

# 第六章 输入输出系统



# 关于输入输出系统

- 计算机系统的一个重要组成部分是I/O系统。
- 在该系统中包括有用于实现信息输入、输出和存储功能的设备和相应的设备控制器，在有的大中型机中，还有I/O通道或I/O处理机。
- 设备管理的对象主要是I/O设备，还可能要涉及到设备控制器和I/O通道。而设备管理的基本任务是完成用户提出的I/O请求，提高I/O速率以及改善I/O设备的利用率。
- 设备管理的主要功能有缓冲区管理、设备分配、设备处理、虚拟设备及实现设备独立性等。我们主要对I/O设备和设备控制器等硬件作一扼要的阐述。



# 第六章 输入输出系统

- 6.1 I/O系统的功能、模型和接口**
- 6.2 I/O设备和设备控制器**
- 6.3 中断机构和中断处理程序**
- 6.4 设备驱动程序**
- 6.5 与设备无关的I/O软件**
- 6.6 用户层的I/O软件**
- 6.7 缓冲区管理**
- 6.8 磁盘存储器的性能和调度**



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

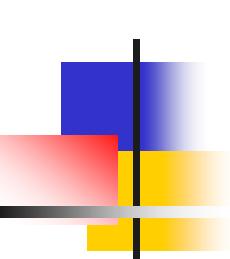
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**



# 6.1 I/O系统

## I/O系统的主要任务

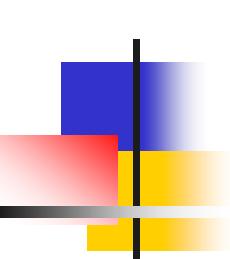
- 完成用户提出的I/O请求，提高I/O的速率，以及提高设备的利用率，并能为更高层的进程方便地使用这些设备提供手段。

## I/O系统的层次结构

- 用户层I/O软件
- 设备独立性软件
- 设备驱动程序
- 中断处理程序

## I/O系统接口

- 根据设备类型的不同，可分为若干个接口。



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

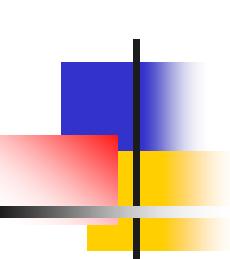
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

**6.7 缓冲区管理**

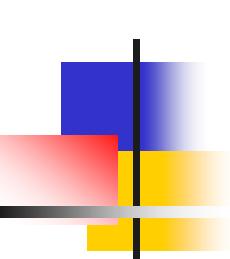
**6.8 磁盘存储器的性能和调度**



## 6.2.1 I/O设备

将分为以下两部分来了解**I/O**设备：

1. **I/O**设备的类型
2. 设备与控制器之间的接口

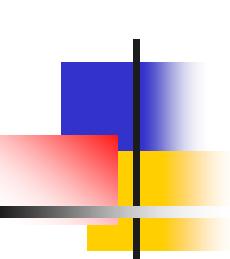


## 6.2.1 I/O设备

### 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：数据传输速率、数据的传输单位、设备共享属性等。因而从以下不同角度进行分类。

- 按传输速率分类
- 按信息交换的单位分类
- 按设备的共享属性分类
- 按设备的使用特性

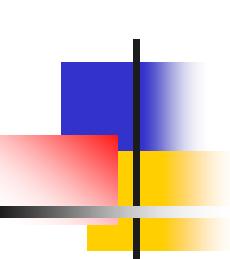


## 6.2.1 I/O设备

### 1. I/O设备的类型

- 设备按传输速率分类：

- **低速设备**：传输速率仅为每秒钟几个字节至数百个字节的一类设备。如键盘、鼠标、语音输入和输出设备等。
- **中速设备**：传输速率为每秒钟数千个字节至数万个字节的一类设备。如行式打印机、激光打印机等。
- **高速设备**：传输速率为每秒钟数百千个字节至数十兆字节的一类设备。如磁带机、磁盘机、光盘机等。



## 6.2.1 I/O设备

### 1. I/O设备的类型

- 设备按传输速率分类
- 设备按信息交换的单位分类：
  - **块设备**：用于存储信息。对于信息的存取总是以数据块为单位。典型例子是磁盘。该类设备基本特征是传输速率较高，另一特征是可寻址。工作方式常采用**DMA**方式。
  - **字符设备**：用于数据的输入和输出。基本单位是字符。如交互式终端、打印机等。其基本特征是传输速率较低，另一特征是不可寻址。工作方式常采用中断方式。

# 6.2.1 I/O设备

## 1. I/O设备的类型

- 设备按传输速率分类
- 设备按信息交换的单位分类
- 设备按其共享属性分类：
  - 独占设备：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。应互斥的访问之。
  - 共享设备：指在一段时间内允许多个进程同时访问的设备。对每一时刻而言仍然是一个进程访问。如磁盘。
  - 虚拟设备：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。
- 设备按其使用特性分类：存储设备、输入\输出设备

# 6.2.1 I/O设备

## 1. I/O设备的类型

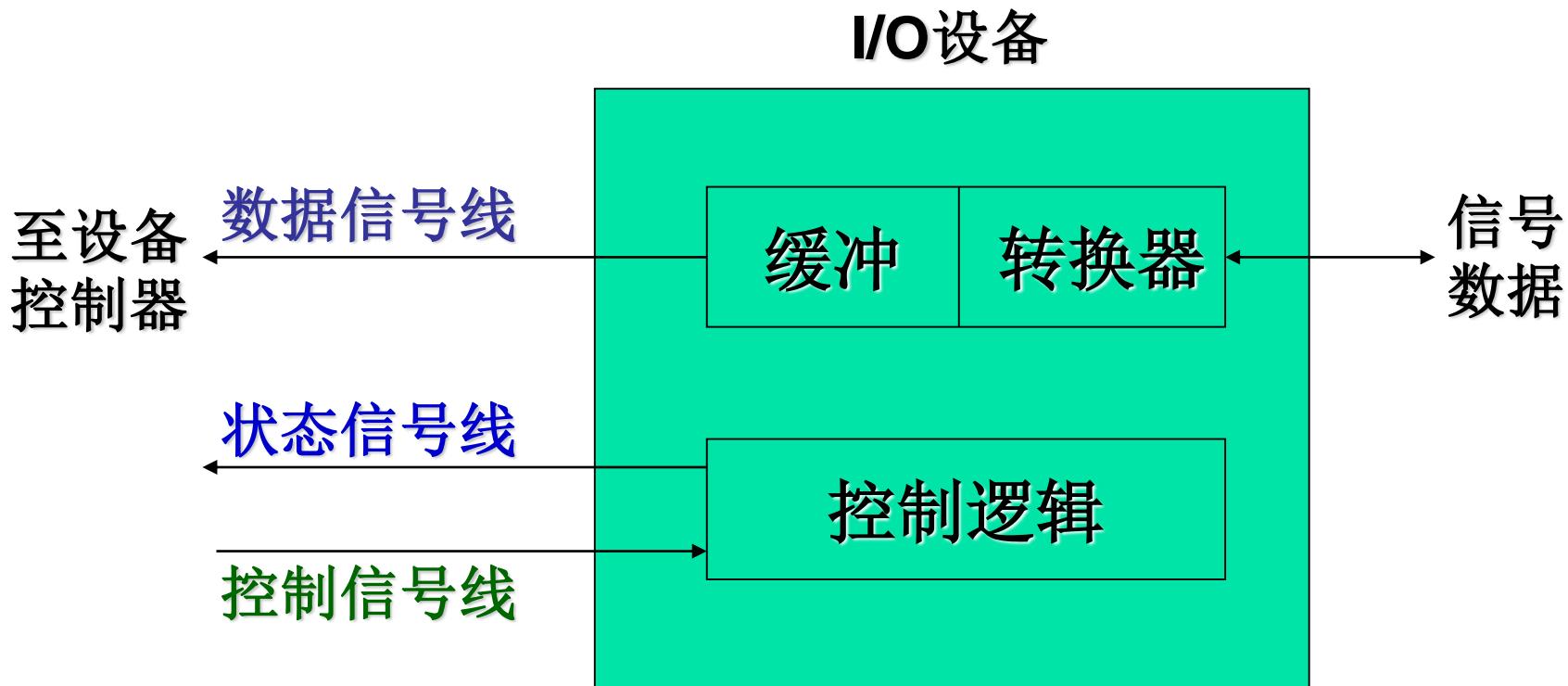
## 2. 设备与控制器之间的接口

通常设备并不是直接与**CPU**进行通信，而是与设备控制器通信，因此，在设备与设备控制器之间有一接口，在该接口中有三种类型的信号，各对应一条信号线。

(如下页图)

- 数据信号线
- 控制信号线
- 状态信号线

# 设备与控制器之间的接口



# 6.2.1 I/O设备

## 2. 设备与控制器之间的接口

### ■ 数据信号线

用于在设备和设备控制器之间传送数据信号。对输入设备而言，由外界输入的信号经转换器转换后所形成的数据，通常先送入缓冲区中，当数据量达到一定量时，在从缓冲器通过一组数据信号线传送给设备控制器。对输出设备而言，则是将从设备控制器经过数据信号线传送来的一批数据，先暂存于缓冲器中，经转换器作适当转换后逐个字符的输出。

### ■ 控制信号线

作为设备控制器向I/O设备发送控制信号的通路。该信号规定了设备将要执行的操作，如读（指由设备向控制器传送数据）或写操作（从控制器接收数据），或执行磁头移动等操作。

