第7章设备管理

一、设备管理概述

1.设备(Device)

- 含义
- I/O操作

2.设备分类

- 按设备的用途分类：存储设备，输入/输出设备
- 按设备的传输速度分类：慢速设备，快速设备
- 按设备的数据组织分类：
  - 字符设备(Character Device)
  - 块设备(Block Device)
- 按设备的管理分类：物理设备，逻辑设备
- 按设备的固有属性分类
  - 独占设备
  - 共享设备
  - 虚拟设备

3.设备独立性(Device Independence)

- 系统资源
- PnP技术(Plug-and-Play)
- I/O软件的层次结构

用户或程序中使用的设备与具体的物理设备无关，用户或程序使用的是逻辑设备，当进程运行时，由操作系统在逻辑设备与物理设备之间建立连接，即设备的分配，把这种设备使用方法的特点称为设备独立性。

设备独立性是计算机操作系统能够方便用户使用计算机的一种体现。

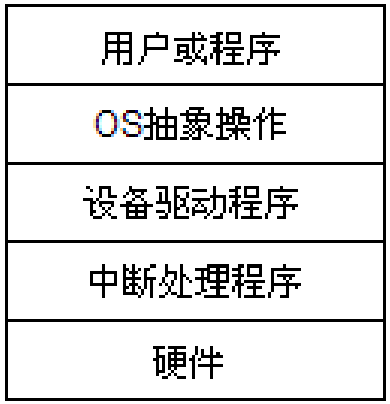

图7-1 I/O软件的层次结构

4.设备管理的主要功能

- 设备的数据传输控制
- 缓冲技术
- 设备分配
- 磁盘驱动调度

二、I/O控制方式

1.程序查询方式
2.中断方式
3.DMA方式
4.通道方式

通道命令 ,通道程序

通道地址字(CAW)

通道状态字(CSW)

通道与处理器协作过程?

