第六章　设备管理

输入输出设备是计算机系统的五官与四肢（处理器和存储器是计算机系统的大脑），是用户与系统交互的工具。它们把外部的信息输送给操作系统，再把经过加工的信息返送给用户。有效地管理和利用这些设备则是操作系统的主要任务之一。

1. 设备如何与计算机通信？

2. 设备管理分哪些模块，与进程管理有关吗？

3. 前面学习的哪些内容涉及到设备？

本章主要内容

6.1 设备管理概述

操作系统中系统负责管理输入与输出设备系统称为I／O系统。

6.1.1 设备的分类

分类目的：简化设备管理程序，不同的设备对应不同的管理程序，但对于同类设备可利用相同的程序或少量的修改即可。

1. 数据传输速率：

第一类是低速设备，这是指其传输速率仅为每秒钟几个字节至数百个字节的一类设备。属于低速设备的典型设备有键盘、鼠标器、语音的输入和输出等设备。

第二类是中速设备，这是指其传输速率在每秒钟数千个字节至数十万个字节的一类设备。典型的中速设备有行式打印机、 激光打印机等。

第三类是高速设备，这是指其传输速率在数十万字节至千兆字节的一类设备。典型的高速设备有磁带机、磁盘机、光盘机等。

2. 数据交换的单位：

（1）块设备(Block Device)，这类设备用于存储信息。信息的存取总是以数据块为单位。它属于有结构设备。

典型的块设备是磁盘，每个盘块的大小为512 B～4 KB。磁盘设备的基本特征是其传输速率较高，通常每秒钟为数MB到数十MB；另一特征是可寻址，即对它可随机地读/写任一块；此外，磁盘设备的I/O常采用DMA方式。

（2）字符设备(Character Device)，基本单位是字符，如交互式终端、打印机等。字符设备的基本特征是其传输速率较低，通常为几个字节至数千字节；另一特征是不可寻址，即输入/输出时不能指定数据的输入源地址及输出的目标地址；此外，字符设备在输入/输出时，常采用中断驱动方式。

3. 设备的共享属性：

这种分类方式可将I/O设备分为如下三类：

（1）独占设备。这是指在一段时间内只允许一个用户(进程)访问的设备，即临界资源。因而，对多个并发进程而言，应互斥地访问这类设备。

（2）共享设备。这是指在一段时间内允许多个进程同时访问的设备。当然，对于每一时刻而言，该类设备仍然只允许一个进程访问。显然，共享设备必须是可寻址的和可随机访问的设备。典型的共享设备是磁盘。

(3) 虚拟设备。这是指通过虚拟技术将一台独占设备变换为若干台逻辑设备，供若干个用户(进程)同时使用。

6.1.2 设备管理的目标

设备管理的主要对象是**I/O设备**和相应的**设备控制器**。设备管理的主要目标是屏蔽I／0设备的硬件特性，向用户提供使用I／0设备的方便接口，充分发挥设备的利用率。

1.设备配置和资源分配。

2.设备分配与释放。

3.控制设备和CPU的数据交换 。

4.隐蔽设备特性，提供独立于设备的统一接口。（命名）

5.提高设备利用率。

6.1.3　设备控制器

I/O设备一般是由执行I/O 操作的机械部分和执行控制I/O的电子部件组成。通常将这两部分分开，执行I/O操作的机械部分就是一般的I/O设备，而执行控制I/O的电子部件则称为设备控制器或适配器（adapter）。在微型机和小型机中的控制器常做成印刷电路卡形式。因而也常称为控制卡、接口卡或网卡，可将它插入计算机的扩展槽中。在有的大、中型计算机系统中，还配置了I/O通道或I/O处理机。

1．设备控制器的基本功能

1）接收和识别命令

CPU可以向控制器发送多种不同的命令，设备控制器应能接收并识别这些命令。为此，在控制器中应具有相应的控制寄存器，用来存放接收的命令和参数，并对所接收的命令进行译码。例如，磁盘控制器可以接收CPU发来的Read、Write、Format等15条不同的命令，而且有些命令还带有参数；相应地，在磁盘控制器中有多个寄存器和命令译码器等。

2）数据交换

这是指实现CPU与控制器之间、控制器与设备之间的数据交换。对于前者，是通过数据总线，由CPU并行地把数据写入控制器，或从控制器中并行地读出数据；对于后者，是设备将数据输入到控制器，或从控制器传送给设备。为此，在控制器中须设置数据寄存器。

3）标识和报告设备的状态

控制器应记下设备的状态供CPU了解。例如，仅当该设备处于发送就绪状态时，CPU才能启动控制器从设备中读出数据。为此，在控制器中应设置一状态寄存器，用其中的每一位来反映设备的某一种状态。当CPU将该寄存器的内容读入后，便可了解该设备的状态。

4）地址识别

就像内存中的每一个单元都有一个地址一样，系统中的每一个设备也都有一个地址，而设备控制器又必须能够识别它所控制的每个设备的地址。此外，为使CPU能向(或从)寄存器中写入(或读出)数据，这些寄存器都应具有唯一的地址。控制器应能正确识别这些地址，为此，在控制器中应配置地址译码器。

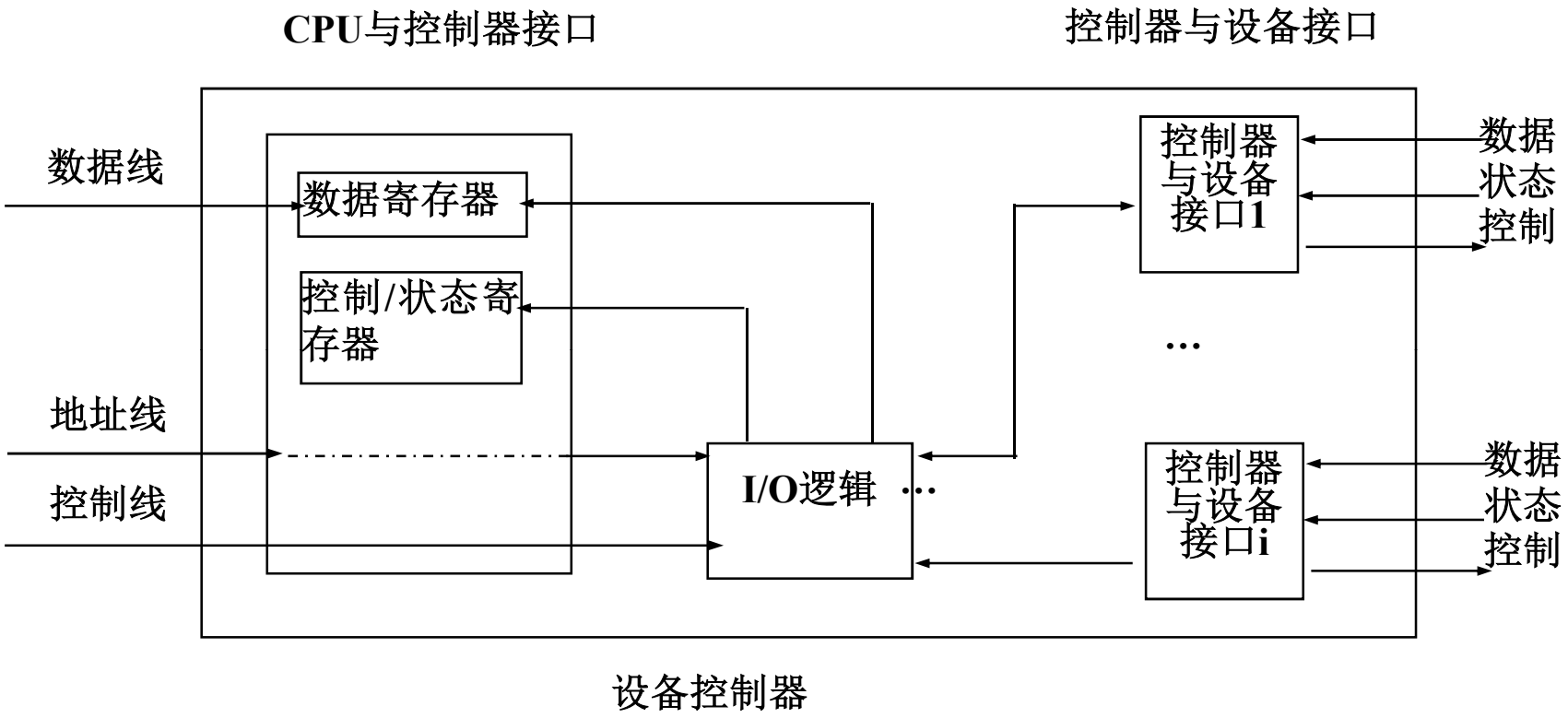
5）数据缓冲

由于I/O设备的速率较低而CPU和内存的速率却很高，故在控制器中必须设置一缓冲器。

在输出时，用此缓冲器暂存由主机高速传来的数据，然后才以I/O设备所具有的速率将缓冲器中的数据传送给I/O设备；在输入时，缓冲器则用于暂存从I/O设备送来的数据，待接收到一批数据后，再将缓冲器中的数据高速地传送给主机。

6）差错控制

设备控制器还兼管对由I/O设备传送来的数据进行差错检测。若发现传送中出现了错误，通常是将差错检测码置位，并向CPU报告，于是CPU将本次传送来的数据作废，并重新进行一次传送。这样便可保证数据输入的正确性。



1. 接收和识别命令；2. 数据交换；3. 标识和报告设备的状态；
4. 地址识别 ；5. 差错控制

6.1.4 I/O系统的层次结构

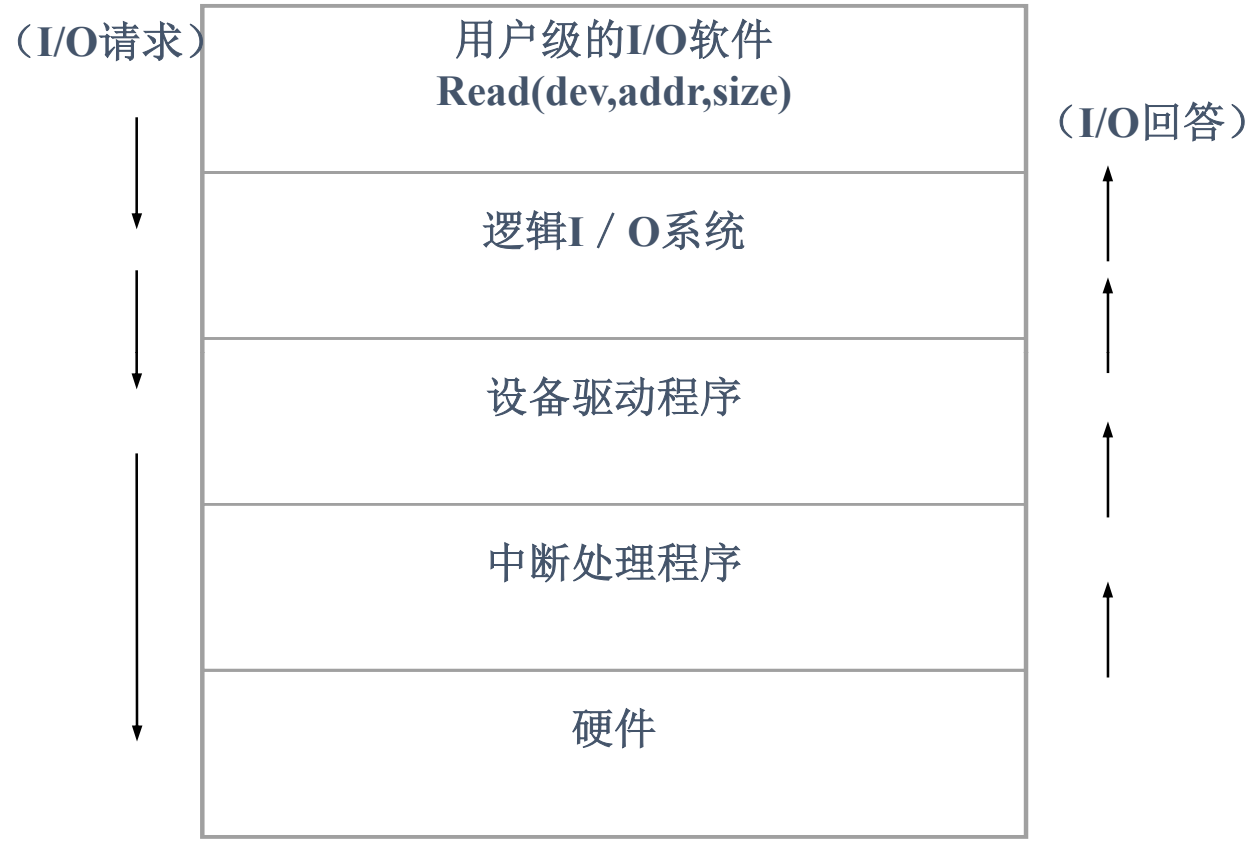
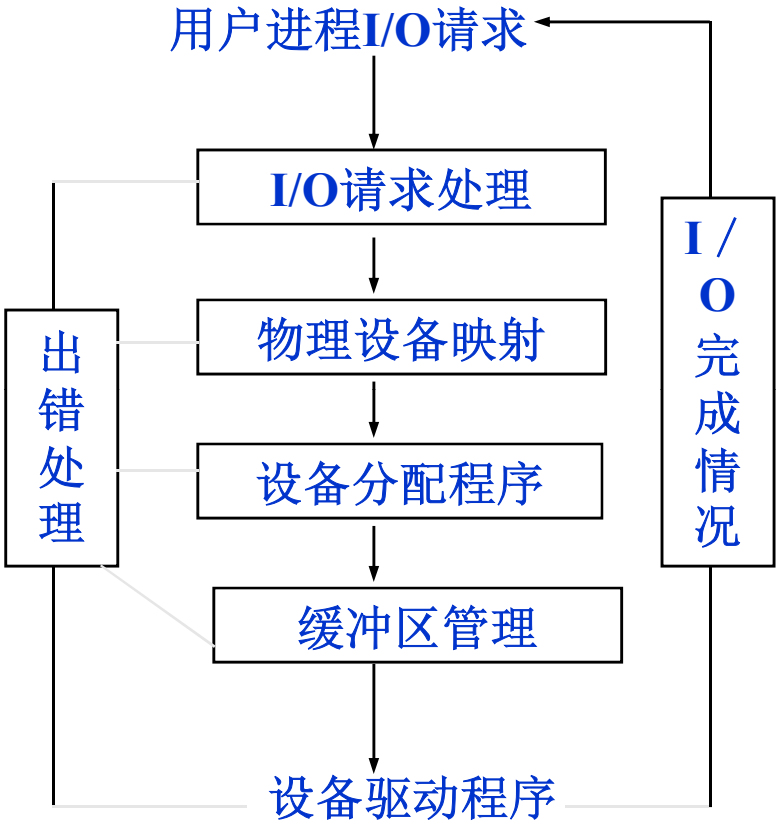


