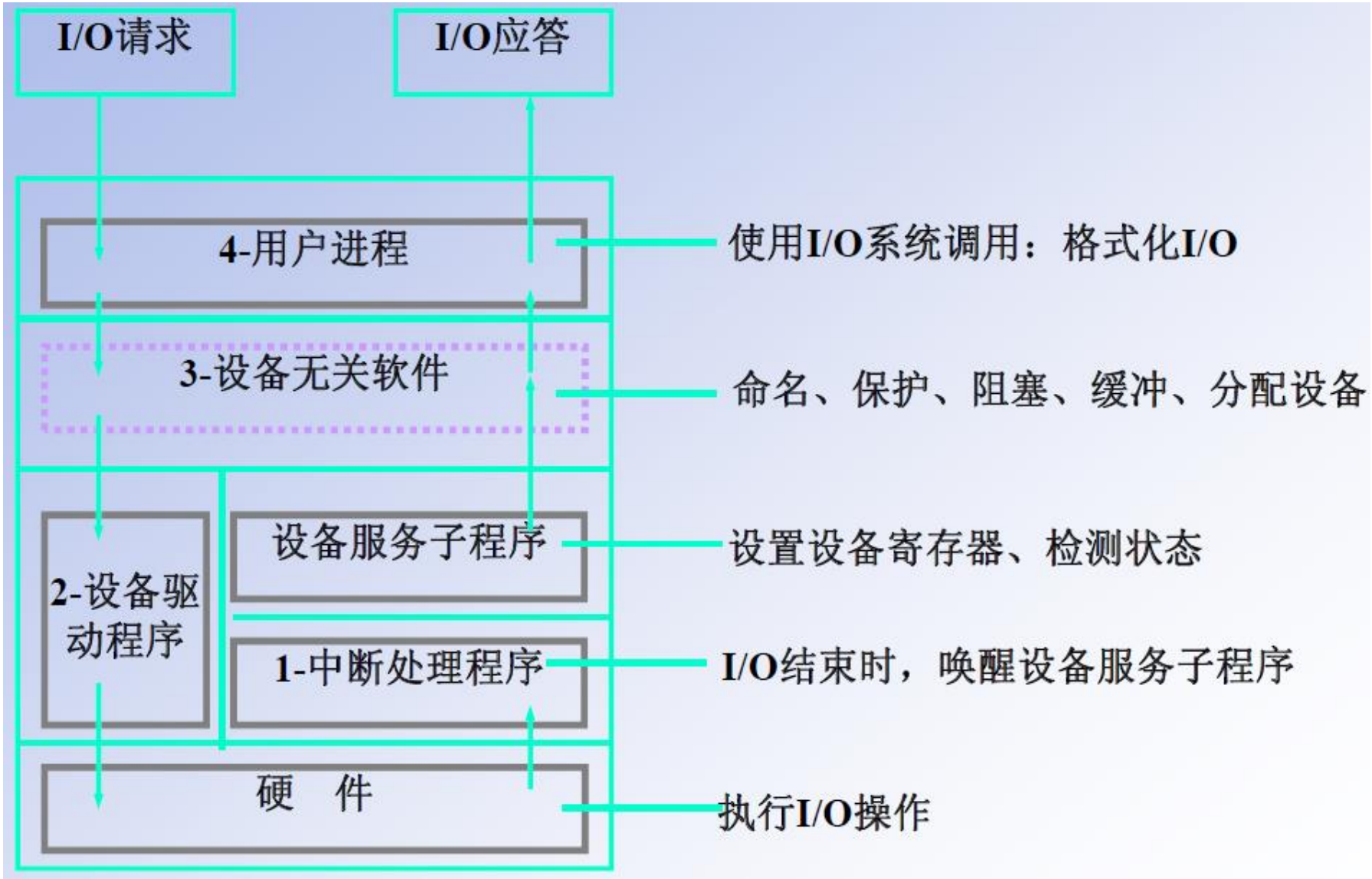
- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

I/O软件设计

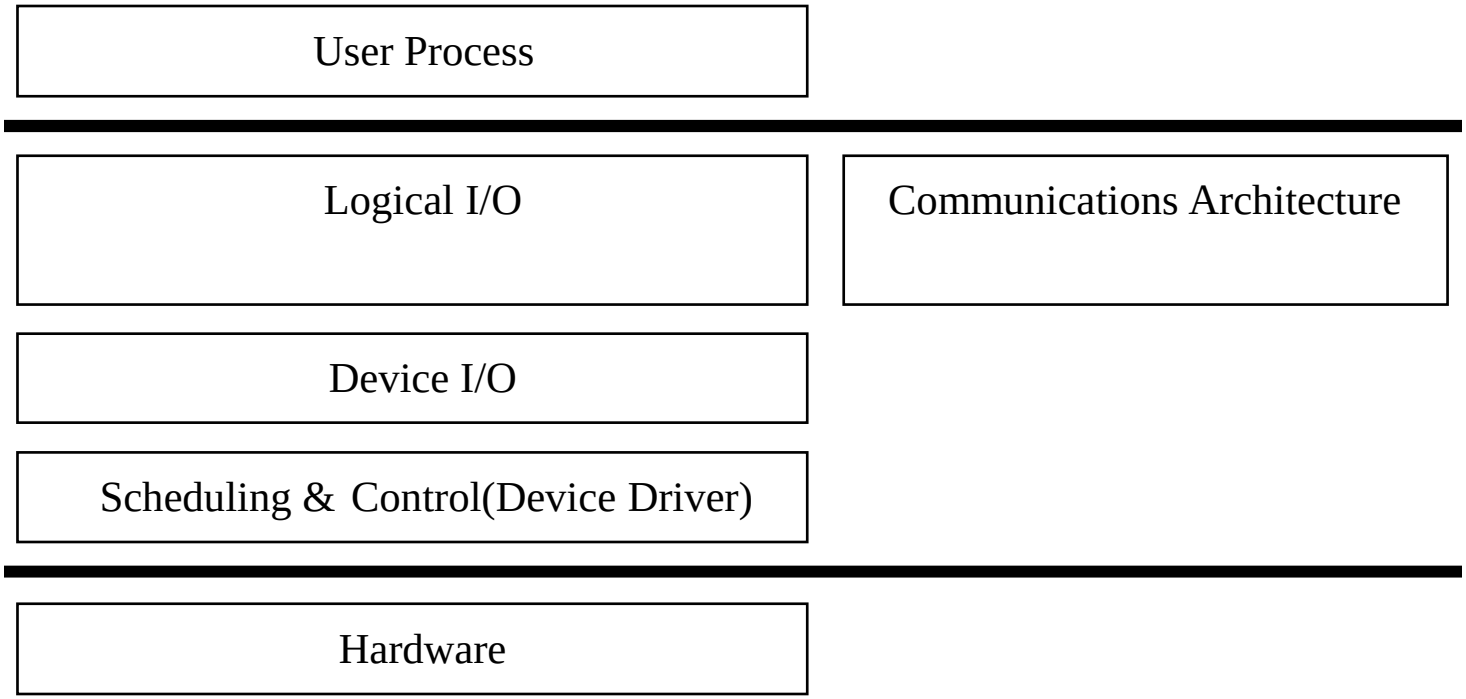
分层设计思想

- 把I/O软件组织称多个层次。
- 每一层都执行OS所需的功能的一个子集，它依赖于更低一层所执行的更原始的功能，从而隐藏这些功能的细节；同时又给高一层提供服务。
- 较低层更多的考虑硬件的特性，并向较高层软件提供接口。
- 较高层不依赖于硬件，并向用户提供一个友好的、清晰的、简单的、功能更强的接口。

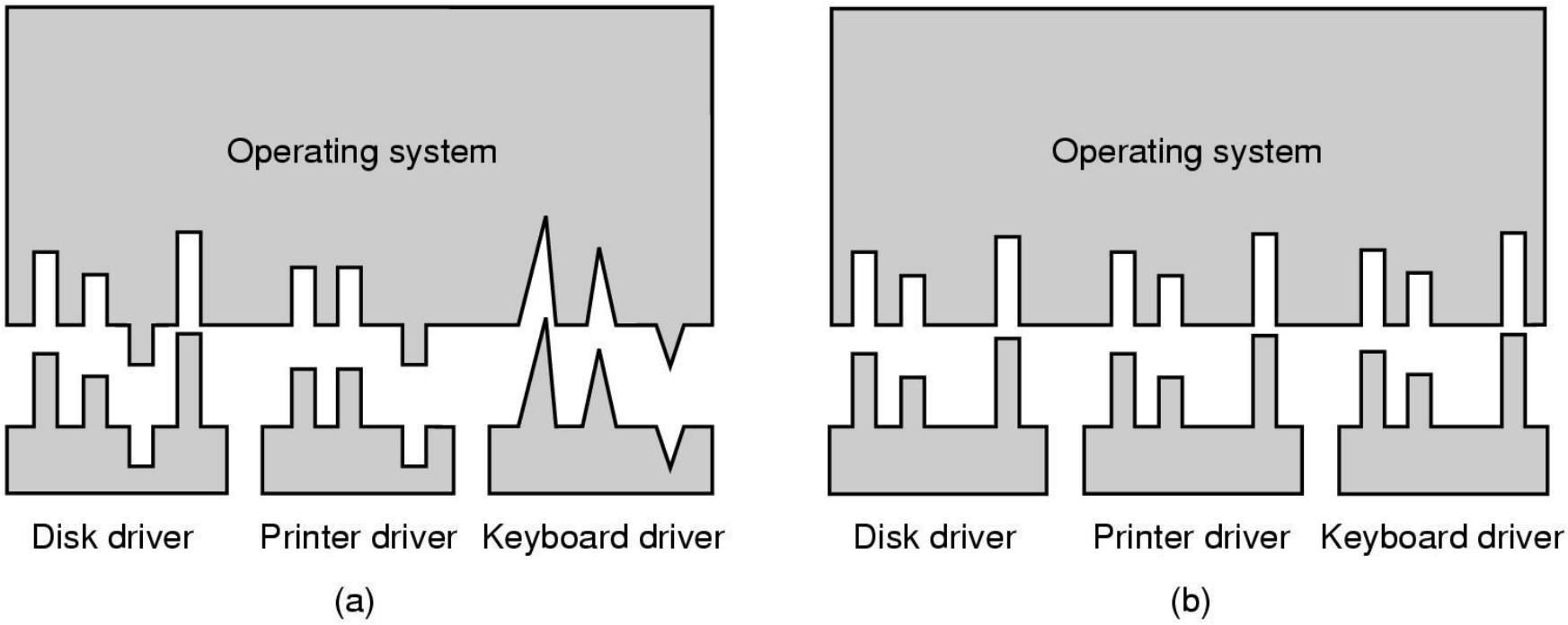
I/O相关软件的层次关系



I/O的层次


User Process
Logical I/O
Communications Architecture
Device I/O
Scheduling & Control(Device Driver)
Hardware

标准接口


Operating system
Disk driver
Printer driver
Keyboard driver
(a)
Operating system
Disk driver
Printer driver
Keyboard driver
(b)

设备独立性

- 为了实现设备独立性而引入了逻辑设备和物理设备这两个概念，在应用程序中，使用逻辑设备名称来请求使用某类设备，而系统在实际执行时，还必须使用物理设备名称。
- 系统需具有将逻辑设备名称转换为某物理设备名称的功能，这非常类似于存储器管理所介绍的逻辑地址和物理地址的概念，在应用程序中使用的是逻辑地址，系统在分配和使用内存时，必须使用物理地址。

设备独立性

- 在实现了设备独立性功能后，可带来以下好处：
  - 设备分配时的灵活性，当进程以物理名称来请求使用指定设备时，如果该设备已经分配给其他进程或正在检修，而此时尽管还有几台其他的相同设备正在空闲，该进程仍然阻塞。但若进程能够以逻辑设备名称来请求某类设备时，系统可立即将该类设备中的任一台分配给进程，仅当所有此类设备全部分配完毕时，进程才会阻塞。
  - 易于实现I/O重定向，用于I/O操作的设备可以更换，而不必改变应用程序。

逻辑设备名到物理设备名的映射

- **逻辑设备表：**为了实现设备的独立性，系统必须设置一张逻辑设备表LUT(Logical Unit Table)，用于将应用程序中所使用的逻辑设备名映射为物理设备名。
- 该表的每个表目中包含了三项，逻辑设备名、物理设备名、设备驱动程序的入口地址。通过逻辑设备名，系统可以查找LUT，便可找到物理设备和驱动程序。

