# 银行家算法

首先是银行家算法中的进程：

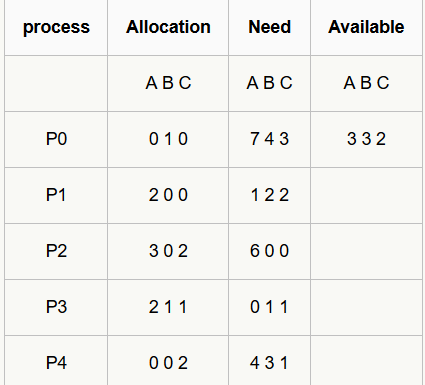
包含进程Pi的需求资源数量（也是最大需求资源数量，MAX）

已分配给该进程的资源A（Allocation）

还需要的资源数量N（Need=M-A）

Available为空闲资源数量，即资源池（注意：资源池的剩余资源数量+已分配给所有进程的资源数量=系统中的资源总量）

**举个实际例子，假设下面的初始状态：**



首先，进入算法第一步，初始化。那么Work = Available = [3 3 2]

首先看P0：P0的Need为[7 4 3]，Available不能满足，于是跳过去

P1的Need为[1 2 2]可以满足，我们令Work = Allocation[P1] + Work

此时Work = [5 3 2]

再看P2，P2的Need为[6 0 0]，那么现有资源不满足。跳过去。

那么看P3，Work可以满足。那么令Work = Allocation[P3] + Work，此时Work = [7 4 3]

再看P4，Work可以满足。令Work = Allocation[P4] + Work ，此时Work = [7 4 5]

到此第一轮循环完毕，由于找到了可用进程，那么进入第二轮循环。

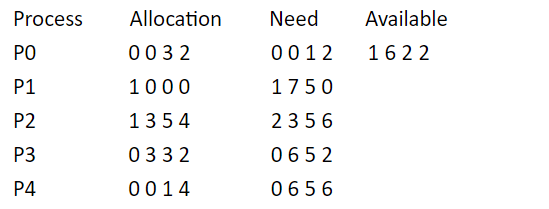
看P0，Work此时可以满足。令Work = Allocation[P0] + Work ，此时Work = [7 5 5]

再看P2，此时Work可以满足P2。令Work = Allocation[P2] + Work ， 此时Work = [10 5 7]

至此，算法运行完毕。找到安全序列<P1,P3,P4,P0,P2>，证明此时没有死锁危险。(安全序列未必唯一)

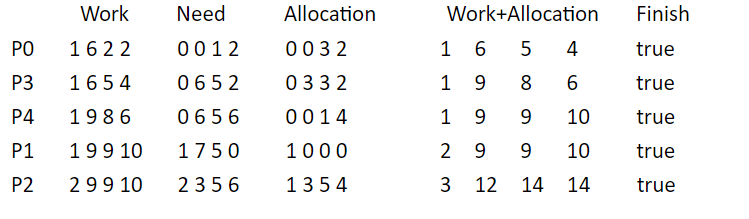
**#来个例子**

在银行家算法中，若出现下述资源分配情况：



（1）该状态是否安全？ （2）若进程P2提出请求Request（1，2，2，2）后，系统能否将资源分配给它？

（1）利用安全性算法对上面的状态进行分析（见下表），找到了一个安全序列{P0,P3,P4,P1,P2}，故系统是安全的。



（2）P2发出请求向量Request(1,2,2,2),系统按银行家算法进行检查：

①Request2(1,2,2,2)<=Need2(2,3,5,6)

②Request2(1,2,2,2)<=Available(1,6,2,2)

③系统先假定可为P2分配资源，并修改Available，Allocation2和Need2向量：

Available=(0,4,0,0)

Allocation2=(2,5,7,6)

Need2=(1,1,3,4)

此时再进行安全性检查，发现 Available=(0,4,0,0) 不能满足任何一个进程，所以判定系统进入不安全状态，即不能分配给P2相应的Request(1,2,2,2)。