图6-2 I/O系统的层次结构

1. 用户级的I／O软件__系统调用

2. 逻辑I／O系统



逻辑I／O功能

3. 设备驱动程序

设备驱动程序是I/O进程与设备控制器之间的通信程序。

1．设备驱动程序的功能

为了实现I/O进程与设备控制器之间的通信，设备驱动程序应具有以下功能：

(1) 接收由设备独立性软件发来的命令和参数，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）及时响应由控制器或通道发来的中断请求，并根据其中断类型调用相应的中断处理程序进行处理。

(5) 对于设置有通道的计算机系统，驱动程序还应能够根据用户的I/O请求，自动地构成通道程序。

驱动程序与设备控制器和I/O设备的硬件特性紧密相关，因而对不同类型的设备应配置不同驱动程序。

驱动程序与I/O设备所采用的I/O控制方式紧密相关。

4. 中断处理程序

I／O中断处理程序位于最底层，它响应I／O的中断请求，完成相应的中断处理。I／O中断处理程序的基本工作包括：保留现场；唤醒因等待该I／O操作完成而被阻塞的某个进程（如设备驱动进程或请求I／O的进程），通知I／O已完成；最终转入进程调度程序重新进行调度。

例：如用户进程在运行过程中，调用系统调用write(dev，addr，size)，向一个磁盘文件写入一组数据。

•系统调用将向逻辑I／O系统发送一个I／O请求，用户进程将进入阻塞状态，等待数据传输的完成。

•逻辑I／O系统接受该请求，检查相应的设备是否已准备好，为数据传输分配必要的缓冲，调用相应的磁盘驱动程序，完成数据传输。

•在数据传输完成后，由硬件产生一个中断，转入中断处理程序。

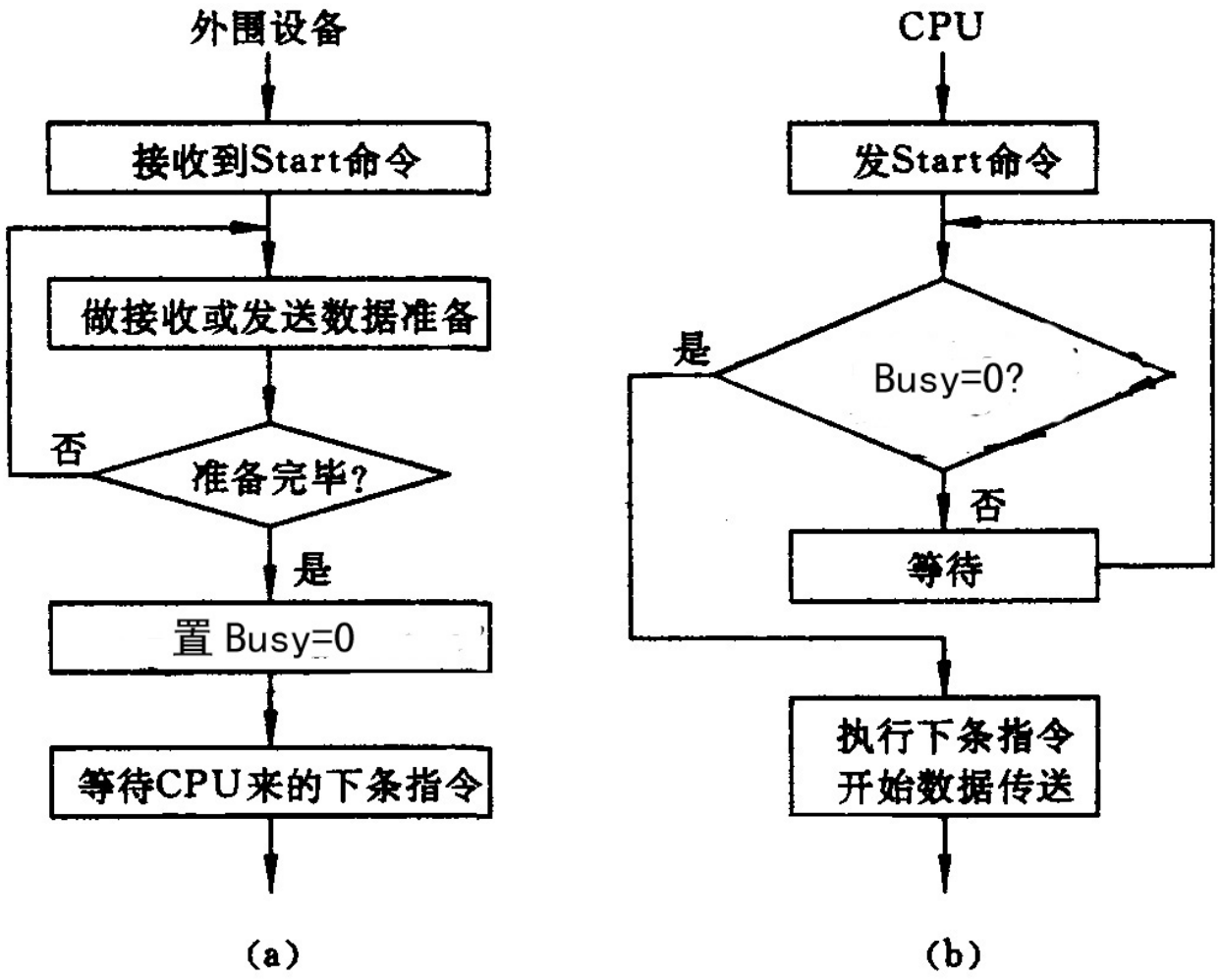
•中断处理程序检查中断的原因，知道是磁盘读取操作已完成，于是将完成信息传送给磁盘驱动程序，由磁盘驱动程序将数据传送完成的情况传送给逻辑I／O系统，逻辑I／O系统唤醒用户进程，结束此次I／O请求。

6.2 数据传送控制方式

1. 程序I/0方式

早期的计算机系统中，由于无中断机构，处理机对I/O设备的控制采取程序I/O(Programmed I/O)方式，或称为忙—等待方式。

即在处理机向控制器发出一条I/O指令启动输入设备输入数据时，要同时把状态寄存器中的忙/闲标志busy置为1，然后便不断地循环测试busy。当busy=1时，表示输入机尚未输完一个字(符)，处理机应继续对该标志进行测试，直至busy=0，表明输入机已将输入数据送入控制器的数据寄存器中。于是处理机将数据寄存器中的数据取出，送入内存指定单元中，这样便完成了一个字(符)的I/O。接着再去启动读下一个数据，并置busy=1。下图示出了程序I/O方式的流程。

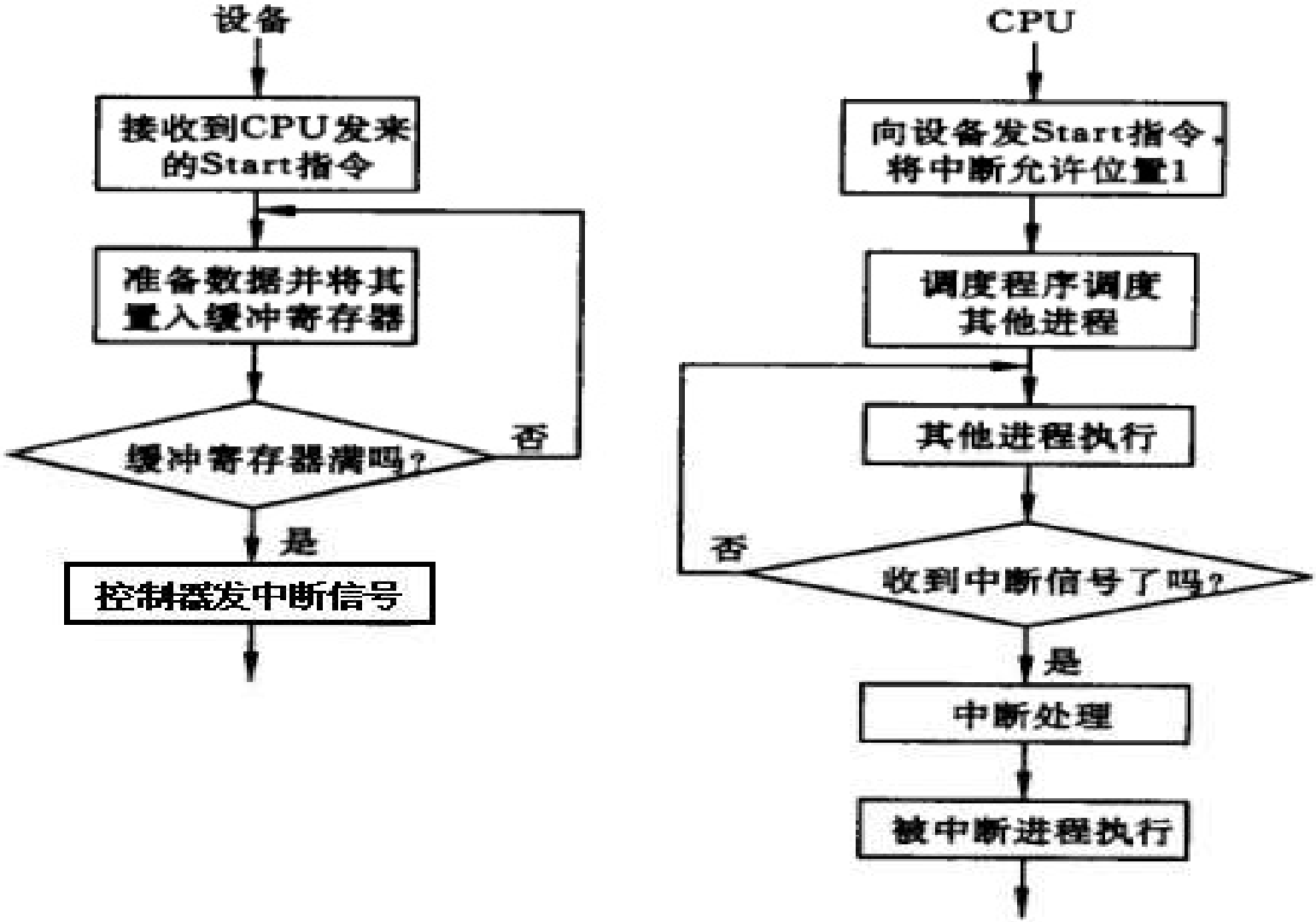


优缺点：控制简单，接口硬件少；但CPU严重浪费，外设备不能并行工作，不能处理外设发生的错误。

2. 中断驱动I/O控制方式

现代计算机系统中引入了中断机构，对I/O设备的控制，广泛采用中断驱动(Interrupt Driven)方式。即当某进程要启动某个I/O设备工作时，便由CPU向相应的设备控制器发出一条I/O命令，然后立即返回继续执行原来的任务。设备控制器于是按照该命令的要求去控制指定I/O设备。此时，CPU与I/O设备并行操作。

例如，在输入时，当设备控制器收到CPU发来的读命令后，便去控制相应的输入设备读数据。一旦数据进入数据寄存器，控制器便通过控制线向CPU发送一中断信号，由CPU检查输入过程中是否出错，若无错，便向控制器发送取走数据的信号，然后再通过控制器及数据线将数据写入内存指定单元中。下图示出了中断驱动方式的流程。


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
缓冲寄存器满吗？
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

在I/O设备输入每个数据的过程中，由于无需CPU干预，因而可使CPU与I/O设备并行工作。仅当输完一个数据时，才需CPU花费极短的时间去做些中断处理。

例如，从终端输入一个字符的时间约为100 ms，而将字符送入终端缓冲区的时间小于0.1 ms。若采用程序I/O方式，CPU约有99.9 ms的时间处于忙—等待的过程中。但采用中断驱动方式后，CPU可利用这99.9 ms的时间去做其它的事情，而仅用0.1 ms的时间来处理由控制器发来的中断请求。可见，中断驱动方式可以成百倍地提高CPU的利用率。

缺点：传输数据较多时，中断次数也多；设备多时，中断次数急剧增加，造成CPU无法响应中断，出现数据丢失现象。

3. 直接存储器访问(DMA)I/O控制方式

1） DMA(Direct Memory Access)控制方式的引入

虽然中断驱动I/O比程序I/O方式更有效，但它仍是以字(节)为单位进行I/O的，每当完成一个字(节)的I/O时，控制器便要向CPU请求一次中断。如果将这种方式用于块设备的I/O，显然是极其低效的。例如，为了从磁盘中读出1 KB的数据块，需要中断CPU 1K次。为了进一步减少CPU对I/O的干预而引入了直接存储器访问方式。