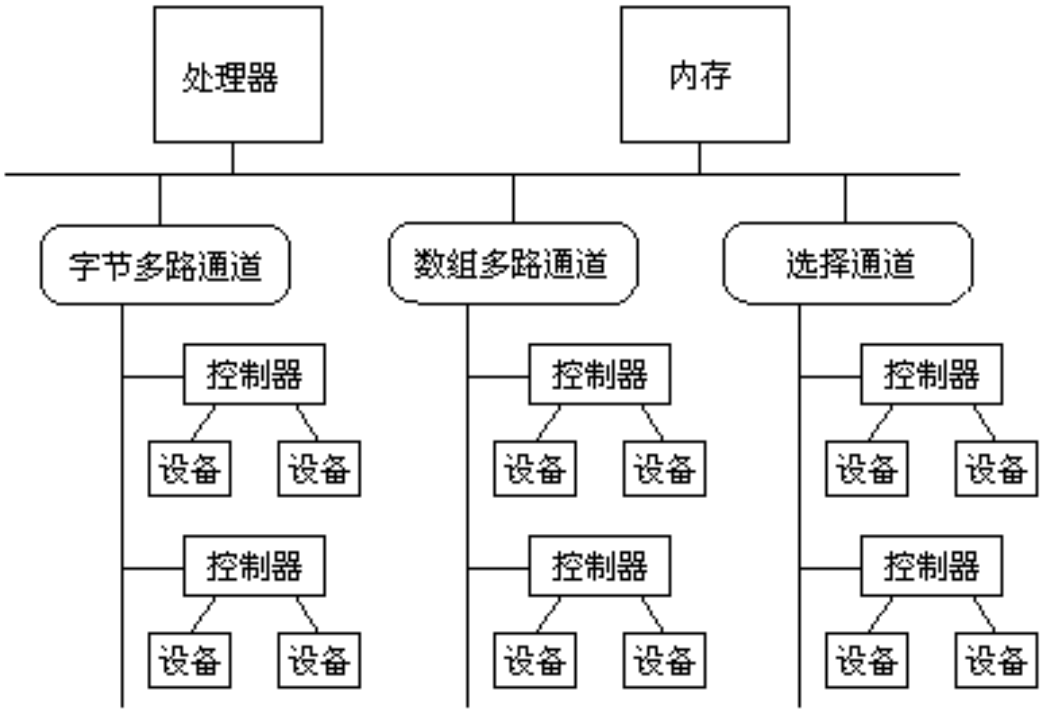

图7-14 具有通道的计算机系统的结构示意图

三、设备分配

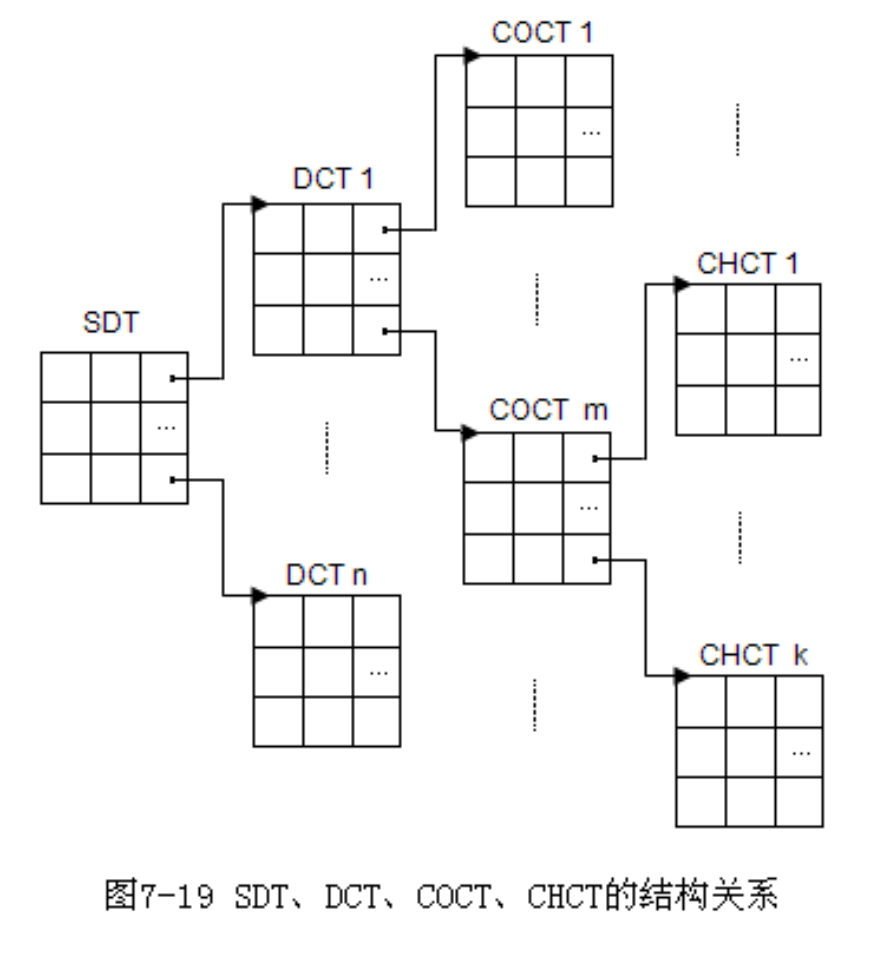

图7-19 SDT、DCT、COCT、CHCT的结构关系

设备分配原则：

1. 提高的并行性
2. 方便用户使用
3. 系统的安全性

设备分配的安全性

设备分配方式分静态分配和动态分配两种， 静态分配保证系统的安全性

动态分配又分为：单请求方式和多请求方式，单请求方式保证系统的安全性

在多请求方式中，为了解决死锁问题，一种方法是综合第4.4.3小节介绍的“剥夺资源”和“资源暂时释放”的预防方法，其思想是：

当一个进程Pi申请新资源R得不到满足时，检查占有新资源R的进程Pj，比较Pi和Pj的进程标识符(pid)：

