第六章 输入输出系统

I/O设备
种类繁多

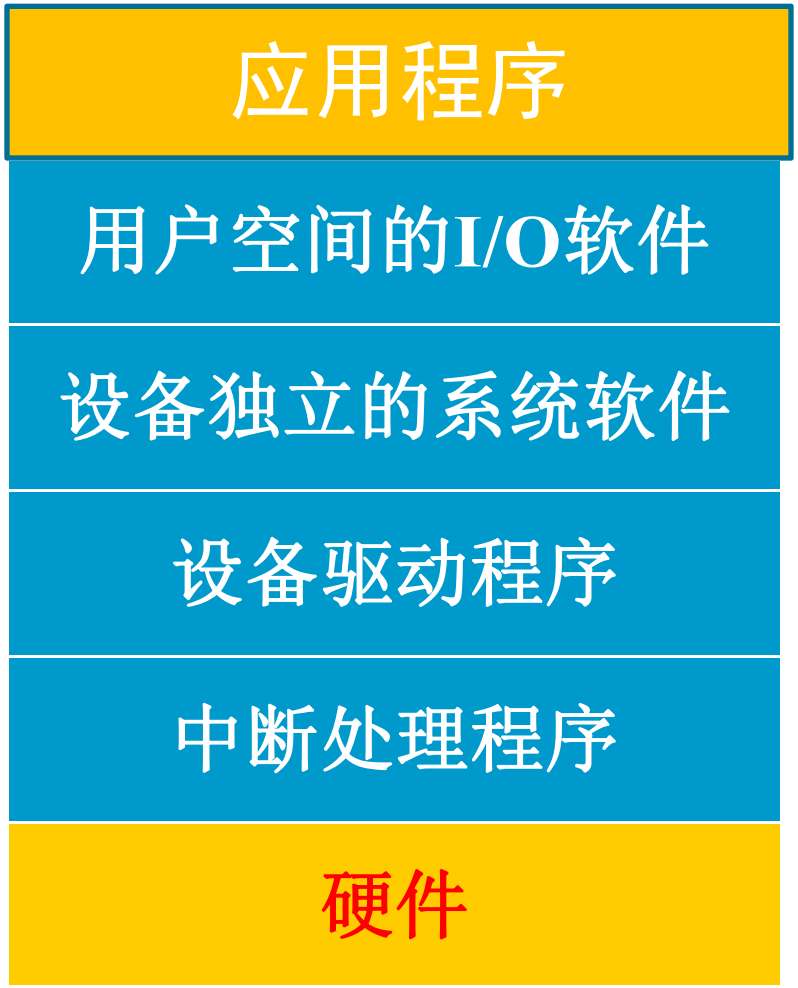
6.1 I/O系统功能模型和接口

I/O系统基本功能

- 隐藏物理设备细节
- 与设备无关性
- 提高处理机和I/O设备利用率
- 对I/O设备进行控制
- 确保对设备的正确共享
- 错误处理

I/O系统层次结构

- I/O 设备管理软件的基本思想是采用分层的结构，把各种设备管理软件组织成一系列的层次。
- 低层与硬件特性相关，它把硬件和较高层的软件隔离开来。
- 一般可分为四层，每一层实现特定的功能，相邻的层次之间有良好的调用接口。



I/O软件系统的层次

- I/O软件的层次结构
  - 用户层软件：API
  - 设备独立性软件：命名、保护、缓冲、分配释放等
  - 设备驱动程序：硬件相关
  - 中断处理程序
  - 例：用户进程从文件读取一个数据块
    - 系统调用－－－设备独立软件 查找页面－－－－如果没有；调用设备驱动程序向硬件请求；驱动程序从磁盘读数据块－－－－磁盘操作完成；硬件产生中断－－－唤醒用户进程结束I/O操作

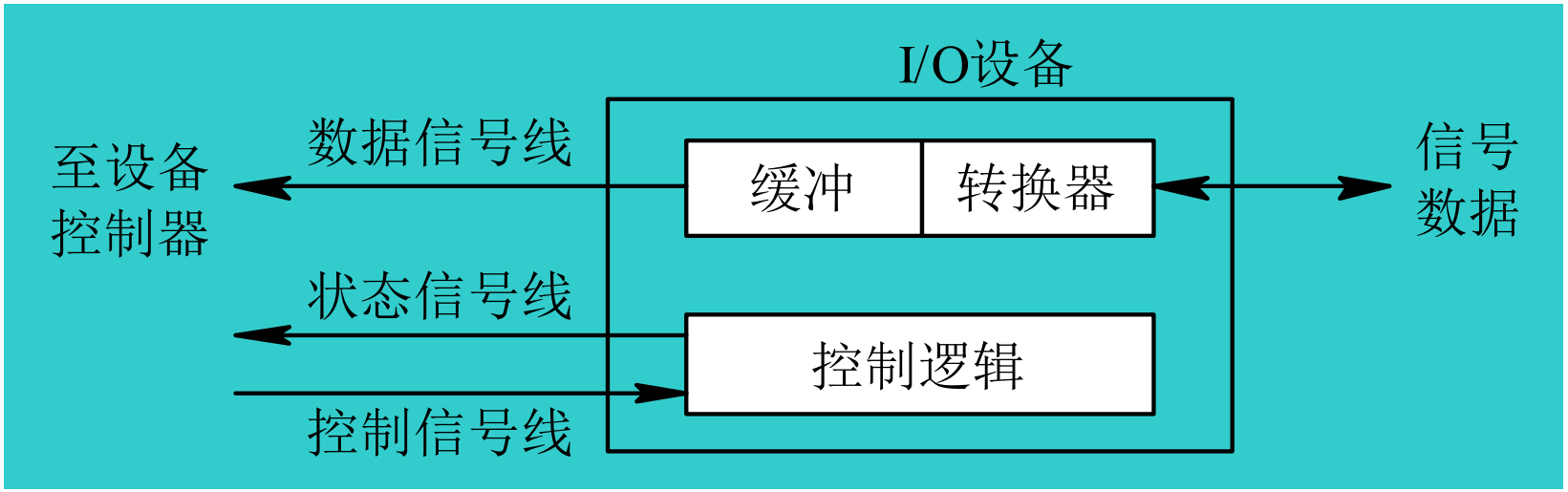
6.2 I/O 设备和设备控制器

- **6.2.1 I/O设备**

  1 I/O设备的类型

  - 按使用特性
    - 存储设备、输入输出设备
  - 按传输速率分类
    - 低速设备：键盘、 鼠标器、语音的输入和输出等设备。
    - 中速设备：打印机。
    - 高速设备：磁带机、 磁盘机、 光盘机等。
  - 按信息交换的单位分类
    - 块设备：磁盘
    - 字符设备