逻辑设备名	物理设备名	驱动程序入口地址
/dev/tty	3	1024
/dev/printer	5	2046

逻辑设备名到物理设备名的映射

- **LUT的设置**可以采用两种方式：
  - 在整个系统中设置一张LUT（由于系统中所有进程的设备分配情况都记录在同一张LUT中，因而不允许在LUT中具有相同的逻辑设备名，这就要求所有用户都不能使用相同的逻辑设备名，多用户下难以做到，单用户很好实现）。
  - 为每个用户设置一张LUT（每当用户登录时，便为该用户建立一个进程，同时建立一张LUT，并将该表放入进程的PCB中）。

逻辑设备名	系统设备表指针
/dev/tty	3
/dev/printer	5

设备驱动程序

- 与设备密切相关的代码放在设备驱动程序中，每个设备驱动程序处理一种设备类型。
- 设备驱动程序的任务是接收来自与设备无关的上层软件的抽象请求，并执行这个请求。
- 每一个控制器都设有一个或多个设备寄存器，用来存放向设备发送的命令和参数。设备驱动程序负责释放这些命令，并监督它们正确执行。

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

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。在初始化时被调用一次。
- I/O操作子程序：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。
- 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；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

驱动程序的接口

- 驱动程序初始化函数
  - 向操作系统登记该驱动程序的接口函数，该初始化函数在系统启动时安装入内核执行。
- 驱动程序卸载函数，申请设备函数，释放设备函数
- I/O操作函数
  - 对独占设备，包含启动I/O的命令；对共享设备，将I/O请求挂入请求队列。
- 中断处理函数
  - I/O完成做善后处理，一般是唤醒等待刚完成I/O求的阻塞进程，时期能进一步做后续工作；如果存在I/O请求队列，则启动下一个I/O请求。

I/O请求的处理过程

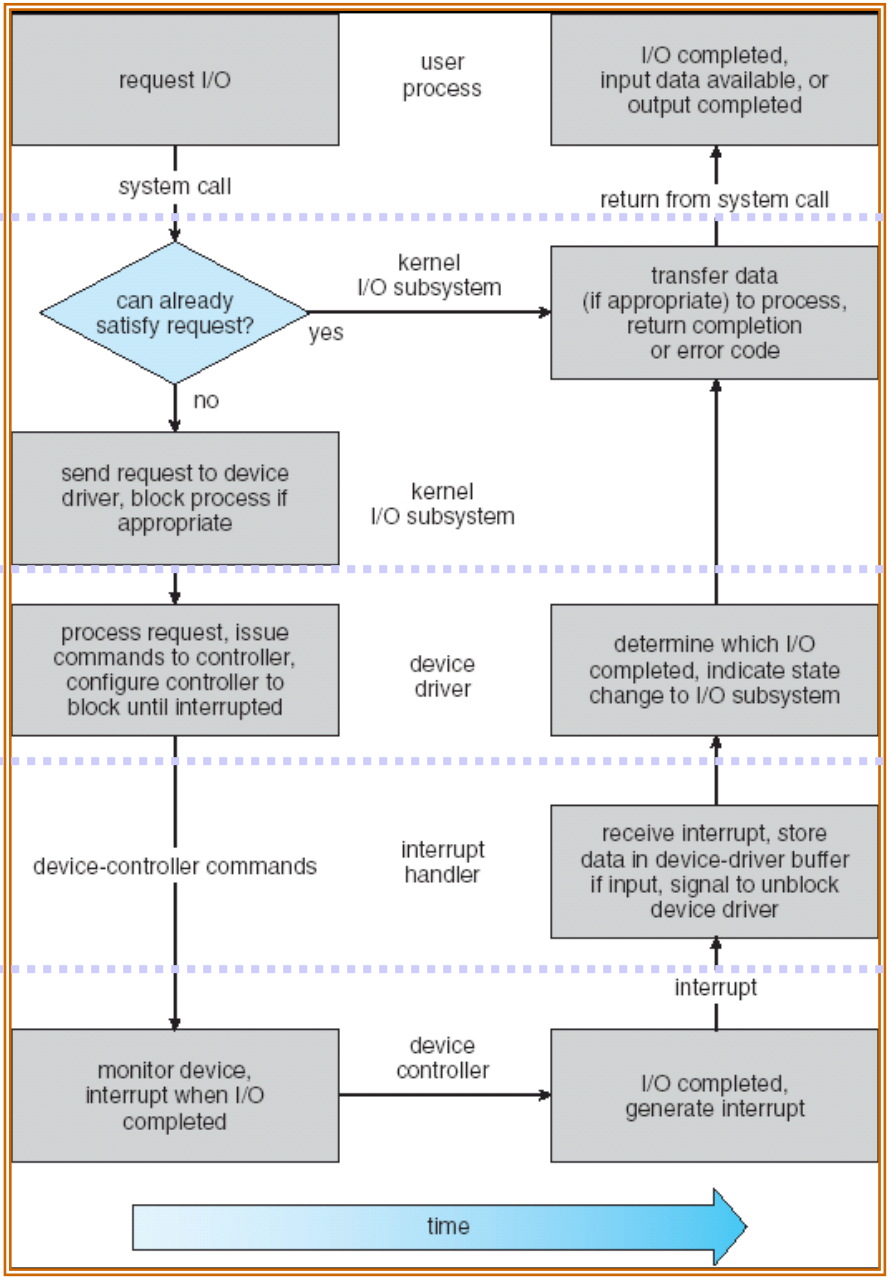
用户程序

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



驱动程序与应用程序的区别

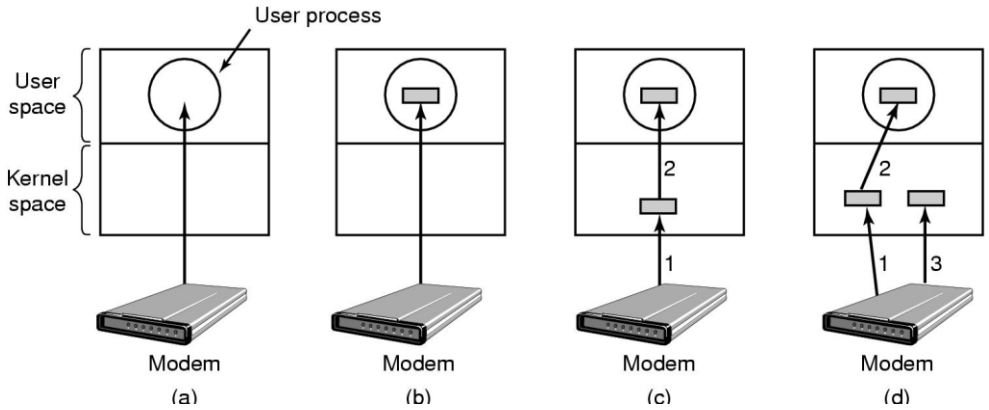
- 应用程序以main开始
- 驱动程序没有main，它以一个模块初始化函数作为入口
- 应用程序从头到尾执行一个任务
- 驱动程序完成初始化之后不再运行，等待系统调用
- 应用程序可以使用glibc等标准C函数库
- 驱动程序不能使用标准C库

内容提要

- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

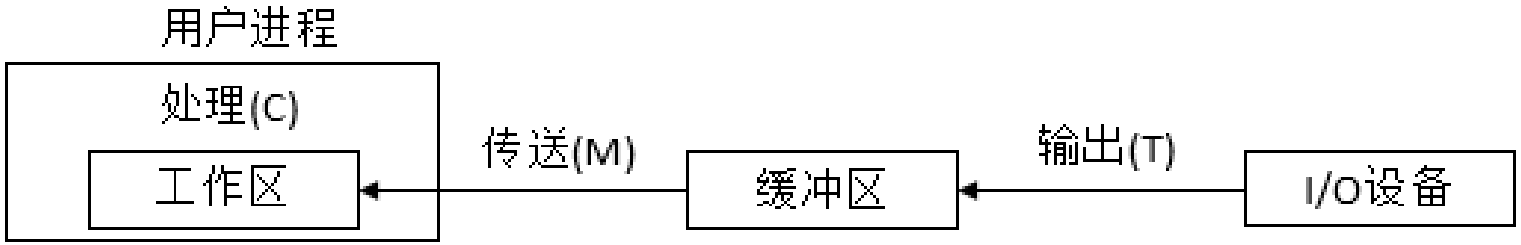
缓冲技术

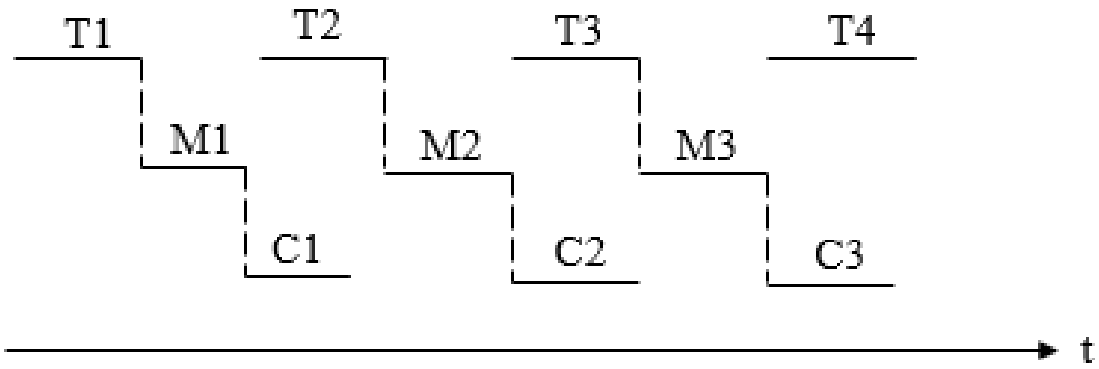
- 缓冲技术可提高外设利用率。
- 原因
  - 匹配CPU与外设的不同处理速度
  - 减少对CPU的中断次数。
  - 提高CPU和I/O设备之间的并行性。



单缓冲(single buffer)

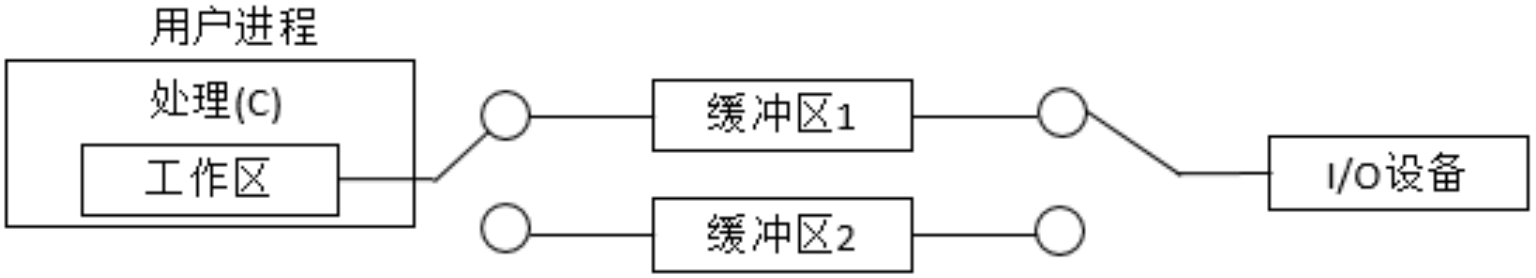
- 每当用户进程发出一个I/O请求时，操作系统便在主存中为之分配一个缓冲区，T、M，C的定义见下图。由于T和C是可以并行的，当T>C时，系统对每一块数据的处理时间为M+T，反之，为M+C，系统对每一块数据的处理时间为Max(C，T) + M。

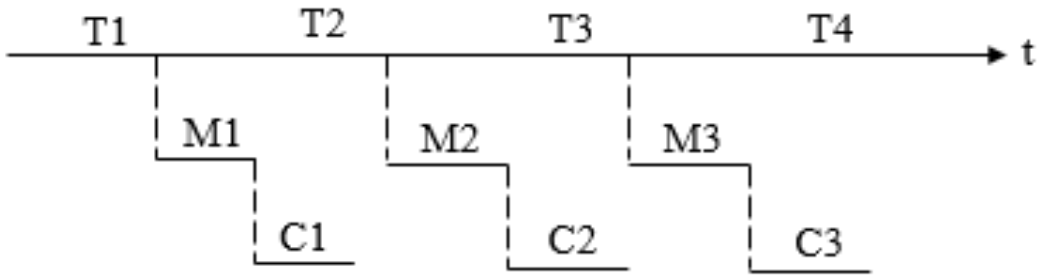




双缓冲(double buffer)

- 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。在双缓冲时，系统处理一块数据的时间可以粗略地认为是Max(C+M, T)，如果C<T，可使块设备连续输入，如果C>T，则可使CPU不必等待设备输入。