DMA方式的特点是：

(1) 数据传输的基本单位是数据块，即在CPU与I/O设备之间，每次传送至少一个数据块；

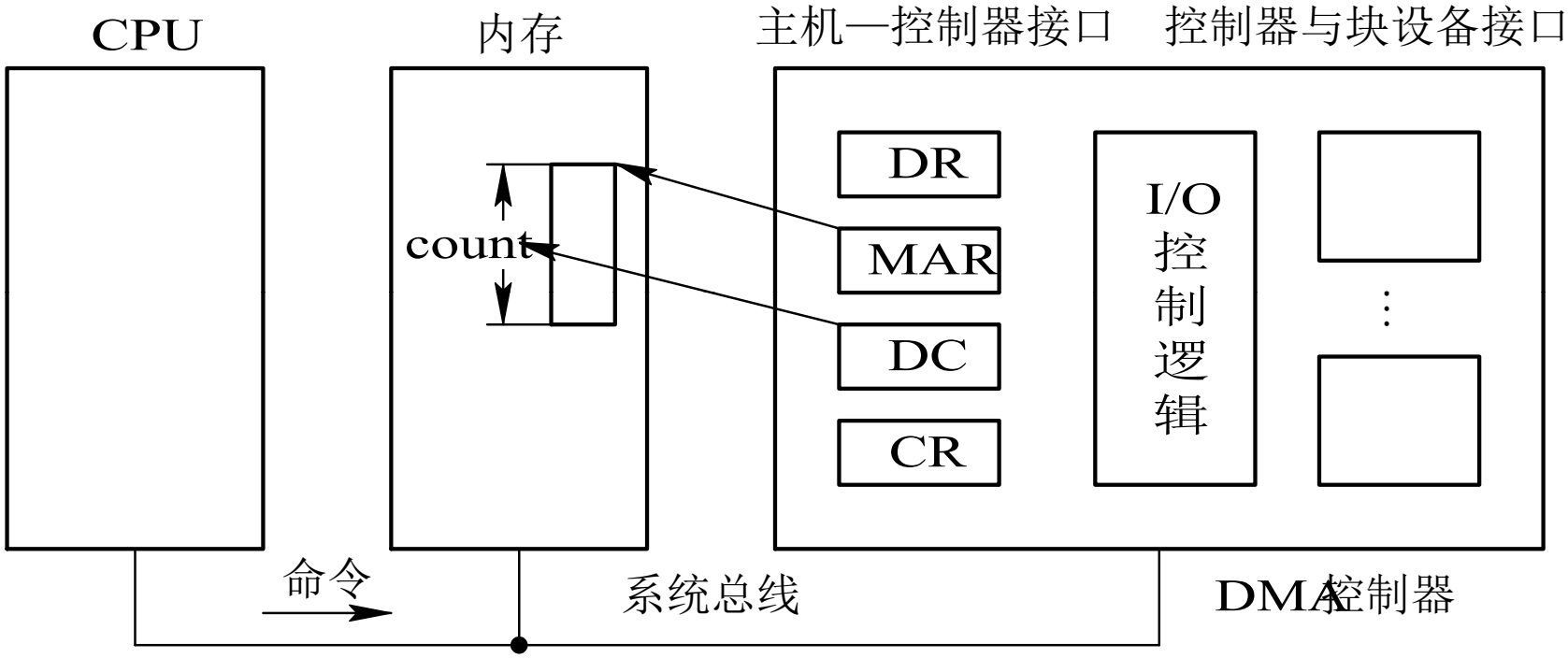
(2) 所传送的数据是从设备直接送入内存的，或者相反；

(3) 仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在控制器的控制下完成的。

可见，DMA方式较之中断驱动方式，又是成百倍地减少了CPU对I/O的干预，进一步提高了CPU与I/O设备的并行操作程度。

2)DMA控制器的组成

DMA控制器由三部分组成：主机与DMA控制器的接口；DMA控制器与块设备的接口；I/O控制逻辑。



DMA控制器的组成

为了实现在主机与控制器之间成块数据的直接交换，必须在DMA控制器中设置如下四类寄存器：

(1) 命令/状态寄存器(CR)。用于接收从CPU发来的I/O命令，或有关控制信息，或设备的状态。

(2) 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。

(3) 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。

(4) 数据计数器(DC)。存放本次CPU要读或写的字(节)数。

3）DMA工作过程

例：从磁盘读入数据过程

当CPU要从磁盘读入一数据块时，便向磁盘控制器发送一条读命令。该命令被送到其中的命令寄存器(CR)中。

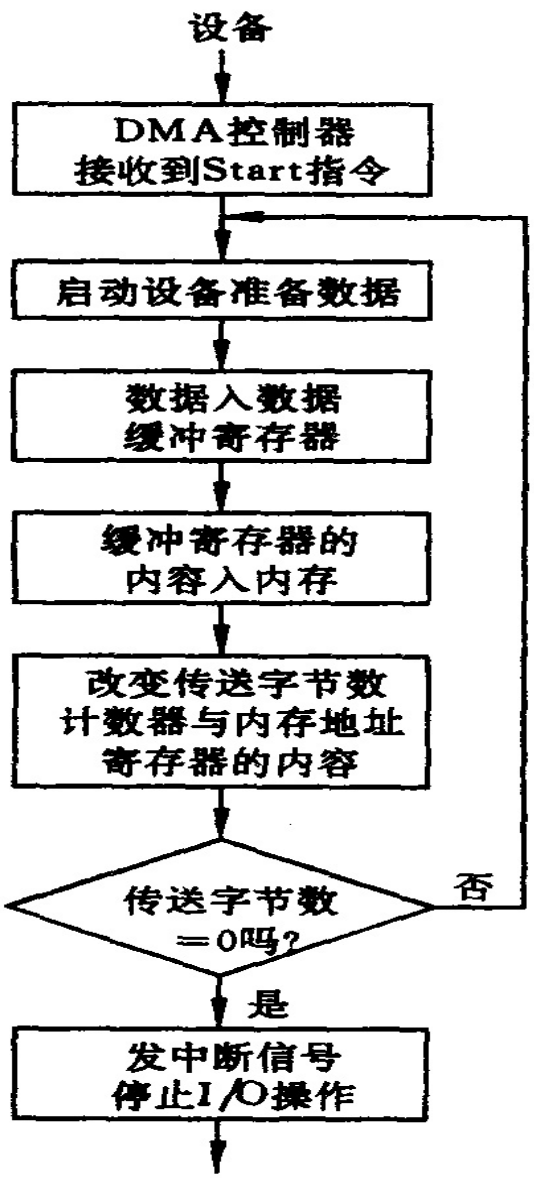
同时，还须发送本次要将数据读入的内存起始目标地址，该地址被送入内存地址寄存器（MAR）中；

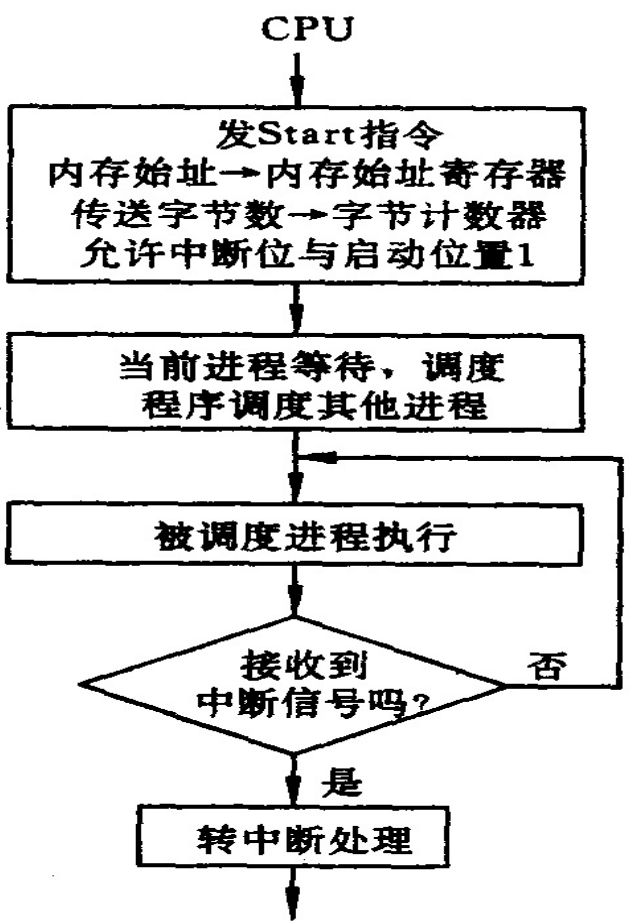
本次要读数据的字(节)数则送入数据计数器(DC)中，还须将磁盘中的源地址直接送至DMA控制器的I/O控制逻辑上。

然后，启动DMA控制器进行数据传送，以后，CPU便可去处理其它任务。此后，整个数据传送过程便由DMA控制器进行控制。

当DMA控制器已从磁盘中读入一个字(节)的数据并送入数据寄存器(DR)后，再挪用一个存储器周期，将该字(节)传送到MAR所指示的内存单元中。

接着便对MAR内容加1，将DC内容减1。若减1后DC内容不为0，表示传送未完，便继续传送下一个字(节)；否则，由DMA控制器发出中断请求。


设备
DMA控制器
接收到Start指令
启动设备准备数据
数据入数据
缓冲寄存器
缓冲寄存器的
内容入内存
改变传送字节数
计数器与内存地址
寄存器的内容
传送字节数
=0吗?
否
是
发中断信号
停止I/O操作


CPU
发Start指令
内存始址→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
当前进程等待，调度
程序调度其他进程
被调度进程执行
接收到
中断信号吗?
否
是
转中断处理

4）与中断的区别：

1）中断处理次数不同；

2）DMA直接完成与内存的数据交换，不受CPU控制。

5）缺点：

1）大量不连续的数据传送仍需CPU控制，多块仍需中断；

2）多个DMA控制器同时使用会引起内存地址冲突；

3）需DMA控制器硬件支持。

6.2.4 I/O通道控制方式

1. 通道

I/O通道是一种特殊的处理机，它具有执行I/O指令的能力，并通过执行通道(I/O)程序来控制I/O操作。

但I/O通道又与一般的处理机不同，一是其指令类型单一，其所能执行的命令主要局限于与I/O操作有关的指令；二是通道没有自己的内存，通道所执行的通道程序是放在主机的内存中的，换言之，是通道与CPU共享内存。

2．I/O通道控制方式过程

虽然DMA方式比起中断方式来已经显著地减少了CPU的干预，但CPU每发出一条I/O指令，也只能去读(或写)一个连续的数据块。

而当我们需要一次去读多个数据块且将它们分别传送到不同的内存区域，或者相反时，则须由CPU分别发出多条I/O指令及进行多次中断处理才能完成。

I/O通道方式是DMA方式的发展，它可进一步减少CPU的干预，即把对一个数据块的读(或写)为单位的干预减少为对一组数据块的读(或写)及有关的控制和管理为单位的干预。同时，又可实现CPU、通道和I/O设备三者的并行操作，从而更有效地提高整个系统的资源利用率。

例如，当CPU要完成一组相关的读(或写)操作及有关控制时，只需向I/O通道发送一条I/O指令，以给出其所要执行的通道程序的首址和要访问的I/O设备，通道接到该指令后，通过执行通道程序便可完成CPU指定的I/O任务。

3. 通道程序

通道是通过执行通道程序，并与设备控制器共同实现对I/O设备的控制的。通道程序是由一系列通道指令(或称为通道命令)所构成的。通道指令与一般的机器指令不同，在它的每条指令中都包含下列诸信息：

(1) 操作码。操作码规定了指令所执行的操作，如读、写、控制等操作。

(2) 内存地址。内存地址标明字符送入内存(读操作)和从内存取出(写操作)时的内存首址。

(3) 计数。该信息表示本条指令所要读(或写)数据的字节数。

（4）通道程序结束位P。该位用于表示通道程序是否结束。P=1表示本条指令是通道程序的最后一条指令。

（5）记录结束标志R。R=0表示本通道指令与下一条指令所处理的数据是同属于一个记录；R=1表示这是处理某记录的最后一条指令。

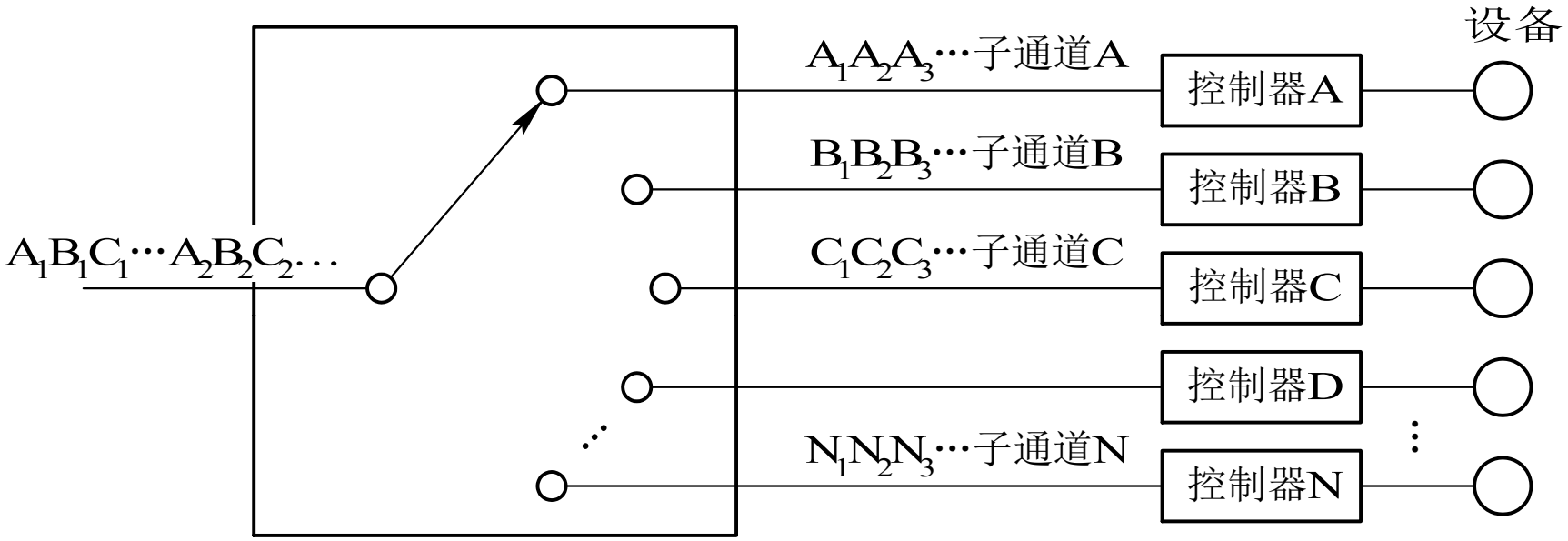
下面示出了一个由六条通道指令所构成的简单的通道程序。其中，前三条指令是分别将813～892单元中的80个字符和1034～1173单元中的140个字符及5830～5889单元中的60个字符写成一个记录；第4条指令是单独写一个具有300个字符的记录；第5、6条指令共写含500个字符的记录。

操　作	P	R	计　数	内存地址
WRITE	0	0	80	813
WRITE	0	0	140	1034
WRITE	0	1	60	5830
WRITE	0	1	300	2000
WRITE	0	0	250	1650
WRITE	1	1	250	2720

4．通道类型

1）字节多路通道(Byte Multiplexor Channel)

这是一种按字节交叉方式工作的通道。它通常都含有许多非分配型子通道，其数量可从几十到数百个，每一个子通道连接一台I/O设备，并控制该设备的I/O操作。这些子通道按时间片轮转方式共享主通道。这样，只要字节多路通道扫描每个子通道的速率足够快，而连接到子通道上的设备的速率不是太高时，便不致丢失信息。


