第五章 设 备 管 理

5.1 I/O系统

5.2 I/O控制方式

5.3 缓冲管理

5.4 I/O软件

5.5 设备分配

5.6 磁盘存储器的管理

I/O管理的重要性

- I/O设备就像计算机系统的五官和四肢
- I/O性能经常成为系统性能的瓶颈
- CPU性能不等于系统性能，CPU性能越高，与I/O差距越大
- 操作系统庞大复杂的原因之一：资源多、杂，并发，均来自I/O
- I/O速度是导致并发技术的直接原因
- 理解I/O的工作过程与结构是理解操作系统的工作过程与结构的关键
- 与其它OS功能联系密切，特别是文件系统

设备管理的功能

- 设备的分配与回收：表格、分配/回收算法
- 设备的启动
- 设备的事件处理（I/O故障、I/O完成）
- 设备缓冲区管理
- 实现虚拟设备
- 磁盘调度

设备管理目的

- 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
- 防止设备误操作

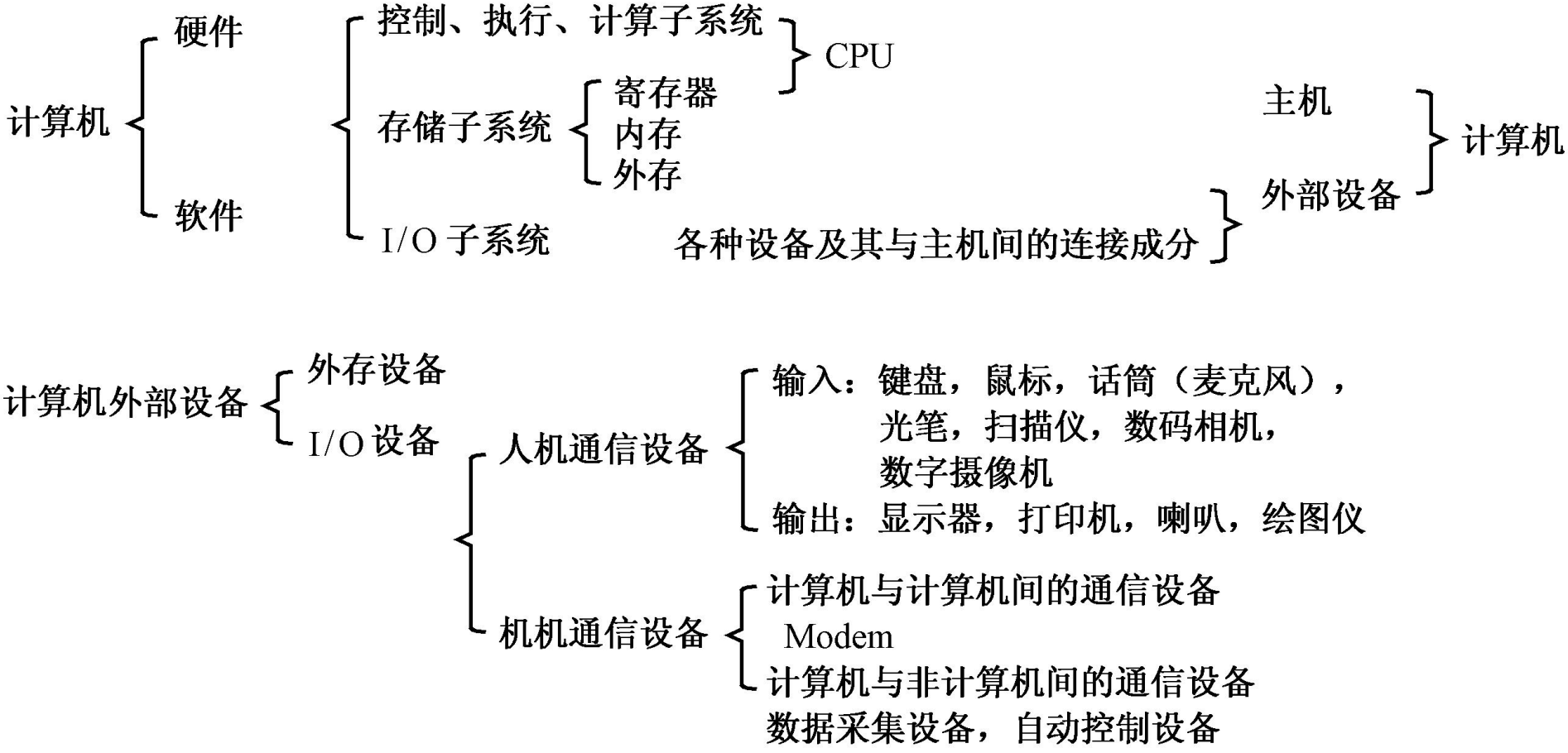
5.1 I/O系统

5.1.1 I/O设备

1．I/O设备的类型

I/O设备的类型繁多，从OS观点看，其重要的性能指标有: 设备使用特性、数据传输速率、数据的传输单位、设备共享属性等。因而可从不同角度对它们进行分类。

外设分类图例


计算机
硬件
软件
控制、执行、计算子系统
CPU
存储子系统
寄存器
内存
外存
I/O 子系统
各种设备及其与主机间的连接成分
主机
外部设备
计算机
计算机外部设备
外存设备
I/O 设备
人机通信设备
输入：键盘，鼠标，话筒（麦克风），
光笔，扫描仪，数码相机，
数字摄像机
输出：显示器，打印机，喇叭，绘图仪
机机通信设备
计算机与计算机间的通信设备
Modem
计算机与非计算机间的通信设备
数据采集设备，自动控制设备

设备的分类

1）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
- 输入/输出设备

2）按传输速率分类

- 低速设备：每秒几个到数百字节。如Modem、键盘
- 中速设备：每秒数千到数万字节。如打印机
- 高速设备：每秒数百K到数兆。如磁盘、磁带

3）按信息交换的单位分类

- 字符设备：I/O传输的单位是字节。如打印机
- 块设备：I/O传输的单位是块。如磁盘、磁带

设备的分类

4）按设备的共享属性分类

- 独占设备：在任一段时间内最多有一个进程占用它，字符设备及磁带机属独占型设备。即临界资源。
- 共享设备：多个进程对它的访问可以交叉进行，除磁带机外的块设备属共享设备。
- 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拟设备。

5.1.1 I/O设备

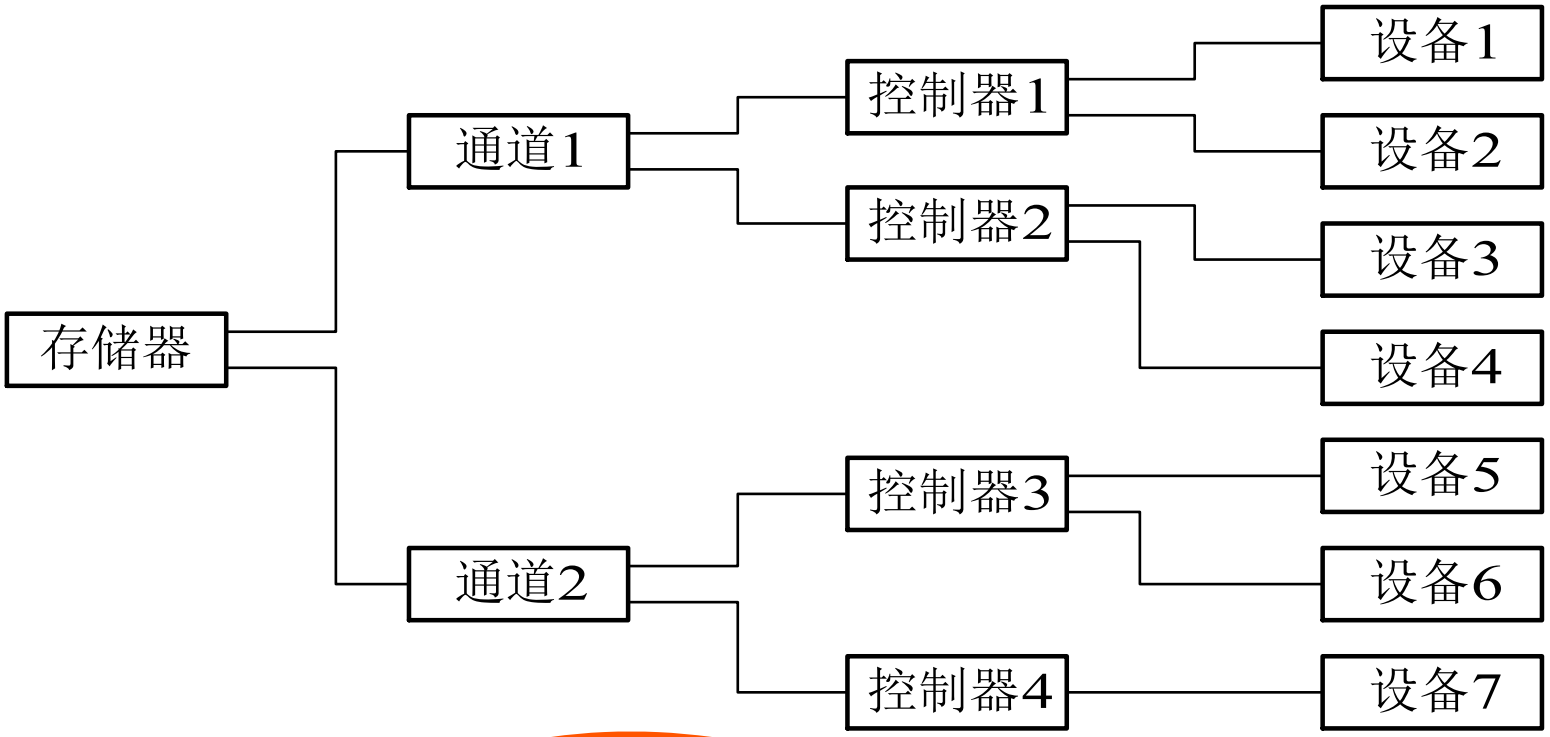


图5-4 单通路I/O系统

5.1.1 I/O设备

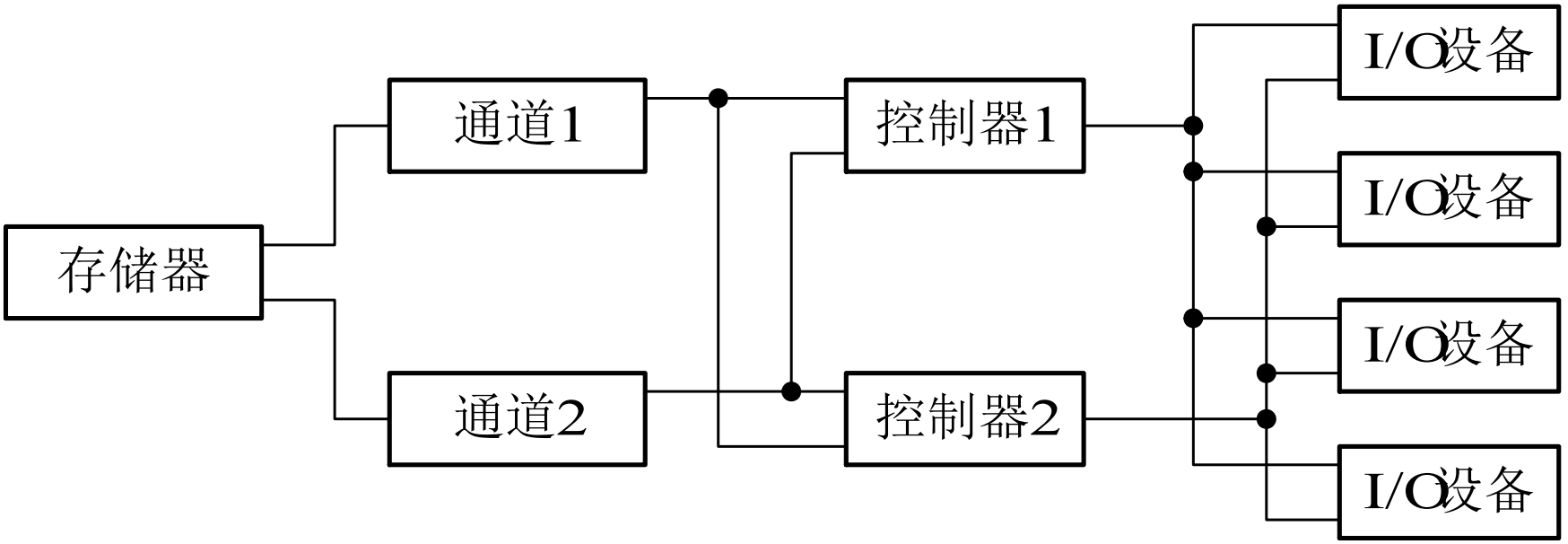


图5-5 多通路I/O系统

5.2 I/O控制(数据传输)方式

- 四种I/O控制方式
  - 程序I/O方式(循环等待，Polling)
  - 中断驱动I/O控制方式(Interrupts)
  - 直接存储器存取（ DMA ）
  - I/O通道控制方式 (Channel)
- 每种方式占用CPU的比例不同，硬件开销不同

5.3 缓冲管理

5.3.1 缓冲的引入

(1) 缓和CPU与I/O设备间速度不匹配的矛盾。事实上，凡在数据到达速率与其离去速率不同的地方，都可设置缓冲区，以缓和它们之间速率不匹配的矛盾。

众所周知，CPU的运算速率远远高于I/O设备的速率，如果没有缓冲区，则在输出数据时，必然会由于打印机的速度跟不上而使CPU停下来等待；然而在计算阶段，打印机又空闲无事。显然，如果在打印机或控制器中设置一缓冲区，用于快速暂存程序的输出数据，以后由打印机“慢慢地”从中取出数据打印，这样，就可提高CPU的工作效率。类似地，在输入设备与CPU之间也设置缓冲区，也可使CPU的工作效率得以提高。

