第10章 I/O设备管理

孙承杰

E-mail：sunchengjie@hit.edu.cn

哈工大计算学部人工智能教研室

2023年秋季学期

主要内容

10.1 设备管理概述

10.2 I/O设备（分类与控制器）

10.3 I/O控制方式

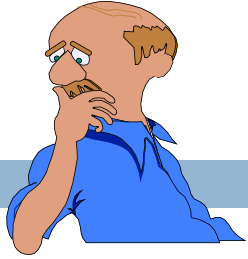
10.4 缓冲技术

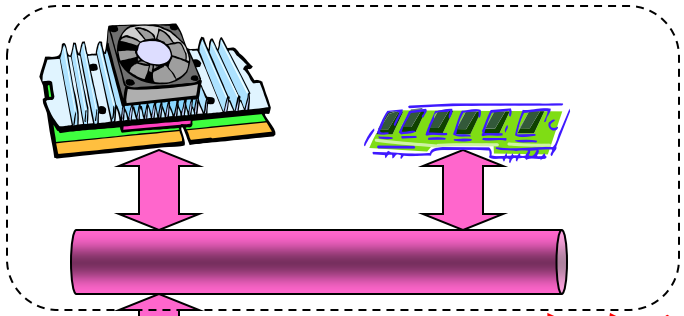
10.5 I/O软件层次

10.6 设备分配

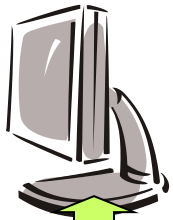
10.7 Linux IO

认识计算机外设与计算机!





CPU-内存总线



图形控制器

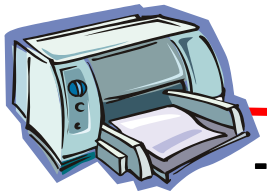
总线控制器

PCI总线

IDE控制器

扩展总线接口

扩展总线



并行口

认识计算机外设与计算机!

关于IO设备管理的思考?

- 计算机包含不同厂商的各种外设
- 外设不断更新换代
- 操作系统管理这些外设，为用户提供服务
- CPU速度很快，外设速度很慢

接口规范，内存缓冲，高层抽象

关于IO设备管理的思考?

操作系统管理软硬件资源，为用户提供服务!

提高设备的利用率

- 尽量提高CPU与I/O设备之间的并行工作程度
- 主要技术：中断技术、DMA技术、缓冲技术。

为用户提供方便、统一的界面

- 方便，是指用户能独立于具体设备的复杂物理特性之外而方便地使用设备
- 统一，是指对不同的设备尽量使用统一的操作方式，例如各种字符设备用 一种I/O操作方式。

10.1 设备管理概述

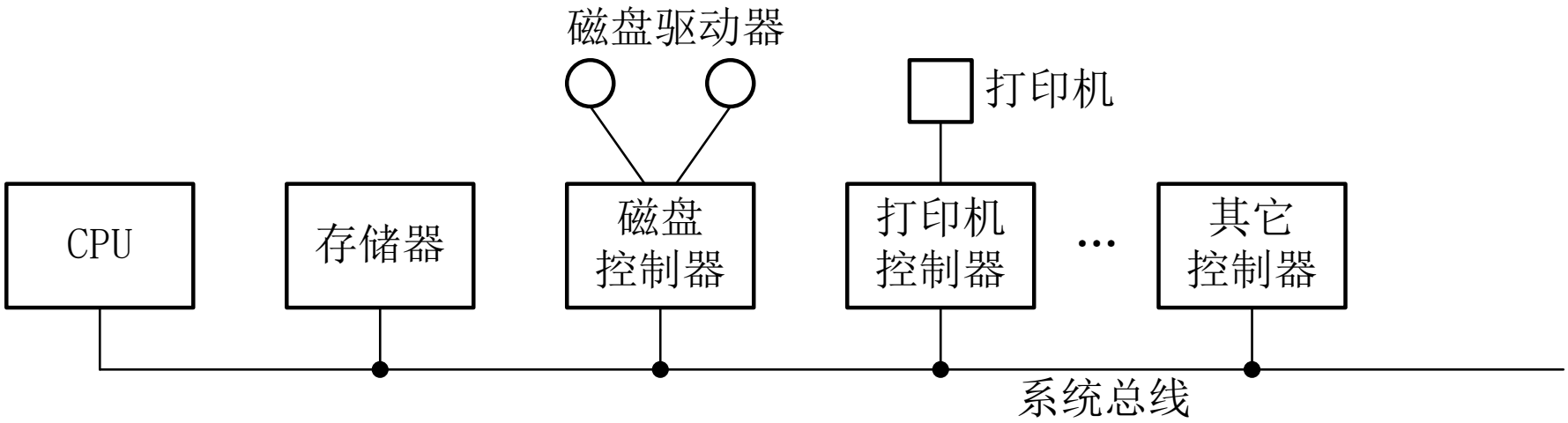
1、I/O系统的组成

- **I/O系统**是用于实现数据输入、输出及数据存储的系统。
- **I/O系统包括：**
  - I/O和存储信息的设备
  - 设备控制器
  - 高速总线
  - I/O通道或I/O处理机

10.1 设备管理概述

总线系统

- 计算机系统中的各部件，如CPU、内存及各种I/O设备之间的联系，都是通过总线来实现的。
- 总线的性能用总线的时钟频率、带宽和相应的总线传输速率等指标来衡量。



总线系统

ISA和EISA总线

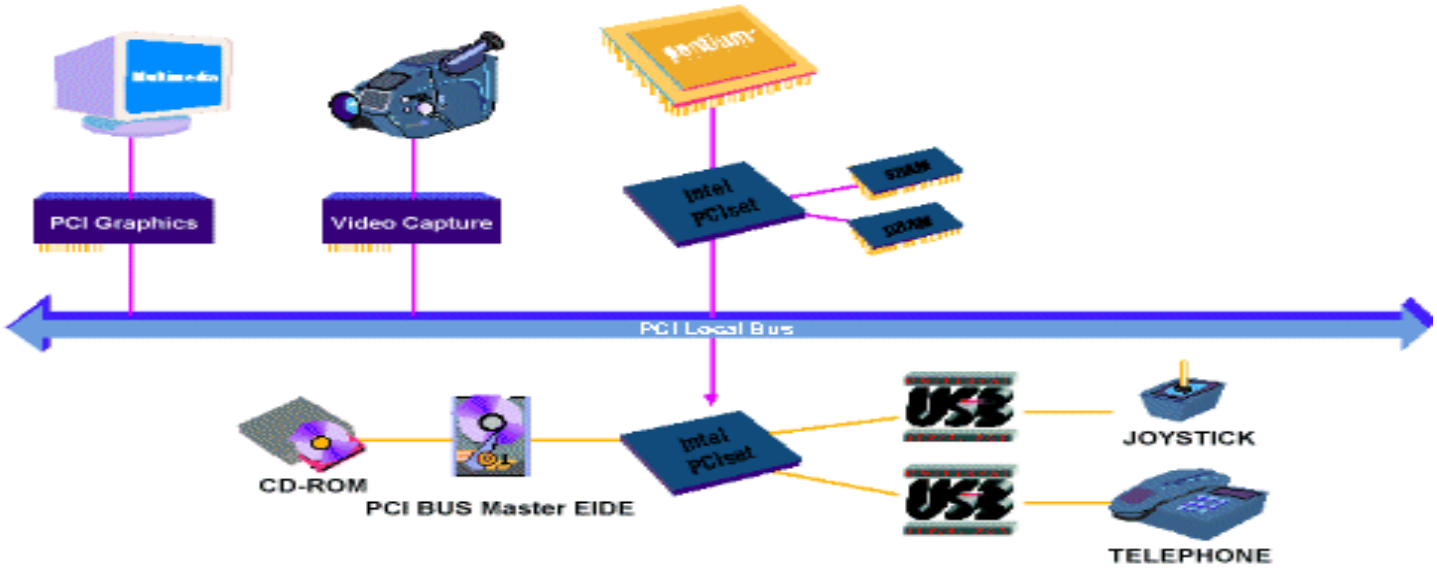
- **ISA(Industry Standard Architecture)总线**
  - 这是为了1984年推出的80286型微机而设计的总线结构。 其总线的带宽为8位，最高传输速率为2 Mb/s。之后不久又推出了16位的总线，其最高传输速率为8 Mb/s，后又升至16 Mb/s， 能连接12台设备。
- **EISA(Extended ISA)总线**
  - 到80年代末期，ISA总线已难于满足带宽和传输速率的要求，于是人们又开发出扩展ISA(EISA)总线，其带宽为32位，总线的传输速率高达32 Mb/s，同样可以连接12台外部设备。

10.1 设备管理概述

总线系统

局部总线(Local Bus)

- VESA(Video Electronic Standard Association)总线
  - 带宽为32位，最高传输速率为132Mb/s，所能连接的设备数仅为2—4台，在控制器中无缓冲。
- PCI(Peripheral Component Interface)总线
  - 支持64位系统，PCI最多能支持10种外设



2、I/O设备的特点

- CPU与I/O的速度差别大
  - 尽量使两者交替运行
  - 减少由于速度差异造成的整体性能开销
- I/O性能经常成为系统性能的瓶颈
- 操作系统庞大复杂的原因之一：资源多、杂，并发，
  - 外设种类繁多，结构各异
  - 输入输出数据信号类型不同
  - 速度差异很大
- 与其他功能联系密切，特别是文件系统

3、I/O设备的管理目标与任务

□按照用户的请求，控制设备的各种操作，完成I/O设备与内存之间的数据交换，最终完成用户的I/O请求

① 设备分配与回收

- 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

② 执行设备驱动程序，实现真正的I/O操作

③ 设备中断处理：处理外部设备的中断

④ 缓冲区管理：管理I/O缓冲区

3、I/O设备的管理目标与任务

口建立方便、统一的独立于设备的接口

- **方便性：**向用户提供使用外部设备的方便接口，使用户编程时不考虑设备的复杂物理特性
- **统一性：**对不同的设备采取统一的操作方式，在用户程序中使用的是逻辑设备
- **独立性：**逻辑设备与物理设备、屏蔽硬件细节（设备的物理细节，错误处理，不同I/O的差异性）
  1. 提高操作系统的可适应性和可扩展性
  2. **从用户角度：**用户在编制程序时，使用逻辑设备名，由系统实现从逻辑设备到物理设备（实际设备）的转换，并实施I/O操作
  3. **从系统角度：**设计并实现I / O软件时，除了直接与设备打交道的低层软件之外，其他部分的软件不依赖于硬件

3、I/O设备的管理目标与任务

- 充分利用各种技术（通道，中断，缓冲，异步I/O等）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，提高资源利用率。
  - 并行性
  - 均衡性（使设备充分忙碌）
- 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

10.1 设备管理概述

4、I/O设备管理基本方法

- 设备管理的方法主要有3种：

（1）操作系统直接操纵设备的运行，例如直接程序控制、中断方式控制

（2）操作系统间接操纵设备的运行，例如DMA和通道方式

（3）操作系统通过使用设备驱动程序，将设备管理工作通过设备专有驱动程序形式来体现。OS只需制定标准，将具体操纵设备的程序交给不同的制造商去开发

1、I/O设备的类型

按设备的使用特性分类

- **存储设备：**也称外存或后备存储器、辅助存储器，是计算机系统用以存储信息的主要设备。该类设备存取速度较内存慢，但容量比内存大得多，相对价格也便宜。
- **输入/输出设备：**
  - **输入设备**
  - **输出设备**
  - **数据通信设备**

10.2 I/O设备

1、I/O设备的类型

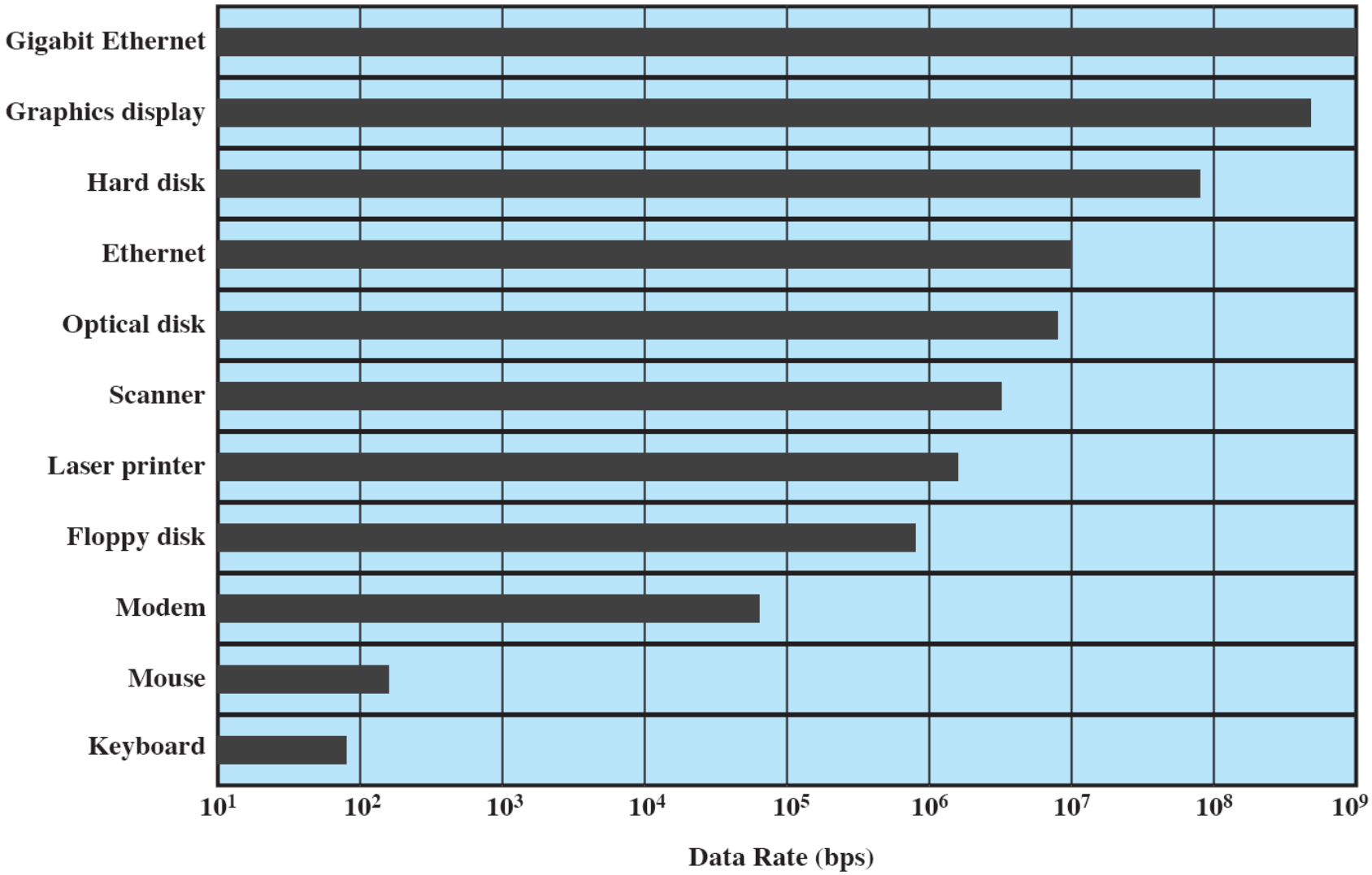
口按传输速率分类

- **低速设备：**传输率每秒几个 ~ 数百字节，键盘、鼠标
- **中速设备：**传输率每秒数千个 ~ 数万字节，打印机
- **高速设备：**传输率每秒数十万 ~ 数兆字节，磁盘机、光盘机

口按信息交换的单位分类

- **块设备**(Block Device)：用于存储信息，基本单位是块。典型块设备是磁盘。其传输速率高，可寻址，I/O系统采用DMA方式。
- **字符设备**(Character Device)：用于数据的输入和输出。基本单位是字符。传输速率低，不可寻址，I/O采用中断驱动方式。典型字符设备如交互式终端、打印机等。