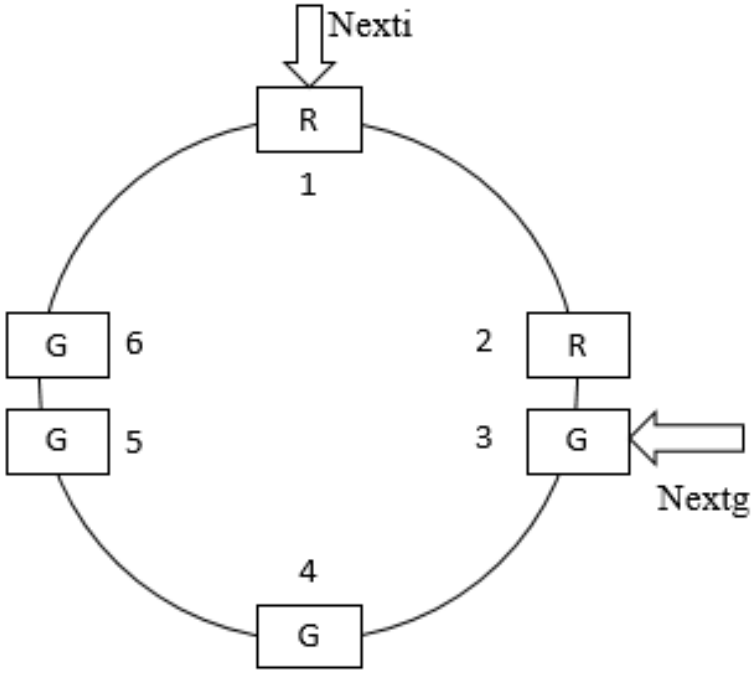


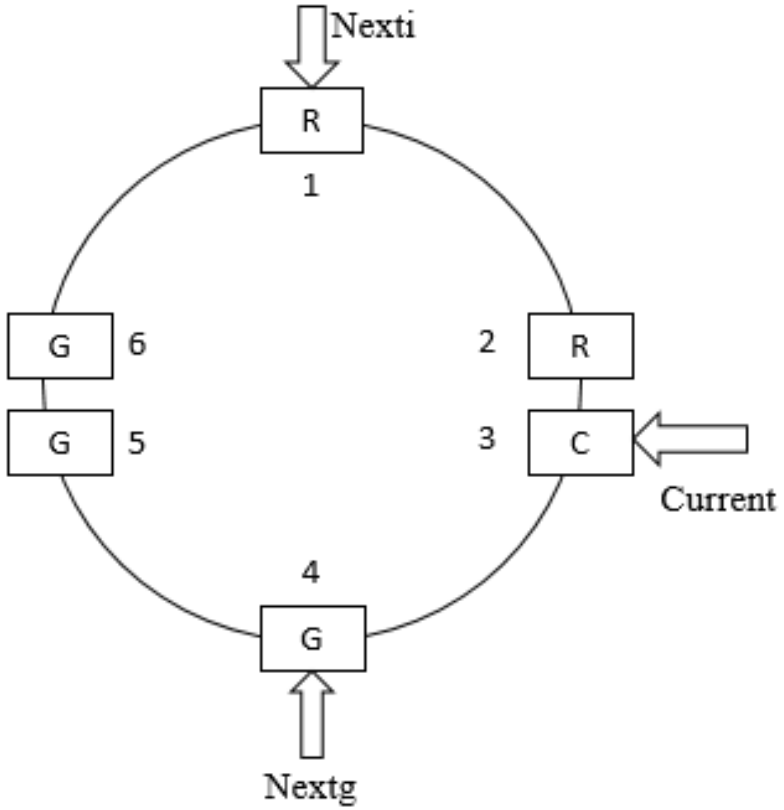


环形缓冲(circular buffer)

- 若CPU和外设的处理速度差较大，双缓冲效果不够理想，因此引入了多缓冲机制，可将多个缓冲组织成循环缓冲形式。对于用作输入的循环缓冲，通常是提供给输入进程或计算进程使用，输入进程不断向空缓冲去输入数据，而计算进程则从中提取数据进行计算。
- 循环缓冲区的组成如下
  - 多个缓冲区，在循环缓冲区中包括多个缓冲区，每个缓冲区的大小相同，作为输入的多缓冲区可分为三种类型，用于装输入数据的空缓冲区R、已装满数据的缓冲区G以及计算进程正在使用的工作缓冲区C。
  - 多个指针，作为输入的缓冲区可设置三个指针，用于指示计算进程下一个可用缓冲区G的指针Nextg、指示输入进程下次可用的空缓冲区R的指针Nexti、以及用于指示计算进程正在使用的缓冲区C的指针Current。

环形缓冲(circular buffer)

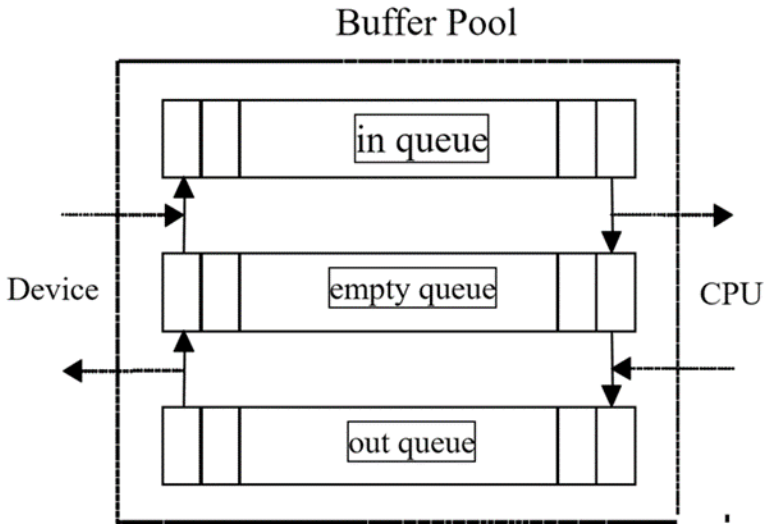




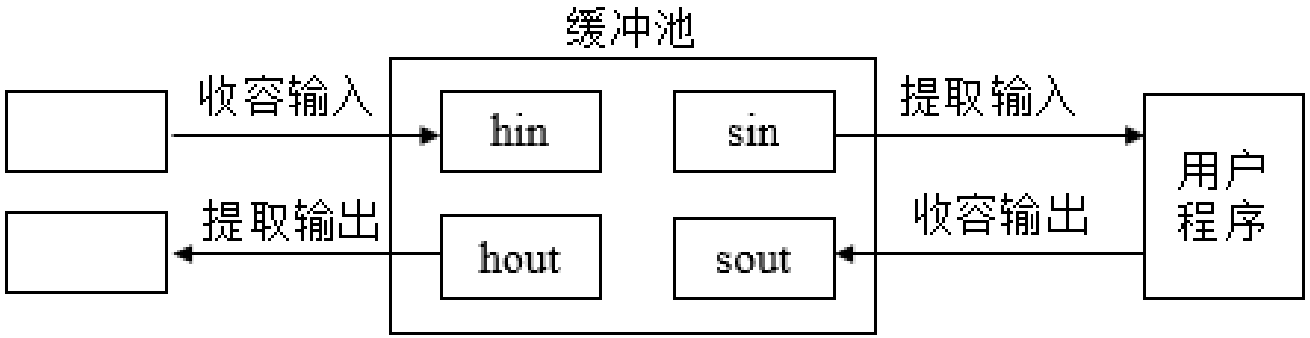
缓冲池(buffer pool)

- 上述的缓冲区仅适用于某特定的I/O进程和计算进程，因而它们属于专用缓冲。
- 当系统较大时，将会有许多这样的循环缓冲，这样会消耗大量的内存空间，而且利用率不高，为了提高缓冲区的利用率，引入缓冲池，在池中设置了多个可供若干个进程共享的缓冲区。
- 对于既可以用于输出的共用缓冲池，其中至少包含有一下三种类型的缓冲区。
  - 空（闲）缓冲区。
  - 装满输入数据的缓冲区。
  - 装满输出数据的缓冲区。

缓冲池



- 为了管理方便，将相同类型的缓冲区链成一个队列，形成了：空缓冲队列emq，输入队列inq，输出队列outq。
- 缓冲区可以工作在收容输入、提取输入、收容输出、提取输出四种工作方式下。



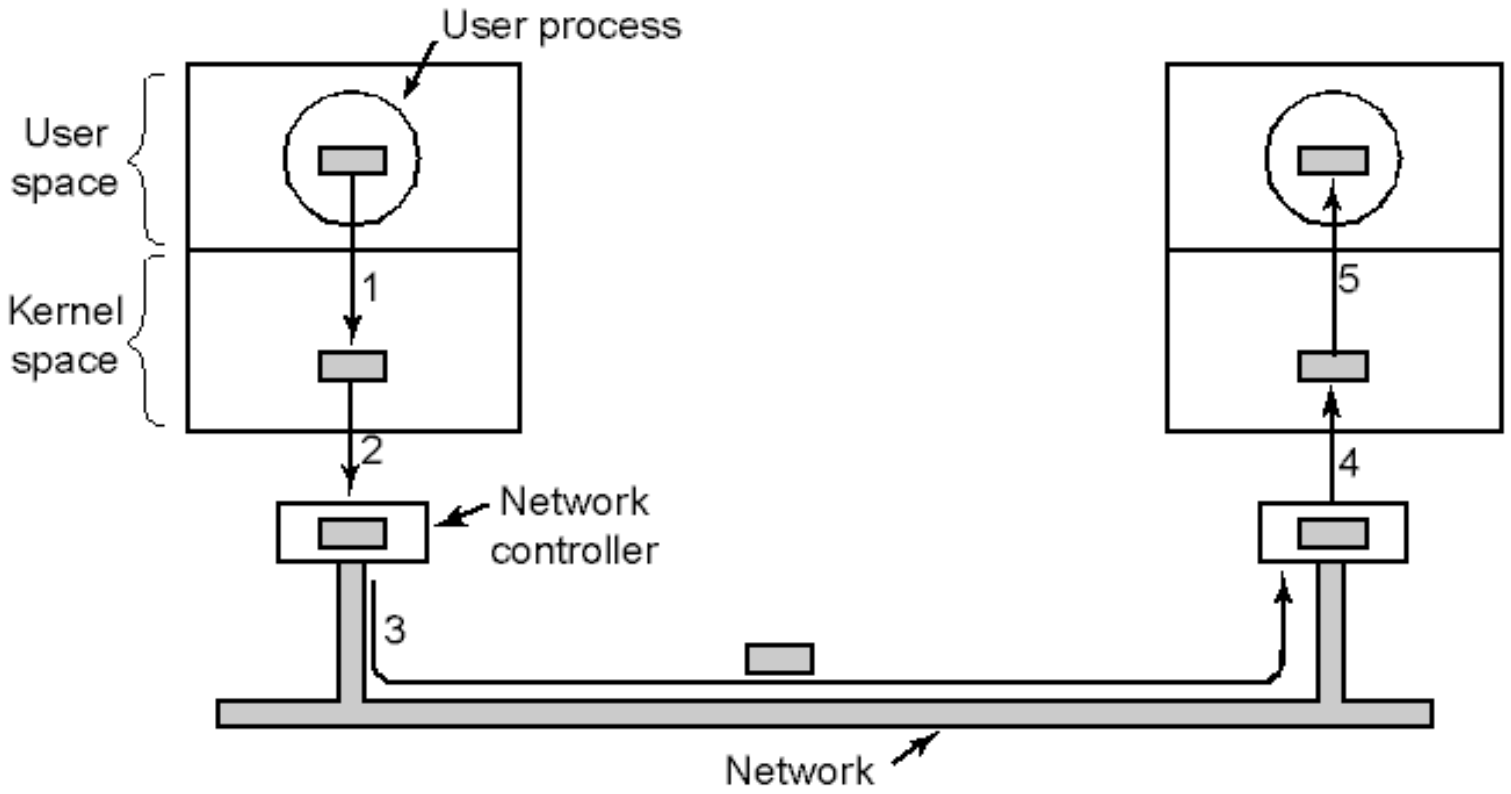
工作方式

- **收容输入**：在输入进程需要输入数据时，便调用Getbuf(emp)过程，从空缓冲队列的队首取出一个空缓冲区，把它作为收容输入工作缓冲hin，然后，把数据输入其中，装满后再调用Putbuf(inq, hin)过程，将该缓冲区挂在输入队列上。
- **提取输入**：当计算进程需要输入数据时，调用Getbuf(inq)过程，从输入队列队首取出一个缓冲区，作为提取输入工作缓冲区sin，计算进程从中提取数据，计算进程用完该数据后，再调用Putbuf(emq, sin)过程，将该缓冲区挂到空缓冲队列emq上。