5.3.1 缓冲的引入

(2) 减少对CPU的中断频率，放宽对CPU中断响应时间的限制。在远程通信系统中，如果从远地终端发来的数据仅用一位缓冲来接收，如图5-10(a)所示，则必须在每收到一位数据时便中断一次CPU，这样，对于速率为9.6 Kb/s的数据通信来说，就意味着其中断CPU的频率也为9.6 Kb/s，即每100 μs就要中断CPU一次，而且CPU必须在100 μs内予以响应，否则缓冲区内的数据将被冲掉。

倘若设置一个具有8位的缓冲(移位)寄存器，如图5-10(b)所示，则可使CPU被中断的频率降低为原来的1/8；若再设置一个8位寄存器，如图5-10(c)所示，则又可把CPU对中断的响应时间放宽到800 μs。

5.3.1 缓冲的引入

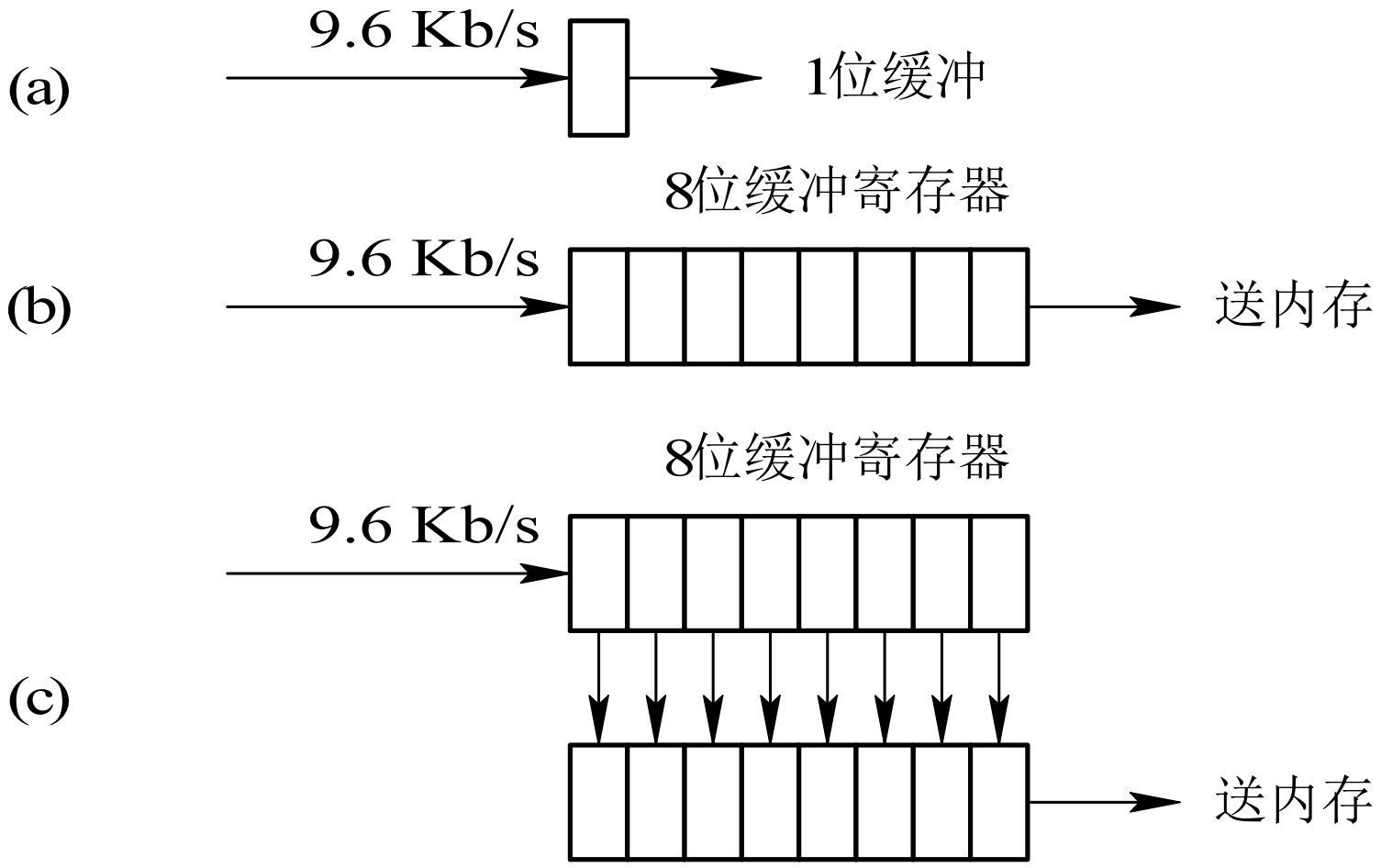


图5-10 利用缓冲寄存器实现缓冲

5.3.1 缓冲的引入

(3) 提高CPU和I/O设备之间的并行性。缓冲的引入可显著地提高CPU和I/O设备间的并行操作程度，提高系统的吞吐量和设备的利用率。例如，在CPU和打印机之间设置了缓冲区后，便可使CPU与打印机并行工作。

5.3.2 单缓冲和双缓冲

1．单缓冲(Single Buffer)

在单缓冲情况下，每当用户进程发出一I/O请求时，操作系统便在主存中为之分配一缓冲区，如图5-11 所示。在块设备输入时，从磁盘把一块数据输入到缓冲区，之后操作系统将该缓冲区中的数据传送到用户区。

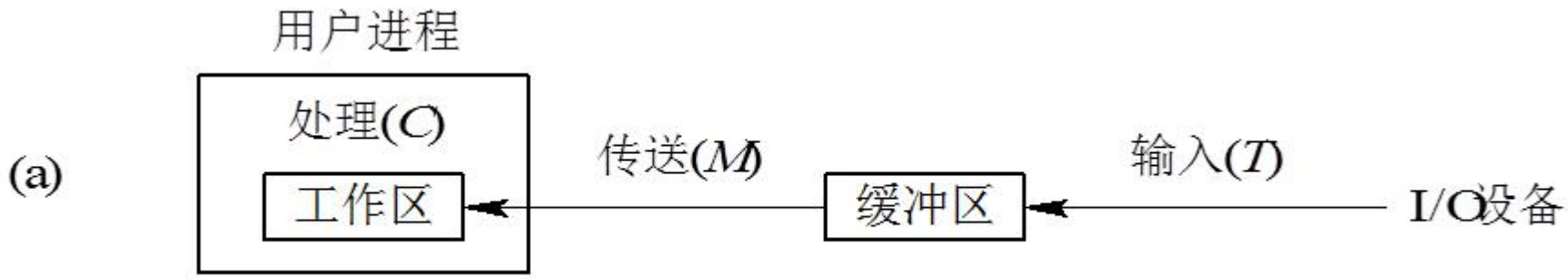


图5-11　单缓冲工作示意图

5.3.2 单缓冲和双缓冲

2．双缓冲(Double Buffer)

为了加快输入和输出速度，提高设备利用率，人们又引入了双缓冲区机制。在设备输入时，先将数据送入第一缓冲区，装满后便转向第二缓冲区。此时操作系统可以从第一缓冲区中移出数据，并送入用户进程(见图5-12)。接着由CPU对数据进行计算。

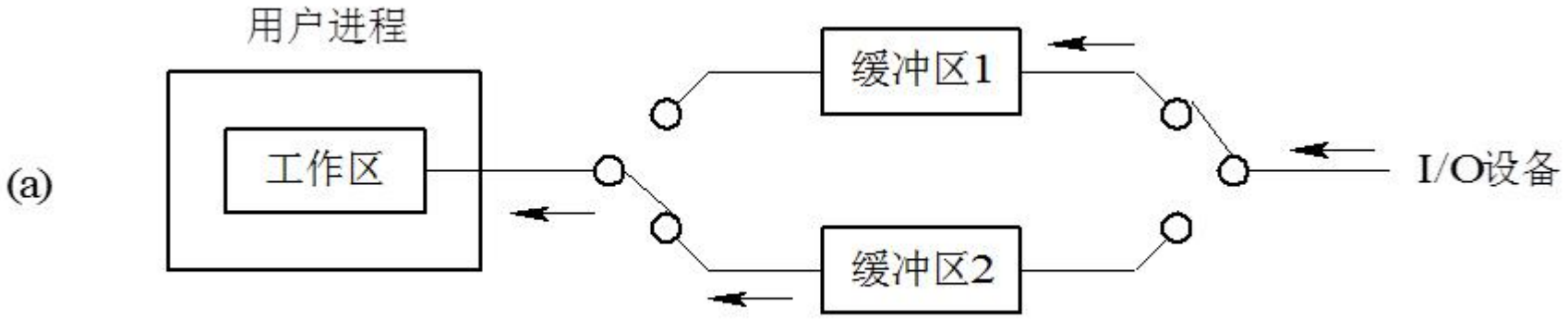


图5-12　双缓冲工作示意图

5.3.2 单缓冲和双缓冲

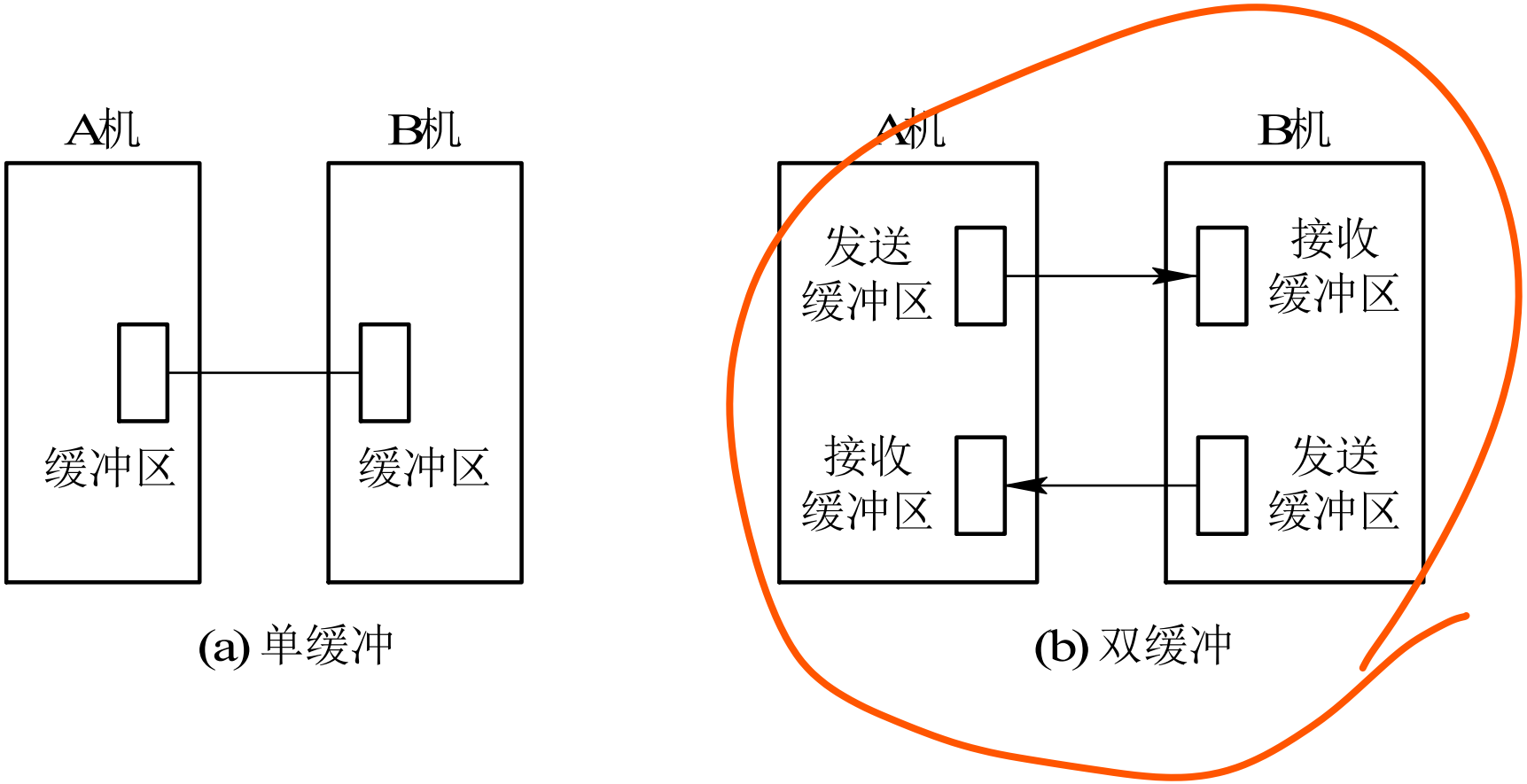


图5-13 双机通信时缓冲区的设置

5.3.3 循环缓冲

1．循环缓冲的组成

(1) 多个缓冲区。在循环缓冲中包括多个缓冲区，其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现行工作缓冲区C，如图5-14所示。

