第六章 输入输出系统

6.1 概述
6.2 I/O控制方式
6.3 缓冲技术
6.4 设 备 分 配
6.5 磁盘概述
6.6 磁盘调度算法

6.1 概述

6.1.1 I/O系统的功能和层次结构

6.1.2 I/O设备的分类

6.1.1 I/O系统功能及层次结构

1、I/O系统功能

（1）方便用户使用

隐藏物理设备细节：向上层提供抽象功能

与设备无关性：逻辑设备名

（2）提高利用率

提高CPU和I/O设备利用率

对I/O设备进行控制：轮询、中断、DMA、通道

（3）共享

确保正确共享：独占设备、共享设备

错误处理：临时错误、持久错误

6.1.1 I/O系统功能及层次结构

2、I/O系统层次结构

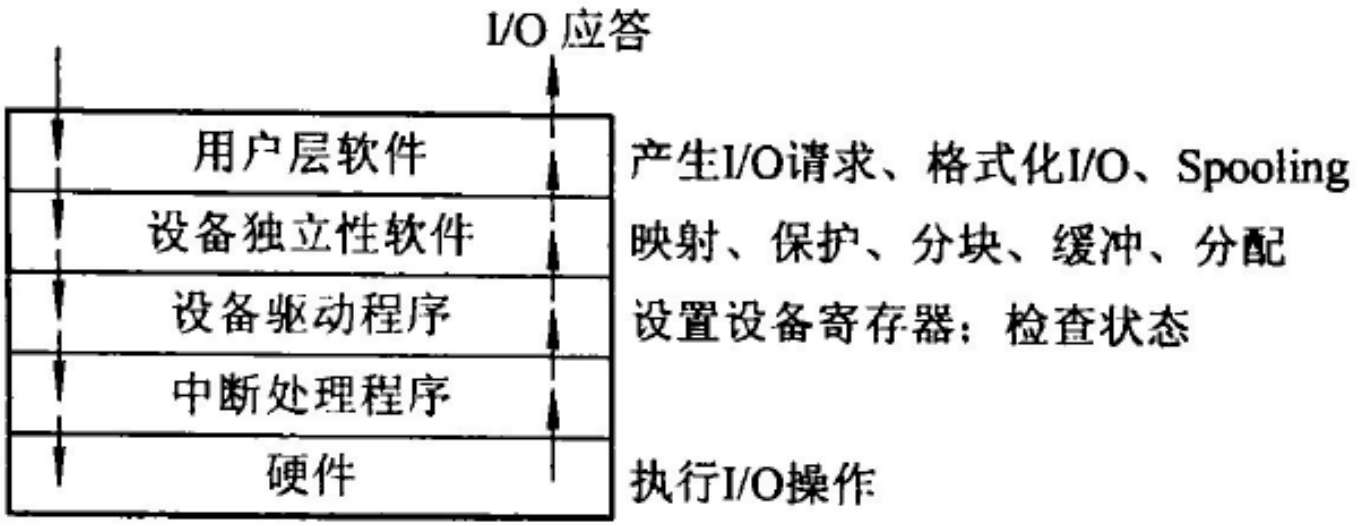


图 6-1　I/O 系统的层次结构

I/O设备和设备控制器

I/O设备：机械部分（I/O设备）、电子部分（设备控制器）

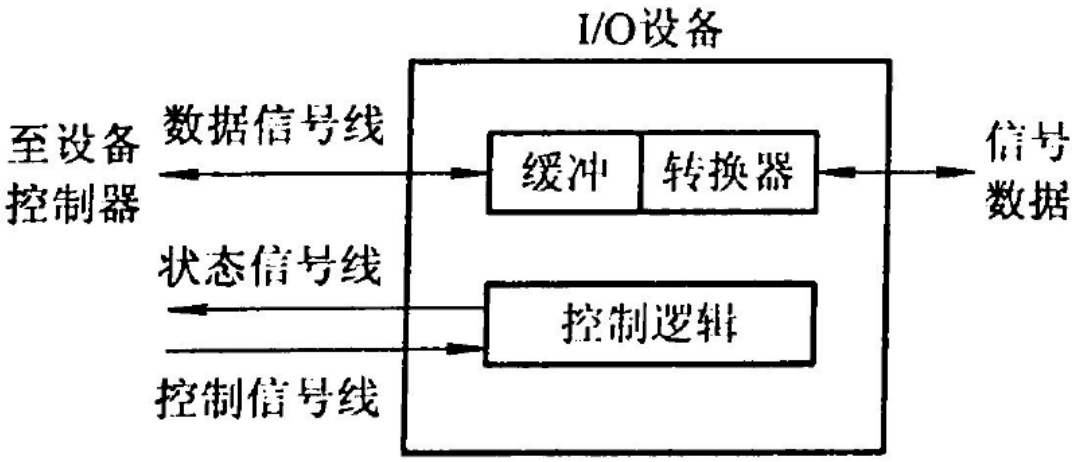


图 6-3　设备与控制器间的接口

I/O设备和设备控制器

设备控制器功能：接收和识别命令、数据交换、标识和报告设备状态、地址识别、数据缓冲区、差错控制

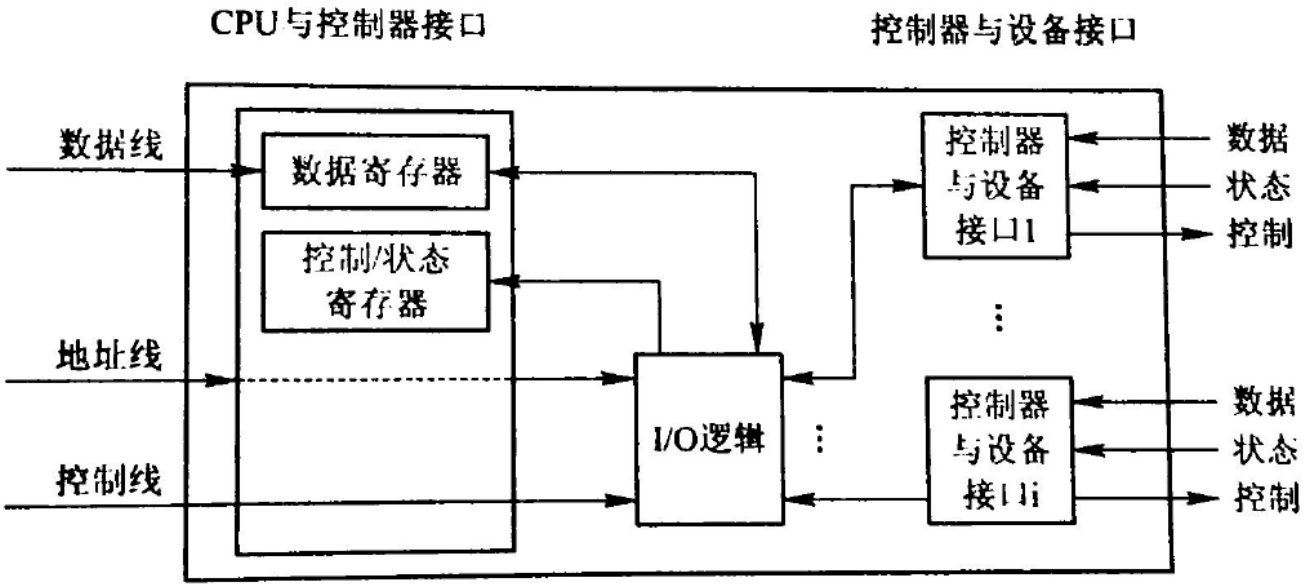


图 6-4　设备控制器的组成

I/O设备和设备控制器

I/O通道：减少CPU的干预

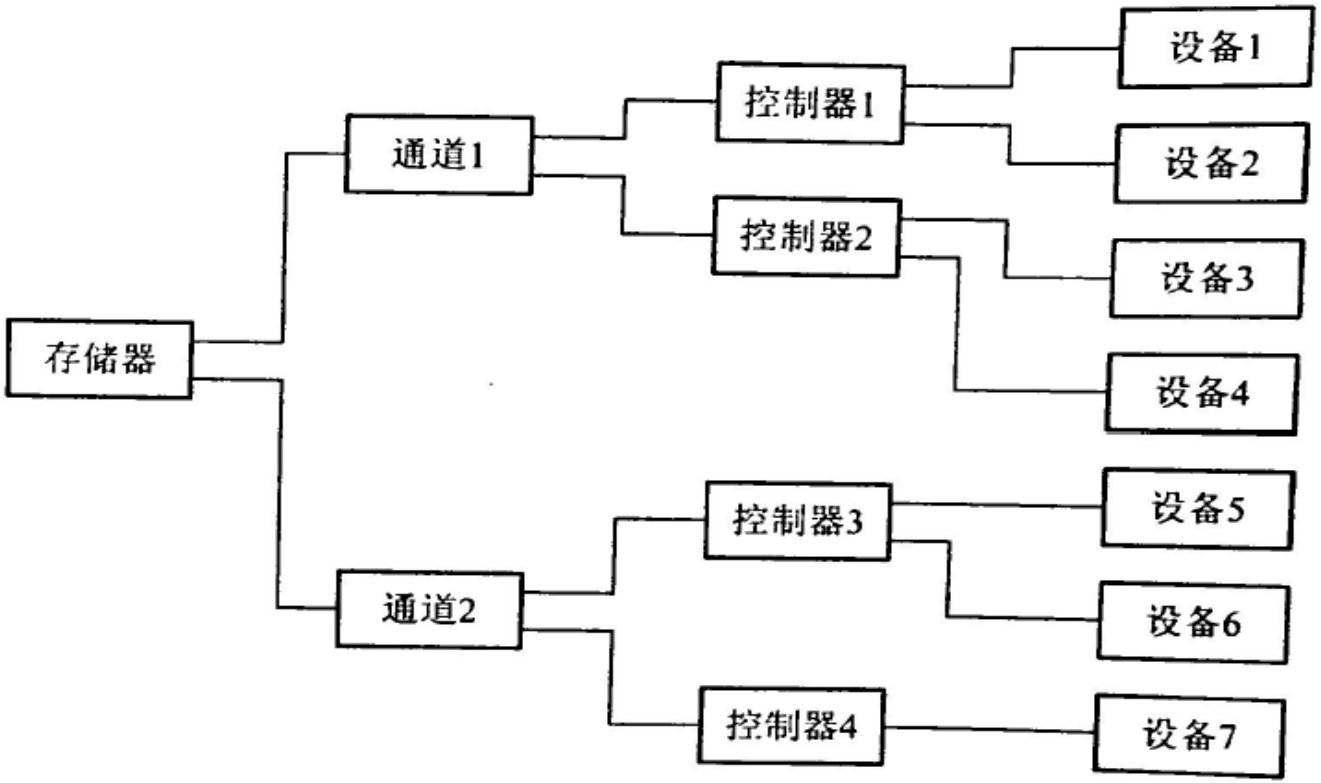


图 6-7　单通路 I/O 系统

I/O设备和设备控制器

I/O通道：减少CPU的干预

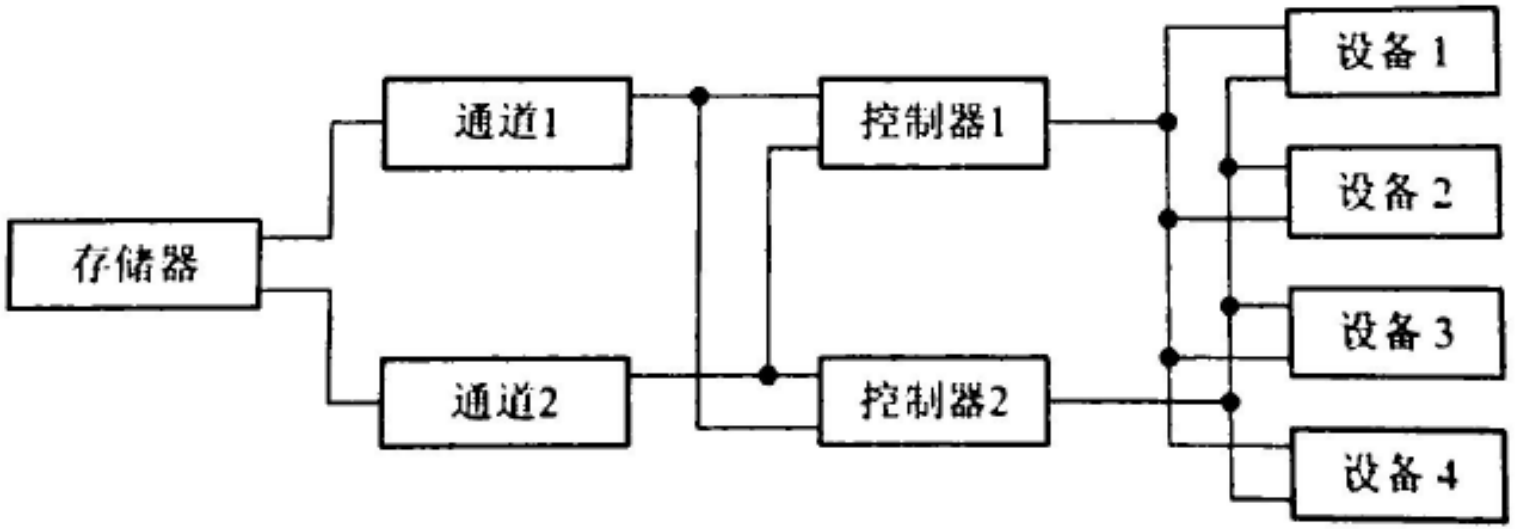


图 6-8　多通路 I/O 系统

中断机构和中断处理程序

中断： 中断（外中断）、陷入（内中断）

I/O设备完成操作，通过设备控制器向CPU发中断请求

中断处理程序步骤：

（1）检测中断请求（2）保护现场

（3）中断处理：判断I/O状态，若成功则

CPU←→I/O 寄存器

（4）恢复现场

设备驱动程序

设备驱动程序：接收抽象I/O要求，转换为具体命令，发送给设备控制器

设备驱动程序处理过程：

（1）将抽象要求转换为具体要求

（2）对服务请求进行校验（合法性检查）

（3）检查设备状态

（4）传递参数

（5）启动I/O设备

设备独立性软件

与设备无关的软件：

（1）设备驱动程序的统一接口

（2）缓冲管理

（3）差错控制：暂时性错误、持久性错误

（4）设备的分配与回收

（5）独立于设备的逻辑数据块（大小）

用户层的I/O软件

实现与用户交互的接口

用户可直接调用此层提供的函数（系统调用）

6.1.1 I/O系统功能及层次结构

2、I/O系统层次结构

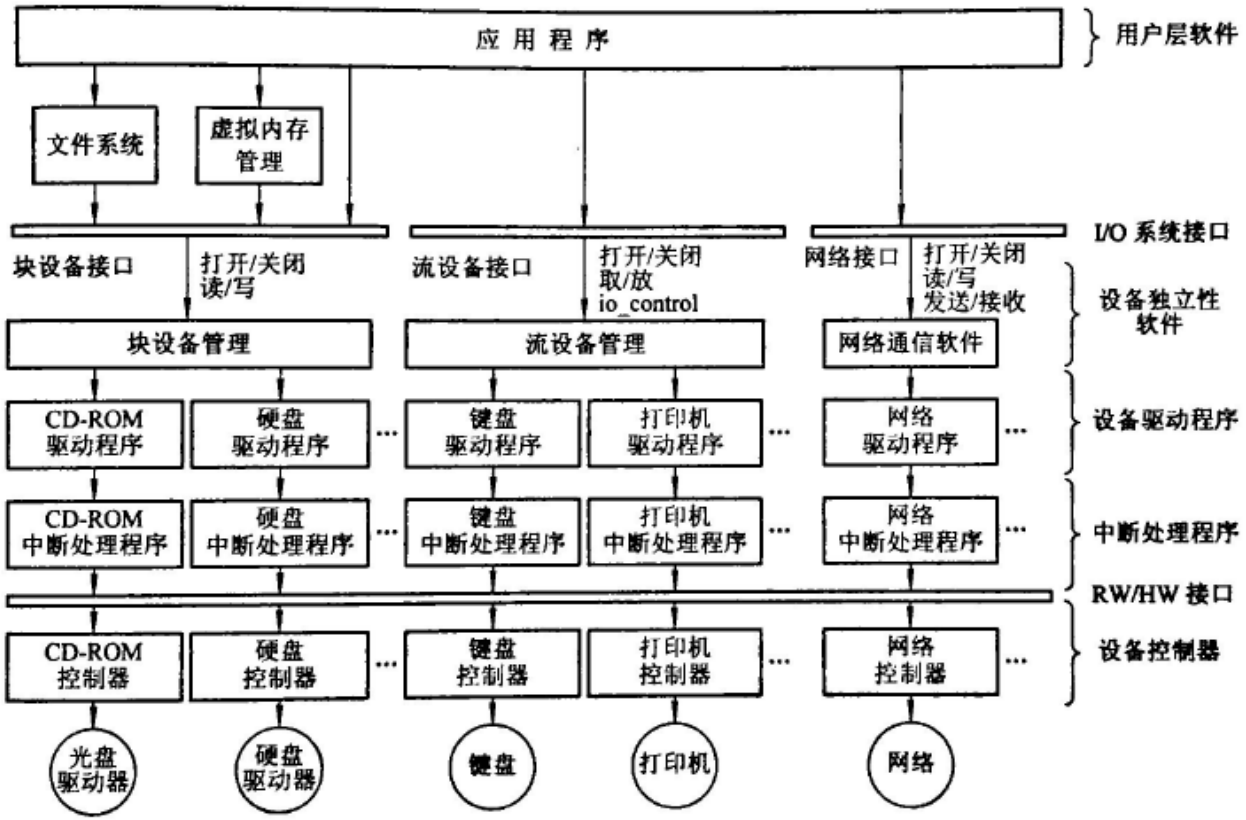


图 6-2 I/O 系统中各种模块之间的层次视图

6.1.2 设备的分类

1、按传输速率分

低速设备：每秒几个到数百字节。如Modem

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到千兆。如磁盘、磁带

2、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、modem等。特征：速率较低、中断驱动。

块设备：I/O传输的单位是块，如磁盘、磁带。特征：速率高、可随机访问任一块、DMA方式、通道方式驱动。

3. 按资源管理方式分类

独占型设备：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。

共享型设备：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：通过虚拟技术将一台独占设备变换为若干台逻辑设备，供若干进程同时使用。把这种经过虚拟技术处理后的设备称为虚拟设备。