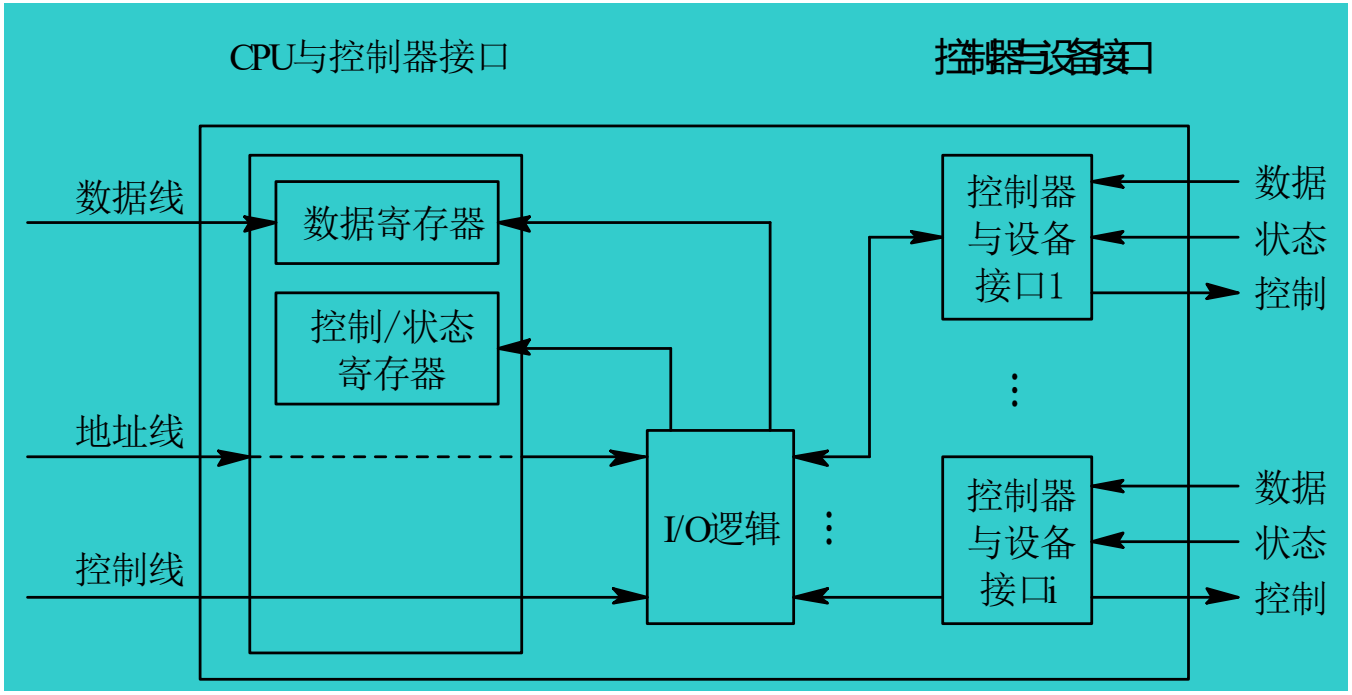
2. 设备与控制器之间的接口



6.2.2 设备控制器

- 1. 设备控制器的基本功能
  - 充当CPU与外设的接口，接受CPU的命令去控制设备工作
  - 1)接收和识别命令
  - 2) 数据交换
  - 3) 标识和报告设备的状态
  - 4) 地址识别
  - 5) 数据缓冲
  - 6) 差错控制

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



6.2.3 内存映像

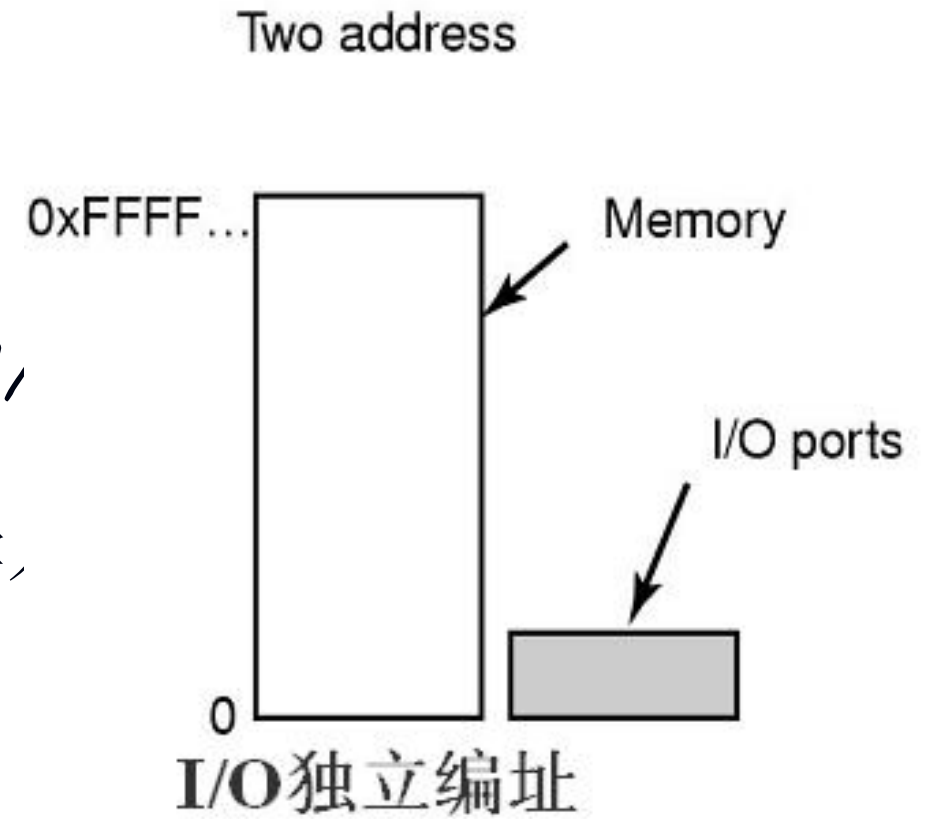
现在的问题是：CPU如何与设备控制器当中的寄存器进行通信？如何访问设备的数据缓冲区？因为这不是普通的内存访问！

方法有三种：

- I/O独立编址；
- 内存映像编址；
- 混合编址。

1. I/O独立编址

- 基本思路：给控制器中的每一个寄存器分配一个唯一的I/O端口（I/O port）编号，称为I/O端口地址，然后用专门的I/O指令对端口进行操作；
- 这些端口地址所构成的地址空间是完全独立的，与内存的地址空间没有关系。例如：
  IN RO [4] 表示读入I/
  端口地址为4的内容；
  MOV RO [4] 表示读
  内存地址为4的内容；



I/O独立编址

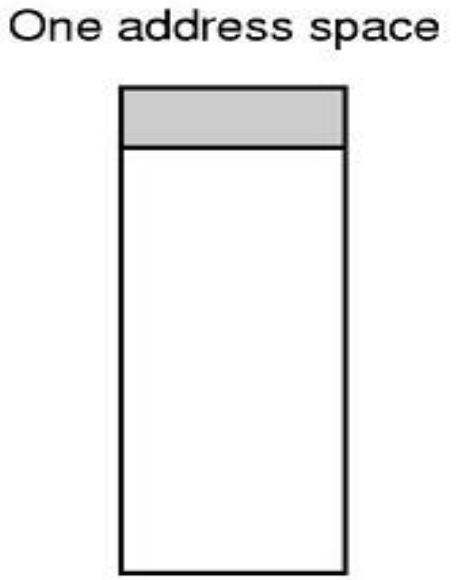
- 优点：I/O设备不占用内存地址空间，而且程序设计时，易于区分是对内存操作还是对I/O端口操作。
- 例子：8086/8088，给I/O端口分配的地址空间64K，0000H~FFFFH，只有IN和OUT指令进行读写操作。

Linux0.11/boot/setup.s

```
mov al,#0x11    ! initialization sequence
out #0x20,al    ! send it to 8259A-1
mov al,#0x20    ! start of hardware int's(0x20)
out #0x21,al
mov al,#0x28    ! start of hardware int's(0x28)
out #0xA1,al
.....
in al,#0x64     ! 8042 status port
                ! 键盘控制器状态寄存器
test al,#2
jnz empty_8042  ! is input
```

2. 内存映像编址

- 基本思路：把所有控制器当中的每一个寄存器都映射为一个内存地址，专门用于*I/O*操作（功能上），对这些单元的读写操作即为普通的内存访问操作。
- 端口地址空间与内存的地址空间统一编址，前者是后者的一部分，一般位于后者的顶端部分。



内存映像编址

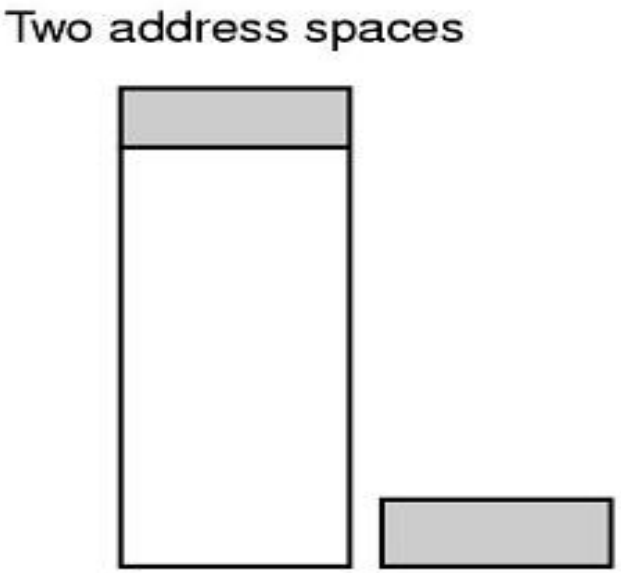
优点：

1 编程方便，无需专门的I/O指令；

2 对普通的内存单元可进行的所有操作指令均可作用于I/O端口，如TEST指令；

3. 混合编址

- 基本思路：对于设备控制器中的寄存器，采用独立编址的方法；而对于设备的数据缓冲区，采用内存映像编址的方法。
- 例如：Pentium，把内存地址空间640K~1M保留作为设备的数据缓冲区，另外，还有一个独立的I/O端口地址空间，从0到64K。