**某系统有A、B、C、D四类资源可供五个进程P1、P2、P3、P4、P5共享。系统对这四类资源的拥有量为:A类3个、B类14个、C类12个、D类12个。进程对资源的需求和分配情况如下：**

**进程 已占有资源 最大需求数**



**按银行家算法回答下列问题：**

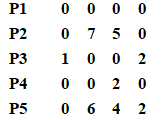
**（1）现在系统中的各类资源还剩余多少？（4分）**

**（2）现在系统是否处于安全状态？为什么？（6分）**

**（3）如果现在进程P2提出需要A类资源0个、B类资源4个、C类资源2个和D类资源0个，系统能否去满足它的请求？请说明原因。（6）**

（1）A：1；B：5；C：2；D：0

（2）need矩阵为：



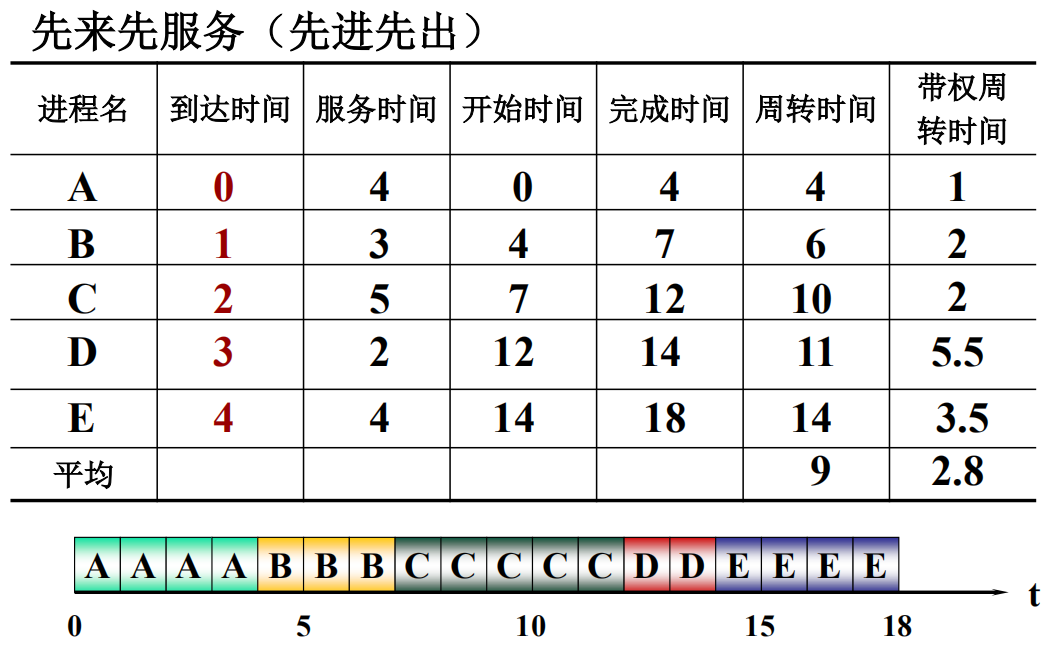
存在安全序列，如P1，P3，P4，P5，P2，所以安全

1. 能，因为试探分配后，可用资源为1，1，0，0。可找到安全序列，所以可分配。

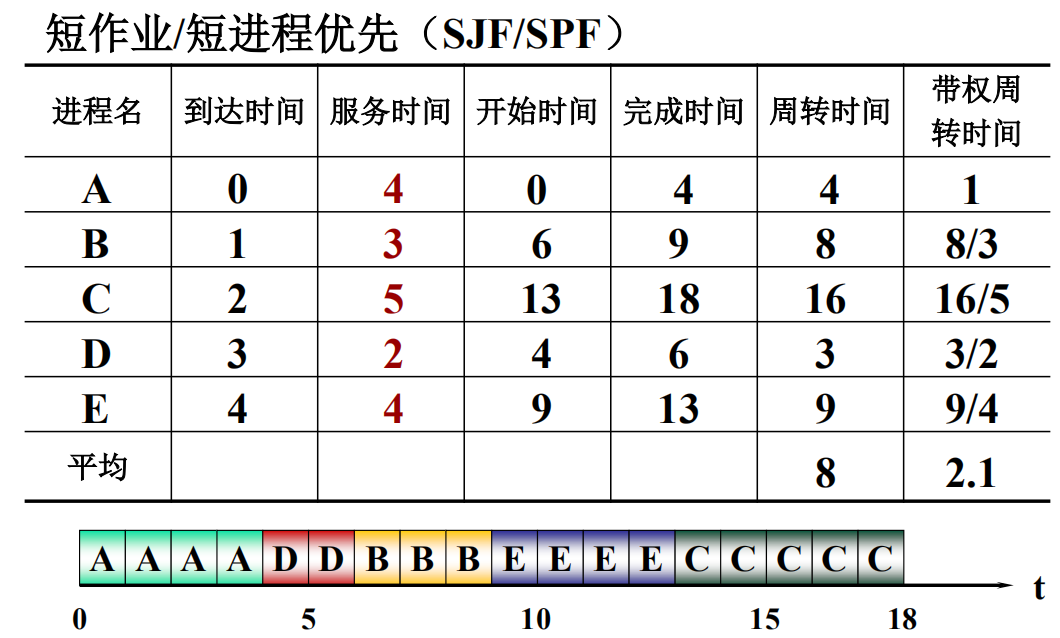
# 进程调度算法

书P95起