如果Pi的进程标识符更小，则允许Pi阻塞，

否则，选择如下之一进一步处理：

1)Pi终止，或Pi归还已经得到资源后进入阻塞状态

2)Pi抢占Pj的资源R

这样，可以破坏“环路等待”条件，实现死锁的预防。

四、缓冲技术

1.什么缓冲技术

2.引入缓冲的目的

- ◆ 缓解设备和处理器之间的速度不匹配的矛盾，提高系统工作的并行程度
- ◆ 减少I/O操作的次数
- ◆ 减少中断次数
- ◆ 提高系统的及时性，方便用户操作

3.缓冲类型

- ◆ 硬件缓冲和软件缓冲
- ◆ 软件缓冲分为
  - 单缓冲
  - 双缓冲
  - 缓冲池

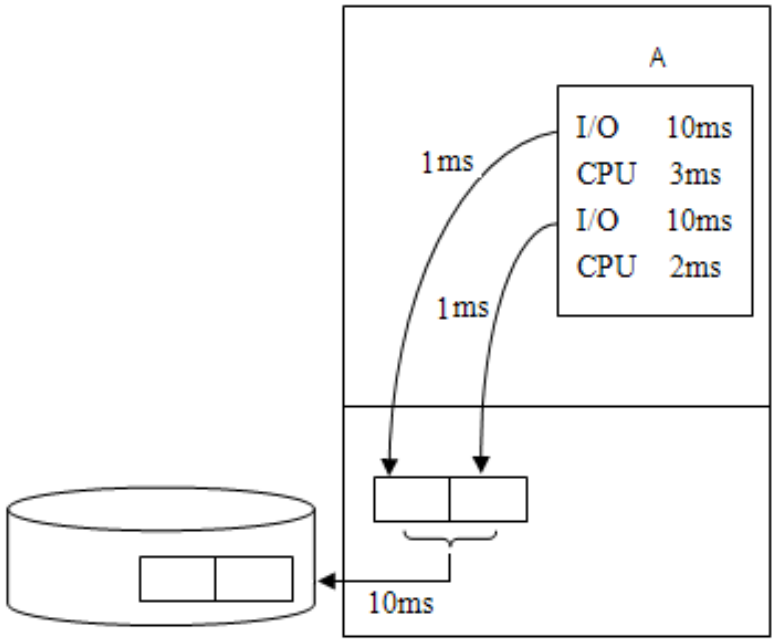

图7-20 缓冲技术例子

3.缓冲池管理

- 空缓冲队列(em),输入缓冲队列(in)和输出缓冲队列(out)