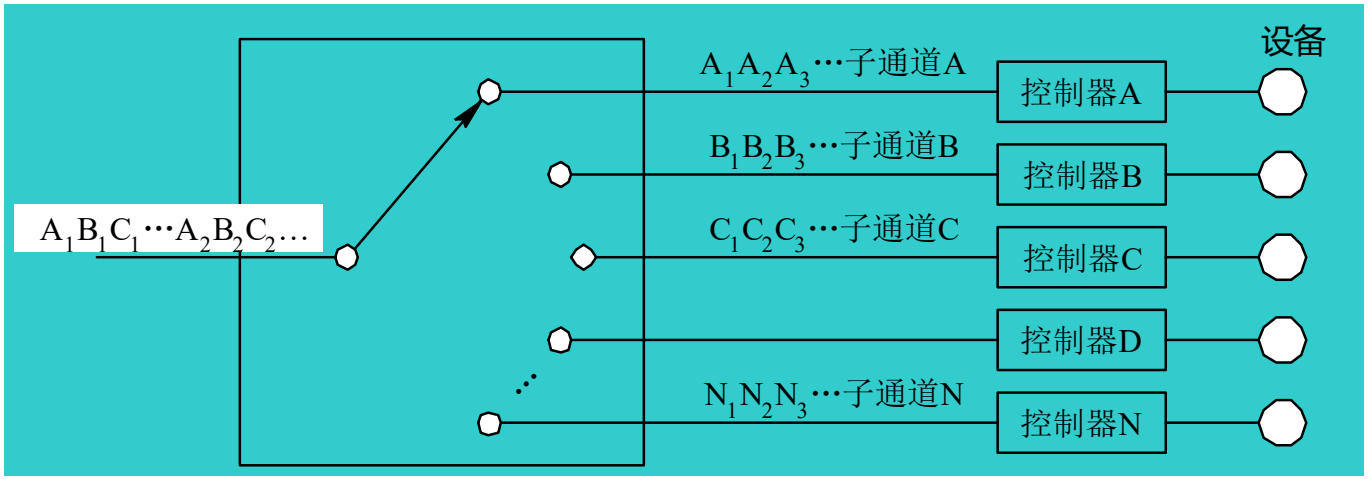


混合编址

6.2.4 I/O通道

- 1. I/O通道(I/O Channel)设备的引入
  - 是一种特殊的处理机
    - 具有执行I/O指令的能力，并通过执行通道(I/O)程序来控制I/O操作。
  - 与一般的处理机不同
    - 指令类型单一
    - 通道没有自己的内存，通道所执行的通道程序是放在主机的内存中的， 换言之，是通道与CPU共享内存。

- 2. 通道类型
  - 根据信息交换方式不同
  - 1) 字节多路通道(Byte Multiplexor Channel)
    - 子通道以时间片轮转方式使用主通道
  - 2) 数组选择通道(Block Selector Channel)
    - 一个分配型通道；在一段时间只有一台设备传送数据
  - 3) 数组多路通道(Block Multiplexor Channel)



6.3 中断机构和中断处理程序

- 中断
  - 中断：CPU对I/O设备发来的中断信号的一种响应,暂停当前程序，执行中断处理程序。
  - 陷入：CPU内部事件引起。
- 中断向量表和中断优先级
  - 中断程序入口地址表
  - 不同中断类型可能有不同优先级
- 多中断源处理方式
  - 屏蔽
  - 嵌套

6.3 中断机构和中断处理程序

- 中断处理程序
  - 测试是否有未响应中断信号（每条指令执行完成后测试）
  - 保护被中断的CPU环境
  - 转入相应设备处理程序
  - 中断处理
  - 回复现场并退出中断

6.4 设备驱动程序

- 设备驱动程序的功能
  - 接收设备独立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
  - 检查I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，设置设备工作方式
  - 发出I/O命令
  - 响应控制器或通道的中断请求，处理中断

设备驱动程序

- 设备驱动程序特点
  - 请求IO的进程与设备控制器之间的通信和转换程序。IO请求传给控制器；控制器记录的设备状态信息传给请求IO进程
  - 硬件相关
  - 部分固化
  - 可重入
  - 与I/O控制方式相关

设备驱动程序

- 设备驱动程序的处理过程
  - 1) 将抽象要求转为具体要求
  - 2) 检查IO请求的合法性
  - 3) 读出和检查设备的状态
  - 4) 传送必要的参数
  - 5) 工作方式的设置
  - 6) 启动IO设备
  - 设备控制器控制IO操作，驱动程序阻塞，直到设备中断唤醒。

I/O设备的控制方式

1　程序I/O方式

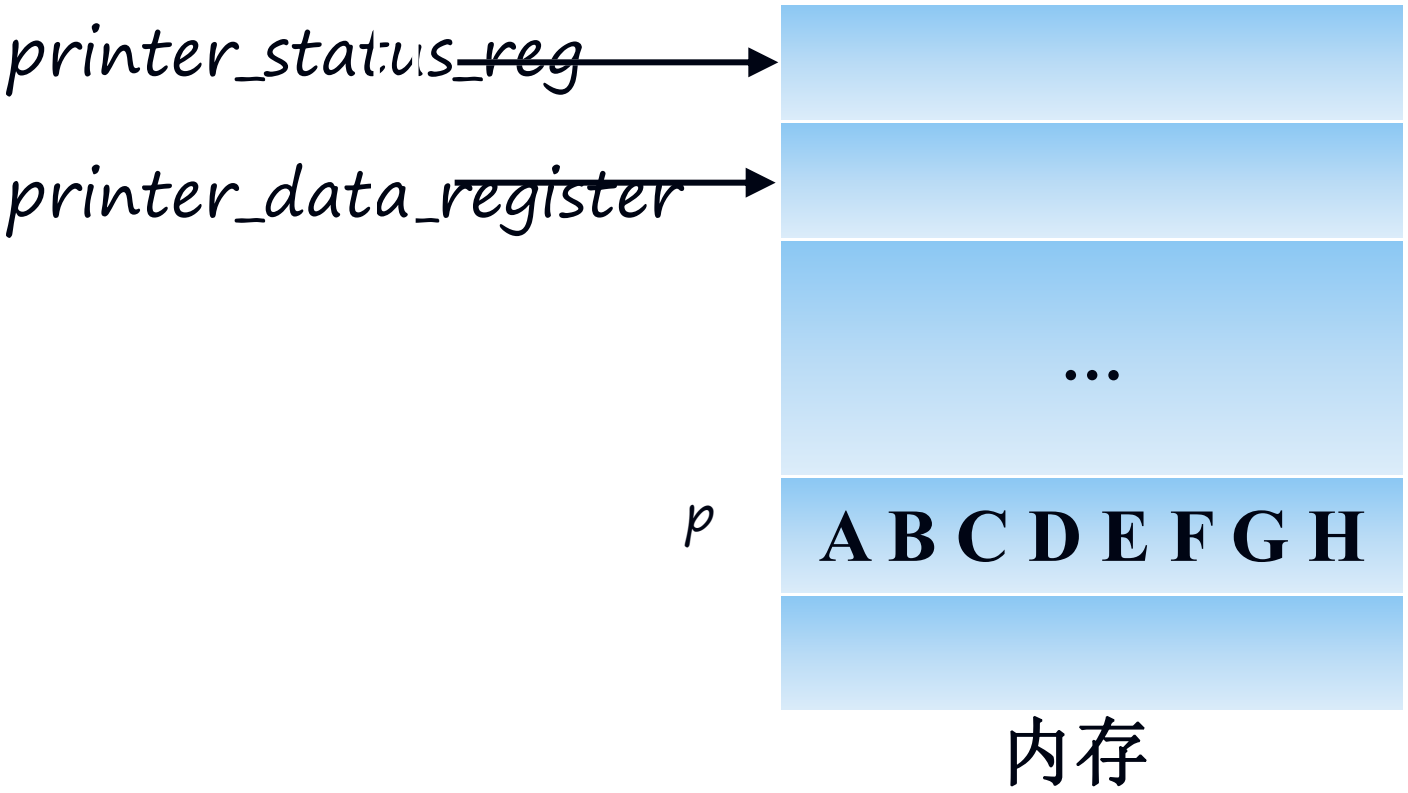
CPU不断测试状态寄存器中的忙闲状态.