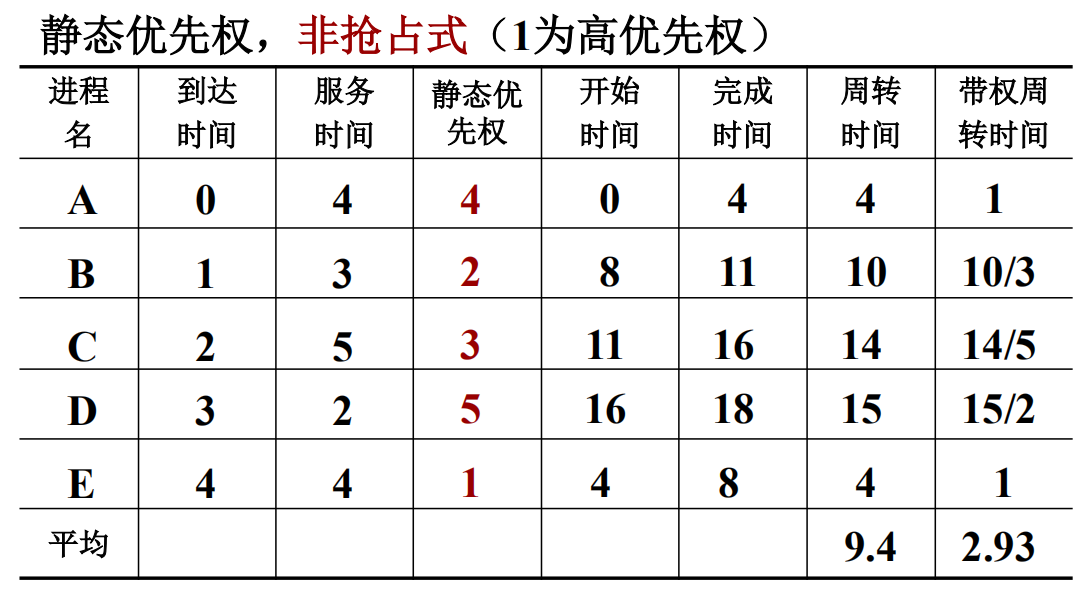
**先来先服务调度算法**

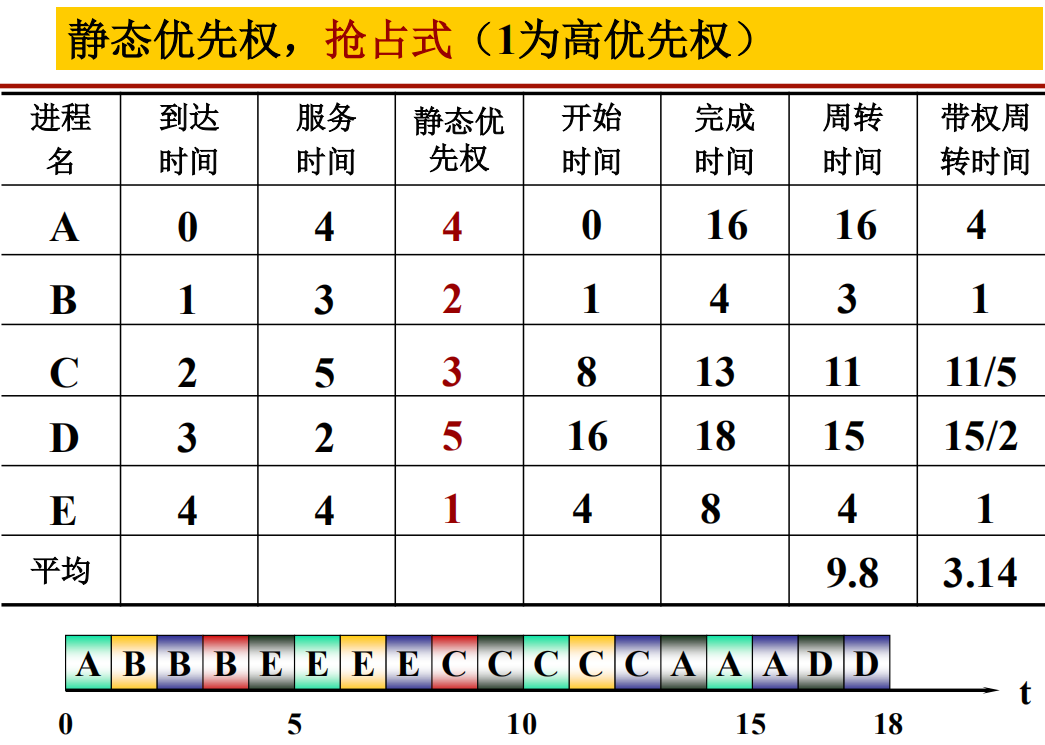
****

**短作业（进程）优先调度算法**

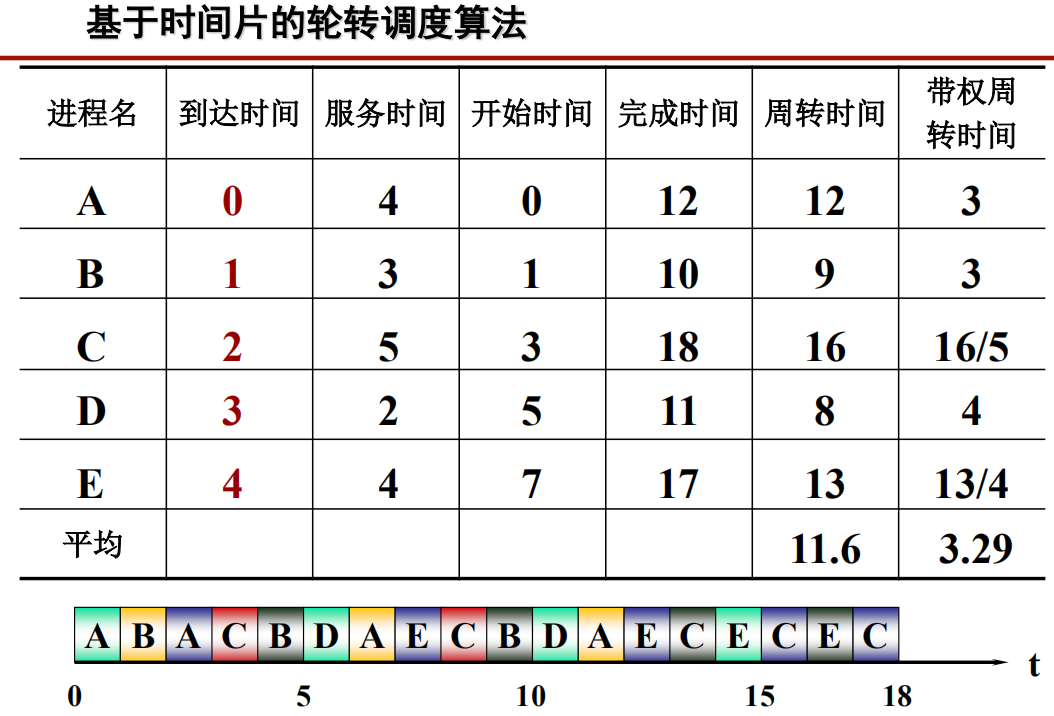


**最高响应比优先调度算法**





**时间片轮转调度算法**

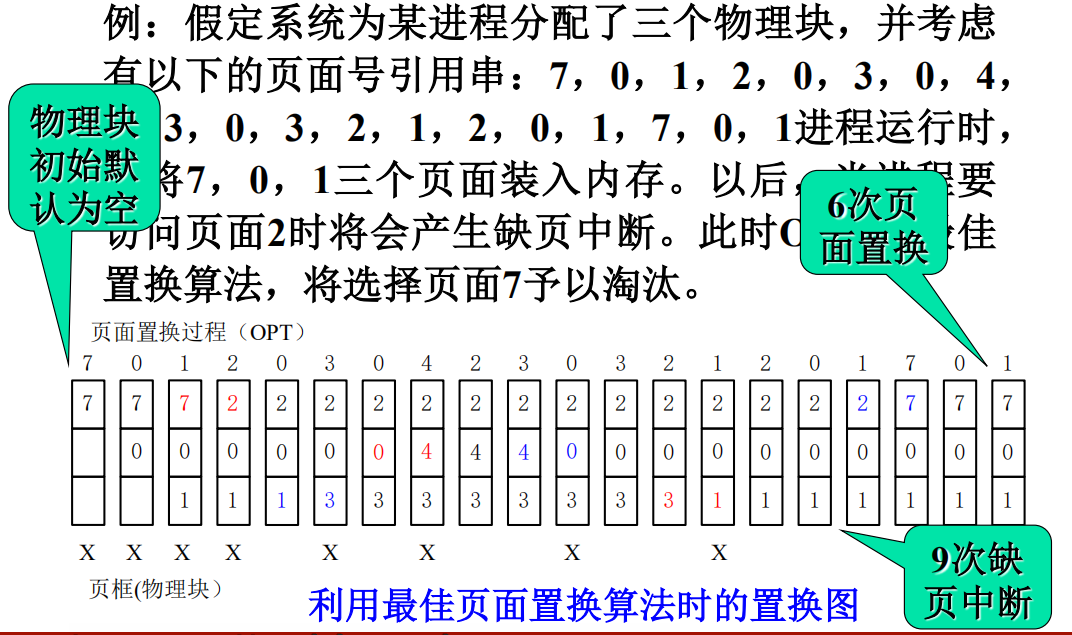