5.3.3 循环缓冲

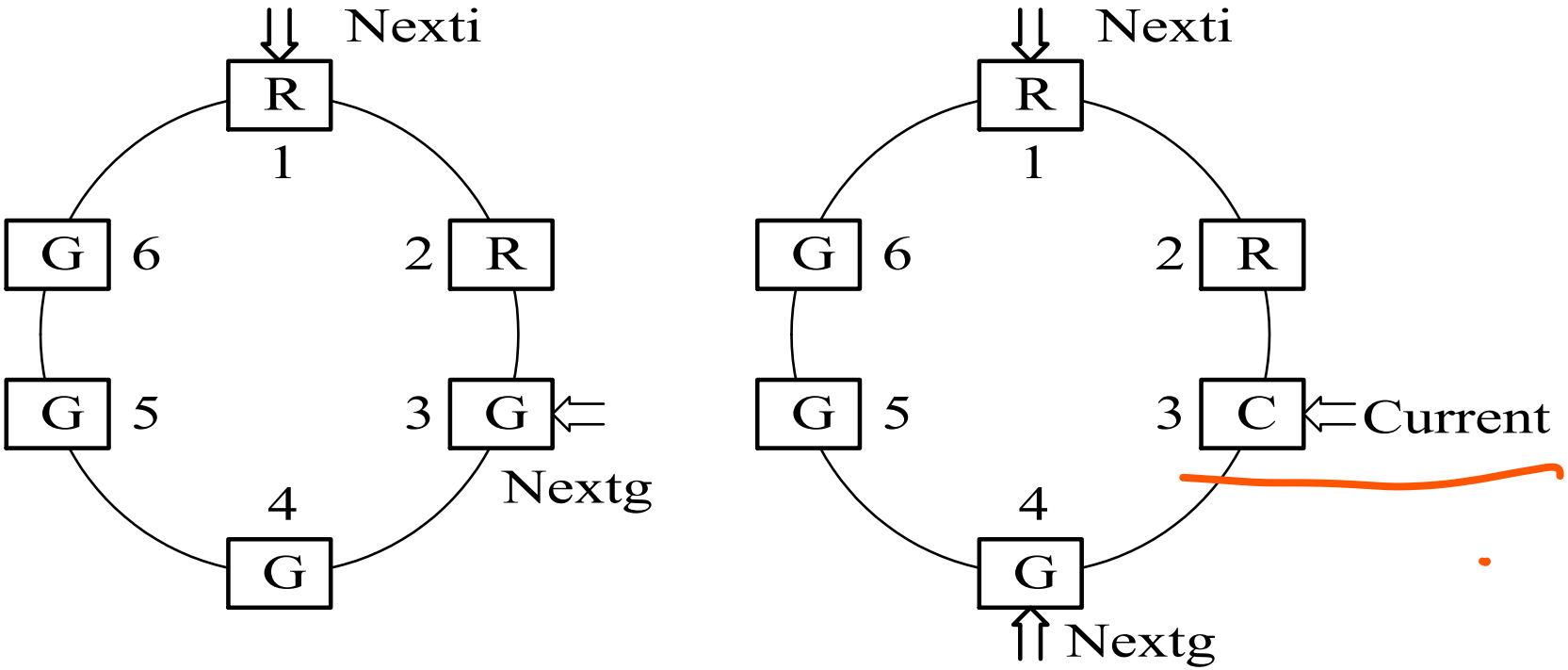


图5-14 循环缓冲

5.3.3 循环缓冲

(2) 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。

5.3.4 缓冲池

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:

- 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。

5.3.4 缓冲池

- 三个缓冲队列：空缓冲队列emq、输入队列inq和输出队列outq。
- 四种工作缓冲区：
  - 用于收容输入数据的工作缓冲区hin
  - 用于提取输入数据的工作缓冲区sin
  - 用于收容输出数据的工作缓冲区hout
  - 用于提取输出数据的工作缓冲区sout

5.3.4 缓冲池

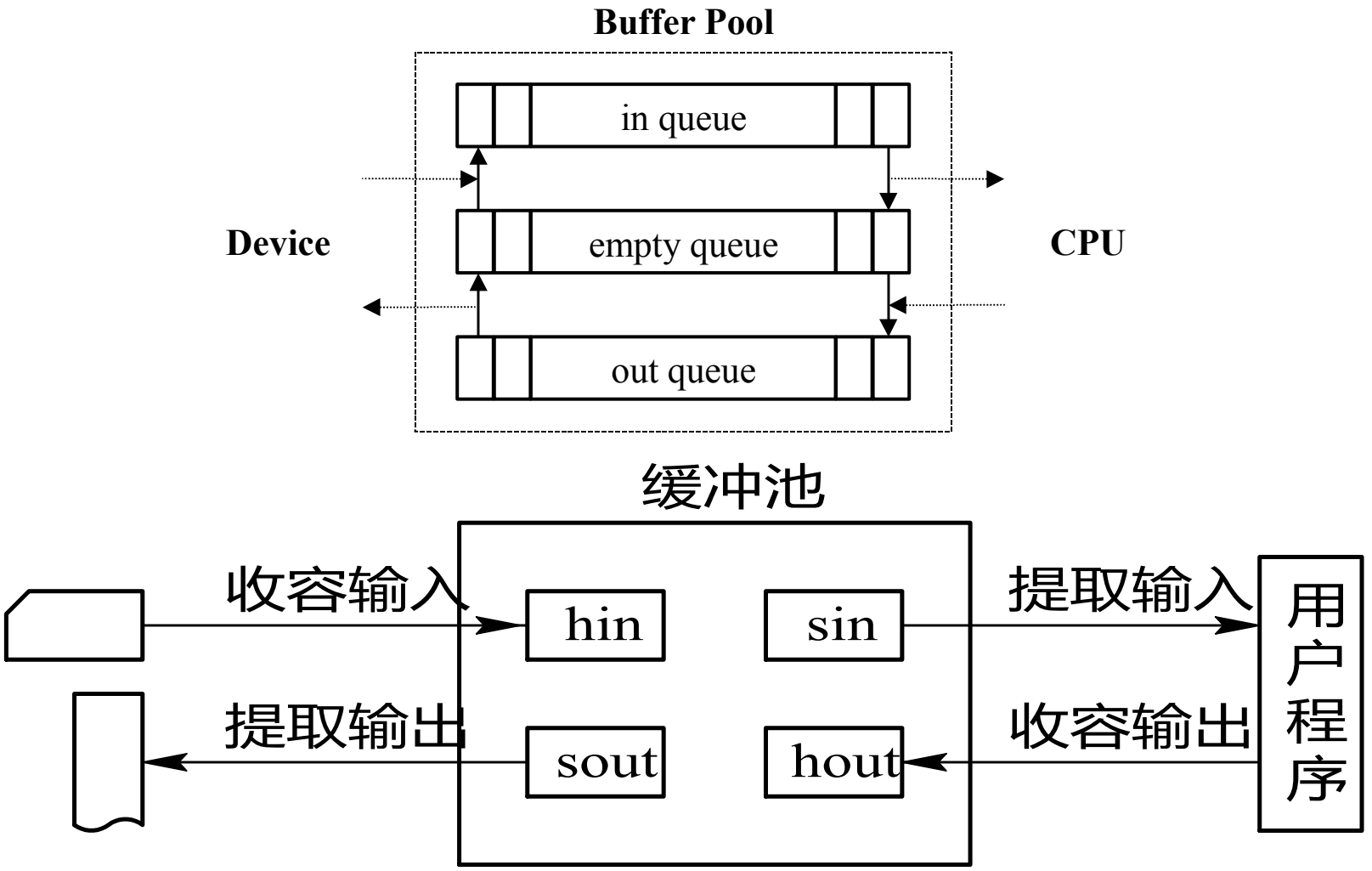


图5-15 缓冲区的工作方式

5.3.4 缓冲池

(1) 收容输入。在输入进程需要输入数据时，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，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，再将该缓冲区挂到空缓冲队列emq上。

5.3.4 缓冲池

(3) 收容输出。当计算进程需要输出时，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，将该缓冲区挂在outq末尾。

(4) 提取输出。由输出进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，将该缓冲区挂在空缓冲队列末尾。

5.4 I/O 软 件

5.4.1 I/O软件的设计目标和原则

计算机系统中包含了众多的I/O设备，其种类繁多，硬件构造复杂，物理特性各异，造成对设备的操作和管理非常复杂和琐碎。

因此，从系统的观点出发，采用多种技术和措施，解决由于外部设备与CPU速度不匹配所引起的问题，提高主机和外设的并行工作能力，提高系统效率，成为操作系统的一个重要目标。

5.4.1 I/O软件的设计目标和原则

另一方面，对设备的操作和管理的复杂性，也给用户的使用带来了极大的困难。用户必须掌握I/O 系统的原理，对接口和控制器及设备的物理特性要有深入了解，这就使计算机的推广应用受到很大限制。

所以，设法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，也是I/O软件设计的一个重要原则。

5.4.1 I/O软件的设计目标和原则

具体而言，I/O软件应达到下面的几个目标：

1) 与具体设备无关

对于I/O系统中许多种类不同的设备，作为程序员，只需要知道如何使用这些资源来完成所需要的操作，而无需了解设备的有关具体实现细节。例如，应用程序访问文件时，不必去考虑被访问的是硬盘、软盘还是CD-ROM。

5.4.1 I/O软件的设计目标和原则

2) 统一命名

要实现上述的设备无关性，其中一项重要的工作就是如何给I/O设备命名。一般而言，是在系统中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，换言之，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。

5.4.1 I/O软件的设计目标和原则

3) 对错误的处理

一般而言，错误多数是与设备紧密相关的，因此对于错误的处理，应该尽可能在接近硬件的层面处理，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。许多情况下，错误恢复可以在低层得到解决，而高层软件不需要知道。

5.4.1 I/O软件的设计目标和原则

4) 缓冲技术

由于CPU与设备之间的速度差异，无论是块设备还是字符设备，都需要使用缓冲技术。对于不同类型的设备，其缓冲区(块)的大小是不一样的，块设备的缓冲是以数据块为单位的，而字符设备的缓冲则以字节为单位。就是同类型的设备，其缓冲区(块)的大小也是存在差异的，如不同的磁盘，其扇区的大小有可能不同。因此，I/O软件应能屏蔽这种差异，向高层软件提供统一大小的数据块或字符单元，使得高层软件能够只与逻辑块大小一致的抽象设备进行交互。

5.4.1 I/O软件的设计目标和原则

5) 设备的分配和释放

对于系统中的共享设备，如磁盘等，可以同时为多个用户服务。对于这样的设备，应该允许多个进程同时对其提出I/O请求。但对于独占设备，如键盘和打印机等，在某一段时间只能供一个用户使用，对其分配和释放的不当，将引起混乱，甚至死锁。对于独占设备和共享设备带来的许多问题，I/O软件必须能够同时进行妥善的解决。

5.4.1 I/O软件的设计目标和原则

6) I/O控制方式

针对具有不同传输速率的设备，综合系统效率和系统代价等因素，合理选择I/O控制方式，如像打印机等低速设备应采用中断驱动方式，而对磁盘等高速设备则采用DMA控制方式等，以提高系统的利用率。为方便用户，I/O软件也应屏蔽这种差异，向高层软件提供统一的操作接口。

5.4.1 I/O软件的设计目标和原则

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、存储器管理、文件管理等都存在着一定的联系，即它们都可能需要I/O软件来实现I/O操作。

为使十分复杂的I/O软件能具有清晰的结构，更好的可移植性和易适应性，目前在I/O软件中已普遍采用了层次式结构，将系统中的设备操作和管理软件分为若干个层次，每一层都利用其下层提供的服务，完成输入、输出功能中的某些子功能，并屏蔽这些功能实现的细节，向高层提供服务。

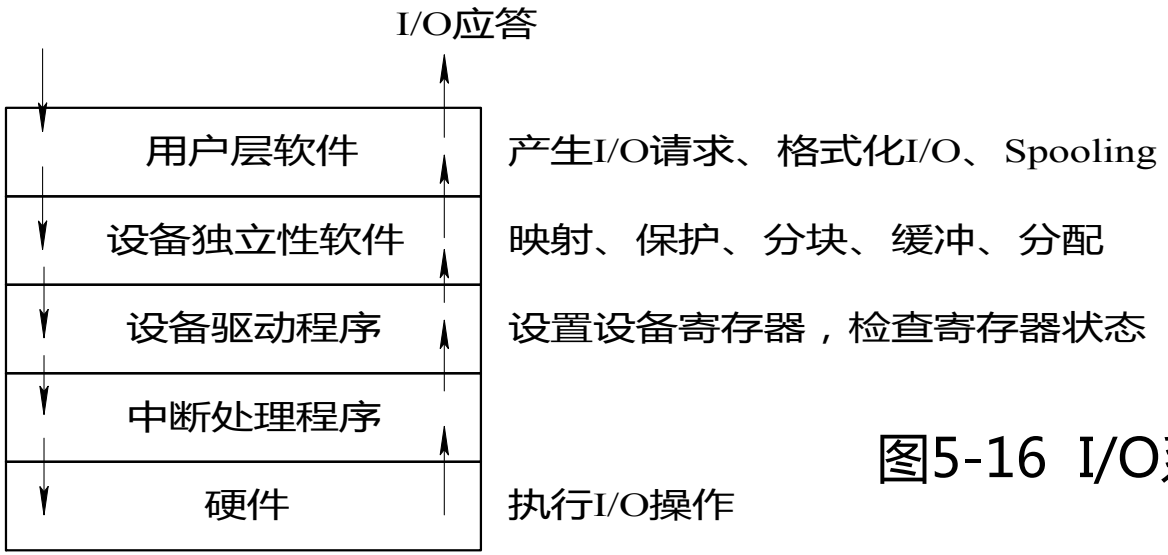


图5-16 I/O系统的层次及功能

5.4.1 I/O软件的设计目标和原则

通常把 I/O 软件组织成四个层次，如图5-16所示(图中的箭头表示I/O的控制流)。各层次及其功能如下所述：

(1) 用户层软件：实现与用户交互的接口，用户可直接调用在用户层提供的、与I/O操作有关的库函数，对设备进行操作。

(2) 设备独立性软件：负责实现与设备驱动器的统一接口、设备命名、设备的保护以及设备的分配与释放等，同时为设备管理和数据传送提供必要的存储空间。

(3) 设备驱动程序：与硬件直接相关，负责具体实现系统对设备发出的操作指令，驱动I/O设备工作的驱动程序。

(4) 中断处理程序：用于保存被中断进程的CPU环境，转入相应的中断处理程序进行处理，处理完后再恢复被中断进程的现场后返回到被中断进程。

5.4.2 用户层的I/O软件

一般而言，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（如Spooling系统）。

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，用户程序通过调用对应的库函数使用系统调用。

5.4.3 设备独立性软件

1．设备独立性的概念

为了提高OS的可适应性和可扩展性，在现代OS中都毫无例外地实现了设备独立性(Device Independence)，也称为设备无关性。其基本含义是: 应用程序独立于具体使用的物理设备。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。因此，系统须具有将逻辑设备名称转换为某物理设备名称的功能。在实现了设备独立性的功能后，可带来以下两方面的好处。