6.2 I/O控制方式

6.2.1 循环测试I/O方式

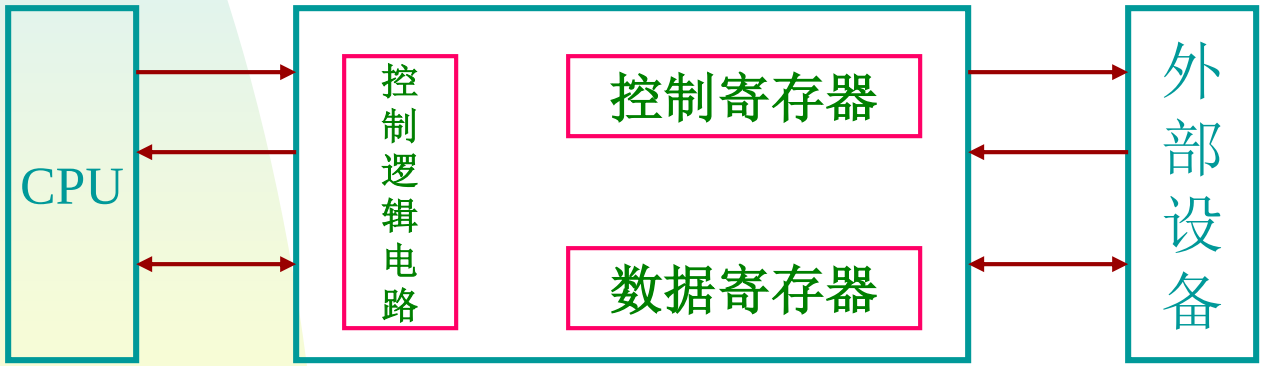
6.2.2 I/O中断方式

6.2.3 DMA方式

6.2.4 通道方式

6.2.1 循环测试I/O方式

早期，I/O控制器是CPU同外设之间的接口。它有两个寄存器：数据缓冲寄存器、控制寄存器。控制寄存器有几个重要的信息位：启动位、完成位、忙位。



I/O控制器

工作过程

以输入为例

1、 把启动位置1

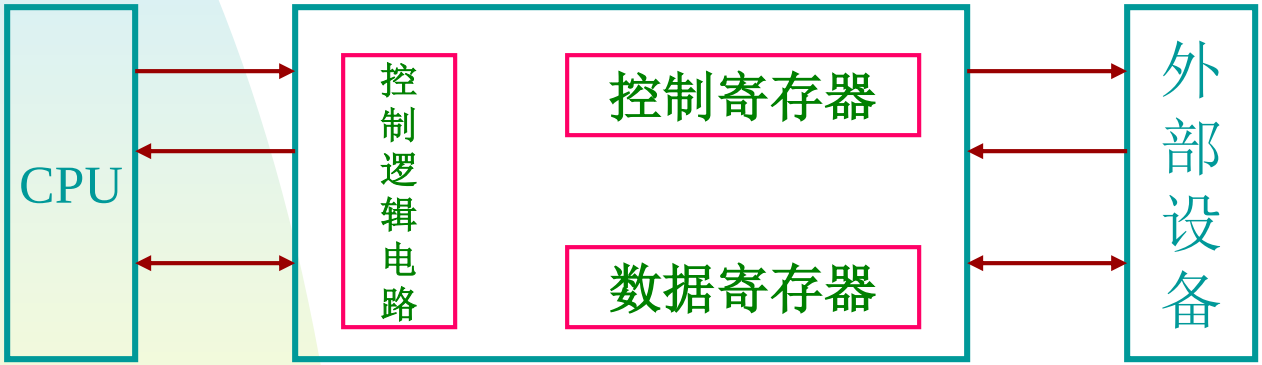
2、 反复测试完成位，为0转2，为1转3

3、 把数据从数据缓冲寄存器中读走。

浪费大量CPU时间

6.2.2 I/O中断方式

I/O控制器能发中断。



I/O控制器

工作过程

1、把启动位置1，本进程（A）变为等待状态，转进程调度，调度另一进程B。

2、输入完成时，控制器发出中断，中断B，通过中断进入中断处理程序。

3、在中断处理程序中把数据缓冲寄存器中的数取走，放入内存特定位置M，唤醒等待进程A，中断返回到B的断点继续执行。

4、在以后的某个时刻OS调度要求输入的进程A。A从M取数处理。

分析

同前相比，CPU利用率大大提高。

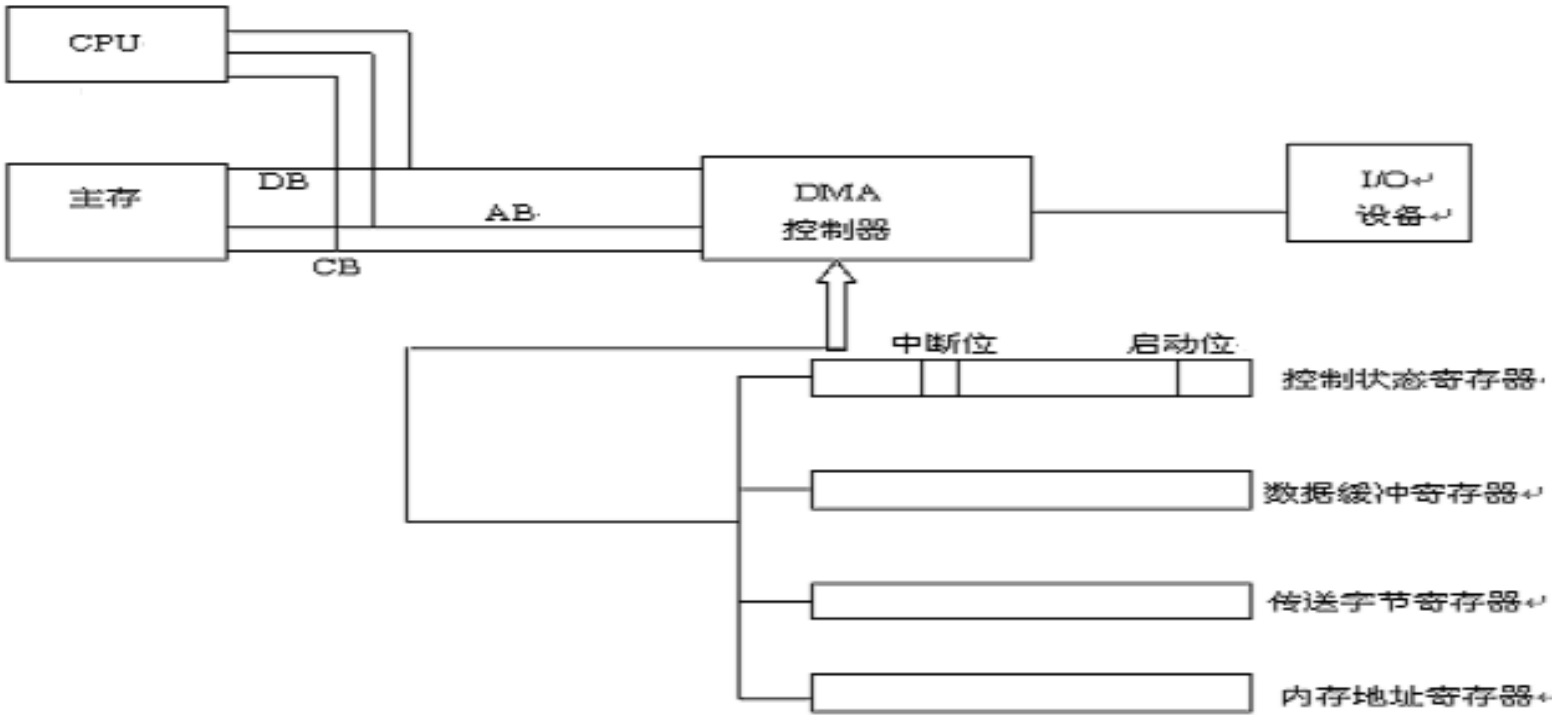
缺点：

每台设备每输入输出一个字节的数据都有一次中断。如果设备较多时，中断次数会很多，使CPU的计算时间大大减少。

为减少中断对CPU造成的负担，可采用DMA方式和通道方式。

6.2.3 DMA方式

控制器功能更强，除有中断功能外，还有一个DMA控制机构。在DMA控制器的控制下，设备同主存之间可成批交换数据，不用CPU干预。



工作过程

1、当进程要求输入时，把内存始址（M）和要传的字节数送入DMA的内存地址寄存器和传送字数寄存器

2、把启动位置1。设备开始工作。进程（A）挂起。调度另一进程（B）

3、一批数据输入完成后，DMA中断B，转向中断处理程序。

4、中断处理程序唤醒A，返回B的断点继续执行。

5、以后OS调度A运行时，A从M处取数据处理。

DMA方式与中断的主要区别

中断方式是在数据缓冲寄存区满后，发中断请求，CPU进行中断处理。

DMA方式则是在所要求传送的数据块全部传送结束时要求CPU进行中断处理。

大大减少了CPU进行中断处理的次数。

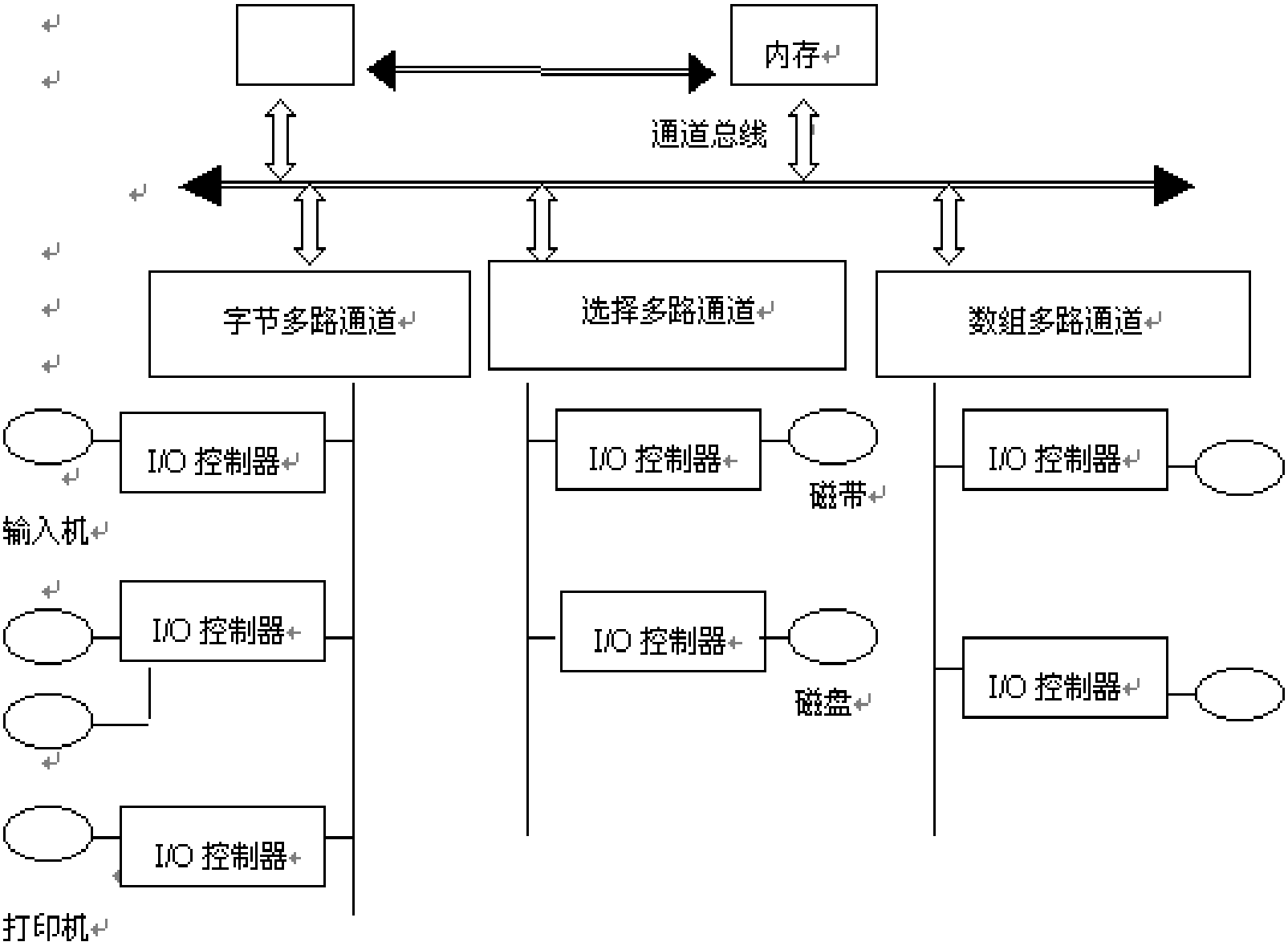
中断方式的数据传送是由CPU控制完成的。

而DMA方式则是在DMA控制器的控制下不经过CPU控制完成的。

6.2.4 通道方式

1、 I/O系统结构

在大型计算机系统中较为典型的I/O系统结构是主机、通道、控制器和外部设备。


内存
通道总线
字节多路通道
选择多路通道
数组多路通道
I/O 控制器
I/O 控制器
I/O 控制器
I/O 控制器
I/O 控制器
I/O 控制器
输入机
磁带
磁盘
打印机

