课程主要内容

- 操作系统引论（第1章）
- 进程管理（第2-3章）
- 存储管理（第4-5章）
- 设备管理（第6章）
- 文件管理（第7章）
- 操作系统接口（第9章）

设备管理

- I/O系统
- I/O控制方式
- 缓冲管理
- I/O软件
- 设备分配
- 磁盘存储器管理（第8章）

本章练习

6.1 I/O 系统

I/O 系统的组成： I/O设备、设备控制器、I/O通道、总线及相应软件

- I/O设备
- 设备控制器
- I/O通道
- I/O系统的总线系统
- I/O系统的结构

6.1.1 I/O 设备

1、 I/O设备的类型

1）按使用方式/共享属性分类

2）按传输速率分类

3）按信息交换的单位分类

1）按使用方式/共享属性分类 { 独享设备 / 共享设备 / 虚拟设备

独享/独占设备：在一段时间只允许一个用户进程访问的设备。多数低速设备属此类，打印机就是典型的独享设备。

共享设备：在一段时间允许多个用户进程同时访问的设备。磁盘就是典型的共享设备。

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，通常把这种经过虚拟技术处理后的设备称为虚拟设备。

2）按传输速率分类 { 低速设备 / 中速设备 / 高速设备

低速设备：传输速率仅为每秒钟几个字节至数百个字节的设备。典型的有：键盘、鼠标、语音的输入/输出等。

中速设备：传输速率为每秒钟数千个字节至数万个字节的设备。典型的有：打印机等。

高速设备：传输速率为每秒钟数百KB至数十MB的设备。典型的有：磁盘机、磁带机、光盘机等。

3）按信息交换的单位分类 { 块设备 / 字符设备

块设备：信息交换的基本单位为**字符块**，属于有结构设备，块大小一般为512B---4KB，典型的有：磁盘、磁带等。块设备的**基本特征**：传输速率较高（几MB/s）、可寻址、I/O常采用DMA方式。

字符设备：信息交换的基本单位为**字符**，典型的有：键盘、打印机和显示器等。**基本特征**：传输速率较低，几字节~数千B/s、不可寻址、I/O常采用中断驱动方式。

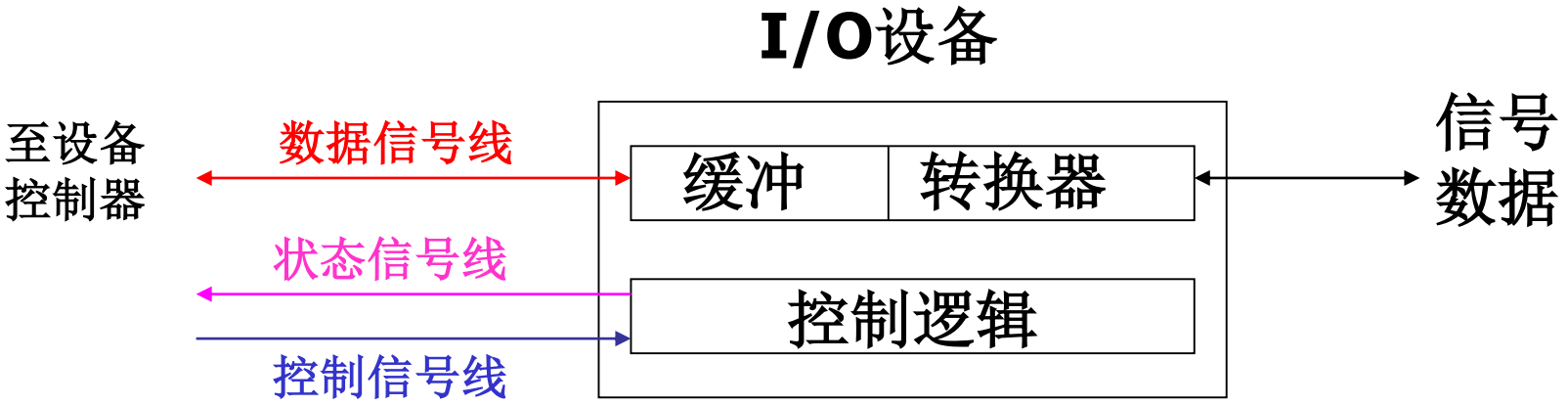
2、设备与控制器之间的接口 （见下图即P183 图6-3）

设备间接通过设备控制器与CPU进行通信，在设备与设备控制器之间有一接口，传递三类信号：

数据信号线：双向（输入输出），用于传送数据信号

控制信号线：控制器发给设备；要求设备完成相关操作

状态信号线：设备发给控制器，用于传送指示设备当前状态的信号



6.1.2 设备控制器

❖设备控制器

是处于CPU与I/O设备之间的接口，接收CPU发来的命令，并控制I /O设备工作，是一个可编址设备。

❖ 功能：接收CPU命令，控制I/O设备工作，解放CPU

1. 接收和识别命令：应有相应的Register来存放命令（设置多个“命令寄存器”和“命令译码器”）
2. 实现数据交换：CPU——控制器的数据寄存器——设备
3. 了解设备状态：设备控制器中应有“状态寄存器”

6.1.2 设备控制器

❖ **设备控制器**

是处于CPU与I/O设备之间的接口，接收CPU发来的命令，并控制I /O设备工作，是一个可编址设备。

❖ **功能：** 接收CPU命令，控制I/O设备工作，解放CPU

（续）

4. 识别设备地址：CPU通过“地址”与设备通信，设备控制器应能识别它所控制的设备地址以及其各寄存器的地址。应配置“地址译码器”。

5. 数据缓冲：在控制器中必须设置一缓冲器。

6. 差错控制：对I/O设备传送来的数据进行差错检测

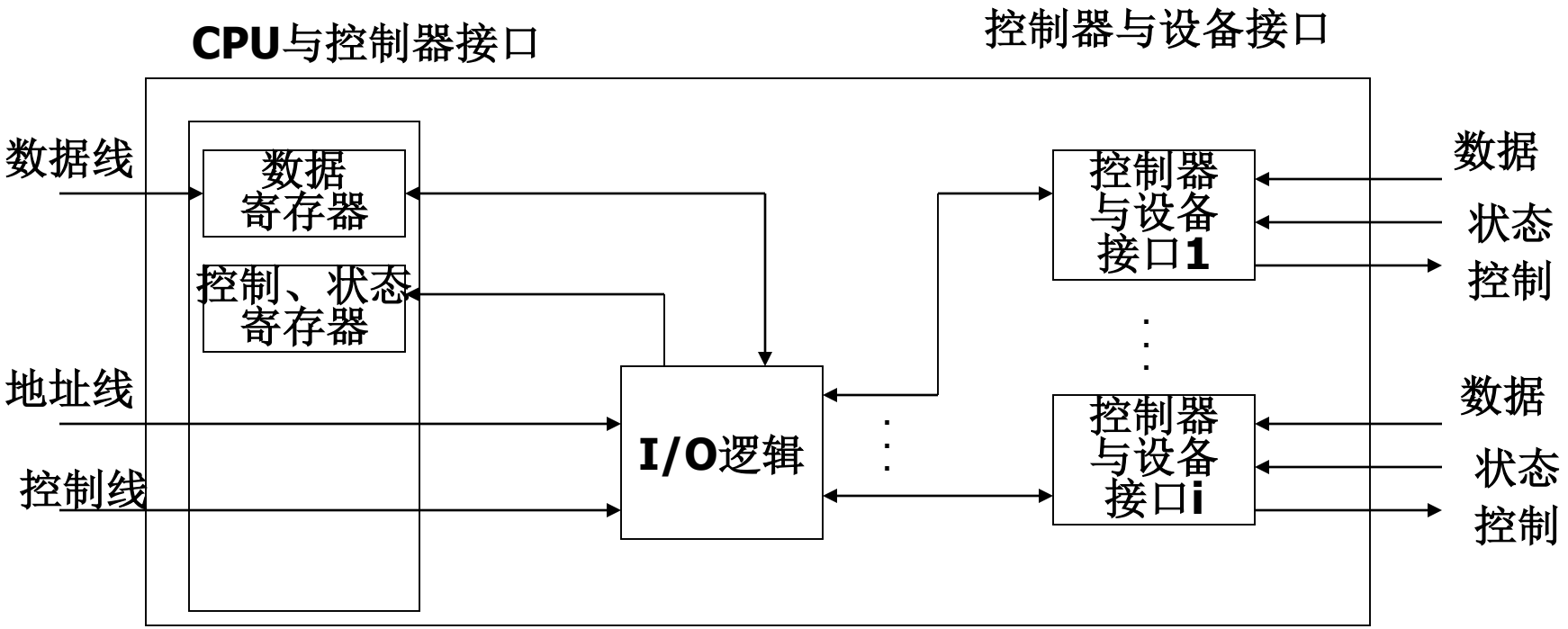
6.1.2 设备控制器

- 设备控制器的组成
  - 设备控制器与处理机的接口
  - 设备控制器与设备接口
  - I/O逻辑

寄存器：控制寄存器（存放命令及参数）、数据寄存器（存放数据）、状态寄存器（记录设备状态）。

见下页图

设备控制器的组成


CPU与控制器接口
控制器与设备接口
数据线
数据寄存器
控制、状态寄存器
地址线
控制线
I/O逻辑
控制器与设备接口1
控制器与设备接口i
数据
状态
控制
数据
状态
控制

6.1.3 I/O通道（1）

一、引入

在CPU和I/O设备之间增加了设备控制器减轻了CPU对数据输入输出的控制、使得CPU的效率得到显著的提高。

而通道的出现则进一步提高了CPU的效率。

这是因为通道是一个特殊功能的处理器，它有自己的指令和程序专门负责数据输入输出的传输控制。而CPU将“传输控制”的功能下放给通道后只负责“数据处理”功能。这样，通道与CPU分时使用内存，实现了CPU内部运算与I/O设备的并行工作

6.1.3 I/O 通道（2）

通道：一种特殊的执行I/O指令的**处理机**，指令类型单一，没有自己的内存，与CPU共享内存。

引入目的：建立独立的I/O操作，解脱CPU对I/O的组织、管理。

CPU只需发送I/O命令给通道，通道通过调用内存中的相应通道程序完成任务。

通道的基本功能：执行通道指令，组织外围设备和内存进行数据传输，按I/O指令要求启动外围设备，向CPU报告中断等，具体有以下五项任务：

6.1.3 I/O 通道（3）

(1)接受CPU的I/O指令，按指令要求与指定的外围设备进行通信。

(2)从内存选取属于该通道程序的通道指令，经译码后向设备控制器和设备发送各种命令。

(3)组织外围设备和内存之间进行数据传送，并根据需要提供数据缓存的空间，以及提供数据存入内存的地址和传送的数据量。

6.1.3 I/O 通道（4）

(4)从外围设备得到设备的状态信息，形成并保存通道本身的状态信息，根据要求将这些状态信息送到内存的指定单元，供CPU使用。

(5)将外围设备的中断请求和通道本身的中断请求，按次序及时报告给CPU。

6.1.3 I/O 通道（5）

二、CPU对通道的管理

CPU通过执行I/O指令以及处理来自通道的中断，实现对通道的管理。来自通道的中断有两种，一种是数据传送结束中断，另一种是故障中断。

管态: CPU运行操作系统的管理程序的状态。

目态 :CPU执行用户程序时的状态。

大中型计算机的I/O指令都是管态指令，只有当CPU处于管态时，才能运行I/O指令，目态时不能运行I/O指令。这是因为大中型计算机的软、硬件资源为多个用户所共享，而不是分给某个用户专用。

6.1.3 I/O 通道（6）

三、通道对设备控制器的管理

通道通过使用通道指令控制设备控制器进行数据传送操作，并以通道状态字接收设备控制器反映的外围设备的状态。

因此，设备控制器是通道对I/O设备实现传输控制的执行机构。

设备控制器的具体任务如下：

6.1.3 I/O 通道（7）

(1)从通道接受通道指令，控制外围设备完成所要求的操作；

(2)向通道反映外围设备的状态；

(3)将各种外围设备的不同信号转换成通道能够识别的标准信号。

6.1.3 I/O 通道（8）

四、通道的类型

根据信息交换方式的不同，通道可分成以下几种类型：

- 字节多路通道
- 数组选择通道
- 数组多路通道

注：“瓶颈”问题

字节多路通道（1）

- 工作原理:字节多路通道主要用于连接大量的低速设备，如键盘、打印机等等。例如数据传输率是1000B/s，即传送1个字节的间隔是1ms，而通道从设备接收或发送一个字节只需要几百纳秒，因此通道在传送两个字节之间有很多空闲时间，字节多路通道正是利用这个空闲时间为其他设备服务。数据传送是按字节交叉方式工作。