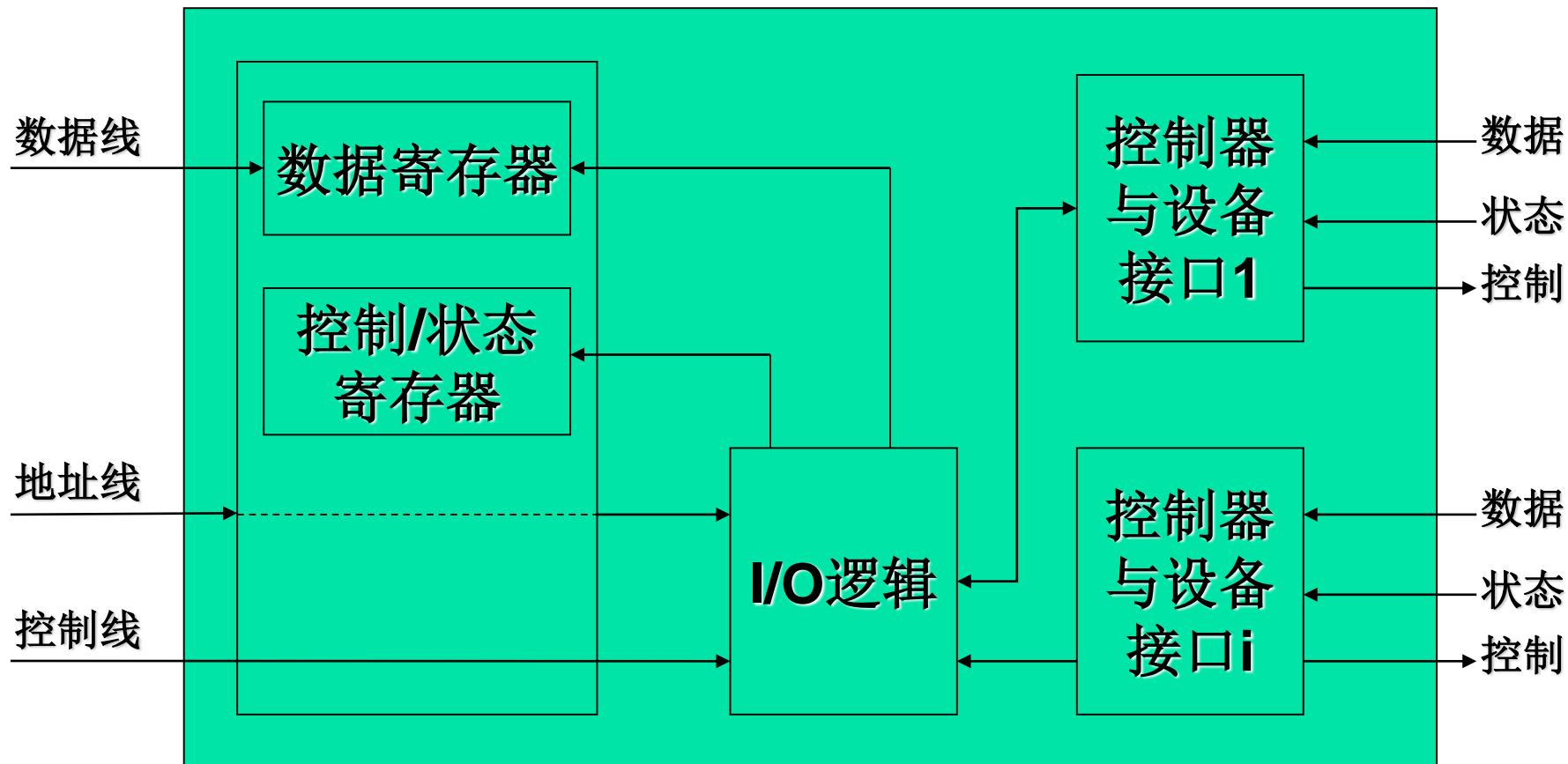
### ■ 状态信号线

用于传送指示设备当前状态的信号。设备的当前状态有正在读（或写）；设备已读（写）完成，并准备好新的数据传送。

## 6.2.2 设备控制器

CPU与控制器接口

控制器与设备接口



设备控制器的组成

## 6.2.4 I/O通道

### ■ I/O通道设备的引入

尽管有了设备控制器，已能大大减少**CPU**对**I/O**的干预，但当主机的外设很多时，**CPU**的负担仍然很重。为此又在**CPU**和设备控制器之间增设了通道。其主要目的是为了建立独立的**I/O**操作，去解放**CPU**。在设置通道后，**CPU**只需向通道发送一条**I/O**指令。通道完成任务后向**CPU**发中断信号。

**I/O**通道是一种特殊的处理机。与一般处理机不同于两方面：

- 指令类型单一，只用于**I/O**操作；
- 通道没有内存，它与**CPU**共享内存。

### ■ 通道类型

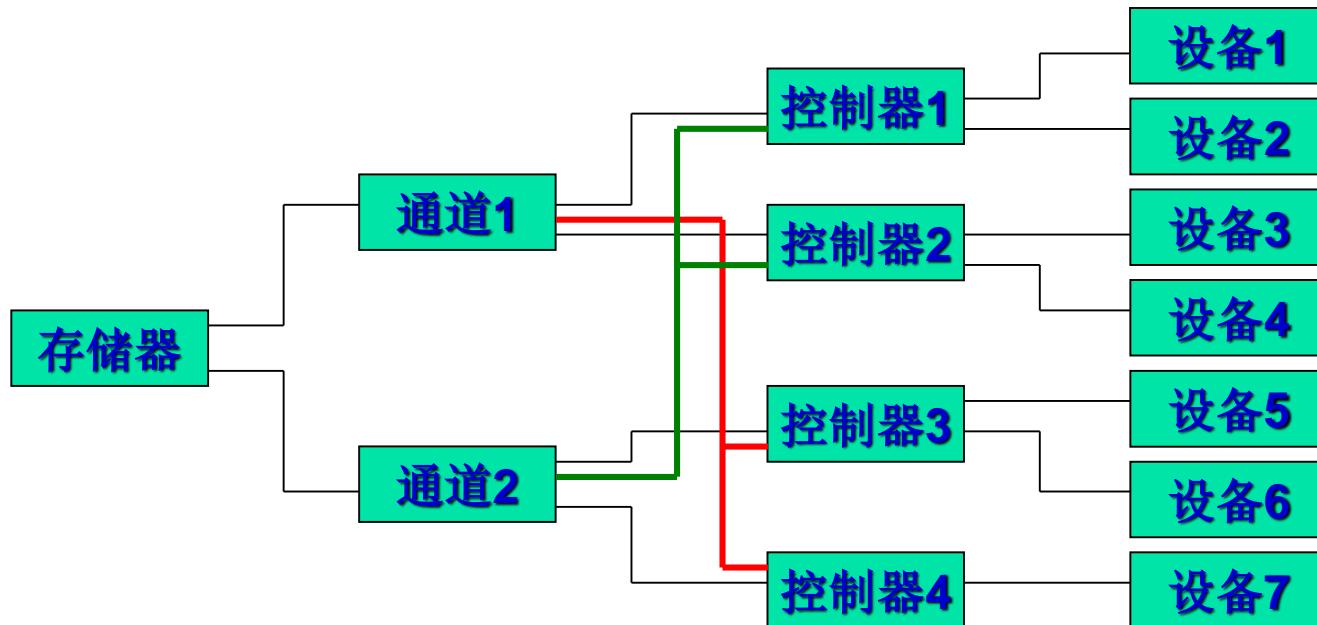
根据信息交换方式可分为以下三种类型：

- 字节多路通道
- 数组选择通道
- 数组多路通道

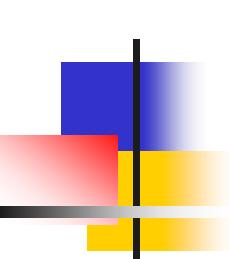
## 6.2.4 I/O通道

### “瓶颈”问题

由于通道价格昂贵，致使数量较少，使它成为I/O系统的瓶颈，进而造成系统吞吐量的下降。如下例所示：



- 解决“瓶颈”问题最有效的办法便是增加设备到主机间的通路而不增加通道，如上图所示：



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序P189**

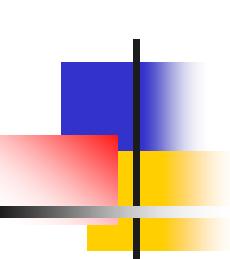
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

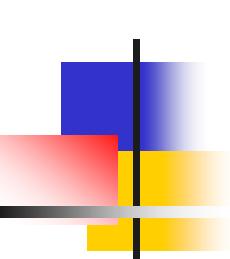
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

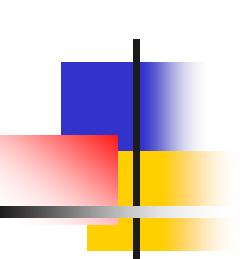
**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**



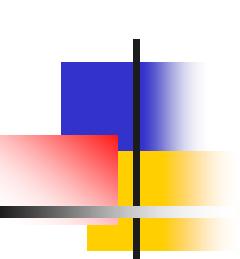
# 设备驱动程序

- 设备处理程序通常又称为设备驱动程序，它是**I/O**系统的高层与设备控制器之间的通信程序。
- 由于驱动程序与硬件密切相关，故通常应为每一类设备配置一种驱动程序。比如打印机和显示器需要不同的驱动程序。



# 对 I/O 设备的控制方式

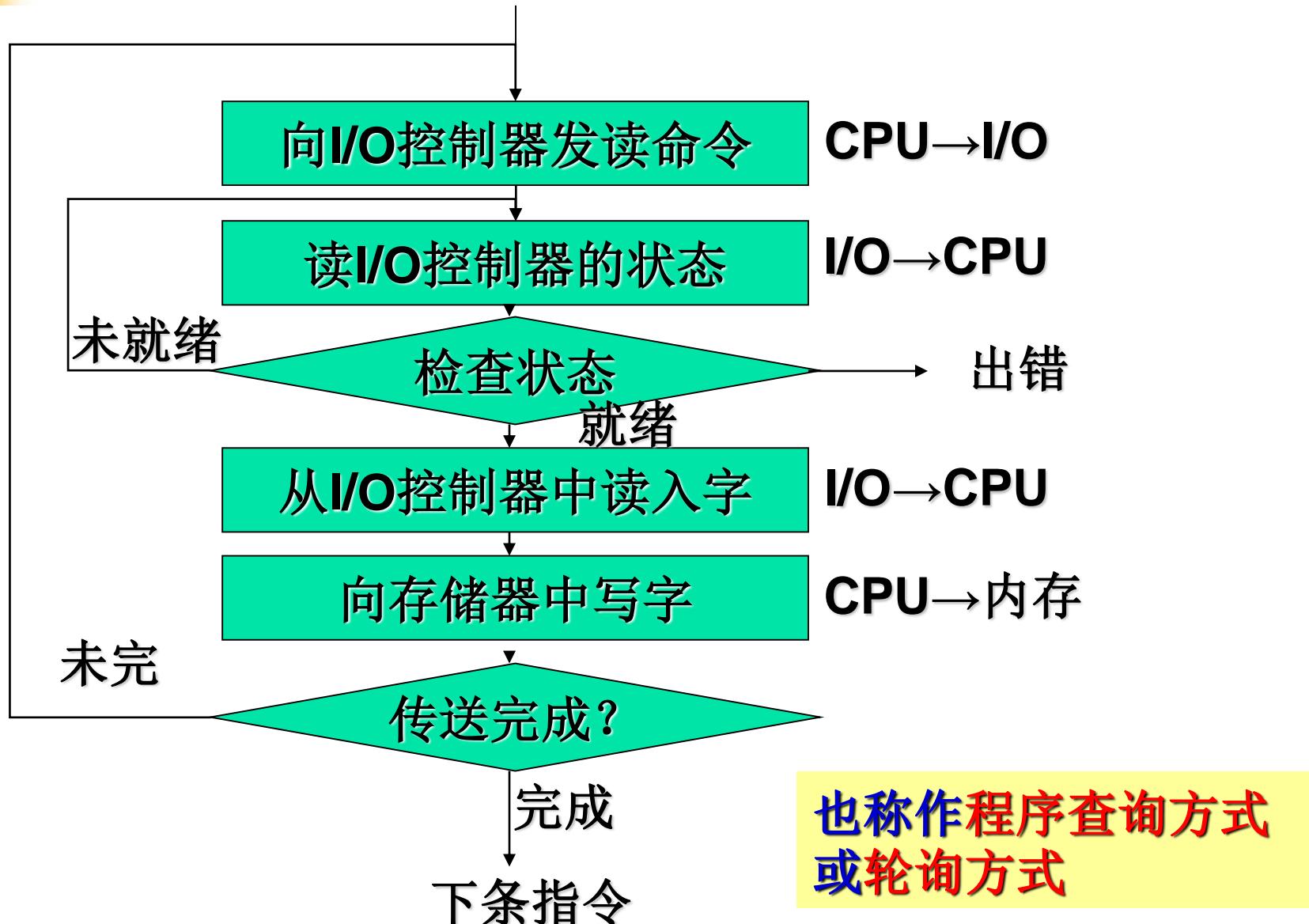
- 使用轮询的可编程I/O方式
- 使用中断的可编程I/O方式
- 直接存储器访问方式
- I/O通道控制方式