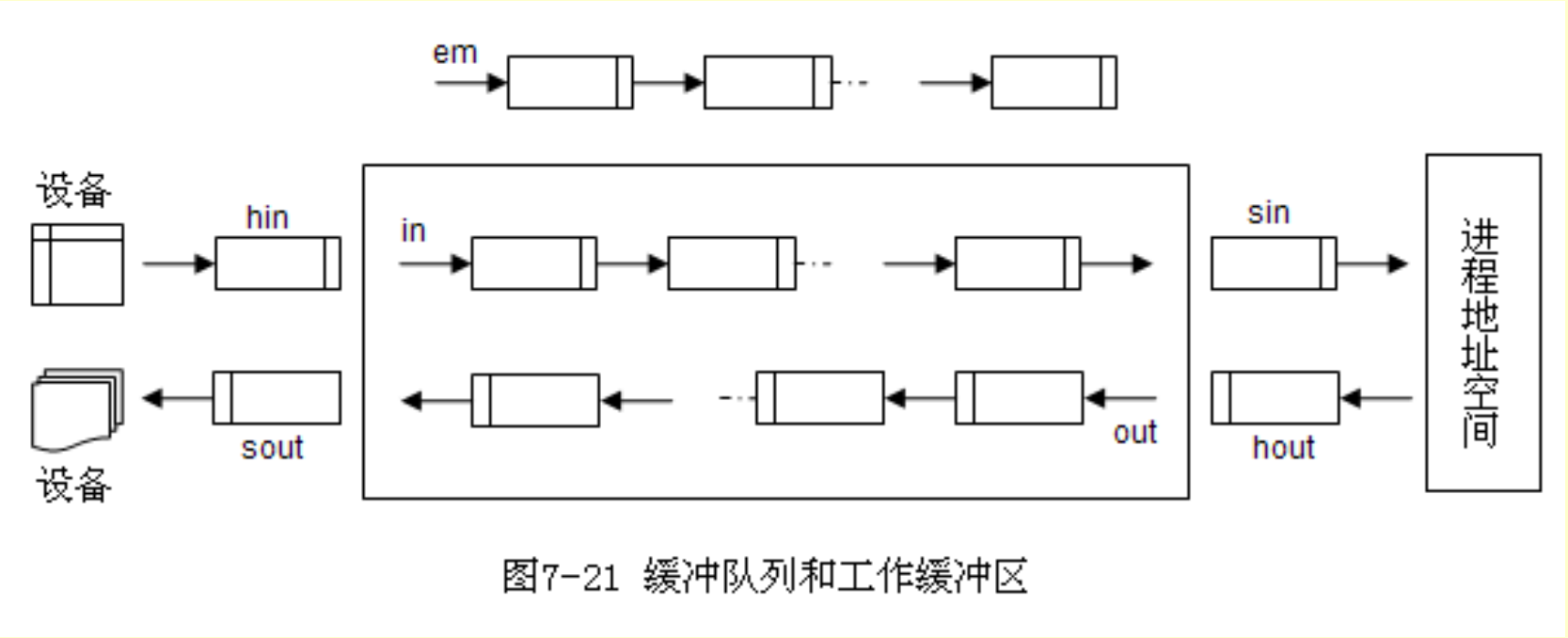

图7-21 缓冲队列和工作缓冲区

缓冲池管理的设计:

同步信号量数组RS[type]

互斥信号量数组MS[type]

其中type=em、in或out

定义一组缓冲队列的操作：

1)take_buf(type)

功能：从type指示的缓冲队列中按一定策略取出一个缓冲区；

输入：参数type指示缓冲队列：em、in、out；

输出：返回一个缓冲区。

2)add_buf(type,buf)

功能：将缓冲区buf按一定策略加入type指示的缓冲区队列中；

输入：参数type指示缓冲队列：em、in、out；

参数buf指要加入队列的缓冲区；

输出：无。

```
get_buf(type)
{
   p(RS[type])；
   p(MS[type])；
   buf = take_buf(type)；
   v(MS[type])；
   return buf。
}
```

```
put_buf(type,buf)
{
   p(MS[type])；
   add_buf(type,buf)；
   v(MS[type])；
   v(RS[type])；

}
```