字节多路通道（2）

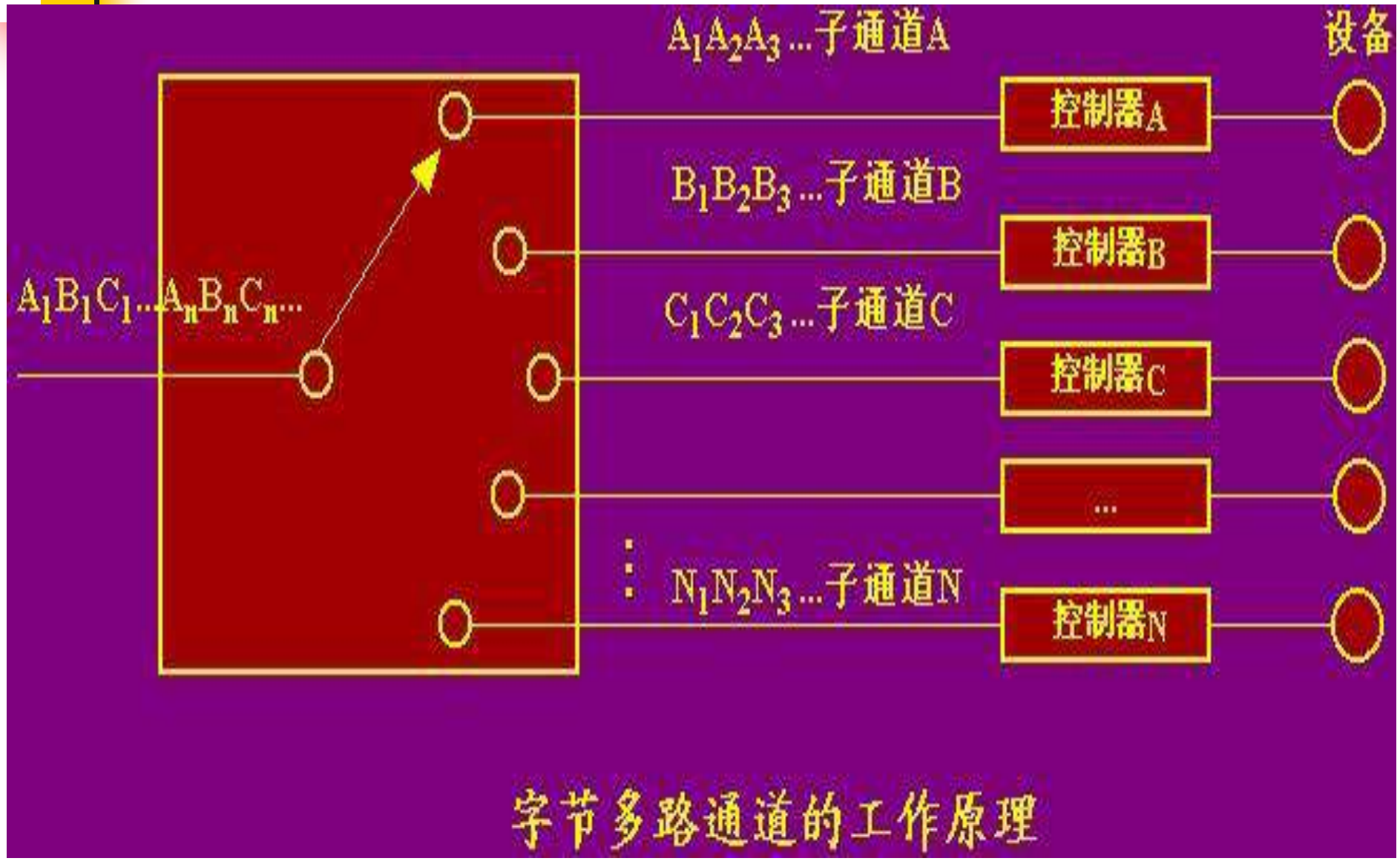
1）有一个主通道。

2）含有多个非分配型子通道A、B、C……

3）每个子通道通过一个控制器与一台中/低速的I/O设备相连，可同时并行向主通道传数据。

4）各子通道以时间片轮转方式按字节交叉使用主通道。

- 优点：可连多台中/低速设备；能分时并行操作。
- 缺点：传输率较低，不适于连接高速设备。



字节多路通道的工作原理

数组选择通道（1）

- 工作原理：

选择通道又称高速通道，在物理上它可以连接多个设备，但是这些设备不能同时工作，在某一段时间内通道只能选择一个设备进行工作。选择通道很像一个单道程序的处理器，在一段时间内只允许执行一个设备的通道程序，只有当这个设备的通道程序全部执行完毕后，才能执行其他设备的通道程序。数据传送是按成组方式进行工作，每次传输一批数据。主要用于连接高速I/O设备。

数组选择通道（2）

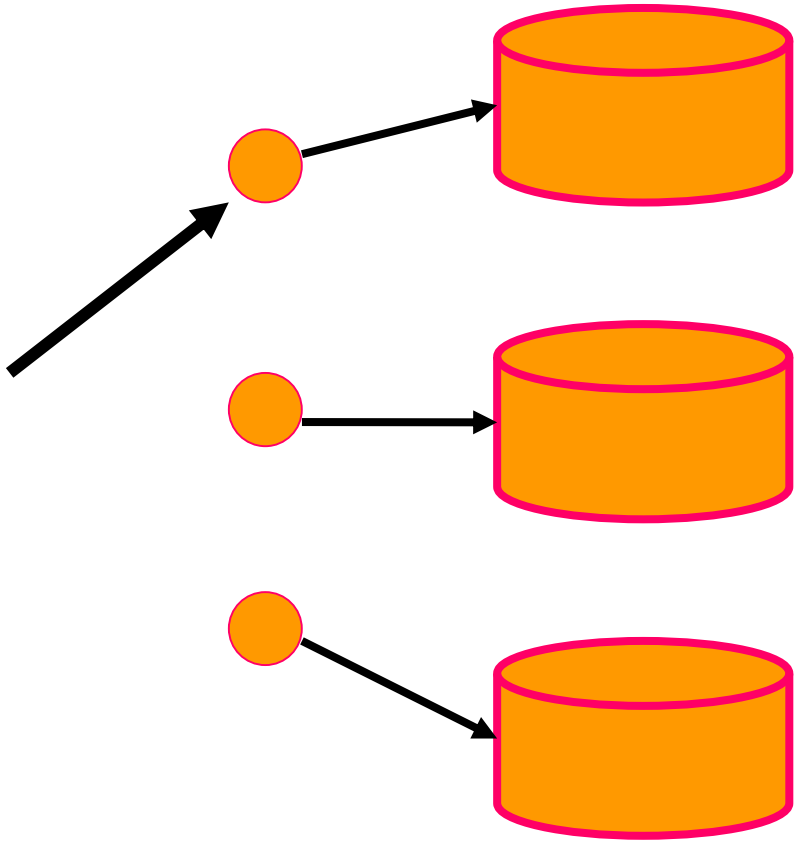
1)有一个主通道

2)含有多个子通道A、B、C……

3)每个子通道通过一个控制器与一台高速的I/O设备相连，在一段时间内只能选择一个子通道程序执行，控制一台设备进行数据传送。

- 优点：可连多台高速设备；传输速率较高。
- 缺点：某设备占用该通道后，如果不传输数据，就会使主通道闲置，其它子通道也不能传输数据（即每次只允许一台设备传输数据）。所以通道的利用率很低。