工作方式

- **收容输出**：当计算进程需要输出时，调用Getbuf(emq)过程从空缓冲区队列emq的队首取出一个空缓冲区，作为收容输出工作缓冲区hout，当其中装满输出数据后，又调用Putbuf(outq, hout)过程，将该缓冲区挂在outq末尾。
- **提取输出**：由输出进程调用Getbuf(outq)过程，从输出队列队首取出一个装满输出数据的缓冲区，作为提取输出工作缓冲区sout，在数据提取完后，再调用Putbuf(emq, sout)过程，将该缓冲区挂在空缓冲队列末尾。

缓冲的问题：多次复制一个数据包



内容提要

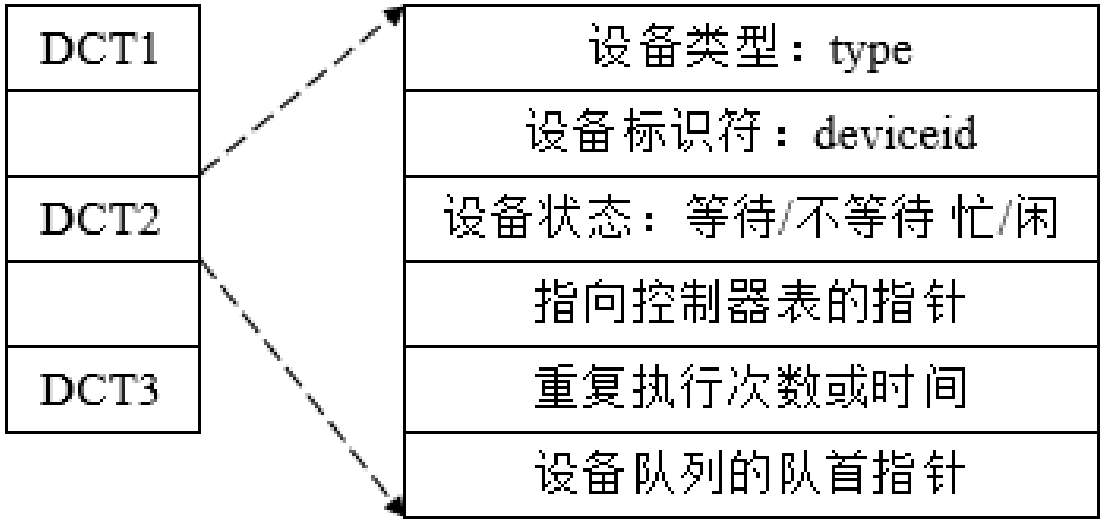
- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

I/O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。
- 有两种常见作法：
  - 在进程间切换使用外设，如键盘和鼠标；
  - 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

1. 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。



DCT说明

- 设备队列队首指针：凡因为请求本设备而未得到满足的进程，其PCB都应按照一定的策略排成一个队，称该队列为设备请求队列或简称设备队列，其队首指针指向队首PCB；
- 设备状态：当设备处于使用状态时，应该设备设置为忙/闲标志置为1；
- 控制器表指针：该指针指向该设备所连接的控制器的控制表；
- 重复执行次数：外部设备在传送数据时，较容易发生数据传送错误。在许多系统中，如果发生传送错误，并不立即认为传送失败，而是令它重传，并由系统规定设备在工作中发生错误时应重复执行的次数。

数据结构

2. 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。

3. 通道控制表(CHCT, CHannel Control Table)：每个通道一张，描述通道工作状态。

控制器标识符：controllerid
控制器状态：忙/闲
与控制器连接的通道表指针
控制器队列的队首指针
控制器队列的队尾指针

控制器表COCT

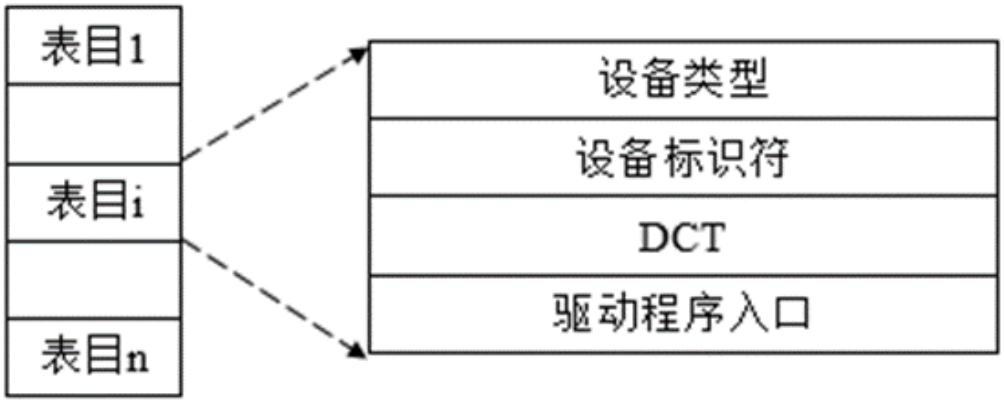
通道标识符：channelid
通道状态：忙/闲
与通道连接的控制器表首址
通道队列的队首指针
通道队列的队尾指针

通道表CHCT

数据结构

4. 系统设备表(SDT, System Device Table)：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;



系统设备表SDT

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。

单（多）通路I/O系统的设备分配

单（多）通路：一个设备对应一（多）个控制器，一个控制器对应一（多）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。

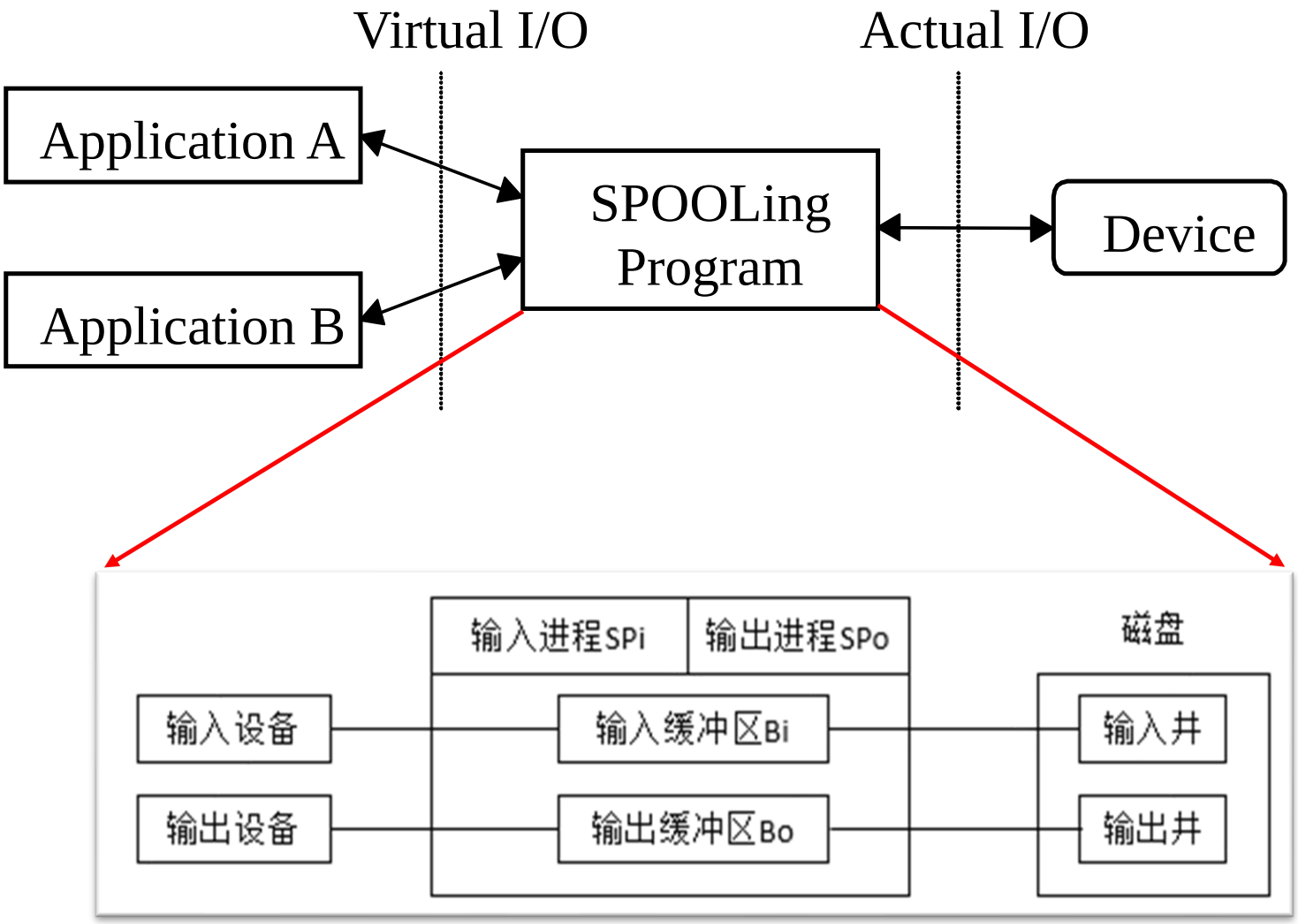
用户空间的I/O软件：SPOOLing技术

- 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）来完成对设备的I/O操作。无需使用外围I/O处理机。

SPOOLing 系统

- 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



SPOOLing 系统

- SPOOLing系统由三部分组成：
  - 输入井和输出井，这是在磁盘上开辟的两个大存储空间。输入井是模拟脱机输入时的磁盘设备，用于暂存I/O设备输入的数据，输出井是模拟脱机输出时的磁盘，用于暂存用户程序和输出数据。
  - 输入缓冲区和输出缓冲区，缓和CPU与磁盘之间速度不匹配，在内存中开辟的两个缓冲区，输入缓冲区用于暂存由输入设备送来的数据，以后再传送到输入井，输出缓冲区用于暂存从输出井送来的数据，以后再传送给输出设备。

SPOOLing 系统

- 输入进程SPi和输出进程SPo，利用两个进程来模拟脱机I/O时的外围控制机，其中，进程SPi模拟脱机输入时的外围控制机，将用户要求的数据从输入机通过输入缓冲区再送到输入井，当CPU需要输入数据时，直接从输入井读入内存；进程SPo模拟脱机输出时的外围控制机，把用户要求输出的数据先从内存送到输出井，待输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备上。

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。

内容提要

- I/O管理概述
- I/O硬件组成
- I/O控制方式
- I/O软件的组成
- I/O缓冲管理
- I/O设备管理
- I/O性能问题

I/O性能问题

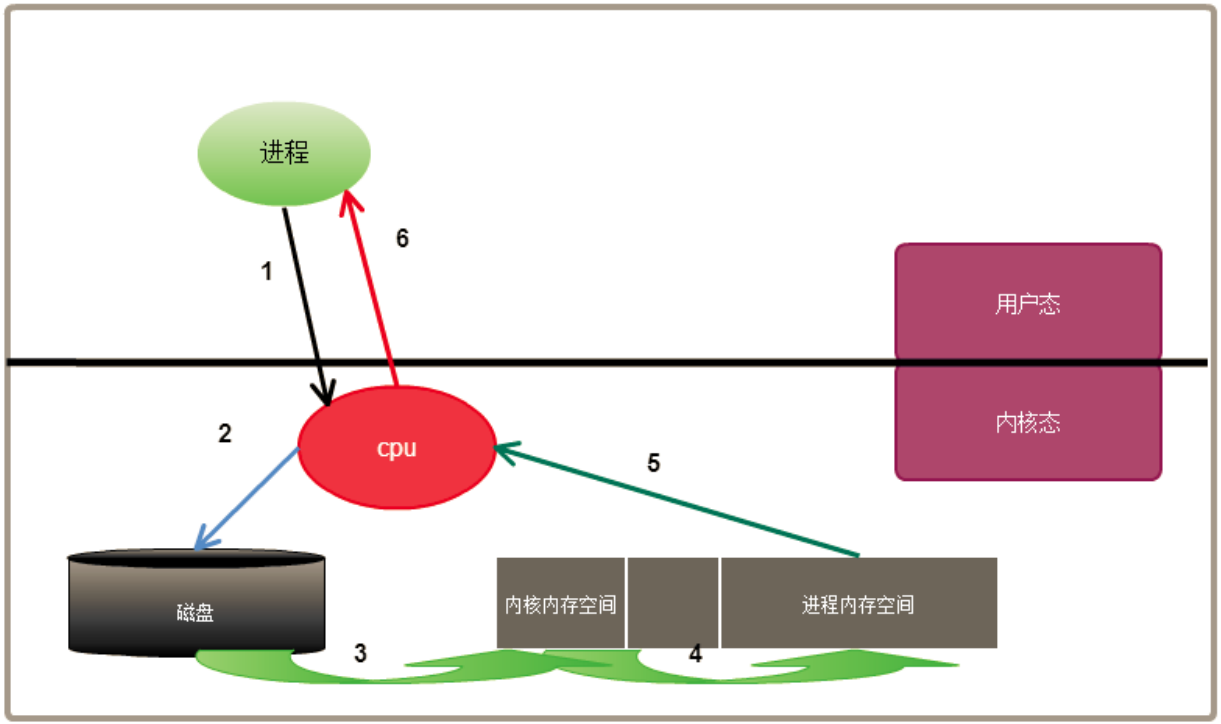
解决I/O性能问题的两个途径：

- **使CPU利用率尽可能不被I/O降低**
  - 可以使用缓冲技术减少或缓解速度差异，同时使用异步I/O使CPU不等待 I/O。
- **使CPU尽可能摆脱I/O**
  - 使用DMA、通道等I/O部件让CPU摆脱I/O操作的影响。