5.4.3 设备独立性软件

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检修，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。

5.4.3 设备独立性软件

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，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。

5.4.3 设备独立性软件

2．设备独立性软件

设备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，称为设备独立性软件。总的来说，设备独立性软件的主要功能可分为以下两个方面：

5.4.3 设备独立性软件

(1) 执行所有设备的公有操作，包括:

① 对独立设备的分配与回收；

② 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；

③ 对设备进行保护，禁止用户直接访问设备；

④ 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；

⑤ 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，故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；

5.4.3 设备独立性软件

⑥ 提供独立于设备的逻辑块。不同类型的设备信息交换单位是不同的，读取和传输速率也各不相同，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，如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，因此，设备独立性软件应负责隐藏这些差异，对逻辑设备使用并向高层软件提供大小统一的逻辑数据块。

5.4.3 设备独立性软件

(2) 向用户层(或文件层)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。例如，对各种设备的读操作，在应用程序中都使用read；而对各种设备的写操作，也都使用write。

5.4.3 设备独立性软件

3．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)，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。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，如图5-19(a)所示。

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。

5.4.3 设备独立性软件

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)

图5-19　逻辑设备表（整个系统一张a和每个用户一张b）

5.4.4 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，又由于它常以进程的形式存在，简称为设备驱动进程。

其主要任务是接收上层软件发来的抽象I/O要求，如read或write命令，在把它转换为具体要求后，发送给设备控制器，启动设备去执行；此外，它也将由设备控制器发来的信号传送给上层软件。

由于驱动程序与硬件密切相关，故应为每一类设备配置一种驱动程序；有时也可为非常类似的两类设备配置一个驱动程序。例如，打印机和显示器需要不同的驱动程序；但SCSI磁盘驱动程序通常可以处理不同大小和不同速度的多个SCSI磁盘，甚至还可以处理SCSI CD-ROM。

5.4.4 设备驱动程序

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。

5.4.4 设备驱动程序

(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。

5.4.4 设备驱动程序

3．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。

5.4.4 设备驱动程序

(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。

5.4.4 设备驱动程序

4．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/O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/O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。

5.4.4 设备驱动程序

以下是设备驱动程序的处理过程。

1) 将抽象要求转换为具体要求

通常在每个设备控制器中都含有若干个寄存器，分别用于暂存命令、数据和参数等。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。

5.4.4 设备驱动程序

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。

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。

5.4.4 设备驱动程序

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。

5.4.4 设备驱动程序

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。

5.4.5 中断处理程序

1．唤醒被阻塞的驱动(程序)进程

当中断处理程序开始执行时，首先去唤醒处于阻塞状态的驱动(程序)进程。如果是采用了信号量机制，则可通过执行signal操作，将处于阻塞状态的驱动(程序)进程唤醒；在采用信号机制时，将发送一信号给阻塞进程。

5.4.5 中断处理程序

2．保护被中断进程的CPU环境

通常，由硬件自动将被中断进程的CPU现场信息(包括处理机状态字PSW、程序计数器(PC)和所有的CPU寄存器，如通用寄存器、段寄存器等内容)都压入中断栈中。图5-17给出了一个简单的保护中断现场的示意图。该程序是指令在N位置时被中断的，程序计数器中的内容为N+1，所有寄存器的内容都被保留在栈中。

5.4.5 中断处理程序

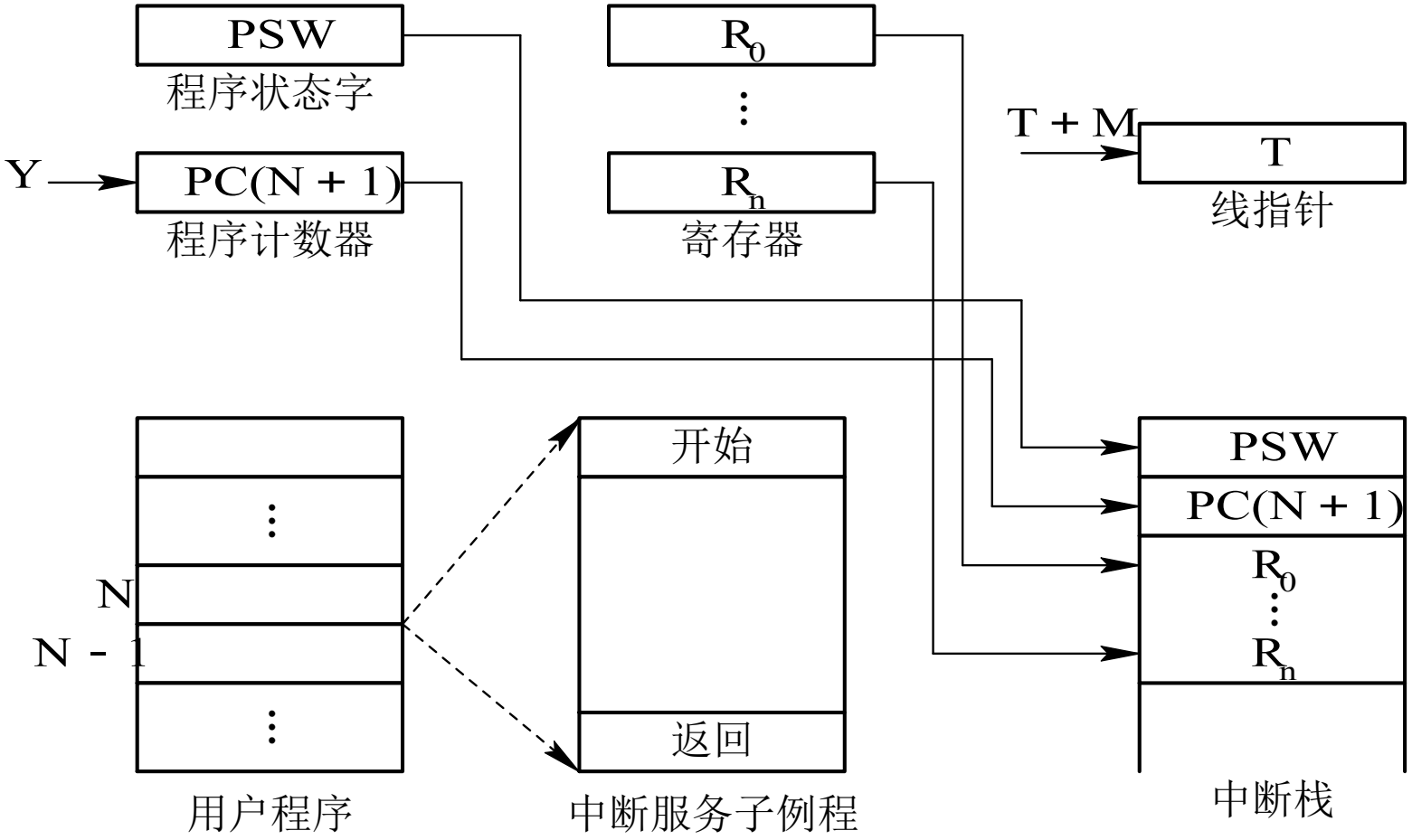


图5-17　中断现场保护示意图

5.4.5 中断处理程序

3．转入相应的设备处理程序

由处理机对各个中断源进行测试，以确定引起本次中断的I/O设备，并发送一应答信号给发出中断请求的进程，使之消除该中断请求信号，然后将相应的设备中断处理程序的入口地址装入到程序计数器中，使处理机转向中断处理程序。

5.4.5 中断处理程序

4．中断处理

对于不同的设备，有不同的中断处理程序。该程序首先从设备控制器中读出设备状态，以判别本次中断是正常完成中断，还是异常结束中断。若是前者，中断程序便进行结束处理；若还有命令，可再向控制器发送新的命令，进行新一轮的数据传送。若是异常结束中断，则根据发生异常的原因做相应的处理。

5.4.5 中断处理程序

5．恢复被中断进程的现场

当中断处理完成以后，便可将保存在中断栈中的被中断进程的现场信息取出，并装入到相应的寄存器中，其中包括该程序下一次要执行的指令的地址N+1、处理机状态字PSW，以及各通用寄存器和段寄存器的内容。这样，当处理机再执行本程序时，便从N+1处开始，最终返回到被中断的程序。

I/O操作完成后，驱动程序必须检查本次I/O操作中是否发生了错误，并向上层软件报告，最终向调用者报告本次I/O的执行情况。

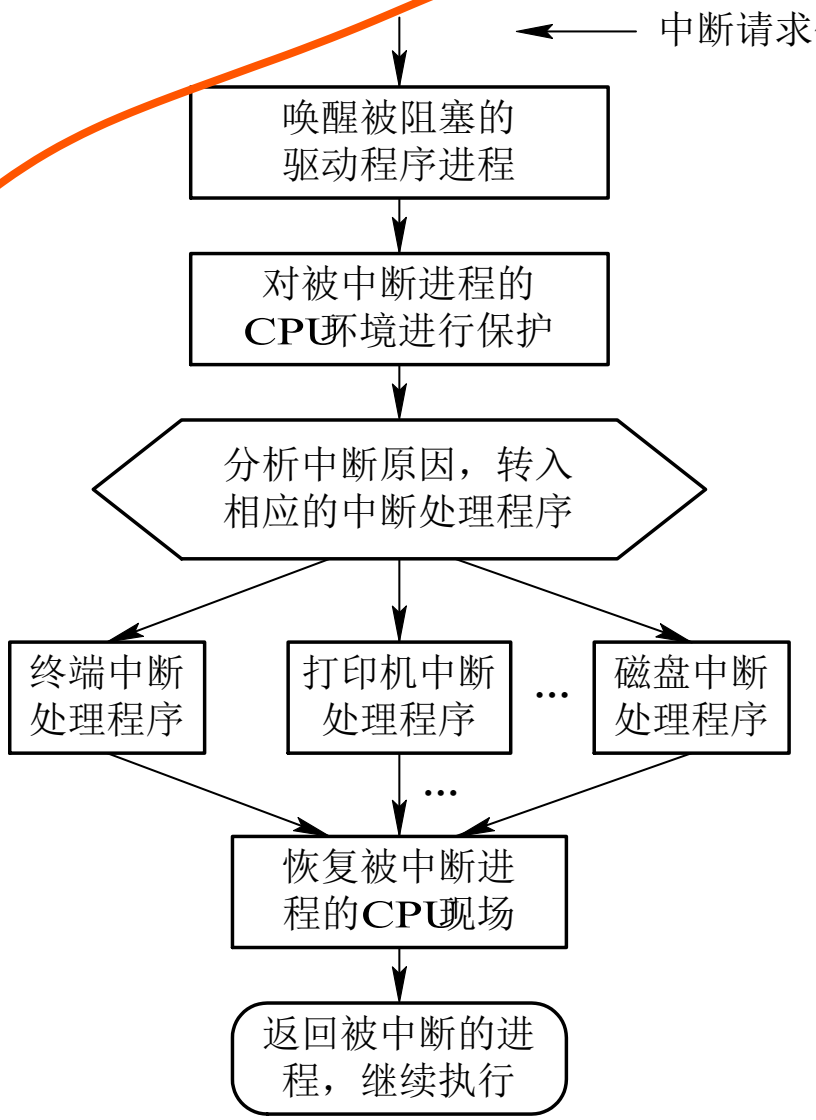
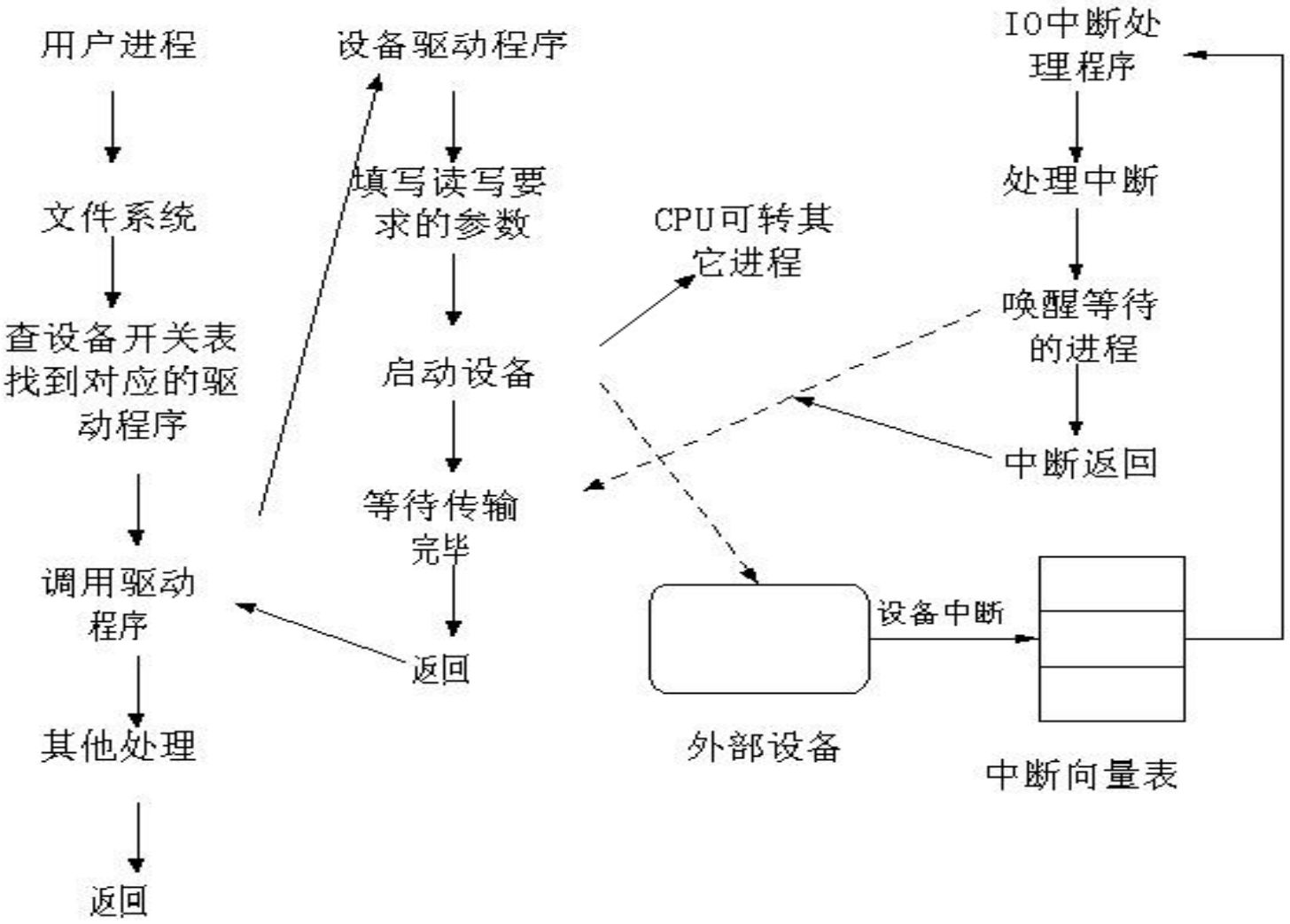