选择通道



数组多路通道

数据传送仍是按数组方式工作。

- 工作原理（结合两者：并行+数组）

1)有一个主通道

2)含有多个非分配型子通道A、B、C……

3)每个子通道通过一个控制器与一台高/中速的I/O设备相连，可分时并行向主通道传数据。

4)各子通道以时间片轮转方式共享主通道，按数组方式进行数据传输。

- 优点：可连多台高/中速设备；能分时并行操作，传输速率较高。

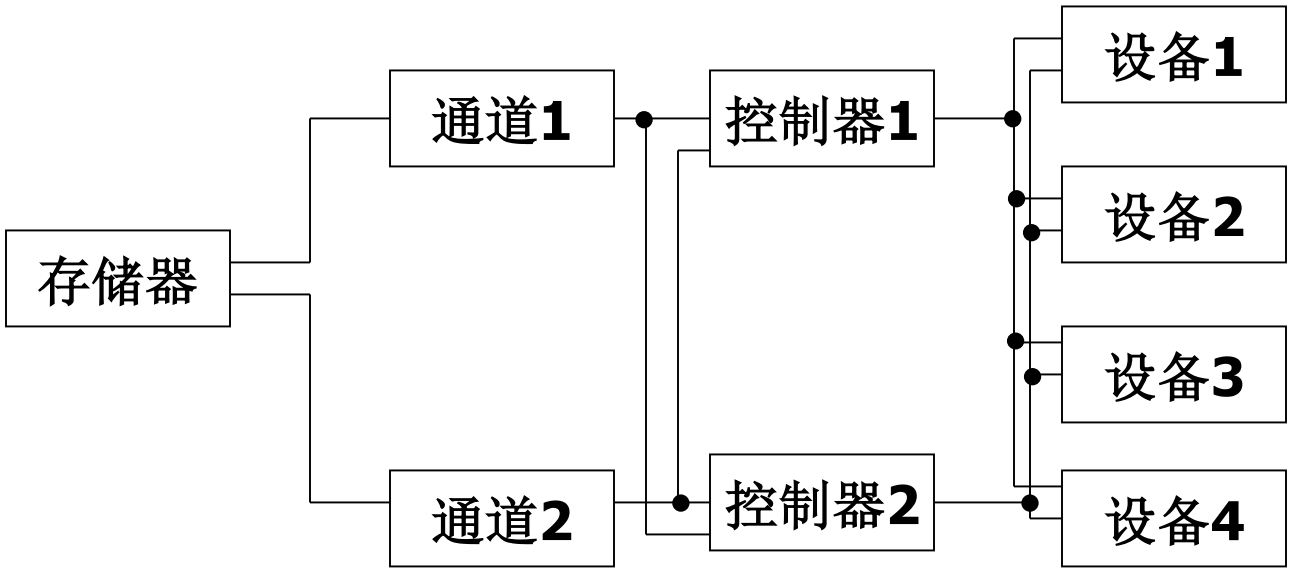
“瓶颈”问题

通道数量少，造成整个系统吞吐量下降

单通路I/O系统

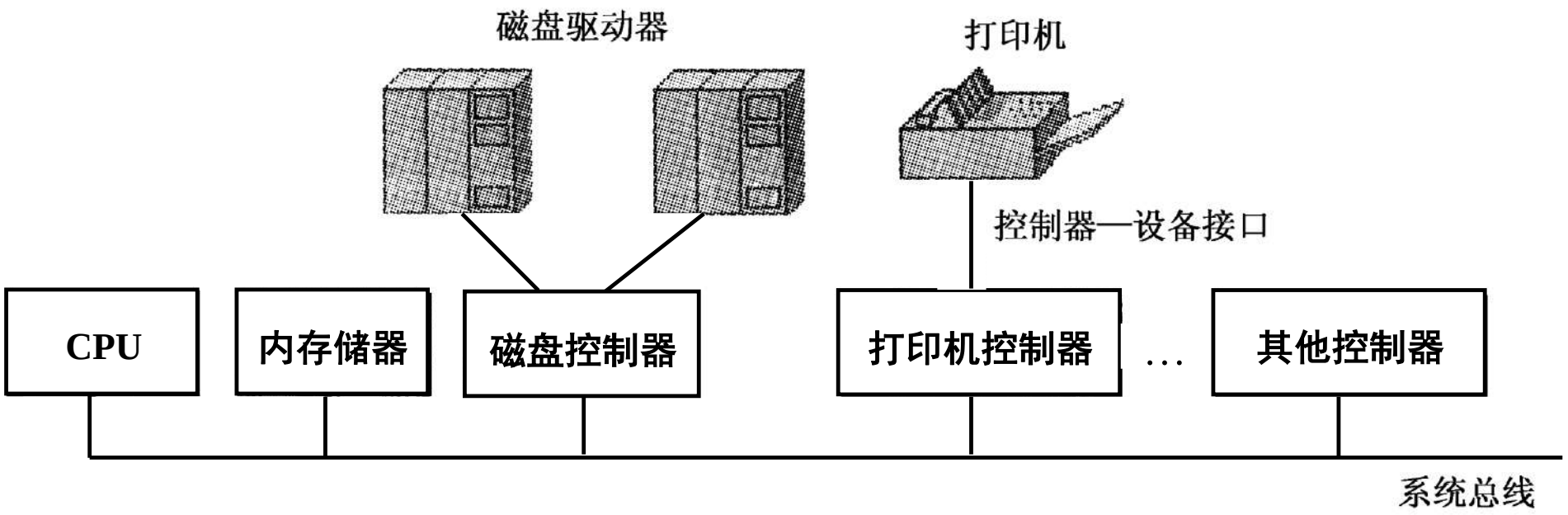
解决“瓶颈”问题的方法-多路方式

解决问题的方法是增加设备到主机的通路而不增加通道：即把一个设备连接到多个控制器上，一个控制器连接到多个通道上



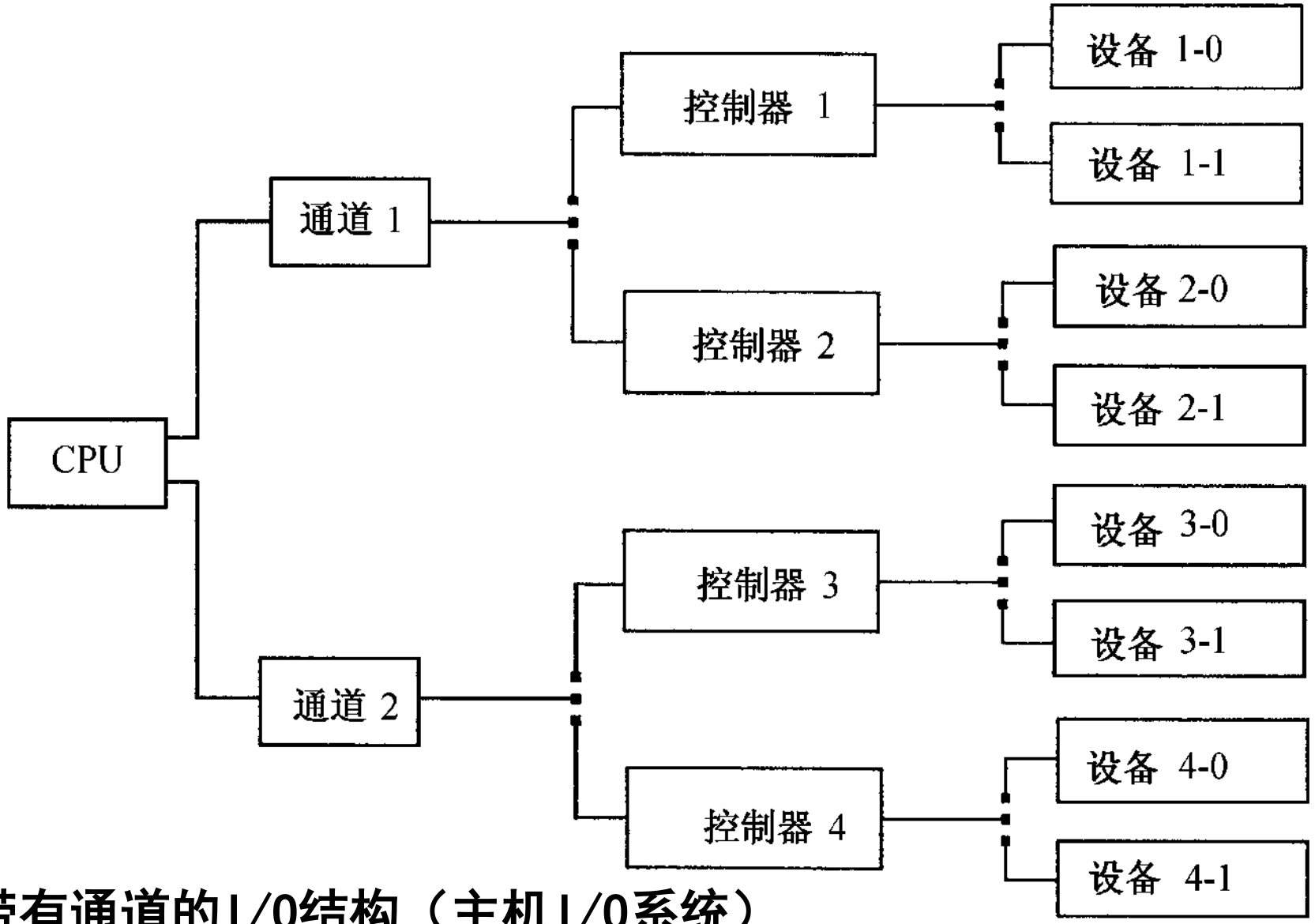
多通路I/O系统

6.1.4 I/O 系统的结构（1）



微型机I/O系统结构---总线型

6.1.4 I/O 系统的结构（2）



带有通道的I/O结构（主机I/O系统）

6.2 I/O 控制方式

按照I/O控制器功能的强弱，以及和CPU之间联系方式的不同，对I/O设备的控制方式分类。

主要差别在于：CPU和外围设备并行工作的方式不同，并行工作的程度不同

常用的输入/输出控制方式：

1、程序I/O方式

2、中断驱动I/O控制方式

3、直接存储器访问DMA I/O控制方式

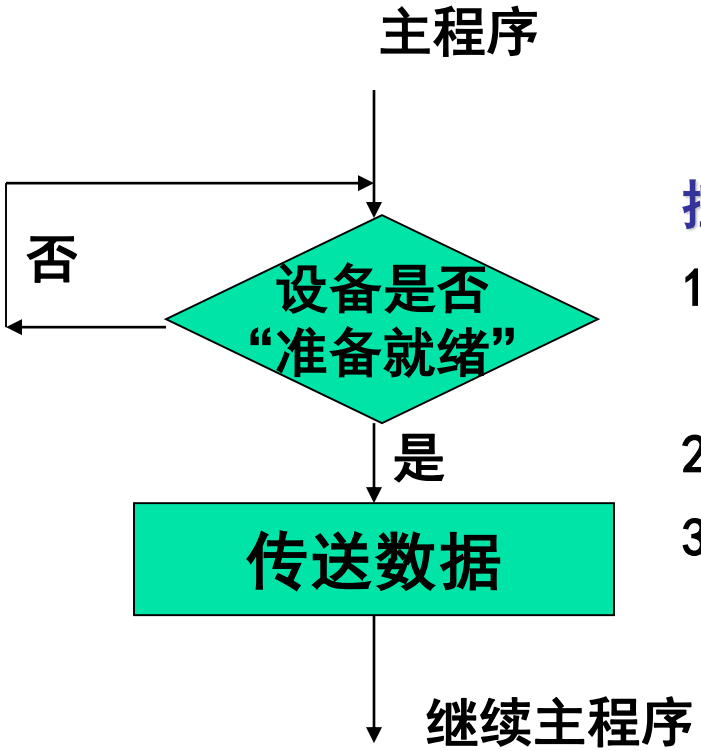
4、I/O通道控制方式

宗旨：减少主机对I/O控制的干预，提高并行度

1、程序直接控制方式（1）

处理机对I/O的控制采用程序直接控制方式。

工作原理：



控制设备状态的控制位

1）command-ready（等待命令）

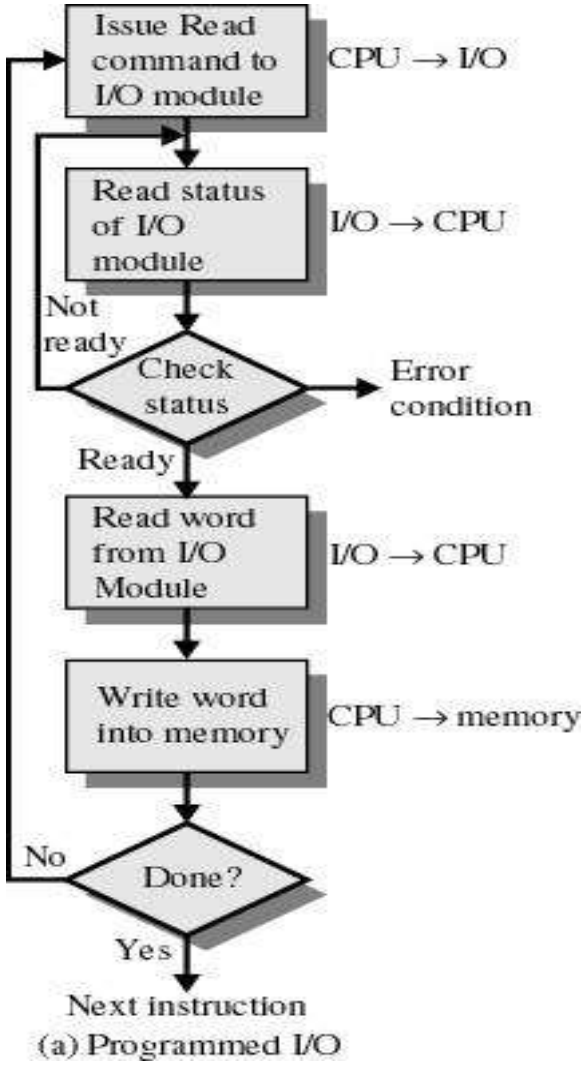
2）Busy（忙）

3）Error（错误）

特点：控制简单，但CPU的利用率低（串行），出现忙等待（循环等待设备的I/O操作）--即轮询（Polling）

1、程序直接控制方式（2）

- 一旦CPU启动I/O设备，便不断查询I/O设备的准备情况，终止原程序的执行，浪费CPU时间；
- I/O准备就绪后，CPU参与数据传送工作，而不能执行原程序；
- CPU和I/O设备串行工作，使主机不能充分发挥效率，外围设备也不能得到合理使用，整个系统效率很低。



(a) Programmed I/O

2、中断驱动I/O控制方式（1）

1）需数据的进程向CPU发出指令启动I/O设备输入数据。

2）该进程放弃处理机，等待输入完成。

3）输入完成后，I/O控制器向CPU发出中断请求，CPU收到后，转向中断服务程序。中断服务程序将数据寄存器中的数据送到指定内存单元，并将原进程唤醒，继续执行。

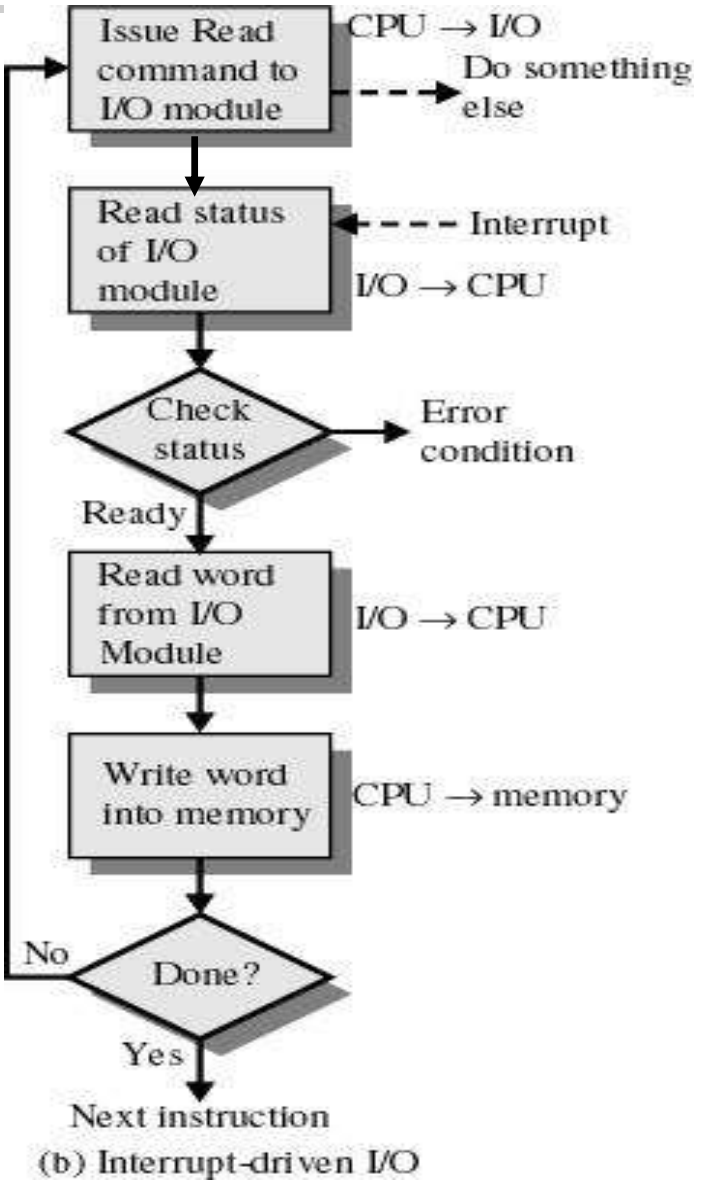
4）以后某时刻，该进程再次被调度，从内存单元取出数据进行处理。

优点—CPU利用率大大提高（可以与I/O设备并行工作）。

缺点—若中断次数较多，将耗去大量CPU处理时间。

2、中断驱动I/O控制方式（2）

- 向I/O发命令——返回——执行其它任务。
- I/O中断产生——CPU转相应中断处理程序。
- 如：读数据，读完后以中断方式通知CPU，CPU完成数据从I/O——内存



(b) Interrupt-driven I/O

2、中断驱动I/O控制方式（3）

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—等待中。 采用中断驱动方式后，CPU可利用这 99.9 ms的时间去做其它事情，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。