10.2 I/O设备



Typical I/O Device Data Rates

1、I/O设备的类型

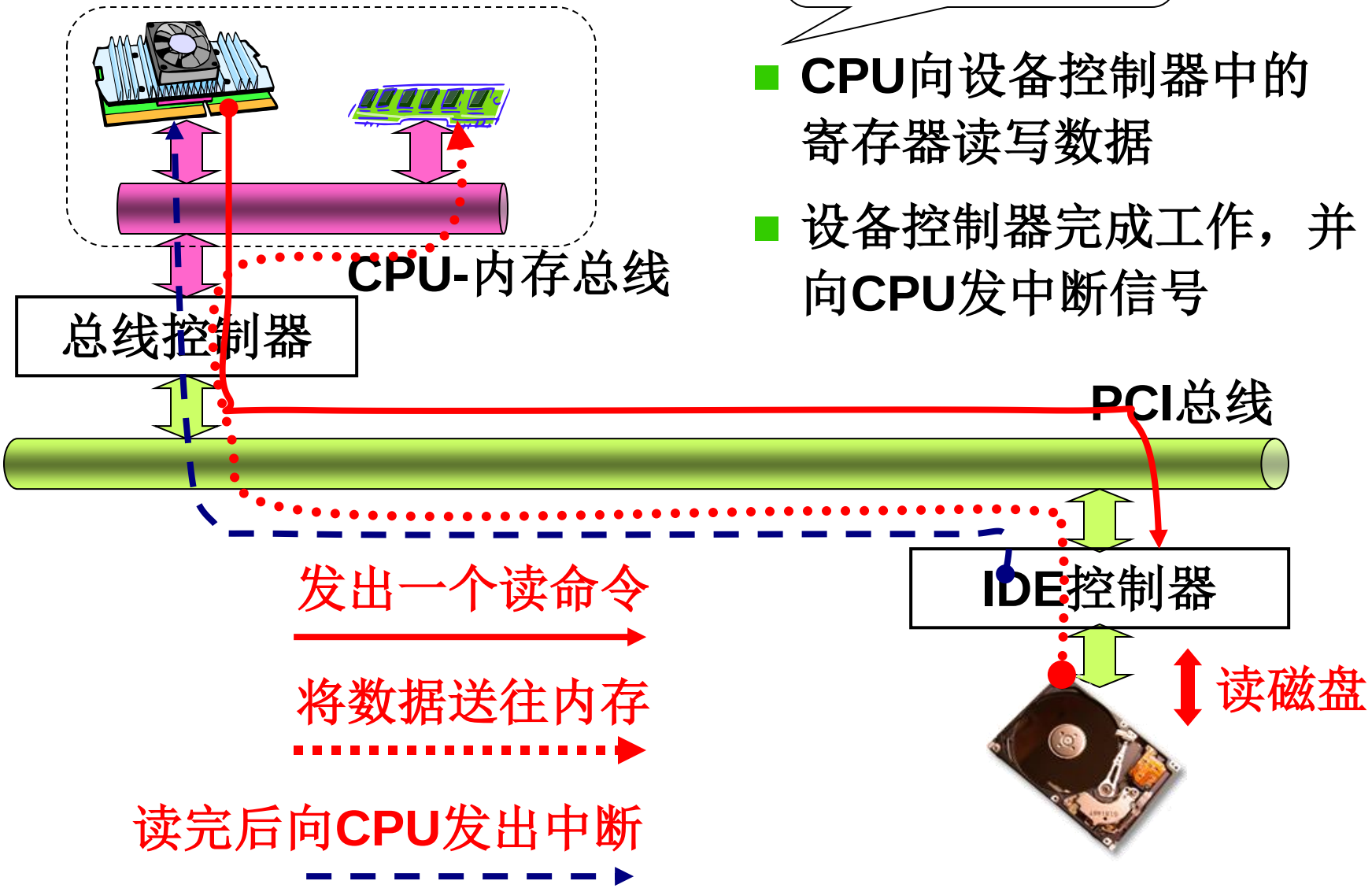
按设备的共享属性分类

- **独占设备：**
  - 在一段时间内只能有一个进程使用的设备，一般为低速I/O设备（如打印机，磁带等）
- **共享设备：**
  - 在一段时间内可有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，其资源利用率高（如硬盘）
- **虚拟设备：**
  - 在一类设备上模拟另一类设备，常用共享设备模拟独占设备，用高速设备模拟低速设备，被模拟的设备称为虚设备
  - 目的：将慢速的独占设备改造成多个用户可共享的设备，提高设备的利用率
    - 如SPOOLing技术，利用虚设备技术——用硬盘模拟输入输出设备

想一想外设怎么工作?

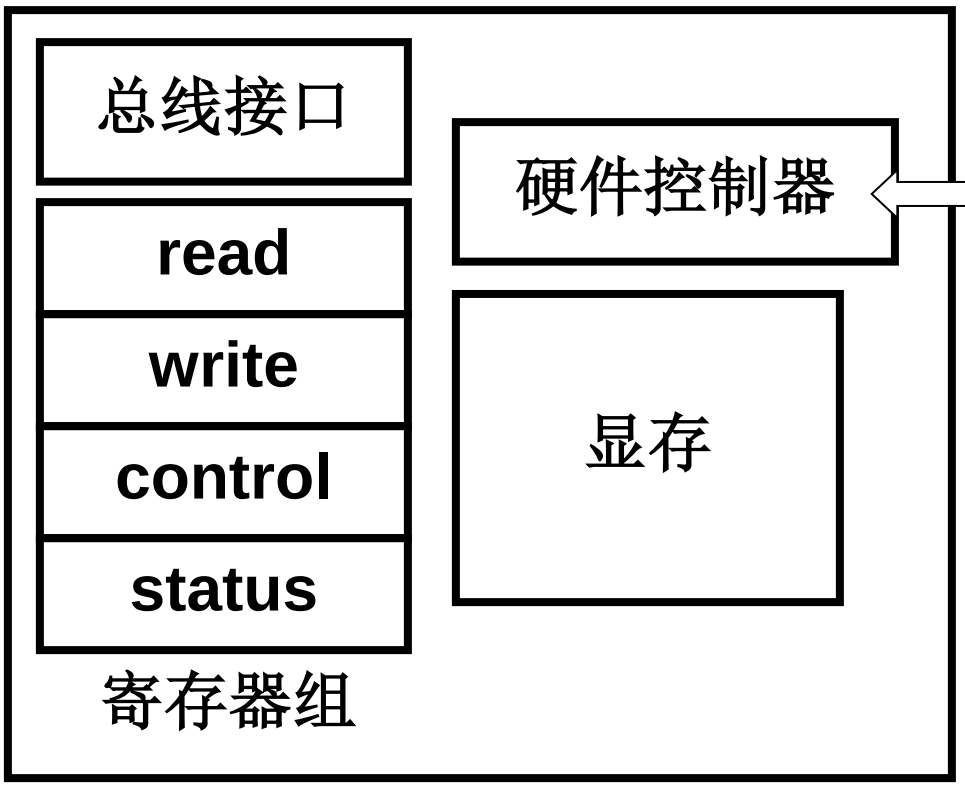
想让外设工作
并不复杂!

- CPU向设备控制器中的寄存器读写数据
- 设备控制器完成工作，并向CPU发中断信号

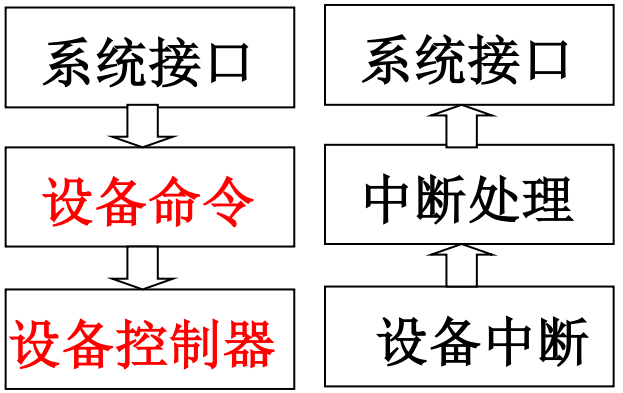


2、I/O设备控制器

I/O系统如何向设备发命令?

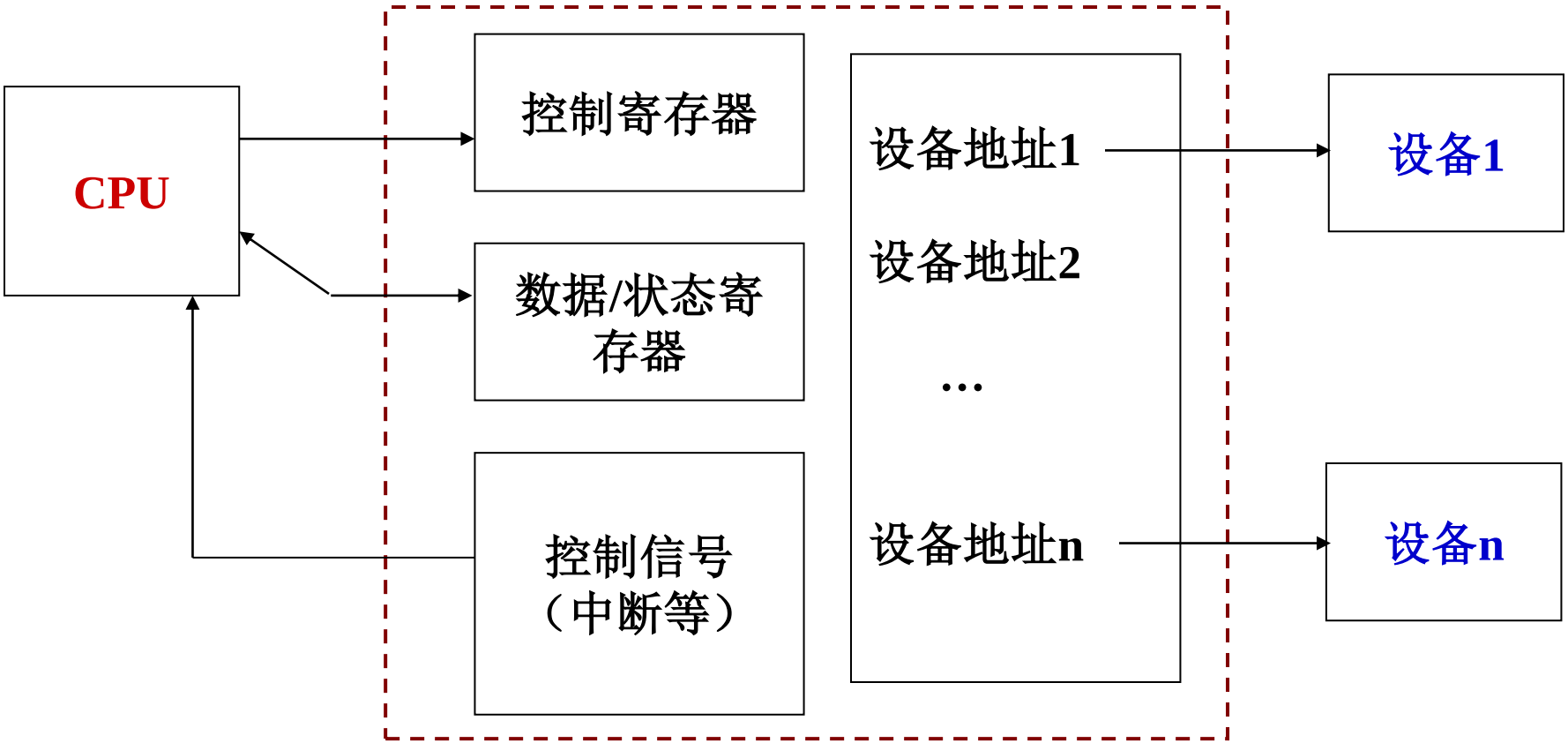


设备控制器的结构



2、I/O设备控制器

- CPU、设备控制器与设备之间关系



10.2 I/O设备

2、I/O设备控制器

- **设备控制器**是CPU与I/O设备之间的**接口**。
  - 功能：完成设备与主机间的连接和通信
  - 在小型和微型机中，它常采用印刷电路卡插入计算机主板上的总线插槽
  - 通过若干接口寄存器或接口缓冲区与CPU通信
- 设备控制器是一个**可编址**的设备
- 设备控制器可分为两大类：
  - **控制字符设备**的控制器；
  - **控制块设备**的控制器。

3、设备控制器的基本功能

接收和识别命令

- **控制寄存器：**来存放接收的命令和参数，
- **命令译码器：**对接收的命令进行译码
- 操作系统将命令写入控制器的控制寄存器（或接口缓冲区）中，以实现输入 / 输出，并从控制寄存器读取状态信息或结果信息

数据交换

- CPU ———— 控制器（数据寄存器）————设备

标识和报告设备的状态

- 设备控制器中应用“状态寄存器”

3、设备控制器的基本功能

地址识别

- CPU通过“地址”与设备通信，设备控制器应能识别它所控制的设备地址以及其各寄存器的地址。
- 配置地址译码器
- I/O指令形式与I/O地址是相互关联的
  - 内存映像编址（内存映像I/O模式）
  - I/O独立编址（I/O专用指令）

数据缓冲

- 用于解决I/O设备速率低，而CPU和内存的速率很高的矛盾

差错控制

- 进行差错检测，并向CPU报告

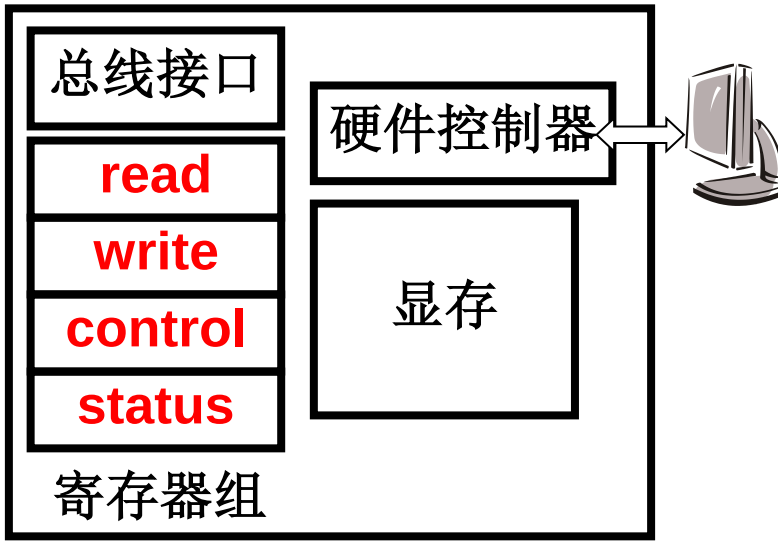
10.2 I/O设备

- PC上的I/O控制器与其对应的I/O地址

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)

10.2 I/O设备

3、设备控制器的基本功能



- 读写设备控制器的寄存器!
- 怎么读写? mov [200], ax

关键是地址

- 设备寄存器的编址
- 独立编址: 需要独立的指令。比如X86:in,out;

  只能与DX,AX,AL寄存器结合使用。如out 0x21, AL
- 内存映像编址: 是内存物理地址空间的一部分，使用

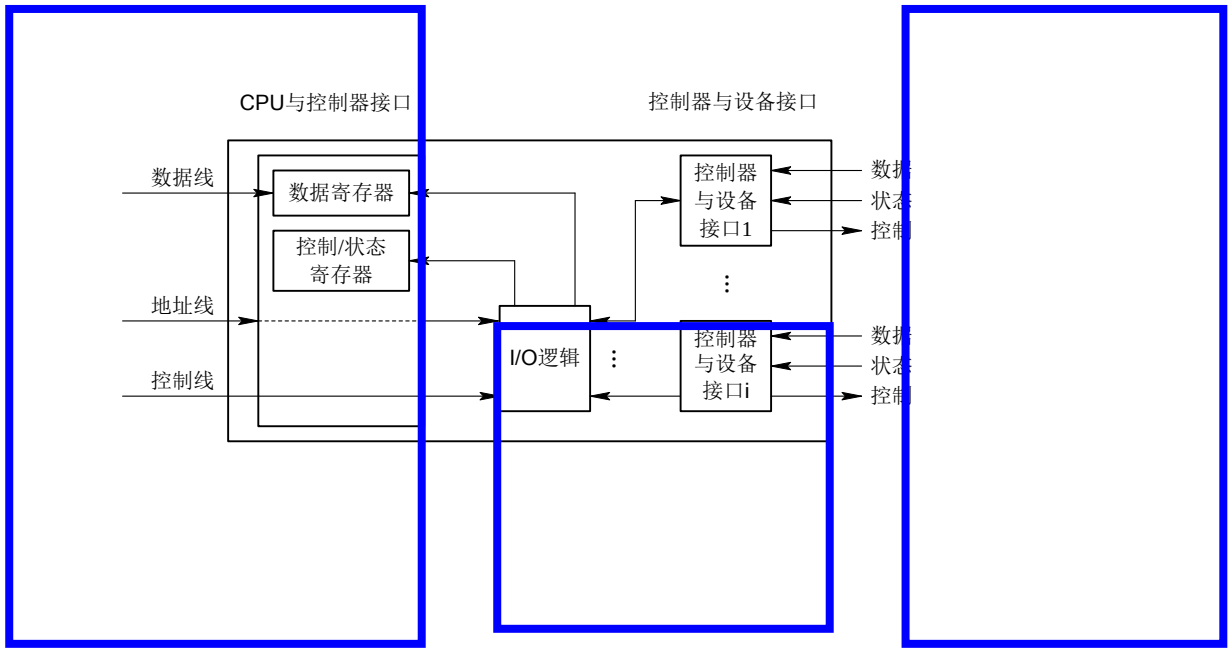
  mov命令，如mov [0x8000f000], AL

查查硬件手册就知道了!