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

2）数组选择通道(Block Selector Channel)

字节多路通道不适于连接高速设备，这推动了按数组方式进行数据传送的数组选择通道的形成。

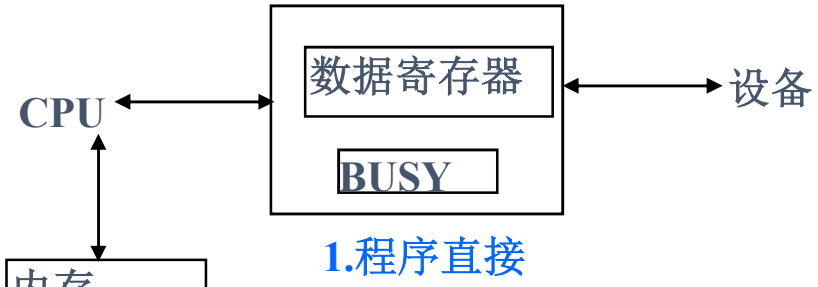
这种通道虽然可以连接多台高速设备，但由于它只含有一个分配型子通道，在一段时间内只能执行一道通道程序，控制一台设备进行数据传送，致使当某台设备占用了该通道后，便一直由它独占，即使是它无数据传送，通道被闲置，也不允许其它设备使用该通道，直至该设备传送完毕释放该通道。可见，这种通道速率快，利用率很低。

3）数组多路通道(Block Multiplexor Channel)

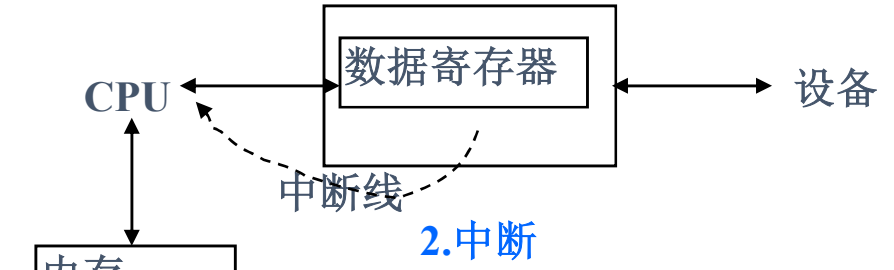
连接控制多个高速外设并以成组交叉方式传送数据的通道称为数组多路通道。数组多路通道是对选择通道的一种改进，当某个设备进行数据传送时，通道只为该设备提供服务；当设备在执行寻址等控制性动作时，通道暂时断开与该设备的连接，挂起该设备的通道程序，而转去为其他设备提供服务，即执行其他设备的通道程序。所以，数组多路通道很像一个多道程序的处理器。

对于磁盘一类的高速外设，采用数组多路通道，可在其中一个外设占用通道进行数据传送时，让其他外设进行寻址等辅助操作，使一个设备的数据传送操作与其他设备的寻址操作彼此重叠，实现成组交叉方式的数据传送，从而使通道具备多路并行工作的能力，充分发挥通道高速信息交换的效能。

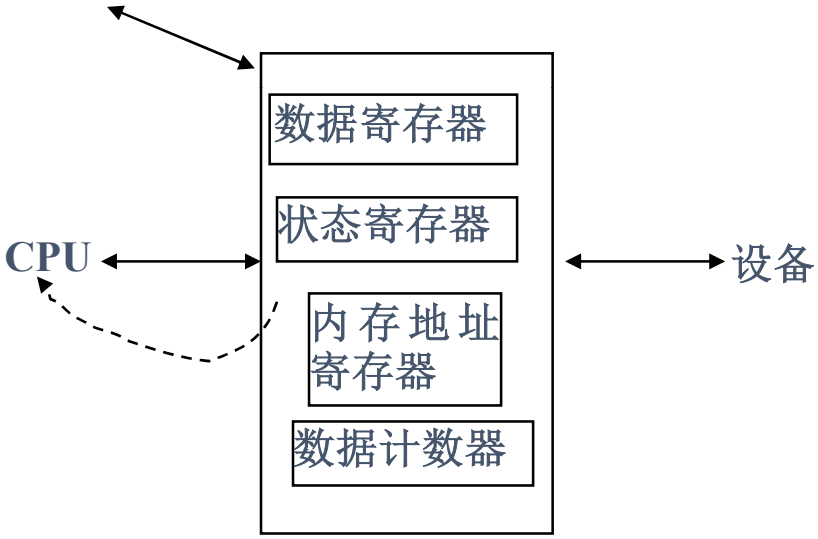
总结：不同控制方式的接口



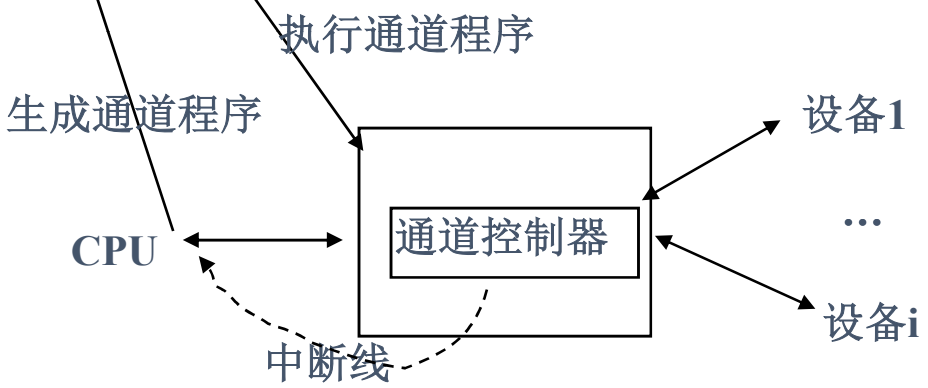
1.程序直接



2.中断



3.DMA方式



4.通道方式

6.3 中断与设备驱动程序

中断处理过程

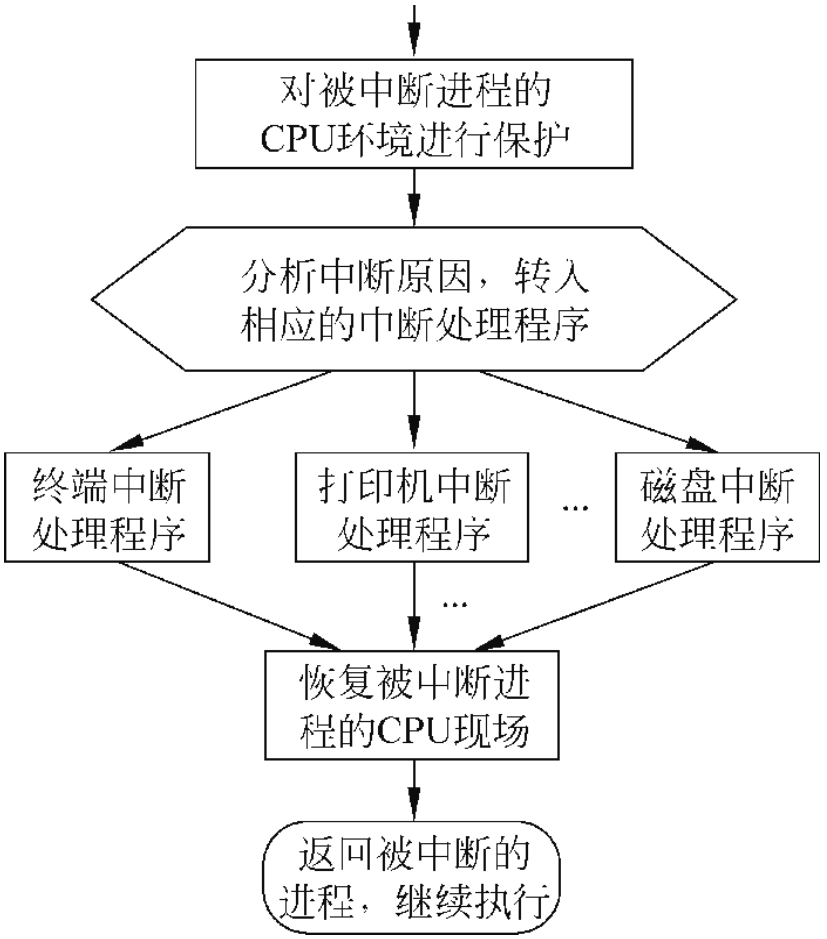


图6-8 中断处理流程

6.3.2 设备驱动程序

1. 设备驱动程序的特点

设备驱动程序属于低级的系统例程，它与一般的应用程序及系统程序之间有下述明显差异：

(1) 驱动程序主要是指在请求I/O的进程与设备控制器之间的一个通信和转换程序。它将进程的I/O请求经过转换后，传送给控制器；又把控制器中所记录的设备状态和I/O操作完成情况及时地反映给请求I/O的进程。

（2）驱动程序与设备控制器和I/O设备的硬件特性紧密相关，因而对不同类型的设备应配置不同的驱动程序。例如，可以为相同的多个终端设置一个终端驱动程序，但有时即使是同一类型的设备，由于其生产厂家不同，它们也可能并不完全兼容，此时也须为它们配置不同的驱动程序。

（3）驱动程序与I/O设备所采用的I/O控制方式紧密相关。常用的I/O控制方式是中断驱动和DMA方式，这两种方式的驱动程序明显不同，因为后者应按数组方式启动设备及进行中断处理。

(4) 由于驱动程序与硬件紧密相关，因而其中的一部分必须用汇编语言书写。目前有很多驱动程序的基本部分，已经固化在ROM中。

(5) 驱动程序应允许可重入。一个正在运行的驱动程序常会在一次调用完成前被再次调用。例如，网络驱动程序正在处理一个到来的数据包时，另一个数据包可能到达。

(6) 驱动程序不允许系统调用。但是为了满足其与内核其它部分的交互，可以允许对某些内核过程的调用，如通过调用内核过程来分配和释放内存页面作为缓冲区，以及调用其它过程来管理MMU定时器、DMA控制器、中断控制器等。

（5）不同的操作系统对设备驱动程序的结构要求不同。

一般来说，在操作系统中的相关文档中，都有对设备驱动程序结构的要求描述。每个设备生产厂商和软件开发商都必须按照设备驱动程序的标准结构编写独立的设备驱动程序，当系统需要时，再将它安装配置到系统中。

2．设备驱动程序的处理过程

不同类型的设备应有不同的设备驱动程序，但大体上它们都可以分成两部分，其中，除了要有能够驱动I/0设备工作的驱动程序外，还需要有设备中断处理程序，以处理I/0完成后的工作。

设备驱动程序的主要任务是启动指定设备。但在启动之前，还必须完成必要的准备工作，如检测设备状态是否为“忙”等。在完成所有的准备工作后，才最后向设备控制器发送一条启动命令。

设备驱动程序的处理过程：

1）将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。

2）检查I/O请求的合法性

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。

3）读出和检查设备的状态

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。

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。

6）启动I/O设备

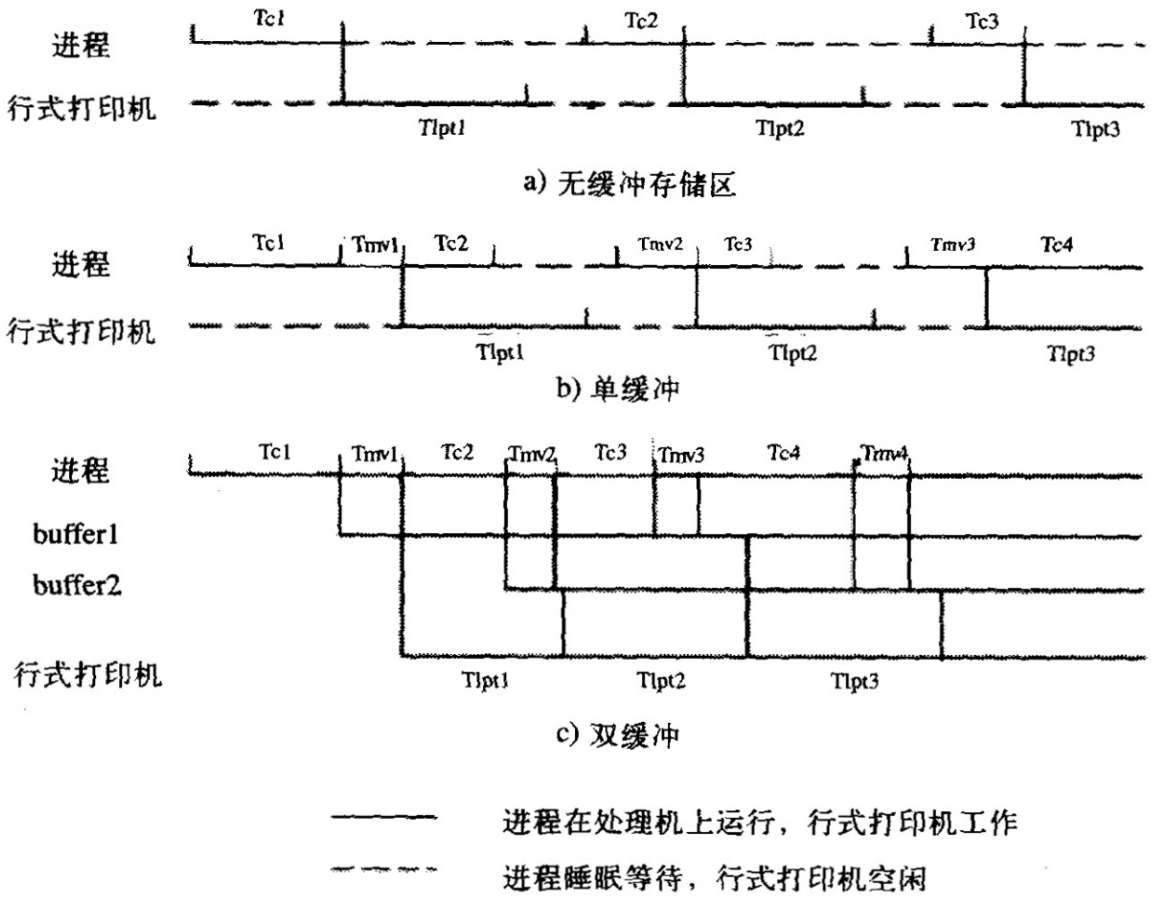
在完成上述各项准备工作之后，驱动程序可以向控制器中的命令寄存器传送相应的控制命令。对于字符设备，若发出的是写命令，驱动程序将把一个数据传送给控制器；若发出的是读命令，则驱动程序等待接收数据，并通过从控制器中的状态寄存器读入状态字的方法，来确定数据是否到达。