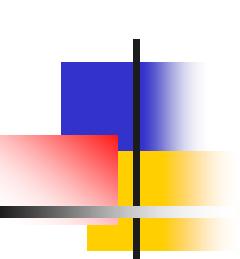


# 对 I/O 设备的控制方式

- 使用轮询的可编程I/O方式
- 使用中断的可编程I/O方式
- 直接存储器访问方式
- I/O通道控制方式

# 程序I/O方式流程图(从外设读数据)



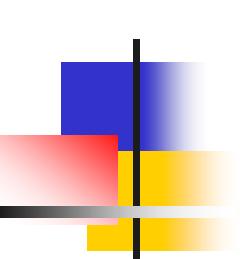


# 对 I/O 设备的控制方式

- 使用轮询的可编程I/O方式
- 使用中断的可编程I/O方式
- 直接存储器访问方式
- I/O通道控制方式

# 中断驱动I/O方式

- 当某进程要启动某个I/O设备时，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命令的要求去控制指定I/O设备。这时CPU与I/O设备并行操作。



# 对 I/O 设备的控制方式

- 使用轮询的可编程I/O方式
- 使用中断的可编程I/O方式
- 直接存储器访问方式
- I/O通道控制方式

# 直接存储器访问DMA I/O控制方式

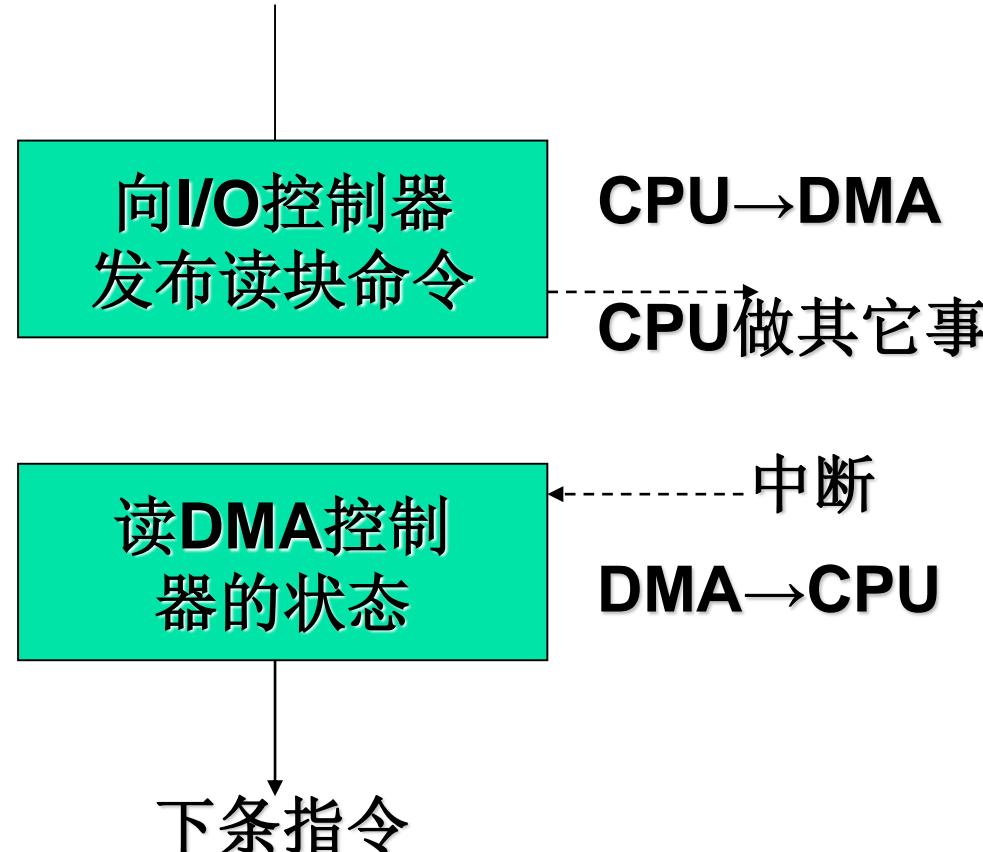
## ■ DMA (**Direct Memory Access**) 控制方式的引入

虽然中断方式比程序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，控制器便要请求一次中断。显然是极其低效的。由此便引入了直接存储器访问方式。

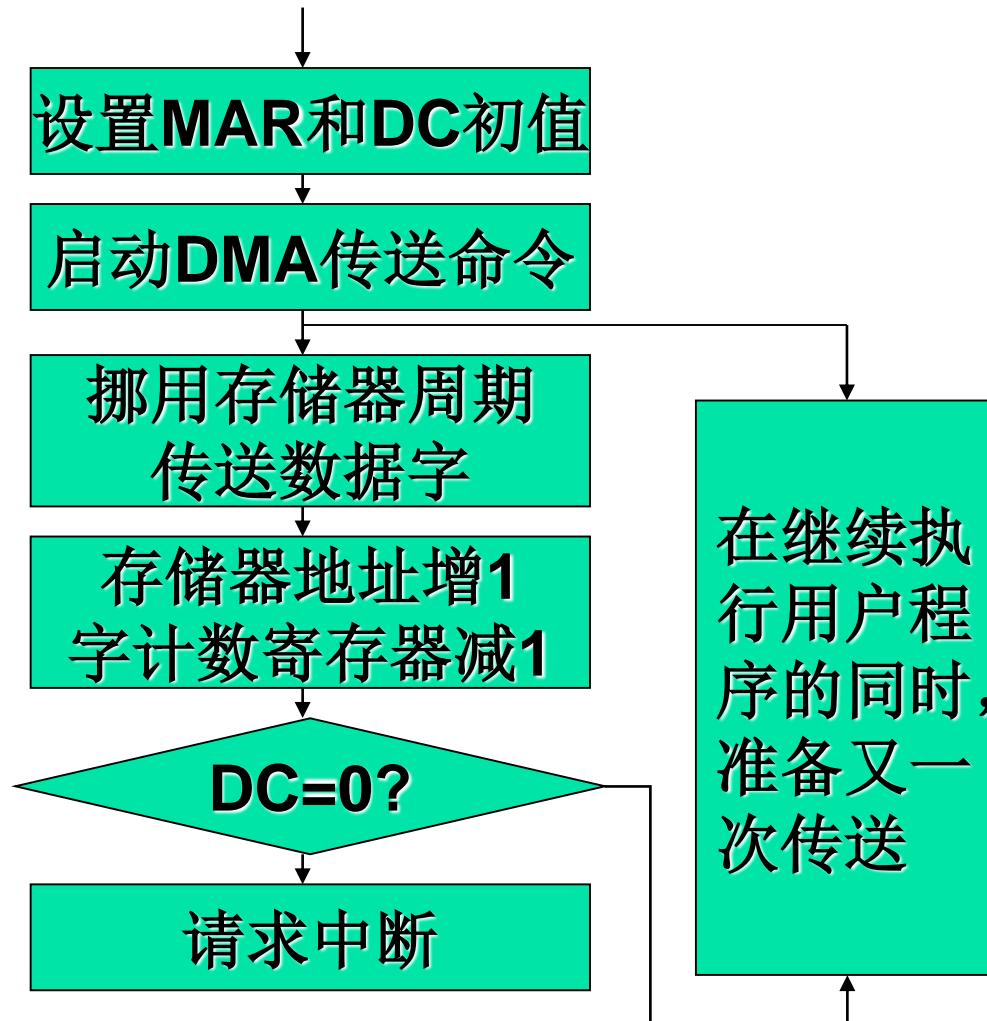
## ■ 该方式的特点是：

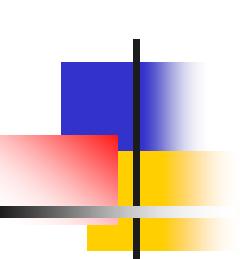
数据传输的基本单位是数据块；所传送的数据是从设备直接送入内存的，或者相反；仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在控制器的控制下完成的。可见DMA方式又是成百倍的减少了CPU对I/O的干预，进一步提高了CPU与I/O设备的并行操作程度。