3、DMA I/O控制方式（1）

如果I/O设备能直接与主存交换数据而不占用CPU，CPU的利用率还可提高，这就出现了直接存储器访问（DMA）方式：

1）需数据的进程向CPU发出指令，向DMA控制器写入数据存放的内存始址、传送的字节数，并置中断位和启动位，启动I/O设备输入数据并允许中断。

2）该进程放弃处理机等待输入完成，处理机被其它进程占用。

3）DMA控制器挪用CPU周期，将一批数据写入内存中。

3、DMA I/O控制方式（2）

4）DMA控制器传送完数据后，向CPU发中断请求，CPU响应后转向中断服务程序，唤醒进程，并返回被中断进程。

5）以后某时刻，该进程再次被调度，从内存单元取出数据进行处理。

3、DMA I/O控制方式（3）

特点—数据传输的基本单位是数据块，在主机与I/O设备之间每次至少传送一个数据块；

所传送的数据块是从设备直接送入内存的（或相反）；

在传送一个或多个数据块的开始和结束才需CPU干预，传送过程是在控制器的控制下完成的

优点— CPU利用率进一步提高（并行度有所提高）。

缺点—数据传送方向、字节数、内存地址等需由CPU控制，且每一设备需一台DMA控制器，设备增多时，不经济。

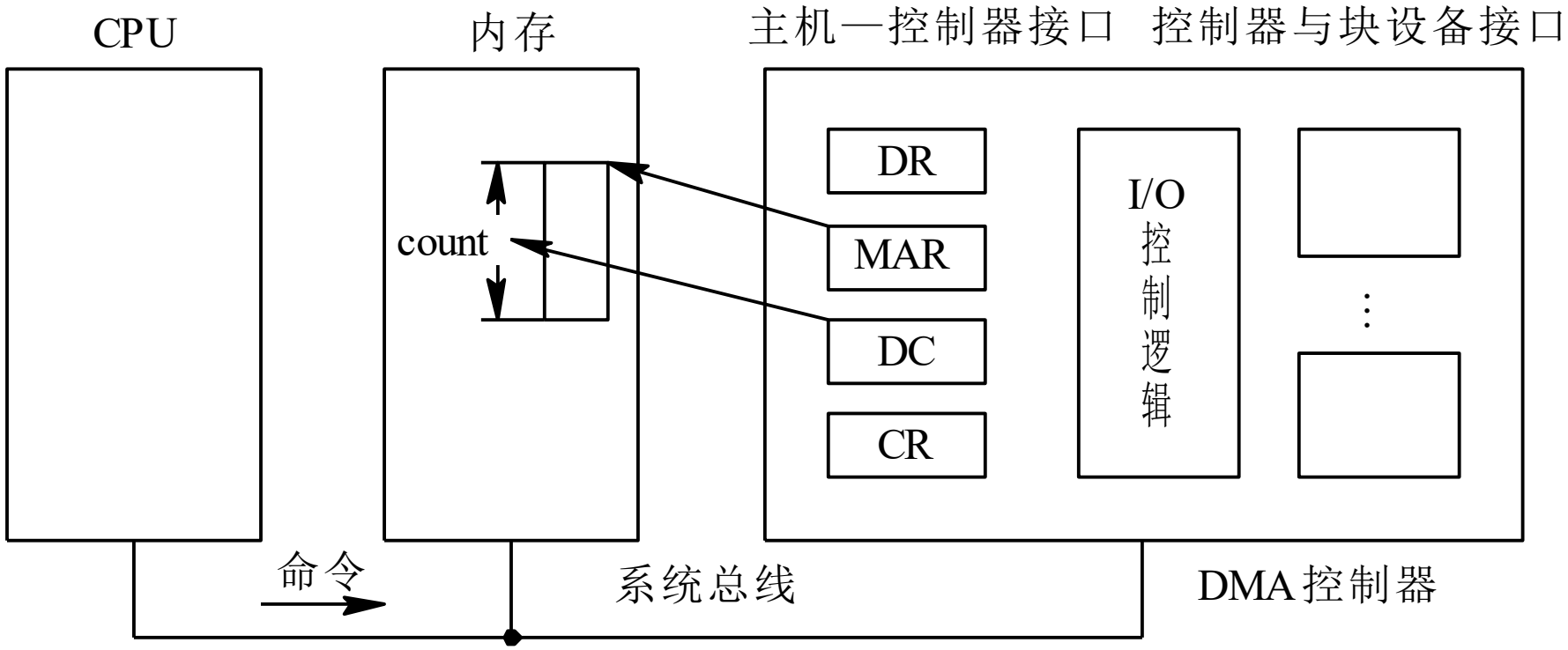
3、DMA I/O控制方式（4）

DMA控制器的组成部件（四类寄存器）：

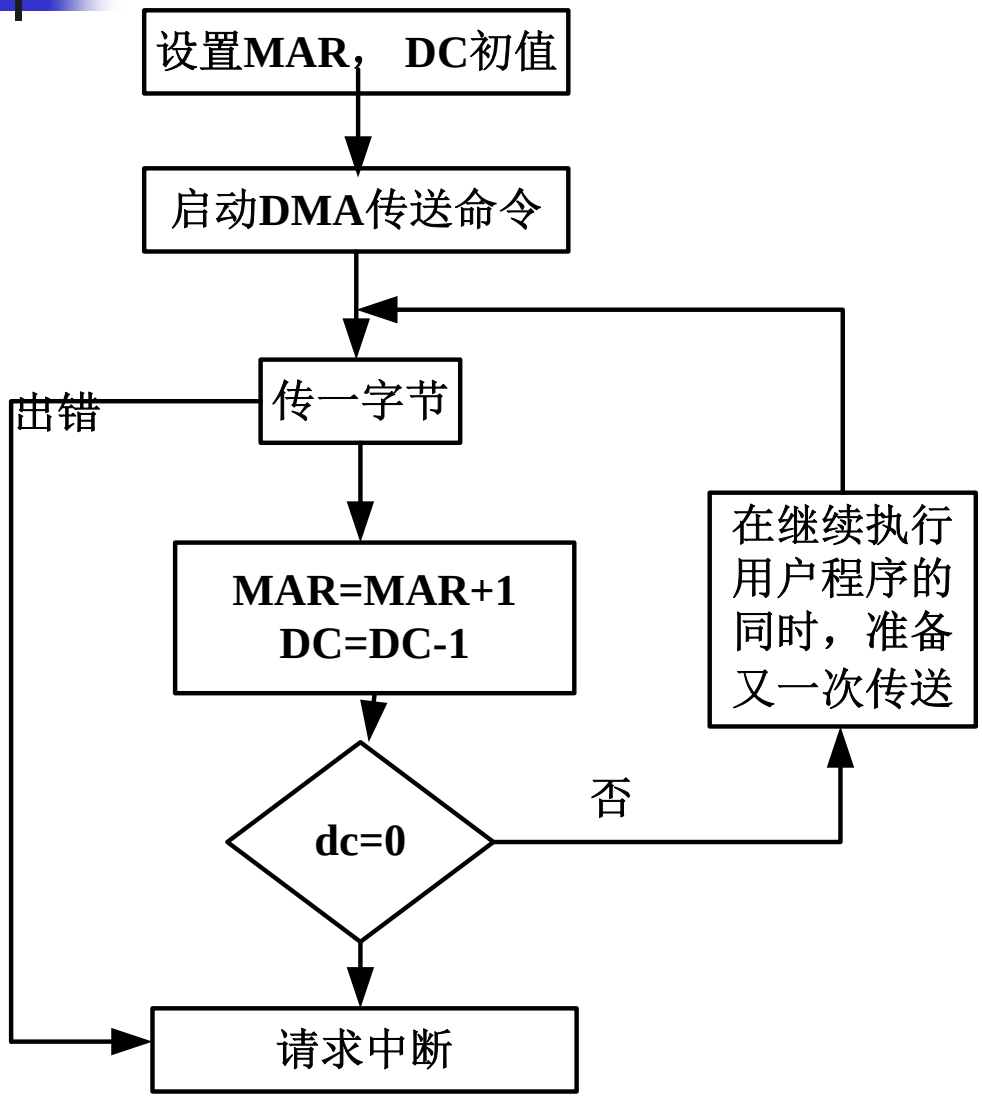
- 命令/状态寄存器 CR：用于接收从CPU发来的I/O命令或有关控制信息，或设备的状态。
- 内存地址寄存器 MAR：在输入时，它存放把数据从设备传送到内存的起始目标地址；在输出时，它存放由内存到设备的内存源地址。
- 数据寄存器 DR：用于暂存从设备到内存，或从内存到设备的数据。
- 数据计数器 DC：存放本次CPU要读或写的字(节)数。

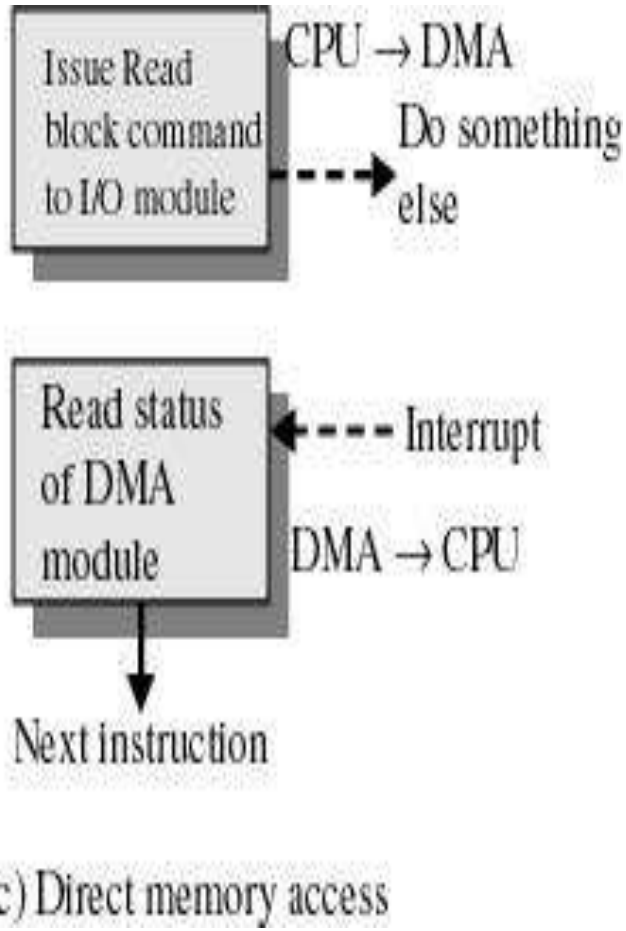
3、DMA I/O控制方式（5）

DMA控制器的组成：



3、DMA I/O控制方式（6）





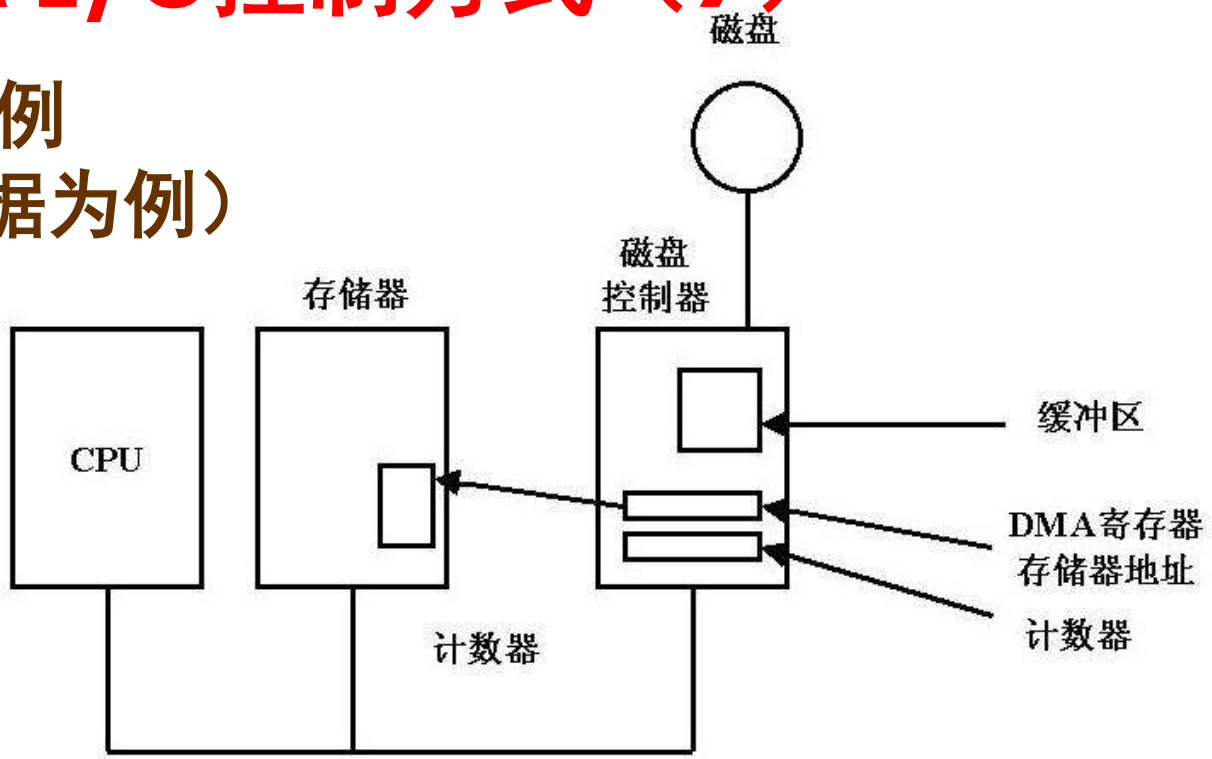
(c) Direct memory access

3、DMA I/O控制方式（7）

DMA工作示例
（以硬盘读入数据为例）

CPU提供
- 被读取块磁盘地址
- 目标存储地址
- 待读取字节数

整块数据读进缓冲区
核准校验



控制器按照指定存储器地址，把第一个字节送入主存
然后，按指定字节数进行数据传送
每当传送一个字节后，字节计数器值减1，直到字节计数器等于0
此时，控制器引发中断，通知操作系统，操作完成

4、通道控制方式（1）

为了获得CPU和外围设备间更高的并行工作能力，也为了让种类繁多，物理特性不同的外围设备能以标准的接口连接到系统中，计算机系统引入了自成独立体系的通道结构

由通道管理和控制I/O操作，减少了外围设备和CPU的逻辑联系，把CPU从琐碎的I/O操作中解放出来。

4、通道控制方式（2）

1）需数据的进程向CPU发出指令，CPU发启动指令指明I/O操作、设备号和对应的通道。

2）该进程放弃CPU等待输入完成，CPU被其它进程占用。

3）通道接收到CPU发来的启动指令后，取出内存中的通道程序执行，控制设备将数据传送到内存指定区域。

4）传送完数据后，通道向CPU发中断请求，CPU响应后转向中断服务程序，唤醒进程，并返回被中断进程。

5）以后某时刻，该进程再次被调度，从内存取出数据进行处理。

4、通道控制方式（3）

通道控制方式与DMA方式相类似，也是一种内存和设备直接进行数据交换的方式。与DMA方式不同的是，在通道控制方式中，数据传送方向、存放数据的内存始址及传送的数据块长度均由一个专门负责输入/输出的硬件——通道—来控制。

另外，DMA方式每台设备至少需要一个DMA控制器，而通道控制方式中，一个通道可控制多台设备与内存进行数据交换。

优点—一个通道可控制多台设备，所需CPU干预更少。
CPU利用率较高（并行度较高）。

缺点—通道价格较高。

4、通道控制方式（4）

每条通道指令应包含以下的内容：

（1）操作码：它规定指令所执行的操作，如读、写等。

（2）内存地址：标明数据传送时内存的首址。

（3）计数：表示传送数据的字节数。

（4）通道程序结束位P，表示通道程序是否结束。

（5）记录结束标志R，表示所处理的记录是否结束

操作	P	Record	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	1850
Write	1	1	250	720

4、通道控制方式（5）

通道相当于一个功能单纯的处理机，它具有自己的指令系统，包括读、写、控制、转移、结束以及空操作等指令，并可以执行由这些指令编写的通道程序。

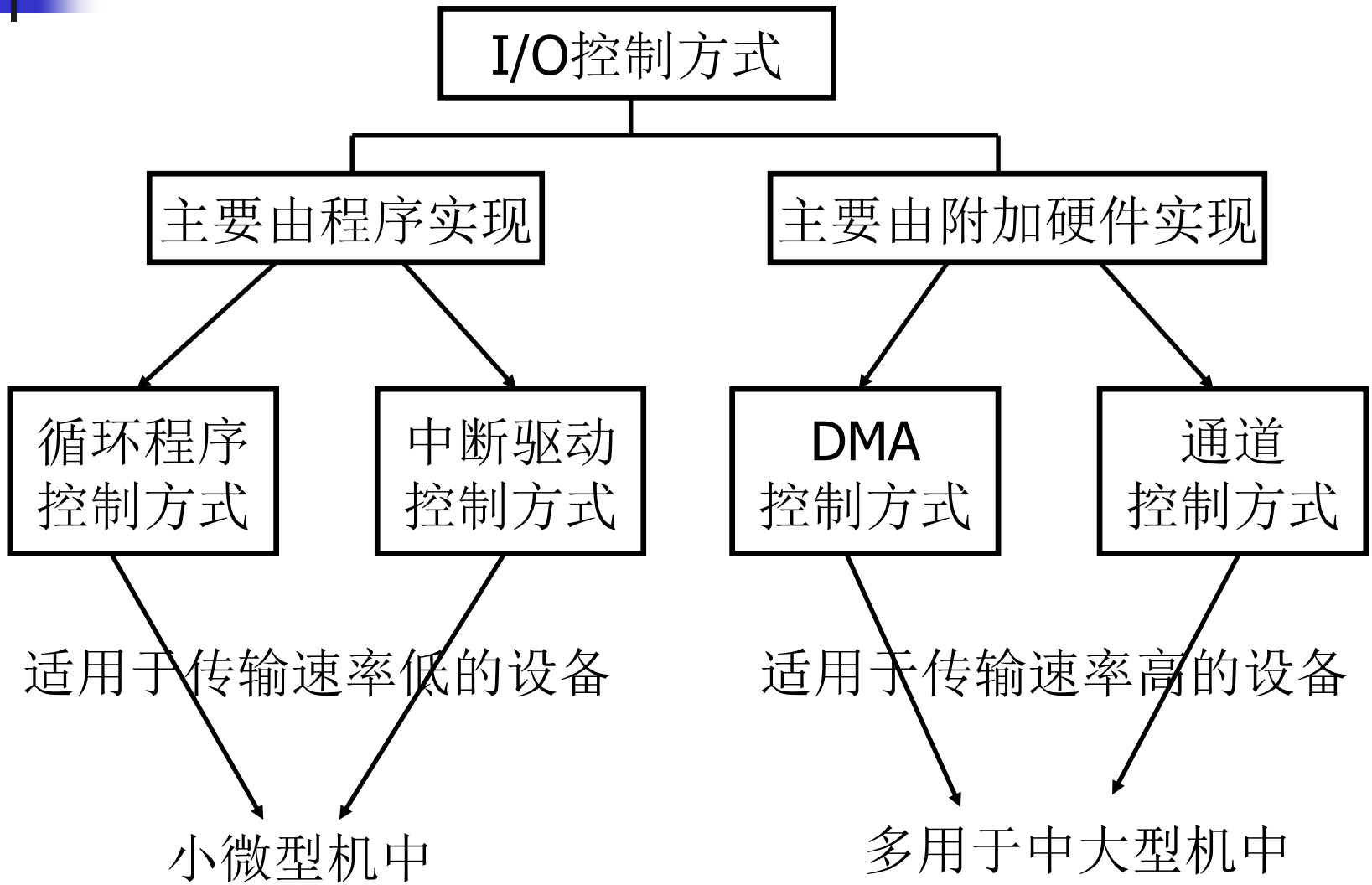
通道的运算控制部件包括：

① 通道地址字（CAW）：记录下一条通道指令存放的地址，其功能类似于中央处理机的地址寄存器。

② 通道命令字（CCW）：记录正在执行的通道指令，其作用相当于中央处理机的指令寄存器。

③ 通道状态字（CSW）：记录通道、控制器、设备的状态，包括I/O传输完成信息、出错信息、重复执行次数等。

总结：I/O控制方式


I/O控制方式
主要由程序实现
主要由附加硬件实现
循环程序控制方式
中断驱动控制方式
DMA控制方式
通道控制方式
适用于传输速率低的设备
适用于传输速率高的设备
小微型机中
多用于中大型机中