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。通常，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。

6.4 缓冲技术

在现代操作系统中，几乎所有的I/O设备在与处理机交换数据时都用了缓冲区。缓冲区是一个存储区域，它可以由专门的硬件寄存器组成，但由于硬件的成本较高，容量也较小，一般仅用在对速度要求非常高的场合，如存储器管理中所用的联想寄存器；设备控制器中用的数据缓冲区等。在一般情况下，更多的是利用内存作为缓冲区。本节所要介绍的也正是由内存组成的缓冲区。缓冲区管理的主要功能是组织好这些缓冲区，并提供获得和释放缓冲区的手段。

6.4.1 缓冲引入的原因



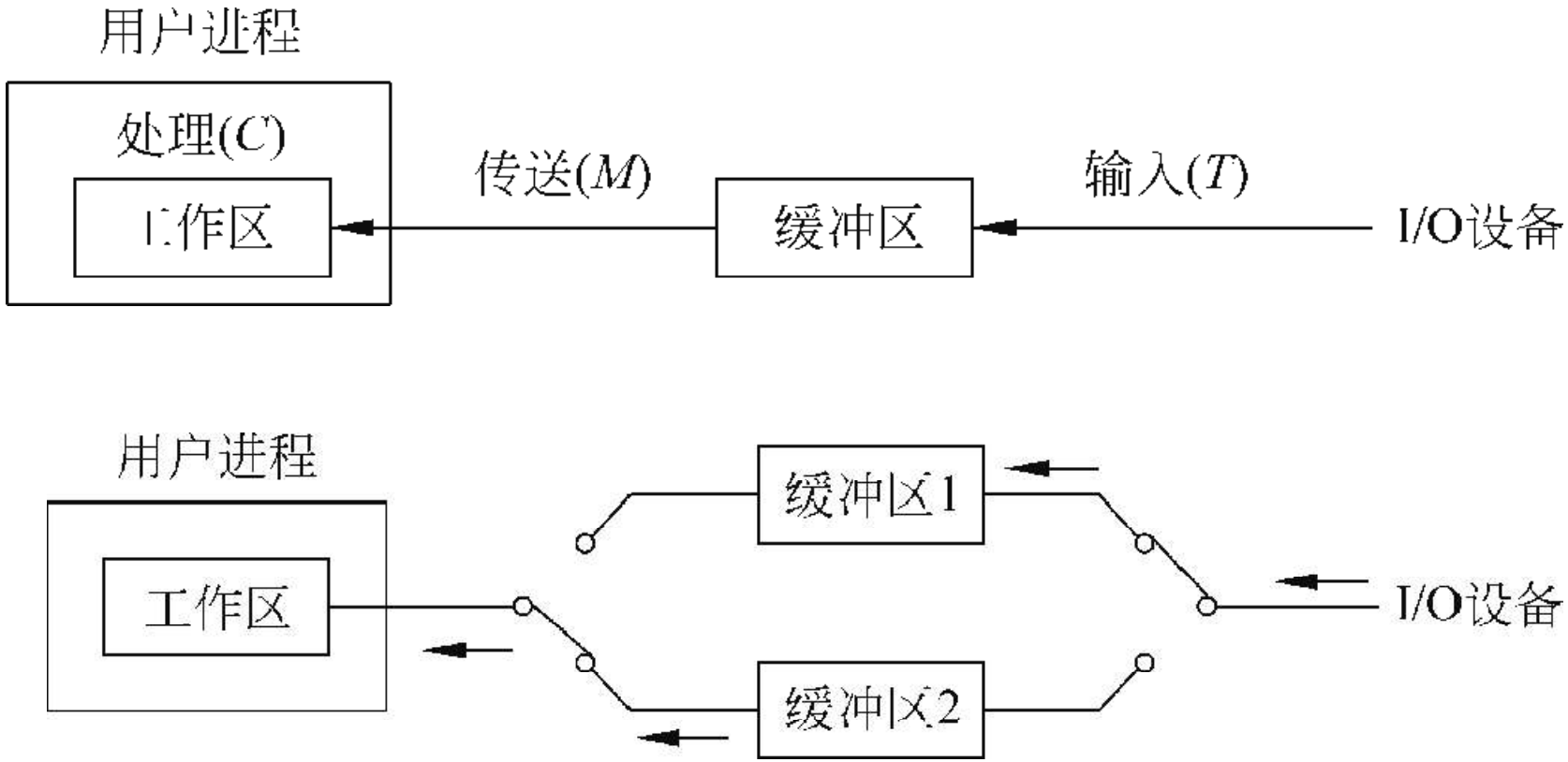
（1）为了解决中央处理机和外部设备的速度不匹配和负荷不均衡问题；

（2）减少处理调度及中断次数；

（3）提高各种设备的工作效率，增加系统中各部分的并行工作速度。在设备管理中引入了缓冲技术。

6.4.1 缓冲的种类

单缓冲、双缓冲、缓冲池



6.4.2 缓冲池的管理

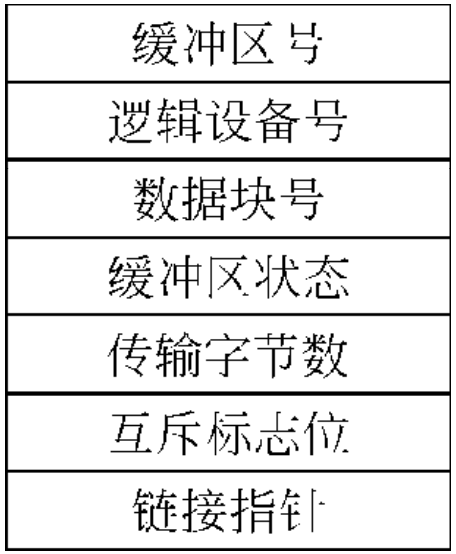
1．缓冲池的组成

对于既可用于输入又可用于输出的公用缓冲池，其中至少应含有以下三种类型的缓冲区：

① 空(闲)缓冲区；

② 装满输入数据的缓冲区；

③ 装满输出数据的缓冲区。



为了管理上的方便，可将相同类型的缓冲区链成一个队列，于是可形成以下三个队列:

（1）空缓冲队列emq。这是由空缓冲区所链成的队列。其队首指针F(emq)和队尾指针L(emq)分别指向该队列的首缓冲区和尾缓冲区。

(2) 输入队列inq。这是由装满输入数据的缓冲区所链成的队列。其队首指针F(inq)和队尾指针L(inq)分别指向该队列的首缓冲区和尾缓冲区。

(3) 输出队列outq。这是由装满输出数据的缓冲区所链成的队列。其队首指针F(outq)和队尾指针L(outq)分别指向该队列的首缓冲区和尾缓冲区。

除了上述三个队列外，还应具有四种工作缓冲区：① 用于收容输入数据的工作缓冲区；② 用于提取输入数据的工作缓冲区；③ 用于收容输出数据的工作缓冲区；④ 用于提取输出数据的工作缓冲区。

2. Getbuf过程和Putbuf过程

在“数据结构”课程中，曾介绍过队列和对队列进行操作的两个过程，它们是：

(1) Addbuf(type，number)过程。该过程用于将由参数number所指示的缓冲区B挂在type队列上。

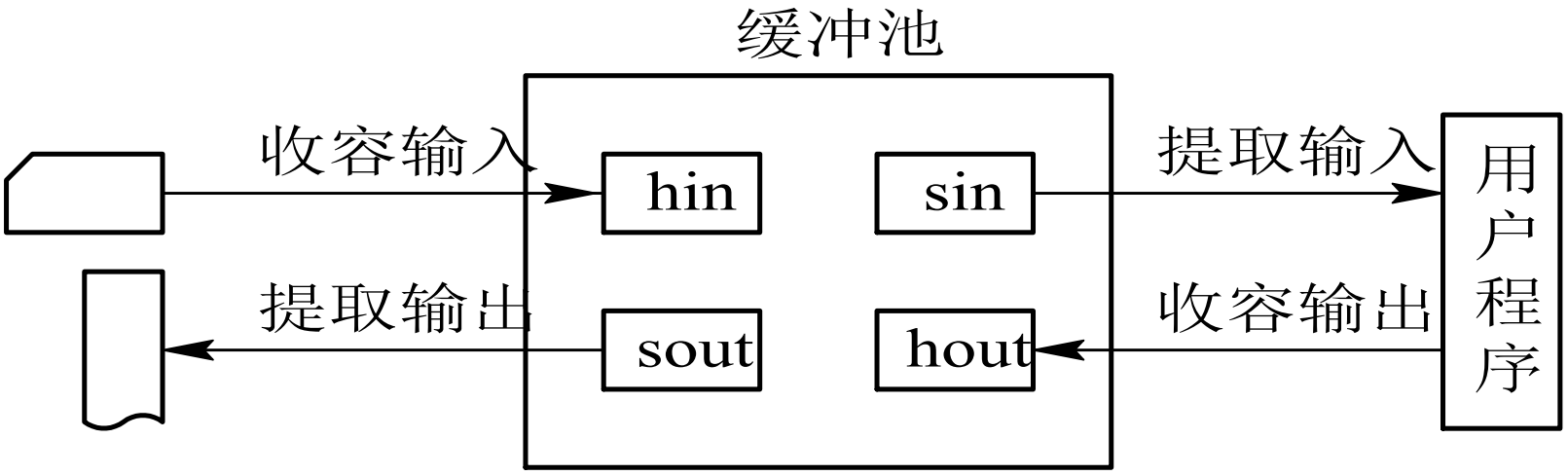
(2) Takebuf(type)过程。该过程用于从type所指示的队列的队首摘下一个缓冲区。

为使诸进程能互斥地访问缓冲池队列，可为每一队列设置一个互斥信号量MS(type)。此外，为了保证诸进程同步地使用缓冲区，又为每个缓冲队列设置了一个资源信号量RS(type)。既可实现互斥又可保证同步的Getbuf过程和Putbuf过程描述如下：

```
Procedure Getbuf(type)
  begin
    P(RS(type));
    P(MS(type));
    B(number):=Takebuf(type);
    V(MS(type));
  end
Procedure Putbuf(type, number)
  begin
    P(MS(type));
    Addbuf(type, number);
    V(MS(type));
    V(RS(type));
  end
```

3．缓冲区的工作方式

缓冲区可以工作在收容输入、提取输入、收容输出和提取输出四种工作方式下，如下图。

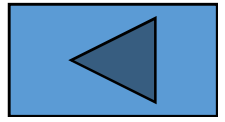


（1）收容输入。在输入进程需要输入数据时，便调用Getbuf(emq)过程，从空缓冲队列emq的队首摘下一空缓冲区，把它作为收容输入工作缓冲区hin。然后，把数据输入其中，装满后再调用Putbuf(inq，hin)过程，将该缓冲区挂在输入队列inq上。

(2) 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inq的队首取得一个缓冲区，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。

(3)收容输出。当计算进程需要输出时，调用Getbuf(emq)过程从空缓冲队列emq的队首取得一个空缓冲区，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。

(4) 提取输出。由输出进程调用Getbuf(outq)过程，从输出队列的队首取得一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。

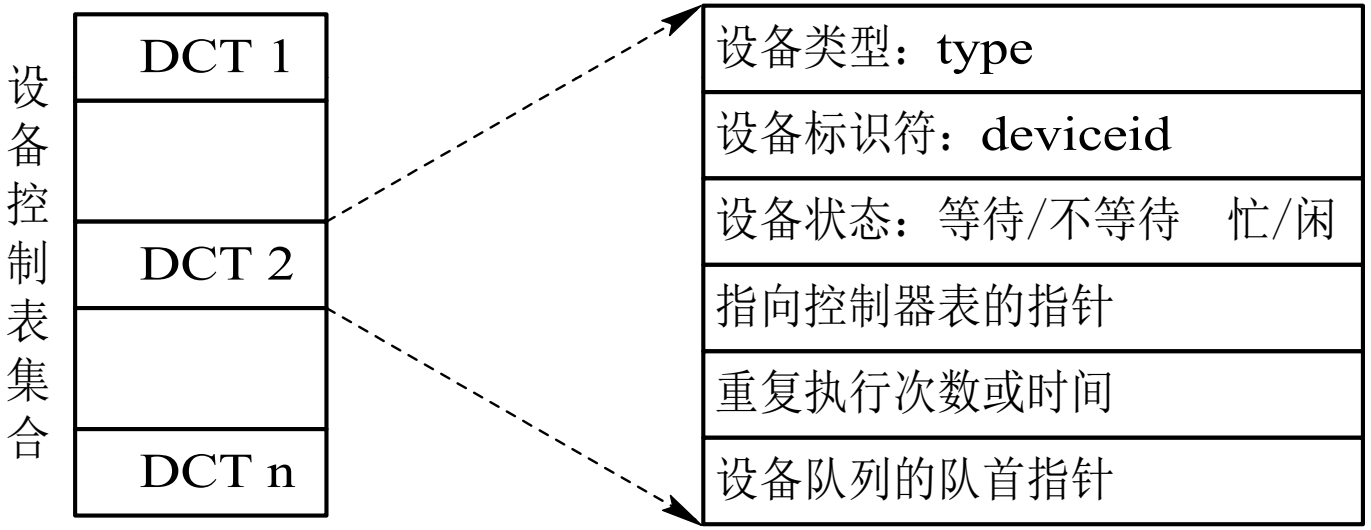


6.5 设备分配

1. 设备分配中的数据结构

1）设备控制表(DCT)

系统为每一个设备都配置了一张设备控制表，用于记录本设备的情况，如图5-20所示。



设备控制表中，除了有用于指示设备类型的字段type和设备标识字段deviceid外，还应含有下列字段：

（1）设备队列队首指针。凡因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列，称该队列为设备队列。其队首指针指向队首PCB。在有的系统中还设置了队尾指针。

(2) 设备状态。当设备自身正处于使用状态时，应将设备的忙/闲标志置“1”。若与该设备相连接的控制器或通道正忙，也不能启动该设备，此时则应将设备的等待标志置“1”。

（3）与设备连接的控制器表指针。该指针指向该设备所连接的控制器的控制表。在设备到主机之间具有多条通路的情况下，一个设备将与多个控制器相连接。

(4) 重复执行次数。由于外部设备在传送数据时，较易发生数据传送错误，因而在许多系统中，如果发生传送错误，并不立即认为传送失败，而是令它重新传送，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。