# 页面置换算法

好的页面置换算法，应具有较低的页面更换频率

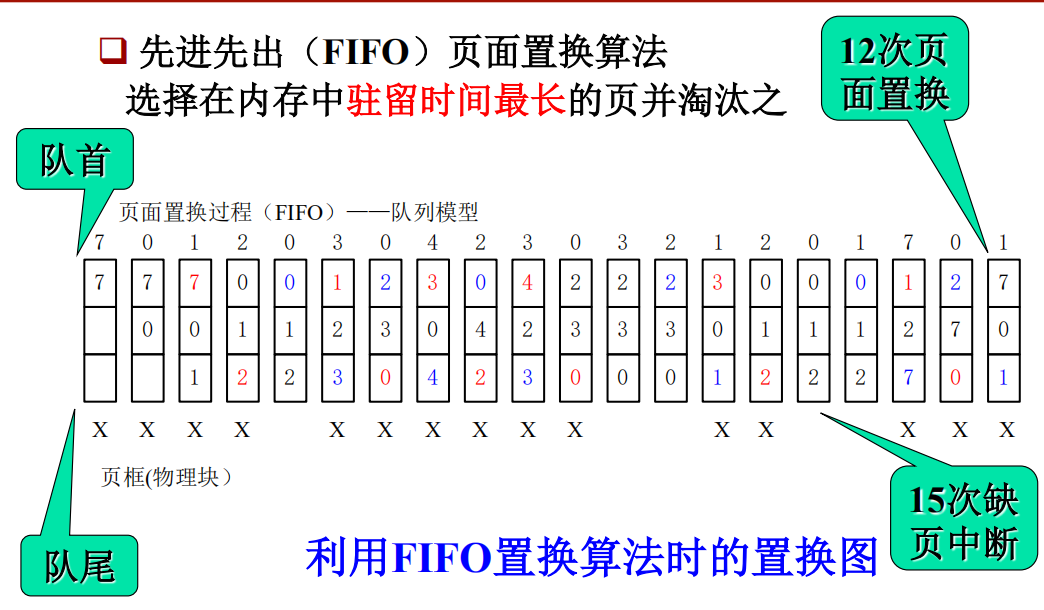
**最佳置换算法** ：保证获得最低的缺页率。

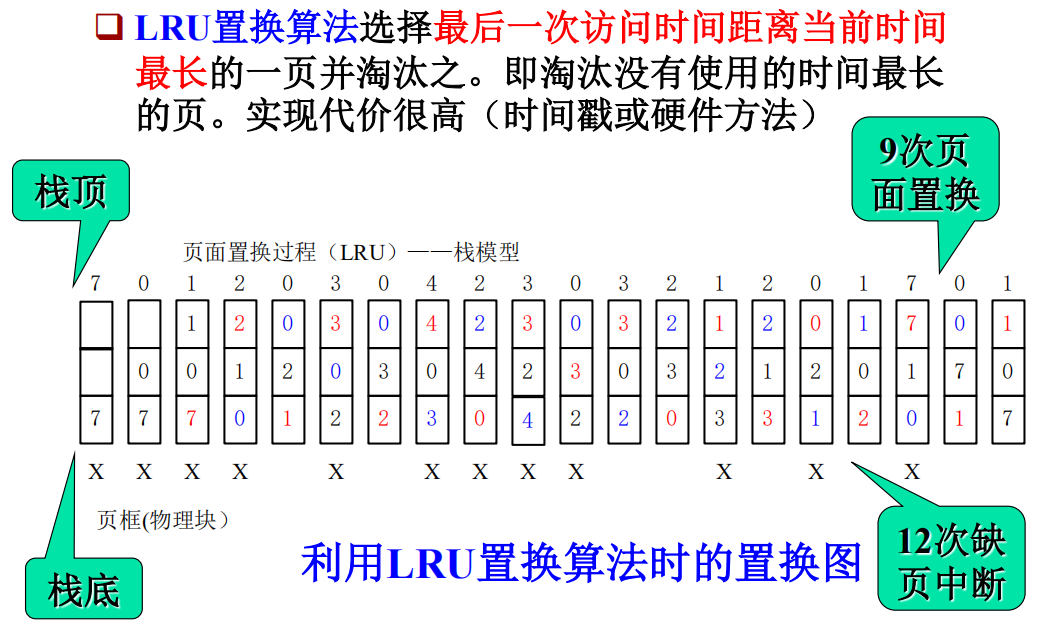
7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1



**先进先出置换算法：**淘汰最先进入内存的页面

7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

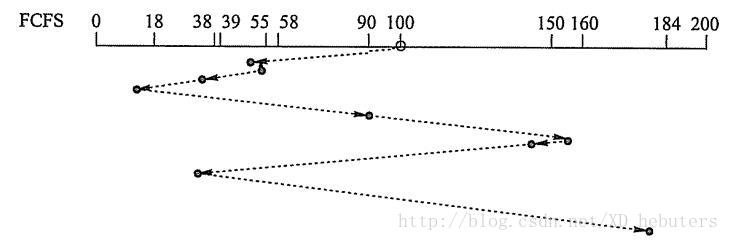


**LRU（最近最久未使用）置换算法：**选择最后一次访问时间距离当前时间最长的一页并淘汰之。

# 磁盘调度算法

**先来先服务FCFS**

先来先服务FCFS (First come, First Serve)：是一种自然公平的调度策略。先来先到，无特权用户。FCFS算法根据进程请求访问磁盘的先后顺序进行调度，这是一种最简单的调度算法，如图所示。