忙–等待方式

一个例子

已知I/O地址采用内存映像编址的方式，现需要在打印机上打印一个字符串“ABCDEFGH”。

基本思路：把这8个字符逐个送到打印机设备的I/O端口地址（内存地址）。



```
for (i = 0; i < count; i++)
{
        while(*printer_status_reg != READY);
        *printer_data_register = p[i];
}
```

逐个打印每一个字符：先判断打印机是否空闲，若不空闲则循环等待。然后把第 i 个字符复制给打印机的数据寄存器（内存单元）。

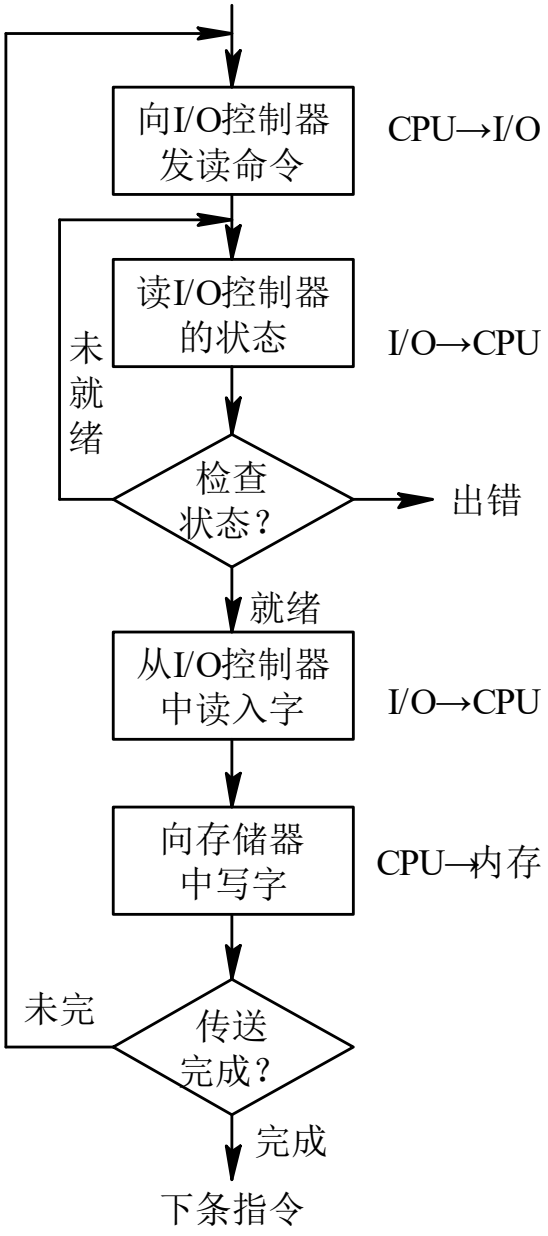
中断驱动方式

循环检测的控制方法占用了太多的CPU时间，可能会造成CPU时间的浪费。例如：假设打印机的打印速度为100字符/秒，在循环检测方式下，当一个字符被写入到打印机的数据寄存器中后，CPU需要等待10毫秒才能写入下一个字符。

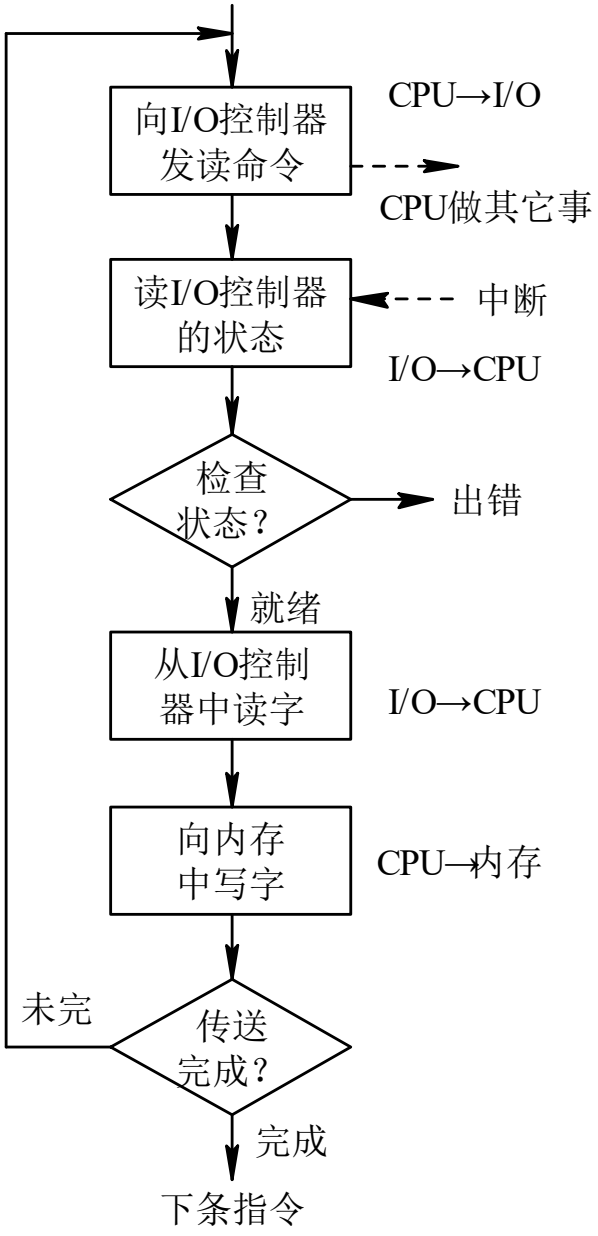
一种解决的办法：中断驱动的控制方式

用户进程

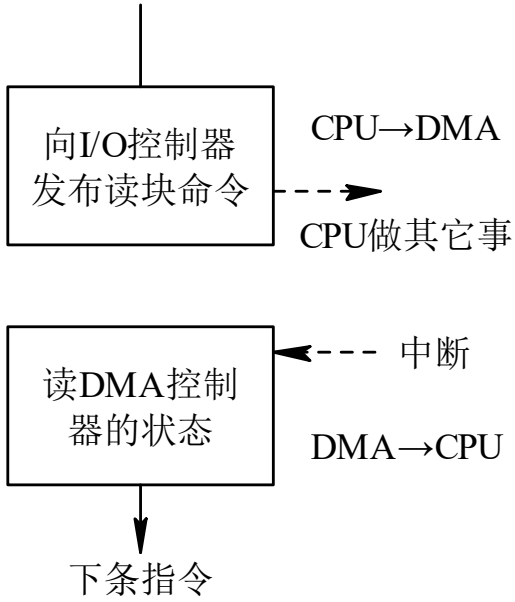
```
strcpy(buffer, “ABCDEFGH”);
print(buffer, strlen(buffer));
```