外部设备通常由机械的和电子的两部分组成，电子部分构成控制器，也叫适配器。

一个控制器可交替地控制几台同类设备，例如一个磁盘控制器可以控制两台磁盘驱动器。

在没有通道的计算机系统中，中央处理机是通过控制器控制I/O操作的。

2、通道概念

为使中央处理机从繁忙的I/O处理中摆脱出来，现代大、中型计算机系统中设置了专门的处理I/O操作的处理机，并把这种处理机称为通道。

通道在CPU的控制下独立地执行通道程序，对外部设备的I/O操作进行控制，以实现内存与外设之间成批的数据交换。

通道=I/O处理机

当完成CPU交给的任务后，向CPU发出中断信号，请求CPU的处理。

这样就使得CPU基本上摆脱了I/O操作的处理工作，提高了CPU与设备之间的并行程度，从而提高了整个计算机系统的效率。

通道程序是由通道指令组成，一个通道可以分时的方式执行几道程序。

每道程序控制一台外部设备，因此每道通道程序称为子通道。

3、通道的种类

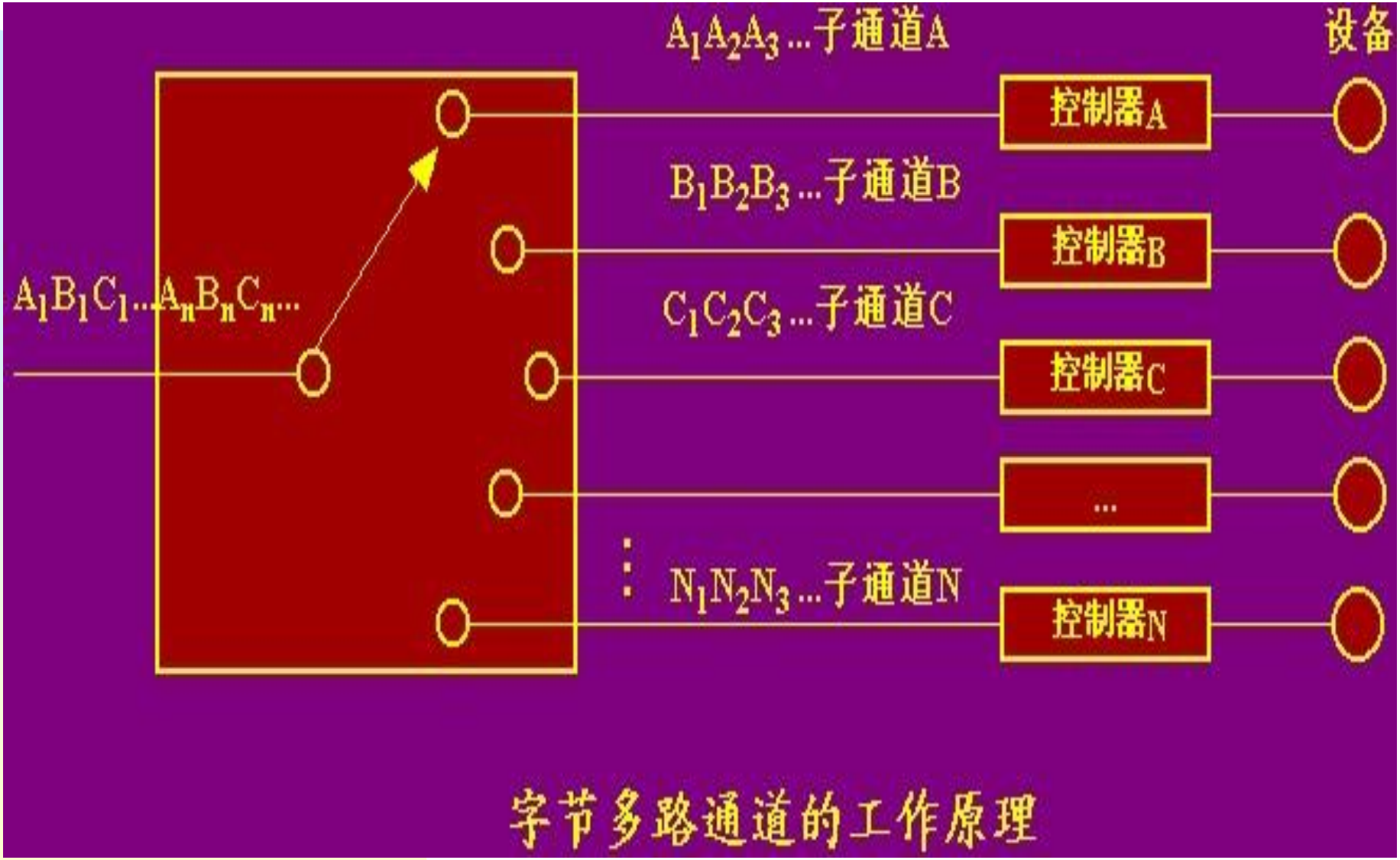
字节多路通道：

字节多路通道是以字节为基本传输单位。

当一子通道控制的某台外设交换了一个字节后，就转向下一个子通道，以控制下一台设备传送一个字节。

这就实现了子通道的循环轮转，以达到多路控制的目的。

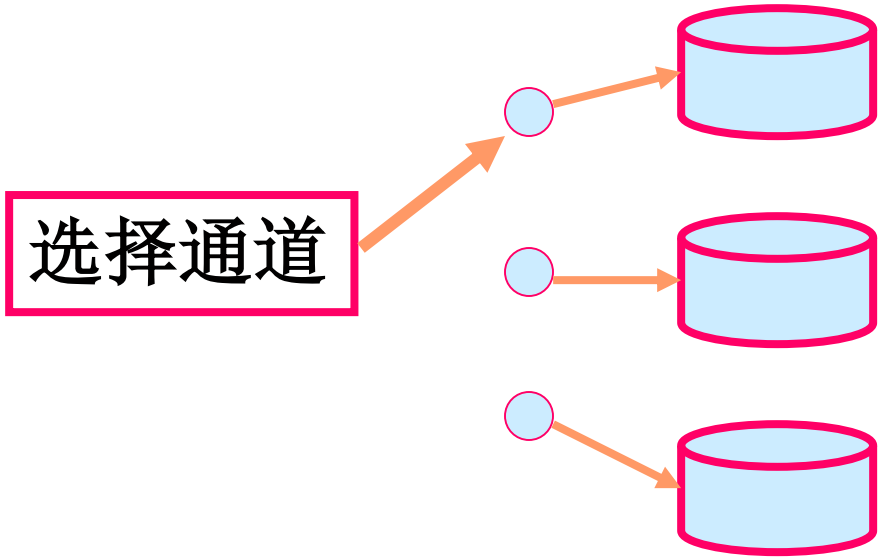
字节多路通道主要用来控制低速、并且以字节为基本传送单位的设备。如打印机。


$A_1A_2A_3$ ...子通道A
设备
控制器A
$B_1B_2B_3$ ...子通道B
控制器B
$A_1B_1C_1$...$A_nB_nC_n$...
$C_1C_2C_3$ ...子通道C
控制器C
...
$N_1N_2N_3$ ...子通道N
控制器N
字节多路通道的工作原理