10.2 I/O设备

4、设备控制器的组成

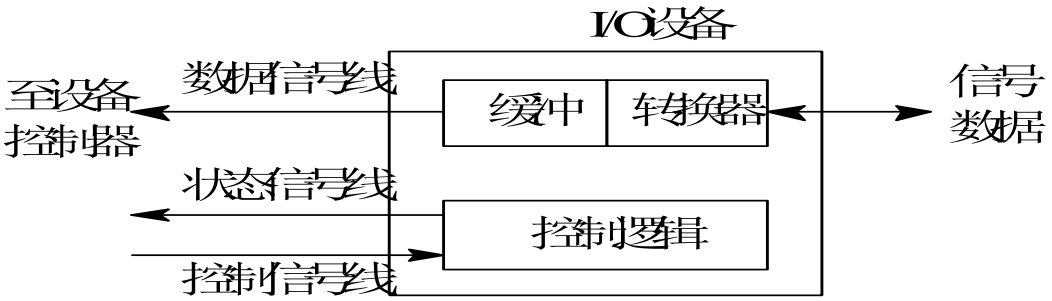
- 设备控制器与处理机的接口
- 设备控制器与设备的接口
- I/O逻辑：在其控制下完成与CPU、设备的通信。



设备控制器的组成

5、设备与控制器之间的接口

- CPU——控制器——设备
- 三种信号线：
  - 数据信号线：双向，有缓存
  - 控制信号线：控制器发给设备，要求其完成相关操作
  - 状态信号线：设备发给控制器，后者“显示”



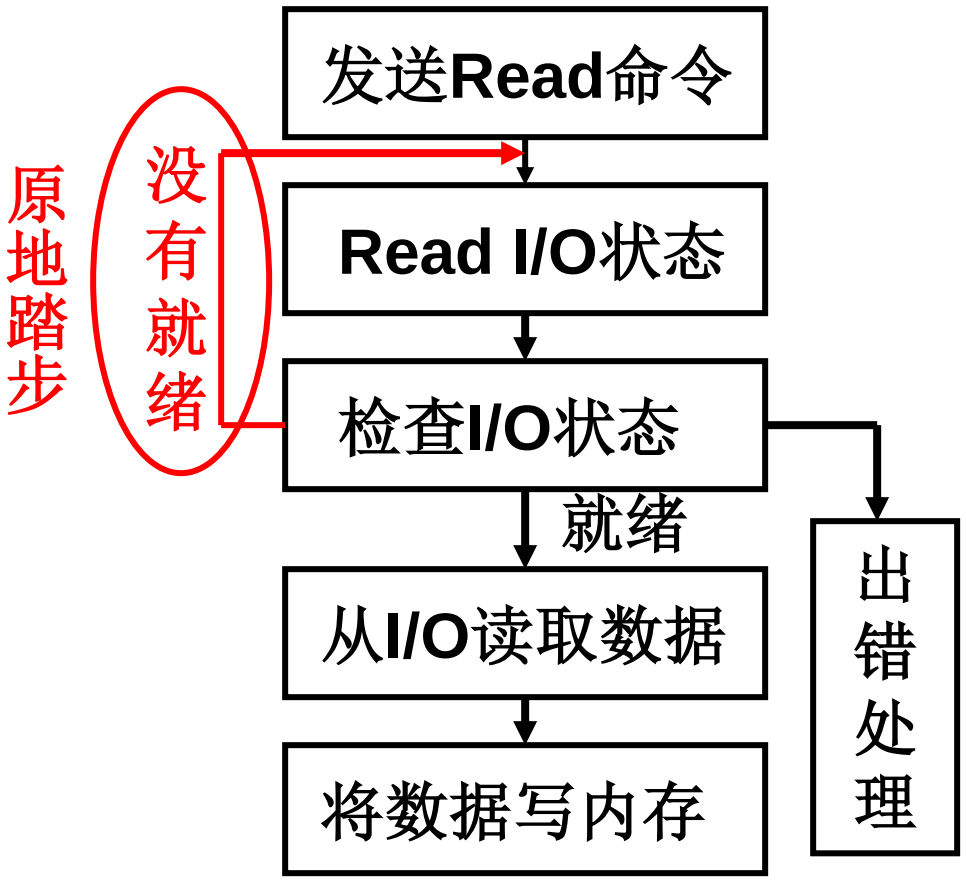
10.3 I/O控制方式

控制I/O硬件的方式?

- 设备管理的主要任务之一是控制设备和内存与CPU之间的数据传送
- I/O控制方式一般有4种：
  - 程序直接控制（查询）方式
  - 中断控制方式
  - 直接内存存取（DMA）方式
  - 通道控制方式（智能设备）

控制I/O硬件的方式

- 方案1: 原地踏步等待!



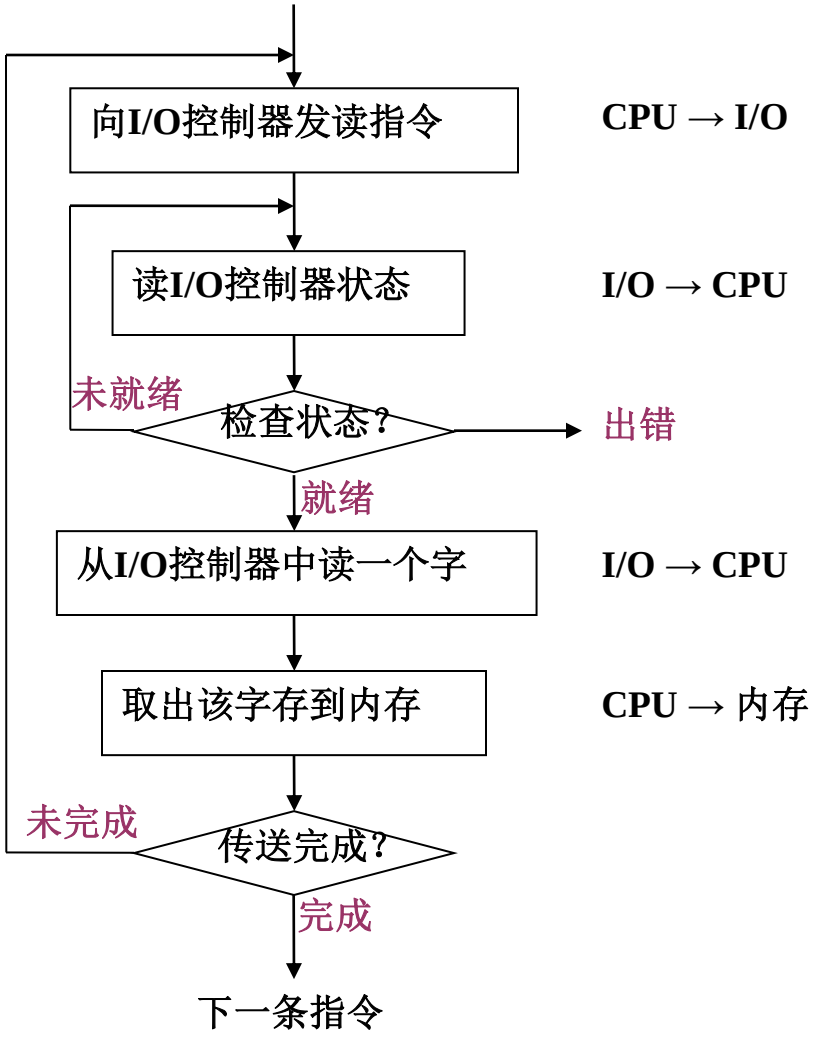
查询（轮询）

```
in AL, 0x??
while(AL!=ready)
{
   in AL, 0x??
}
读数据..
```

轮询!

浪费CPU资源(CPU比外设快太多了)!

例子：程序方法控制I/O设备读入数据流程



例子：程序方法控制I/O设备读入数据流程

程序直接控制（查询）方式 工作步骤小结：

（1）当某进程需要输入/输出数据时，由CPU向设备控制器发出一条I/O指令启动设备工作（对于输出操作，则CPU还要向数据寄存器中存放输出数据）；

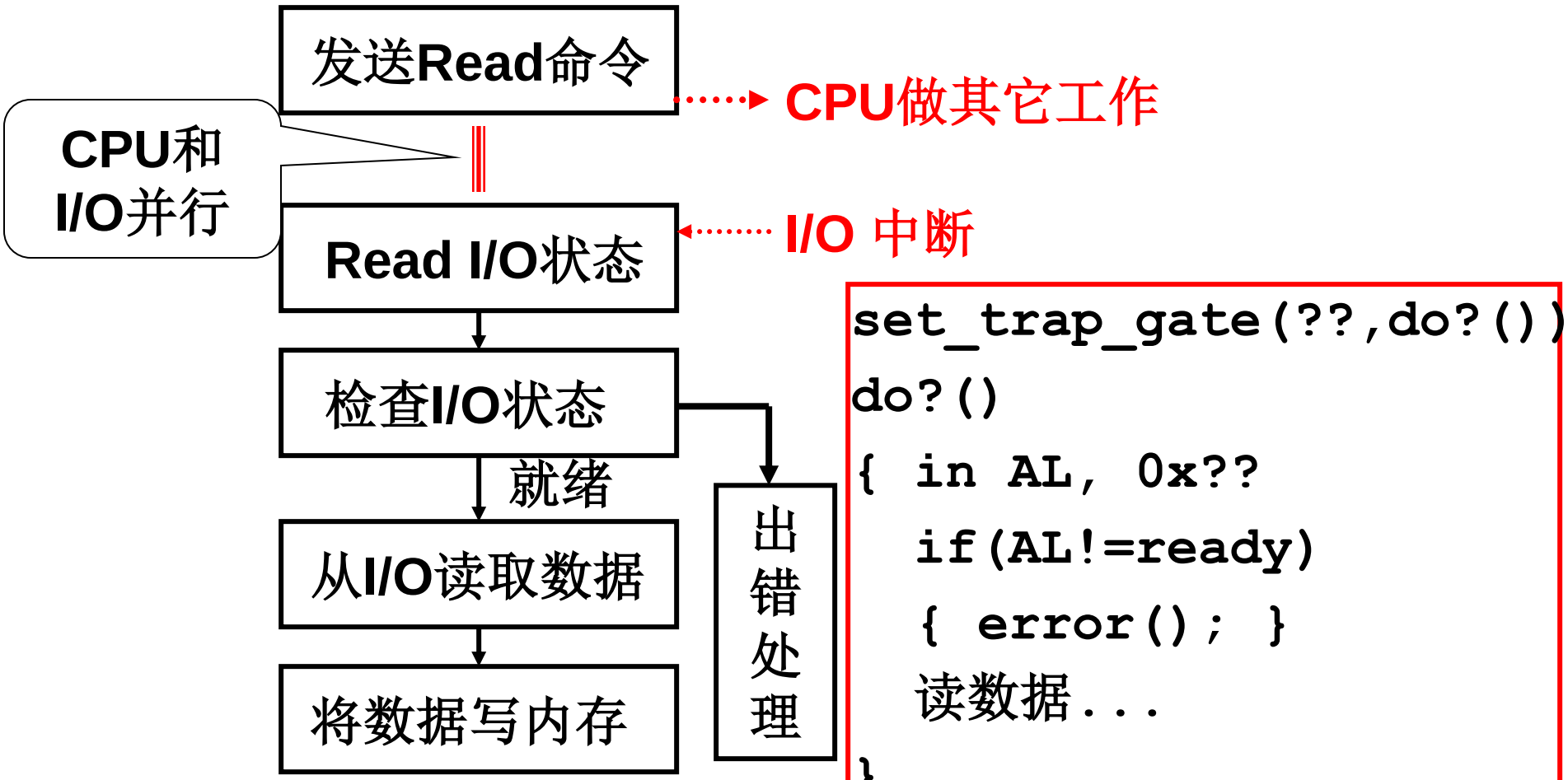
（2）在设备输入/输出数据期间，CPU不断地循环进行查询设备状态寄存器的值（检查I/O工作是否完成）。

（3）若完成，对输入操作来说CPU则从数据寄存器中取出数据，然后进行下一次的输入/输出数据或结束。

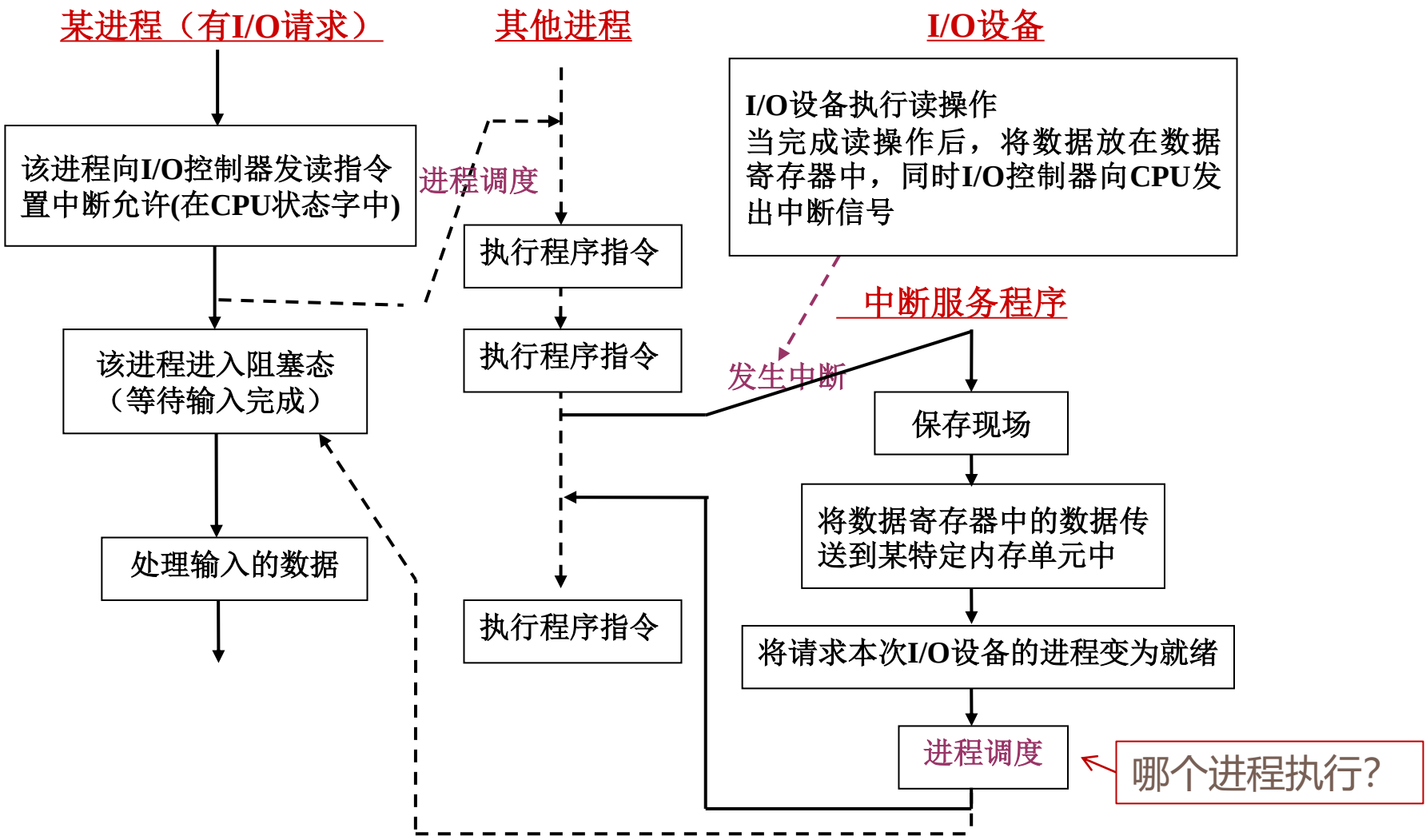
控制I/O硬件的方式?

中断是大部分I/O的处理方式!