输入I/O操作的缓冲池实现

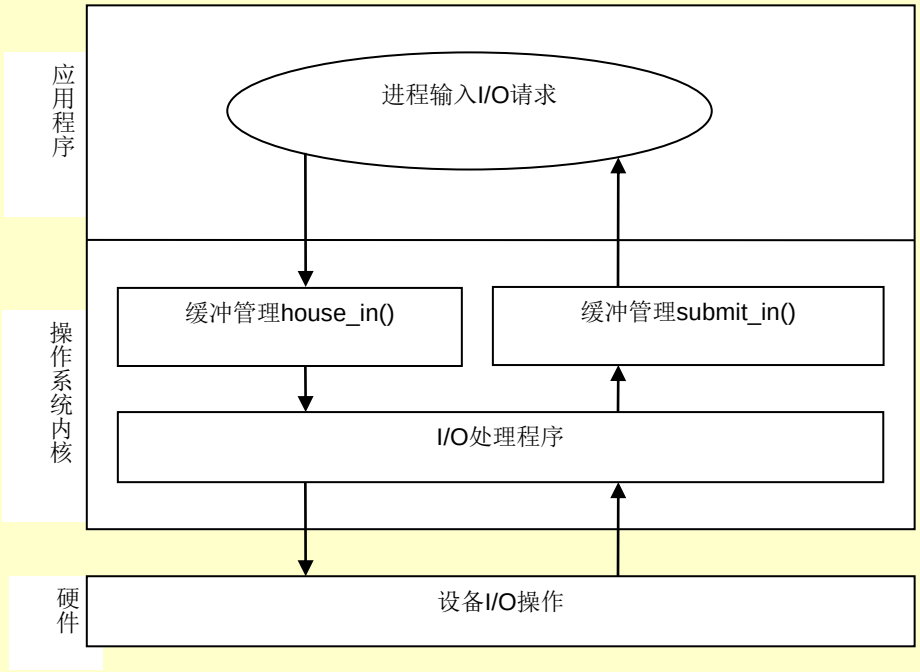

图7-22 输入I/O操作过程的缓冲技术

```
house_in(){
收容输入缓冲区hin=get_buf(em)；
将用户的输入请求和缓冲区hin，提交给内核的I/O处理程序。
}
```

```
submit_in()
{
  提取输入缓冲区sin=get_buf(in)；
  将缓冲区sin中的输入数据复制到对应请求进程的地址空间；
  唤醒输入I/O请求进程；
  put_buf(em，sin)；
}
```

输出I/O操作的缓冲池实现

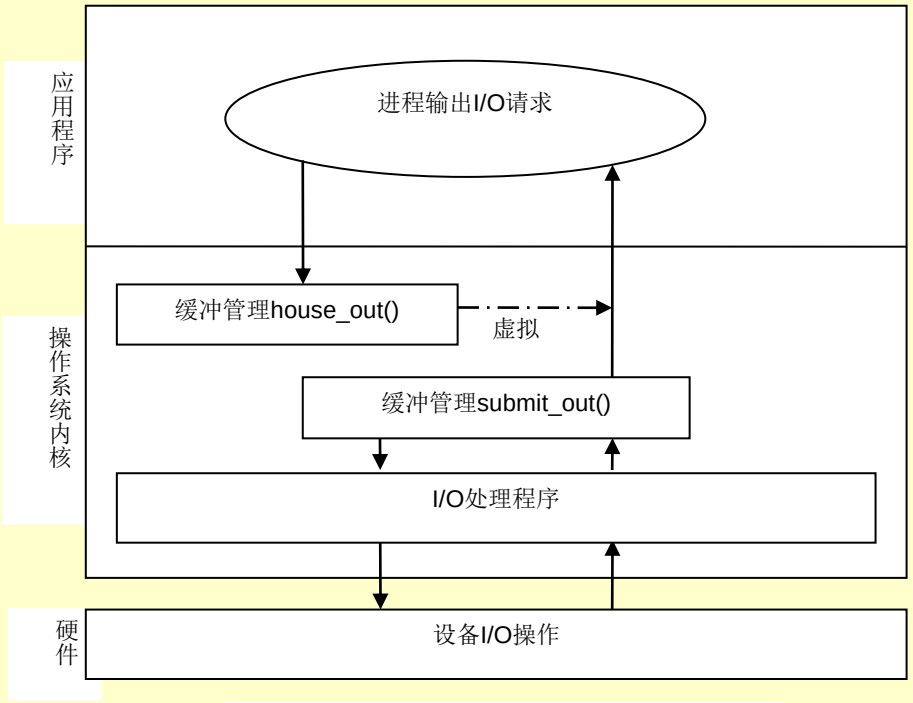

图7-23 输入I/O操作过程的缓冲技术

```
house_out(){
 收容输出缓冲区hout=get_buf(em)；
 将用户的输出数据复制到hout；
 put_buf(out,hout)；
}
```

```
submit_out(){
while(1){
  提取输出缓冲区sout=get_buf(out)；
  调用内核的I/O处理程序，
  将缓冲区 数据输出指定设备。
  输出I/O操作完成。
  put_buf(em，sout)；
  在没有采用虚拟设备时唤醒sout
   对应的用户进程。
   }
}
```