中断请求信号
唤醒被阻塞的
驱动程序进程
对被中断进程的
CPU环境进行保护
分析中断原因，转入
相应的中断处理程序
终端中断
处理程序
打印机中断
处理程序
...
磁盘中断
处理程序
...
恢复被中断进
程的CPU现场
返回被中断的进
程，继续执行

图5-18　中断处理流程

5.4.5 中断处理程序


用户进程
文件系统
查设备开关表找到对应的驱动程序
调用驱动程序
其他处理
返回
设备驱动程序
填写读写要求的参数
启动设备
等待传输完毕
返回
CPU可转其它进程
IO中断处理程序
处理中断
唤醒等待的进程
中断返回
设备中断
外部设备
中断向量表

5.5 设备分配

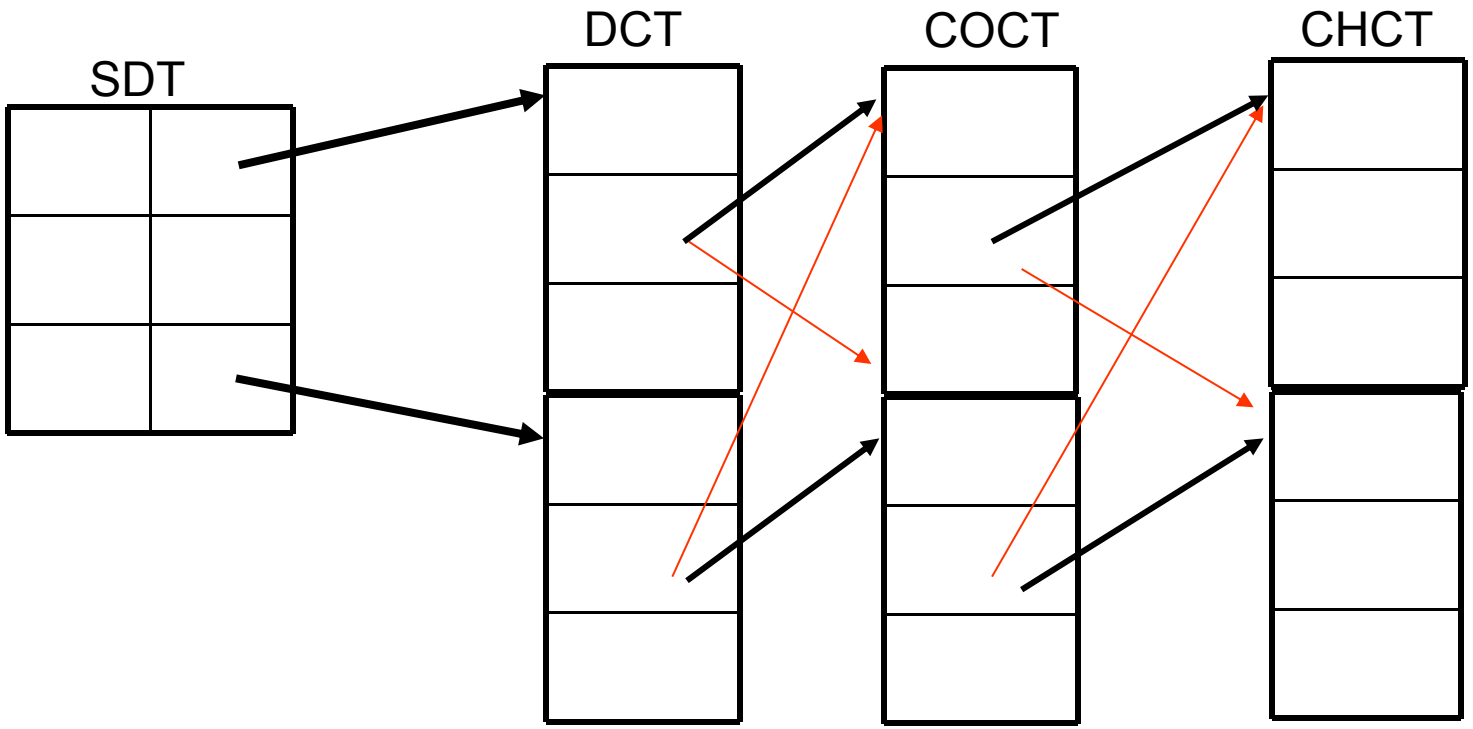
5.5.1 设备分配中的数据结构

— 系统设备表（SDT, 整个系统一张）：设备名、设备控制块指针

— 设备控制表(DCT, 每设备一张): 设备状态、分配状态、等待队列、控制器控制块指针

— 控制器控制表（COCT, 每控制器一张): 控制器状态、等待队列、通道控制块指针

— 通道控制表（CHCT，每通道一张)：通道状态、等待队列



5.5.1 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。

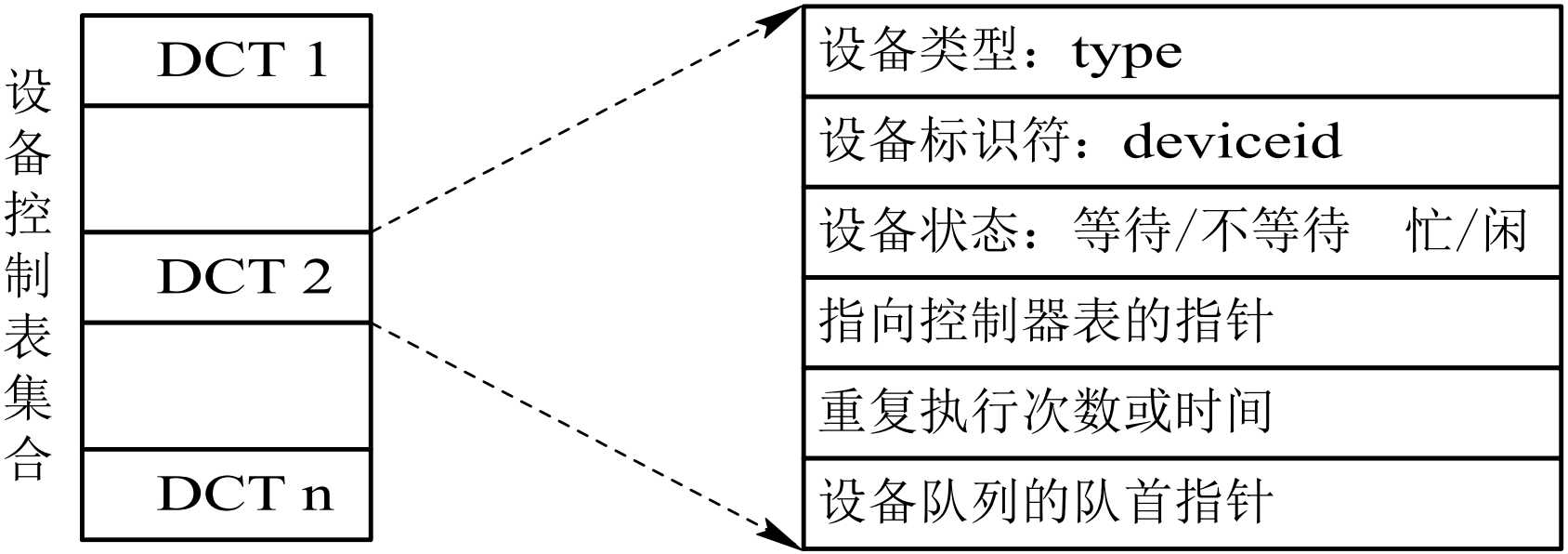


图5-20　设备控制表

5.5.1 设备分配中的数据结构

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，称该队列为设备请求队列或简称设备队列。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”。

5.5.1 设备分配中的数据结构

(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。此时，在DCT中还应设置多个控制器表指针。

(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。在重复执行时，若能恢复正常传送，则仍认为传送成功。仅当屡次失败，致使重复执行次数达到规定值而传送仍不成功时，才认为传送失败。

5.5.1 设备分配中的数据结构

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，如图5-21(a)所示。

(2) 通道控制表(CHCT)。每个通道都配有一张通道控制表，如图5-21(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，如图5-21(c)所示。

5.5.1 设备分配中的数据结构

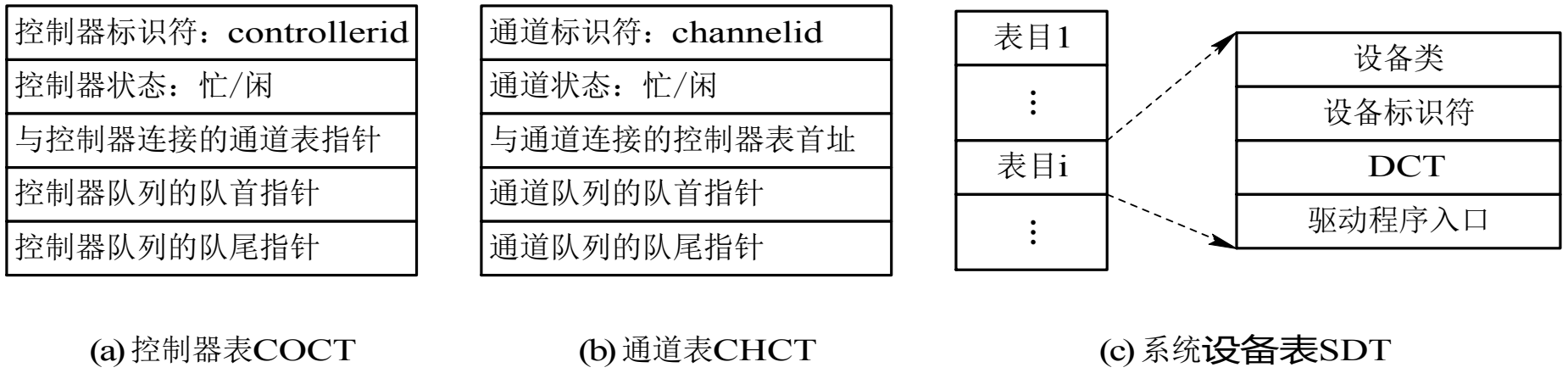


图5-21　COCT、CHCT和SDT

5.5.2 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；④ 设备独立性。

1．设备的固有属性

在分配设备时，首先应考虑与设备分配有关的设备属性。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”；第二种是共享性，指这种设备允许多个进程同时共享；第三种是可虚拟设备，指设备本身虽是独占设备，但经过某种技术处理，可以把它改造成虚拟设备。对上述的独占、 共享、可虚拟三种设备应采取不同的分配策略。

5.5.2 设备分配时应考虑的因素

(1) 独占设备。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) 共享设备。对于共享设备，可同时分配给多个进程使用，对这些进程访问该设备的先后次序要进行合理的调度。

(3) 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，因而说，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。