2)控制器控制表、通道控制表和系统设备表

（1） 控制器控制表(COCT)。系统为每一个控制器都设置了一张用于记录本控制器情况的控制器控制表，如下图（a）所示。

(2) 通道控制表(CHCT)。每个通道都配有一张通道控制表，如下图（b）所示。

(3) 系统设备表(SDT)。这是系统范围的数据结构，其中记录了系统中全部设备的情况。每个设备占一个表目，其中包括有设备类型、设备标识符、设备控制表及设备驱动程序的入口等项，如下图(c)所示。

(a) 控制器表COCT　　(b) 通道表CHCT　　(c) 系统设备表SDT

COCT、CHCT和SDT

2. 设备分配时应考虑的因素

为了使系统有条不紊地工作，系统在分配设备时，应考虑这样几个因素:

① 设备的固有属性;

② 设备分配算法;

③ 设备分配时的安全性。

1) 设备的固有属性

设备的固有属性可分成三种:

第一种是独占性，是指这种设备在一段时间内只允许一个进程独占，此即第二章所说的“临界资源”；对于独占设备，应采用独享分配策略，即将一个设备分配给某进程后，便由该进程独占，直至该进程完成或释放该设备，然后，系统才能再将该设备分配给其他进程使用。这种分配策略的缺点是，设备得不到充分利用，而且还可能引起死锁。

(2) 第二种是共享性，指这种设备允许多个进程同时共享；对于共享设备，可同时分配给多个进程使用，此时须注意对这些进程访问该设备的先后次序进行合理的调度。

(3)第三种是可虚拟设备。由于可虚拟设备是指一台物理设备在采用虚拟技术后，可变成多台逻辑上的所谓虚拟设备，一台可虚拟设备是可共享的设备，可以将它同时分配给多个进程使用，并对这些访问该(物理)设备的先后次序进行控制。

2）设备分配算法

对设备进行分配的算法，与进程调度的算法有些相似之处，但前者相对简单，通常只采用以下两种分配算法：

(1) 先来先服务。当有多个进程对同一设备提出I/O请求时，该算法是根据诸进程对某设备提出请求的先后次序，将这些进程排成一个设备请求队列，设备分配程序总是把设备首先分配给队首进程。

(2) 优先级高者优先。在进程调度中的这种策略，是优先权高的进程优先获得处理机。如果对这种高优先权进程所提出的I/O请求也赋予高优先权，显然有助于这种进程尽快完成。

3）设备分配中的安全性

从进程运行的安全性考虑，设备分配有以下两种方式。

（1） 安全分配方式

在这种分配方式中，每当进程发出I/O请求后，便进入阻塞状态，直到其I/O操作完成时才被唤醒。在采用这种分配策略时，一旦进程已经获得某种设备(资源)后便阻塞，使该进程不可能再请求任何资源，而在它运行时又不保持任何资源。因此，这种分配方式已经摒弃了造成死锁的四个必要条件之一的“请求和保持”条件，从而使设备分配是安全的。其缺点是进程进展缓慢，即CPU与I/O设备是串行工作的。

（2） 不安全分配方式

在这种分配方式中，进程在发出I/O请求后仍继续运行，需要时又发出第二个I/O请求、 第三个I/O请求等。仅当进程所请求的设备已被另一进程占用时，请求进程才进入阻塞状态。这种分配方式的优点是，一个进程可同时操作多个设备，使进程推进迅速。其缺点是分配不安全，因为它可能具备“请求和保持”条件，从而可能造成死锁。

因此，在设备分配程序中，还应再增加一个功能，以用于对本次的设备分配是否会发生死锁进行安全性计算，仅当计算结果说明分配是安全的情况下才进行设备分配。

3. 独占设备的分配程序

1)基本的设备分配程序

（1） 分配设备

首先根据I/O请求中的物理设备名，查找系统设备表(SDT)，从中找出该设备的DCT，再根据DCT中的设备状态字段，可知该设备是否正忙。若忙，便将请求I/O进程的PCB挂在设备队列上；否则，便按照一定的算法来计算本次设备分配的安全性。如果不会导致系统进入不安全状态，便将设备分配给请求进程；否则，仍将其PCB插入设备等待队列。

（2）分配控制器

在系统把设备分配给请求I/O的进程后，再到其DCT中找出与该设备连接的控制器的COCT，从COCT的状态字段中可知该控制器是否忙碌。若忙，便将请求I/O进程的PCB挂在该控制器的等待队列上；否则，便将该控制器分配给进程。

（3）分配通道

在该COCT中又可找到与该控制器连接的通道的CHCT，再根据CHCT内的状态信息，可知该通道是否忙碌。若忙，便将请求I/O的进程挂在该通道的等待队列上；否则，将该通道分配给进程。只有在设备、控制器和通道三者都分配成功时，这次的设备分配才算成功。然后，便可启动该I/O设备进行数据传送。

2) 设备分配程序的改进

仔细研究上述基本的设备分配程序后可以发现:

① 进程是以物理设备名来提出I/O请求的;

② 采用的是单通路的I/O系统结构，容易产生“瓶颈”现象。为此，应从以下两方面对基本的设备分配程序加以改进，以使独占设备的分配程序具有更强的灵活性，并提高分配的成功率。

（1）增加设备的独立性

为了获得设备的独立性，进程应使用逻辑设备名请求I/O。这样，系统首先从SDT中找出第一个该类设备的DCT。若该设备忙，又查找第二个该类设备的DCT，仅当所有该类设备都忙时，才把进程挂在该类设备的等待队列上；而只要有一个该类设备可用，系统便进一步计算分配该设备的安全性。

(2) 考虑多通路情况

为了防止在I/O系统中出现“瓶颈”现象，通常都采用多通路的I/O系统结构。此时对控制器和通道的分配同样要经过几次反复，即若设备(控制器)所连接的第一个控制器(通道)忙时，应查看其所连接的第二个控制器(通道)，仅当所有的控制器(通道)都忙时，此次的控制器(通道)分配才算失败，才把进程挂在控制器(通道)的等待队列上。而只要有一个控制器(通道)可用，系统便可将它分配给进程。

5.5.4. 逻辑设备名到物理设备名映射的实现

1. 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)，在该表的每个表目中包含了三项：逻辑设备名、物理设备名和设备驱动程序的入口地址。

当进程用逻辑设备名请求分配I/O设备时，系统为它分配相应的物理设备，并在LUT上建立一个表目，填上应用程序中使用的逻辑设备名和系统分配的物理设备名，以及该设备驱动程序的入口地址。

当以后进程再利用该逻辑设备名请求I/O操作时，系统通过查找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/printe	5	2046
⋮	⋮	⋮

(a)

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

(b)

逻辑设备表

2. LUT的设置问题

LUT的设置可采取两种方式：

第一种方式是在整个系统中只设置一张LUT。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使用相同的逻辑设备名。这种方式主要用于单用户系统中。

第二种方式是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，同时也为之建立一张LUT，并将该表放入进程的PCB中。由于通常在多用户系统中，都配置了系统设备表，故此时的逻辑设备表可以采用图 5-19(b)中的格式。

6.5.4 SPOOLING技术

1．什么是SPOOLing

为了缓和CPU的高速性与I/O设备低速性间的矛盾而引入了脱机输入、脱机输出技术。该技术是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。事实上，当系统中引入了多道程序技术后，完全可以利用其中的一道程序，来模拟脱机输入时的外围控制机功能，把低速I/O设备上的数据传送到高速磁盘上；再用另一道程序来模拟脱机输出时外围控制机的功能，把数据从磁盘传送到低速输出设备上。

这样，便可在主机的直接控制下，实现脱机输入、输出功能。此时的外围操作与CPU对数据的处理同时进行，我们把这种在联机情况下实现的同时外围操作称为SPOOLing(Simultaneaus Periphernal Operating On Line)，或称为假脱机操作。

2．SPOOLing系统的组成

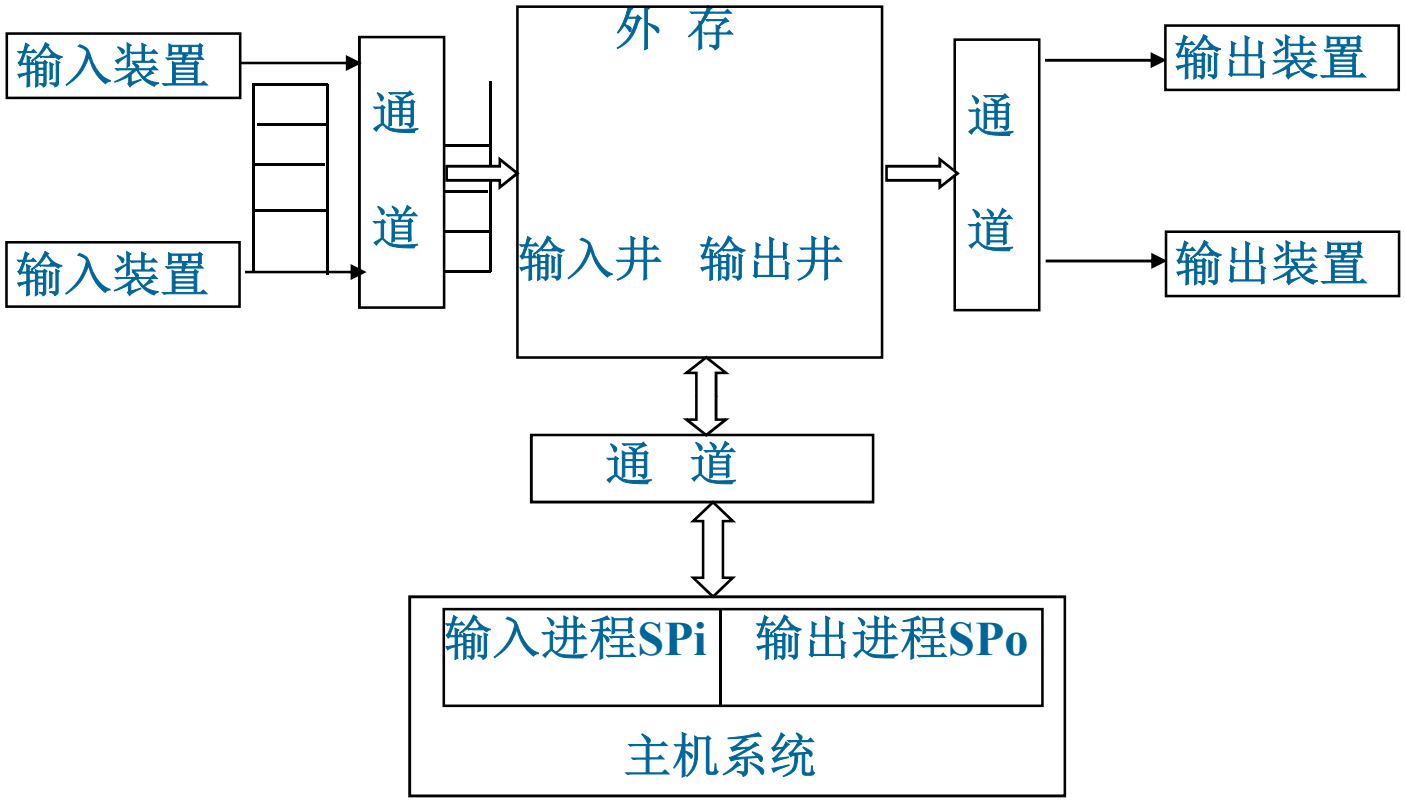
SPOOLing系统必须建立在具有多道程序功能的操作系统上，而且还应有高速随机外存的支持，这通常是采用磁盘存储技术。

SPOOLing系统主要有以下三部分：

(1) 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，用于暂存用户程序的输出数据。

(2) 输入缓冲区和输出缓冲区。为了缓和CPU和磁盘之间速度不匹配的矛盾，在内存中要开辟两个缓冲区：输入缓冲区和输出缓冲区。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。

(3) 输入进程SPi和输出进程SPo。这里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。


外 存
输入装置
输入装置
通
道
输入井
输出井
通
道
输出装置
输出装置
通 道
输入进程SPi
输出进程SPo
主机系统

3．共享打印机

打印机是经常要用到的独占设备。利用SPOOLing技术，可将之改造为一台可供多个用户共享的设备。共享打印机技术已被广泛地用于多用户系统和局域网络中。当用户进程请求打印输出时，SPOOLing系统同意为它打印输出，但并不真正立即把打印机分配给该用户进程，而只为它做两件事：

① 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；

② 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。如果还有进程要求打印输出，系统仍可接受该请求，也同样为该进程做上述两件事。

如果打印机空闲，输出进程将从请求打印队列的队首取出一张请求打印表，根据表中的要求将要打印的数据，从输出井传送到内存缓冲区，再由打印机进行打印。打印完后，输出进程再查看请求打印队列中是否还有等待打印的请求表。若有，又取出队列中的第一张表，并根据其中的要求进行打印，如此下去，直至请求打印队列为空，输出进程才将自己阻塞起来。仅当下次再有打印请求时，输出进程才被唤醒。