五、磁盘驱动调度

1.磁盘I/O操作的时间组成

- 寻道时间Ts
- 旋转延迟时间Tr
- 传输时间Tt

2.磁盘驱动调度

- 移臂调度(Disk Arm Scheduling Algorithm)
- 旋转调度

3.移臂调度算法

- 先来先服务算法(FCFS)

例7-1 假定某磁盘的一组I/O操作的访问请求，它们的请求提出顺序依次是：55、72、100、88、93和66(I/O操作请求所在的柱面号)。当前磁盘位于90号柱面。采用移臂调度采用FCFS算法时，系统服务的顺序和磁头移动的距离以及磁头改变方向次数？

服务顺序：90→55→72→100→88→93→66
移动距离：35+17+28+12+5+27=124(跨越的柱面总数)
其中磁头改变方向的次数为4次。

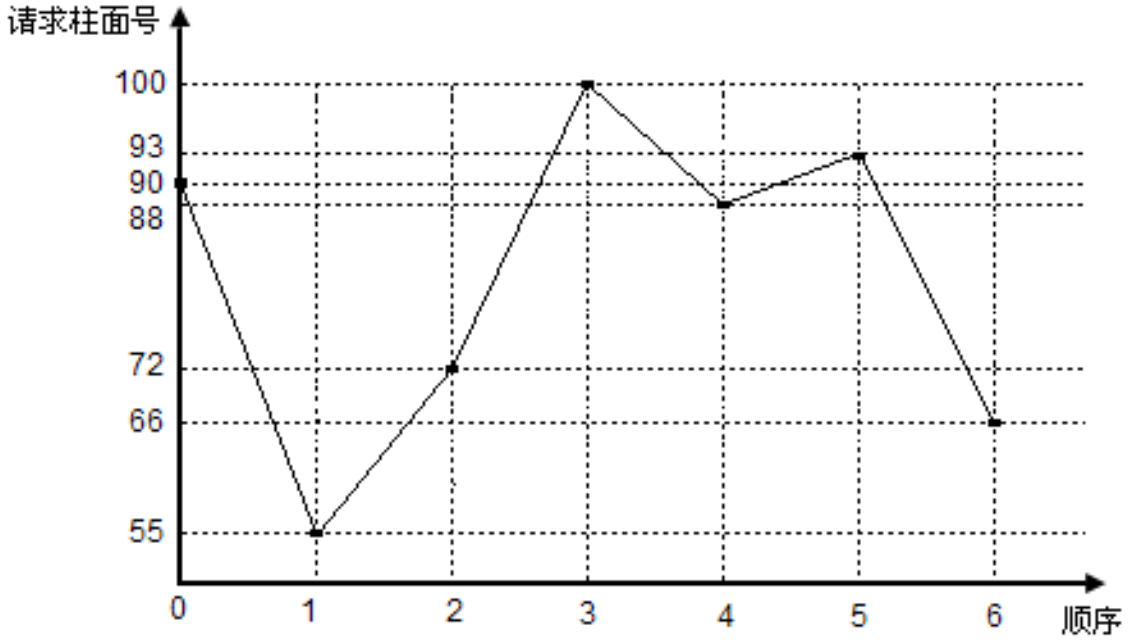

图7-22 FCFS移臂调度算法例子

◆ 最短寻道优先算法(SSTF，Shortest Seek Time First)

例7-1的一组I/O操作请求。
服务顺序：90→88→93→100→72→66→55
移动距离：2+5+7+28+6+11=59(跨越的柱面总数)
其中磁头改变方向的次数为2次。