- 方案2: 设备就绪了告诉CPU一声! 中断

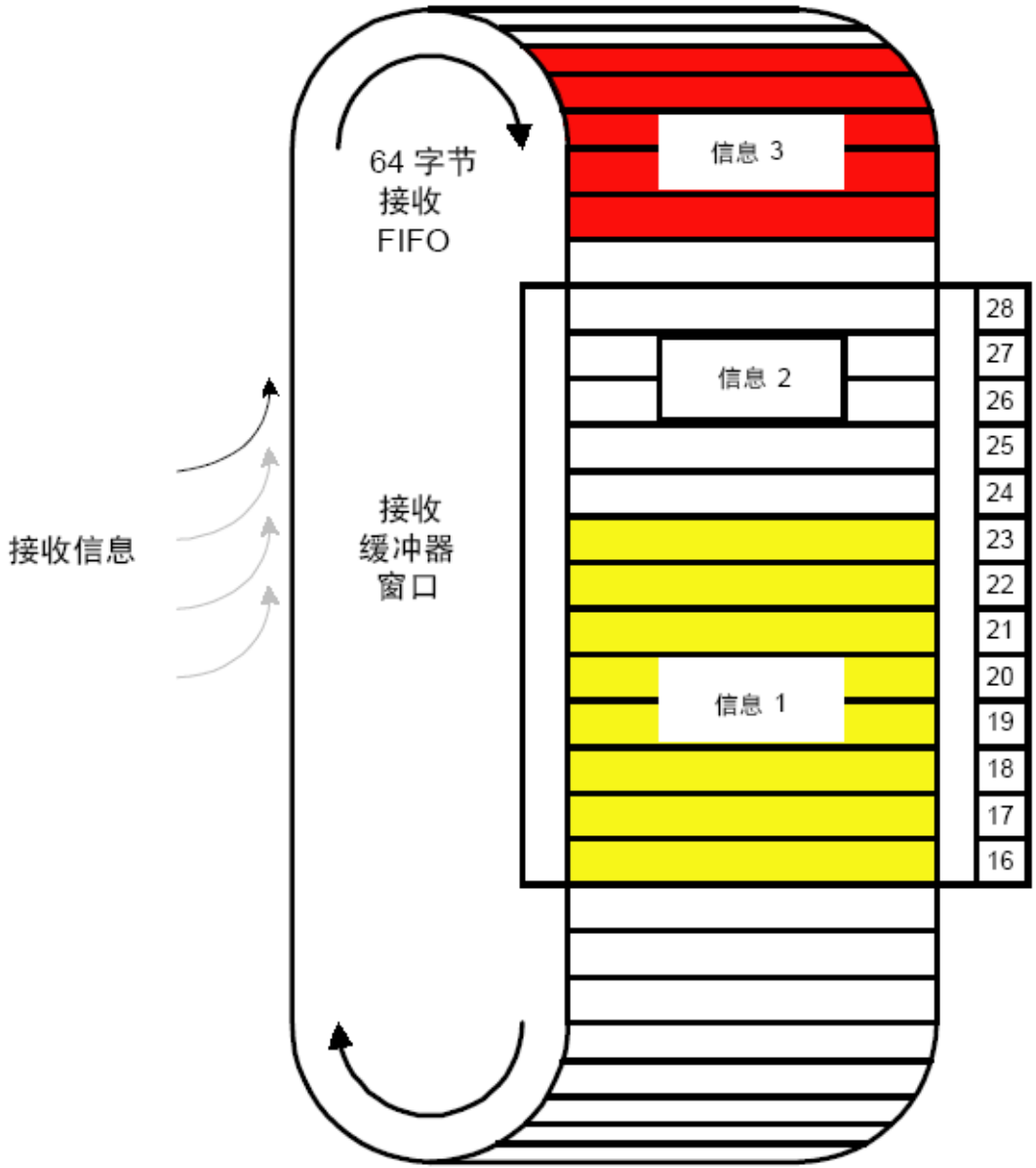


例子：中断方法控制I/O设备读入数据流程



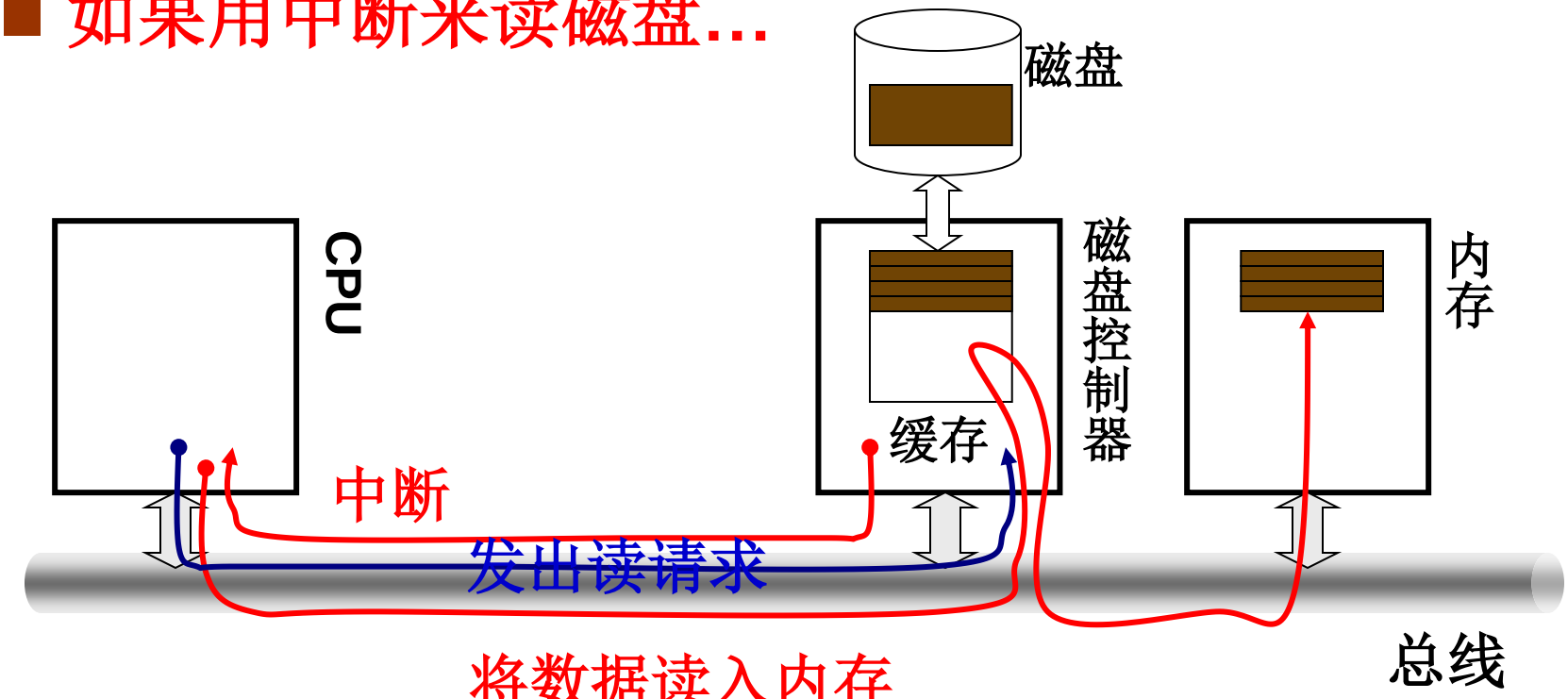
硬件缓冲区

- CPU速度快
- IO收发慢



中断在某些场合还不够!

- 如果用中断来读磁盘...

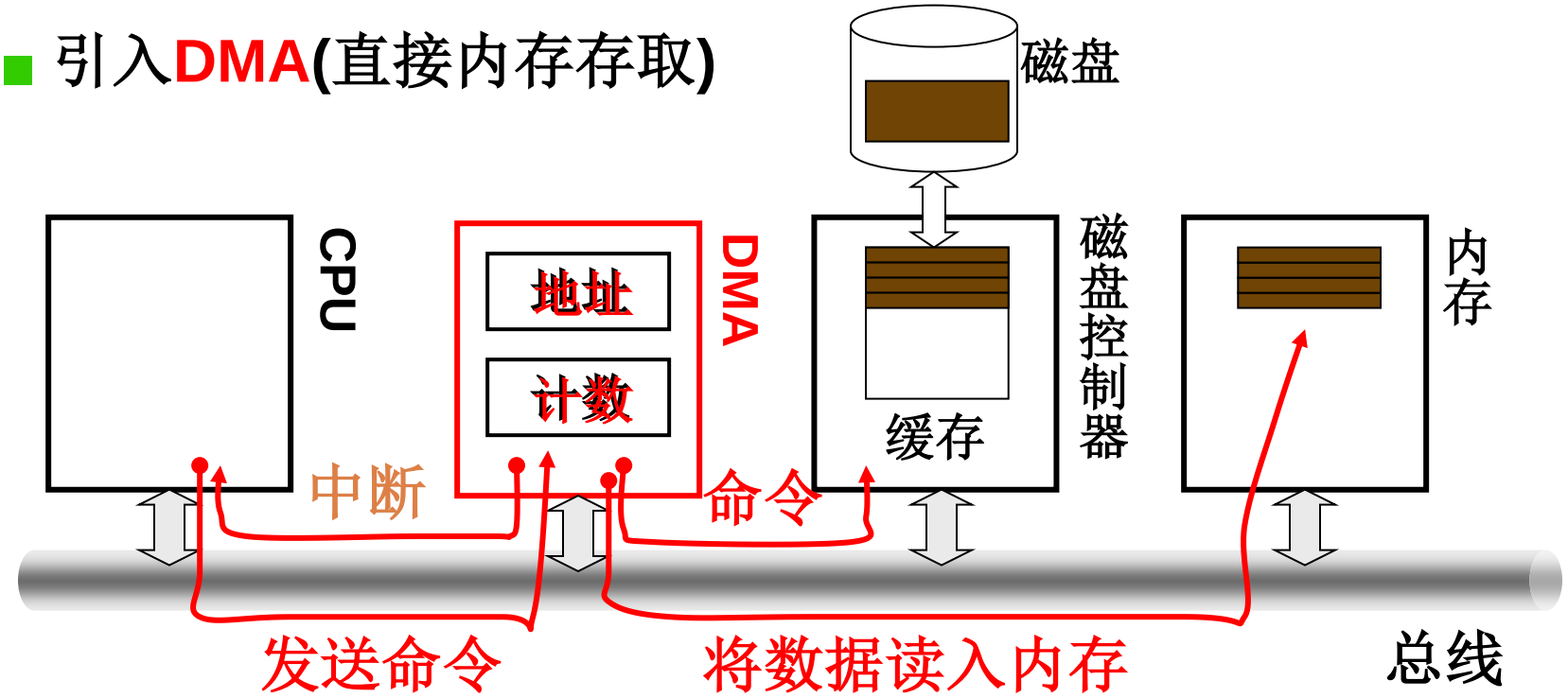


- 每个字节从缓存移动内存都由CPU负责完成

可以设计有一定处理能力的外围设备，
将一些简单任务交给它!

I/O系统发完命令后做什么?

- 方案3: 简单任务自己做，完成了告诉CPU一声!
  - 引入DMA(直接内存存取)



  - 幸运的是: 该方式的细节由DMA设计者考虑，对于操作系统而言，考虑的仍然只是中断处理

例子：DMA方式数据输入过程

（1）当一个进程要求设备输入数据时，CPU对DMA进行初始化工作：

- 存放数据的内存起始地址 - DMA控制器的内存地址寄存器；
- 要输入数据的字节数 - DMA控制器的传送字节数寄存器；
- 控制字(中断允许、DMA启动位=1) - DMA控制器的控制状态寄存器；
- 启动位被置1，则启动DMA控制器开始进行数据传输。

（2）该进程放弃CPU，进入阻塞等待状态，等待第一批数据输入完成。

进程调度程序调度其他进程运行。

（3）由DMA控制器控制整个数据的传输。

- 当输入设备将一个数据送入DMA控制器的数据缓冲寄存器后，DMA控制器立即取代CPU，接管数据地址总线的控制权（CPU工作周期挪用），将数据送至相应的内存单元；
- DMA控制器中的传输字节数寄存器计数减1；
- 恢复CPU对数据地址总线的控制权；
- 第（3）步过程循环直到数据传输完毕。

（4）当一批数据输入完成，DMA控制器向CPU发出中断信号，请求中断运行进程并转向执行中断处理程序。

（5）中断程序首先保存被中断进程的现场，唤醒等待输入数据的那个进程，使其变成就绪状态，恢复现场，返回被中断的进程继续执行。

（6）当进程调度程序调度到要求输入数据的那个进程时，该进程就到指定的内存地址中读取数据进行处理。

如果有更复杂的IO需求怎么办?

- 方案4: 可以交办复杂任务，完成后汇报!
  - 引入通道（channel）方式
  - 通道具有简单的CPU功能，可编程，可管理多个设备同时工作。从而真正实现了CPU与外部设备的并行工作。

通道控制方式的工作过程：

（1）当一个进程要求输入输出数据时，CPU根据请求形成有关通道程序，然后执行输入输出指令启动通道工作；

（2）申请输入输出数据的进程放弃CPU进入阻塞等待状态，等待数据输入输出工作的完成，于是进程调度程序调度其他进程运行；

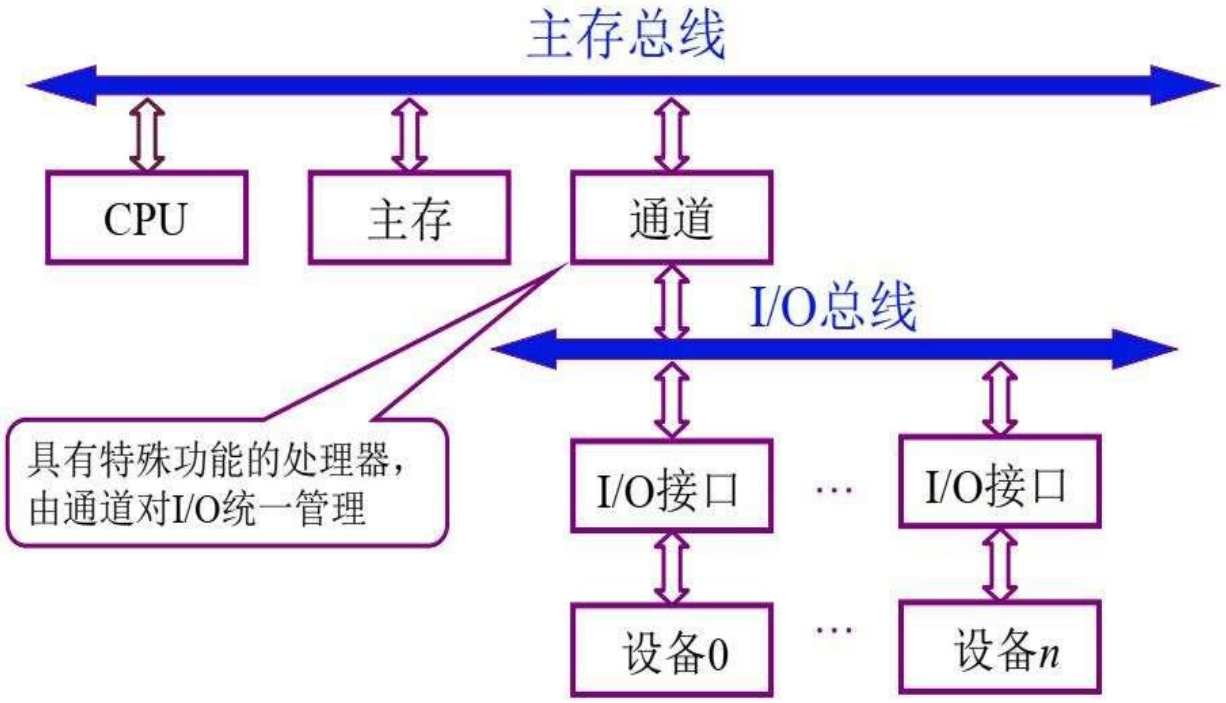
（3）通道开始执行CPU放在主存中的通道程序，独立负责外设与主存的数据交换；

（4）当数据交换完成后，通道向CPU发出中断信号，中断正在运行的进程，转向中断处理程序；

（5）中断处理程序首先保护被中断进程的现场，唤醒申请输入输出的那个进程，使其变为就绪状态，关闭通道，然后恢复现场，返回被中断的进程继续运行；

（6）当进程调度程序调度到申请输入输出数据的那个进程时，该进程就到指定的内存地址中进行数据处理。

10.4 缓冲技术



10.4 缓冲技术

- 缓冲的目的：(1)解决CPU和外设速度不匹配的矛盾，(2)提高CPU与外设之间的并行性，(3)减少对CPU的中断频率；(4)提升进程的执行效率。
- 缓冲技术的实现方法：硬件缓冲、软件缓冲
  - ✓**软件缓冲：**借助操作系统的管理，在内存中专门开辟若干单元作为缓冲区。
  - ✓**硬件缓冲：**利用专门的硬件寄存器作为缓冲区，一般由外设自带的专用寄存器构成。