数据选择通道：

这种通道一次执行一个通道程序，控制一台设备连续地传送一批数据，当一个程序执行完后，才转向下一个程序。

它的优点是传输速度高，缺点是一次只能控制一台设备进行I/O操作。它主要用来控制高速外设。如磁盘。



数据多路通道：

这种通道是上述两种通道的折中，可以分时的方式执行多道程序，每道程序可传送一组数据。

它主要用于中速设备的控制。如磁带机。

在一大型系统中可以同时存在这三种类型的通道以便控制各种不同类型的设备。

4、通道指令和通道程序

通道有它自己的指令系统，用这些指令编写的程序叫通道程序。

通道只能执行通道程序，不可能执行用户进程。

通道程序保存在内存中

5、通道的工作过程

某进程在运行过程中，若提出了I/O请求，则通过系统调用进入操作系统。

系统首先为I/O操作分配通道和外设，然后按I/O请求生成通道程序并存入内存，把起始地址送入通道的**首地址寄存器**（CAW）。

接着CPU发出启动通道的指令。

中央处理机启动通道后，通道的工作过程为：

根据CAW，从内存取出通道指令，送入**通道控制字寄存器**（CCW），并修改CAW，使其指向下一条通道指令。

执行CCW中的通道指令，进行实际的I/O操作，执行完毕后，如果还有下一条指令，则返回前一步，否则转下一步。

发出中断信号通知CPU通道程序已执行完成。

6.3 缓冲技术

6.3.1 引言

6.3.2 常用的缓冲技术

6.3.1 引言

引入缓冲区原因

（1）CPU与I/O设备速度不匹配

（2）减少对CPU的中断频率

（3）数据粒度不匹配

（4）提高CPU和I/O设备的并行性

6.3.2 常用的缓冲技术

1、单缓冲

2、双缓冲

3、环形缓冲

4、缓冲池

1、单缓冲

进程发出一I/O请求时，分配一缓冲区

输入：先送入缓冲区，OS再将数据传给进程

输出：先传入缓冲区，OS再将数据送出到设备

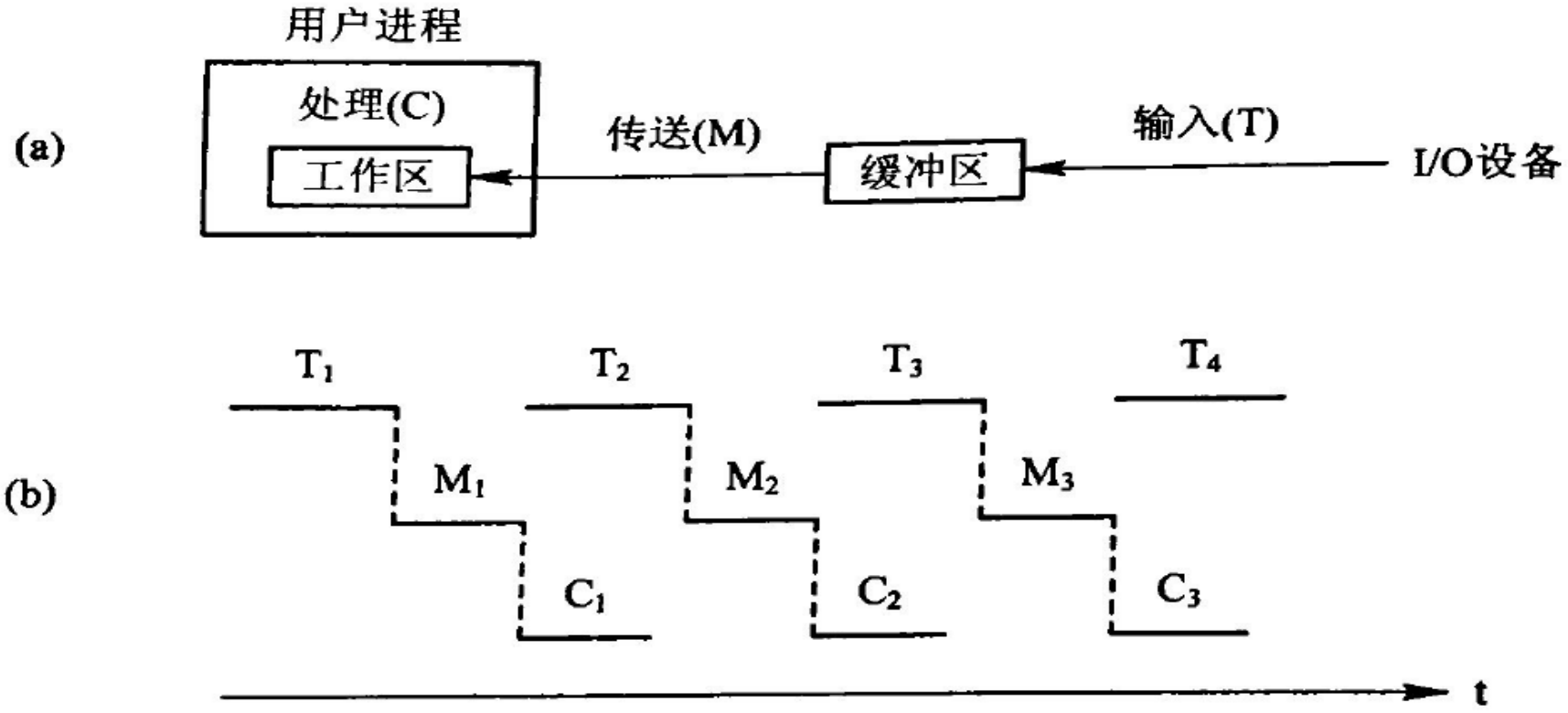


图 6-23　单缓冲工作示意图

2、双缓冲技术

为了加快输入输出速度，引入双缓冲技术。

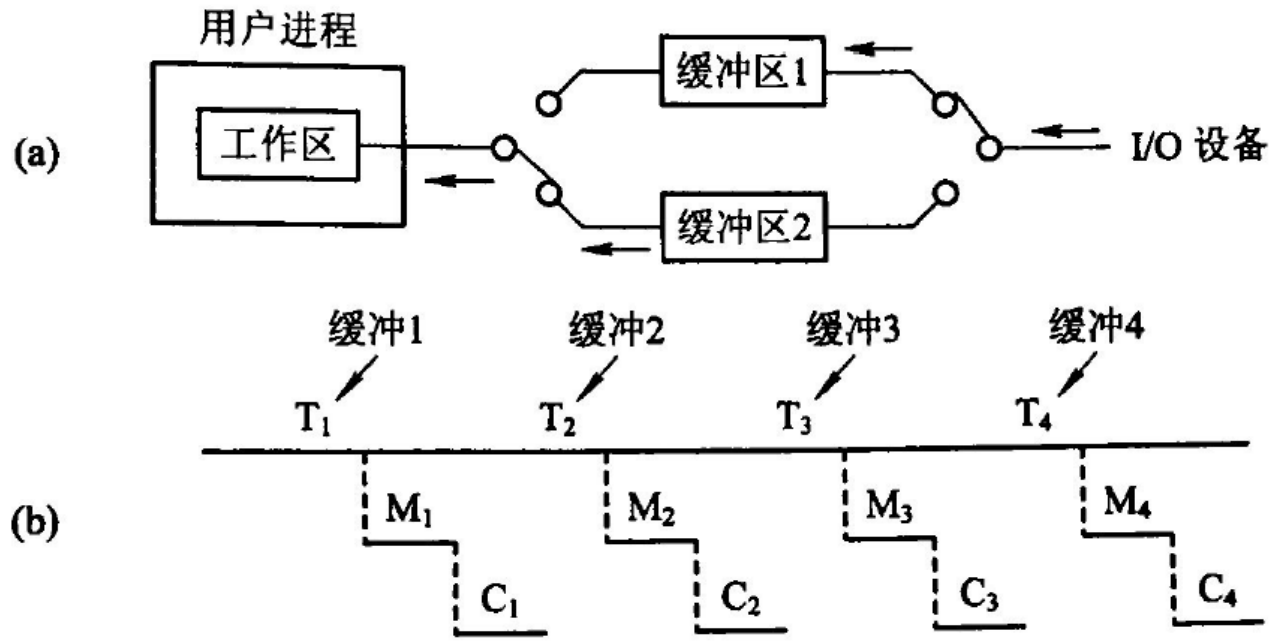


图 6-24　双缓冲工作示意图

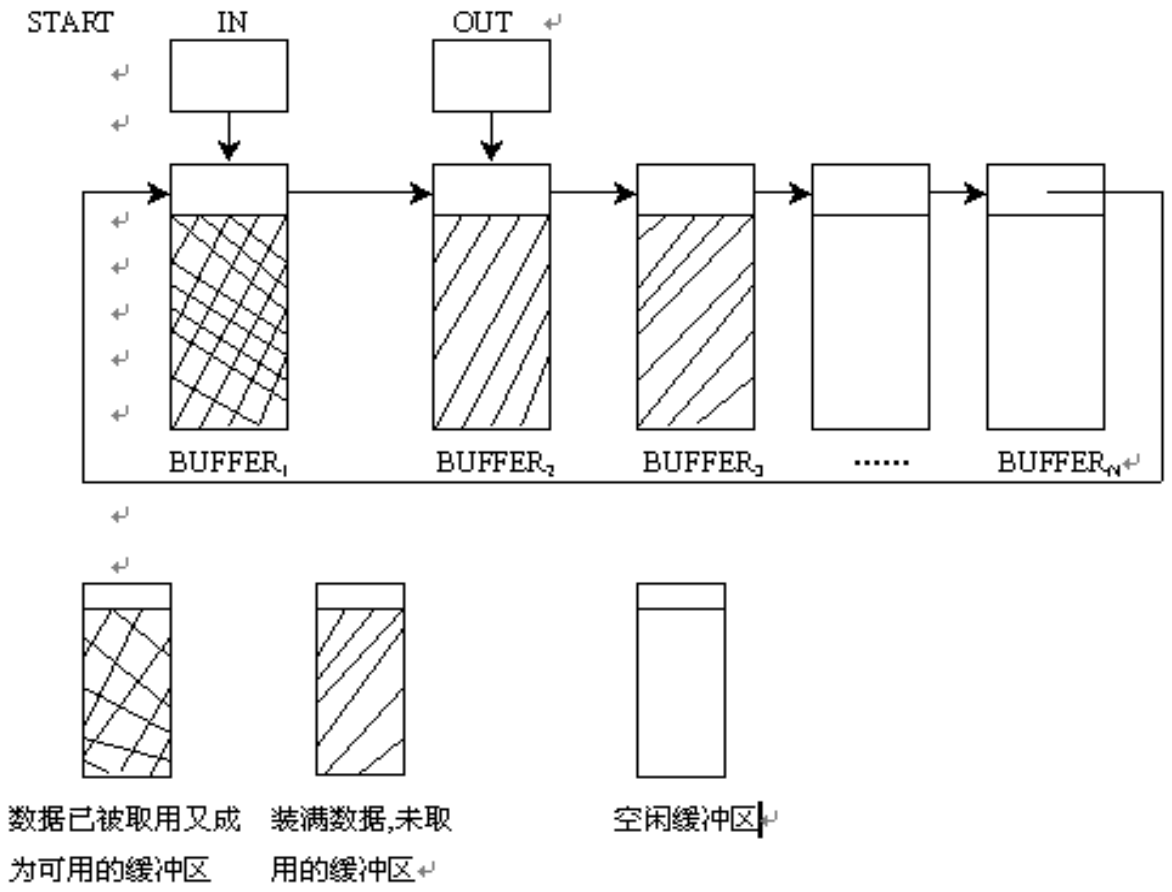
3 环形缓冲技术

当生产和消费数据的速度基本匹配时，双缓冲能获得较好效果

若两者速度相差甚远时，效果不太理想

随着缓冲区的数量增加，使情况有所改善。因此引入环形缓冲技术

在主存中分配**一组大小相等**的存储区作为缓冲区，并将这些缓冲区链接起来形成环形缓冲链。



4、缓冲池

环形缓冲区一般用于特定的进程，属于专用缓冲区