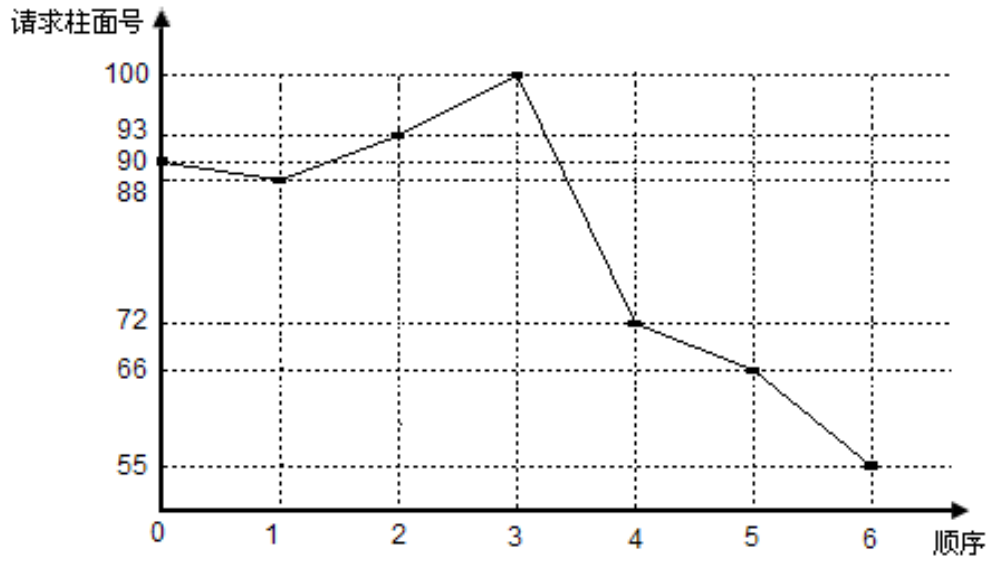

图7-23 SSTF移臂调度算法例子

◆ 扫描算法(SCAN)

例7-2 假定某磁盘共256柱面(柱面号0-255)，当前磁头位于90号柱面并且向柱面号小的方向移动。现在一组I/O操作的访问请求，它们的请求提出顺序依次是：55、72、100、88、93和66(I/O请求所在的柱面号)。那么，移臂调度采用SCAN算法时，系统服务的顺序和磁头移动的距离？

服务顺序：90→88→72→66→55→0→93→100
移动距离：2+16++6+11+55+93+7 =190(跨越的柱面总数)
其中磁头改变方向的次数为1次。