6.3 缓冲管理（1）

1、提高处理机与I/O设备的并行工作的技术：

1） 数据传送控制方式

2） 缓冲技术

2、操作系统中，引入缓冲的主要原因

1）缓冲CPU与I/O设备间速度不匹配的矛盾。

2）减少中断CPU的次数

3）提高CPU与I/O设备的并行性

6.3 缓冲管理（2）

3、缓冲就是用来对数据传送速度不同的设备的传送速度进行匹配/缓冲的一种常用手段，例：

1）CPU与内存之间设置高速缓存（Cache Memory）

2）主存与显示器之间设置显示缓存

3）主存与打印机之间有打印缓存等等

4、缓冲技术的分类

单缓冲　双缓冲　循环缓冲　缓冲池

6.3 缓冲管理（3）

5、缓冲技术实现的基本思想：

- 进程执行写操作输出数据时，向系统申请一个缓冲区，若为顺序写请求，则不断把数据填到缓冲区，直到被装满。此后，进程继续它的计算，系统将缓冲区内容写到I/O设备上。
- 进程执行读操作输入数据时，向系统申请一个缓冲区，系统将一个物理记录的内容读到缓冲区，根据进程要求，把当前需要的逻辑记录从缓冲区中选出并传送给进程

6.3 缓冲管理（4）

5、缓冲技术实现的基本思想（续）：

- 在输出数据时，只有在系统还来不及腾空缓冲区而进程又要写数据时，它才需要等待；
- 在输入数据时，仅当缓冲区空而进程又要从中读取数据时，它才被迫等待

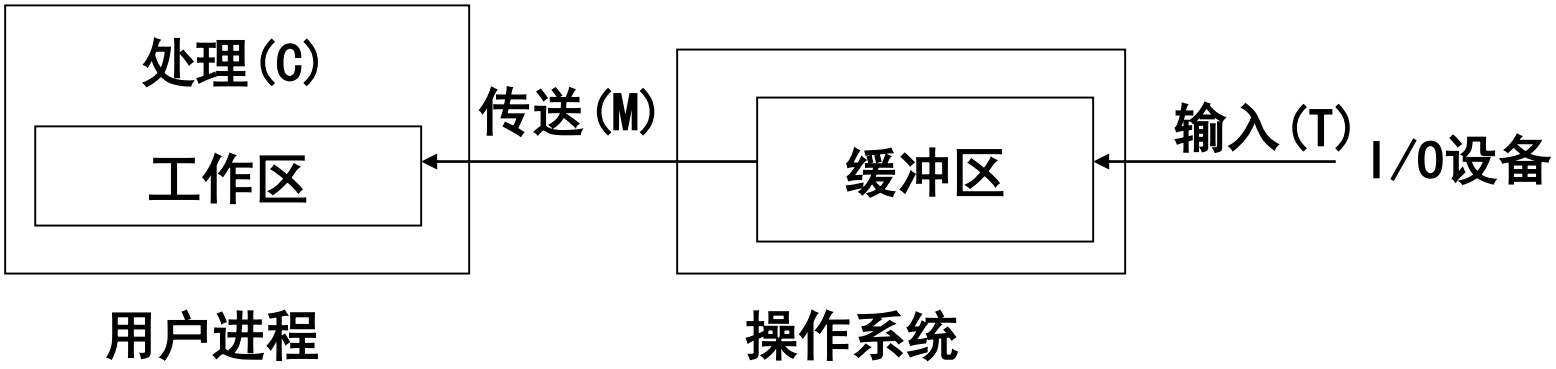
6、缓冲的实现方式：

- 采用硬件缓冲器实现；
- 在内存划出一块区域，专门用来存放临时输入输出的数据，这个区域称为缓冲区

单缓冲

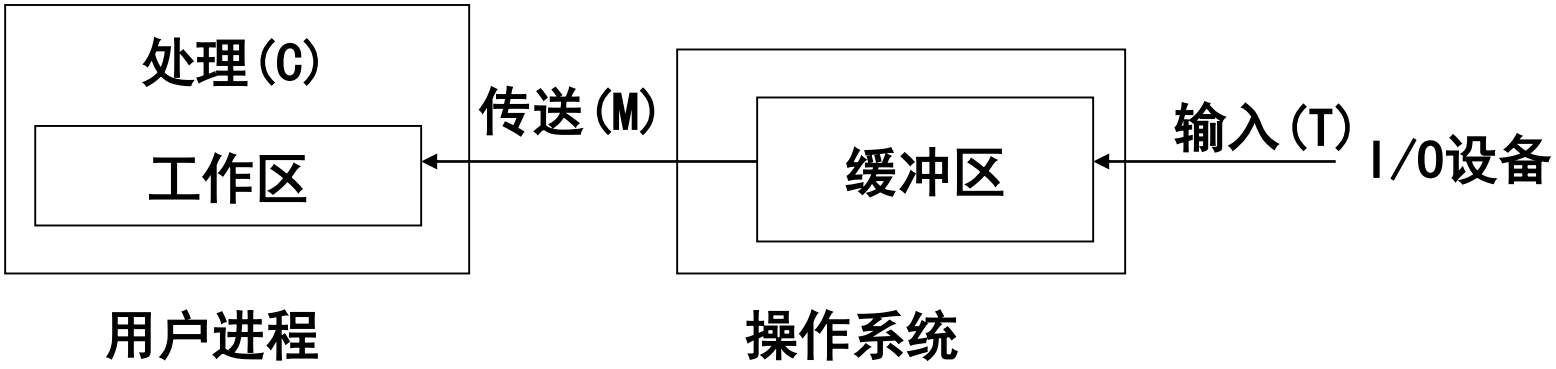
在设备和处理机之间设置一个缓冲。设备与处理机交换数据时，先把交换的数据写入缓冲区，然后需要数据的设备/处理机再从缓冲区中取走数据。

特点：缓冲区数只有一个；设备与处理机对缓冲区的操作是串行的。



一块数据的处理时间

- 在某系统中，从磁盘将一块数据输入到缓冲区需要花费的时间T，CPU对一块数据进行处理的时间为C，将缓冲区的数据传送到用户区所花时间为M，那么在单缓冲情况下，系统处理大量数据时，一块数据的处理时间为多少？

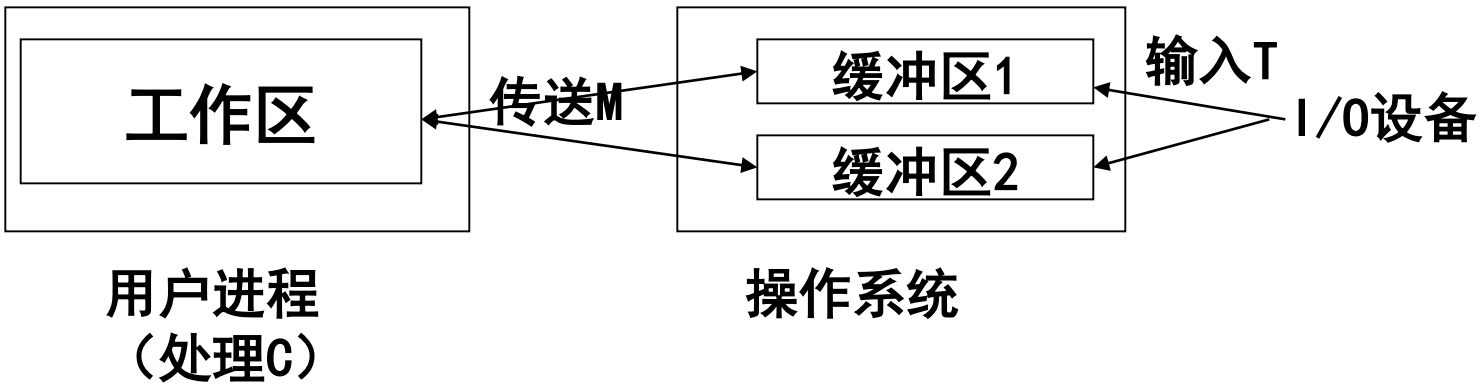


Max(T,C)+M

双缓冲

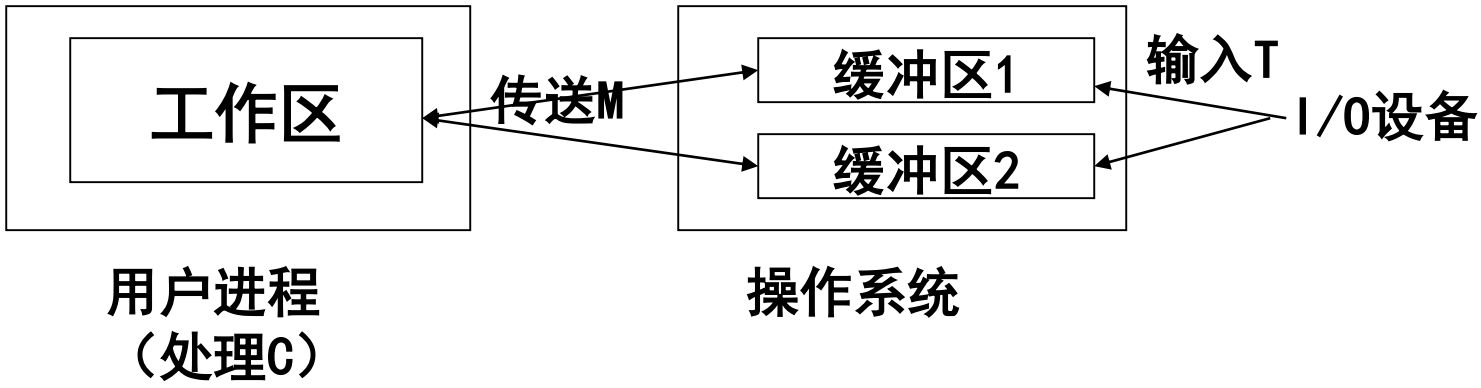
在设备和处理机之间设置2个缓冲。设备与处理机交换数据时，先把交换的数据写入缓冲区，然后需要数据的设备/处理机再从缓冲区中取走数据。因缓冲区有2个，提高了设备与处理机并行操作的程度，只有当两个均为空时，需数据的进程才等待。

特点：缓冲区数有2个；设备与处理机对缓冲区的操作可并行，提高了设备与处理机并行操作的程度。



一块数据的处理时间

- 在某系统中，从磁盘将一块数据输入到缓冲区需要花费的时间T，CPU对一块数据进行处理的时间为C，将缓冲区的数据传送到用户区所花时间为M，那么在双缓冲情况下，系统处理大量数据时，一块数据的处理时间为多少？



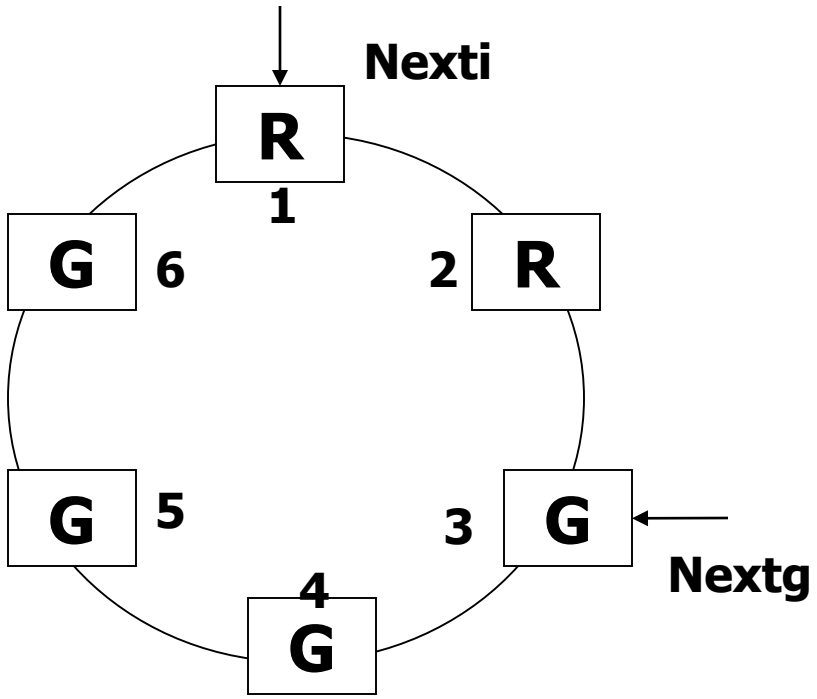
一块数据的处理时间： MAX（C+M，T）

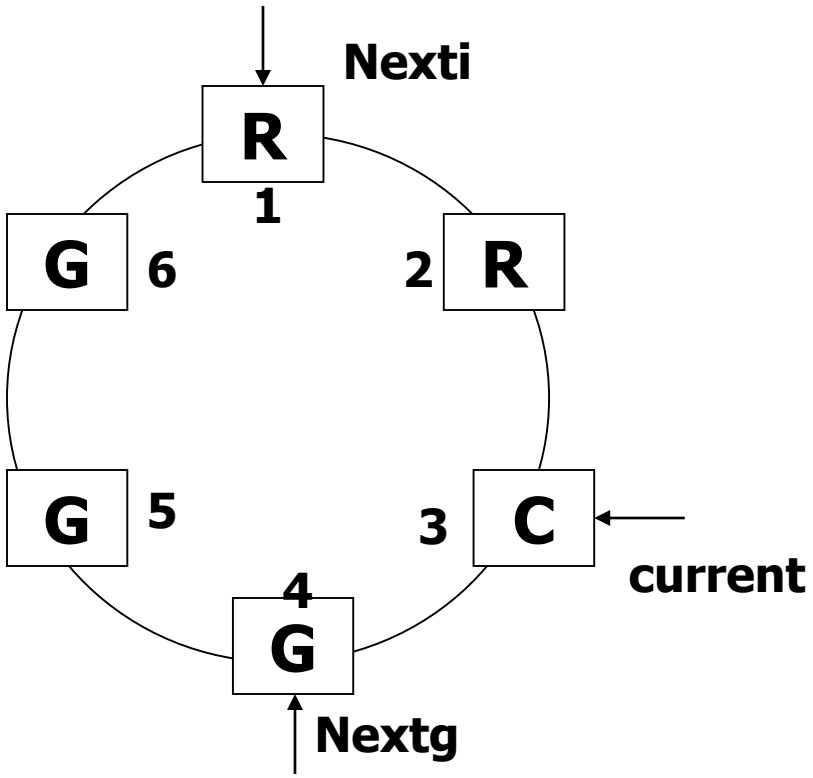
循环缓冲

在设备和处理机之间设置多个大小相等的缓冲区，这些缓冲区构成环形，每一个缓冲区中含一指针指向下一个缓冲区，最后一个指向第一个缓冲区，同时还含有2个用于输入/输出的指针IN和OUT。