# DMA方式示意图



# DMA工作过程





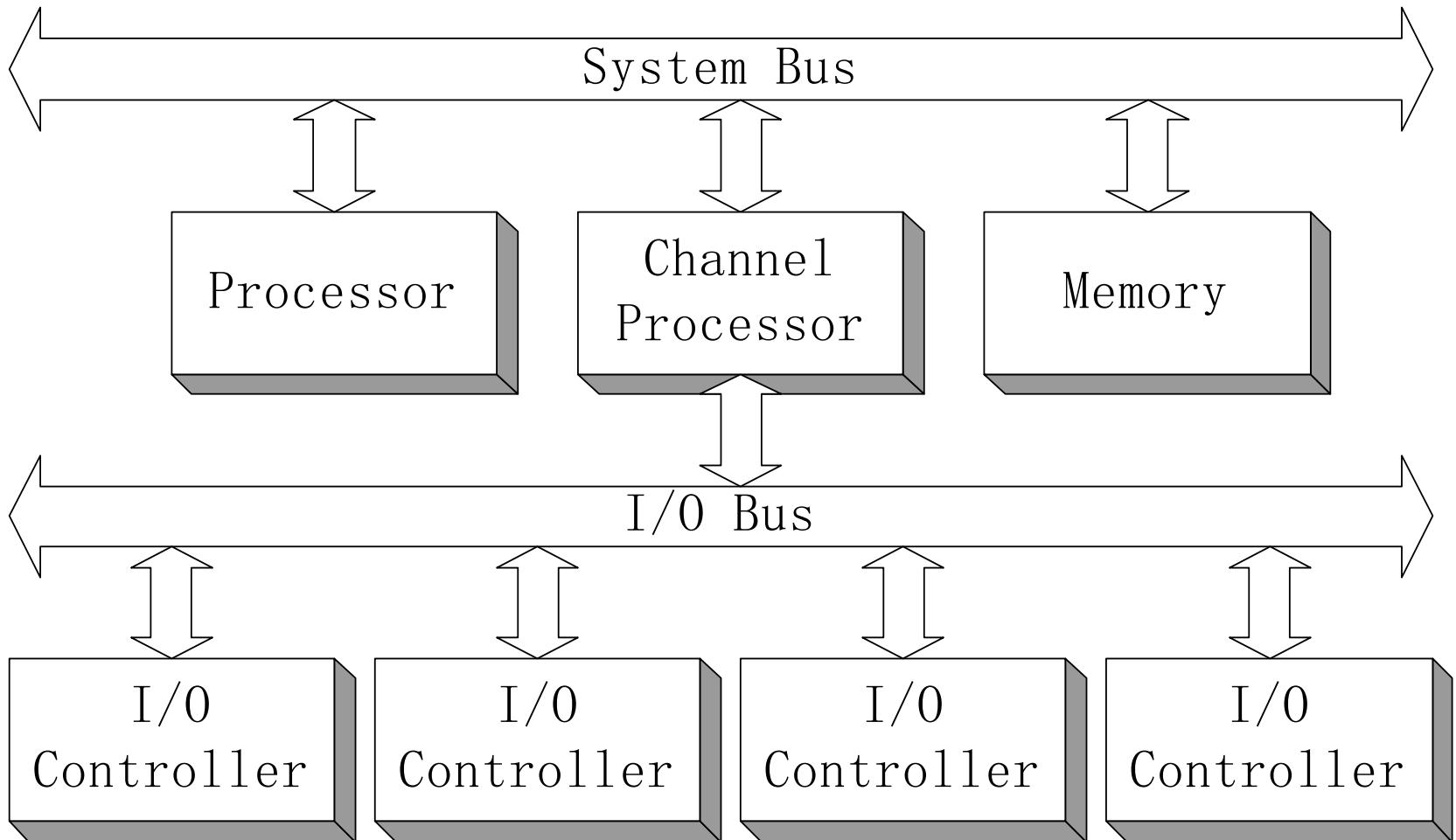
# 对 I/O 设备的控制方式

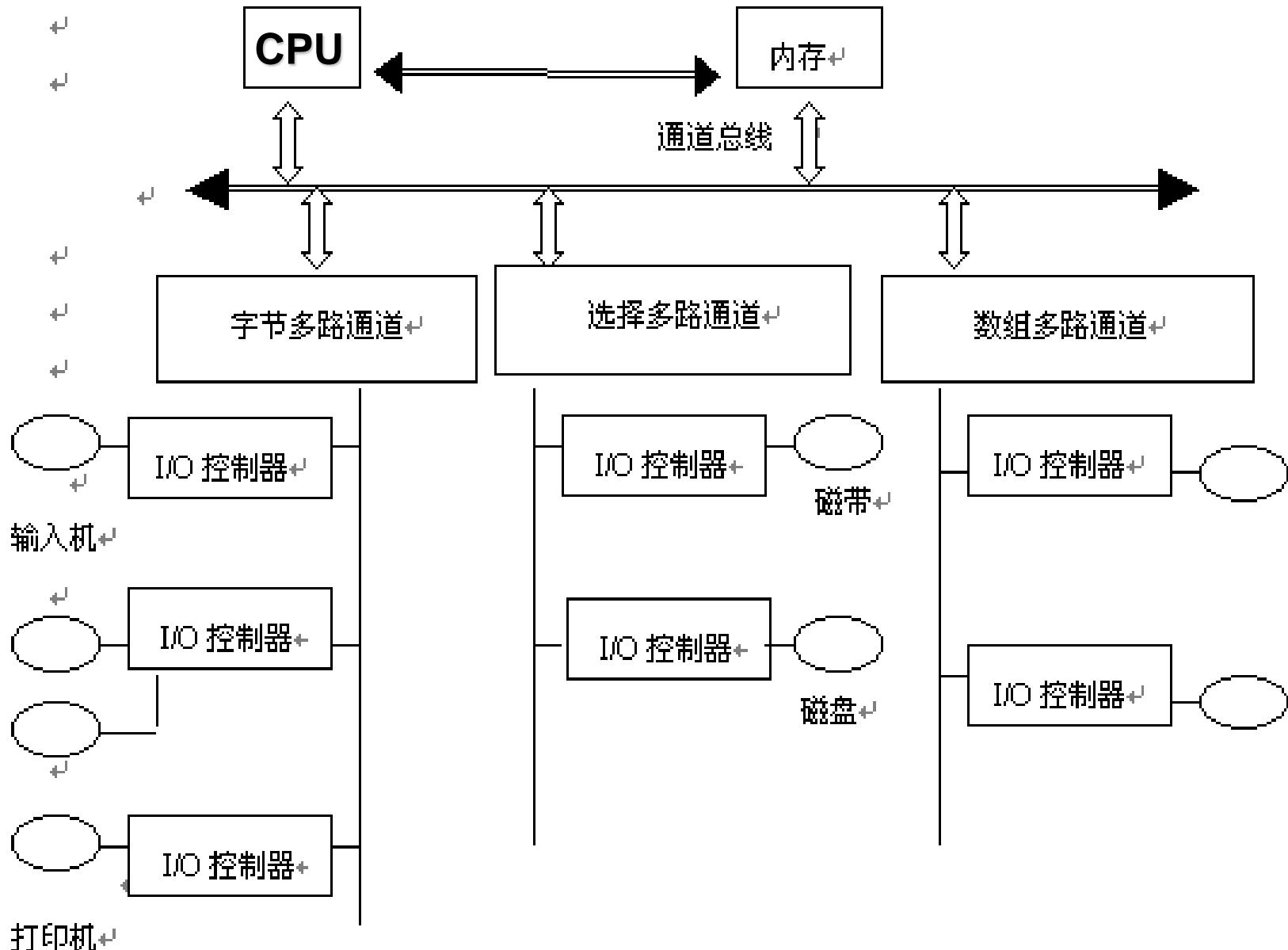
- 使用轮询的可编程I/O方式
- 使用中断的可编程I/O方式
- 直接存储器访问方式
- **I/O通道控制方式**

# I/O通道控制方式

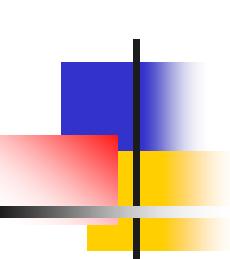
- I/O通道控制方式是**DMA**方式的发展，它可进一步减少**CPU**的干预，即把对一个数据块的读（写）为单位的干预，减少为对一组数据块的读（写）及有关的控制和管理为单位的干预。同时又可实现**CPU**、通道和**I/O**设备三者的并行操作，从而更有效的提高整个系统的资源利用率。
- 通道程序：通道是通过执行通道程序，并与设备控制器共同实现对**I/O**设备的控制的。通道程序由一系列通道指令所构成的。通道指令一般包含下列信息：
  - 操作码。规定指令所执行的操作。
  - 内存地址。
  - 计数。表示本指令所要操作的字节数。
  - 通道程序结束位。用以表示程序是否结束。
  - 记录结束标志。表示该指令是否与下条指令有关。

# I/O通道控制方式





通道方式的数据传送结构



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

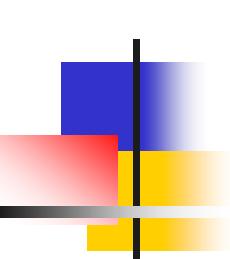
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**



## 6.5 与设备无关的I/O软件

为了方便用户和提高**OS**的可适应性与可扩展性，在现代**OS**的**I/O**系统中都无一例外地增加了与设备无关的**I/O**软件，以实现设备独立性，也称为设备无关性。

## 6.5.3 设备分配

在进行设备分配时，通常都要借助一些表格的帮助。在表格中记录了相应设备或控制器的状态及对设备或控制器进行控制所需的信息。

对于配备通道的计算机中，在进行设备分配时所需的数据结构有：设备控制表、控制器控制表、通道控制表和系统设备表等。

## 系统设备表

SDT

	设备名
	设备标识符
:	使用设备的进程
	DCT指针
:	驱动程序入口

## 设备控制表

DCT

DCB
DCB
:
DCB
:

## 设备控制块

DCB

设备类型
设备标识符
设备状态
COCT指针
重复执行计数
设备队列首指针
设备队列尾指针

## 控制器控制表

COCT

COCB	控制器标识符
COCB	控制器状态
:	CHCT指针
COCB	控制器队列首指针
:	控制器队列尾指针

## 控制器控制块

COCB

控制器标识符
控制器状态
CHCT指针
控制器队列首指针
控制器队列尾指针

## 通道控制表

CHCT

CHCB
CHCB
:
CHCB
:

## 通道控制块

CHCB

通道标识符
通道状态
通道队列首指针
通道队列尾指针
:

# 设备分配时应考虑的因素

## ■ 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为三种：独占性、共享性和虚拟性设备。**独占设备**在一段时间内只能由一个进程使用。**共享设备**允许多个进程共享。**虚拟设备**是经过某种处理由独占设备变为虚拟设备。

## ■ 设备分配算法

- **先来先服务：**根据请求的先后次序排成一个队列，设备总是分配给队首进程。
- **优先级高者优先：**利用该算法形成队列时，将优先权高的进程安排在设备队列前面，优先级相同的先来先服务。

## ■ 设备分配中的安全性

从进程运行的安全性上考虑，设备分配有以下两种方式。

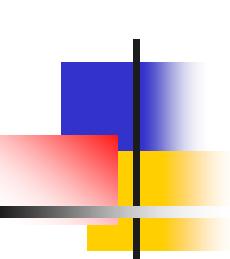
- **安全分配方式：**每当进程发出**I/O**请求后便阻塞，直到**I/O**完成后被唤醒。虽安全但缓慢。
- **不安全分配方式：**不断发出**I/O**请求，直到所请求的设备已经被另一进程占用才阻塞。虽迅速但不安全。

# 独占设备的分配程序

## 基本的设备分配程序

- 1) **分配设备**: 根据物理设备名在SDT中找出该设备的DCT, 若设备忙, 便将请求I/O的进程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, 若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插入设备队列。
- 2) **分配控制器**: 分配设备给进程后, 再到其DCT中找出与该设备连接的控制器的COCT。若控制器忙, 便将请求I/O进程的PCB挂在该控制器的等待队列上; 否则, 将该控制器分配给进程。
- 3) **分配通道**: 分配控制器后, 再在COCT中找到与该控制器连接的CHCT。若通道忙, 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; 之后便可启动该I/O设备进行数据传送。



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**

# 假脱机（SPOOLing）系统

如前所述，虚拟性是OS的四大特征之一。如果说通过多道程序技术将一台物理CPU虚拟为多台逻辑CPU，从而允许多个用户共享一台主机，那么，通过SPOOLing技术便可将一台物理I/O设备虚拟为多台逻辑I/O设备，同样允许多个用户共享一台物理I/O设备。

## ■ 什么是SPOOLing

为缓和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，将低速设备上的数据传送到高速磁盘上；或者相反。这样就可以在主机的直接控制下实现脱机输入输出。此时外围操作与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（Simultaneous Peripheral Operating On-Line），或称为假脱机操作。