(*a*) 程序I/O方式



(*b*) 中断驱动方式



(*c*) DMA方式

在I/O设备输入每个数据的过程中，由于无须CPU干预，因而可使CPU与I/O设备并行工作。

仅当输完一个数据时，才需CPU花费极短的时间去做些中断处理。

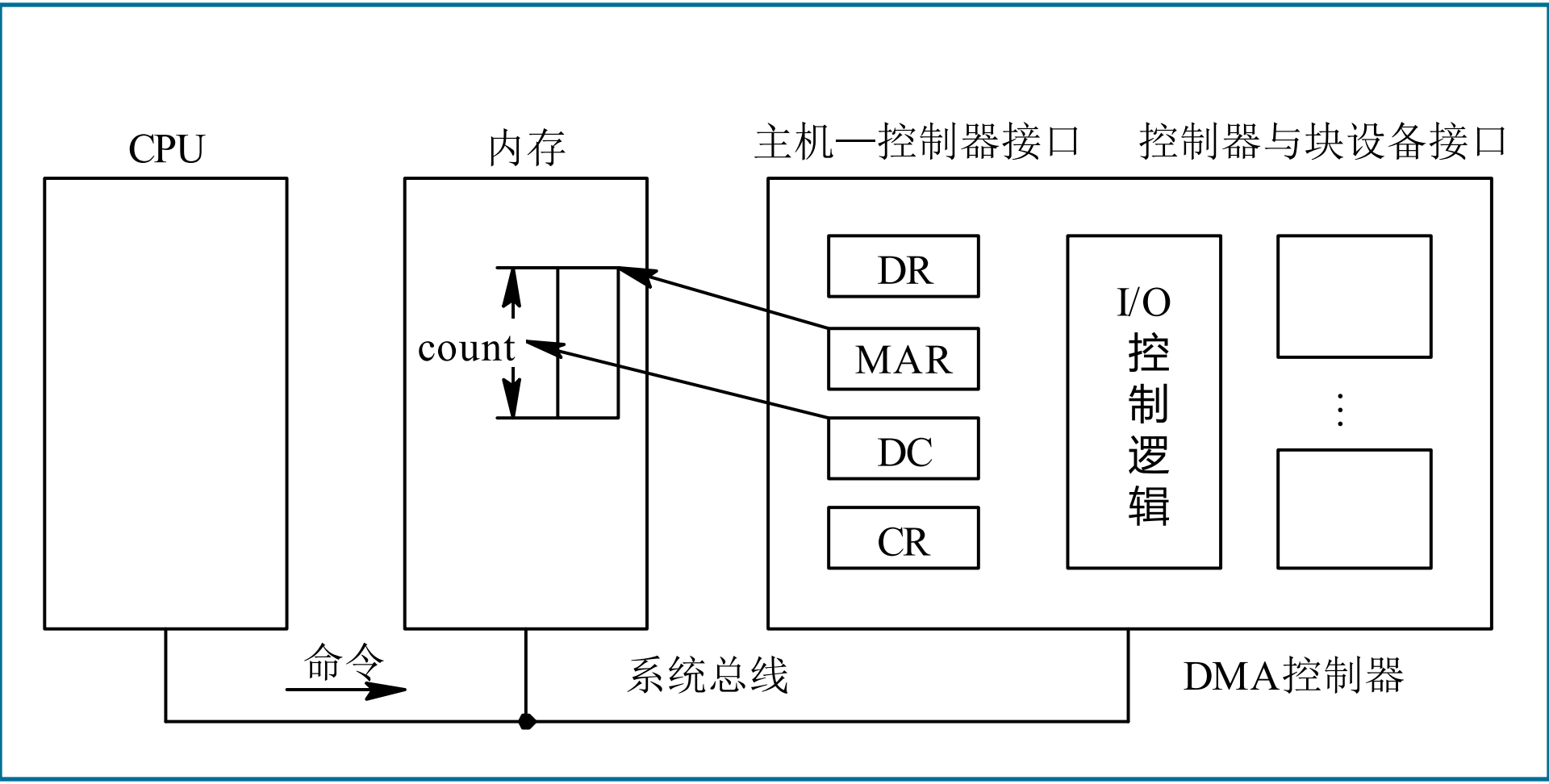
例如，从终端输入一个字符的时间约为100 ms， 而将字符送入终端缓冲区的时间小于 0.1 ms。 若采用程序I/O方式，CPU约有 99.9 ms的时间处于忙–等待中。 采用中断驱动方式后，CPU可利用这 99.9 ms的时间去做其它事情，而仅用0.1 ms的时间来处理由控制器发来的中断请求。

直接存储器访问DMA I/O控制方式

1. DMA(Direct Memory Access)控制方式的引入

该方式的特点是：① 数据传输的基本单位是数据块，即在CPU与I/O设备之间，每次传送至少一个数据块；② 所传送的数据是从设备直接送入内存的，或者相反； ③ 仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在控制器的控制下完成的。可见，DMA方式较之中断驱动方式，又是成百倍地减少了CPU对I/O的干预，进一步提高了CPU与I/O设备的并行操作程度。

2. DMA控制器的组成



DMA控制器的组成

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

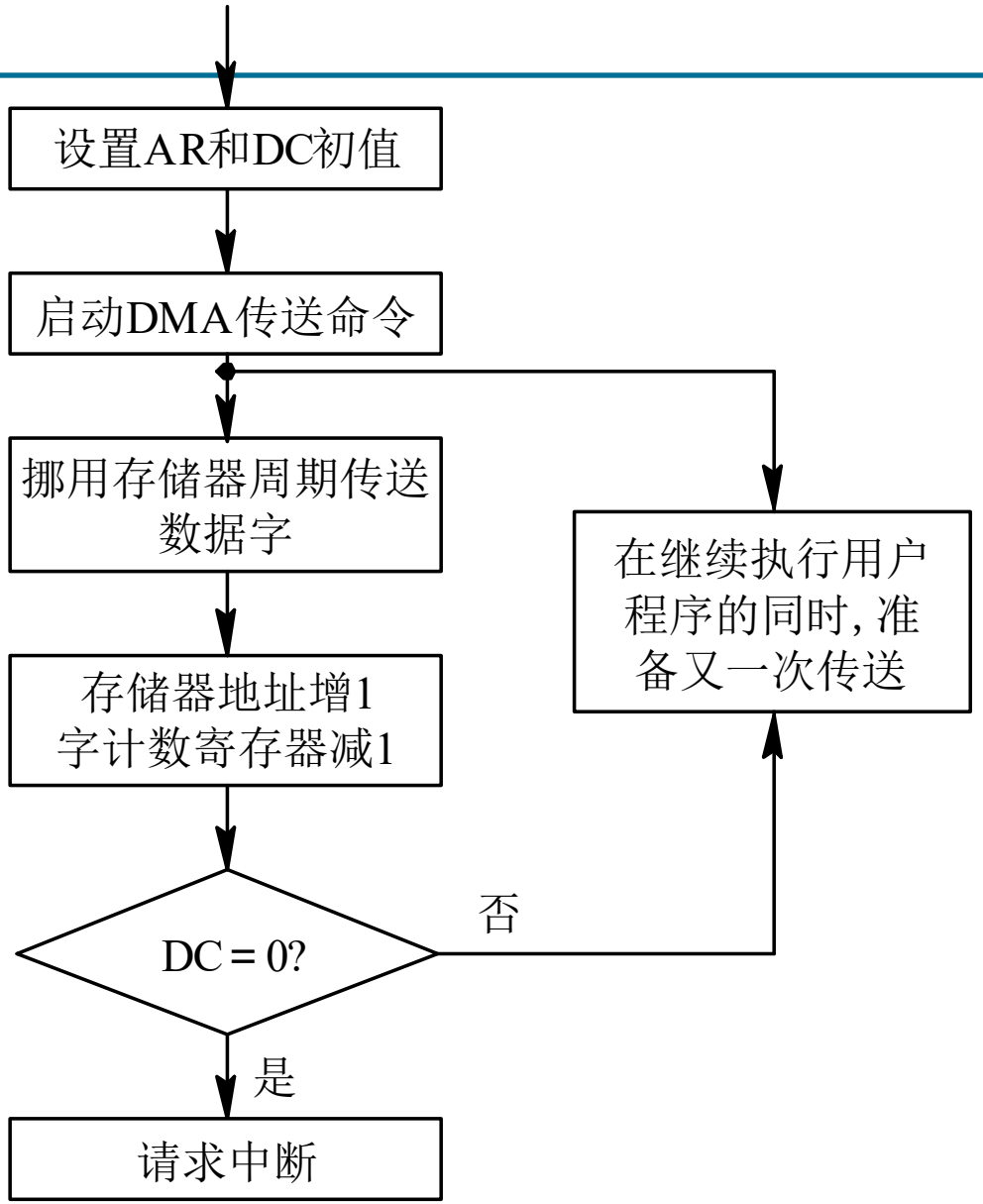
(1) 命令/状态寄存器CR。用于接收从CPU发来的I/O命令或有关控制信息， 或设备的状态。囍