I/O操作

I/O操作分为俩个步骤：

- 磁盘把数据装载进内核的内存空间
- 内核的内存空间的数据copy到用户的内存空间中



I/O操作过程

- 从系统调用的开始到系统调用结束经过的步骤：

① 进程向内核发起一个系统调用；

② 内核接收到系统调用，知道是对文件的请求，于是告诉磁盘，把文件读取出来；

③ 磁盘接收到来着内核的命令后，把文件载入到内核的内存空间里面；

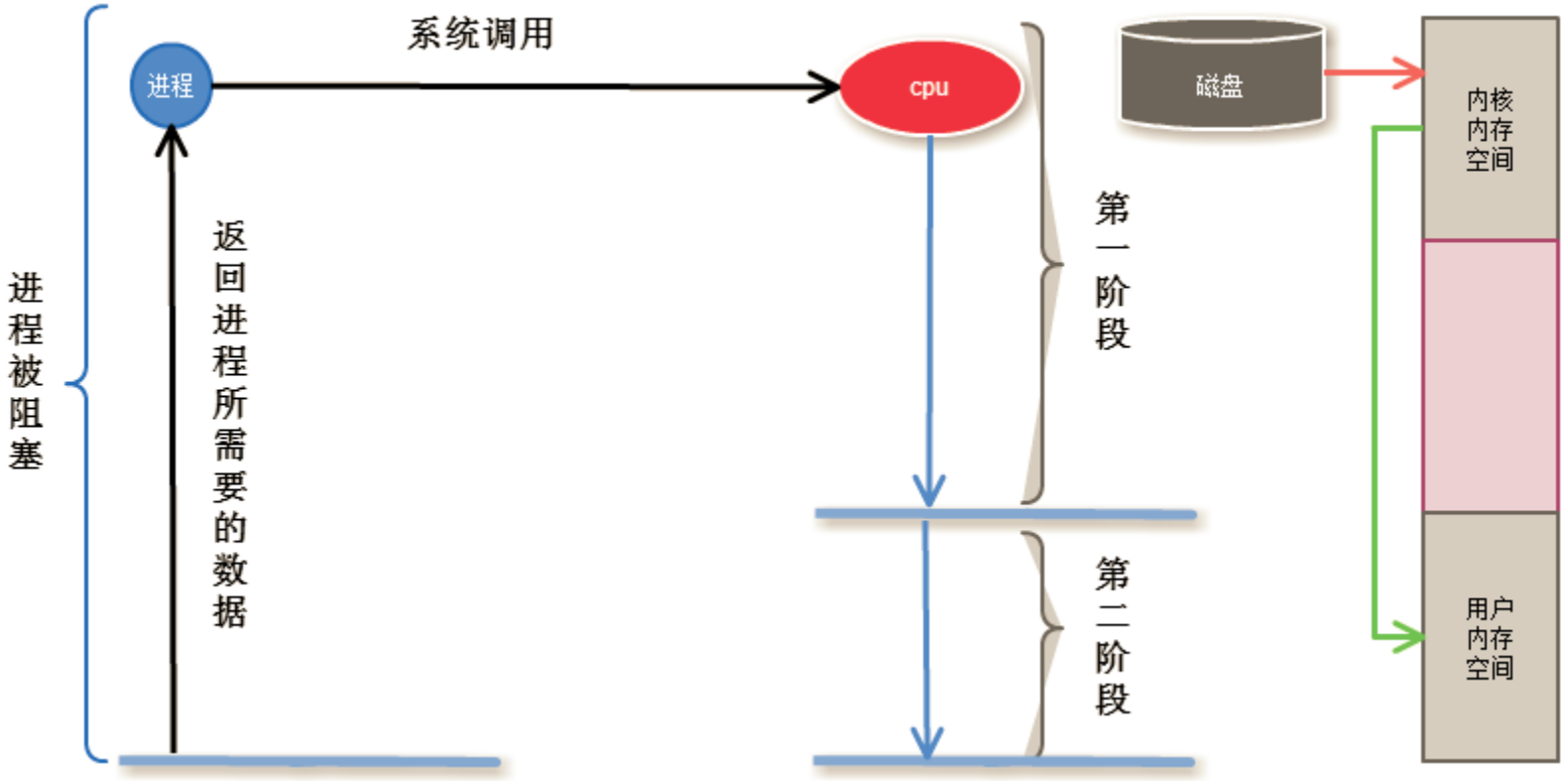
④ 内核的内存空间接收到数据之后，把数据copy到用户进程的内存空间(此过程是I/O发生的地方)；

⑤ 进程内存空间得到数据后，给内核发送通知；

⑥ 内核把接收到的通知回复给进程，此过程为唤醒进程，然后进程得到数据，进行下一步操作。

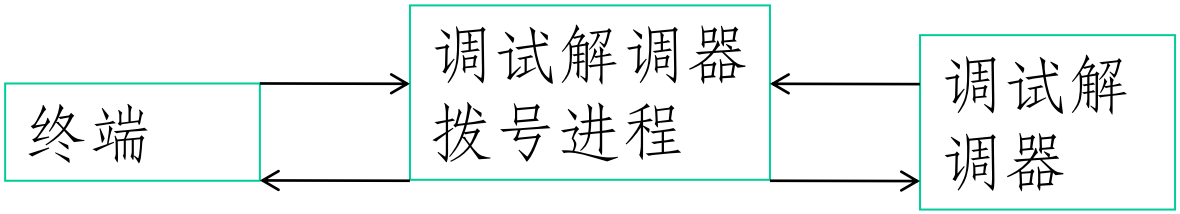
阻塞I/O

- 是指调用结果返回之前,当前线程会被挂起(线程进入睡眠状态) 函数只有在得到结果之后,才会返回,才能继续执行。



I/O多路复用

- while((n=read(STDIN_FILENO, buf, BUFSIZ)) > 0)
- if(write(STDOUT_FILENO, buf, n) != n)
- err_sys(“write error”);

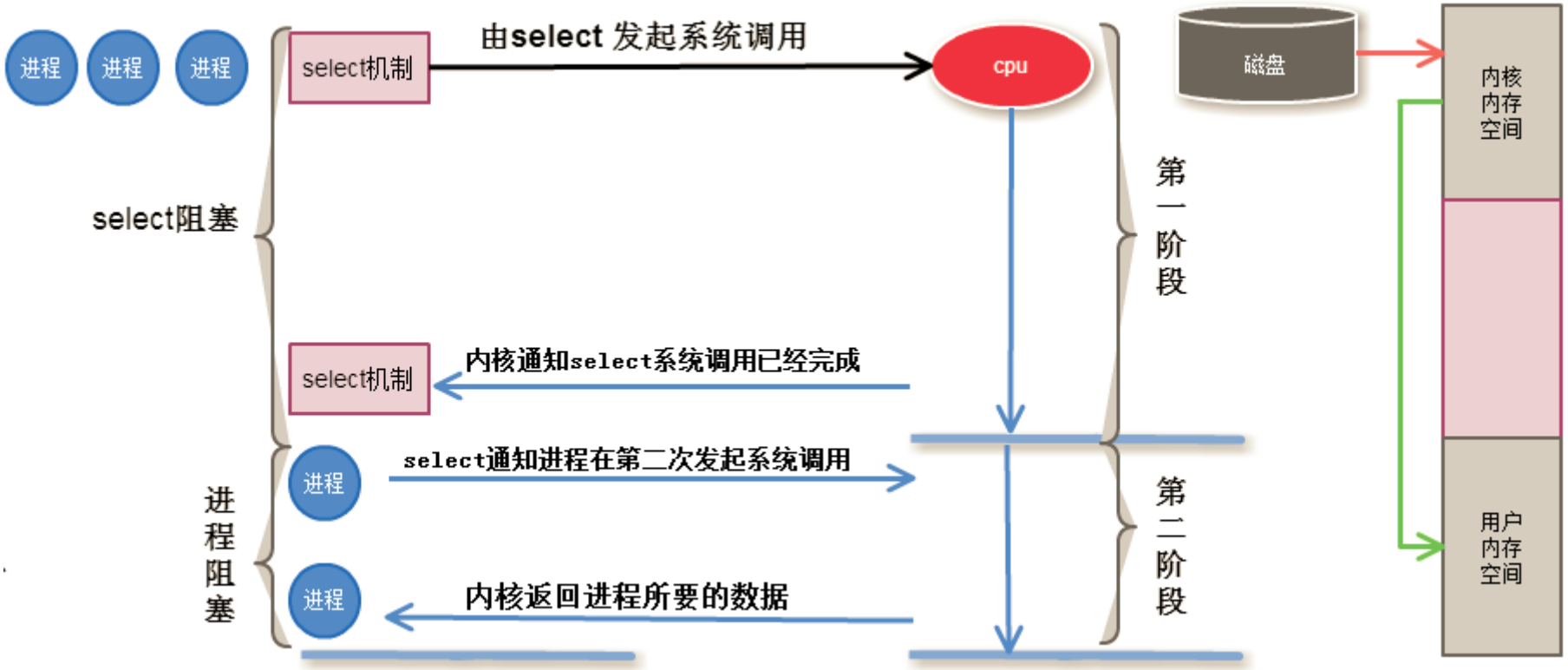


I/O多路复用

- I/O多路复用
  - 工作进程调用一个管理I/O的特殊库函数，此库函数可以接受并管理多个I/O请求，工作进程则可以同时等待多个I/O请求，虽然是阻塞，但是阻塞在多个进程上，可以提高效率。第二阶段依然需要工作进程参与库函数把内核空间数据复制到用户空间，第二阶段依旧阻塞。
- 为什么要用I/O多路复用
  - 某个进程阻塞多个IO上,一个进程即要等待从键盘输入信息,另一个准备从硬盘装入信息;
  - 比如通过read这样的命令,调用了一个IO操作,键盘IO阻塞着,磁盘IO完成了；这个进程也是不能响应,因为键盘IO还没有完成,还在阻塞着,这个进程还在睡眠状态,这个时候怎么办?