5.5.2 设备分配时应考虑的因素

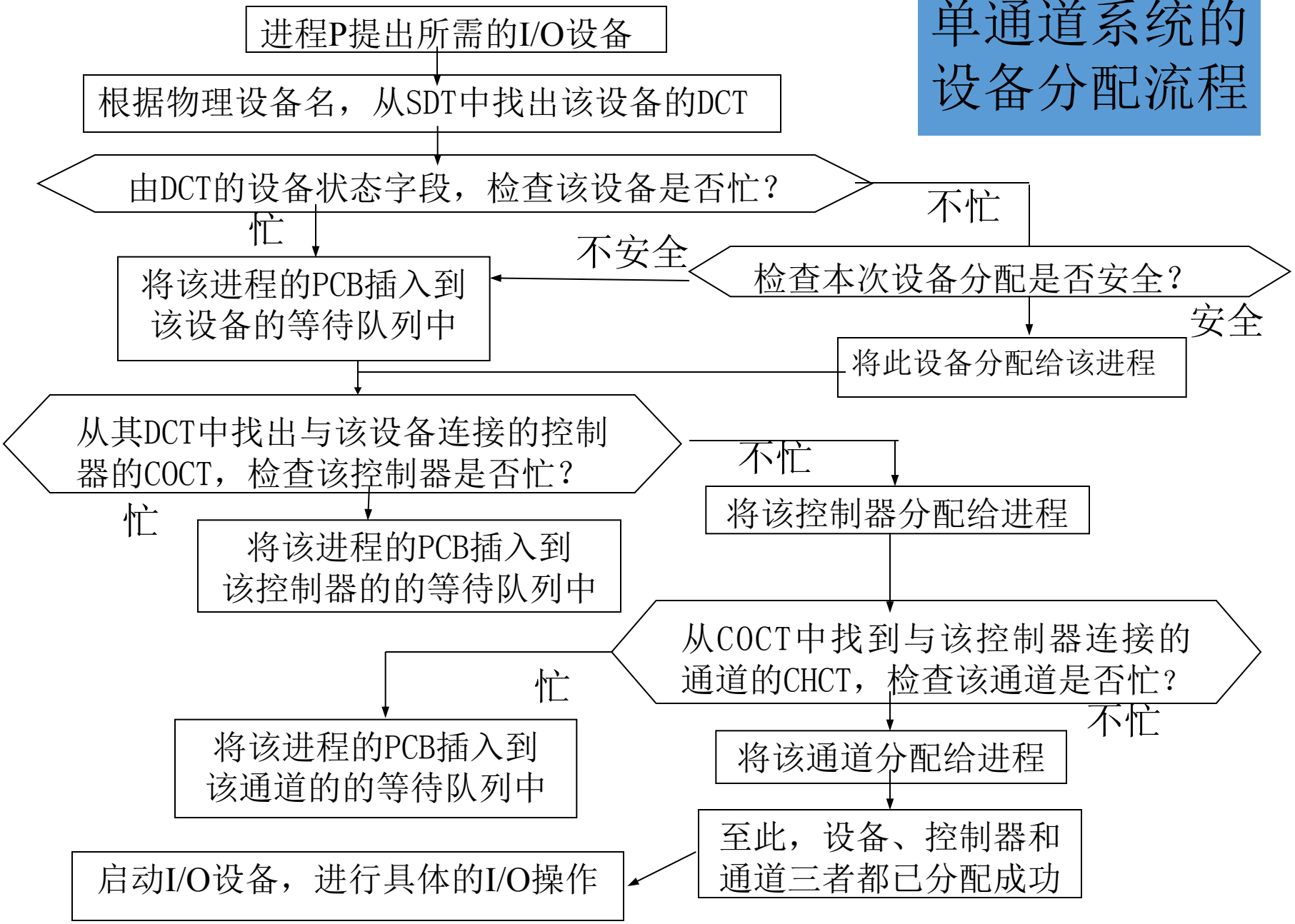
2．设备分配算法

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，该算法是根据多个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。在利用该算法形成设备队列时，将优先权高的进程排在设备队列前面，而对于优先级相同的I/O请求，则按先来先服务原则排队。

单通道系统的设备分配流程



5.5.3 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。

5.5.3 独占设备的分配程序

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。

5.5.3 独占设备的分配程序

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。

5.5.3 独占设备的分配程序

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。

5.5.3 独占设备的分配程序

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。

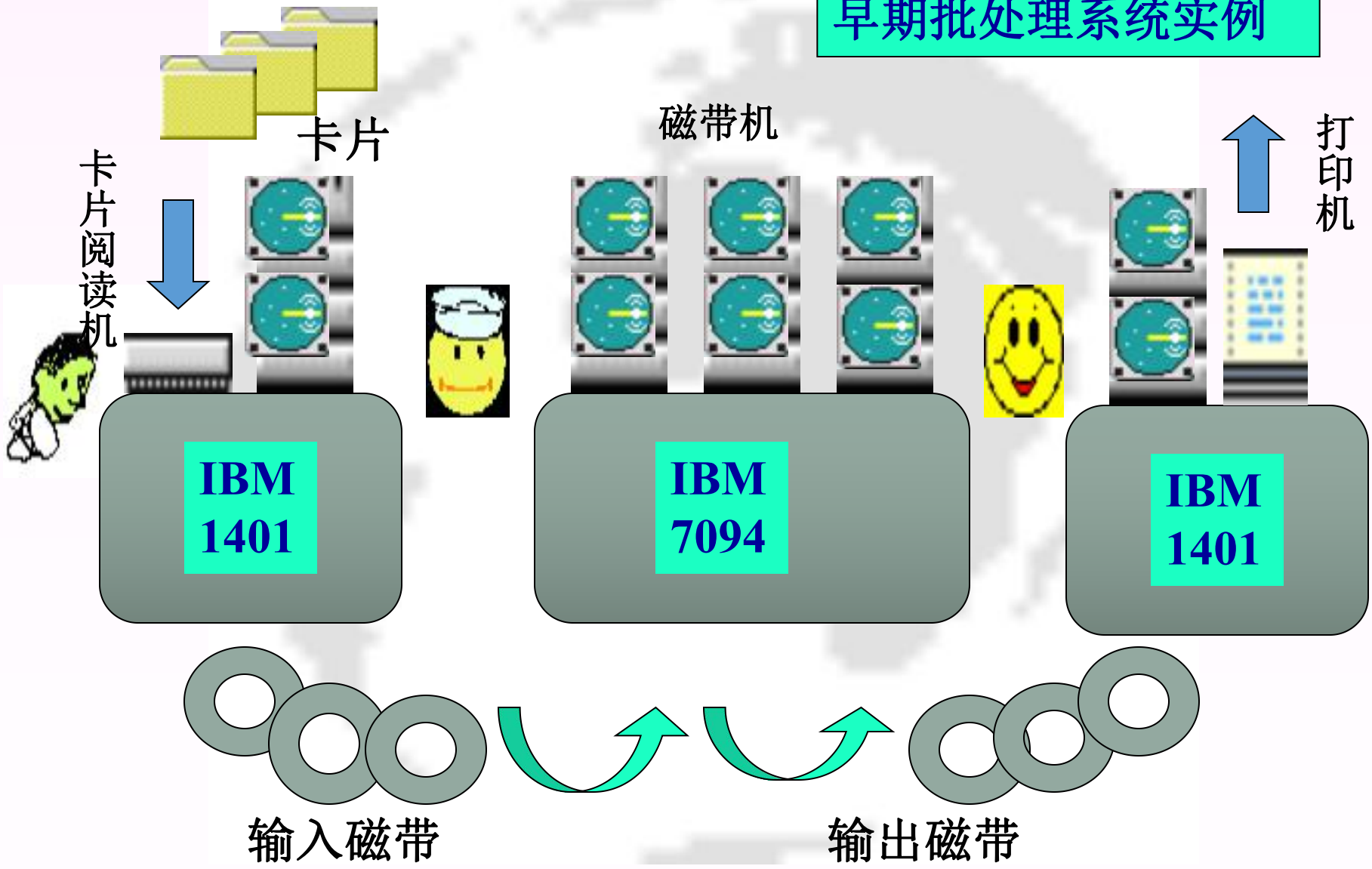
5.5.4 SPOOLing技术

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。

早期批处理系统实例



5.5.4 SPOOLing技术

2．SPOOLing系统的组成

由上所述得知，SPOOLing技术是对脱机输入、输出系统的模拟。相应地，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。

5.5.4 SPOOLing技术

(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SP_i和输出进程SP_o。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SP_i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SP_o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。

5.5.4 SPOOLing技术

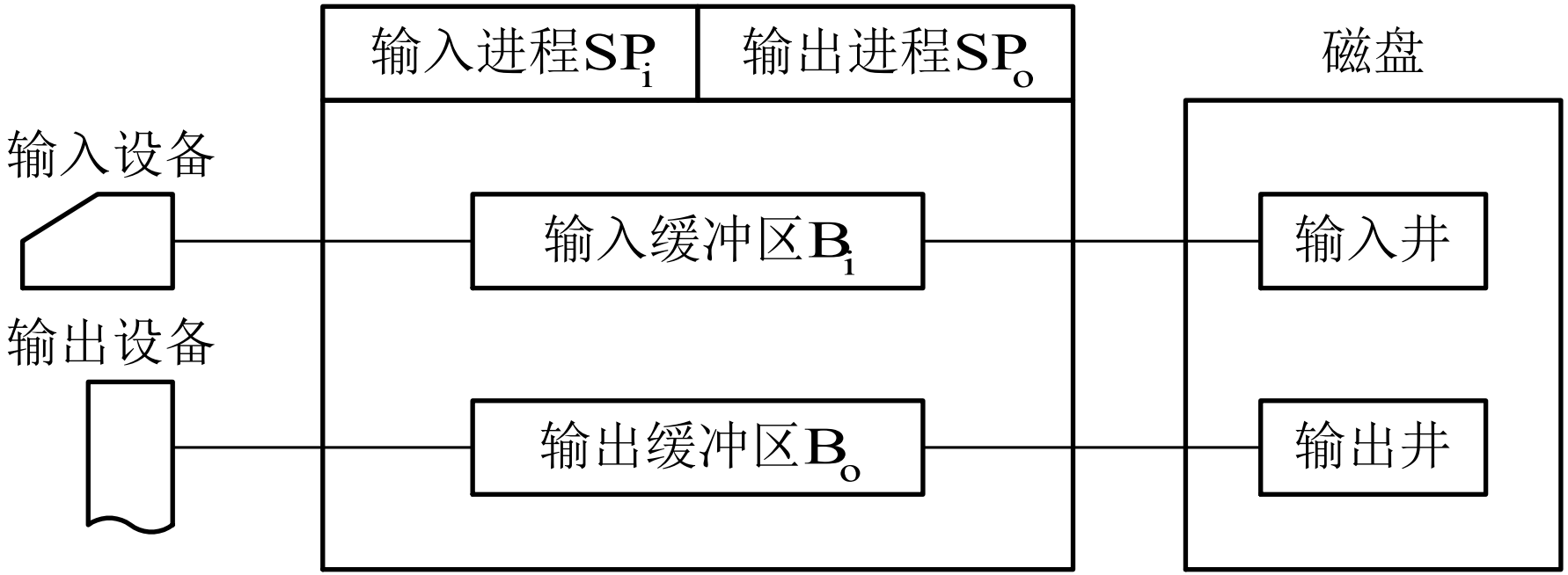


图5-22　SPOOLing系统的组成

5.5.4 SPOOLing技术

3．共享打印机

打印机是经常要用到的输出设备，属于独占设备。利用SPOOLing技术，可将之改造为一台可供多个用户共享的设备，从而提高设备的利用率，也方便了用户。

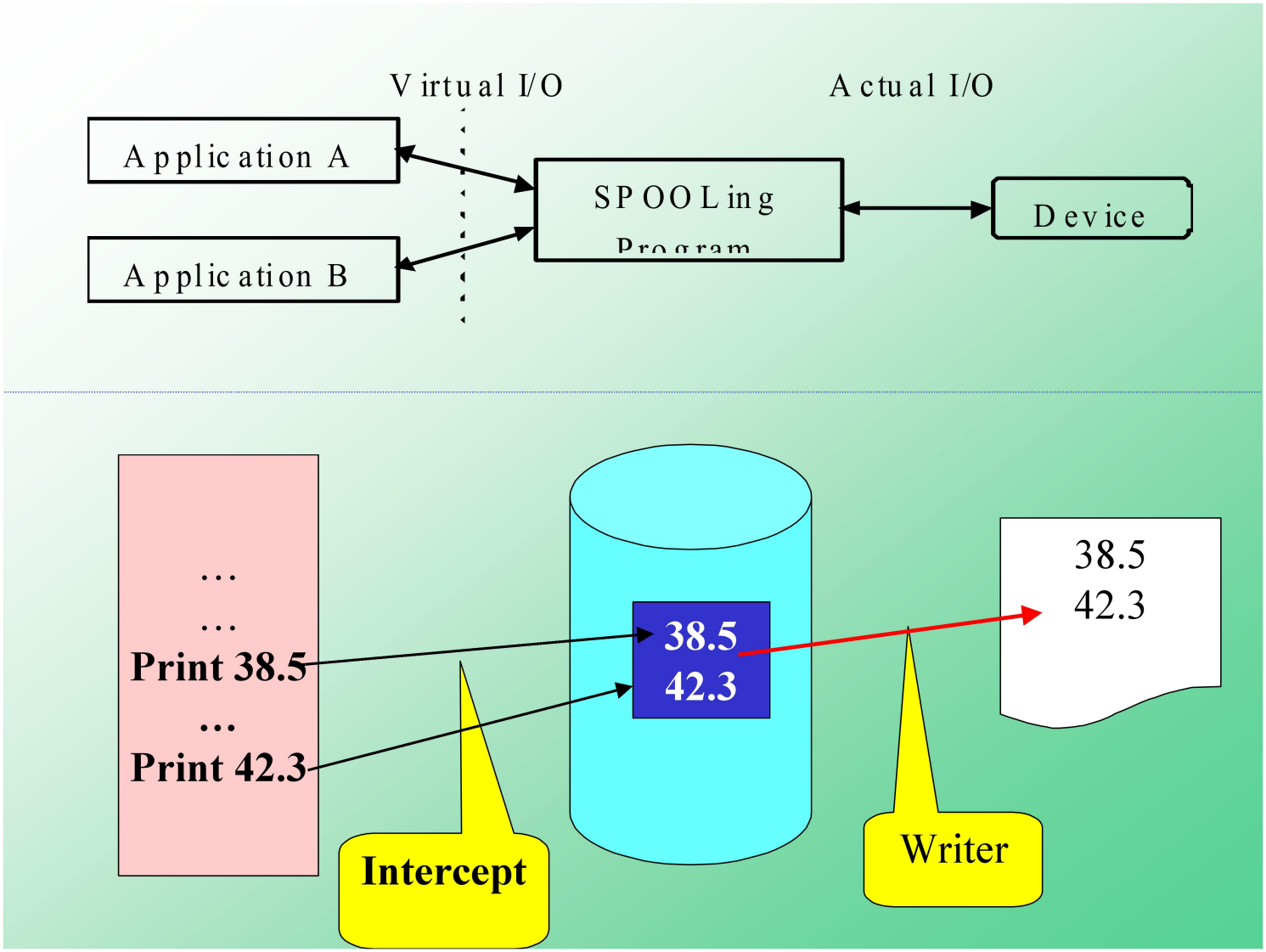
当用户进程请求打印输出时，SPOOLing系统同意为它打印输出，但并不真正立即把打印机分配给该用户进程，而只为它做两件事：① 由输出进程在输出井中为之申请一个空闲磁盘块区，并将要打印的数据送入其中；② 输出进程再为用户进程申请一张空白的用户请求打印表，并将用户的打印要求填入其中，再将该表挂到请求打印队列上。如果还有进程要求打印输出，系统仍可接受该请求，也同样为该进程做上述两件事。