    例如：Printer、CD-ROM等

缓冲技术–软件缓冲的4种实现方法

- **单缓冲，双缓冲，环形缓冲，缓冲池**

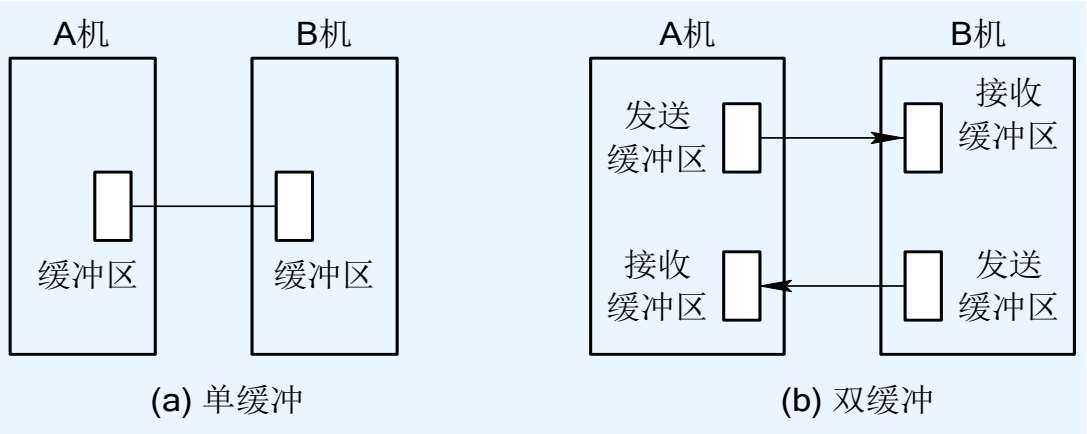
1.单缓冲：在内存中开辟一个固定大小的区域作为缓冲区

- 外设和CPU交换数据时，先将被交换的数据写入缓冲区，然后再由需要数据的CPU或外设从缓冲区中取出。
- 该方式中，外设与CPU对缓冲区的操作是串行的。

2.双缓冲：在内存中设置2个大小相同的缓冲区。

- 外设和CPU可以交替使用这2个缓冲区，从而在一定程度上实现并行交换数据。

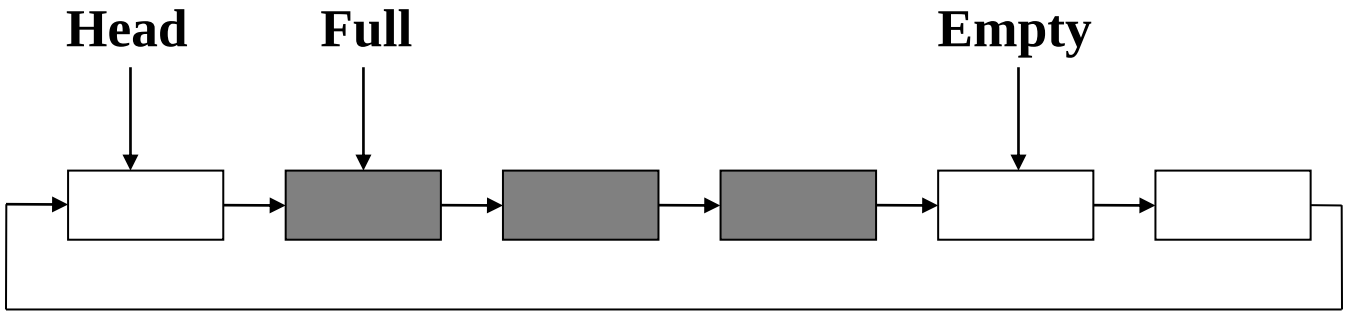
双机通信时缓冲区的设置



缓冲技术–软件缓冲的4种实现方法

- **单缓冲，双缓冲，环形缓冲，缓冲池**

3.环形缓冲：在内存中设置大小相等的多个缓冲区，并将它们链接称为一个环形链表。



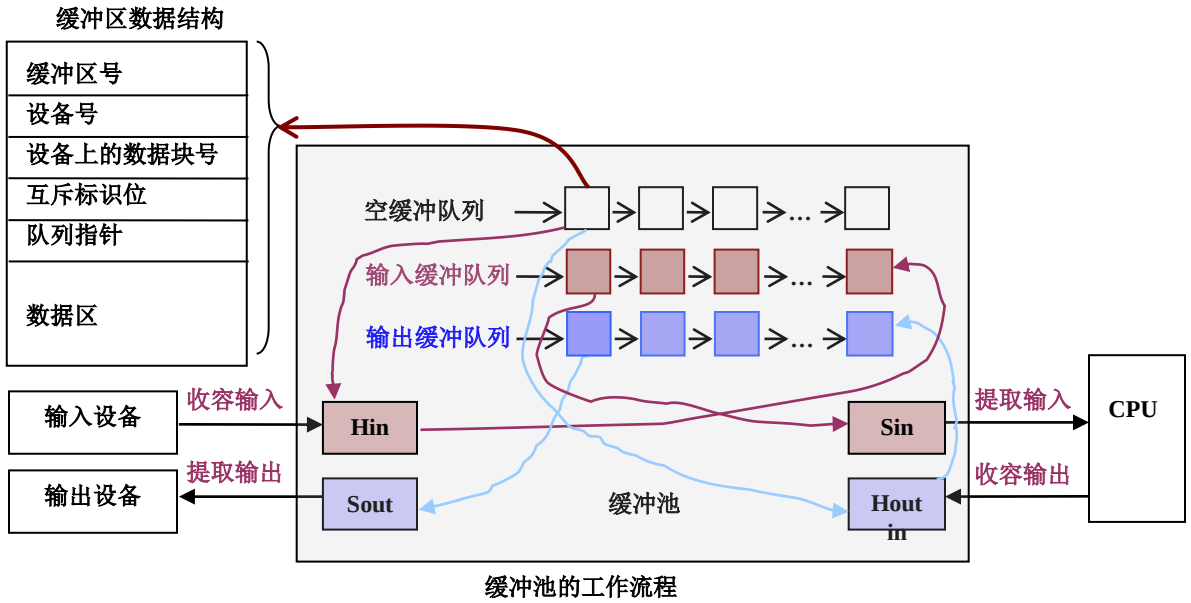
- **Head**一直指向缓冲区链表的第一个缓冲区；
- **Full**一直指向缓冲区链表中的第一个存满数据的缓冲区；
- **Empty**一直指向缓冲区链表中的第一个空白的缓冲区。
- 初始化时：**Head=Full=Empty**，整个缓冲区链表为空；
- 使用过程中：当**Full=Empty <==>** 整个缓冲区链表为空。

缓冲技术–软件缓冲的4种实现方法

- **单缓冲，双缓冲，环形缓冲，缓冲池**

4.缓冲池： 缓冲池是有多个大小相同的缓冲区组成

- 池中的缓冲区是系统公共资源，所有进程均可以共享
- 池由系统管理程序统一管理，负责分配、回收工作
- 池中每个缓冲区既可以用于输入数据，也可以用以输出数据

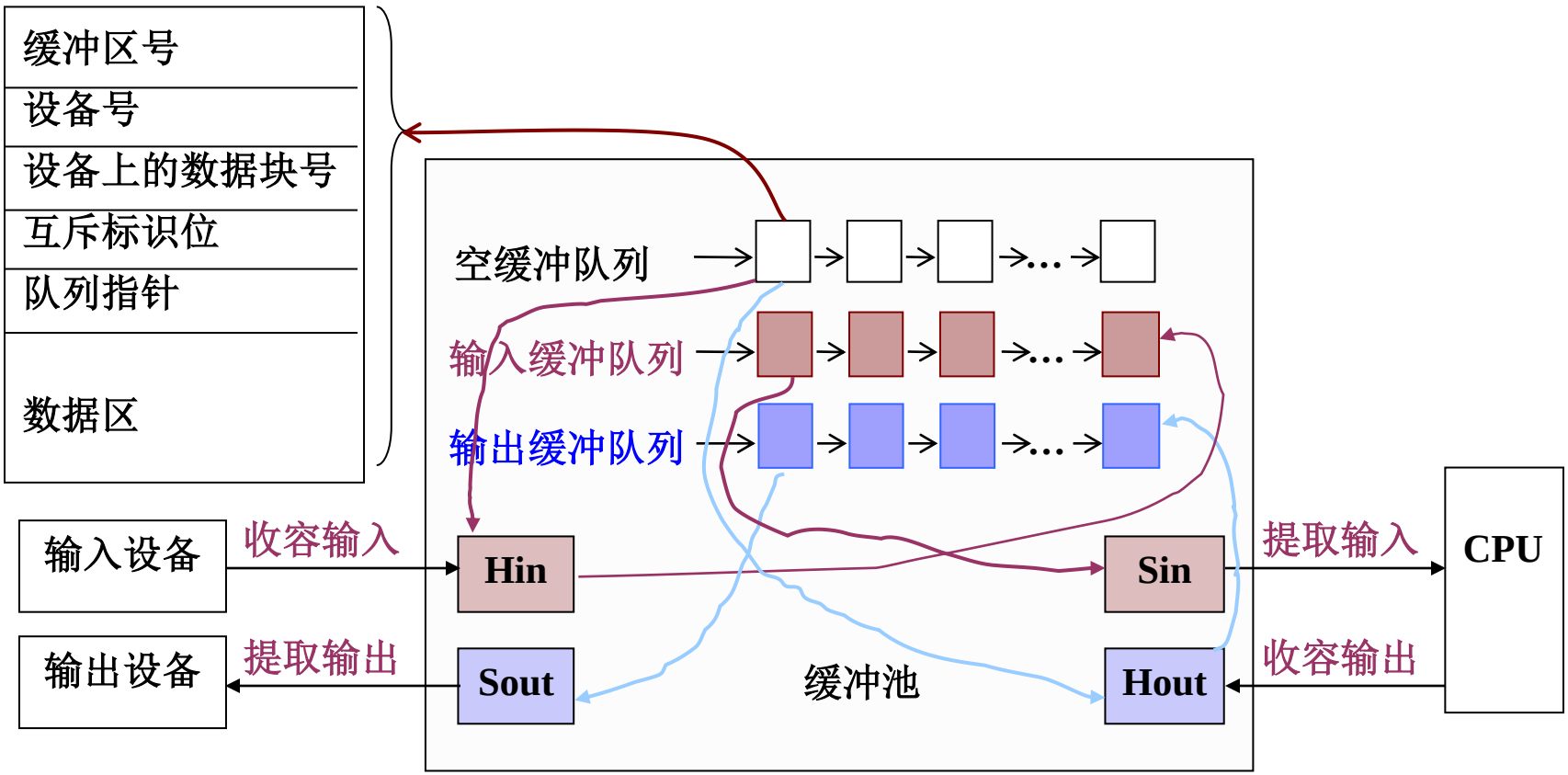


缓冲池的工作流程

缓冲技术-软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，**缓冲池**

缓冲区数据结构



缓冲池的工作流程

缓冲技术-软件缓冲的4种实现方法

- **单缓冲，双缓冲，环形缓冲，缓冲池**

缓冲池的工作流程：

（1）当**输入设备**需要**进行数据输入**时，则从空缓冲队列的队首**取下一个空缓冲区**，将它作为收容输入工作缓冲区，当它**被输入装满数据后，则被链接到输入缓冲队列的队尾**；

（2）当某**进程**需要从缓冲池读入数据时，则**从输入缓冲队列的队首**取一个缓冲区作为**提取输入工作缓冲区**，该进程从中提取数据，取完后，则**将该缓冲区链接到空缓冲区队列的队尾**；

（3）当某进程需要输出数据到缓冲池时，则从空缓冲队列的队首取下一个空缓冲区，将它作为收容输出工作缓冲区，该进程向该缓冲区中存放数据，当它被装满数据后，则被链接到输出缓冲队列的队尾；

（4）当输出设备需要进行数据输出时，则从输出缓冲队列的队首取一个缓冲区作为提取输出工作缓冲区，并从中提取数据输出，取完后，则将该缓冲区链接到空缓冲区队列的队尾。

缓冲技术–软件缓冲的4种实现方法

- Linux系统为了提高读写磁盘的效率，会先将数据放在一些内存buffer中。
- 在写磁盘时并不是立即将数据写到磁盘中，而是先写入这些buffer中了。
- 此时如果重启系统，就可能造成数据丢失。
- sync命令用来flush文件系统buffer，这样数据才会真正的写到磁盘中，并且buffer才能够释放出来。
- 所以常常会在写磁盘后输入sync命令来将数据真正的写入磁盘。

缓冲技术–软件缓冲的4种实现方法

- 如果不去手动的输入sync命令来真正的去写磁盘，linux系统也会有两种写磁盘的时机：
  - 1) kflush内核线程周期性的去写磁盘；
  - 2) buffer已满不得不写。

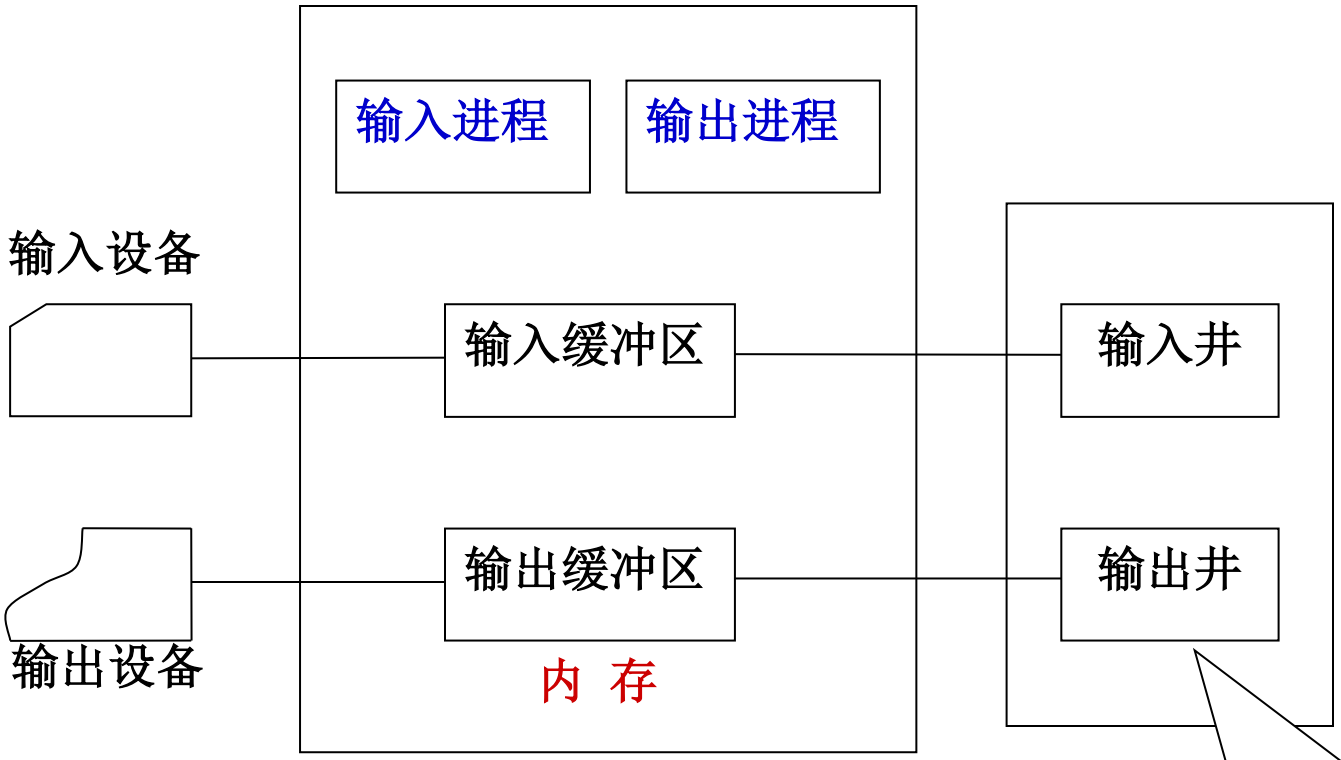
10.4 缓冲技术 – SPOOLING

- SPOOL – Simultaneous Peripheral Operation On Line外部设备联机并行操作，又称假脱机操作。
- SPOOL是操作系统中采用的一项将独占设备改造成共享设备的技术。
  - 一个虚拟设备
  - 一个资源转换技术

（用空间入、输出等换取CPU和IO时间）

- 实现方法：截获向某独享设备输出的数据，暂时保存到内存缓冲区或磁盘文件中，并进行排队，之后逐个输出到外设上
- 实现这一技术的软、硬件系统称为SPOOL系统，或假脱机系统，或SPOOLING系统。

10.4 缓冲技术 – SPOOLING



SPOOLING系统的组成

输入井和输出井：

- 在磁盘上开辟出来的2个存储区域
- 输入井用于收容I/O设备的输入数据
- 输出井用于收容I/O设备的输出数据

10.4 缓冲技术 – SPOOLING

共享打印机

- 当用户进程请求打印输出时，SPOOLing系统为用户进程做：
  - ✓ 由输出进程在输出井中为之申请一个空闲磁盘块区，并将要打印的数据送入其中；
  - ✓ 输出进程再为用户进程申请一张空白的用户请求打印表，并将用户的打印要求填入其中，再将该表挂到请求打印队列上。

10.4 缓冲技术 – SPOOLING

SPOOLing系统的特点

① 提高了I/O的请求速度（只是请求接纳）

② 将独占设备改造为共享设备（本质依旧串行）

③ 实现了虚拟设备功能（能力提升）

10.5 I/O软件层次

I/O系统给用户提供一个什么样的软件抽象视图

看一段linux操纵外设的程序

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

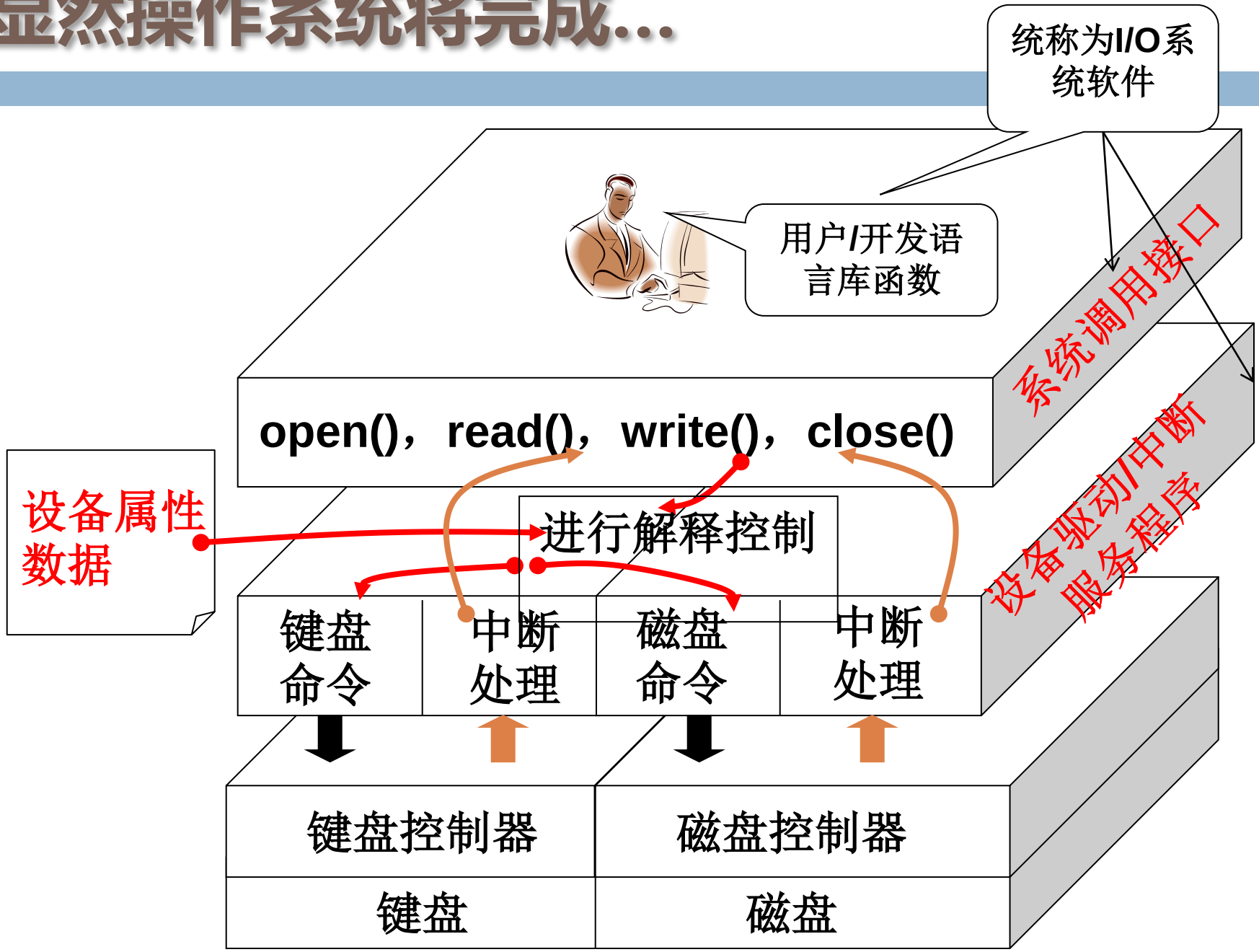
- (1) 不论什么设备都是open, read, write, close

  操作系统为用户提供统一的接口!