(2) 内存地址寄存器MAR。在输入时，它存放把数据从设备传送到内存的起始目标地址；在输出时，它存放由内存到设备的内存源地址。囍

(3) 数据寄存器DR。用于暂存从设备到内存，或从内存到设备的数据。囍

(4) 数据计数器DC。 存放本次CPU要读或写的字(节)数。

3. DMA工作过程



DMA方式的工作流程

I/O通道控制方式

1. I/O通道控制方式的引入

- I/O通道方式是DMA方式的发展，进一步减少CPU的干预

把对一个数据块的读(或写)为单位的干预，减少为对一组数据块的读(或写)及有关的控制和管理为单位的干预。 实现CPU、通道和I/O设备三者的并行操作

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

2. 通道程序

(⑴) 操作码。

(2) 内存地址。

(3) 计数。

(4) 通道程序结束位P。

(5) 记录结束标志R。

操作	P	R	计数	内存地址
WRITE	0	0	80	813
WRITE	0	0	140	1034
WRITE	0	1	60	5830
WRITE	0	1	300	2000
WRITE	0	0	250	1850
WRITE	1	1	250	720

6.5 设备独立的I/O软件

早期以物理设备名使用设备

引入逻辑设备　如 /dev/printer

需要实现逻辑设备名到物理设备名转换

设备独立性软件

- 设备独立性软件
  - 驱动程序为硬件相关，为实现硬件无关而设置
  - 1) 执行所有设备的公有操作
    - ① 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。
  - 2) 向用户层(或文件层)软件提供统一接口

提供与设备无关的数据块大小

磁盘的访问是以扇区为单位，但不同的磁盘可能会有不同的扇区大小，因此，设备独立的I/O软件可以向上层掩盖这一事实，并提供统一的逻辑块大小，例如，它可以将若干个物理扇区合并成一个逻辑块，这样，对于上层的软件来说，它们所面对的都是一些抽象的设备，这些设备都使用相同大小的逻辑块。

设备分配

设备分配用数据结构

- 记录设备和控制器的状态及有关控制信息
- 设备控制表Device Control Table
  - 每个设备
  - 记录设备特性、连接情况
- 控制器表 （Controler Control Table）
  - 每个控制器
- 通道控制表（Channel Control Table）
  - 通道
- 系统设备表（System Device Table）
  - 整个系统

设备分配时考虑的因素

- 1 设备的固有属性
  - 独占、共享、可虚拟设备（独占变为共享）
- 2 设备分配算法 (FCFS, PF)
- 3 设备分配中的安全性
  - 安全分配方式
    - 进程请求IO后阻塞。CPU与IO串行
  - 不安全分配方式
    - 进程请求IO后还可以继续请求其它IO
    - 如申请的设备不可用才阻塞
    - 可能死锁

独占设备的分配程序

- 1　基本的设备分配程序
  - 某进程提出IO请求后
  - 1）　分配设备
    - 根据物理设备名，查找系统设备表SDT，找出该设备的DCT
    - 根据DCT中的设备状态字，可知该设备是否忙碌
    - 如忙，请求进程的PCB挂在设备队列
    - 否则，检查安全性，决定是否分配
  - 2）　分配控制器
    - 设备分配给进程后，从DCT中找到与该设备连接的控制器的COCT
    - 从COCT的状态字段判断该控制器是否忙碌，从而决定进程是等待还是分配控制器给进程

独占设备的分配程序

- 1　基本的设备分配程序
  - 3） 分配通道
    - 从COCT中找到与该控制器连接的通道的CHCT，
    - 判断状态决定是否分配
  - 设备、设备控制器、通道，三者均分配成功，才可是可以启动传输。
  - 缺点：
    - 使用物理设备名；单通道
- 2 设备分配程序的改进
  - 1)增加设备的独立性
    - 使用逻辑设备名：在SDT中找到第一个DCT，如忙，查找下一个DCT
  - 2) 考虑多通路情况
    - 第一个控制器或通道忙时，查找下一个控制器及通道

- 逻辑设备名到物理设备名映射的实现
  - 1) 逻辑设备表 Logic Unit Table
  - 2) LUT的设置
    - 整个系统一张LUT
    - 每个用户一张LUT

逻辑设备名	物理设备名	驱动程序入口地址
/dev/tty	3	1024
/dev/printer	5	2046
⋮	⋮	⋮

(*a*)

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

(*b*)

6.6 用户层的I/O软件

虽然大多数的I/O软件都包含在操作系统中，但也有一小部分是与用户程序进行链接的库函数，或者是完全运行在用户空间的程序。

- 库函数：如C语言里与I/O有关的库函数write、read等，它们实质上只是将它们的参数再传递给系统调用函数，并由后者来完成实际的I/O操作；
- Spooling技术：在多道系统中，一种处理独占设备的方法。

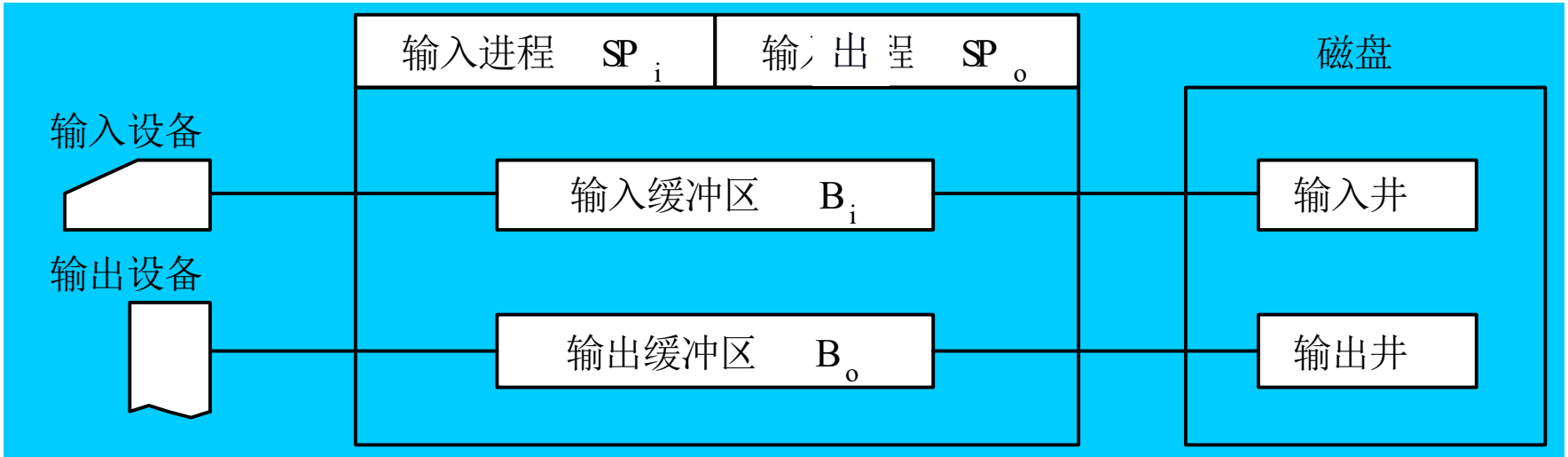
SPOOLING技术

什么是SPOOLing囍

- 缓和CPU的高速性与I/O设备低速性间的矛盾
- 脱机输出技术
  - 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。
- 当系统中引入了多道程序技术后，利用其中的一道程序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，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。

SPOOLING技术

- SPOOLing系统的组成
  - 输入井、输出井
    - 磁盘
  - 输入缓冲区、输出缓冲区
    - 缓和CPU和磁盘间速度不匹配问题
    - 内存中暂存输入输出数据，以便和输入井、输出井传递
  - 输入进程、输出进程
    - 模拟外围控制机



SPOOLING技术

- SPOOLing系统的特点
  - 提高了I/O的速度
    - 改CPU对低速外设操作为磁盘操作
  - 将独占设备改造为共享设备
  - 实现了虚拟设备功能
    - 多个进程同时使用一台独占设备，但对每个进程而言，感觉自己独占设备

SPOOLING技术

- 共享打印机
  - 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
  - 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；
  - 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。
  - 打印机空闲时，输出进程从请求打印队列的队首取出一张请求打印表，根据表内信息，将数据从输出井传到内存缓冲区，再由打印机打印。
  - 重复上一个过程直至请求打印队列空。输出进程自我阻塞，下次打印请求将其唤醒。