特点：缓冲区的数量有多个；设备与处理机对缓冲区的操作可并行，进一步提高了设备与处理机并行操作的程度。

循环缓冲的组成


Nexti
R
1
G
6
2
R
G
5
3
G
4
G
Nextg


Nexti
R
1
G
6
2
R
G
5
3
C
4
G
current
Nextg

循环缓冲

- 缓冲区的使用
  - Getbuf过程
  - Releasebuf过程
- 进程同步
  - Nexti指针追赶上Nextg指针—输入进程阻塞
  - Nextg指针追赶上Nexti指针—计算进程阻塞

缓冲池（1）

1、缓冲池：将系统内所有的缓冲区统一管理起来，就形成了能用于输入/输出的缓冲池。缓冲池通常由若干大小相同的缓冲区组成，是系统的公用资源，任何进程都可以申请使用缓冲池中的各个缓冲区。

2、缓冲池的组成（数据结构）

三个队列：空缓冲队列emq、装满输入数据队列inq、装满输出数据队列outq

四个工作缓冲区： 收容输入数据的缓冲区、提取输入数据的缓冲区、收容输出数据的缓冲区、提取输出数据的缓冲区

缓冲池（2）

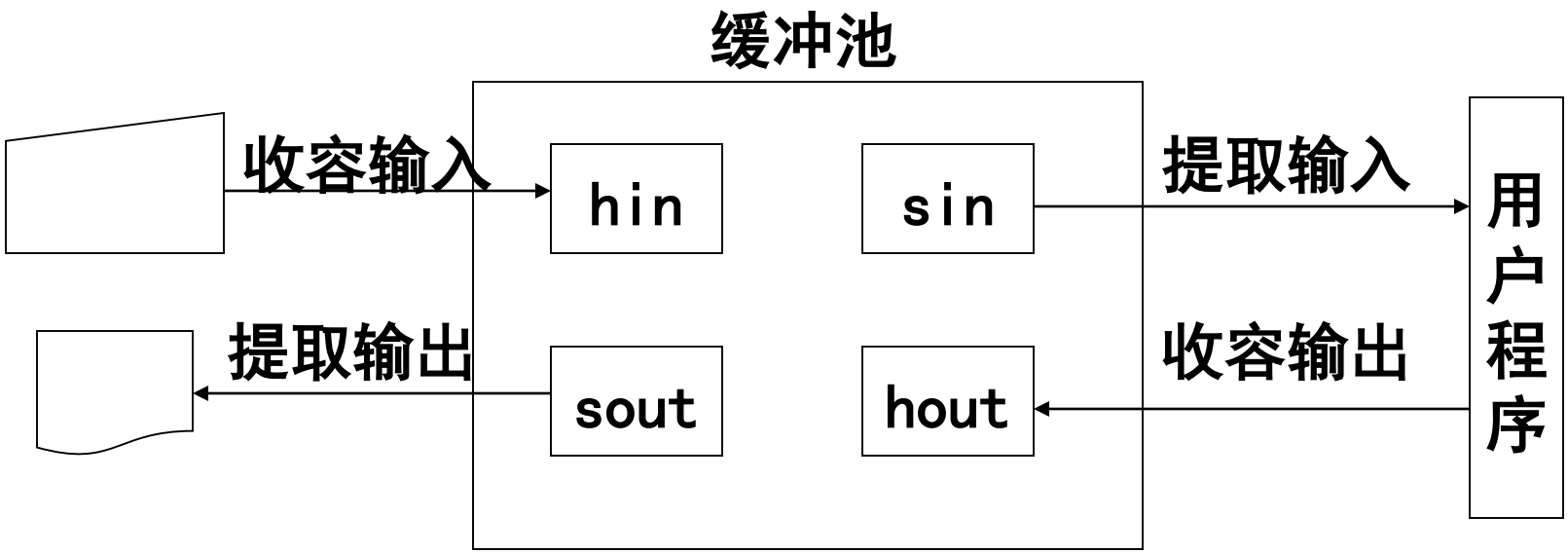
3、Getbuf过程和Putbuf过程

```
Procedure Getbuf(type)
begin
    wait(rs(type));
    wait(ms(type));
B(number):=Takebuf(type);
    signal(ms(type));
   end
```

```
Procedure Putbuf(type)
begin
    wait(ms(type));
  Addbuf(type,number);
    signal(ms(type));
    signal(rs(type));
 end
```

注：Takebuf(type)—用于从**type**所指的队列的队首摘下一缓冲区
Addbuf(type,number)—用于将由**number**所指示的缓冲区**B**挂在**type**队列上

4、操作系统对缓冲池的管理--工作方式



缓冲池的工作方式

4、操作系统对缓冲池的管理--工作方式

输入进程需要输入数据时：输入设备 ⟶ 收容输入缓冲区-getbuf(emq)

1)从空缓冲队列的队首取一空缓冲区用作收容输入缓冲区

2)输入设备将数据输入收容输入缓冲区并装满

3)将此缓冲区挂到装满输入数据队列队尾。

计算进程需要输入数据时：提取输入缓冲区 ⟶ CPU - getbuf(inq)

1)从装满输入数据队列队首取一满缓冲区用作提取输入缓冲区

2)CPU从提取输入缓冲区中取出数据至用完

3)将空缓冲区挂到空缓冲队列队尾。

4、操作系统对缓冲池的管理--工作方式

计算进程需要输出数据时：CPU ⟶ 收容输出缓冲区-getbuf(emq)

1)从空缓冲队列队首取一空缓冲区用作收容输出缓冲区

2)CPU将数据输入其中并装满

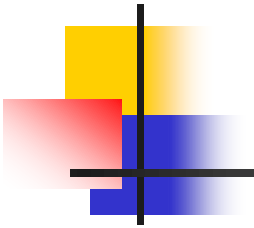
3)将收容输出缓冲区挂到装满输出数据队列队尾。

输出进程需要输出数据时：提取输出缓冲区 ⟶ 输出设备-getbuf(outq)

1)从装满输出数据队列队首取一满缓冲区用作提取输出缓冲区

2)输出设备从中取出数据至用完

3)将空缓冲区挂到空缓冲队列队尾



6.4 I/O 软件

- I/O 软件的设计目标和原则
- 中断处理程序
- 设备驱动程序
- 设备独立性软件
- 用户层的I/O软件

6.4.1 I/O软件的设计目标和原则

- 总体设计目标
  - 高效性
    - 确保I/O设备和CPU并行执行，提高资源利用率
  - 通用性
    - 提供简单抽象、清晰统一的接口，采用统一标准的方法，来管理所有的设备和所需的I/O操作
- 采用层次结构的I/O软件
  - 低层软件
    - 实现与硬件相关的操作，屏蔽硬件的具体细节
  - 高层软件
    - 向用户提供一个简洁、友好、规范的接口

6.4.1 I/O软件的设计目标和原则

- I/O软件应达到以下几个目标
  - 与具体设备无关
    - 屏蔽设备的具体细节，向高层提供抽象的逻辑设备，并完成逻辑设备和具体物理设备的映射
  - 统一命名
    - 所有软件都以逻辑名称访问设备，与具体设备无关

6.4.1 I/O软件的设计目标和原则

- I/O软件应达到以下几个目标（续）
  - 对错误的处理
    - 尽可能在接近硬件的层面处理错误
  - 缓冲技术
  - 设备的分配和释放
  - I/O控制方式
    - 合理选择I/O控制方式，例打印机（中断驱动）、磁盘（DMA控制方式）

6.4.1 I/O软件的设计目标和原则

- 层次式结构的I/O软件（四个层次）
  - 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，对设备操作
  - 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、设备保护以及设备的分配与释放，提供存储空间

6.4.1 I/O软件的设计目标和原则

- 层次式结构的I/O软件（四个层次）（续）
  - 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
  - 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

6.4.2 中断处理程序

中断处理层的主要工作

- 进行进程上下文的切换
- 对处理中断信号源进行测试
- 读取设备状态
- 修改进程状态

中断处理程序的处理过程

- 唤醒被阻塞的驱动程序进程
- 保护被中断进程的CPU环境
- 分析中断原因，转入相应的设备处理程序
- 进行中断处理
- 恢复被中断进程的现场

6.4.3 设备驱动程序

是I/O进程与设备控制器之间的通信程序，常以进程的形式存在，主要任务

- 接收上层软件发来的抽象I/O要求
- 转换为具体要求后，发送给设备控制器，启动设备去执行
- 将由设备控制器发来的信号传送给上层软件
- 修改进程状态

设备驱动程序的功能

设备处理方式

设备驱动程序的特点

设备驱动程序的处理过程

设备驱动程序的功能

- 将接收到的抽象要求转换为具体要求。
- 检查用户I/O请求的合法性，I/O设备状态，传参数，设置设备的工作方式。
- 按处理机的I/O请求去启动指定的设备进行I/O操作
- 及时响应由控制器或通道发来的中断请求，并进行相应处理
- 按I/O请求构成相应通道程序。

设备处理方式

- 为每一类设备设置一进程，专门执行其I/O操作。
- 在整个系统中设置一个进程，执行所有的I/O操作。
- 不设置专门的设备处理进程，而为各类设备设置相应的设备驱动程序。

设备驱动程序的特点

- 是请求I/O的进程与设备控制器之间的一个通信程序。
- 与设备控制器和I/O设备的硬件特性紧密相关
- 与I/O设备所采用的I/O控制方式紧密相关
- 与硬件紧密相关，因而其中一部分程序必须用汇编语言编写。
- 驱动程序应允许可重入
- 驱动程序不允许系统调用，但可以允许对某些内核过程的调用

设备驱动程序的的处理过程

- 将接收到的抽象要求转换为具体要求。
- 检查用户I/O请求的合法性
- 读出和检查 I/O设备状态
- 传送必要参数
- 设置设备的工作方式。
- 按处理机的I/O请求去启动指定的设备进行I/O操作

6.4.4 设备独立性软件

- 设备独立性概念（设备无关性）
- 设备独立性的实现
  - 逻辑设备（应用程序）和物理设备（执行）
    - 设备分配时的灵活性
    - 易于实现I/O重定向
  - 设备独立性软件
    - 执行所有设备的公有操作
    - 向用户层（文件层）软件提供统一的接口
  - 逻辑设备名到物理设备名映射的实现
    - 逻辑设备表LUT(Logical Unit Table)
    - LUT设置问题(图6-19)

用于单用户系统：整个系统设置一张LUT ；

用于多用户系统：每个用户设一张LUT

设备独立性相关概念

- 设备独立性概念（设备无关性）

  为提高OS的可适应性和可扩展性，而将应用程序独立于具体使用的物理设备。

- I/O重定向

  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。

- 所有设备的公有操作

  独立设备的分配与回收；将逻辑设备名映射为物理设备名；对设备进行保护（禁止直接访问）；缓冲管理；差错控制。

6.5 设备分配

- 设备分配中的数据结构
- 设备分配的策略/应考虑的因素
- 独占设备的分配程序
- SPOOLING技术

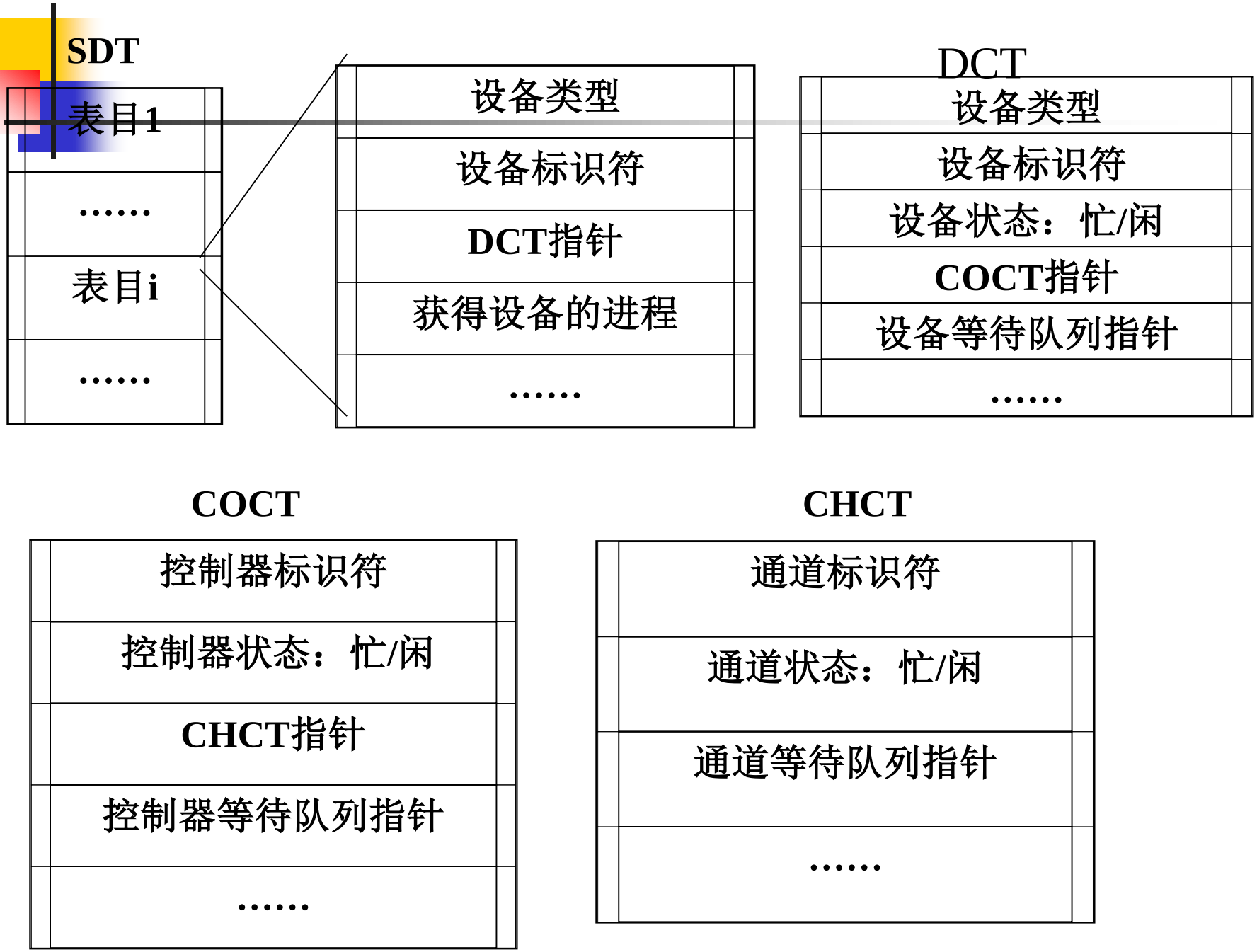
6.5.1 设备分配中的数据结构

设备控制表DCT（Device Control Table）

控制器控制表COCT（COntroller Control Table）

通道控制表CHCT（CHannel Control Table）

系统设备表SDT（System Device Table）


SDT
表目1
……
表目i
……
设备类型
设备标识符
DCT指针
获得设备的进程
……
DCT
设备类型
设备标识符
设备状态：忙/闲
COCT指针
设备等待队列指针
……
COCT
控制器标识符
控制器状态：忙/闲
CHCT指针
控制器等待队列指针
……
CHCT
通道标识符
通道状态：忙/闲
通道等待队列指针
……