- (2) 不同的设备对应不同的文件(设备文件)

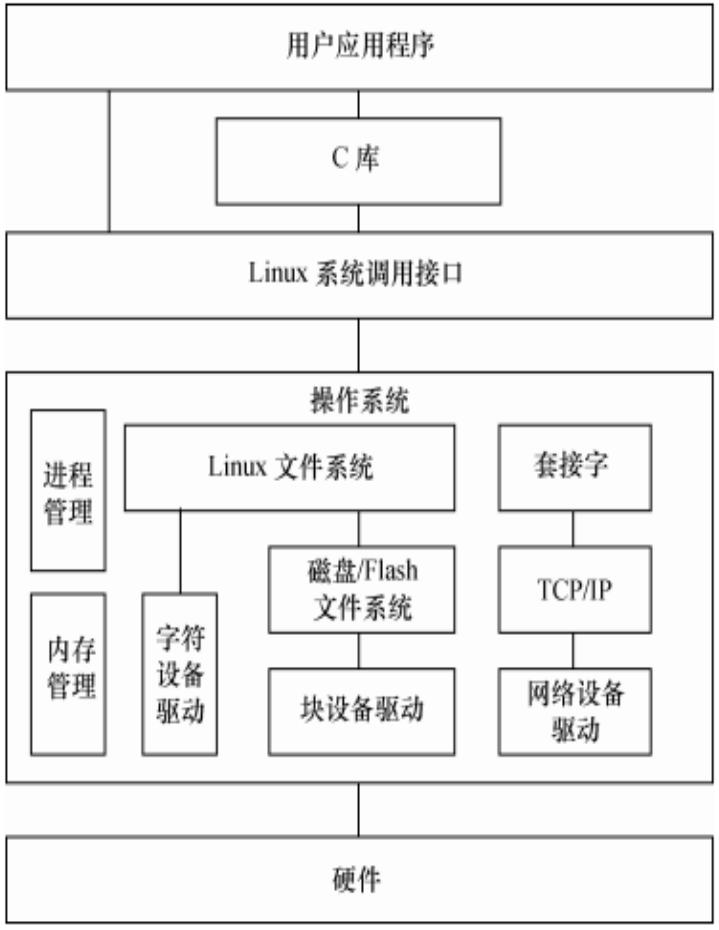
  设备文件中存放了设备的属性!

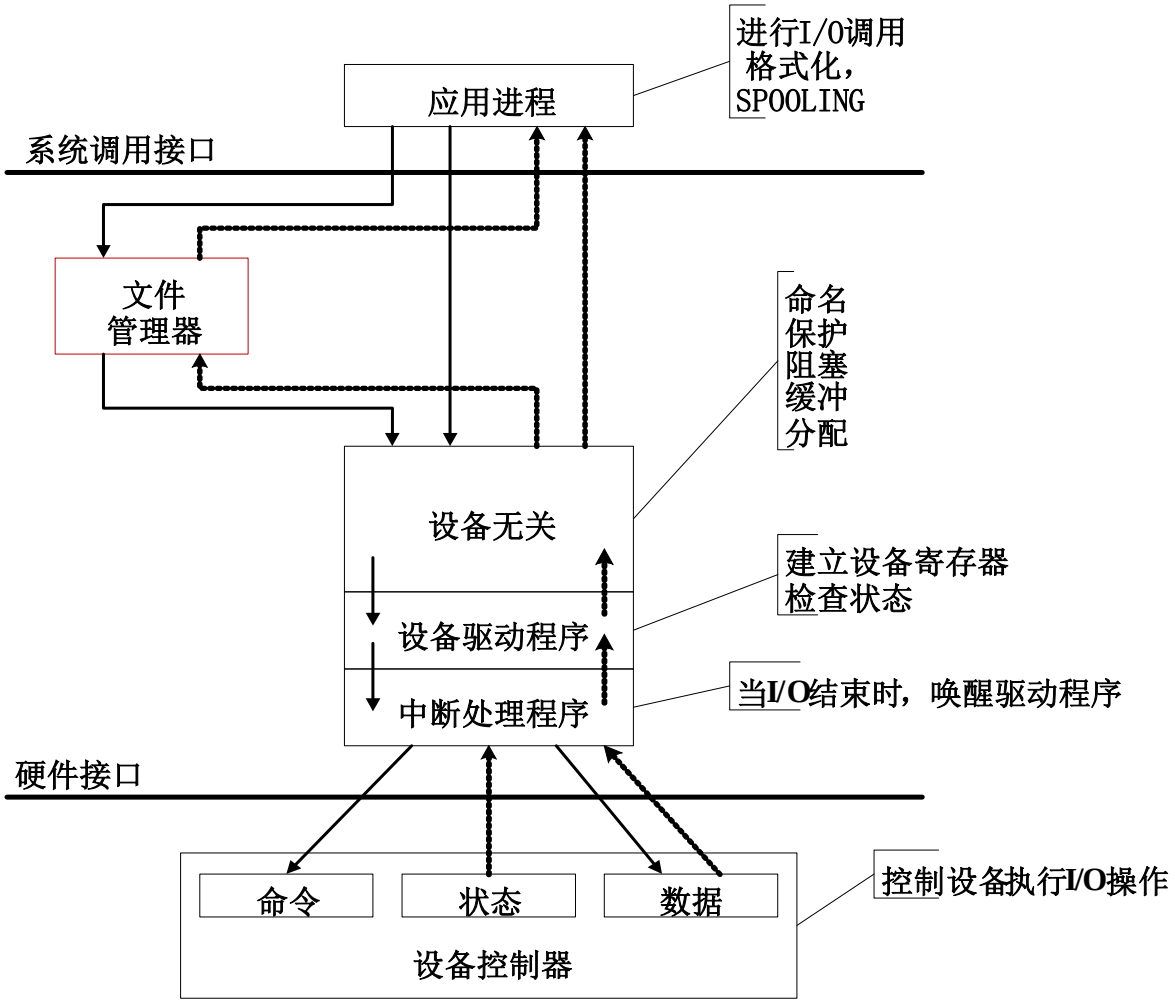
显然操作系统将完成...


统称为I/O系统软件
用户/开发语言库函数
系统调用接口
open()，read()，write()，close()
设备属性数据
进行解释控制
设备驱动/中断服务程序
键盘命令
中断处理
磁盘命令
中断处理
键盘控制器
磁盘控制器
键盘
磁盘

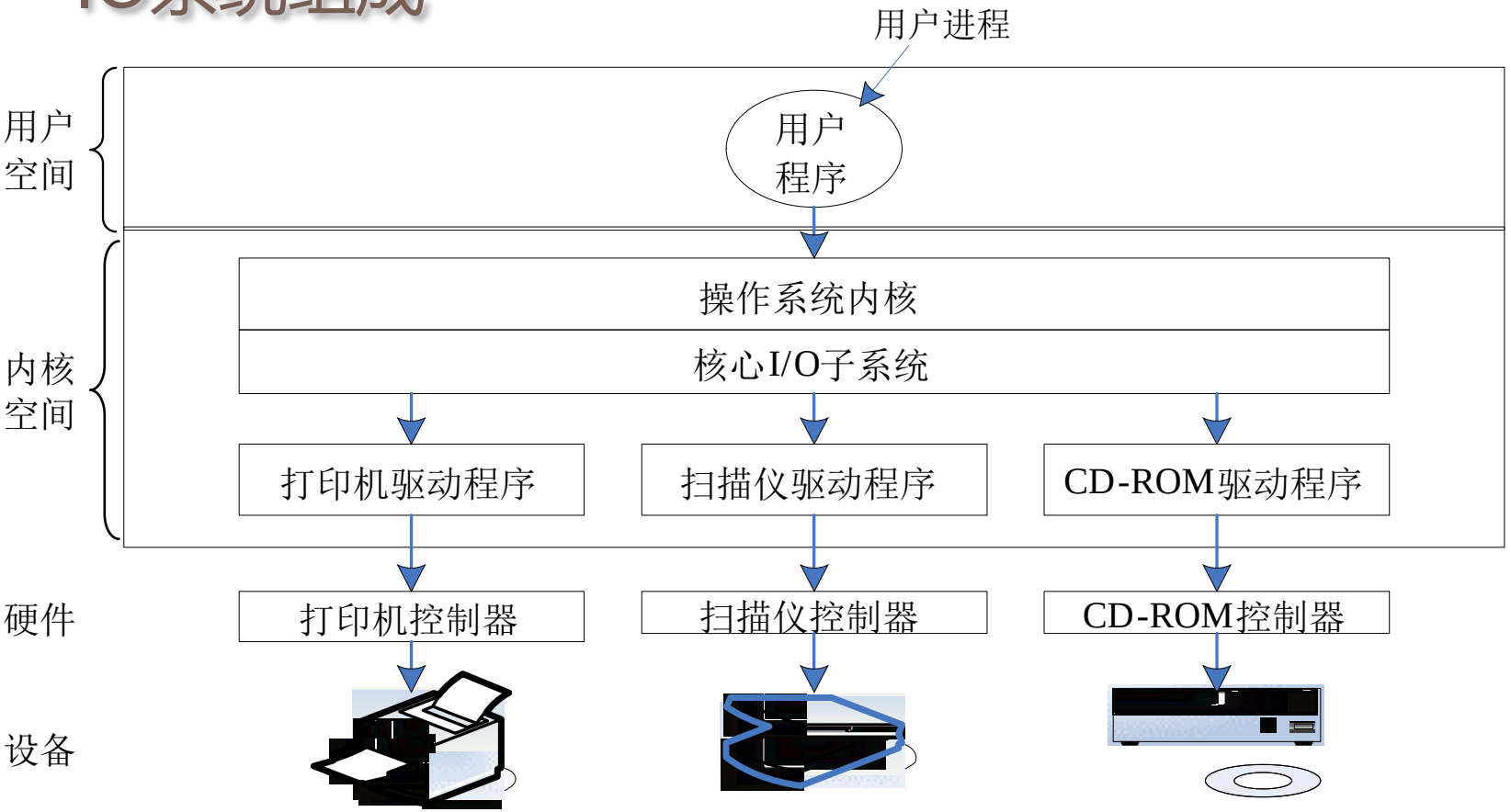
10.5 I/O软件层次

IO系统软件组成





IO系统组成



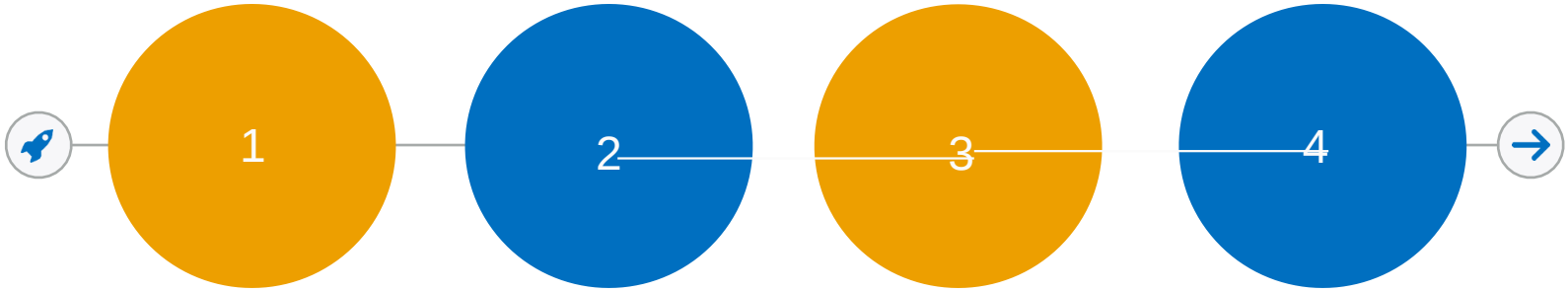
设备驱动程序接口

10.5 I/O软件层次

I/O软件功能需求

提供使用I/O设备的用户接口：
命令接口和编程接口。

设备的访问和控制： 包括
并发访问和差错处理。



设备分配和释放：使用
设备前，需要分配设备
和相应控制器。

I/O缓冲和调度：目标
是提高I/O访问效率

I/O软件组织成以下4个层次

① 用户空间的I/O软件

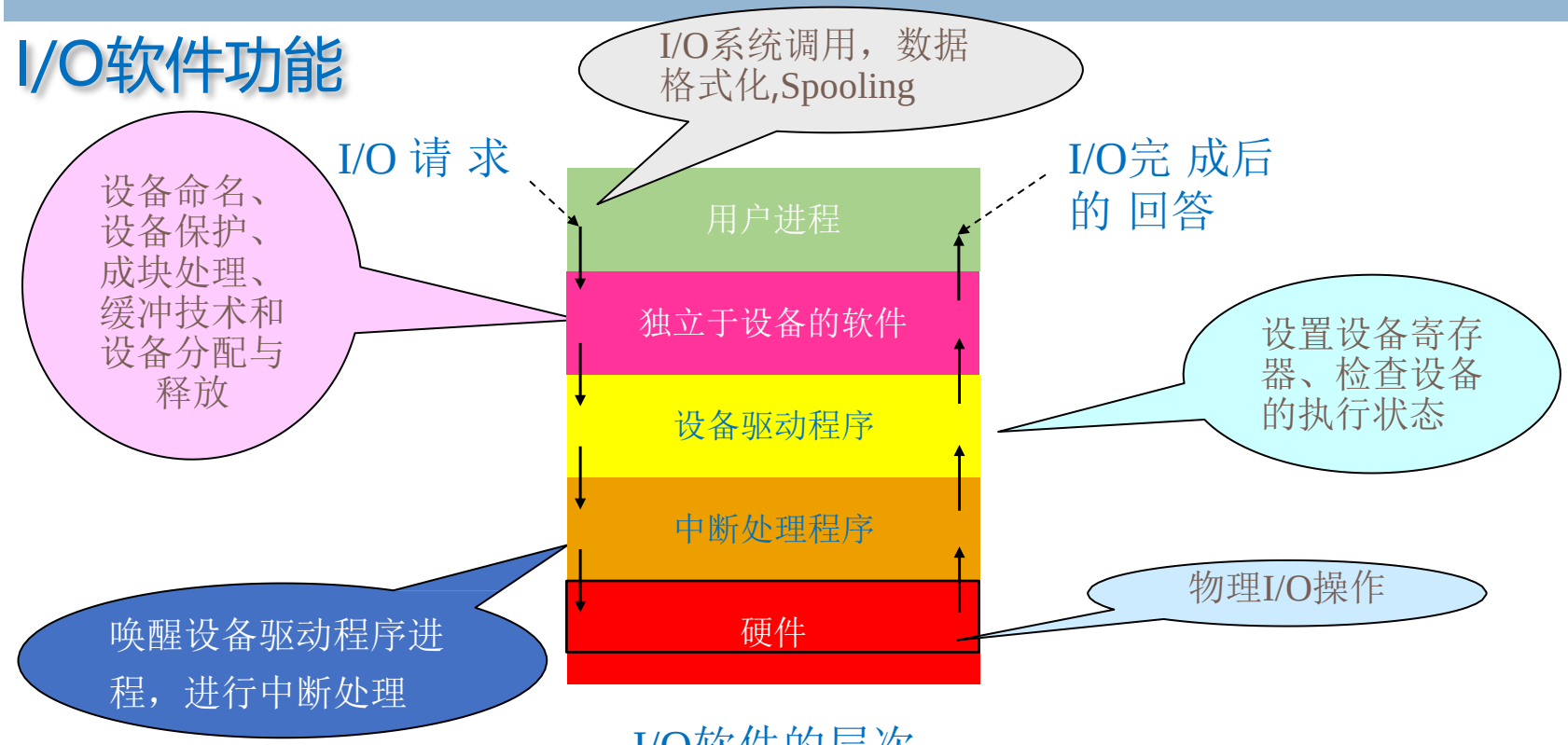
② 与设备无关的I/O软件(设备独立软件)

③ 设备驱动程序

④ 中断处理程序

10.5 I/O软件层次

I/O软件功能



I/O软件的层次

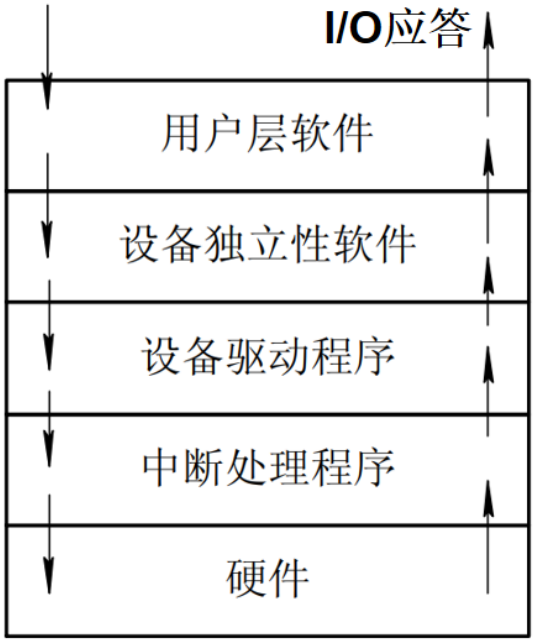
- 大部分I/O软件都包含在OS内核之中，但有一小部分I/O软件是由与用户程序连接在一起的库函数构成，它们运行在OS内核之外
- I/O系统调用，通常由库函数实现。例如:
  - count=write(fd, buffer, nbytes); 显然，write函数是I/O系统的组成部分。