# 假脱机 (SPOOLing) 系统

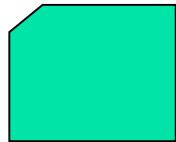
## SPOOLing系统的组成

主要有四大部分：

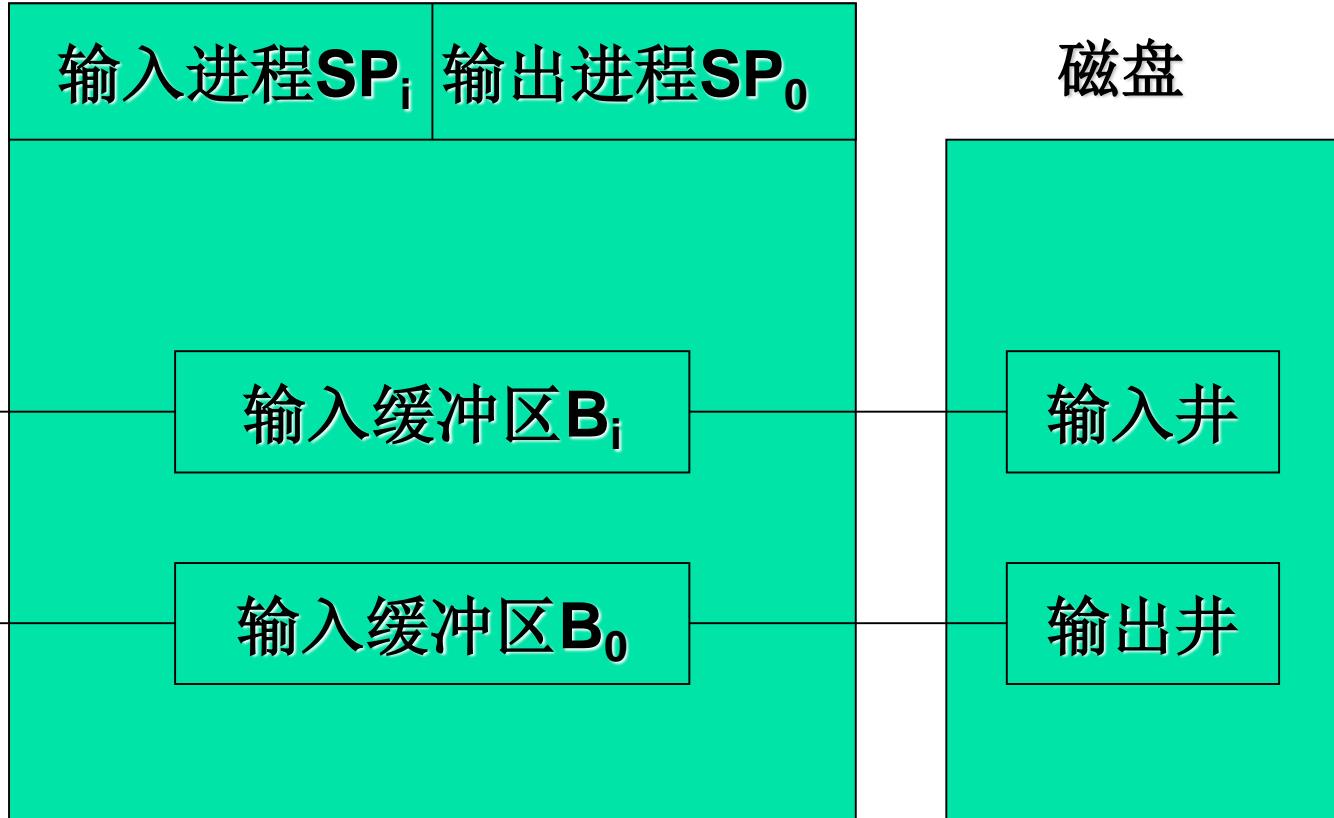
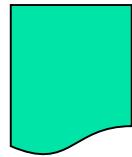
- **输入井和输出井。**是磁盘上开辟的两个大存储空间。输入井模拟脱机输入的磁盘设备，输出井模拟脱机输出时的磁盘。
- **输入缓冲区和输出缓冲区。**在内存中开辟两个缓冲区，输入缓冲区暂存由输入设备送来的数据，后送输入井；输出缓冲区暂存从输出井送来的数据，后送输出设备。
- **输入进程和输出进程。**利用两个进程模拟脱机I/O时的外围处理机。
- **井管理程序。**用于控制作业与磁盘井之间信息的交换。

# 假脱机 (SPOOLing) 系统

输入设备



输出设备

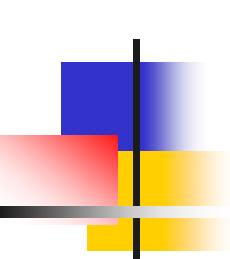


进程 **SP<sub>i</sub>** 模拟脱机输入时的外围控制机，将数据从输入机，通过输入缓冲区再送到输入井。**CPU**直接从输入井读数据入内存。进程 **SP<sub>0</sub>** 模拟脱机输出时的外围控制机，把输出的数据从内存送到输出井，再待输出设备空闲时，将输出井中的数据，经过输出缓冲区送到输出设备上。

# 假脱机（SPOOLing）系统

## SPOOLing系统的特点

- 提高了**I/O**的速度。利用输入输出井模拟成脱机输入输出，缓和了**CPU**和**I/O**设备速度不匹配的矛盾。
- 将独占设备改造为共享设备。并没有为进程分配设备，而是为进程分配一存储区和建立一张**I/O**请求表。
- 实现了虚拟设备功能。多个进程同时使用一台独占设备，虚拟成了多台设备。



# 第六章 输入输出系统

**6.1 I/O系统的功能、模型和接口**

**6.2 I/O设备和设备控制器**

**6.3 中断机构和中断处理程序**

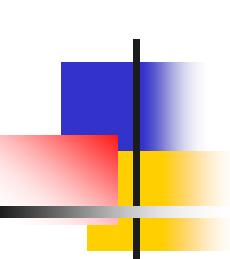
**6.4 设备驱动程序**

**6.5 与设备无关的I/O软件**

**6.6 用户层的I/O软件**

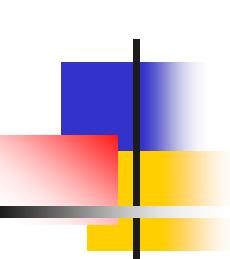
**6.7 缓冲区管理**

**6.8 磁盘存储器的性能和调度**



## 6.7 缓冲管理

为了缓和**CPU**和**I/O**设备速度不匹配的矛盾，提高**CPU**和**I/O**设备的并行性，在现代**OS**中，几乎所有的**I/O**设备与处理机交换数据时，都用了缓冲区。



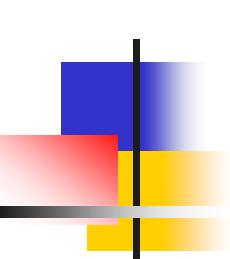
## 6.7 缓冲管理

**6.7.1 缓冲的引入**

**6.7.2 单缓冲区与多缓冲区**

**6.7.3 环形缓冲区**

**6.7.4 缓冲池（Buffer Pool）**



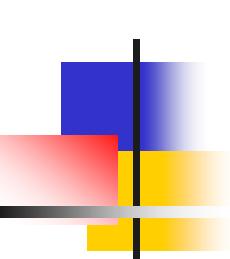
## 6.7 缓冲管理

**6.7.1 缓冲的引入**

**6.7.2 单缓冲区与多缓冲区**

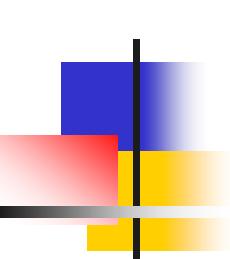
**6.7.3 环形缓冲区**

**6.7.4 缓冲池（Buffer Pool）**



## 6.7.1 缓冲的引入

- 引入缓冲区的主要原因归结为以下几点：
  - 缓和**CPU**与**I/O**设备间速度不匹配的矛盾。
  - 减少对**CPU**的中断频率，放宽对**CPU**中断响应时间的限制
  - 提高**CPU**和**I/O**设备之间的并行性。



## 6.7 缓冲管理

**6.7.1 缓冲的引入**

**6.7.2 单缓冲区与多缓冲区**

**6.7.3 环形缓冲区**

**6.7.4 缓冲池（Buffer Pool）**

## 6.7.2 单缓冲与多缓冲

### ■ 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，OS便在主存中为之分配一缓冲区。
- 在字符设备输入时，缓冲区用于暂存用户输入的一行数据，在输入期间，用户进程被挂起以等待数据输入完毕；在输出时，用户进程将一行数据输入到缓冲区后，继续执行处理。当用户进程已有第二行数据输出时，如果第一行数据尚未被提取完毕，则此时用户进程应阻塞。

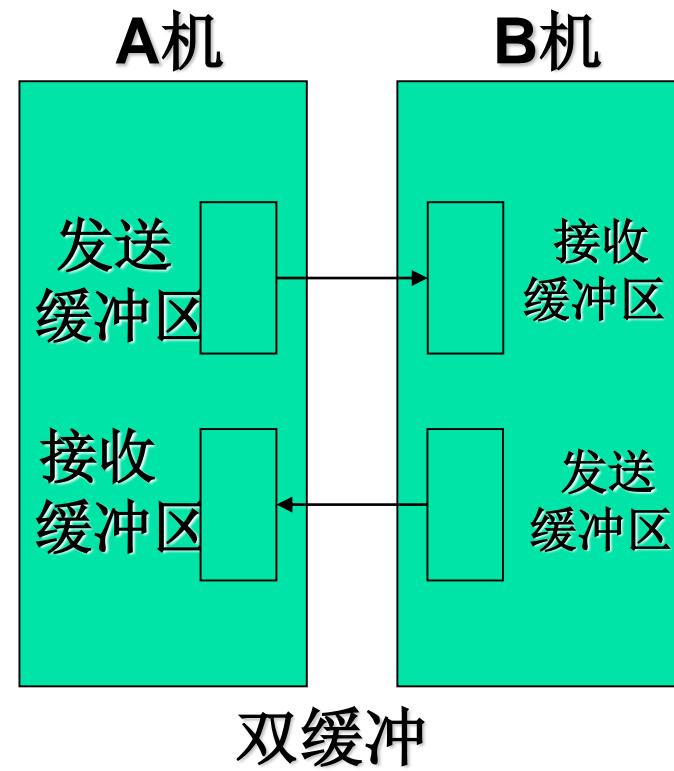
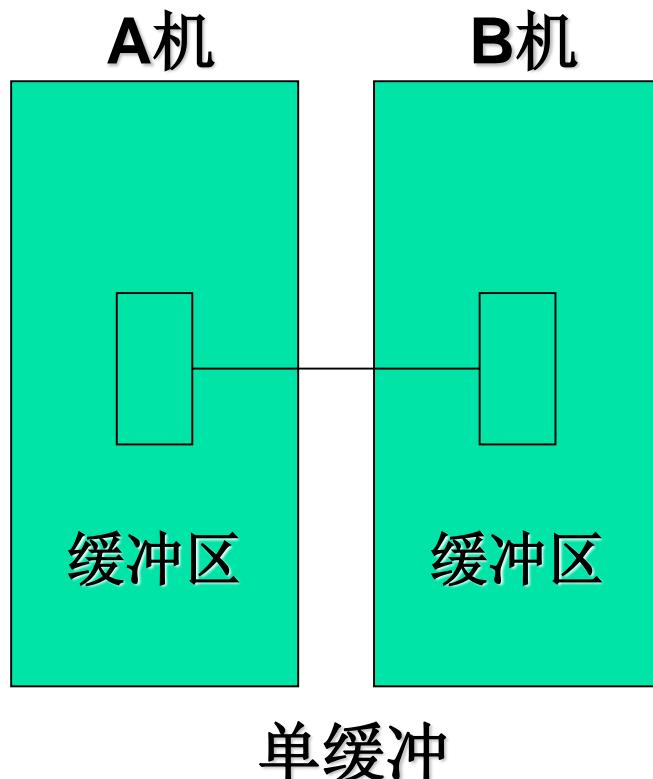
### ■ 双缓冲 (Double Buffer)