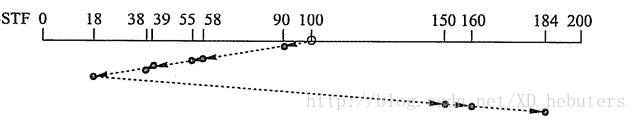
例如:，磁盘请求队列中的请求顺序分别为55、58、39、18、90、160、150、38、184，磁头初始位置是100磁道，釆用FCFS算法磁头的运动过程如图1所示。磁头共移动了 (45+3+19+21+72+70+10+112+146)=498 个磁道，平均寻找长度=498/9=55.3。

该算法的优点是具有公平性。如果只有少量进程需要访问，且大部分请求都是访问块聚的文件扇区，则有望达到较好的性能；但如果有大量进程竞争使用磁盘，那么这种算法在性能上往往接近于随机调度。所以，实际磁盘调度中往往对每个磁盘的读写任务进行区别对待。

**短寻道优先SSF**

短寻道优先SSF：（Shortest Seek First）则是考虑当前磁头离谁的数据最近，谁就优先。由于寻道在磁盘访问时间中占得比重最大，此种策略似乎正中要害，能够缩短磁盘访问时间。

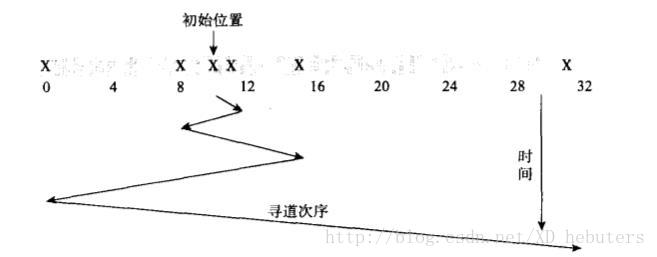
同样使用图1的例子，磁盘请求队列中的请求顺序分别为55、58、39、18、90、160、150、38、184，磁头初始位置是100磁道，釆用SSTF算法磁头的访问顺序：90、58、55、39、38、18、150、160、184。运动过程如图2所示。磁头共移动了 (10+32+3+16+1+20+132+10+24)=248 个磁道，平均寻找长度=248/9=27.5。



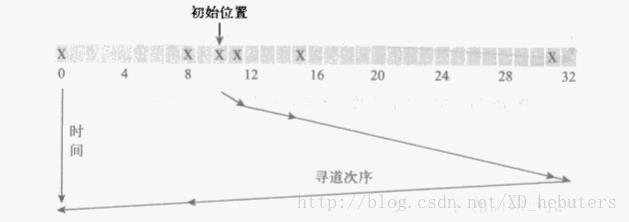
**扫描算法（SCAN）**

不过SSF调度算法缩短并不是绝对的。例如，如果当前的磁盘读写如图3所示，则磁盘读写请求的执行呈现的是一种左右摇摆的模式。这种情况下总寻道数大幅增加，系统花在寻道上的时间迅速增加。改进的办法就是不要左右摆动，而令其单向运动，即电梯调度策略。

这里写图片描述



**电梯调度策略ES：**（Elevator Scheduling），又称扫描算法（SCAN），先满足一个方向上所有请求，再满足相反方向的所有请求，循环往复。磁头向每个方向运动时（由里向外和由外向里），皆扫描到头。



**循环扫描算法**

书P234

防止当磁头刚从里向外移动而越过了某一磁道时，恰好又有一进程请求访问此磁道，致使该请求被大大延迟。即：当磁头移到最外的磁道并访问后，磁头立即返回到最里的欲访问磁道。

# PV原语

处理进程间的同步与互斥问题：<https://blog.csdn.net/yeyuangen/article/details/7361976>

六、物理块地址转换