4．SPOOLing系统的特点

SPOOLing系统具有如下主要特点：

(1) 提高了I/O的速度。

(2)将独占设备改造为共享设备。

(3) 实现了虚拟设备功能。

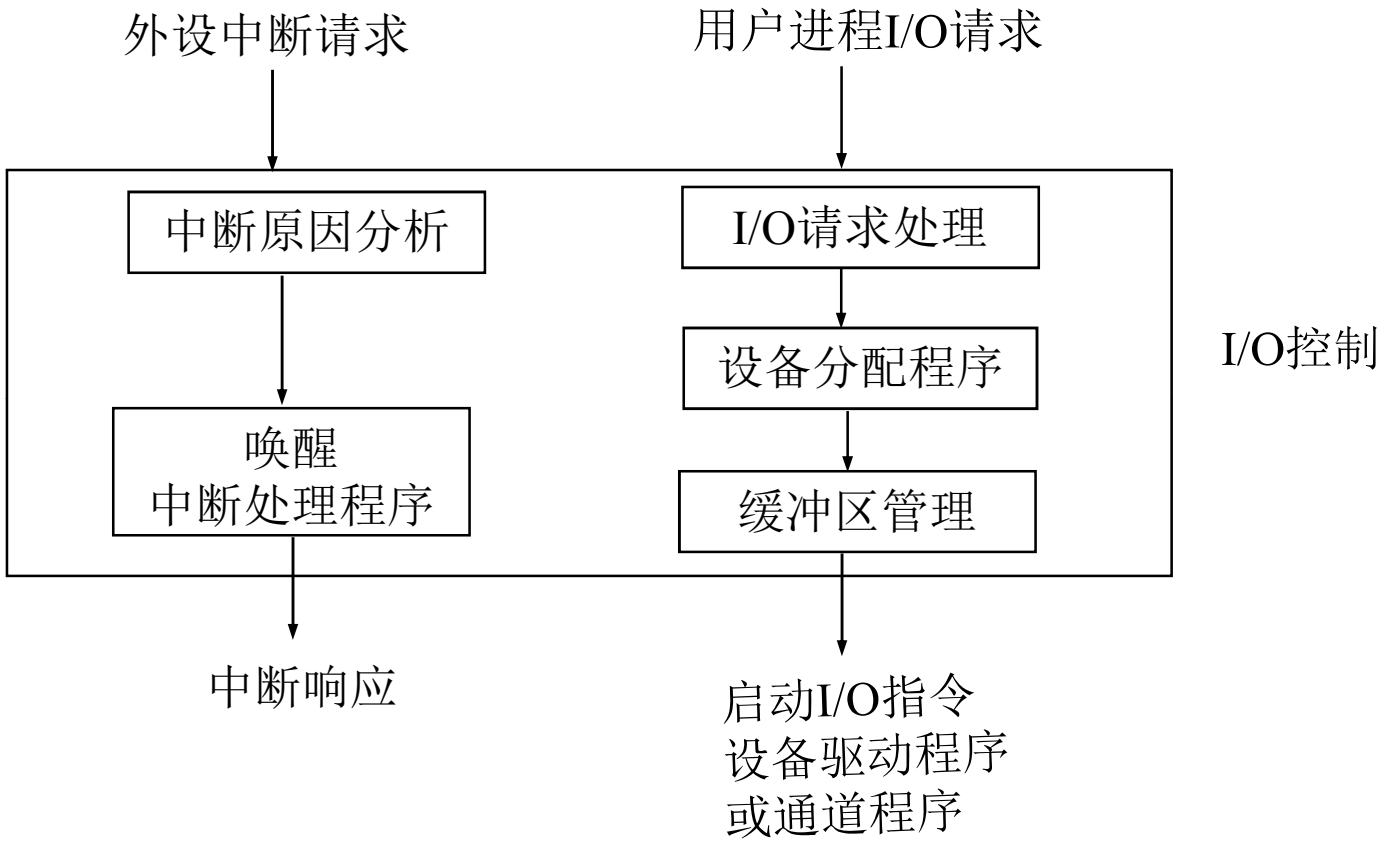
4．守护进程（demon）

人们对假脱机方案进行了一些修改，如取消该方案中的假脱机管理进程，为打印机建立一个守护进程，由它执行一部分原来由假脱机管理进程实现的功能。如为用户在磁盘缓冲区中申请一个空闲盘块，并将要打印的数据送入其中，将该盘块的首址返回给请求进程。另一部分由请求进程自己完成，每个要求打印的进程首先生成一份要求打印的文件，其中包含对打印的要求和指向装有打印输出数据盘块的指针等信息，然后将用户请求打印文件放入假脱机文件队列中。

守护进程是允许使用打印机的唯一进程。所有需要使用打印机进行打印的进程都须将一份要求打印的文件放在假脱机文件队列（目录）中。如果守护进程正在睡眠，便将它唤醒，由它按照目录中第一个文件中的说明进行打印，打印完成后，再按照目录中第二个文件中的说明进行打印，如此逐份文件的进行打印，直到目录中的全部文件打印完毕，守护进程无事可做，又去睡眠，等待用户进程再次发来打印请求。

除了打印守护进程之外，还可能有许多其它的守护进程，如服务器守护进程和网络守护进程等。事实上，凡是需要将独占设备改造为可供多个进程共享的设备时，都要为该设备配置一个守护进程和一个假脱机文件队列（目录）。同样，守护进程是允许使用该独占设备的唯一进程，所有其他进程都不能直接使用该设备，只能将对该设备的使用要求写入一份文件中，放在假脱机目录中。由守护进程按照目录中的文件依次来完成诸进程对该设备的的请求，这样就把一台独占设备改造为可为多个进程共享的设备。

6.6 逻辑I/O系统


外设中断请求
用户进程I/O请求
中断原因分析
I/O请求处理
设备分配程序
唤醒
中断处理程序
缓冲区管理
I/O控制
中断响应
启动I/O指令
设备驱动程序
或通道程序

逻辑I/O系统是I/O系统的最高层软件，在它下面的是设备驱动程序。在逻辑I/O系统中，包括了执行所有设备公有操作的软件，具体如下：

1. 设备驱动程序的统一接口

为了使所有的设备驱动程序有着统一的接口。

一方面，要求每个设备驱动程序与OS之间都有着相同的接口，或者相近的接口，这样会使添加一个新的设备驱动程序变得很容易，方便了开发人员对设备驱动程序的编制。

另一方面，将抽象的设备名转换为具体的物理设备名，并进一步可以找到相应物理设备的驱动程序入口。

此外，还应对设备进行保护，禁止用户直接访问设备，以防止无权访问的用户使用。

2. 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；

3. **差错控制**，由于在I/O操作中的绝大多数错误都与设备有关。错误可分为以下两类。

（1）暂时性错误。暂时性错误是因发生暂时性事件引起的，如电源的波动。它可以通过重试操作来纠正。例如，在网络传输中，会经常发生在网络中传输的数据包丢失或延误性的暂时性错误。这种错误可以通过重新传送来纠正。一般的，设备出现故障后，主要由设备驱动程序处理，而设备独立性软件只处理那些设备驱动程序无法处理的错误。

（2）持久性错误。持久性错误是由持久性故障引起的，如电源掉电、磁盘上出现划痕或者在计算中发生除以零的情况。有些错误时只要重复执行相同的程序就会再现。要排除持久性错误，通常需要查清发生错误的原因。

有某些持久性硬件错误可由操作系统进行有效的处理，而不用涉及高层软件。

如　磁盘上的少数盘块遭到破坏

4. 对独立设备的分配与回收

在系统中有两类设备：独占设备和共享设备。对于独占设备，为了避免诸进程对独占设备的争夺，必须由系统来统一分配。

5. 提供独立于设备的逻辑块

不同类型的设备，信息交换单位、读取和传输速率都各不相同。（如字符型设备以单个字符为单位，块设备是以一个数据块为单位）即使同一类型的设备，其信息交换单位大小也是有差异的（不同磁盘由于扇区大小的不同，可能造成数据块大小的不一致）。

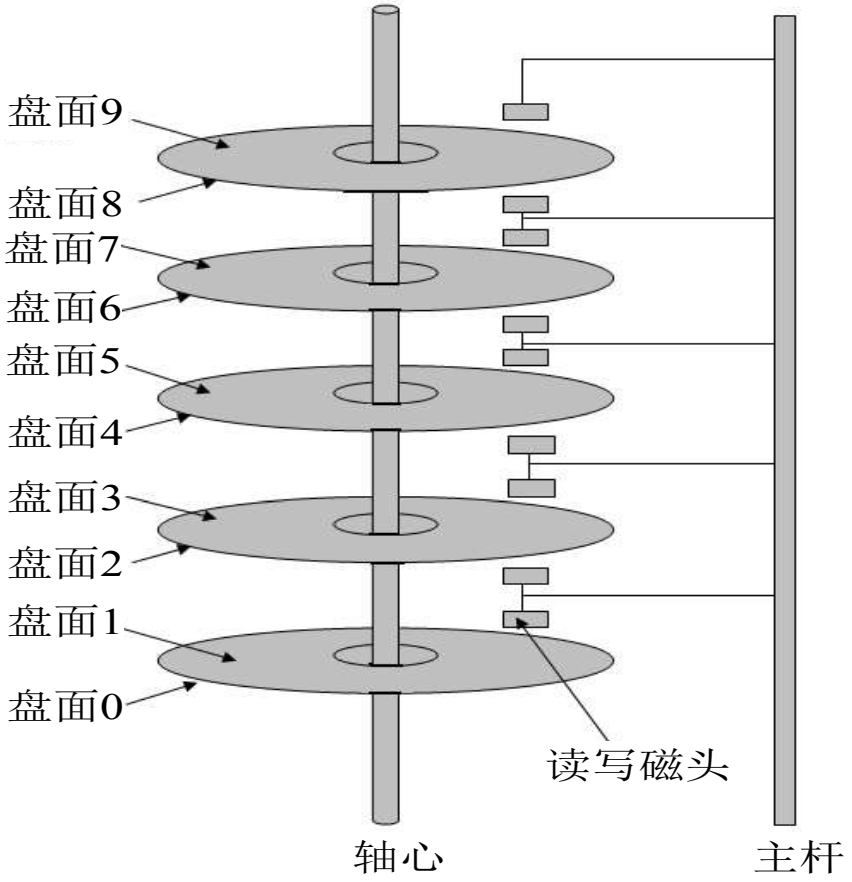
设备独立性软件应负责隐藏这些差异，并向高层软件提供大小统一的逻辑数据块。

6.8 磁盘存储器的管理

6.8.1 磁盘性能简述

1．数据的组织和格式

磁盘设备可包括一或多个物理盘片，每个磁盘片分一个或两个存储面(surface)(见图5-23(a))，每个磁盘面被组织成若干个同心环，这种环称为磁道(track)，各磁道之间留有必要的间隙。为使处理简单起见，在每条磁道上可存储相同数目的二进制位。这样，磁盘密度即每英寸中所存储的位数，显然是内层磁道的密度较外层磁道的密度高。每条磁道又被逻辑上划分成若干个扇区(sectors)，软盘大约为8～32个扇区，硬盘则可多达数百个，图5-23(b)显示了一个磁道分成8个扇区。一个扇区称为一个盘块(或数据块)，常常叫做磁盘扇区。各扇区之间保留一定的间隙。



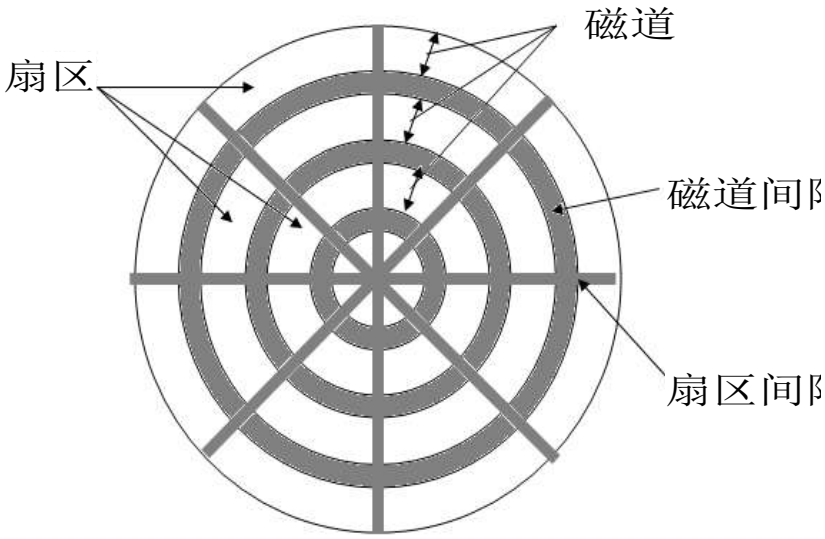


图5-23　磁盘的结构和布局

一个物理记录存储在一个扇区上，磁盘上存储的物理记录块数目是由扇区数、磁道数以及磁盘面数所决定的。例如，一个10 GB容量的磁盘，有8个双面可存储盘片，共16个存储面(盘面)，每面有16 383个磁道(也称柱面)，63个扇区。

为了提高磁盘的存储容量，充分利用磁盘外面磁道的存储能力，现代磁盘不再把内外磁道划分为相同数目的扇区，而是利用外层磁道容量较内层磁道大的特点，将盘面划分成若干条环带，使得同一环带内的所有磁道具有相同的扇区数。显然，外层环带的磁道拥有较内层环带的磁道更多的扇区。为了减少这种磁道和扇区在盘面分布的几何形式变化对驱动程序的影响，大多数现代磁盘都隐藏了这些细节，向操作系统提供虚拟几何的磁盘规格，而不是实际的物理几何规格。

为了在磁盘上存储数据，必须先将磁盘低级格式化。图5-24示出了一种温盘(温切斯特盘)中一条磁道格式化的情况。其中每条磁道含有30个固定大小的扇区，每个扇区容量为600个字节，其中512个字节存放数据，其余的用于存放控制信息。每个扇区包括两个字段：