5.5.4 SPOOLing技术

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。

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。

打印机利用SPOOLing技术例



5.5.4 SPOOLing技术

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。这里，对数据所进行的I/O操作，已从对低速I/O设备进行的I/O操作，演变为对输入井或输出井中数据的存取，如同脱机输入输出一样，提高了I/O速度，缓和了CPU与低速I/O设备之间速度不匹配的矛盾。

5.5.4 SPOOLing技术

(2) 将独占设备改造为共享设备。因为在SPOOLing系统中，实际上并没为任何进程分配设备，而只是在输入井或输出井中为进程分配一个存储区和建立一张I/O请求表。这样，便把独占设备改造为共享设备。

(3) 实现了虚拟设备功能。宏观上，虽然是多个进程在同时使用一台独占设备，而对于每一个进程而言，它们都会认为自己是独占了一个设备。当然，该设备只是逻辑上的设备。SPOOLing系统实现了将独占设备变换为若干台对应的逻辑设备的功能。

5.6 磁盘存储器的管理

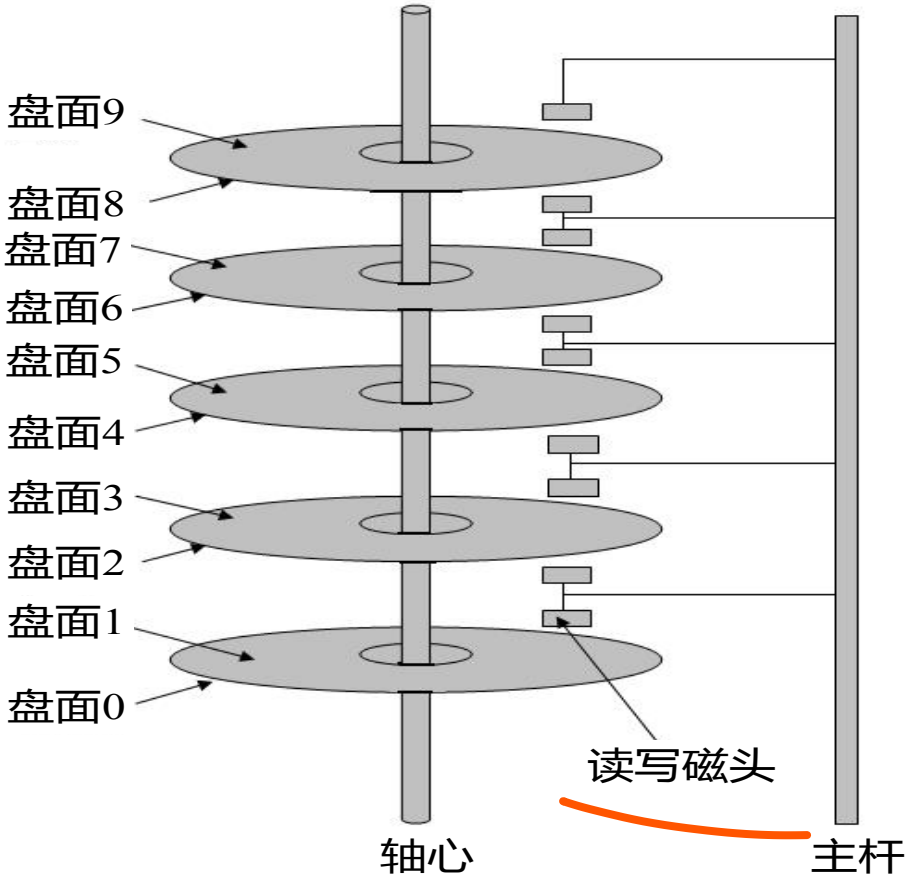
5.6.1 磁盘性能简述

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。

5.6.1 磁盘性能简述



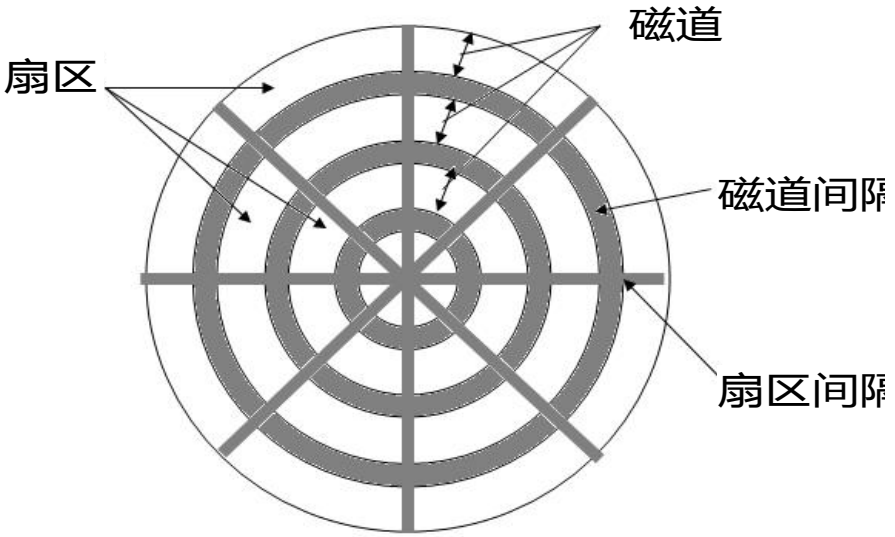


图5-23　磁盘的结构和布局

5.6.1 磁盘性能简述

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。

硬盘地址转换

物理地址转换:3维（c, h, s) ←→1维（块号b）

柱面号c、磁头号h和扇区号s

- ✓ 设：有n个柱面，m个磁头，k个扇区

  则: 总块数=n*m*k

  块号b=f(c, h ,s)=c*m*k+h*k+s

- ✓ 已知块号b,则：

  c=【b/(m*k)】

  h=【b/k】% m

  s= b % k

5.6.1 磁盘性能简述

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。

5.6.1 磁盘性能简述

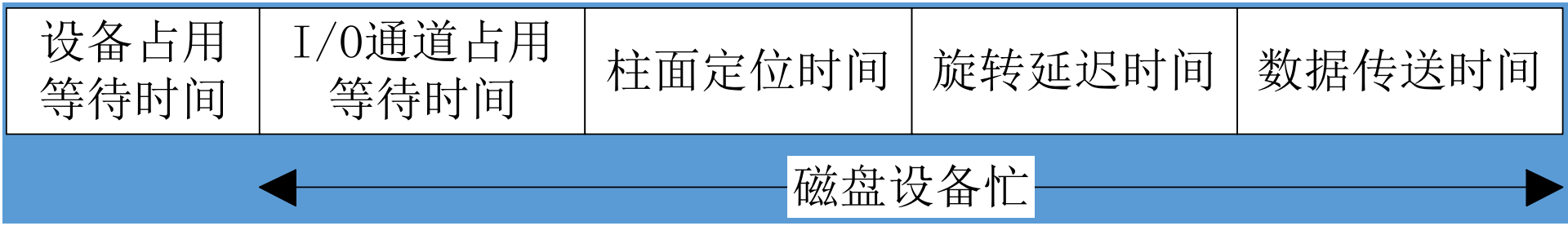
2) 移动头磁盘

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。

5.6.1 磁盘性能简述

3．磁盘访问时间

- 盘块的地址：柱面号，磁头号，扇区号
- 柱面定位时间：磁头移动到指定柱面的机械运动时间；机械运动，花费时间最多。(几十毫秒）
- 旋转延迟时间：磁盘旋转到指定扇区的机械运动时间；它与磁盘转速相关，如：软盘转速可为600rpm(每分钟转速)，硬盘可为3600rpm。（十几毫秒)
- 数据传送时间：从指定扇区读写数据的时间。(几毫秒）