6.7 缓冲管理

缓冲的引入

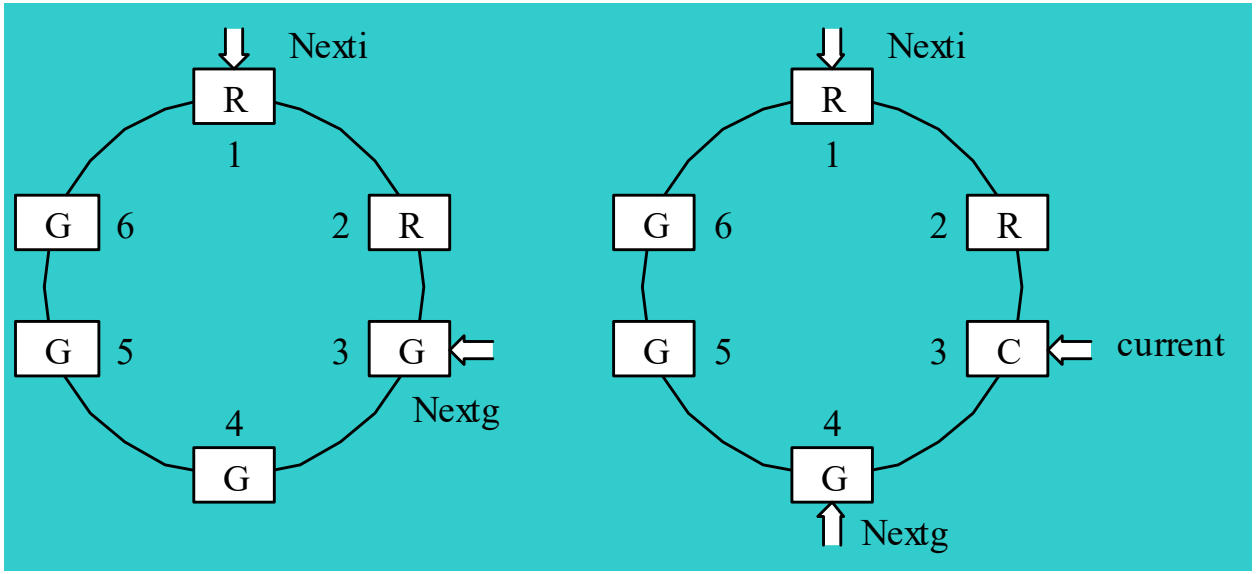
- (1) 缓和CPU与I/O设备间速度不匹配的矛盾。
- (2) 减少对CPU的中断频率， 放宽对CPU中断响应时间的限制。
- (3) 提高CPU和I/O设备之间的并行性。

单缓冲和双缓冲

- 1. 单缓冲(Single Buffer)
- 2. 双缓冲(Double Buffer)

循环缓冲

- 1. 循环缓冲的组成
  - 多个缓冲区：空缓冲区R、有数据的缓冲区G、计算进程正在使用的缓冲区C
  - 多个指针：计算进程下一个可用缓冲区的指针Nextg、输入进程下一个可用空缓冲区的指针Nexti、计算进程正在使用的指针Current



循环缓冲

- 2. 循环缓冲区的使用
  - Getbuf
    - 计算进程
      - Nextg指针指向的缓冲区可使用
      - 将其改为现行工作缓冲区，令Current指针指向该缓冲区
      - Nextg下移至下一个G缓冲区
    - 输入进程
      - Nexti指针指向的缓冲区供使用使用
      - Nexti下移至下一个R缓冲区
  - Releasebuf
    - 计算进程
      - 计算进程取走缓冲区C的数据后，释放C。
      - 把C改为R

- 3 进程同步
  - Nexti指针追赶上Nextg指针
    - 输入数据速度大于大于计算进程处理速度
    - 输入进程阻塞
  - Nextg指针追赶上Nexti指针
    - 计算进程阻塞

缓冲池

- 公共缓冲池
  - 多个进程共享的缓冲区
  - 1. 缓冲池的组成
    - ① 空(闲)缓冲区；
    - ② 装满输入数据的缓冲区；
    - ③ 装满输出数据的缓冲区。
    - 为了管理上的方便，将相同类型的缓冲区链成队列

缓冲池

- 2 缓冲池操作
  - 队列操作：进队、出队
  - 临界资源、同步互斥
  - MS互斥信号量
  - RS资源信号量

```
Procedure Getbuf(type)囍
  begin囍
    Wait(RS(type));囍
    Wait(MS(type));囍
    B(number)坱：=琊Takebuf(type);
囍
    Signal(MS(type));囍
   end囍
 Procedure Putbuf(type, number)囍
  begin囍
   Wait(MS(type));囍
   Addbuf(type, number);囍
   Signal(MS(type));囍
   Signal(RS(type));囍
  end
```

缓冲池

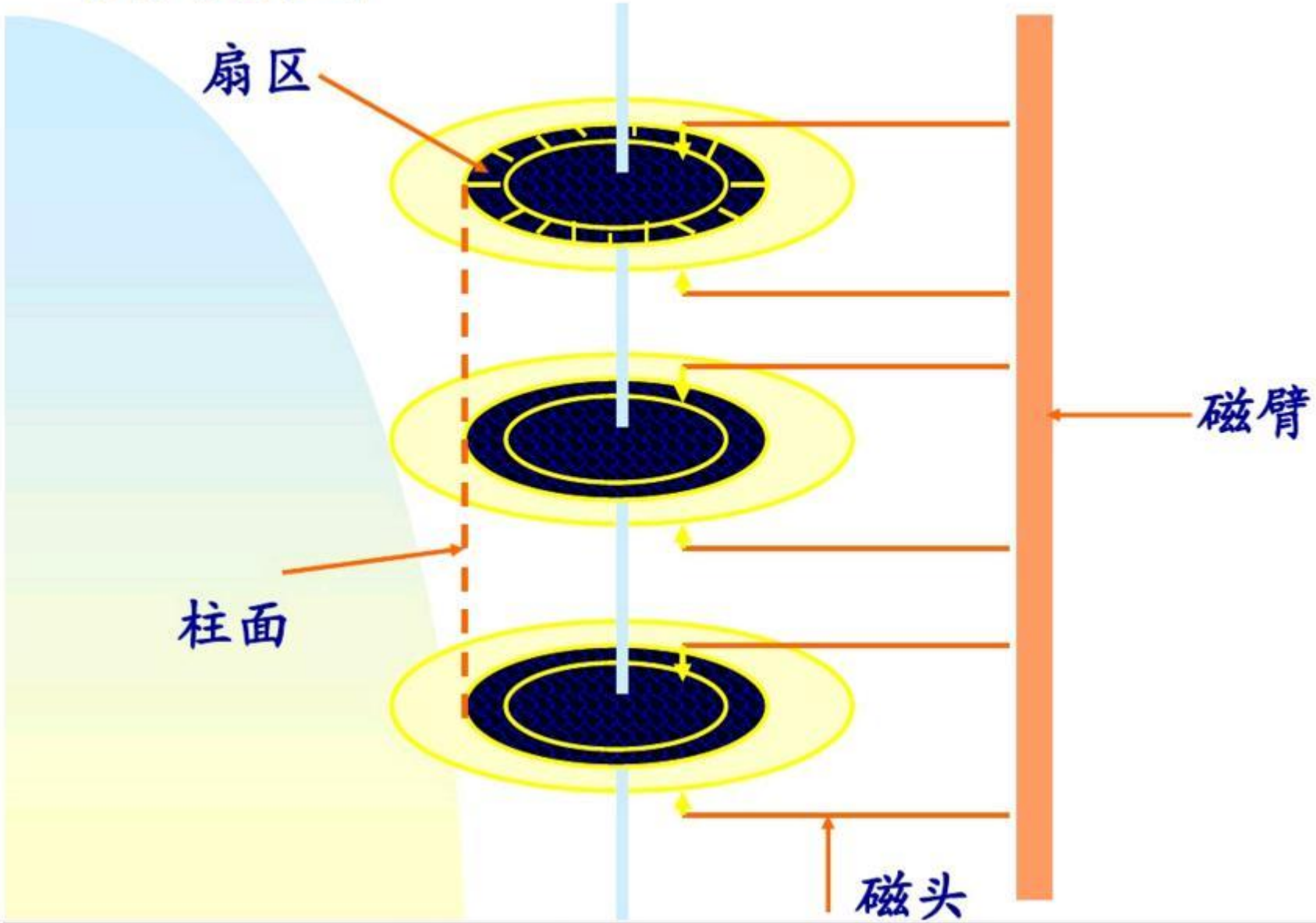
- 3 缓冲区的工作方式
  - 收容输入
    - 输入进程需要输入数据时
    - 取一空缓冲区；将其作为收容输入缓冲区；输入数据到其中；
    - 将该缓冲区挂到输入队列
  - 提取输入
    - 计算进程需要输入数据时
    - 从输入队列取缓冲区；作为提取输入工作缓冲区
    - 提取数据；将缓冲区挂到空缓冲区队列
  - 收容输出
    - 计算进程需要输出数据时
    - 取一空缓冲区；将其作为收容输出缓冲区；输出数据到其中；
    - 将该缓冲区挂到输出队列
  - 提取输出
    - 输出进程从输出队列取以有数据的缓冲区，作为提取输出缓冲区；
    - 取数据，将空缓冲区挂在空缓冲队列队尾；