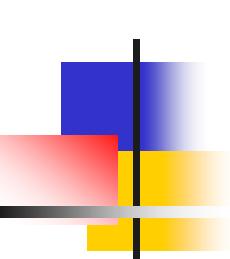
- 为了加快输入和输出速度，提高设备利用率，人们又引入了双缓冲区机制，也称为缓冲对换 (Buffer Swapping)。在设备输入时，先将数据送入第一缓冲区，装满后便转向第二缓冲区。此时OS可以从第一缓冲区中移出数据，并送入用户进程。接着由CPU对数据进行计算。

## 6.7.2 单缓冲与多缓冲

### 双机通讯时缓冲区的设置

若我们在实现两台机器之间的通信时，仅为它们配置了单缓冲，那么它们之间任意时刻都只能实现单方向的数据传输，而绝不允许双方同时向对方发送数据。为了实现双向数据传输，必须在两台机器中都设置两个缓冲区，一个用作发送缓冲区，另一个用作接受缓冲区。





## **6.7 缓冲管理**

**6.7.1 缓冲的引入**

**6.7.2 单缓冲区与多缓冲区**

**6.7.3 环形缓冲区**

**6.7.4 缓冲池（Buffer Pool）**

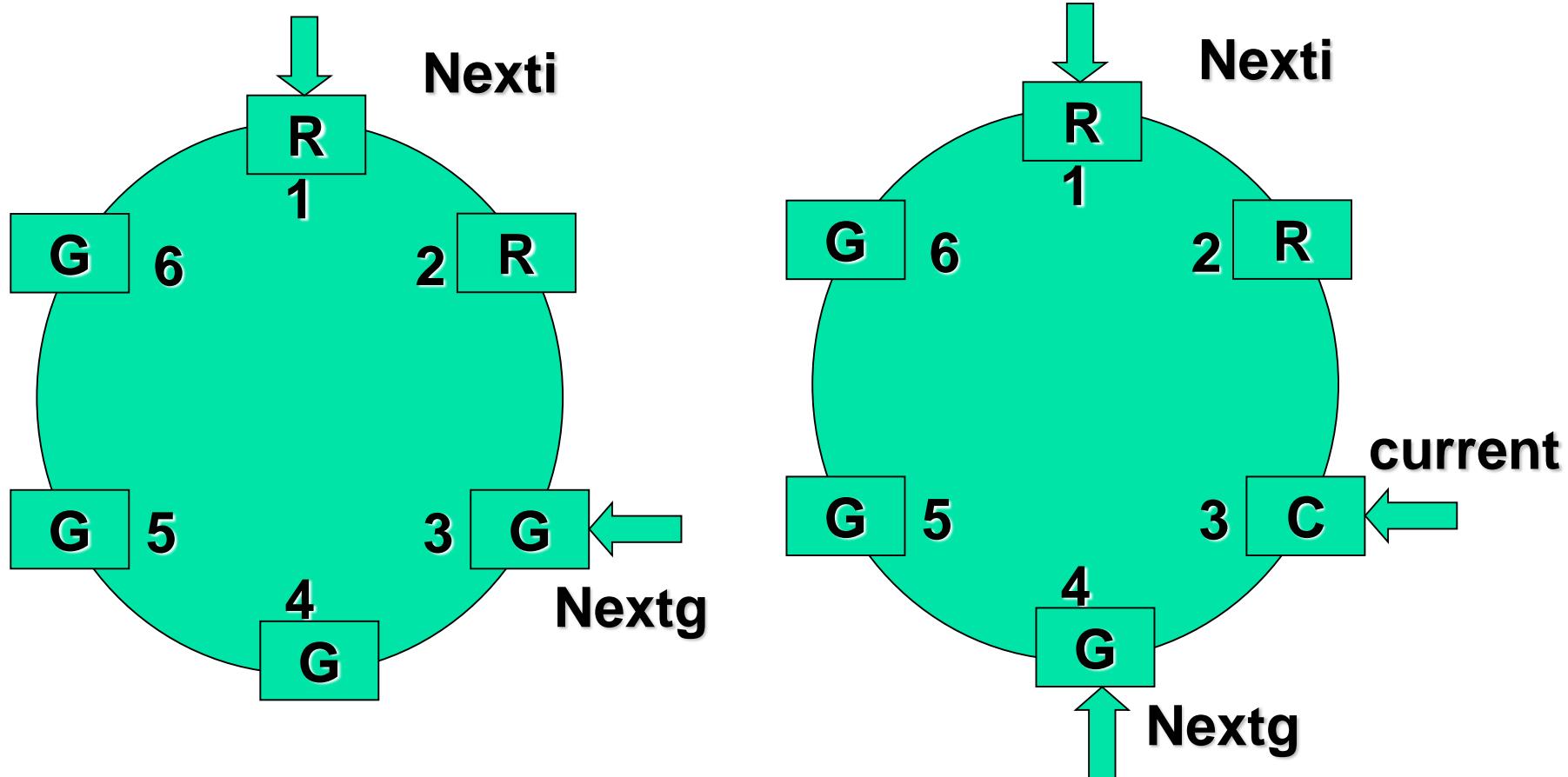
## 6.7.3 环形缓冲区

当输入与输出或生产者与消费者的速度基本相匹配时，采用双缓冲能获得较好的效果，可使生产者和消费者基本上能并行操作。但若两者的速度相差甚远，双缓冲的效果不够理想，但随着缓冲区数量的增加，情况有所改善。因此，又引入了多缓冲机制。可以将缓冲区组织成循环缓冲形式。

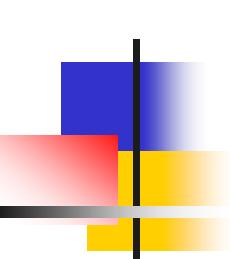
### ■ 循环缓冲的组成

- 多个缓冲区。循环缓冲有多个大小相同的缓冲区，作为输入的缓冲区有三种类型：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**。
- 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**。如下图。

# 环形缓冲区的组成示意图



**R:** 空缓冲; **G:** 已装满数的缓冲; **C:** 正在使用的缓冲



## 6.7 缓冲管理

**6.7.1 缓冲的引入**

**6.7.2 单缓冲区与多缓冲区**

**6.7.3 环形缓冲区**

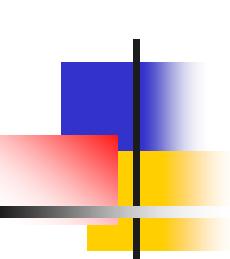
**6.7.4 缓冲池（Buffer Pool）**

## 6.7.4 缓冲池 (Buffer Pool)

上述的缓冲区仅适用于某特定的**I/O**进程和计算进程，因而它们属于专用缓冲。当系统较大时，将会有许多这样的循环缓冲，这不仅消耗大量内存空间，而且利用率不高。为提高缓冲区的利用率，目前广泛流行缓冲池，在池中设置了多个可供若干个进程共享的缓冲区。

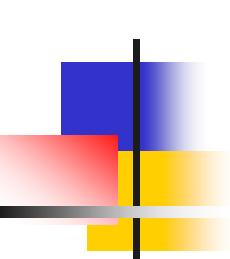
### 缓冲池的组成

- 对于既可输入又可输出的公用缓冲池，至少应含有下列三种类型的缓冲区：
  - 空缓冲区；
  - 装满输入数据的缓冲区；
  - 装满输出数据的缓冲区；



# 第六章 输入输出系统

- 6.1 I/O系统的功能、模型和接口**
- 6.2 I/O设备和设备控制器**
- 6.3 中断机构和中断处理程序**
- 6.4 设备驱动程序**
- 6.5 与设备无关的I/O软件**
- 6.6 用户层的I/O软件**
- 6.7 缓冲区管理**
- 6.8 磁盘存储器的性能和调度**