柱面定位时间在访问时间中占主要部分，改善性能应从该点入手

返回

5.6.1 磁盘性能简述

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。

5.6.1 磁盘性能简述

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。

5.6.1 磁盘性能简述

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

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

5.6.1 磁盘性能简述

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部分。

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。

5.6.2 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。

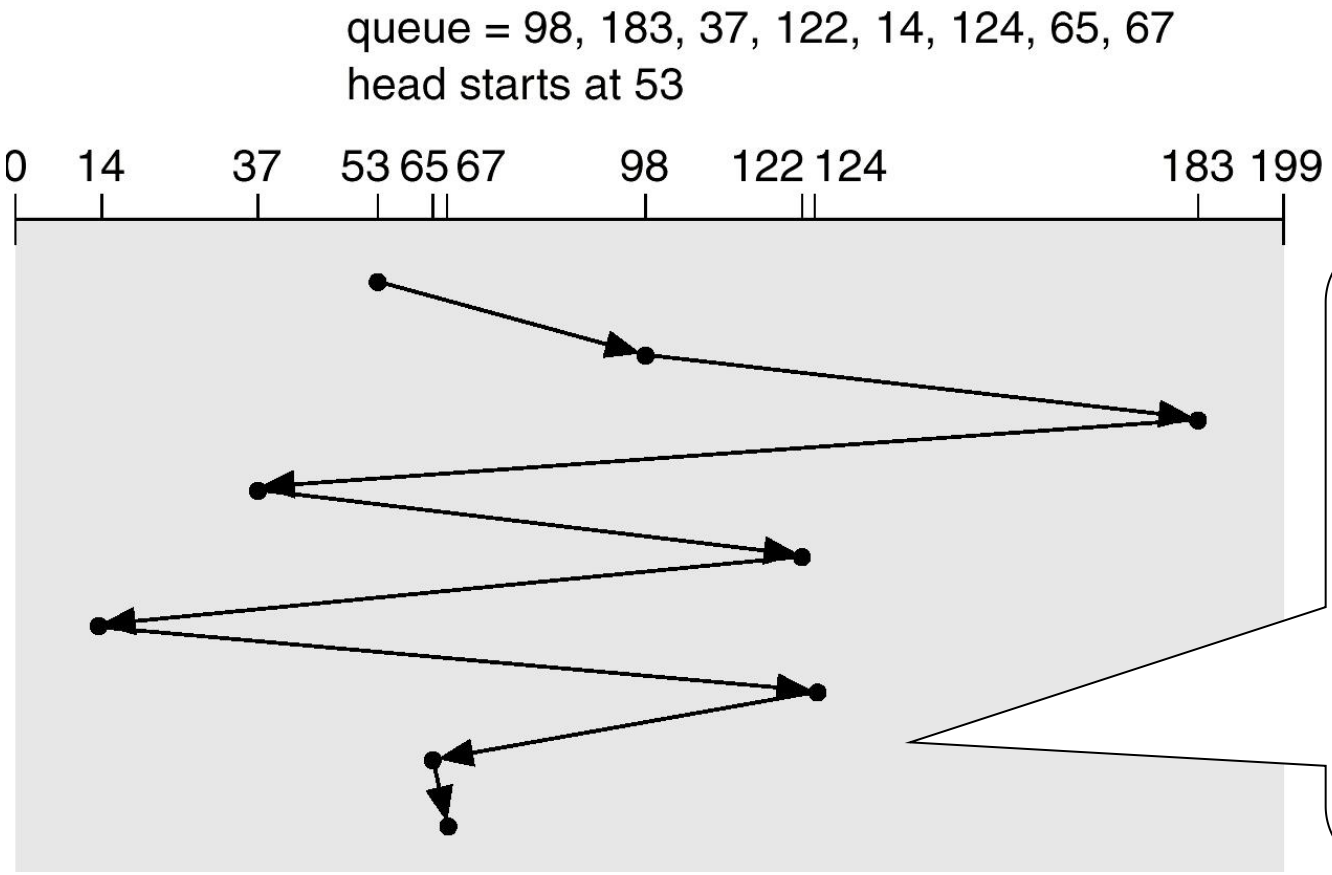
5.6.2 磁盘调度

(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

5.6.2 磁盘调度

采用FCFS例



假设磁盘访问序列：98，183，37，122，14，124，65，67，读写头起始位置：53

5.6.2 磁盘调度

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。

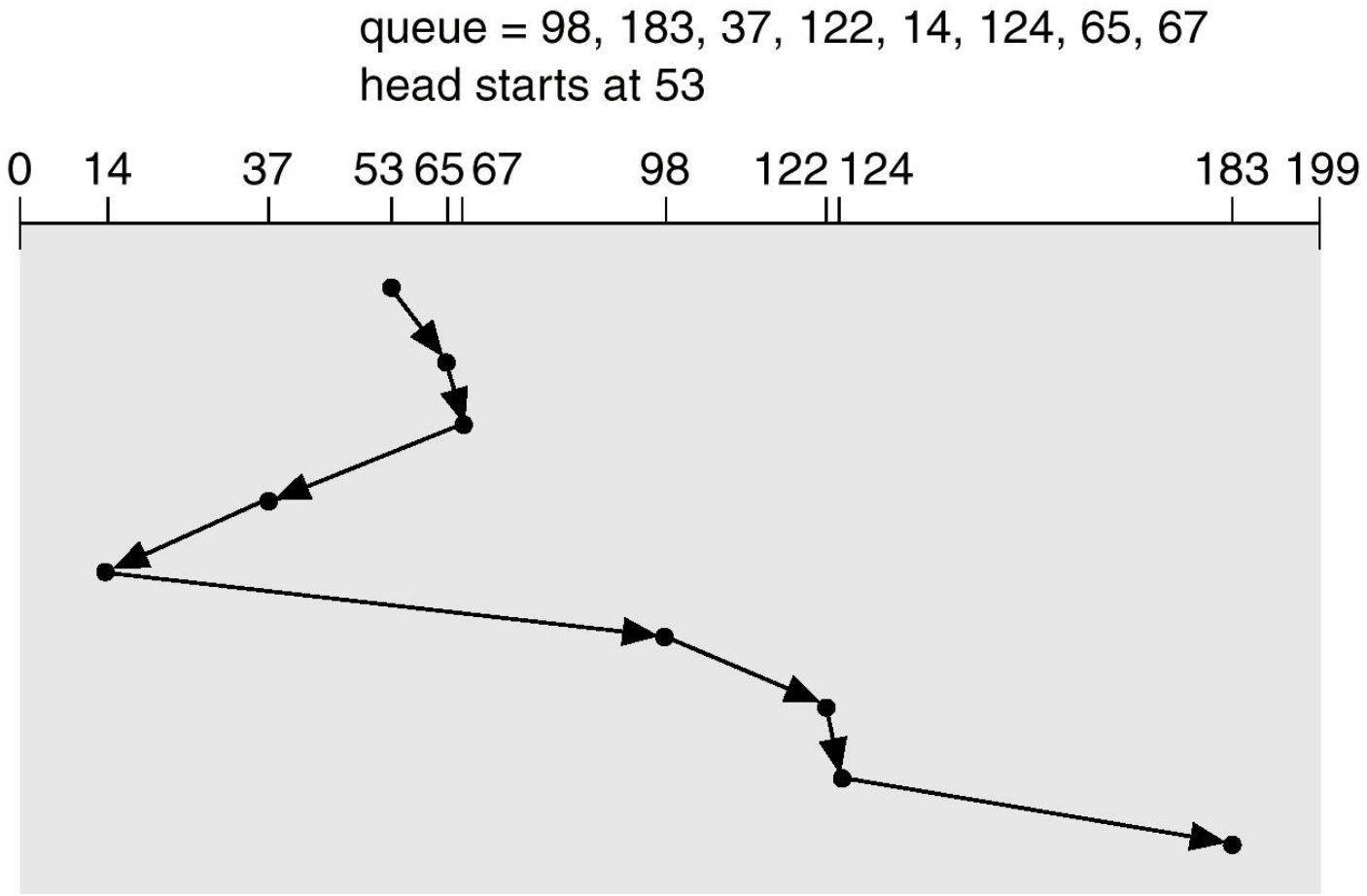
5.6.2 磁盘调度

(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

5.6.2 磁盘调度

SSTF



5.6.2 磁盘调度

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。

5.6.2 磁盘调度

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。

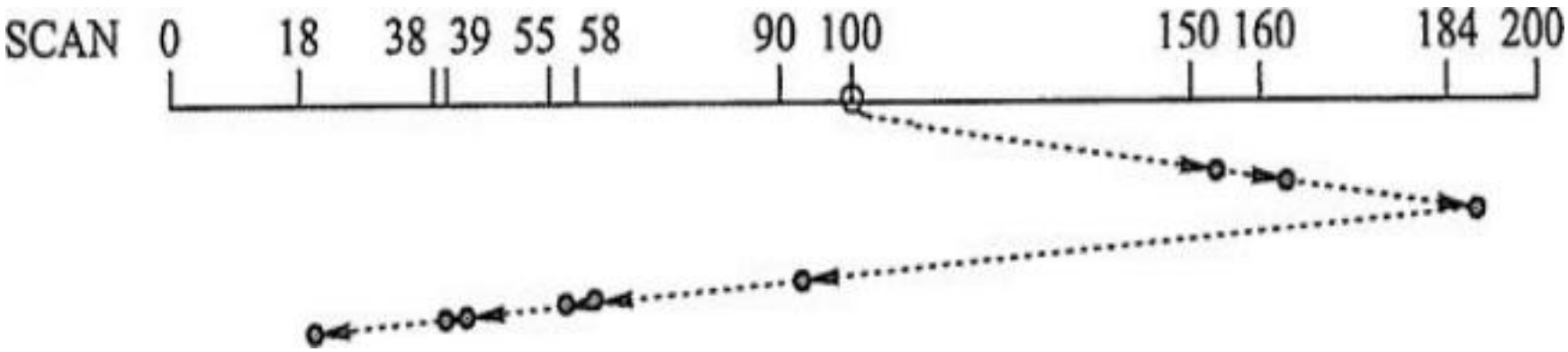
这样，磁头又逐步地从外向里移动，直至再无更里面的磁道要访问，从而避免了出现“饥饿”现象。由于在这种算法中磁头移动的规律颇似电梯的运行，因而又常称之为电梯调度算法。图 5-27示出了按SCAN算法对9个进程进行调度及磁头移动的情况。

5.6.2 磁盘调度

（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

SCAN算法



5.6.3 磁盘高速缓存

1．磁盘高速缓存的形式

这里所说的磁盘高速缓存，并非通常意义下的内存和CPU之间所增设的一个小容量高速存储器，而是指利用内存中的存储空间来暂存从磁盘中读出的一系列盘块中的信息。因此，这里的高速缓存是一组在逻辑上属于磁盘，而物理上是驻留在内存中的盘块。

5.6.3 磁盘高速缓存

2．数据交付方式

数据交付(Data Delivery)是指将磁盘高速缓存中的数据传送给请求者进程。当有一进程请求访问某个盘块中的数据时，由核心先去查看磁盘高速缓冲器，看其中是否存在进程所需访问的盘块数据的拷贝。

若有其拷贝，便直接从高速缓存中提取数据交付给请求者进程，这样，就避免了访盘操作，从而使本次访问速度提高4～6个数量级；否则，应先从磁盘中将所要访问的数据读入并交付给请求者进程，同时也将数据送高速缓存。当以后又需要访问该盘块的数据时，便可直接从高速缓存中提取。

5.6.3 磁盘高速缓存

3．置换算法

如同请求调页(段)一样，在将磁盘中的盘块数据读入高速缓存时，同样会出现因高速缓存中已装满盘块数据而需要将该数据先换出的问题。相应地，也必然存在着采用哪种置换算法的问题。较常用的置换算法仍然是最近最久未使用算法LRU等。

由于请求调页中的联想存储器与高速缓存(磁盘I/O中)的工作情况不同，因此，现在不少系统在设计其高速缓存的置换算法时，除了考虑到最近最久未使用这一原则外，还考虑了以下几点：

5.6.3 磁盘高速缓存

1) 访问频率

通常，每执行一条指令时，便可能访问一次联想存储器，亦即联想存储器的访问频率，基本上与指令执行的频率相当。而对高速缓存的访问频率，则与磁盘I/O的频率相当。因此，对联想存储器的访问频率远远高于对高速缓存的访问频率。

5.6.3 磁盘高速缓存

2) 可预见性

在高速缓存中的各盘块数据，有哪些数据可能在较长时间内不会再被访问，又有哪些数据可能很快就再被访问，会有相当一部分是可预知的。例如，对目录块等，在它被访问后，可能会很久都不再被访问。又如，正在写入数据的未满盘块，可能会很快又被访问。

5.6.3 磁盘高速缓存

3) 数据的一致性

由于高速缓存是做在内存中的，而内存一般又是一种易失性的存储器，一旦系统发生故障，存放在高速缓存中的数据将会丢失；而其中有些盘块(如索引结点盘块)中的数据已被修改，但尚未拷回磁盘，因此，当系统发生故障后，可能会造成数据的不一致性。

5.6.3 磁盘高速缓存

基于上述考虑，有的系统便将高速缓存中的所有盘块数据拉成一条LRU链。对于那些会严重影响到数据一致性的盘块数据和很久都可能不再使用的盘块数据，都放在LRU链的头部，使它们能被优先写回磁盘，以减少发生数据不一致性的概率，或者可以尽早地腾出高速缓存的空间。

对于那些可能在不久之后便要再使用的盘块数据，应挂在LRU链的尾部，以便在不久以后需要时，只要该数据块尚未从链中移至链首而被写回磁盘，便可直接到高速缓存中(即LRU链中)去找到它们。

5.6.3 磁盘高速缓存

4．周期性地写回磁盘

还有一种情况值得注意: 那就是根据LRU算法，那些经常要被访问的盘块数据，可能会一直保留在高速缓存中，长期不会被写回磁盘。(注意，LRU链意味着链中任一元素在被访问之后，总是又被挂到链尾而不被写回磁盘；只是一直未被访问的元素，才有可能移到链首，而被写回磁盘。)

例如，一位学者一上班便开始撰写论文，并边写边修改，他正在写作的论文就一直保存在高速缓存的LRU链中。如果在快下班时，系统突然发生故障，这样，存放在高速缓存中的已写论文将随之消失，致使他枉费了一天的劳动。

5.6.4 提高磁盘I/O速度的其它方法

1．提前读(Read-ahead)

用户(进程)对文件进行访问时，经常采用顺序访问方式，即顺序地访问文件各盘块的数据。在这种情况下，在读当前块时可以预知下一次要读的盘块。因此，可以采取预先读方式，即在读当前块的同时，还要求将下一个盘块(提前读的块)中的数据也读入缓冲区。这样，当下一次要读该盘块中的数据时，由于该数据已被提前读入缓冲区，因而此时便可直接从缓冲区中取得下一盘块的数据，而不需再去启动磁盘I/O，从而大大减少了读数据的时间。

“提前读”功能已被广泛采用，如在UNIX系统、OS/2，以及在3 Plus和Netware等的网络OS中，都已采用该功能。

5.6.4 提高磁盘I/O速度的其它方法

2．延迟写

延迟写是指在缓冲区A中的数据，本应立即写回磁盘，但考虑到该缓冲区中的数据在不久之后可能还会再被本进程或其它进程访问(共享资源)，因而并不立即将该缓冲区A中的数据写入磁盘，而是将它挂在空闲缓冲区队列的末尾。随着空闲缓冲区的使用，缓冲区也缓缓往前移动，直至移到空闲缓冲队列之首。当再有进程申请到该缓冲区时，才将该缓冲区中的数据写入磁盘，而把该缓冲区作为空闲缓冲区分配出去。当该缓冲区A仍在队列中时，任何访问该数据的进程，都可直接读出其中的数据而不必去访问磁盘。

这样，又可进一步减小等效的磁盘I/O时间。同样，“延迟写”功能已在UNIX系统、OS/2等OS中被广泛采用。

5.6.4 提高磁盘I/O速度的其它方法

3．优化物理块的分布

另一种提高磁盘I/O速度的重要措施是优化文件物理块的分布，使磁头的移动距离最小。如果将一个文件的多个物理块安排得过于分散，会增加磁头的移动距离。

例如，将文件的第一个盘块安排在最里的一条磁道上，而把第二个盘块安排在最外的一条磁道上，这样，在读完第一个盘块后转去读第二个盘块时，磁头要从最里的磁道移到最外的磁道上。如果我们将这两个数据块安排在属于同一条磁道的两个盘块上，显然会由于消除了磁头在磁道间的移动，而大大提高对这两个盘块的访问速度。

5.6.4 提高磁盘I/O速度的其它方法

对文件盘块位置的优化，应在为文件分配盘块时进行。如果系统中的空白存储空间是采用位示图方式表示的，则要将同属于一个文件的盘块安排在同一条磁道上或相邻的磁道上是十分容易的事。这时，只要从位示图中找到一片相邻接的多个空闲盘块即可。

但当系统采用线性表(链)法来组织空闲存储空间时，要为一文件分配多个相邻接的盘块，就要困难一些。此时，我们可以将在同一条磁道上的若干个盘块组成一簇，例如，一簇包括4个盘块，在分配存储空间时，以簇为单位进行分配。这样就可以保证在访问这几个盘块时，不必移动磁头或者仅移动一条磁道的距离，从而减少了磁头的平均移动距离。

本章小结

- 设备分类
- 设备的工作原理（组成部件，工作流程）
- 设备管理的功能（分配、启动、中断处理、统一接口、缓冲区管理、虚拟设备）
- 设备数据I/O控制的四种方式（占用CPU时间、适用设备、造价）
- 设备数据结构、分配算法
- 缓冲区技术（作用、分类）
- SPOOLing技术（软件、硬件、实现流程）
- 磁盘调度