10.5 I/O软件层次

1、I/O软件介绍

I/O软件涉及的面非常宽，往下与硬件有着密切的关系，往上又与用户直接交互，它与进程管理、存储器、文件管理等都存在着一定的联系。

- I/O软件的总体目标
  - 有效性：性能、吞吐量
  - 通用性：设备多、杂；以一致的方式来管理设备



I/O系统的4层次及功能

10.5 I/O软件层次

2、 I/O软件设计的原则

- 消除或屏蔽设备硬件内部的低级处理过程，
- 为用户提供一个简便、易用、抽象的逻辑设备接口，
- 保证用户安全、方便地使用各类设备。

□需要层次化、模块化的方法，按分层的思想构造软件

- 较低层的软件要使较高层的软件独立于硬件
- 较高层的软件要向用户提供一个友好、规范、清晰的界面。

10.5 I/O软件层次

3、I/O软件设计的具体目标

① 设备独立性：屏蔽设备的具体细节，向高层软件提供抽象的逻辑设备，并完成逻辑设备与具体物理设备的映射。

② 统一命名：对各类设备采取预先设计的、统一的逻辑名称进行命名，所有软件都以逻辑名称访问设备。这种统一命名与具体设备无关：同一个逻辑设备的名称，在不同的情况下可能对应于不同的物理设备。

③ 异步传输：启动IO后CPU不被影响，直到中断。

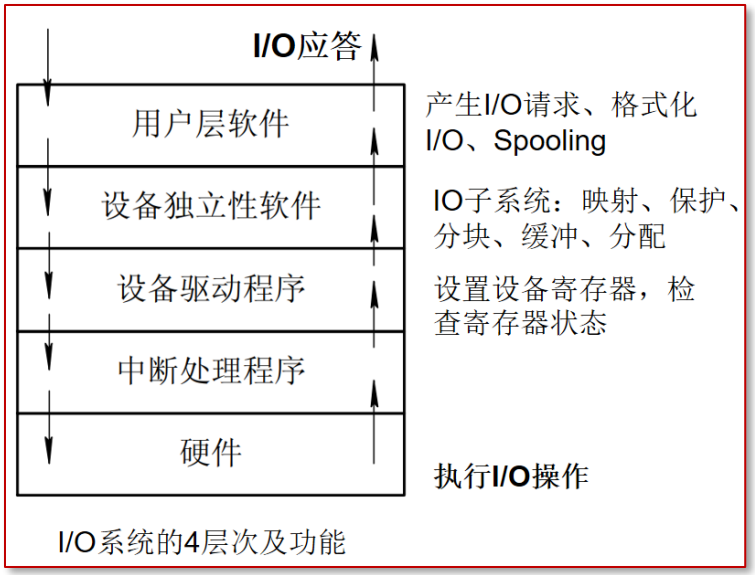
④ 出错处理：错误多数是与设备紧密相关的，在低层软件能够解决的错误就不让高层软件感知，只有低层软件解决不了的错误才通知高层软件解决。

⑤ 设备共享与独占：独占设备影响了系统效率的发挥。 在设备管理中，仍然可以借助于磁盘，把只能独享的设备变为共享。

10.5 I/O软件层次

I/O软件层次——设备驱动程序

- 操作系统以统一的方式,对待不同的I/O 设备
- 具体的差别被称为设备驱动程序的内核模块所封装
  - 所有与设备相关的代码放在设备驱动程序中。
  - 它是I/O进程与设备控制器之间的通信程序，因为它常以进程的形式存在，故也可以称为设备驱动进程。
  - 由于驱动程序与设备硬件密切相关，故应为每一类设备配置一种驱动程序，或为一类密切相关的设备配置一个驱动程序。



I/O系统的4层次及功能

10.5 I/O软件层次

设备驱动程序的功能

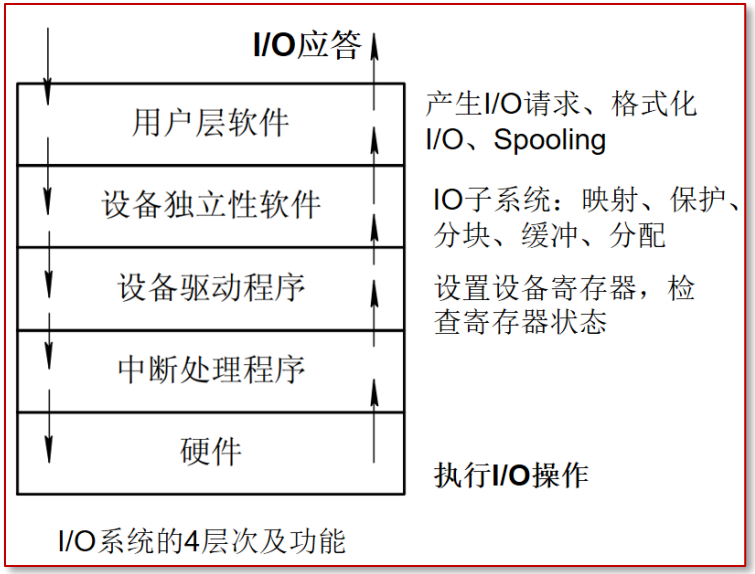
① 将接收到的来自它上一层的与设备无关的命令和参数，并将命令中的抽象要求转换为具体要求，例如，将磁盘块号转换为磁盘的盘面、磁道号及扇区号。

② 检查用户I/O请求的合法性，了解I/O设备的状态，传递有关参数、设置设备的工作方式。

③ 发出I/O命令，如果设备空闲，便立即启动I/O设备去完成指定的I/O操作；如果设备处于忙碌状态，则将请求者的请求块挂在设备队列上等待。

④ 及时响应控制器或通道发来的中断请求，根据其中断类型调用中断处理程序处理。

⑤ 对于有通道的计算机系统，驱动程序还应能根据用户I/O 请求，自动地构成通道程序。



I/O系统的4层次及功能

10.5 I/O软件层次

设备驱动程序的处理过程

1) 将抽象要求转换为具体要求

由于用户及上层软件，对设备控制器的具体情况毫无了解，因而只能向它发出抽象的要求(命令)， 但这些命令无法传送给设备控制器。因此，就需要将这些抽象要求转换为具体要求。

例如，将抽象要求中的盘块号转换为磁盘的盘面、 磁道号及扇区。这一转换工作只能由驱动程序来完成，因为在OS中只有驱动程序，才同时了解抽象要求和设备控制器中的寄存器情况；也只有它才知道命令、 数据和参数应分别送往哪个寄存器。

设备驱动程序的处理过程

2) 检查I/O请求的合法性

对于任何输入设备，都是只能完成一组特定的功能，若该设备不支持这次的I/O请求，则认为这次I/O请求非法。

例如，用户试图请求从打印机输入数据，显然系统应予以拒绝。此外，还有些设备如磁盘和终端，它们虽然都是既可读又可写的，但若在打开这些设备时规定的是读，则用户的写请求必然被拒绝。

设备驱动程序的处理过程

3) 读出和检查设备的状态

在启动某个设备进行I/O操作时，其前提条件应是该设备正处于空闲状态。因此在启动设备之前，要从设备控制器的状态寄存器中，读出设备的状态。

例如，为了向某设备写入数据，此前应先检查该设备是否处于接收就绪状态，仅当它处于接收就绪状态时，才能启动其设备控制器，否则只能等待。

设备驱动程序的处理过程

4) 传送必要的参数

对于许多设备，特别是块设备，除必须向其控制器发出启动命令外，还需传送必要的参数。

例如在启动磁盘进行读/写之前，应先将本次要传送的字节数和数据应到达的主存始址，送入控制器的相应寄存器中。

5) 工作方式的设置

有些设备可具有多种工作方式，典型情况是利用RS-232接口进行异步通信。

在启动该接口之前，应先按通信规程设定参数：波特率、奇偶校验方式、停止位数目及数据字节长度等。

10.5 I/O软件层次

设备驱动程序的处理过程

6) 启动I/O设备

- 在完成上述各项准备工作之后，驱动程序可以向控制器中的命令寄存器，传送相应的控制命令。
  - 对于字符设备，若发出的是写命令，驱动程序将把一个数据传送给控制器；若发出的是读命令，则驱动程序等待接收数据，并通过从控制器中的状态寄存器读入状态字的方法，来确定数据是否到达。
- 驱动程序发出I/O命令后，基本的I/O操作是在设备控制器的控制下进行的。
  - I/O操作所要完成的工作较多，需要一定的时间，如读/写一个盘块中的数据，此时驱动(程序) 进程把自己阻塞起来，直到中断到来时才将它唤醒。

10.5 I/O软件层次

设备驱动程序的框架

(1) 设备驱动程序与外界的接口

对设备驱动程序与外界的接口需要进行严格的定义，主要体现在以下三个方面：

- 设备驱动程序与操作系统内核的接口;
- 设备驱动程序与系统引导的接口;
- 设备驱动程序与设备的接口。

10.5 I/O软件层次

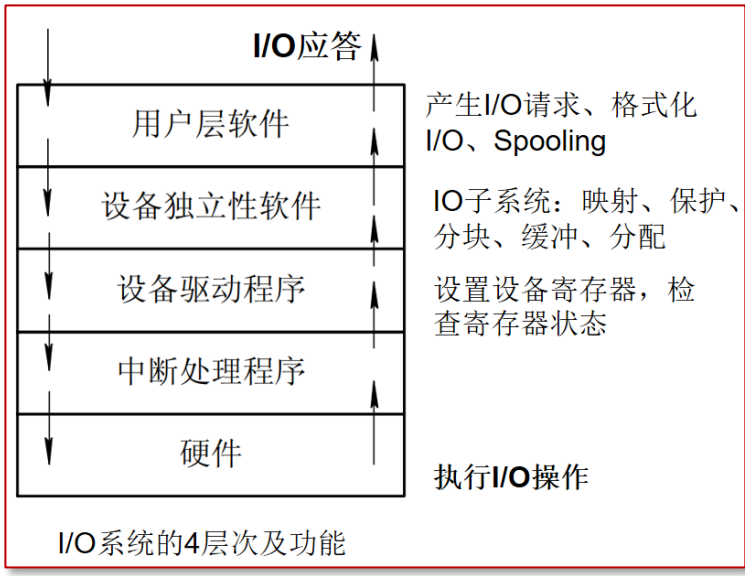
与设备无关的I/O软件（IO子系统）

设备无关的概念（设备独立性）(Device Independence)

- 为了提高OS的适应性和可扩展性，应用程序独立于具体使用的物理设备。

为了实现设备无关性而引入了逻辑设备和物理设备：

- 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，使用物理设备名称。
- 类比逻辑地址和物理地址



I/O系统的4层次及功能

10.5 I/O软件层次

系统的逻辑设备表

逻辑设备号	物理设备号	驱动程序地址
1	7	20420
2	7	20420
3	2	20E00
4	4	1FC10
6	1	20D02
7	7	20420
15	10	1FC10
16	11	1FC120
…	…	……

10.5 I/O软件层次

使用逻辑设备名称的好处

1) 设备分配时的灵活性

- 当应用程序(进程)以物理设备名称来请求使用指定的某台设备时，如果该设备已经分配给其他进程或正在自检，而此时尽管还有几台其它的相同设备正在空闲，该进程却仍阻塞。
- 但若进程能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，仅当所有此类设备已全部分配完毕时，进程才会阻塞。

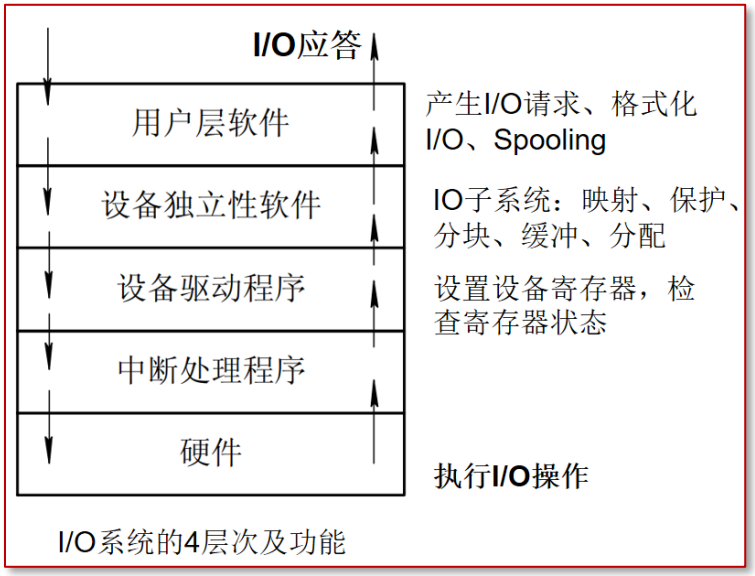
2) 易于实现I/O重定向

- 所谓I/O重定向，是指用于I/O操作的设备可以更换(即重定向)，而不必改变应用程序。
  - 例如，我们在调试一个应用程序时，可将程序的所有输出送往屏幕显示；而在程序调试完后，如需正式将程序的运行结果打印出来，此时便须将I/O重定向的数据结构——逻辑设备表中的显示终端改为打印机， 而不必修改应用程序。I/O重定向功能具有很大的实用价值，现已被广泛地引入到各类OS中。

10.5 I/O软件层次

设备无关的I/O软件（内核IO子系统）

- 驱动程序是一个与硬件(或设备)紧密相关的软件。
- 为了实现设备独立性，必须再在驱动程序之上设置一层软件，称为设备无关软件，又称为I/O子系统。



I/O系统的4层次及功能

设备无关软件的主要功能可分为以下两个方面：

1) 执行所有设备的公有操作。
2) 向用户层软件提供统一接口。

10.5 I/O软件层次

设备无关软件的主要功能

1. 执行所有设备的公有操作
   - 设备命名、名字映射：将逻辑设备名映射为物理设备名，进一步可以找到相应物理设备的驱动程序
   - 保护：对设备进行保护，禁止用户直接访问设备
   - 逻辑块、缓冲管理：即对字符设备和块设备的缓冲区进行有效的管理， 以提高I/O的效率；
   - 分配回收：对独立设备的分配与回收
   - 差错控制：处理设备驱动程序无法处理的错误
2. 向用户层软件提供统一接口
   无论何种设备，它们向用户所提供的接口应该是相同的，如统一的Read和Write操作

10.5 I/O软件层次

I/O软件层次——用户空间的I/O软件

大部分的I/O软件都在操作系统内部，但仍有一小部分在用户层，包括与用户程序链接在一起的库函数，以及完全运行于内核之外的一些程序。

- 用户层软件必须通过一组系统调用,来取得操作系统服务。
- 在现代的高级语言以及C语言中，通常提供了与各系统调用一一对应的库函数。这些库函数与调用程序连接在一起，包含在运行时装入在内存的二进制程序中，如C语言中的库函数write等，显然这些库函数的集合也是I/O系统的组成部分。
- 但在许多现代操作系统中，系统调用本身已经采用C语言编写，并以函数形式提供，所以在使用C语言编写的用户程序中，可以直接使用这些系统调用。

10.5 I/O软件层次

看一段操纵外设的程序

第一类用户空间I/O软件，用户程序与库函数

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

- (1) 不论什么设备都是open, read, write, close

  操作系统为用户提供统一的接口!

- (2) 不同的设备对应不同的文件(设备文件)

  设备文件中存放了设备的属性!

10.5 I/O软件层次

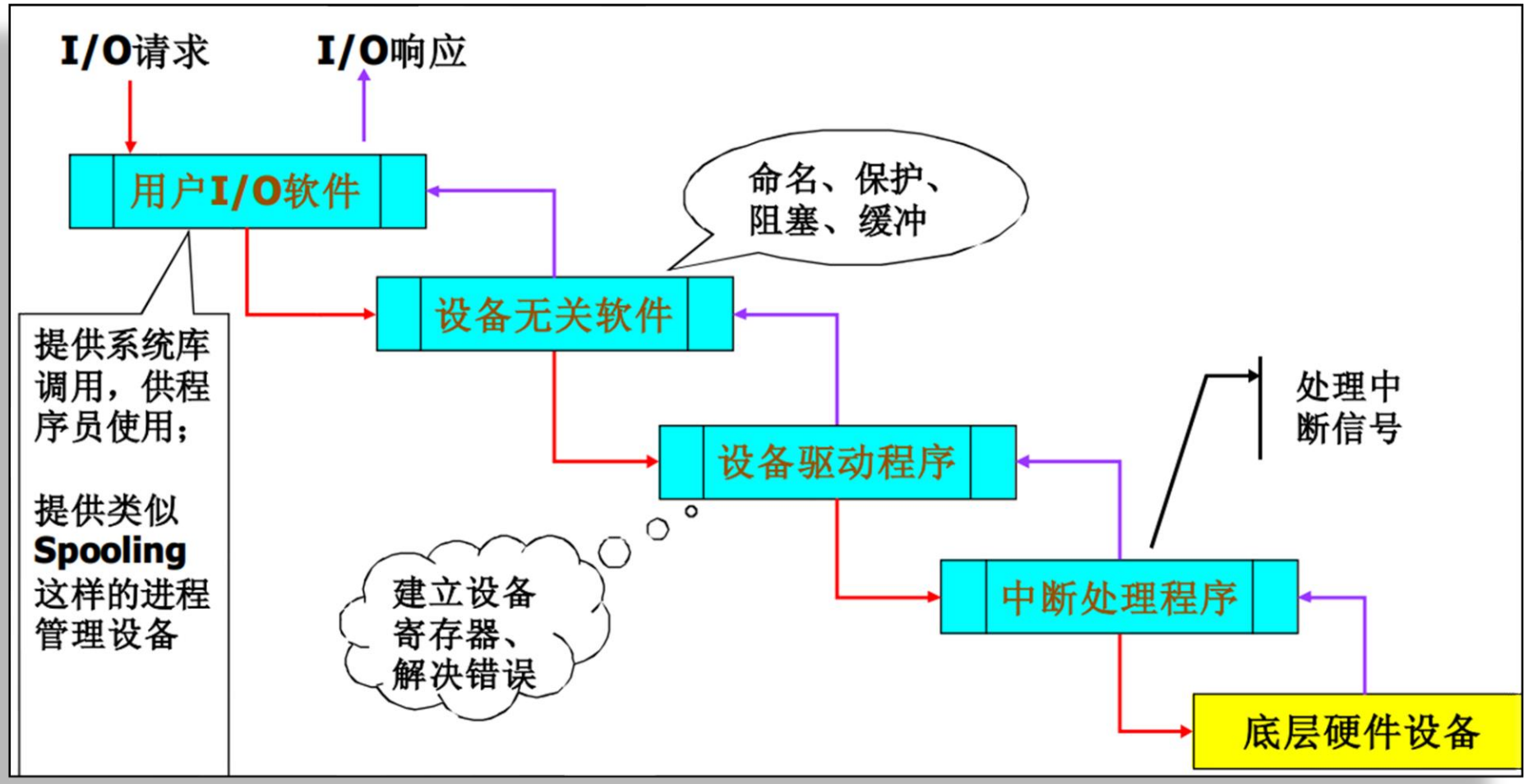
- 第二类用户空间I/O软件

  Spooling (Simultaneous Peripheral Operating On Line)