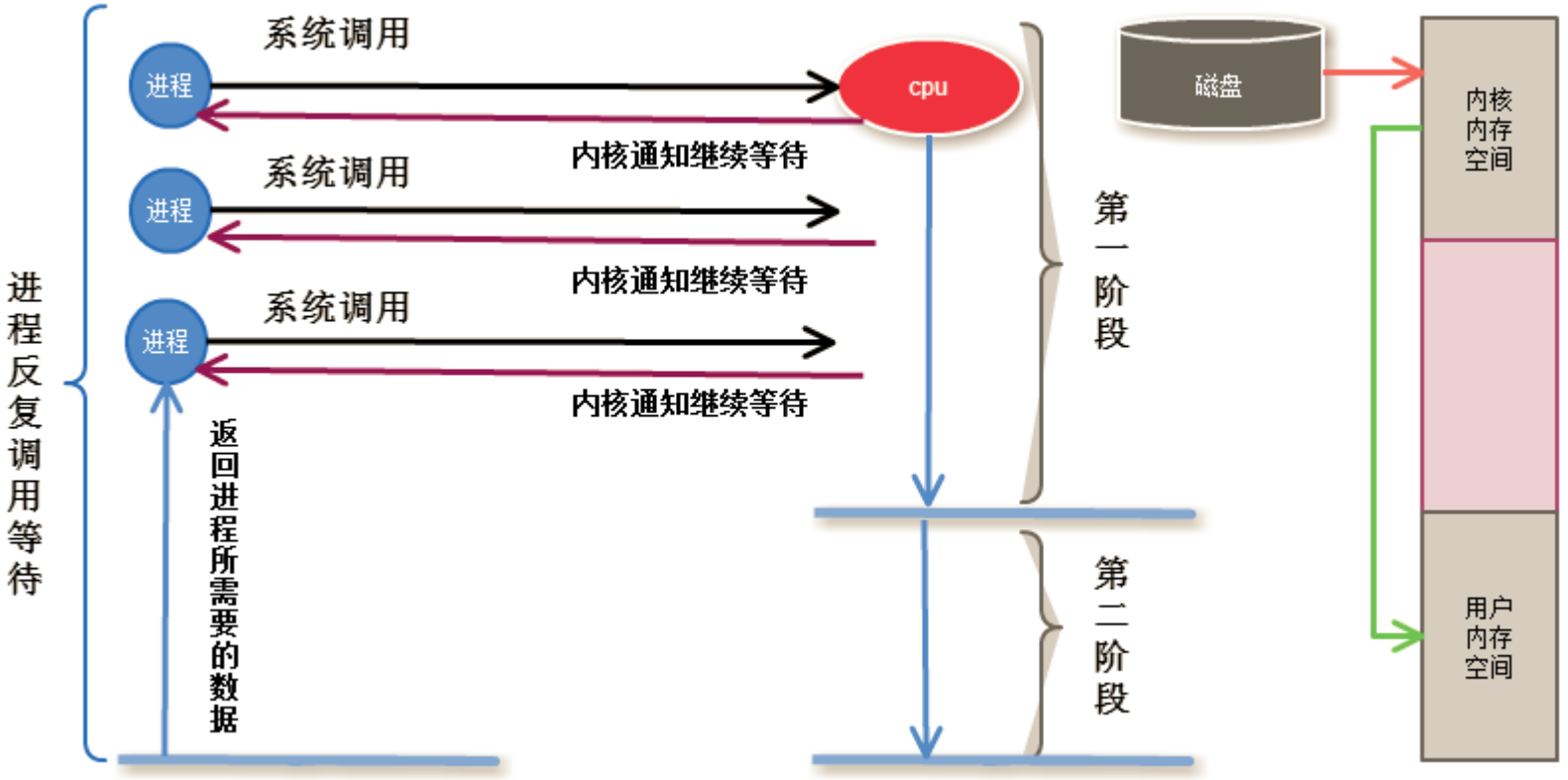
I/O多路复用

- 原本进程和系统内核直接沟通,现在在中间加一个i/o复用select。



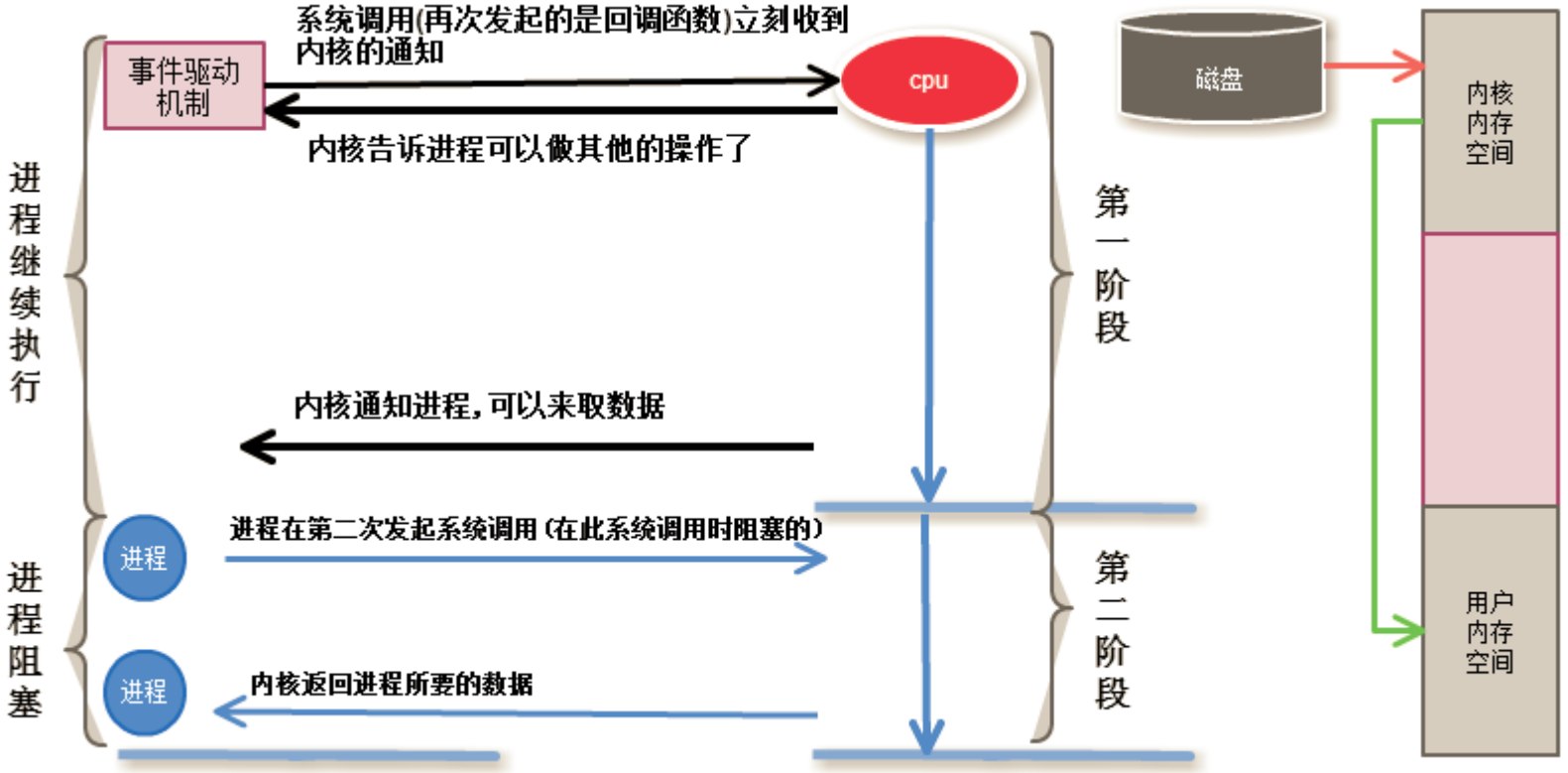
非阻塞I/O

- 非阻塞：进程发起I/O调用，I/O自己知道需过一段时间完成，就立即通知进程进行别的操作，则为非阻塞I/O



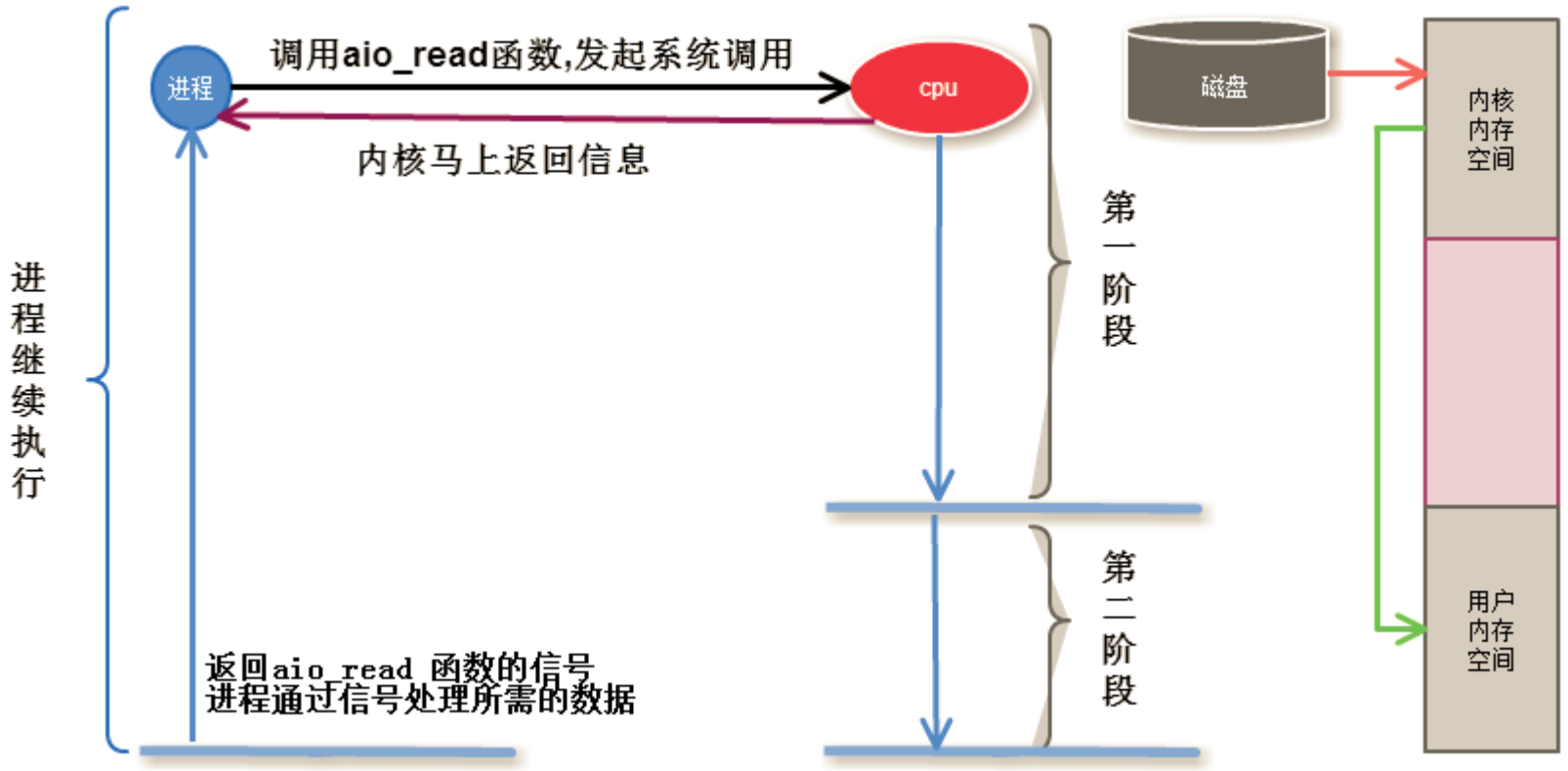
事件（信号）驱动I/O

- 进程发起调用,通过回调函数,内核会记住是那个进程申请的,一旦第一段完成了,就可以向这个进程发起通知,这样第一段就是非阻塞的,进程不需要盲等了,但是第二段依然是阻塞的。

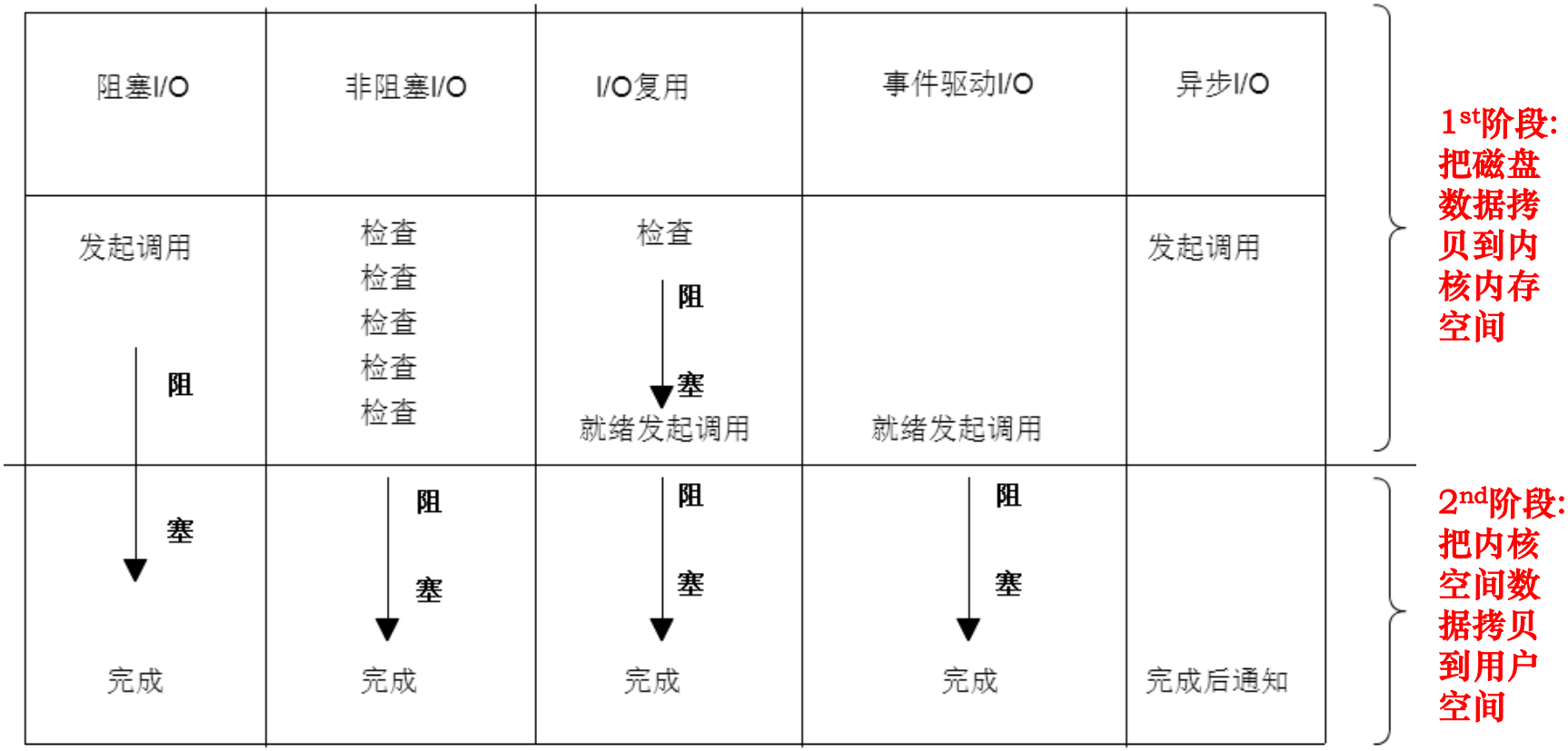


异步I/O

- 无论第一第二段,不再向系统调用提出任何反馈,只有数据完全复制到服务进程内存中后,才向服务进程返回ok的信息,其它时间,进程可以随意做自己的事情,直到内核通知ok信息。