6.5.2 设备分配策略/应考虑的因素

1、设备的使用性质/固有属性

（独享设备 、共享设备、虚拟设备）

2、设备分配算法

（先请求先服务、优先级高者优先 ）

3、设备分配的安全性（防止进程死锁）

设备分配算法

先请求先服务

当有多进程对同一设备提出I/O请求时，系统根据这些进程发出请求的先后次序将它们排成一个设备请求队列，设备分配程序总是把设备分配给队首的进程。

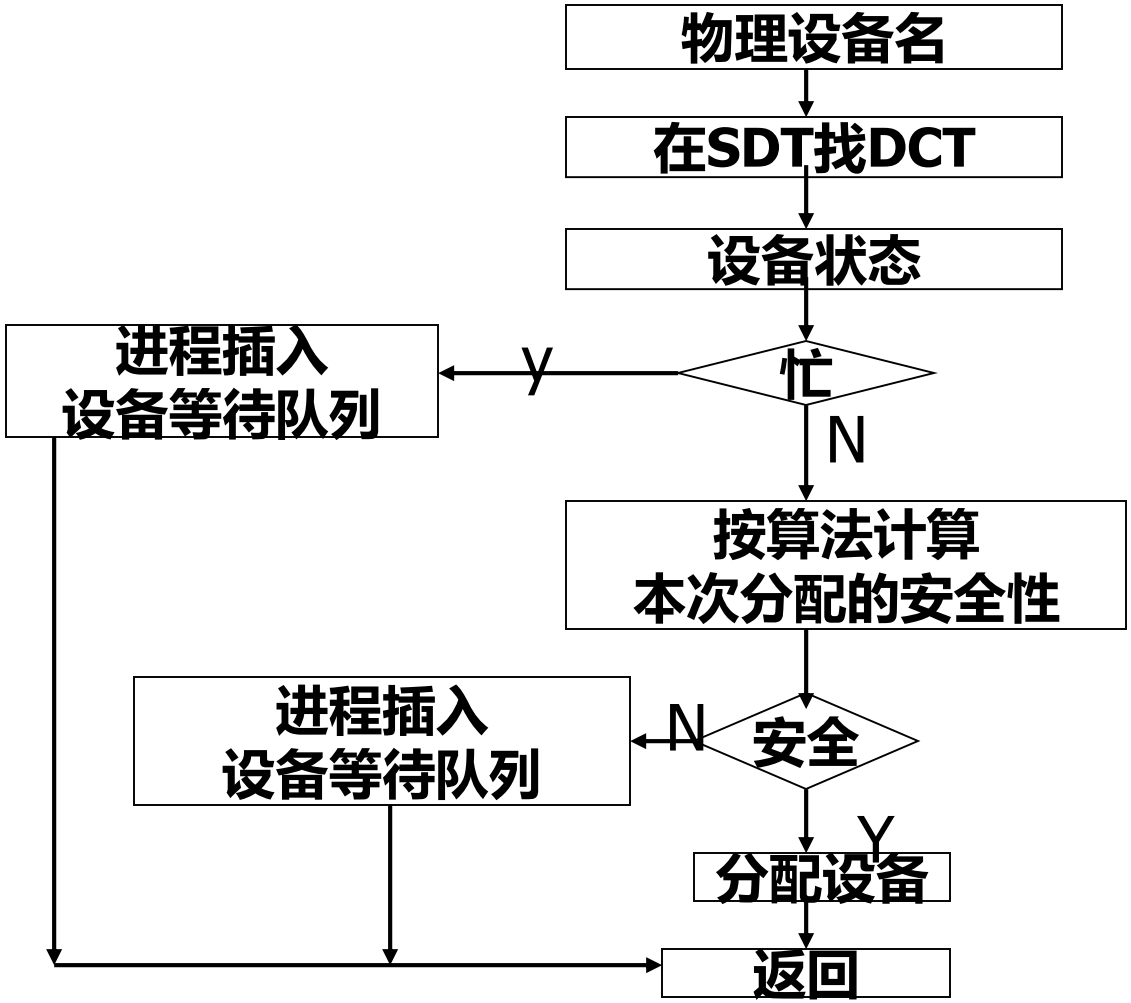
优先级高者优先

按照进程优先级的高低进行分配。即当多进程对同一设备提出I/O请求时，谁优先级高，就将设备分配给谁。若优先级相同，则按先请求先服务进行分配。

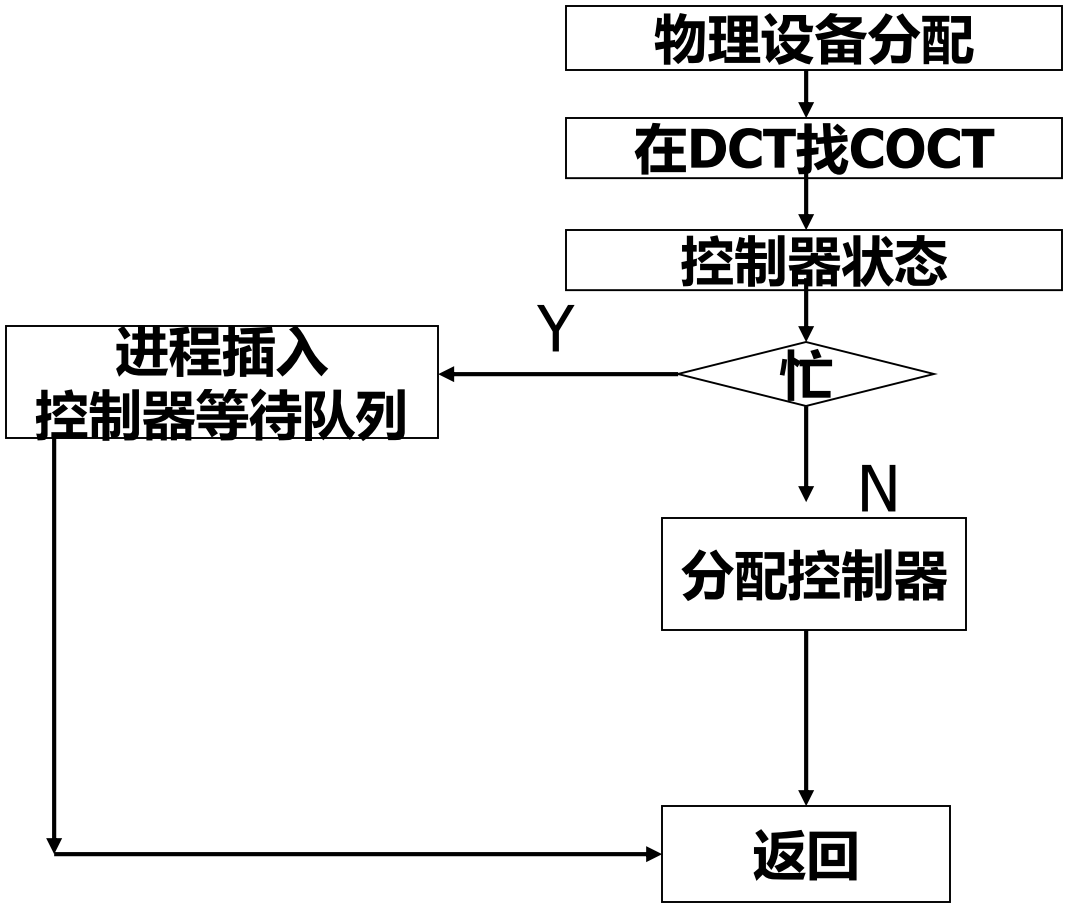
6.5.3 独占设备的分配程序

- 基本的设备分配程序
  - 分配设备
  - 分配控制器
  - 分配通道
  - 问题（“瓶颈”）
    - 进程以物理设备名来提出I/O请求
    - 采用的是单通路的I/O系统结构
- 设备分配程序的改进
  - 增加设备的独立性(进程以逻辑设备名来提出I/O请求)
  - 考虑多通路情况

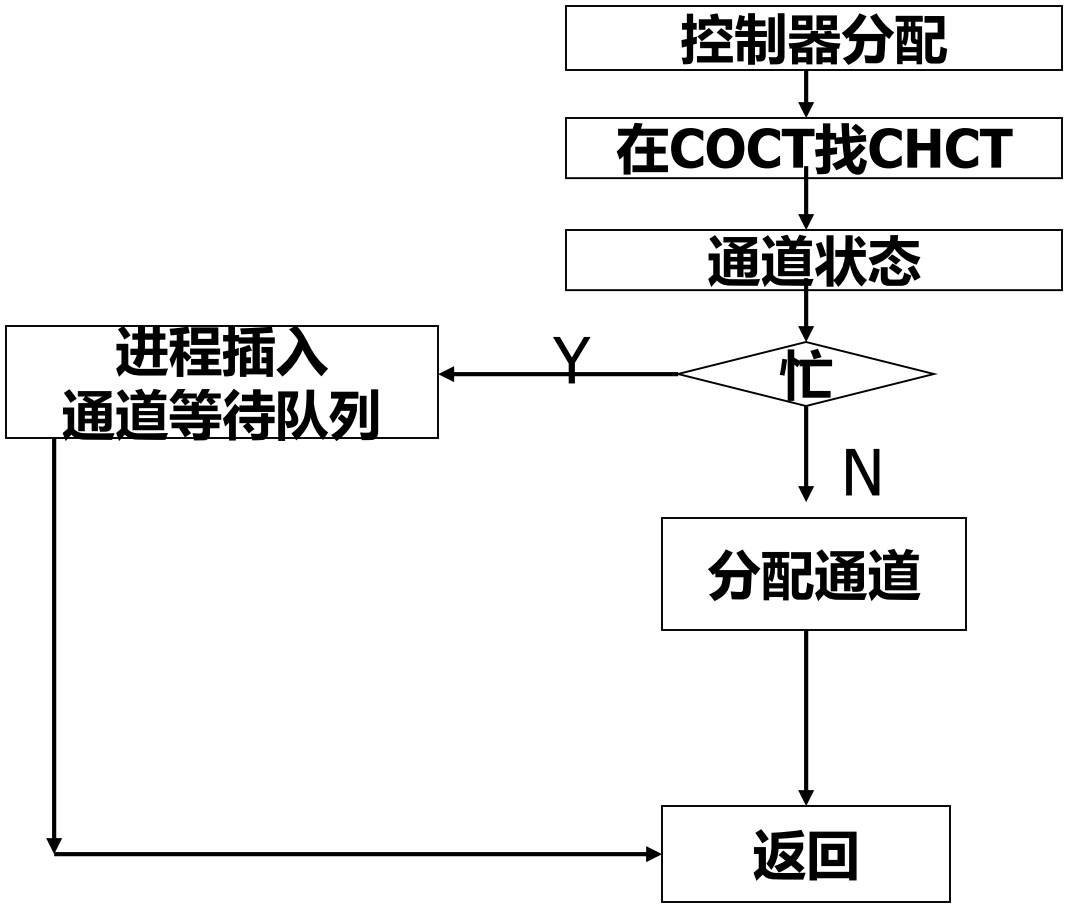
分配设备


物理设备名
在SDT找DCT
设备状态
忙
y
进程插入
设备等待队列
N
按算法计算
本次分配的安全性
安全
N
进程插入
设备等待队列
Y
分配设备
返回

分配控制器



分配通道



SPOOLING技术(Simultaneaus Periphernal Operations On-Line)

- 脱机输入、输出技术

  为了缓和CPU的高速性与I/O设备的低速性间矛盾而引入，该技术在外围控制机的控制下实现低速的I/O设备与高速的磁盘之间进行数据传送。

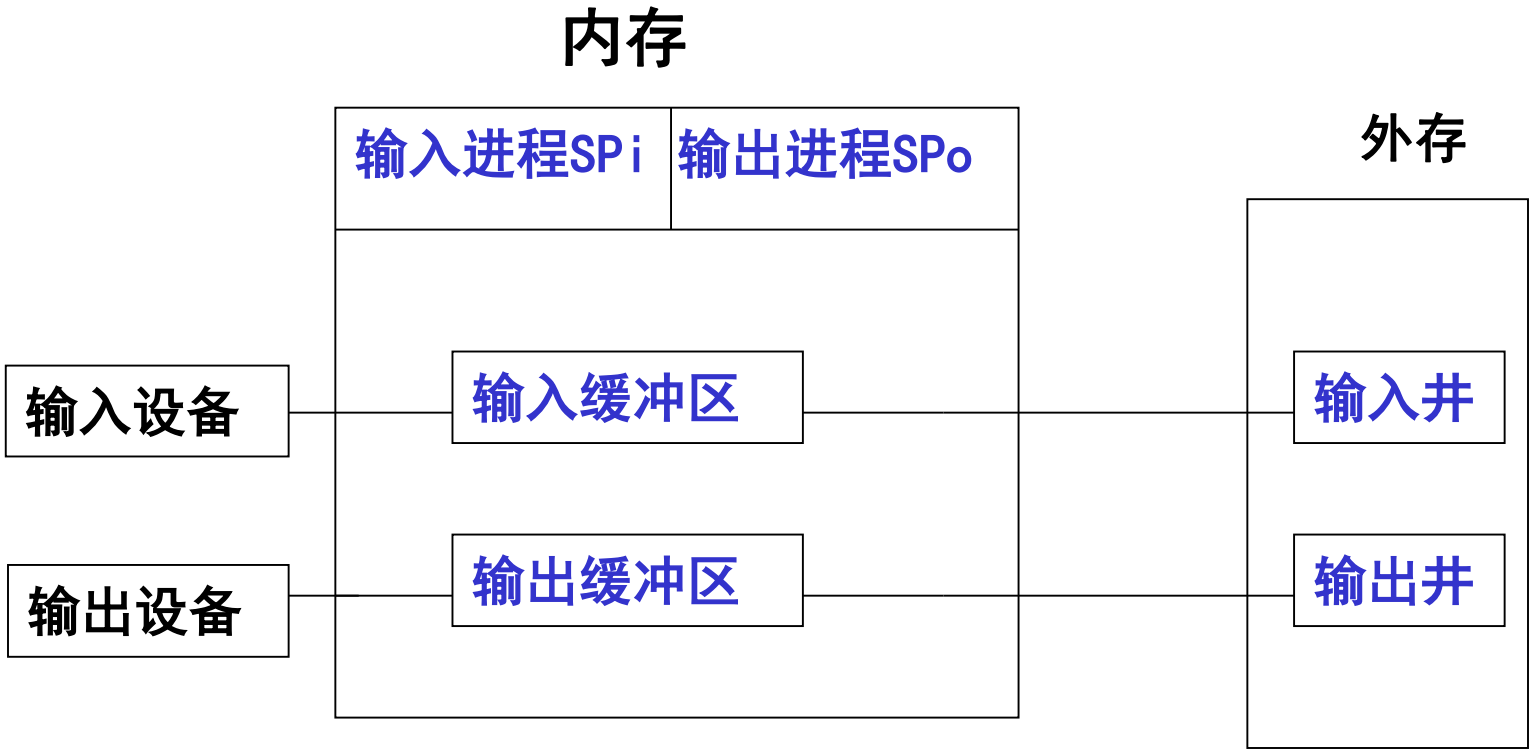
- SPOOLING技术
- SPOOLING系统的组成
- SPOOLING系统的特点
  - 提高了I/O速度
  - 将独占设备改造为共享设备
  - 实现了虚拟设备功能