当系统较大时，将会有许多这样的环形缓冲区，这不仅要消耗大量的内存空间，利用率也不高

缓冲池中的缓冲区可供多个进程共享。

缓冲池由内存中一组大小相等的缓冲区组成

缓冲池属于系统资源，由系统进行管理

缓冲池中各缓冲区可用于输出信息，也可用于输入信息，并可根据需要组成各种缓冲区队列

6.4 设备分配

6.4.1 设备分配方式

6.4.2 设备分配算法

6.4.3 设备分配技术

6.4.1 设备分配方式

静态分配：

在作业级进行的，当一个作业运行之前由系统一次分配满足需要的全部设备，这些设备一直为该作业占用，直到作业撤消。这种分配不会出现死锁，但设备的利用效率较低。

动态分配

在进程运行的过程中进行的，当进程需要使用设备时，通过系统调用命令向系统提出设备请求，系统按一定的分配策略给进程分配所需设备，一旦使用完毕立即释放。显然这种分配方式有利于提高设备的使用效率，但会出现死锁，这是应力求避免的。

6.4.2 设备分配算法

1、先请求先服务

2、优先级高的优先服务

6.4.3 设备分配技术

根据设备的特性把设备分成独占设备、共享设备和虚拟设备三种。

针对这三种设备采用三种分配技术:

独享分配

共享分配

虚拟分配

独享分配

独占型设备：例如打印机，键盘，显示器

若对这些设备不采用独享分配就会造成混乱。因此对独占设备一般采用独享分配，即当进程申请独占设备时，系统把设备分配给这个进程，直到进程释放设备。

共享分配

共享设备：例如磁盘

对这类设备的分配是采用动态分配的方式进行的，当一个进程要请求某个设备时，系统按照某种算法立即分配相应的设备给请求者，请求者使用完后立即释放。

虚拟分配

系统中独占设备的数量有限

往往只有很少时间在工作，别的进程又因得不到而不能运行，影响到整个计算机系统的效率

独占设备一般是低速的，若采用联机操作，也会增加进程的运行时间，影响计算机系统的效率

虚拟分配是针对虚拟设备而言的

其实现的过程是：

申请独占设备→分配共享设备的一部分存储空间

交换信息：系统←→存储空间←→设备

把共享设备中代替独占设备的那部分**存储空间和相应的控制结构**称为**虚拟设备**，并把对这类设备的分配称作虚拟分配。

SPOOLing系统

Simultaneaus Periphernal Operations On-Line(外部设备同时联机操作）。

在单道批处理时期，用脱机I/O可以提高CPU利用率。多道出现后可以利用一道程序来模拟脱机I/O中的卫星机，这样可实现在主机控制下的脱机I/O功能。

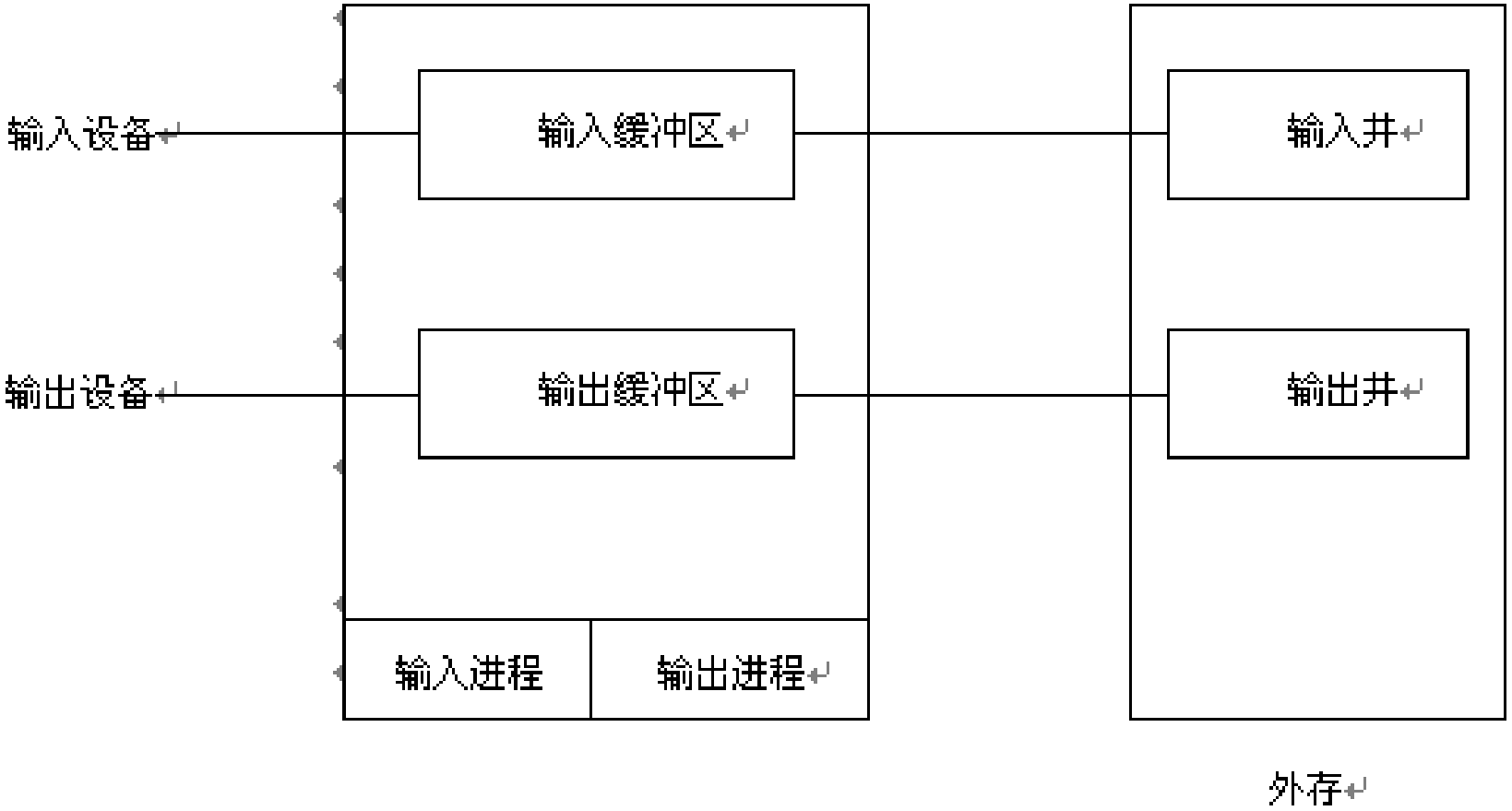
我们把这种在联机情况下实现的同时外围操作称为SPOOLing，也称为假脱机操作。