五种模型比较

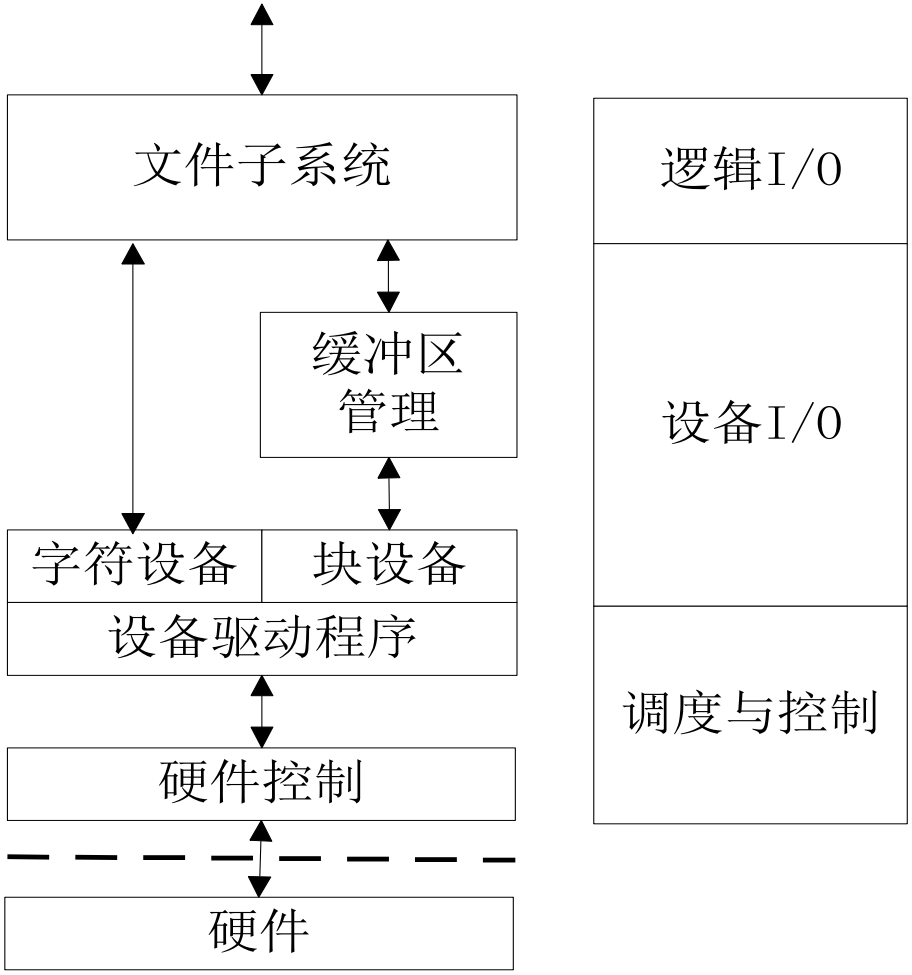

阻塞I/O
非阻塞I/O
I/O复用
事件驱动I/O
异步I/O
发起调用
阻
塞
检查
检查
检查
检查
检查
检查
阻
塞
就绪发起调用
就绪发起调用
发起调用
阻
塞
阻
塞
阻
塞
完成
完成
完成
完成
完成后通知
1st阶段:
把磁盘
数据拷
贝到内
核内存
空间
2nd阶段:
把内核
空间数
据拷贝
到用户
空间

设备管理实例

UNIX的设备管理

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。



UNIX的设备管理

- 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；

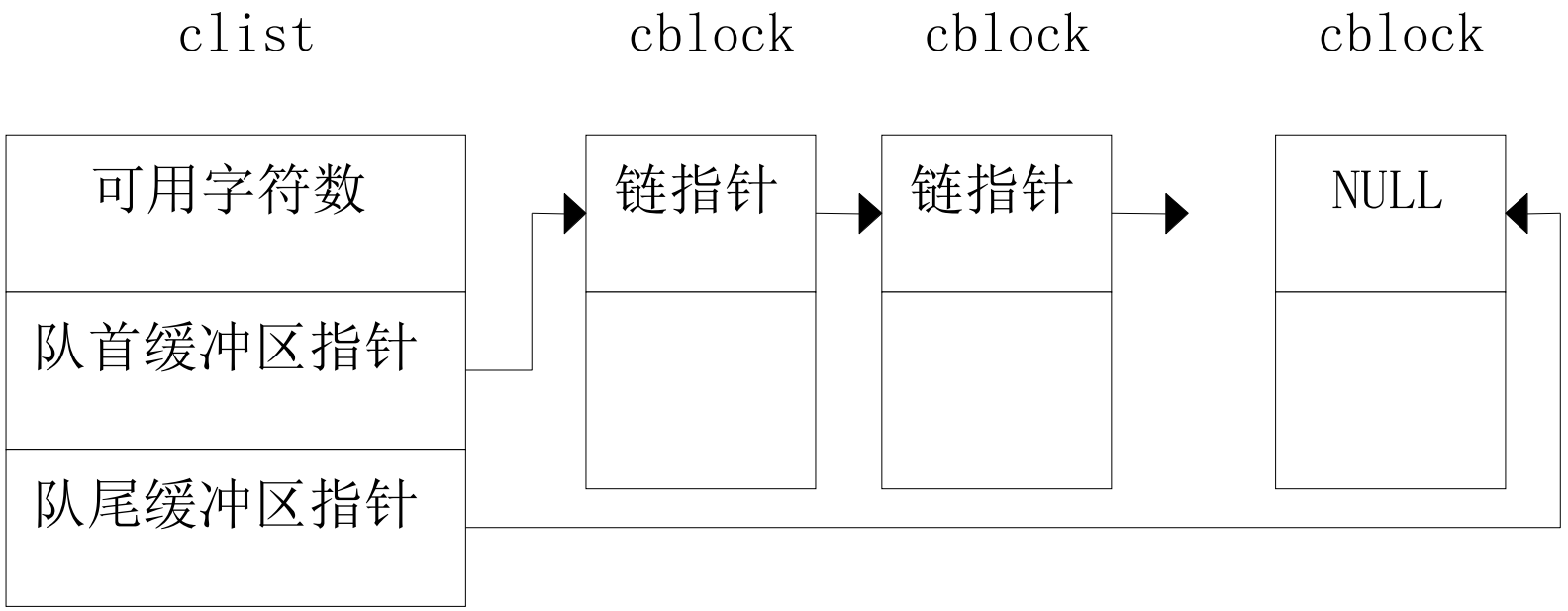
- 缓冲区检索和置换方式
  - 设备缓冲区队列为Hash队列：为了检索方便，设备缓冲区队列为一个按(逻辑设备号，物理块号)组织的Hash队列。把逻辑设备号和物理块号之和对64取模作为哈希函数值，据此建立多个哈希队列（64个队列）。
  - 缓存块可同时链入设备缓冲区队列和空闲缓冲队列：一个缓存块在分配给一个外设后，一直与该外设相关(既使该缓存块在空闲缓冲队列中)，直到分配给另一外设。即：设备释放缓冲区后，该缓冲区可处于“延迟写”状态，等待被写入到外设；该缓冲区被重新分配之前，要将其写入到外设。

- 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。

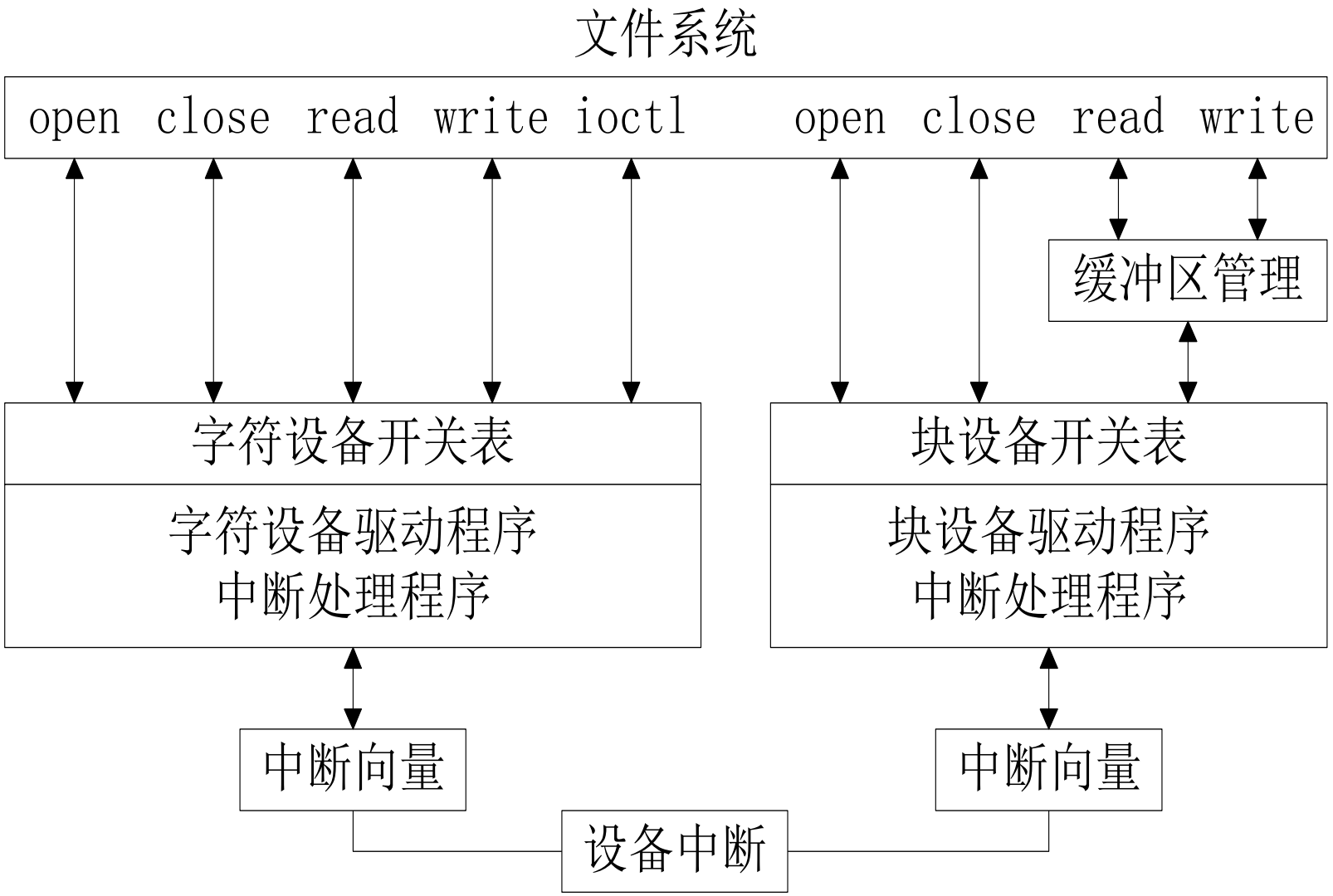
- 字符缓冲池的基本分配单位为字符缓冲区cblock：供各种字符设备（的设备驱动程序）使用。
  - 每个缓冲区大小为70字节，包括：第一个字符和最后一个字符的位置（便于从开头移出字符和向末尾添加字符），指向下一缓冲区的指针c_next，可存放64字符的数据区