SPOOLING技术(Simultaneaus Periphernal Operating On-Line)

❖SPOOLING技术

在多道程序下，用一道程序来模拟输入时的外围控制机功能，将低速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输出时的外围控制机功能，将数据从磁盘传送到低速的输出设备上。从而可在主机的直接控制下，实现脱机输入、输出功能，进而实现外围操作与CPU对数据的处理同时进行,这种在联机情况下实现的同时外围操作称为SPOOLING技术，是对脱机输入、输出工作的模拟，是操作系统中采用的一项将独占设备改造成为共享设备的技术。

SPOOLING系统的组成（1）



组成

1、输入井、输出井　　2、输入缓冲区、输出缓冲区

3、输入进程、输出进程　　4、请求打印队列

SPOOLING系统的组成（2）

- SPOOLing技术是对脱机输入输出的模拟，必须有多道程序功能的操作系统和磁盘存储技术的支持
- 输出井和输入井：在磁盘上开辟的两个大的存储空间，模拟脱机输入/输出时的磁盘设备，暂存数据。
- 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，在内存中开辟了两个缓冲区。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。
- 输入进程SPi和输出进程SPo：两个进程来模拟脱机I/O时的外围控制机

SPOOLING技术

- 在操作系统中，引入虚拟设备的原因

  引入虚拟设备是为了克服独占设备速度较慢、降低设备资源利用率的缺点，从而提高设备的利用率。

- 虚拟设备

  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，供若干个用户进程使用，通常把这种经过虚拟技术处理后的设备称为虚拟设备。

SPOOLING技术

举例：共享打印机—SPOOLing技术的典型实例

- 打印机属于独占设备。用SPOOLing技术转换为共享设备，提高设备的利用效率。
- 用户请求打印后：

1. 由输出进程SPo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SPo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。

3. 打印机空闲时，首先取第一张请求表，将数据从输出井传送到内存缓冲区，进行打印。

SPOOLING系统的特点

- 提高了I/O的速度
  - 对数据进行的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
- 将独占设备改造成共享设备
  - 实际上并没有为任何进程分配设备，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
- 实现了虚拟设备功能
  - 宏观上，多个进程同时使用一台独占设备；对每个进程来说，认为是自己独占了一个设备

6.6 磁盘存储器

提高磁盘I/O速度的主要途径：

（1）选择性能好的磁盘

（2）采用好的磁盘调度算法

（3）设置磁盘高速缓存（Disk Cache）

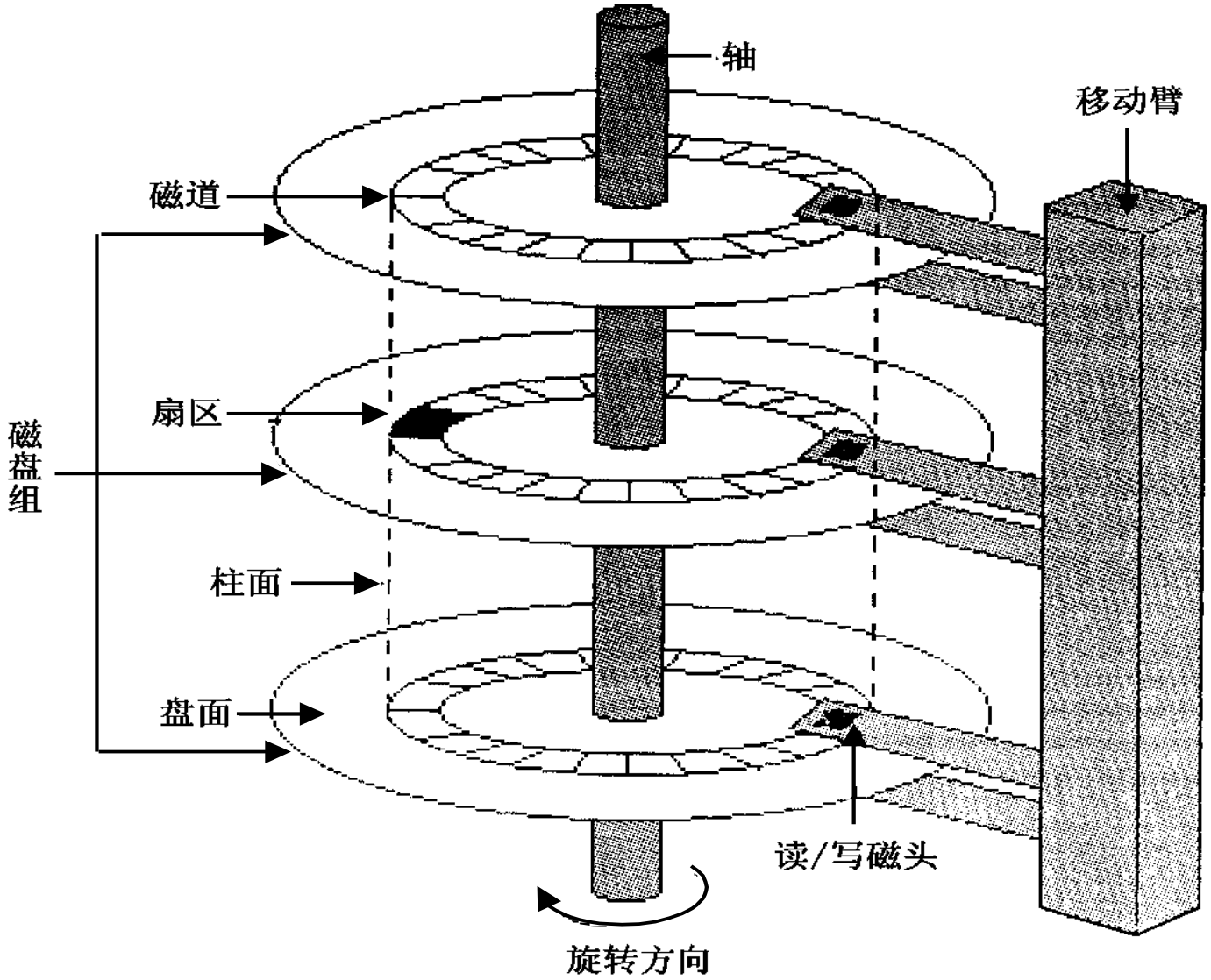
（4）其它方法

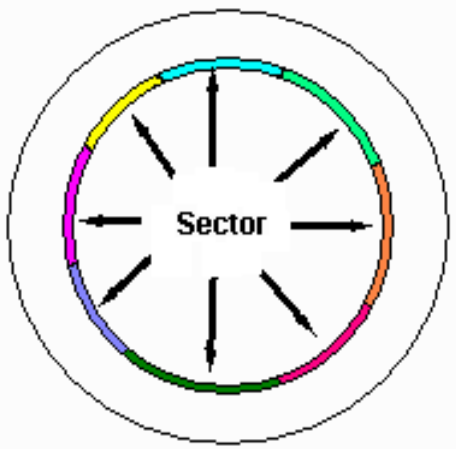
（5）采用高度可靠、快速的容量磁盘系统--廉价磁盘冗余阵列

6.6.1 磁盘性能

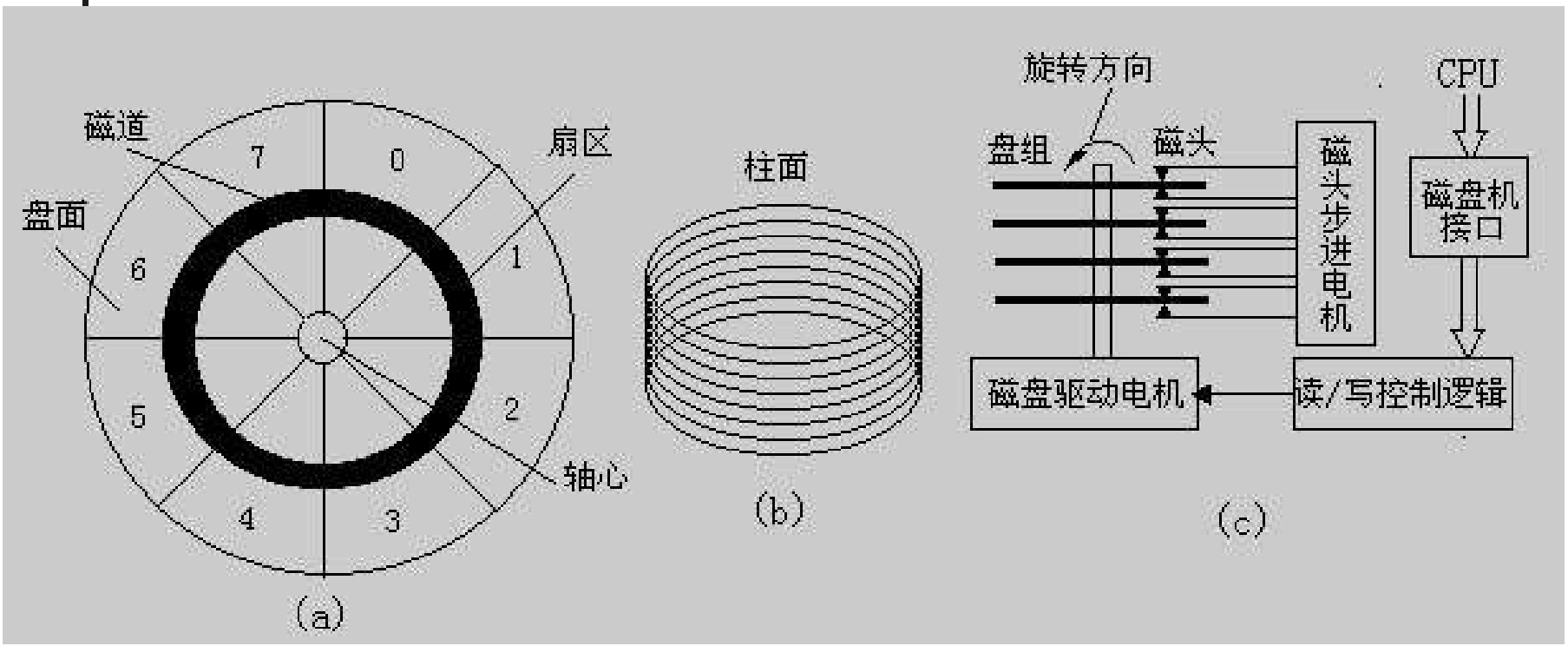
- 磁盘性能简述
  - 数据的组织
    - 磁盘结构：磁道、柱面、扇区、磁盘格式化
    - 磁盘物理块的地址：　柱面号　磁头号　扇区号
  - 磁盘类型（固定头磁盘、移动头磁盘）
  - 磁盘访问时间
    - 寻道时间：将磁头从当前位置移到指定磁道所经历的时间
    - 旋转延迟时间：指定扇区移动到磁头下面所经历的时间
    - 传输时间：将扇区上的数据从磁盘读出/向磁盘写入数据所经历的时间。

磁盘结构（1）





磁盘结构（2）



磁盘调度-当多个进程需要访问磁盘时
磁盘调度的目标是使磁盘的平均寻道时间最少，以使进程对磁盘的平均访问时间最少。

6.6.2 磁盘调度算法

- 磁盘调度算法
  - 早期的磁盘调度算法
    - 先来先服务FCFS
    - 最短寻道时间优先SSTF
  - 扫描算法
    - 扫描(SCAN)算法
    - 循环扫描(CSCAN)算法
    - N-STEP-SCAN调度算法
    - FSCAN调度算法

例：假设一个请求序列：

55, 58, 39, 18, 90, 160, 150, 38, 184　　磁头当前的位置在100。

FCFS 先来先服务

按进程请求访问磁盘的先后次序进行调度。

特点：简单、较合理，但未对寻道进行优化。

FCFS算法（从100#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	

最短寻道时间优先（SSTF-Shortest Seek Time First）

选择从当前磁头位置所需寻道时间最短的请求。

特点：寻道性能比FCFS好，但不能保证平均寻道时间最短，且有可能引起某些请求的饥饿。

SSTF算法（从100#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	

扫描算法（SCAN）（1）

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请求必须优先满足。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。

扫描算法（SCAN）（2）

2)扫描算法

- 磁头从磁盘的一端开始向另一端移动，沿途响应访问请求，直到到达了磁盘的另一端，此时磁头反向移动并继续响应服务请求。有时也称为**电梯算法**。

特点：寻道性能较好，避免了饥饿，但不利于远离磁头一端的访问请求。

（从100#磁道开始，向磁道号增加的方向）

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	

SCAN调度算法示例

循环扫描算法（CSCAN）

规定磁头单向移动

特点：消除了对两端磁道请求的不公平。

（从100#磁道开始，向磁道号增加的方向）

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

N-STEP-SCAN调度算法

- SSTF、SCAN及CSCAN存在的问题---磁臂粘着

  在SSTF、SCAN及CSCAN几种调度算法中，可能出现磁臂停留在某处的情况，即反复请求某一磁道，从而垄断了整个磁盘设备，这种现象称为磁臂粘着。

- N-STEP-SCAN调度算法

  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，而每一子队列按SCAN算法处理。

N=1（每个子队列中只有一个请求）	FCFS算法
N很大（只有一个子队列）	SCAN算法
N取半长度（分成两个子队列）	FSCAN算法

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。

本章练习

1、为什么要设置内存I/O缓冲区？，通常有哪几类缓冲区 ？

2、如何将独占型输入设备改造成可共享使用的虚拟设备？

3、在设备管理中，何谓设备独立性？如何实现设备独立性？

4、SPOOLing系统由哪几部分组成？以打印机为例说明如何利用SPOOLing技术实现多个进程对打印机的共享？

5、设某磁盘有200个柱面，编号为0，1，2，...，199，磁头刚从140磁道移到143磁道完成了读写。若某时刻有9个磁盘请求分别对如下各磁道进行读写：

86，147，91，177，94，150，102，175，130

试分别求FCFS、SSTF及SCAN磁盘调度算法响应请求的次序及磁头移动的总距离。