6.8 磁盘存储器的性能和调度

1　磁盘性能简述

- 1　数据的组织和格式
  - 盘片 —— 磁道 —— 扇区
  - 低级格式化
    - 划分扇区
  - 分区
    - 每个分区为一个独立的逻辑磁盘
    - 0扇区中有主引导记录分区表
    - 有一个分区为活动分区有硬盘引导记录
  - 高级格式化
    - 设置引导块、空闲存储管理、根目录、文件系统

侧视图



磁盘性能简述

- 2. 磁盘的类型
  - 固定头磁盘囍
    - 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
    - 速度快，主要用于大容量磁盘上。囍
  - 移动头磁盘囍
    - 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。
    - 移动磁头仅能以串行方式读/写
    - 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。

磁盘性能简述

- 3. 磁盘访问时间
  - 工作时，磁盘旋转，磁头移动到欲读数据所在磁道，等待数据所在扇区旋转到磁头下，然后开始读数据
  - 寻道时间、旋转延迟时间、传输时间
  - 1) 寻道时间Ts鍏冨
    - 把磁臂(磁头)移动到指定磁道上所经历的时间
    - 为启动磁臂的时间s与磁头移动n条磁道所花费的时间之和， 即珚珚
    - Ts=m×n+s
    - 其中，m是一常数，与磁盘驱动器的速度有关，对一般磁盘，m=0.2；对高速磁盘，m≤0.1,琢磁臂的启动时间约为2 ms。 这样，对一般的温盘， 其寻道时间将随寻道距离的增加而增大，大体上是5~30 ms。

- 3. 磁盘访问时间
  - 2) 旋转延迟时间Tτ鍏暠
    - 指定扇区移动到磁头下面所经历的时间。
    - 对于硬盘，典型的旋转速度大多为5400 r/min，每转需时11.1 ms，平均旋转延迟时间Tτ为5.55 ms；对于软盘，其旋转速度为300 r/min或600 r/min，这样，平均Tτ为50~100 ms。

- 3. 磁盘访问时间
  - 传输时间Tt鎮菖
    - 指把数据从磁盘读出或向磁盘写入数据所经历的时间。 Tt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
    - 访问时间Ta表示为：

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

- 3. 磁盘访问时间
  - 寻道时间和旋转延迟时间与读写数据量无关
  - 寻道时间和旋转延迟时间占主要时间
  - 集中数据传输，有利于提高传输效率

2　磁盘调度

- 问题描述
  - 多个进程要求访问磁盘时，采用合理的访问次序，使各进程对磁盘的平均访问时间最小。
  - 主要考虑寻道时间
- 1. 先来先服务FCFS(First-Come, First Served)
  - 简单公平
  - 平均寻道时间可能较长，适用于请求磁盘IO进程数目较少的场合

（从 100 号磁道开始）	
被访问的下一个磁道号	移动距离（磁道数）
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度：55.3	

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

- 2. 最短寻道时间优先SSTF(Shortest Seek Time First)
  - 访问与当前磁头所在磁道距离最近的磁道
  - 每次寻道时间最短，但不能保证平均寻道时间最短

（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	

- 3 扫描（SCAN）算法
  - 1) 进程“饥饿”现象
    - SSTF算法中，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，其请求必须优先满足，可导致某些老进程出现“饥饿”现象。
  - 2) SCAN算法
    - 除考虑与当前磁道的距离外，优先考虑磁头当前的移动方向，即选择当前移动方向最近的磁道，直到该方向无访问，磁头换方向移动
    - 电梯调度算法

- 3 扫描（SCAN）算法

（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

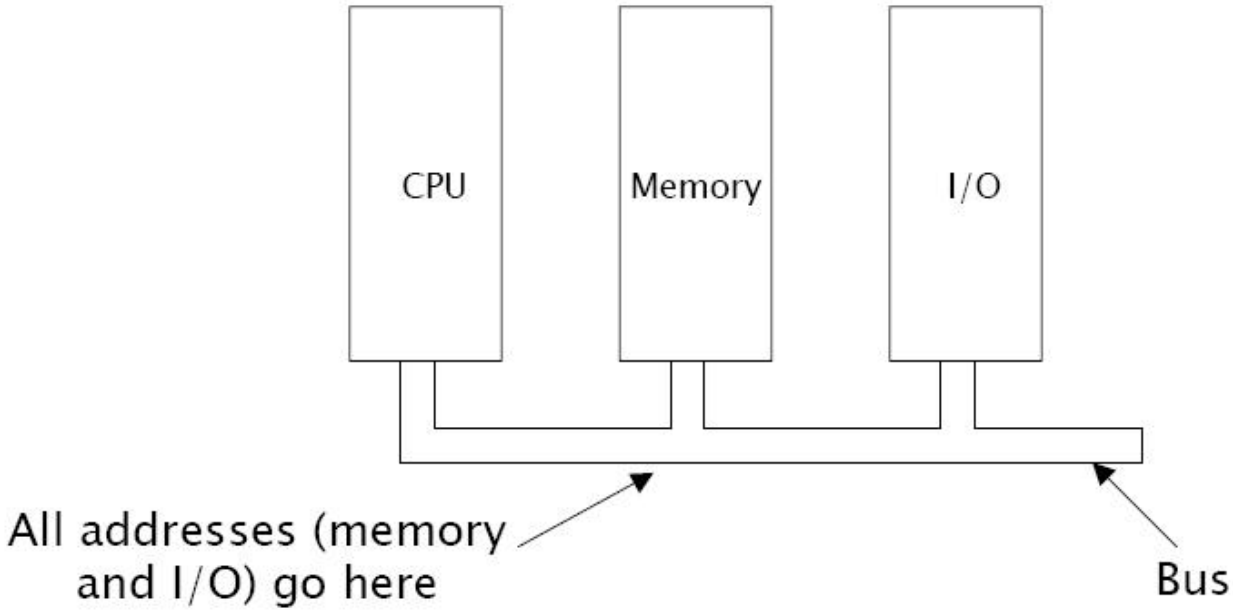
- 4. 循环扫描(CSCAN)算法
  - 磁头单向移动；最大磁道号和最小磁道号构成循环
  - 即磁头自里向外移动，访问最外的磁道后，回到最里的待访问磁道

（从100#磁道开始，向磁道号增加方向访问）	
被访问的下一个磁道号	移动距离（磁道数）
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度：27.5	

- 5. N-Step-SCAN和FSCAN调度算法
  - 1) N-Step-SCAN算法
    - 磁臂粘着(Armstickiness)
      - 在SSTF、 SCAN及CSCAN几种调度算法中， 都可能出现磁臂停留在某处不动的情况， 例如，有一个或几个进程对某一磁道有较高的访问频率， 即这个(些)进程反复请求对某一磁道的I/O操作，从而垄断了整个磁盘设备。
    - 将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。
    - 每处理一个队列时又是按SCAN算法，对一个队列处理完后，再处理其他队列。 当正在处理某子队列时，如果又出现新的磁盘I/O请求，便将新请求进程放入其他队列，这样就可避免出现粘着现象。
    - 当N值取得很大时，会使N步扫描法的性能接近于SCAN算法的性能； 当N=1时， N步SCAN算法便蜕化为FCFS算法。

- 5. N-Step-SCAN和FSCAN调度算法
- 2) FSCAN算法囍
  - 是N步SCAN算法的简化，
  - 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。

- 有一个磁盘队列，其I/O对各个柱面上的请求顺序为98、183、37、122、14、124、65、67，如果磁头开始位于53，是从63柱面移至的。
- FCFS, SSTF, SCAN, CSCAN


CPU
Memory
I/O
All addresses (memory and I/O) go here
Bus