SPOOLing系统的组成

1、输入井和输出井

2、输入缓冲区和输出缓冲区

3、输入进程和输出进程


输入设备
输出设备
输入缓冲区
输出缓冲区
输入进程
输出进程
输入井
输出井
外存

SPOOLing系统工作原理

作业执行前预先将程序和数据输入到输入井中

作业运行后，使用数据时，从输入井中取出

作业执行不必直接启动外设输出数据，只需将这些数据写入输出井中

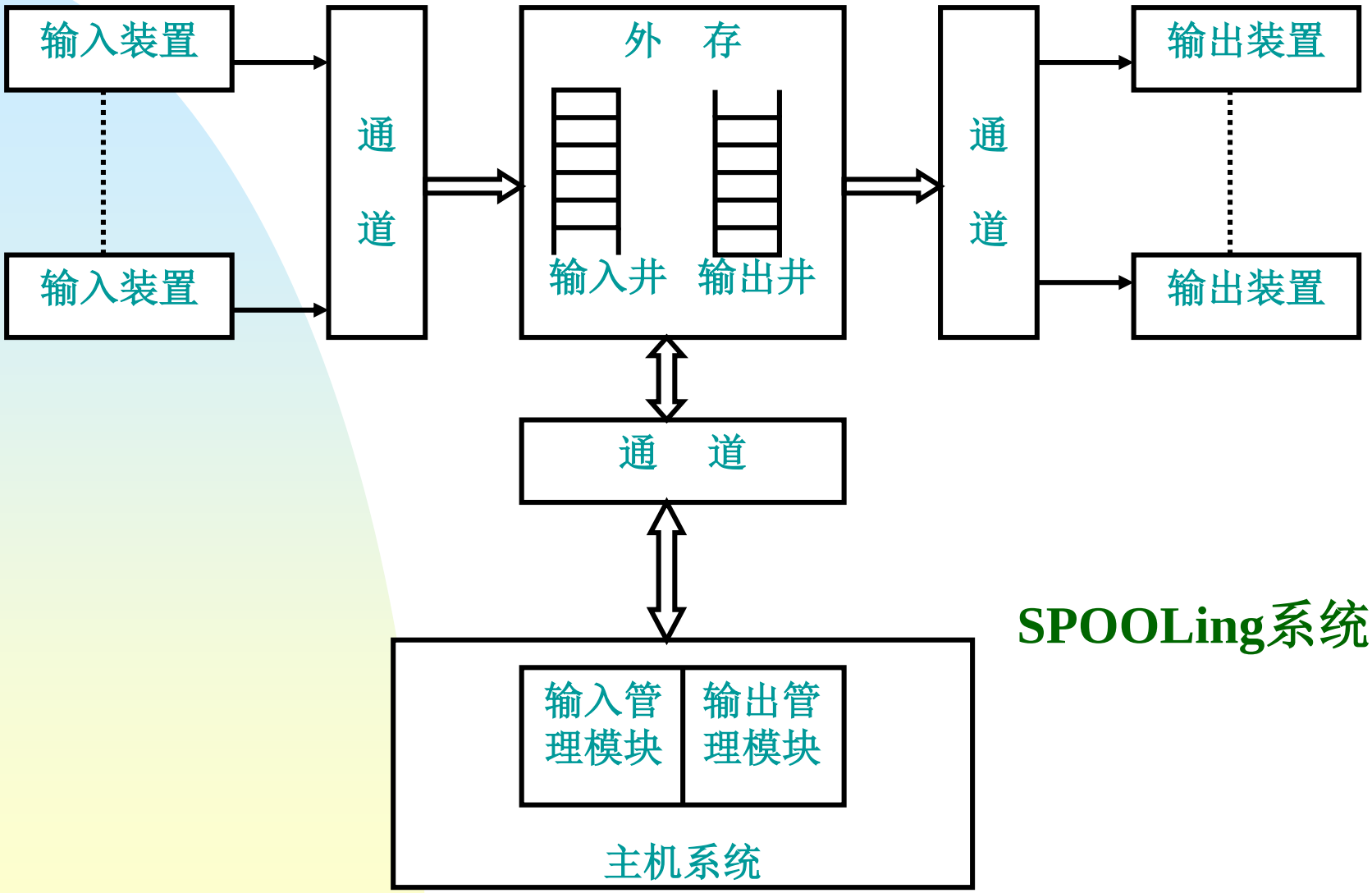
作业全部运行完毕，再由外设输出全部数据和信息

好处：

实现了对作业输入、组织调度和输出的统一管理

使外设在CPU直接控制下，与CPU并行工作（假脱机）

图示



SPOOLing系统

SPOOLing系统的特点

1、提高了I/O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

6.5 磁盘概述

几乎所有随机存取的文件，都存放在磁盘上

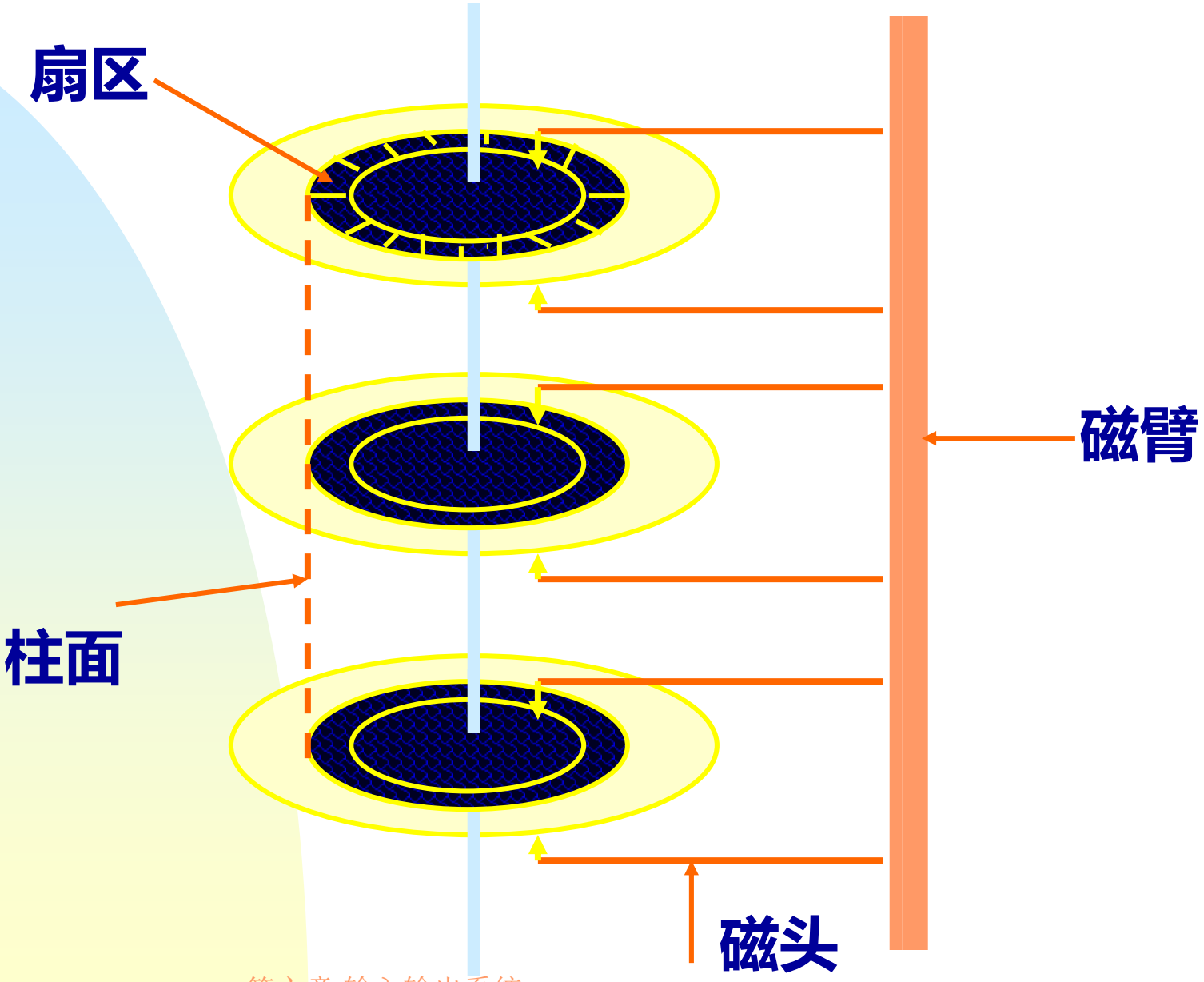
磁盘I/O速度将直接影响文件系统的性能

硬盘分为两种：

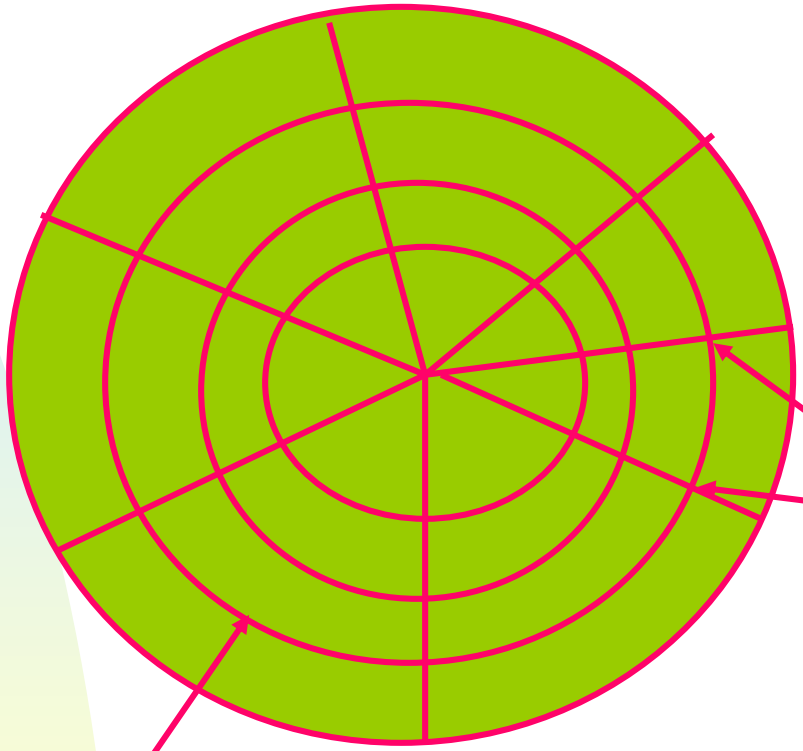
固定头磁盘：每个磁道设置一个磁头

移动头磁盘：一个盘面只有一个磁头

侧视图



俯视图



扇区

磁道

柱面、磁头、扇区

信息记录在磁道上，多个盘片，正反两面都用来记录信息，每面一个磁头

所有盘面中处于同一磁道号上的所有磁道组成一个柱面

第个扇区大小为512字节

物理地址形式：

柱面号

磁头号

扇区号

典型参数

20G：

39813 柱面

16 头

63 扇区

60G：

28733 柱面

16 头

255 扇区

磁盘的访问过程

由三个动作组成：

寻道：磁头移动定位到指定磁道

旋转延迟：等待指定扇区从磁头下旋转经过

数据传输：数据在磁盘与内存之间的实际传输

磁盘的访问时间

寻道时间Ts：大约几ms到几十ms

旋转延迟时间Tr：对于7200转/分，平均延迟时间为4.2ms

数据传输时间Tt：目前磁盘的传输速度一般有几十M/s，传输一个扇区的时间小于0.05ms

访问时间：Ts+1/2r + b/rN