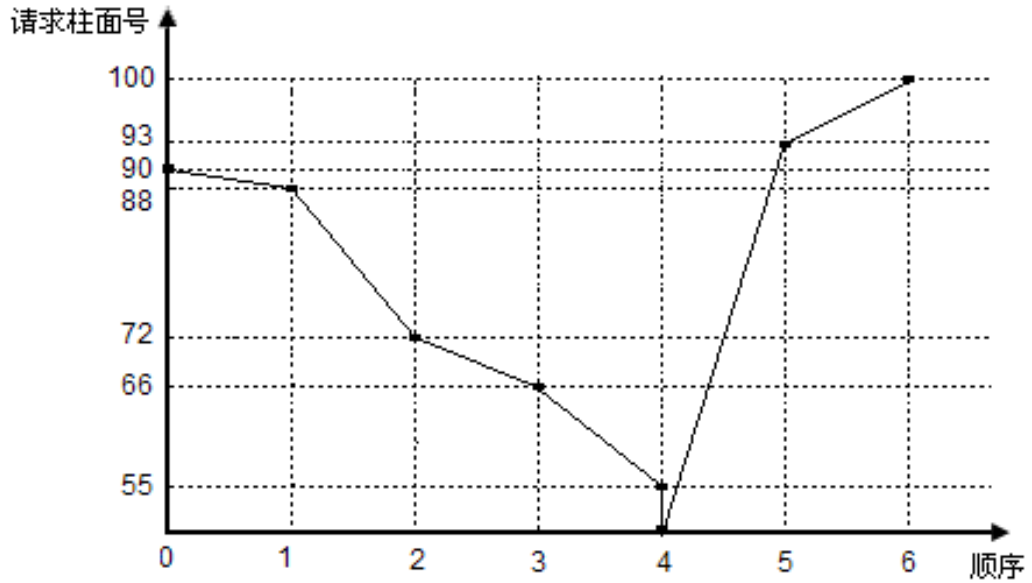

图7-24 SCAN移臂调度算法例子

◆ 电梯算法(Elevator Algorithm)

例7-2中的一组I/O操作请求。
服务顺序：90→88→72→66→55→93→100
移动距离：2+16+6+11+38+7 =80(跨越的柱面总数)
其中磁头改变方向的次数为1次。

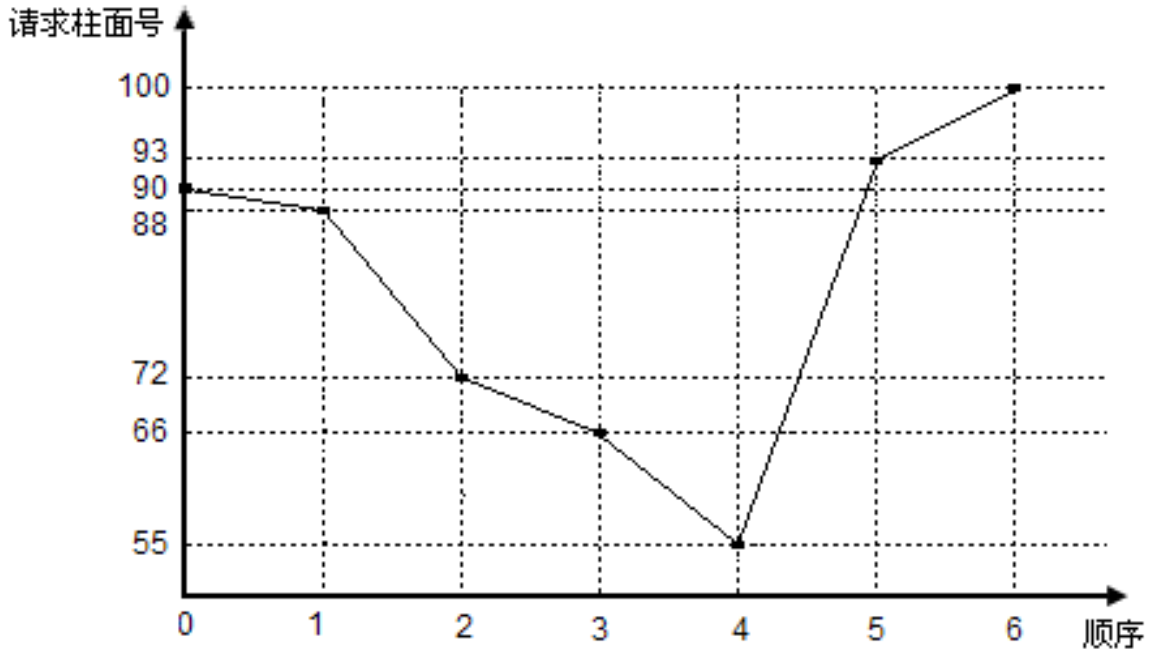

图7-25 电梯移臂调度算法例子

循环扫描算法(C-SCAN)

针对SCAN移臂调度算法和电梯移臂调度算法在公平性方面的不足。

“磁臂粘着”(Arm stickiness)现象

以扫描算法(SCAN)为例，减少“磁臂粘着” 现象的算法：

- ◆N-Step-SCAN算法
- ◆FSCAN算法