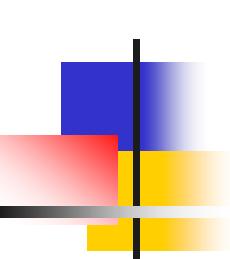


# **6.8 磁盘存储器的性能和调度**

**6.8.1 磁盘性能简述**

**6.8.2 早期的磁盘调度算法**

**6.8.3 基于扫描的磁盘调度算法**



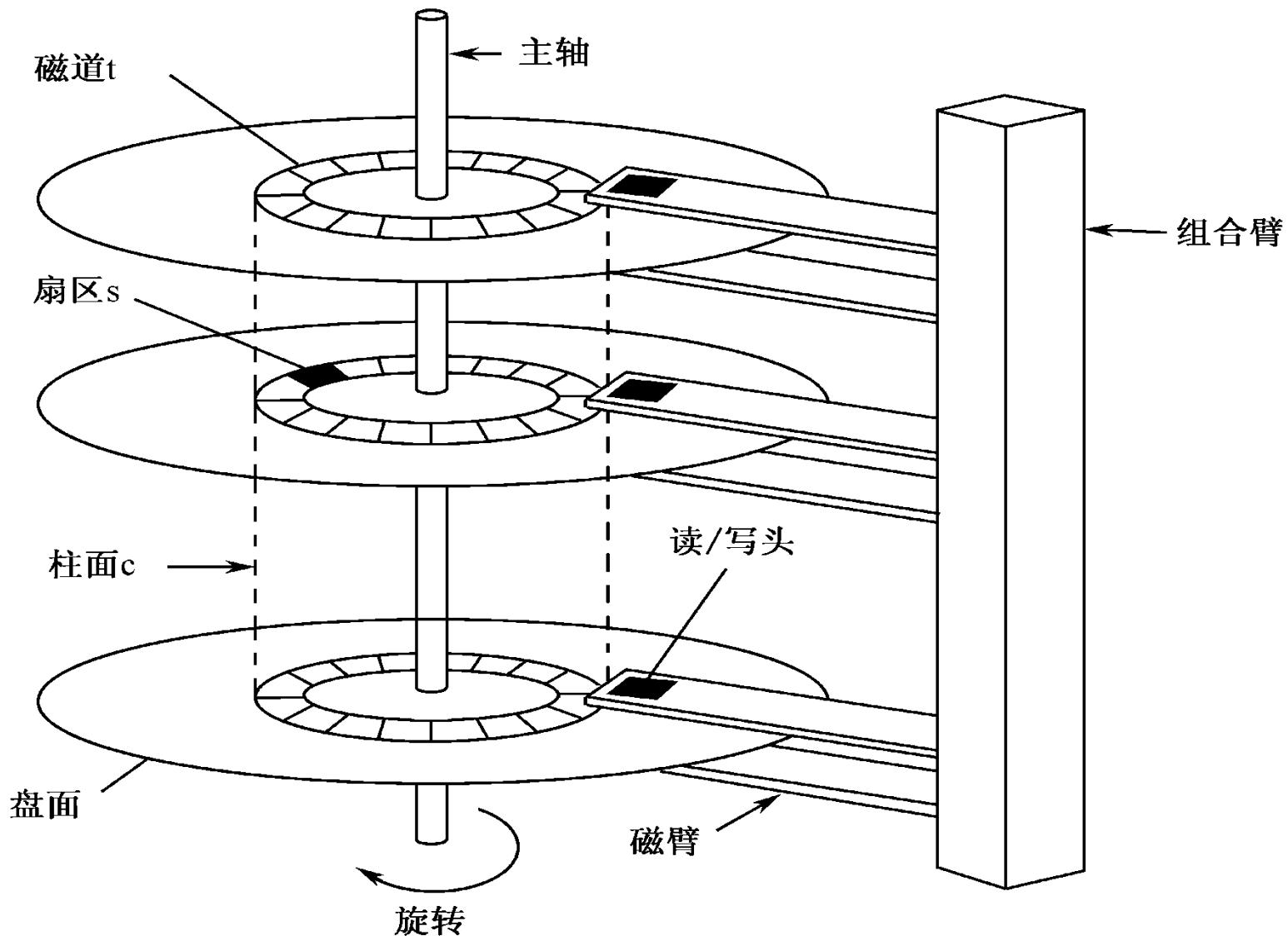
# **6.8 磁盘存储器的性能和调度**

## **6.8.1 磁盘性能简述**

## **6.8.2 早期的磁盘调度算法**

## **6.8.3 基于扫描的磁盘调度算法**

## 6.8.1 磁盘性能简述



## 6.8.1 磁盘性能简述

### ■ 数据的组织和格式

磁盘包含一或多个盘片，每片分两面，每面又可分成若干条磁道，磁道之间留有必要的空隙。

为简单起见，在每条磁道上存储相同数目的二进制位。因而，内层磁道的存储密度（每英寸所存储的位数）较外层磁道的密度高。每条磁道又分成若干个扇区，每个扇区的大小相当于一个盘块，各扇区之间保留一定的间隙。

在磁盘存储数据前要格式化磁盘。

编址方式为： 盘面号， 磁道号， 扇区号。  
(均从**0**开始编址)

# 6.8.1 磁盘性能简述

## ■ 磁盘的类型

磁盘可以从不同的角度进行分类：硬盘和软盘、单片盘和多片盘、固定头磁盘和活动头磁盘等。

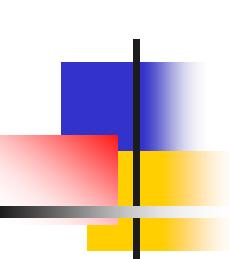
- **固定头磁盘：**每条磁道都有一个读/写磁头，可对磁道并行读/写，I/O速度快，适用于大容量磁盘。
- **移动头磁盘：**每个盘面一个磁头，该磁头能移动以进行寻道。只能进行串行读/写，I/O速度较慢，但结构简单，曾经广泛用于中、小型磁盘设备中。

## ■ 磁盘的访问时间T

$$T = T_s + T_r + T_t$$

- **寻道时间** $T_s$ ：把磁臂从当前位置移到指定磁道上所经历的时间
- **旋转延迟时间** $T_r$ ：指定扇区移动到磁头下面所经历的时间。
- **传输时间** $T_t$ ：数据从磁盘读出或向磁盘写入数据所经历的时间

在访问时间中，**寻道时间和旋转延迟时间**，基本上与所读/写数据的多少无关，通常是占据访问时间的大头。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 磁盘存储器的性能和调度**

## **6.8.1 磁盘性能简述**

## **6.8.2 早期的磁盘调度算法**

## **6.8.3 基于扫描的磁盘调度算法**

## 6.8.2 早期的磁盘调度算法

当有多个进程都请求访问磁盘时，应使各进程对磁盘的平均访问时间（主要是寻道）最小。因此，磁盘调度的目标应是使磁盘的平均寻道时间最少。目前常用的磁盘调度算法有：

### ■ 先来先服务（FCFS）

根据进程请求访问磁盘的先后次序进行调度。优点：公平、简单，且每个进程的请求都能依次得到处理，不会出现某一进程的请求长期得不到满足的情况。

**缺点：**未对寻道进行优化，致使平均寻道时间可能较长。仅适用于请求磁盘I/O的进程数目较少的场合。

### ■ 最短寻道时间优先（SSTF）

算法选择要求访问的磁道与当前磁头所在的磁道距离最近的进程，以使每次的寻道时间最短。

**存在的问题：**可能导致某些进程发生“饥饿”。因为只要不断有所要访问的磁道与磁头当前所在磁道的距离较近的新进程到达，就会出现“老进程饥饿”现象。这种调度算法不能保证平均寻道时间最短。

## ■ 先来先服务

### 调度算法之例

9个进程先后提出读盘请求，访问的磁道号为： 55; 58; 39; 18; 90; 160; 150; 38; 184。目前磁头停留在100道。此时开始磁盘调度；其调度序列为：

从**100**号磁道开始

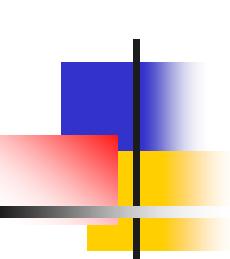
被访问的下一个磁道号	移动距离（磁道数）
<b>55</b>	<b>45</b>
<b>58</b>	<b>3</b>
<b>39</b>	<b>19</b>
<b>18</b>	<b>21</b>
<b>90</b>	<b>72</b>
<b>160</b>	<b>70</b>
<b>150</b>	<b>10</b>
<b>38</b>	<b>112</b>
<b>184</b>	<b>146</b>
平均寻道长度： <b>55.3</b>	

## ■ 最短寻道时间优先调度算法之例

9个进程先后提出读盘请求，访问的磁道号为：55; 58; 39; 18; 90; 160; 150; 38; 184。目前磁头停留在100道。此时开始磁盘调度；其调度序列为：

### 从**100**号磁道开始

被访问的下一个磁道号	移动距离（磁道数）
<b>90</b>	<b>10</b>
<b>58</b>	<b>32</b>
<b>55</b>	<b>3</b>
<b>39</b>	<b>16</b>
<b>38</b>	<b>1</b>
<b>18</b>	<b>20</b>
<b>150</b>	<b>132</b>
<b>160</b>	<b>10</b>
<b>184</b>	<b>24</b>
平均寻道长度： <b>27.5</b>	



# **6.8 磁盘存储器的性能和调度**

## **6.8.1 磁盘性能简述**

## **6.8.2 早期的磁盘调度算法**

## **6.8.3 基于扫描的磁盘调度算法**

## 6.8.3 基于扫描的磁盘调度算法

SCAN算法不仅考虑欲访问的磁道与当前磁道的距离，更优先考虑磁头的当前移动方向。在磁头正在自里向外移动时，所选择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，自外向里移动。这时，每次选择要访问的磁道，在当前磁道之内且距离最近者这样的进程来调度。

SCAN算法中磁头移动的规律似电梯的运行，又称为电梯调度算法。算法既能获得较好的寻道性能，又能防止进程饥饿，被广泛用于大、中、小型机和网络中的磁盘调度。

存在的问题：当磁头刚从里向外移动过某一磁道时，恰有一进程请求访问此磁道，这时该进程必须等待，待磁头从里向外，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严重地推迟。

## ■ 扫描 (SCAN)

### 算法之例

9个进程先后提出读盘请求，访问的磁道号为：55; 58; 39; 18; 90; 160; 150; 38; 184。目前磁头停留在100道。此时开始磁盘调度；其调度序列为：

从 <b>100</b> 号磁道开始，向磁道号增加方向	
被访问的下一个磁道号	移动距离（磁道数）
<b>150</b>	<b>50</b>
<b>160</b>	<b>10</b>
<b>184</b>	<b>24</b>
<b>90</b>	<b>94</b>
<b>58</b>	<b>32</b>
<b>55</b>	<b>3</b>
<b>39</b>	<b>16</b>
<b>38</b>	<b>1</b>
<b>18</b>	<b>20</b>
平均寻道长度： <b>27.8</b>	

## 6.8.3 基于扫描的磁盘调度算法

- 扫描（SCAN）算法
- 循环扫描CSCAN

为了减少请求进程的延迟，CSCAN算法规定磁头单向移动。若规定只自里向外移动，当磁头移到最外的被访问磁道时，磁头立即返回到最里的欲访磁道，即将最小磁道号紧接着最大磁道号构成循环，进行扫描。

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 $T+S_{max}$ ，其中， $T$ 为由里向外（或相反）扫描完所有要访问的磁道所需的寻道时间，而 $S_{max}$ 是将磁头从最外面被访问的磁道直接移到最里边欲访问的磁道所需的寻道时间。

## ■ 循环扫描CSCAN 算法之例

9个进程先后提出读盘请求，访问的磁道号为： 55; 58; 39; 18; 90; 160; 150; 38; 184。目前磁头停留在100道。此时开始磁盘调度；其调度序列为：

从 <b>100</b> 号磁道开始，向磁道号增加方向	
被访问的下一个磁道号	移动距离（磁道数）
<b>150</b>	<b>50</b>
<b>160</b>	<b>10</b>
<b>184</b>	<b>24</b>
<b>18</b>	<b>166</b>
<b>38</b>	<b>20</b>
<b>39</b>	<b>1</b>
<b>55</b>	<b>16</b>
<b>58</b>	<b>3</b>
<b>90</b>	<b>32</b>
平均寻道长度： <b>35.8</b>	