Ts:寻道时间　　r:每秒转速　（1/2r：旋转延迟）

b：读写的字节数　　N：一条磁道的字节数

（b/rN:传输时间）

某软盘有40个磁道，磁头从一个磁道移到另一个磁道需要6ms。文件在磁盘上非连续存放，逻辑上相邻的数据块平均距离是13磁道，每块的旋转延迟时间及传输时间分别为100ms和25ms，问读取一个100块的文件需要多少时间？如果系统对磁盘进行了整理，让同一个磁盘块尽可能靠拢，从而使相邻的数据块平均距离降为2磁道，这时读取100块的文件需要多少时间？

解答：

磁盘访问时间=寻道时间+延迟时间+传输时间

整理前：寻道时间：13*6=78，则读取一块时间：78+100+25=203ms

100块：100*203

整理后：寻道时间：2*6=12

时间：（12+100+25）*100

思考

要提高磁盘的数据访问速度，主要应在哪方面下功夫？

分析

要提高磁盘的访问速度主要应从以下两方面入手：

数据的合理组织

磁盘的调度算法

6.6 磁盘调度算法

当多个访盘请求在等待时，采用一定的策略，对这些请求的服务顺序调整安排，旨在降低平均磁盘服务时间，达到公平、高效

公平：一个I/O请求在有限时间内满足

高效：减少设备机械运动所带来的时间浪费

6.6.1 先来先服务

6.6.2 最短寻道时间优先

6.6.3 扫描算法

6.6.4 单向扫描调度算法

6.6.1 先来先服务

按访问请求到达的先后次序服务

优点：简单，公平；

缺点：效率不高，相邻两次请求可能会造成最内到最外的柱面寻道，使磁头反复移动，增加了服务时间，对机械也不利

例

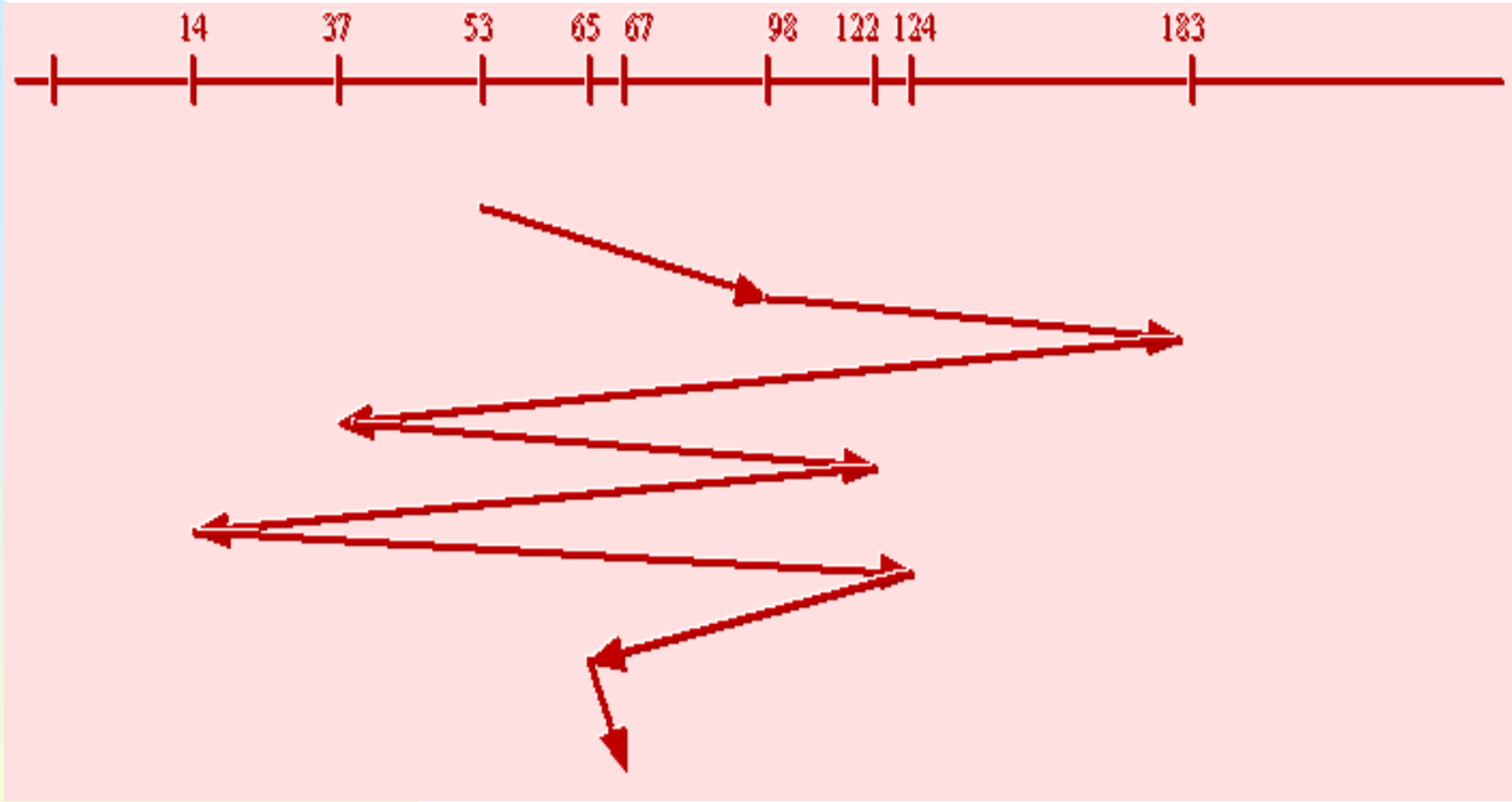
假设磁盘访问序列：98，183，37，122，14，124，65，67

读写头起始位置：53

安排磁头服务序列

计算磁头移动总距离（道数）

图解



98，183，37，122，14，124，65，67

磁头走过的总道数：640

6.6.2 最短寻道时间优先

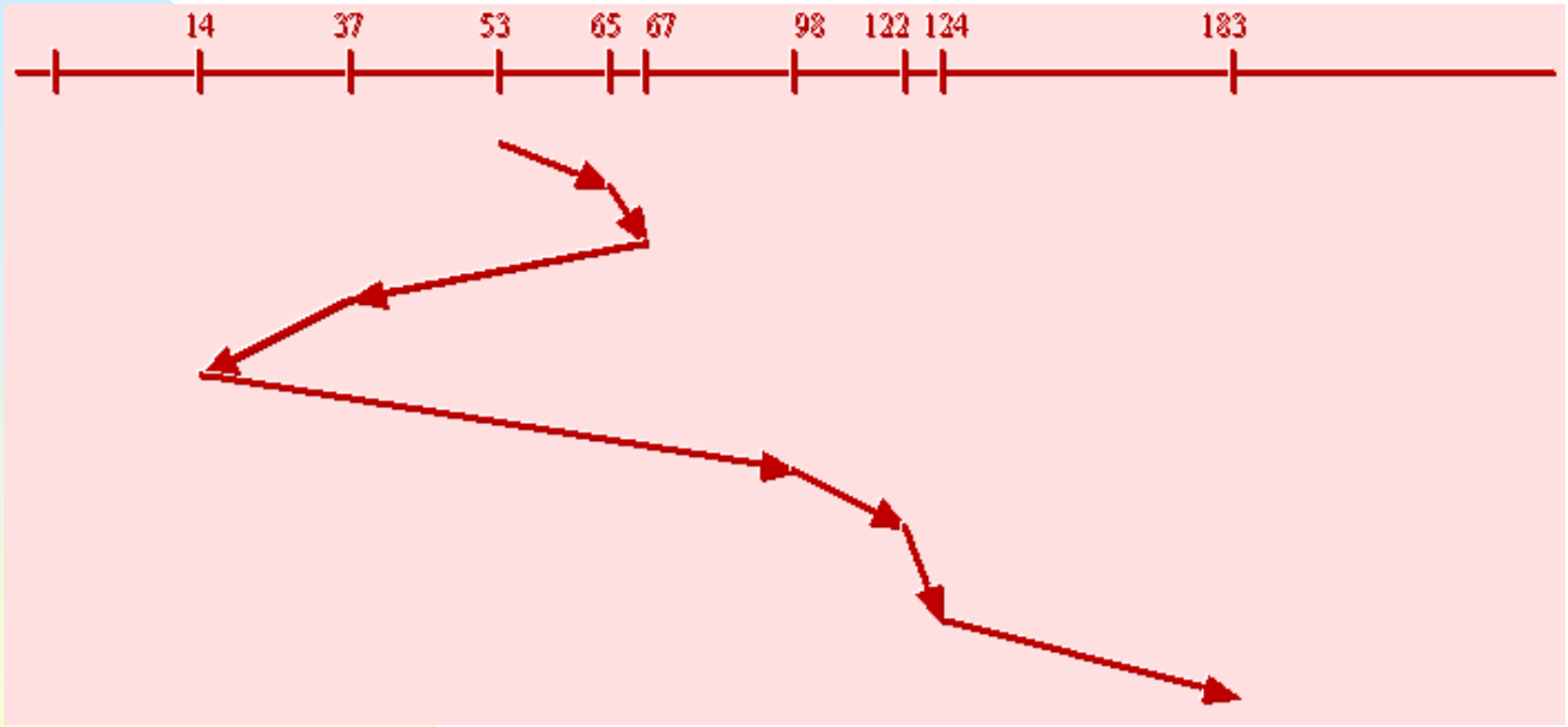
优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先