（1）标识符字段，其中一个字节的SYNCH具有特定的位图像，作为该字段的定界符，利用磁道号、 磁头号及扇区号三者来标识一个扇区；CRC字段用于段校验。

(2) 数据字段，其中可存放512个字节的数据。

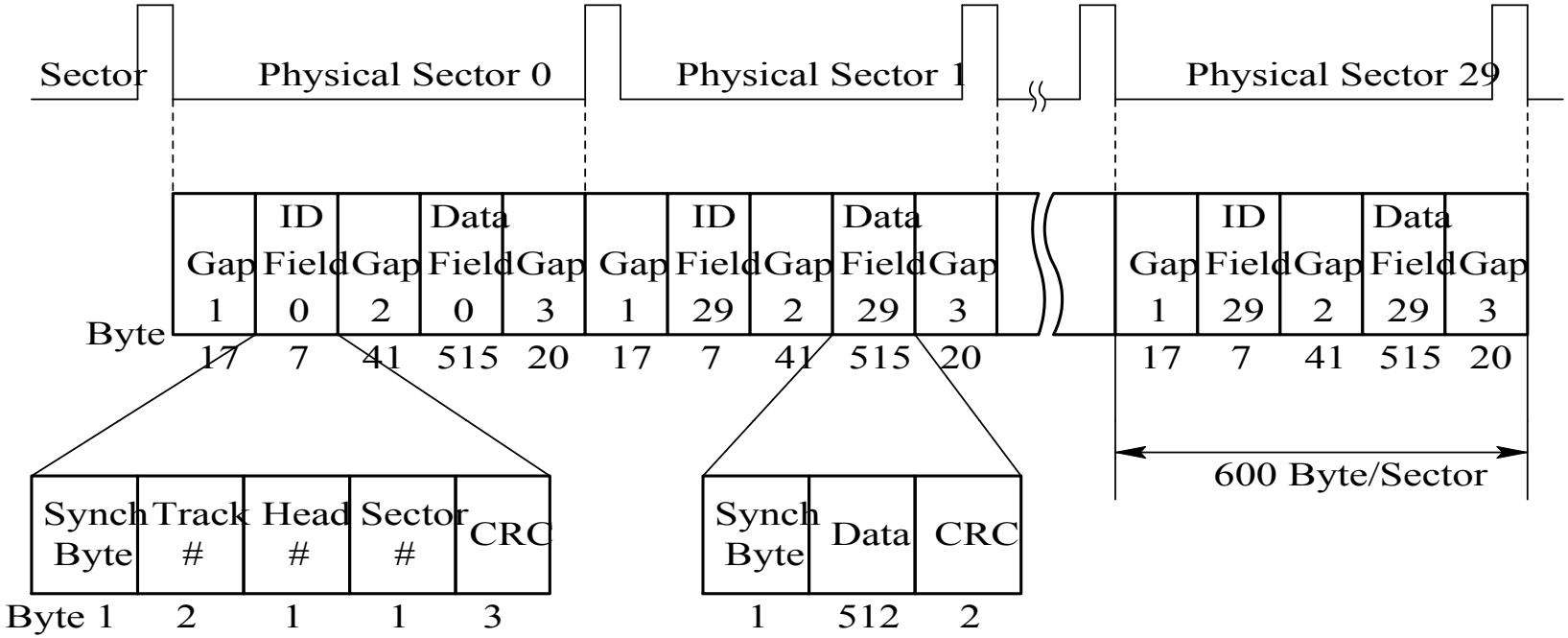

Sector
Physical Sector 0
Physical Sector 1
Physical Sector 29
Gap 1
ID Field 0
Gap 2
Data Field 0
Gap 3
Gap 1
ID Field 29
Gap 2
Data Field 29
Gap 3
Gap 1
ID Field 29
Gap 2
Data Field 29
Gap 3
Byte
17 7 41 515 20 17 7 41 515 20 17 7 41 515 20
600 Byte/Sector
Synch Byte
Track #
Head #
Sector #
CRC
Byte 1 2 1 1 3
Synch Byte
Data
CRC
1 512 2

图5-24　磁盘的格式化

2．磁盘的类型

对磁盘，可以从不同的角度进行分类。最常见的有：将磁盘分成硬盘和软盘、单片盘和多片盘、固定头磁盘和活动头(移动头)磁盘等。下面仅对固定头磁盘和移动头磁盘做些介绍。

1）固定头磁盘

这种磁盘在每条磁道上都有一读/写磁头，所有的磁头都被装在一刚性磁臂中。通过这些磁头可访问所有各磁道，并进行并行读/写，有效地提高了磁盘的I/O速度。这种结构的磁盘主要用于大容量磁盘上。

2）移动头磁盘

每一个盘面仅配有一个磁头，也被装入磁臂中。为能访问该盘面上的所有磁道，该磁头必须能移动以进行寻道。可见，移动磁头仅能以串行方式读/写，致使其I/O速度较慢；但由于其结构简单，故仍广泛应用于中小型磁盘设备中。在微型机上配置的温盘和软盘都采用移动磁头结构，故本节主要针对这类磁盘的I/O进行讨论。

3．磁盘访问时间

1）寻道时间T_S

这是指把磁臂(磁头)移动到指定磁道上所经历的时间。该时间是启动磁臂的时间s与磁头移动n条磁道所花费的时间之和，即

$$T_s = m \times n + s$$

其中，m是一常数，与磁盘驱动器的速度有关。对于一般磁盘，m=0.2；对于高速磁盘，m≤0.1，磁臂的启动时间约为2 ms。这样，对于一般的温盘，其寻道时间将随寻道距离的增加而增大，大体上是5～30 ms。

2) 旋转延迟时间T_r

这是指定扇区移动到磁头下面所经历的时间。不同的磁盘类型中，旋转速度至少相差一个数量级，如软盘为300 r/min，硬盘一般为7200～15 000 r/min，甚至更高。对于磁盘旋转延迟时间而言，如硬盘，旋转速度为15 000 r/min，每转需时4 ms，平均旋转延迟时间T_r为2 ms；而软盘，其旋转速度为 300 r/min或600 r/min，这样，平均T_r为50～100 ms。

3) 传输时间T_t

这是指把数据从磁盘读出或向磁盘写入数据所经历的时间。T_t的大小与每次所读/写的字节数b和旋转速度有关:

$$T_t = \frac{b}{rN}$$

其中，r为磁盘每秒钟的转数；N为一条磁道上的字节数，当一次读/写的字节数相当于半条磁道上的字节数时，T_t与T_r相同。因此，可将访问时间T_a表示为

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

由上式可以看出，在访问时间中，寻道时间和旋转延迟时间基本上都与所读/写数据的多少无关，而且它通常占据了访问时间中的大头。例如，我们假定寻道时间和旋转延迟时间平均为20 ms，而磁盘的传输速率为10 MB/s，如果要传输10 KB的数据，此时总的访问时间为21 ms，可见传输时间所占比例是非常小的。当传输100 KB数据时，其访问时间也只是30 ms，即当传输的数据量增大10倍时，访问时间只增加约50%。目前磁盘的传输速率已达80 MB/s以上，数据传输时间所占的比例更低。可见，适当地集中数据(不要太零散)传输，将有利于提高传输效率。

6.8.2 磁盘调度

1. 先来先服务(FCFS，First Come First Served)

这是一种最简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单，且每个进程的请求都能依次地得到处理，不会出现某一进程的请求长期得不到满足的情况。但此算法由于未对寻道进行优化，致使平均寻道时间可能较长。图5-25示出了有9个进程先后提出磁盘I/O请求时，按FCFS算法进行调度的情况。这里将进程号(请求者)按他们发出请求的先后次序排队。这样，平均寻道距离为55.3条磁道，与后面即将讲到的几种调度算法相比，其平均寻道距离较大，故FCFS算法仅适用于请求磁盘I/O的进程数目较少的场合。

(从 100 号磁道开始)	
被访问的下 一个磁道号	移动距离 (磁道数)
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度: 55.3	

图5-25　FCFS调度算法

2．最短寻道时间优先(SSTF，Shortest Seek Time First)

该算法选择这样的进程：其要求访问的磁道与当前磁头所在的磁道距离最近，以使每次的寻道时间最短。但这种算法不能保证平均寻道时间最短。图5-26示出了按SSTF算法进行调度时，各进程被调度的次序、每次磁头移动的距离，以及9次调度磁头平均移动的距离。比较图5-25和图5-26可以看出，SSTF算法的平均每次磁头移动距离明显低于FCFS的距离，因而SSTF较之FCFS有更好的寻道性能，故过去曾一度被广泛采用。

(从 100 号磁道开始)	
被访问的下 一个磁道号	移动距离 (磁道数)
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
平均寻道长度: 27.5	

图5-26　SSTF调度算法

3．扫描(SCAN)算法

1）进程“饥饿”现象

SSTF算法虽然能获得较好的寻道性能，但却可能导致某个进程发生“饥饿”(Starvation)现象。因为只要不断有新进程的请求到达，且其所要访问的磁道与磁头当前所在磁道的距离较近，这种新进程的I/O请求必然优先满足。对SSTF算法略加修改后所形成的SCAN算法，即可防止老进程出现“饥饿”现象。

2） SCAN算法

该算法不仅考虑到欲访问的磁道与当前磁道间的距离，更优先考虑的是磁头当前的移动方向。例如，当磁头正在自里向外移动时，SCAN算法所考虑的下一个访问对象，应是其欲访问的磁道既在当前磁道之外，又是距离最近的。这样自里向外地访问，直至再无更外的磁道需要访问时，才将磁臂换向为自外向里移动。这时，同样也是每次选择这样的进程来调度，即要访问的磁道在当前位置内距离最近者，这样，磁头又逐步地从外向里移动，直至再无更里面的磁道要访问，从而避免了出现“饥饿”现象。由于在这种算法中磁头移动的规律颇似电梯的运行，因而又常称之为电梯调度算法。图 5-27示出了按SCAN算法对9个进程进行调度及磁头移动的情况。

(从 100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
平均寻道长度：27.8	

图5-27　SCAN调度算法示例

4．循环扫描(CSCAN)算法

SCAN算法既能获得较好的寻道性能，又能防止“饥饿”现象，故被广泛用于大、中、小型机器和网络中的磁盘调度。但SCAN也存在这样的问题：当磁头刚从里向外移动而越过了某一磁道时，恰好又有一进程请求访问此磁道，这时，该进程必须等待，待磁头继续从里向外，然后再从外向里扫描完所有要访问的磁道后，才处理该进程的请求，致使该进程的请求被大大地推迟。

为了减少这种延迟，CSCAN算法规定磁头单向移动，例如，只是自里向外移动，当磁头移到最外的磁道并访问后，磁头立即返回到最里的欲访问的磁道，亦即将最小磁道号紧接着最大磁道号构成循环，进行循环扫描。采用循环扫描方式后，上述请求进程的请求延迟将从原来的2T减为T + Smax，其中，T为由里向外或由外向里单向扫描完要访问的磁道所需的寻道时间，而Smax是将磁头从最外面被访问的磁道直接移到最里面欲访问的磁道(或相反)的寻道时间。图5-28示出了CSCAN算法对9个进程调度的次序及每次磁头移动的距离。

(从 100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度: 35.8	

CSCAN调度算法示例

5．NStepSCAN和FSCAN调度算法

1） NStepSCAN算法

在SSTF、 SCAN及CSCAN几种调度算法中，都可能会出现磁臂停留在某处不动的情况，例如，有一个或几个进程对某一磁道有较高的访问频率，即这个(些)进程反复请求对某一磁道的I/O操作，从而垄断了整个磁盘设备。我们把这一现象称为“磁臂粘着”(Armstickiness)。在高密度磁盘上容易出现此情况。

N步SCAN算法是将磁盘请求队列分成若干个长度为N的子队列，磁盘调度将按FCFS算法依次处理这些子队列。而每处理一个队列时又是按SCAN算法，对一个队列处理完后，再处理其他队列。当正在处理某子队列时，如果又出现新的磁盘I/O请求，便将新请求进程放入其他队列，这样就可避免出现粘着现象。当N值取得很大时，会使N步扫描法的性能接近于SCAN算法的性能；当N=1时，N步SCAN算法便蜕化为FCFS算法。

2） FSCAN算法

FSCAN算法实质上是N步SCAN算法的简化，即FSCAN只将磁盘请求队列分成两个子队列。一个是由当前所有请求磁盘I/O的进程形成的队列，由磁盘调度按SCAN算法进行处理。在扫描期间，将新出现的所有请求磁盘I/O的进程，放入另一个等待处理的请求队列。这样，所有的新请求都将被推迟到下一次扫描时处理。

6.7 Linux的设备管理

Linux把设备看作特殊的文件,系统通过处理文件的接口———虚拟文件系统(VFS)来管理和控制各种设备。

6.7.1 逻辑I/O 管理

1.Linux设备的分类

字符设备、块设备和网络设备。

2.设备映射

Linux系统提供了统一命名文件和设备的方法,即以文件系统的路径名来命名设备。使用设备类型、主设备号、次设备号来识别设备。与普通的目录和文件一样,每个设备也使用一个VFSinode来描述,其中包含设备类型(块设备或字符设备)的主、次设备号。对设备的操作也是通过对文件操作的file_operations结构体来调用驱动程序的设备服务子程序。

3. 缓冲区

每个缓冲区与一个块对应,它相当于磁盘块在内存中的表示。块包含一个或多个扇区,但不能超过一个页面,所以一个页面可以容纳一个或多个内存中的块。目前内核块I/O操作的基本容器由bio结构来表示。它表示了正在现场的(活动的)以片段链表形式组织的块I/O 操作。一个片段是内存中一小块连续的内存区域。每一个块I/O请求都通过一个bio结构体表示。每个请求包含一个或多个块,这些块存储在bio_vec结构体数组中,每个bio_vec结构体描述了一个片段在物理页中的实际位置。每个bio_vec都对应一个页面,这些页面可以不连续存放。整个bio_vec结构体数组构成一个完整的缓冲区,从而保证内核能够方便、高效地完成I/O操作。bio相当于在缓冲区上又封装了一层,使得内核在I/O 操作时只要针对一个或多个内存页面即可,不用再去管理磁盘块的部分。

4. I/O 调度

I/O调度算法叫电梯调度算法。

6.7.2 用户与设备驱动程序

Linux内核是模块化的,内核中的模块可以按需加载, 设备驱动程序以模块形式提供,可以动态地加载和卸载。

系统对设备的控制和操作是由设备驱动程序完成的。设备驱动程序由设备服务子程序和中断处理程序组成。设备服务子程序包括了对设备进行各种操作的代码,中断处理子程序处理设备中断。在Linux中,内核定义了两个数据结构数组,分别称为块设备转换表和字符设备转换表。每个驱动程序在其相应的数组中占一个表项。

用户空间访问设备时,通过open()调用返回的文件描述符找到代表该设备的struct_file结构指针,从而找到设备的i节点,从i节点中检查其类型是块设备还是字符设备,从i节点中提取设备号,通过设备号从块设备或者字符设备转换表中找到相应的设备驱动程序,由file_operations定义设备驱动程序提供给VFS的接口函数。

6.7.3 设备模型

设备模型提供了一个独立的机制来表示设备，并描述设备在系统中的树状结构，使代码重复最小化。用户就可以通过这棵树去遍历所有的设备，建立设备和驱动程序之间的联系，也可以对设备进行归类；引入统一的编程机制，让开发者可以开发出安全、高效的驱动程序。

设备模型的核心结构是kobject，类似面向对象语言中的基类，它嵌入到更大的对象中，用来描述设备模型的组件，用它提供的字段可以创建对象的层次结构。为了方便调试，设备模型开发者决定将设备结构导出为一个文件系统，由此产生了一个新的文件系统—sysfs。sysfs文件系统与kobject结构紧密关联，每个在内核中注册的kobject对象都对应sysfs文件系统中的一个目录。

sysfs文件系统是虚拟的文件系统，它可以产生一个包含所有系统硬件的层次视图，并向用户程序提供详细的内核数据结构信息，用户以一种简单的文件系统方式来观察系统中各种设备的树形结构，给用户提供了一个从用户空间访问内核设备的方法。