## 6.8.3 基于扫描的磁盘调度算法

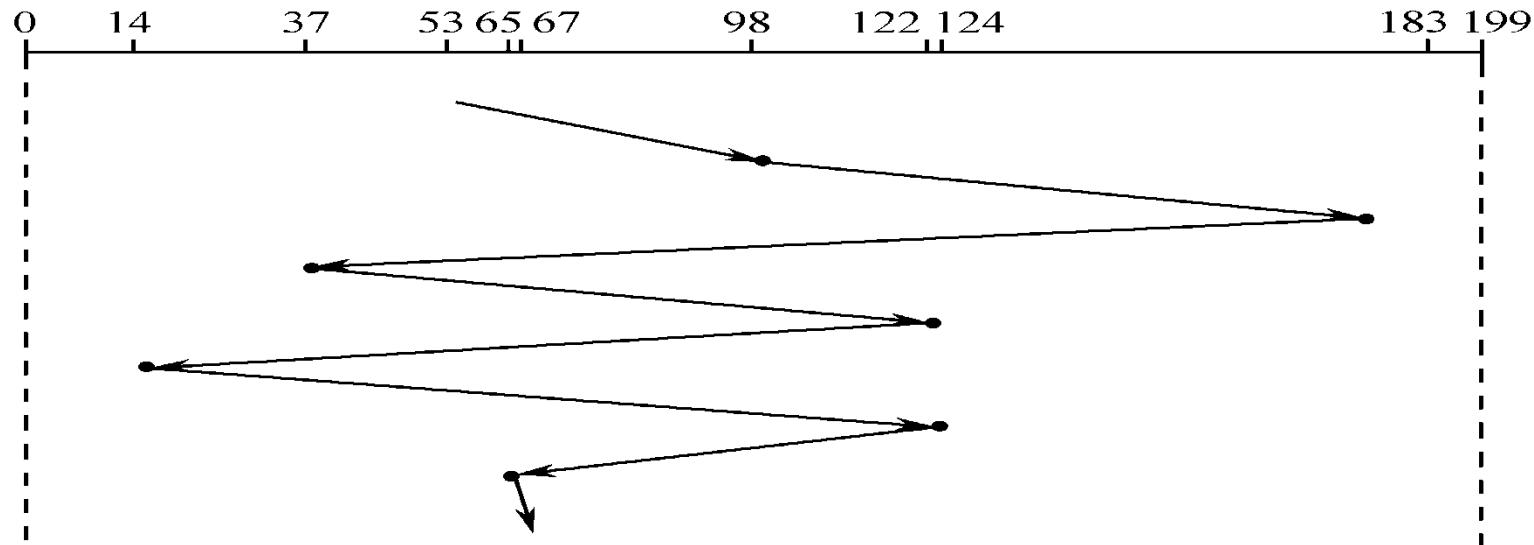
- 扫描(SCAN)算法 ■ 循环扫描CSCAN
- N-Step-SCAN和FSCAN调度算法
- **N-Step-SCAN算法：**SSTF、SCAN、CSCAN几种调度算法都可能出现磁臂停留在某处不动的情况，称为磁臂粘着。在高密度盘上更容易出现此情况。N-Step-SCAN算法将磁盘请求队列分成若干个长度为N的子队列。磁盘调度将按FCFS算法依次处理这些子队列，而每处理一个队列时，又是按SCAN算法。这样就可避免出现粘着现象。N值取得很大时，其性能接近SCAN算法；N=1时，则退化为FCFS算法。
- **FSCAN算法：**本算法是N-Step-SCAN算法的简化。它只将磁盘请求访问队列分成两个子队列。一是当前所有请求磁盘I/O的进程形成的队列，由磁盘调度按SCAN算法进行处理；另一个则是在扫描期间，新出现的所有请求磁盘I/O进程组成的等待处理的请求队列。从而使所有的新请求都将被推迟到下一次扫描时处理。

# 磁盘调度算法之例

## (1) 先来先服务:

按访问请求到达的先后次序服务。

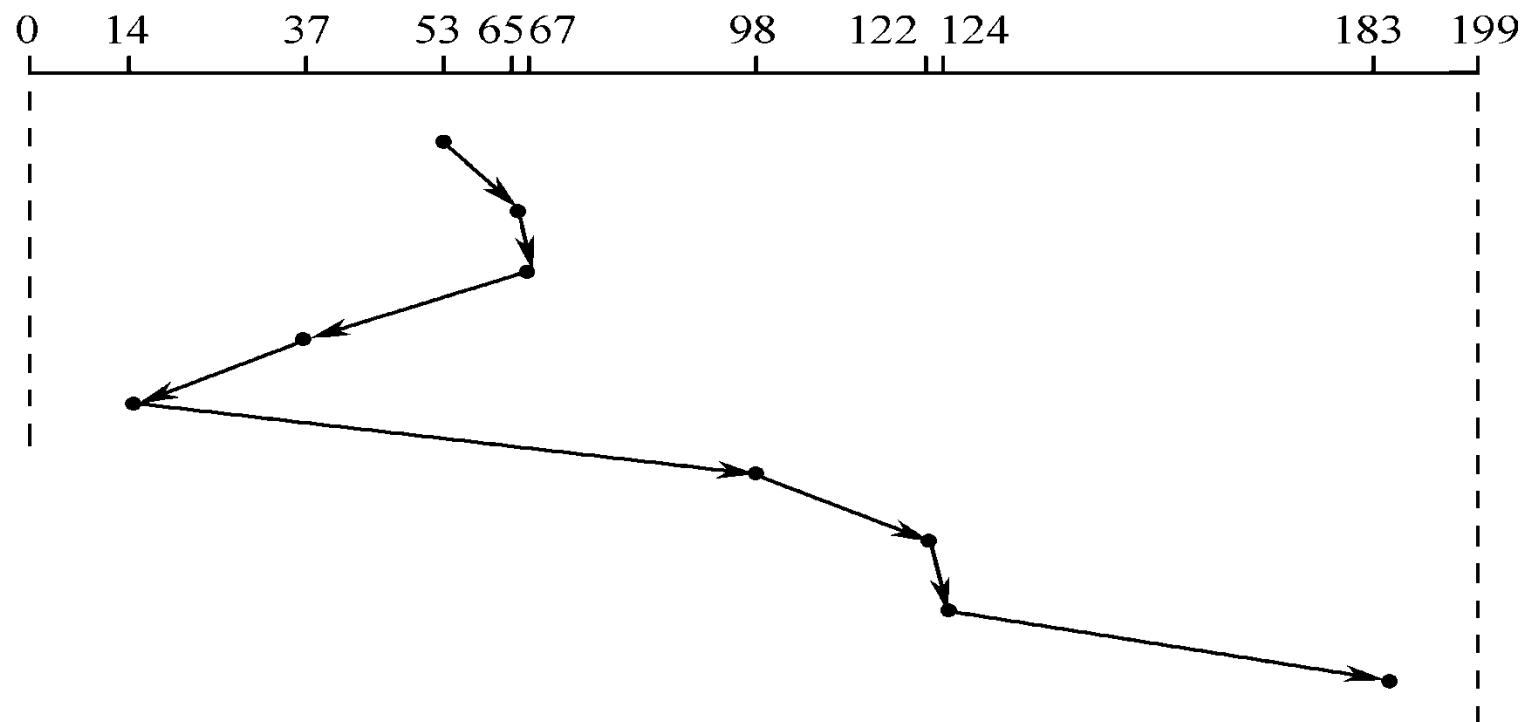
例：假设磁盘访问序列：98，183，37，122，14，124，65，67。读写头起始位置：53，安排磁头服务序列，计算磁头移动总距离（道数）。



## (2) 最短寻道时间优先

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）。

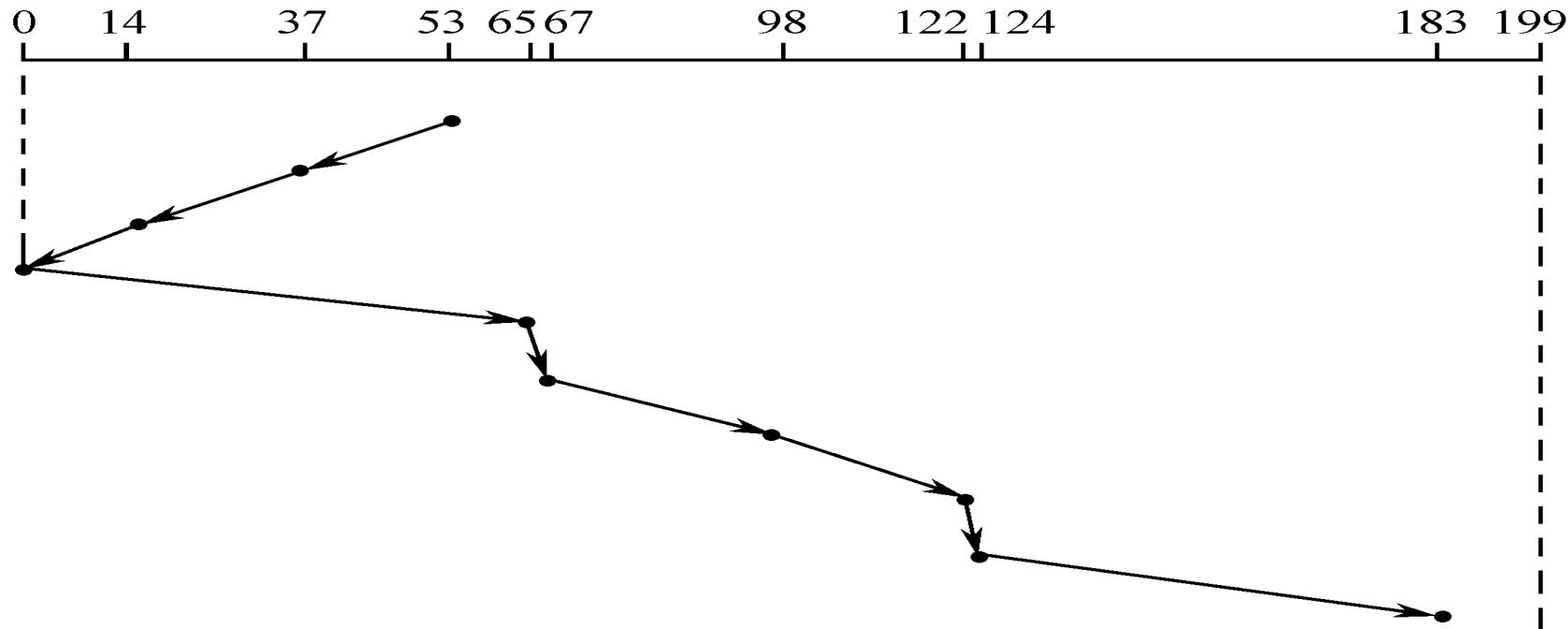
SSTF算法选择与当前磁头位置最近的请求作为下一个服务对象，即寻道时间最短的请求。



### (3) SCAN调度

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）。

磁臂从磁盘的一端向另一端移动，同时当磁头移过每个柱面时，处理位于该柱面上的服务请求。当到达另一端时，磁头改变移动方向，处理继续进行。磁头在整个磁盘上回来扫描。



## (4) C-SCAN调度算法

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）。

C-SCAN将磁头从磁盘一端移到磁盘的另一端，随着移动而不断地请求处理。不过，当磁头移到另一端时，它会马上返回到磁盘开始处，返回时并不处理任何请求。