- 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；

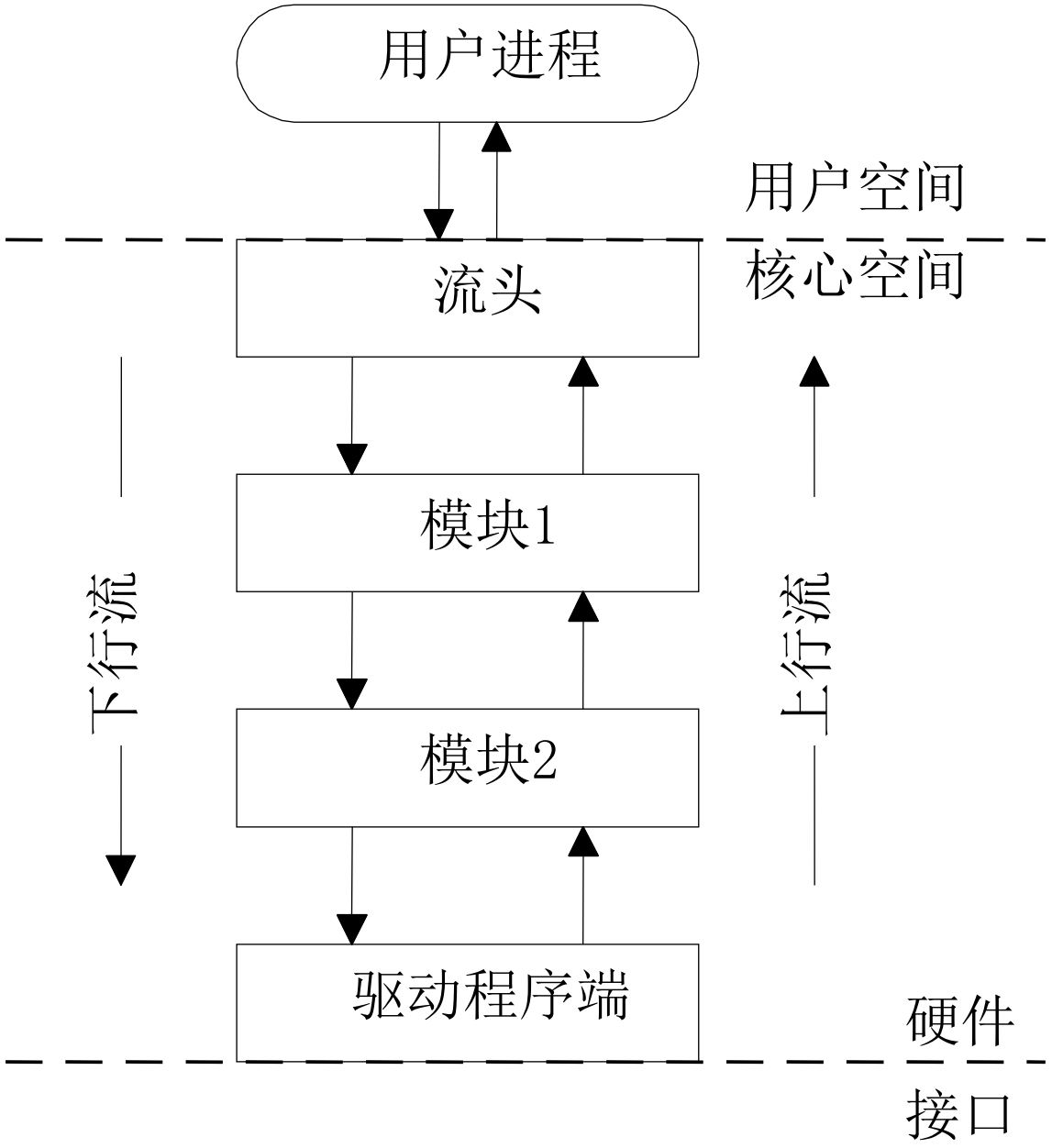
设备开关表(switch table)

- 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
open close read write ioctl
open close read write
缓冲区管理
字符设备开关表
块设备开关表
字符设备驱动程序
中断处理程序
块设备驱动程序
中断处理程序
中断向量
中断向量
设备中断

流机制(streams)

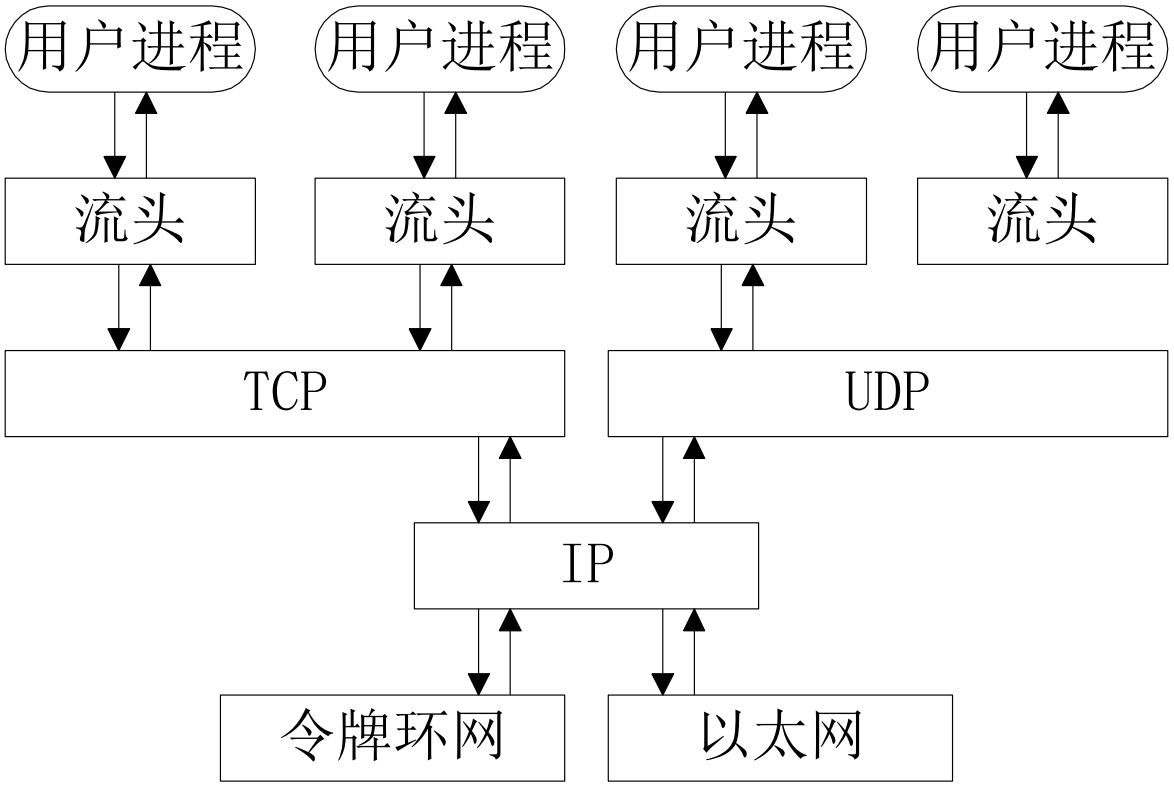
- 流的引入：流的引入是为了解决内核与驱动程序抽象层次过高，而引起的驱动程序功能大量重复。它可提供一个完全基于消息的模块化的驱动程序编写方法。
- 流的定义：流是一组系统调用、内核资源和创建、使用及拆除流的例程的集合，构成一个数据传输通道，两端为读队列和写队列。
- 流的结构：上行流(upstream)和下行流(downstream)



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；

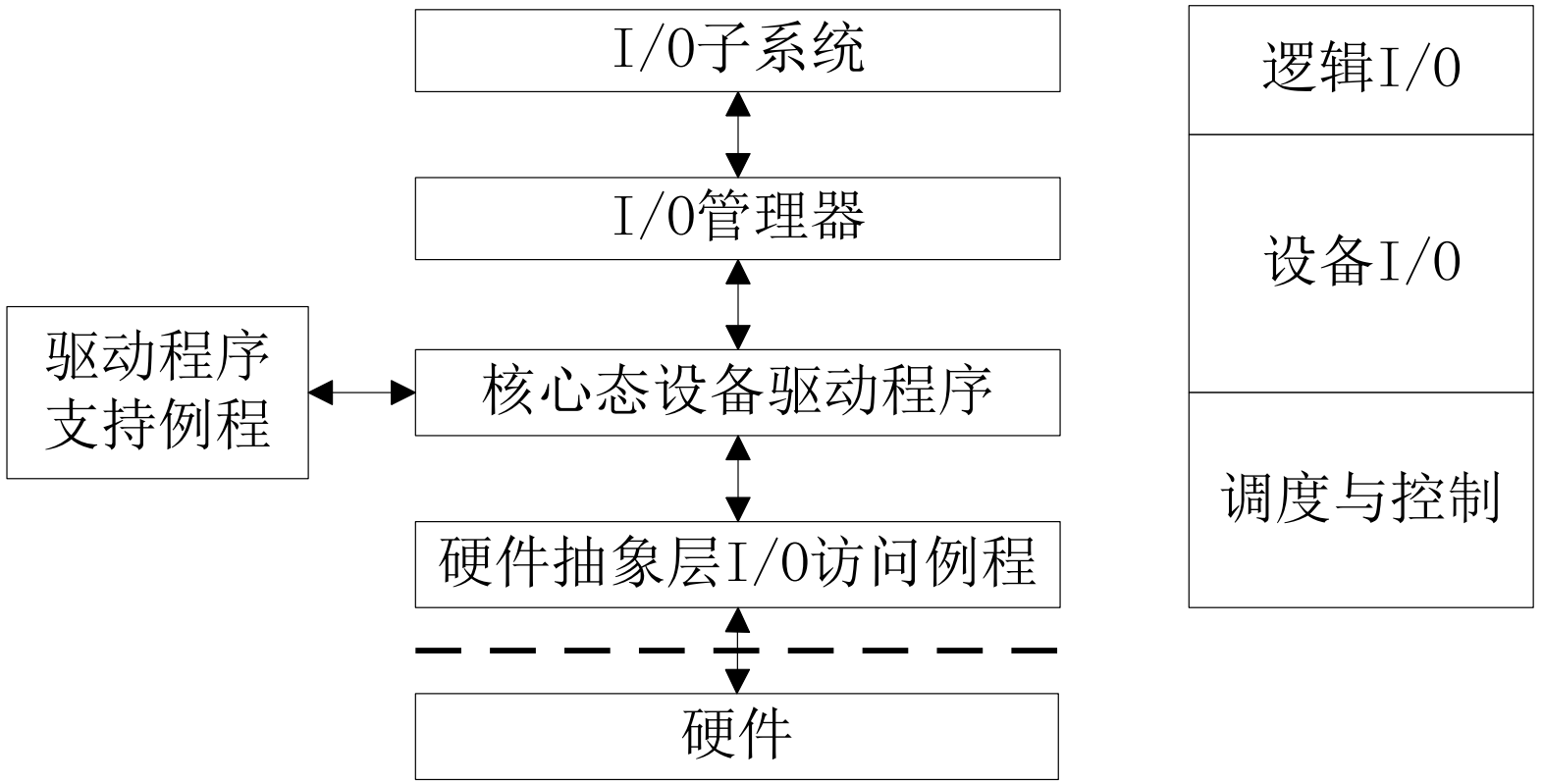

用户进程
用户进程
用户进程
用户进程
流头
流头
流头
流头
TCP
UDP
IP
令牌环网
以太网

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)；


I/O子系统
I/O管理器
驱动程序
支持例程
核心态设备驱动程序
硬件抽象层I/O访问例程
硬件
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设备繁多，OS将其分类管理；
- I/O控制方式有四种，逐步递进，使CPU逐步摆脱IO的影响，但软硬件机制趋于复杂；
- I/O软件的组成，采用分层思想。对用户进程抽象为设备无关的标准操作，设备驱动程序接收用户请求，监控具体物理设备完成相关操作。
- 探讨了I/O缓冲管理和I/O设备管理的具体方法。
- I/O性能问题探讨了不同类型I/O的特点，阻塞vs.非阻塞，同步vs.异步