  - Spooling系统是多道程序设计系统中，处理独占I/O设备的一种方法。

- 联机：程序的I/O工作，直接通过设备进行。
- 脱机：提交了I/O之后，不直接通过设备进行，进程不用等待实际的输入输出。利用专门的外围控制机，将低速I/O设备上的数据传送到高速磁盘上；或者相反。

10.5 I/O软件层次


I/O请求
I/O响应
用户I/O软件
命名、保护、
阻塞、缓冲
设备无关软件
提供系统库
调用，供程
序员使用；
提供类似
Spooling
这样的进程
管理设备
设备驱动程序
处理中
断信号
建立设备
寄存器、
解决错误
中断处理程序
底层硬件设备

10.6 设备分配

1、设备分配方法

设备分配2种方式：

- 静态分配方式　在用户进程创建时，OS便一次性地把进程运行所要求的全部设备都分配给它，并由该进程占有，直到进程撤消。

  不会死锁，但设备利用率极其低下。

- 动态分配方式　在进程执行过程中，随时根据需要，向系统提出设备请求，由系统依据一定算法给进程分配设备，用户进程用完设备，即予释放。

  有利于提高设备利用率，但分配不当即有死锁可能。

10.6 设备分配

动态分配算法：

- 先来先服务
  - 对于多个请求某类设备的用户进程，系统按其发出请求的先后顺序，使它们在设备请求队列里排队，并把设备分配给队列的前列者。
- 优先级高者优先
  - 进入设备请求队列的进程，按优先级排队，优先级相同，则按到达的先后排，系统总是把设备分配给队列的首进程使用。

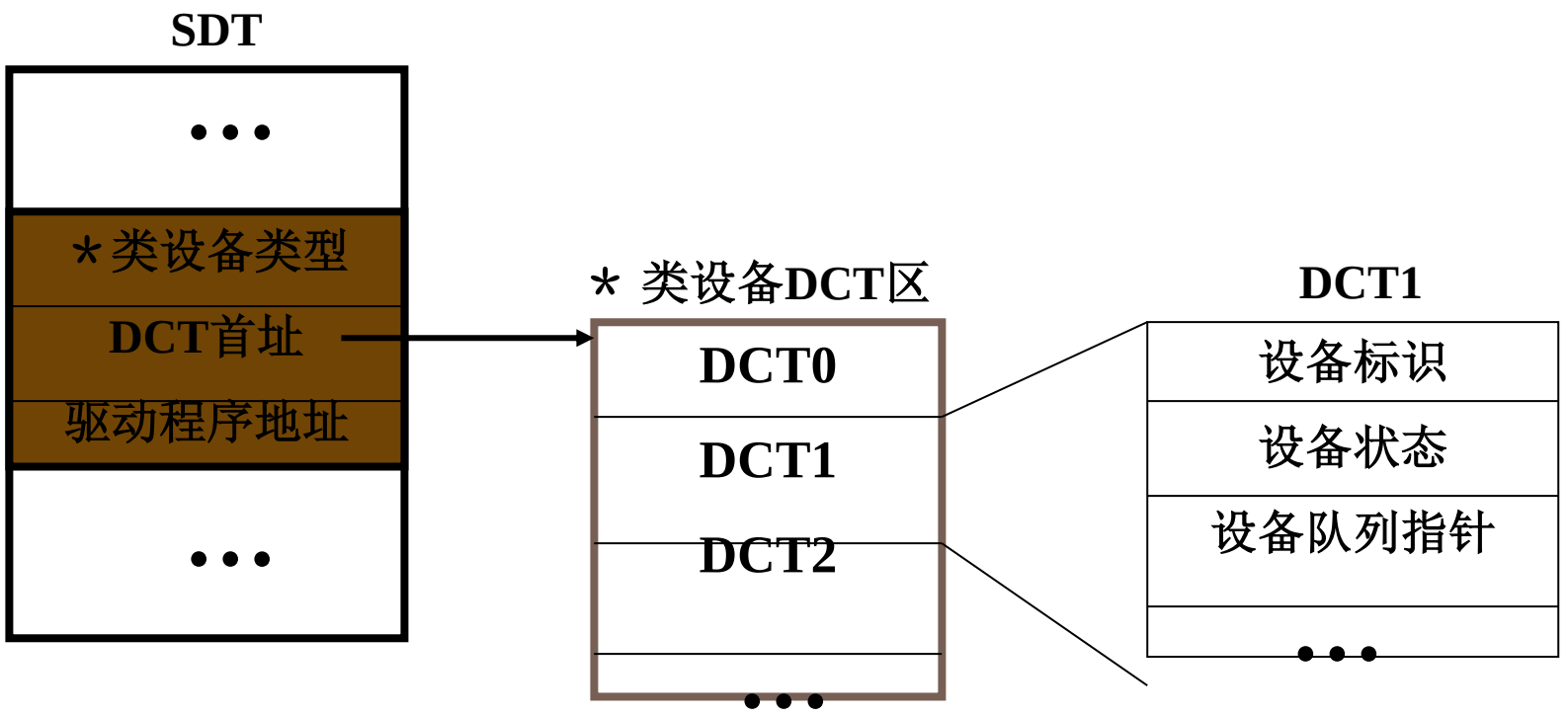
10.6 设备分配

2、设备管理采用的数据结构

- 系统设备表SDT（System Device Table）
  - 整个系统一张，记录了系统中所有外设，每类设备占一个表项。
- 设备控制表DCT（Device Control Table）
  - 系统中每台设备一个，其中随时记录了该设备的基本信息（设备状态、等待使用该设备的阻塞进程等）。

10.6 设备分配

- I/O过程中，OS从SDT内查得某类设备的DCT地址，然后再转到DCT取得具体设备信息。

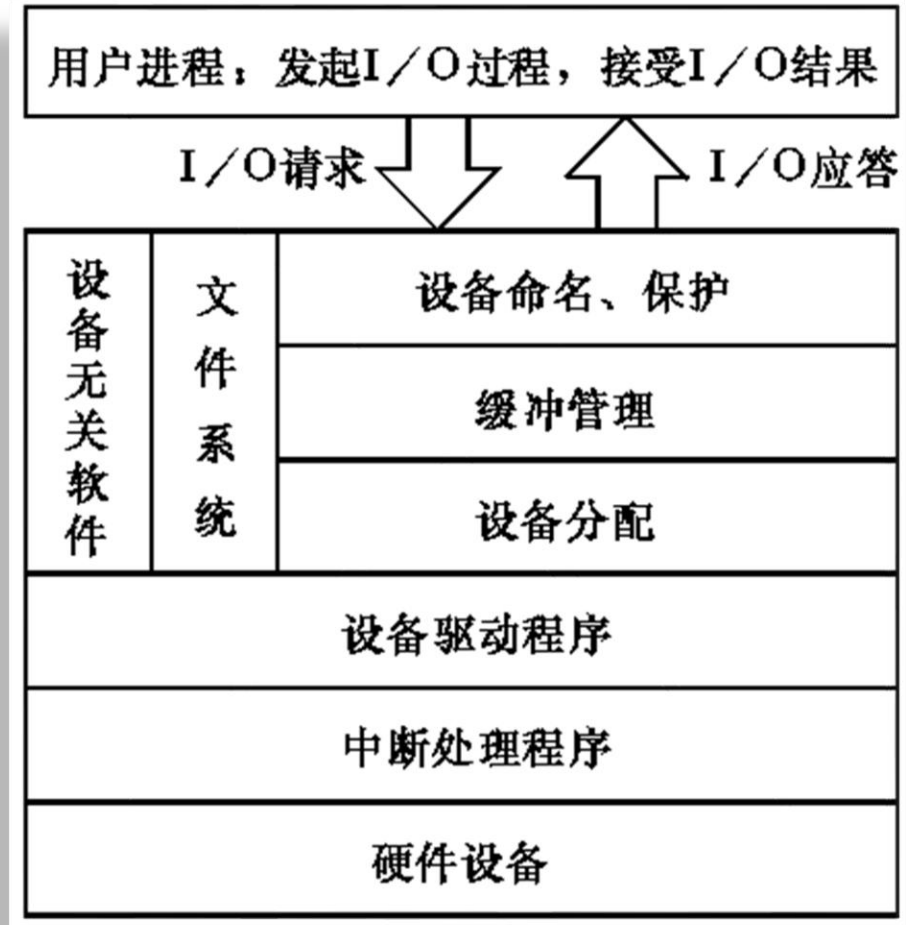


10.7 Linux IO

1、设备文件

每个设备都有自己的设备名，用户通过设备名来访问具体物理设备，设备名存放在/dev目录下。

- ✓ 硬盘：/dev/hda，/dev/sda
- ✓ 软盘：/dev/fd0
- ✓ 光盘：/dev/cdrom
- ✓ 鼠标：/dev/psaux(ps/2)
/dev/input/mice(USB)
- ✓ 打印机：/dev/lp
- ✓ 控制台：/dev/console
- ✓ 网卡：/dev/eth0



10.7 Linux IO

2、文件系统与设备驱动程序的接口

- Linux系统中，用户程序通过文件操作方式如打开、关闭、读写等来使用设备，由文件系统转入设备驱动程序。
- 在Linux中系统提供块设备开关表和字符设备开关表作为核心与设备驱动程序之间的接口。
- 每一种设备类型在表中占用一个表目，包含若干数据项，其中有一项为该类设备驱动程序入口地址，在系统调用时引导核心转向适当的驱动程序接口。

10.7 Linux IO

块设备开关表

操作主设备号	Open	Close	Strategy(read/write)
0	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址
1	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址

字符设备开关表

操作主设备号	Open	Close	Read	Write
0	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址
1	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址	驱动程序入口地址

3、块设备驱动的数据结构

①块设备表

对每一类块设备，分别设置块设备表，记录该类设备的相关信息。其内容包括：

- ✓忙标志：标志设备的忙闲状态，0表示空闲，1表示正忙
- ✓错误次数：指设备I/O出错次数
- ✓设备缓冲区队列头指针：分配给设备的缓冲区队列的头指针
- ✓设备缓冲区队列尾指针：分配给设备的缓冲区队列的尾指针
- ✓I/O请求队列头指针：请求该类设备I/O操作的请求块组成的队列的头指针
- ✓I/O请求队列尾指针：请求该类设备I/O操作的请求块组成的队列的尾指针

3、块设备驱动的数据结构

- ②I/O请求队列
  - ✓ 用户进程的I/O请求包括要求完成I/O操作的逻辑设备名、要求的操作、输送数据在内存中的起始地址、传送数据的长度，将这些信息组织成I/O请求块iorb。
  - ✓ 逻辑设备名相同（同一类设备）的I/O请求块构成一个队列，称为I/O请求队列。

4、字符设备驱动

- 字符设备的传送用一组专用的寄存器来实现。每种字符设备的控制器一般都有3个寄存器：控制寄存器接收CPU发送来的命令，控制设备的操作；状态寄存器保存设备的状态；数据寄存器暂存要传送的数据。

4、字符设备驱动

(1) 数据结构：对每类字符设备分别建立字符设备表，记录使用该类字符设备所需的各种信息。

(2) 终端驱动程序：字符设备种类繁多，其驱动程序各不相同，以终端驱动程序为例，其主要由键盘和显示器构成，终端驱动程序控制终端设备和进程之间的字符数据传输。包括以下5个操作：

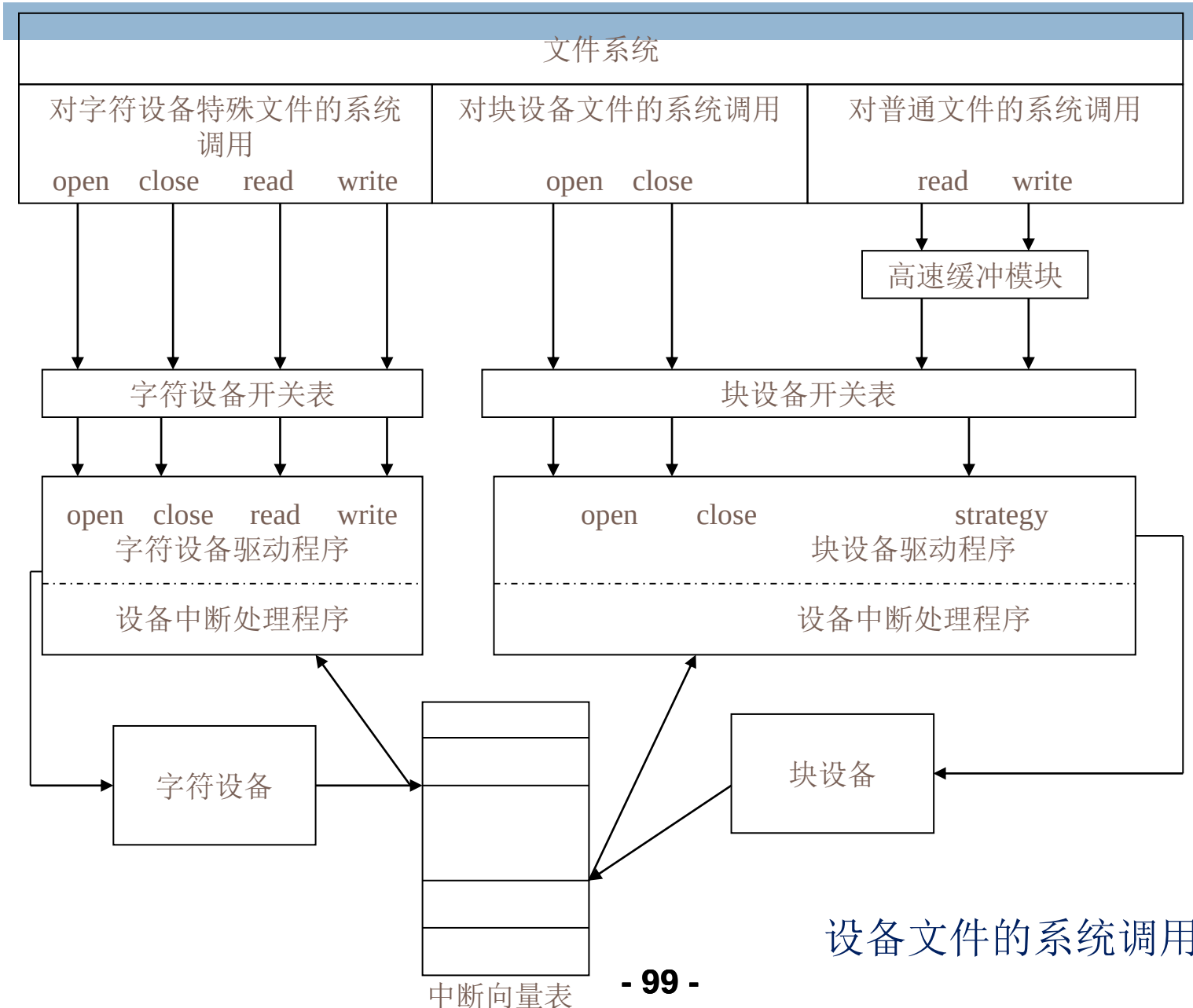
- ttopen：打开终端机，建立终端机和终端进程之间的对应关系。
- ttclose：切断终端机和终端进程的联系。
- ioctl：用于对终端机的控制，例如状态设置、测试、终端机属性值的更改等。
- ttread：从终端键盘读入数据。
- ttwrite：向终端显示器写数据。

10.7 Linux IO

5、设备文件的系统调用过程

- 对设备特殊文件的系统调用，根据文件类型转入块设备开关表或字符开关表进行打开、关闭块设备或字符设备的操作。
- 字符设备特殊文件的系统调用Read、Write转向字符开关表中指示的设备驱动程序，而对普通文件或目录文件的Read、Write系统调用则通过高速缓冲模块转向设备驱动模块中的策略（Strategy）过程。

10.7 Linux IO



设备文件的系统调用过程

I/O设备管理总结

- 如何实现交互? ⇒ 首先需要了解I/O的工作原理
- 从用户如何I/O开始 ⇒ 用户发送一个命令(read)
- 系统调用read ⇒ 被展开成给一些寄存器发送命令的代码
- 发送完命令以后… ⇒ CPU轮询，CPU干其它事情并等中断
- 中断方案最常见 ⇒ 结合设备驱动，完成硬件相关的操作
- 实现独享设备的共享 ⇒ 假脱机系统（SPOOLING）
- IO子系统要实现硬件无关性，包装通用的操作
- 实现CPU与外设速度的矛盾，需要软硬件缓冲技术