优点：改善了磁盘平均服务时间；

缺点：造成某些访问请求长期等待得不到服务

图解

98，183，37，122，14，124，65，67



65，67，37，14，98，122，124，183

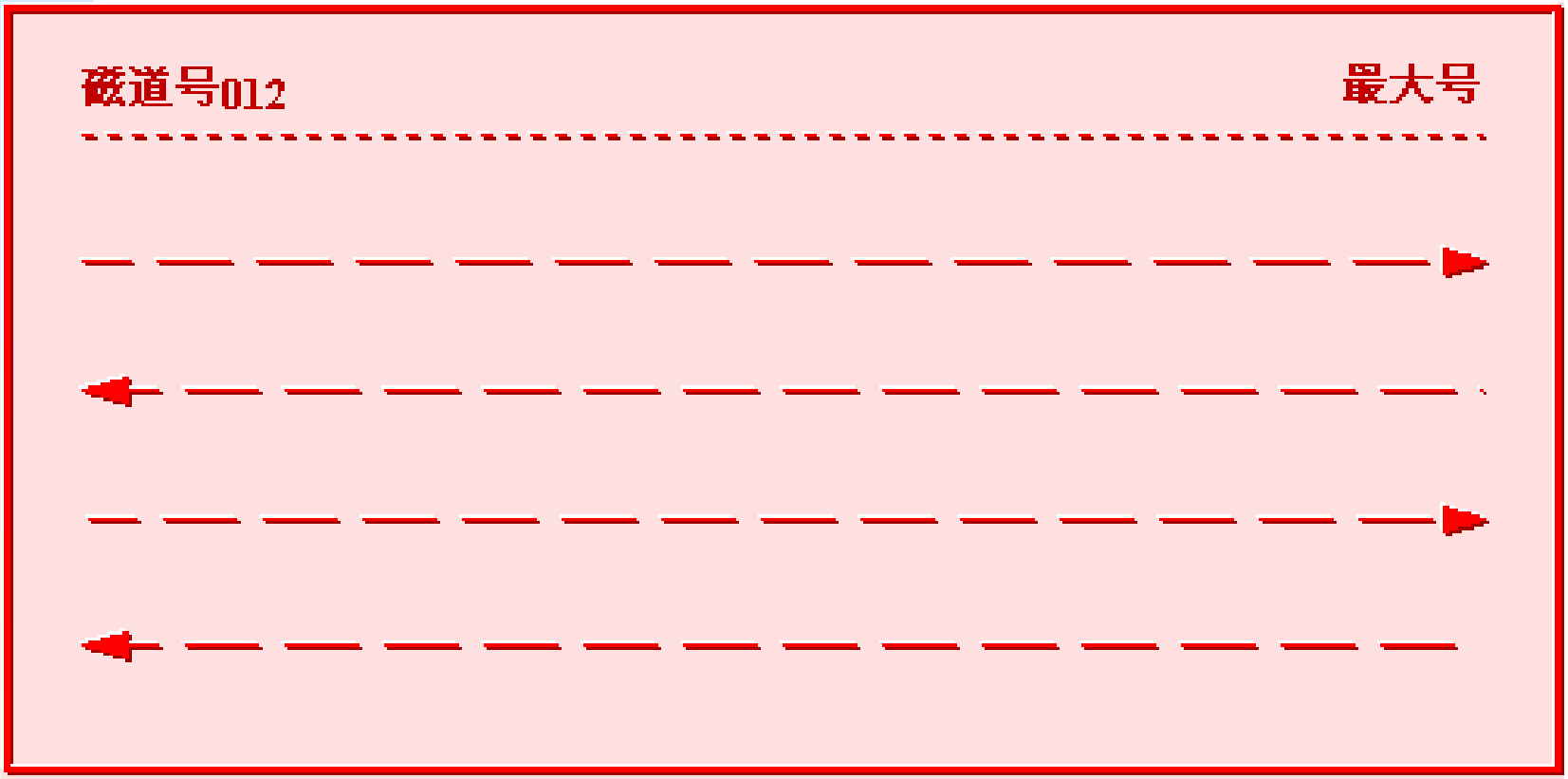
磁头走过的总道数：236

6.6.3 扫描算法（电梯算法）

克服了最短寻道优先的缺点，既考虑了距离，同时又考虑了方向

具体做法：当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复

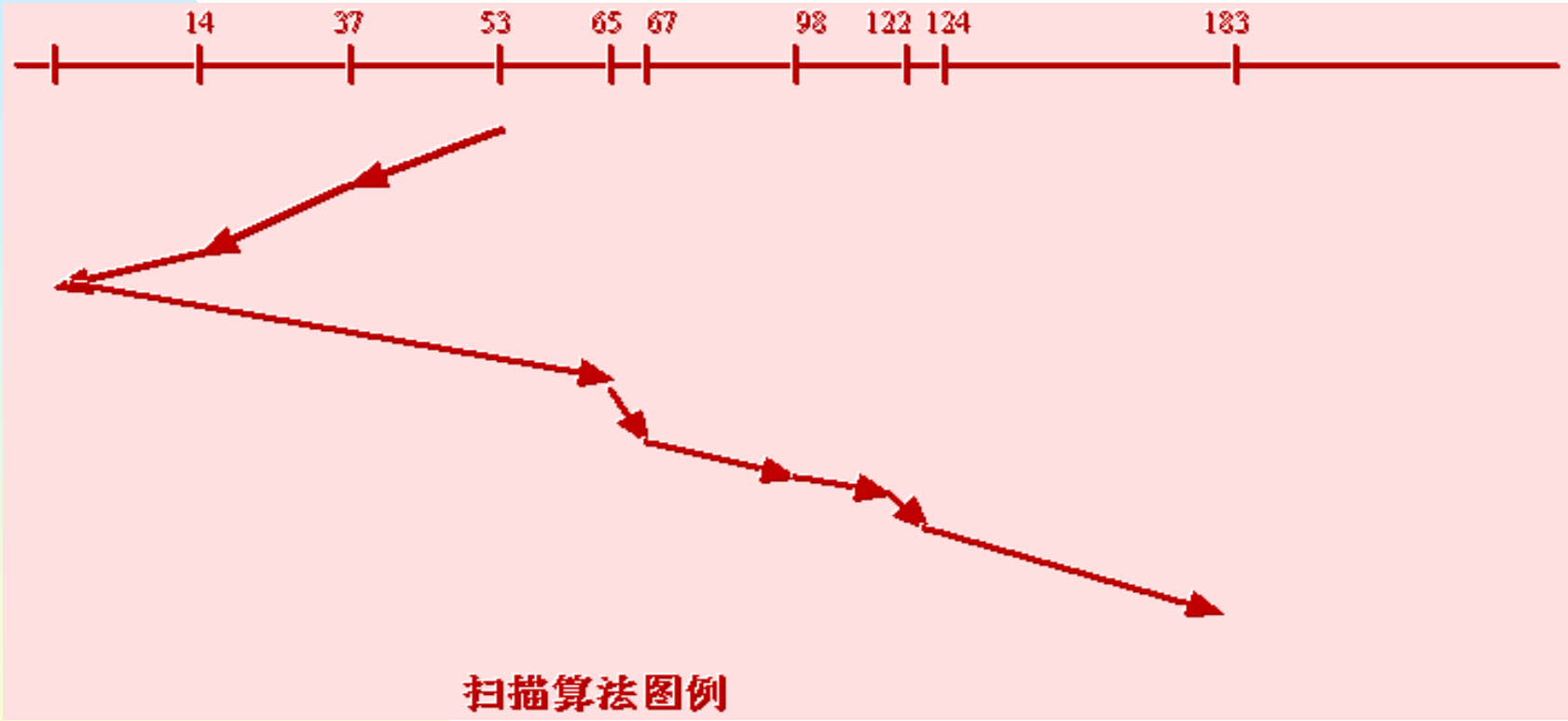
图



扫描算法（电梯算法）的磁头移动轨迹

图解

98，183，37，122，14，124，65，67



37，14，65，67，98，122，124，183

磁头走过的总道数：208

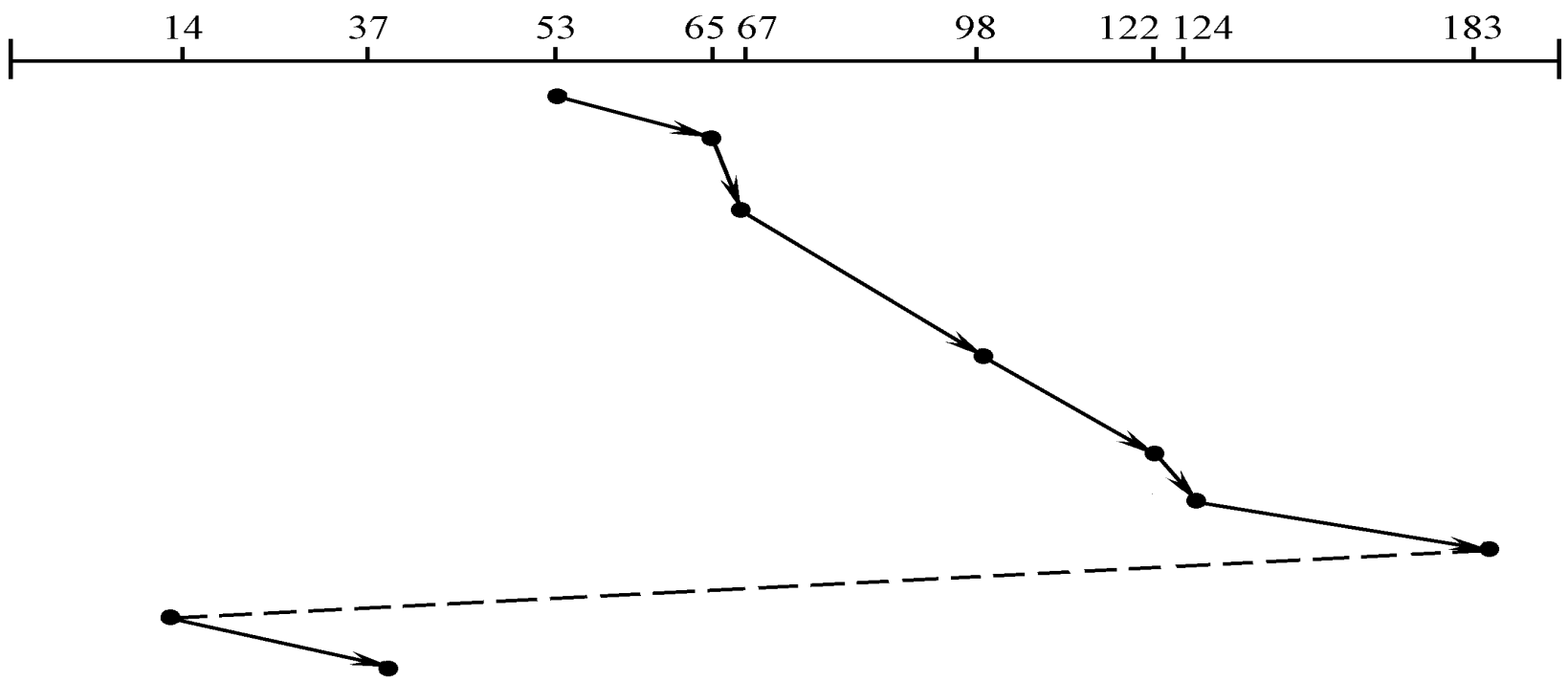
6.6.4 单向扫描调度算法

也称循环扫描算法。

电梯算法杜绝了饥饿，但当请求对磁道的分布是均匀时，磁头回头，近磁头端的请求很少（因为磁头刚经过），而远端请求较多，这些请求等待时间要长一些。

- 总是从0号柱面开始向里扫描。移动臂到达最后个一个柱面后，立即带动读写磁头快速返回到0号柱面。返回时不为任何的等待访问者服务。返回后可再次进行扫描

图解



调度算法的选择

实际系统相当普遍采用最短寻道时间优先算法，因为它简单有效，性价比好。

扫描算法更适于磁盘负担重的系统。

磁盘负担很轻的系统也可以采用先来先服务算法

一般要将磁盘调度算法作为操作系统的单独模块编写，利于修改和更换。