## 第一章 操作系统引论

1. 操作系统定义：操作系统是直接控制和管理计算机硬件、软件资源，合理地对各类资源进行调度，以方便用户使用的程序的集合。
2. 操作系统类型
   1. 批处理系统：缺点是缺乏交互性
   2. 分时系统：人机交互，及时性
   3. 实时系统：实时性
3. 操作系统的基本特征
   1. 并发：并发性是指两个或多个事件在同一时间间隔内发生。

并行性：是指两个或多个事件在同 一时刻发生。

* 1. 共享：共享是指系统中的资源可供内存中多个并发执行的进程使用。实现资源

共享的方式有两种：互斥共享方式、同时访问方式。

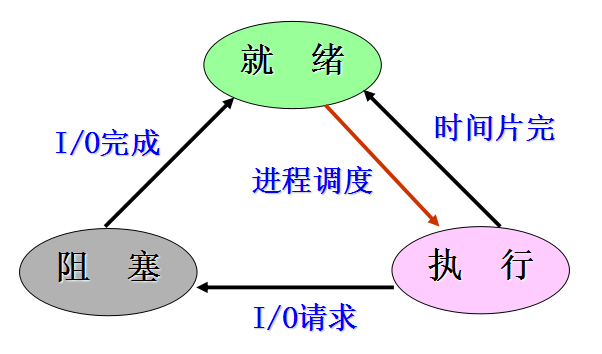
并发和共享是多用户操作系统的两个最基本的特征。

* 1. 虚拟：把通过某种技术将一个物理实体变为若干个逻辑上的对应物的功能称为“虚拟”。
  2. 异步：进程的异步是指进程以人们不可预知的速度向前推进。在操作系统中配置有完善的进程同步机制，能实现可再现性。

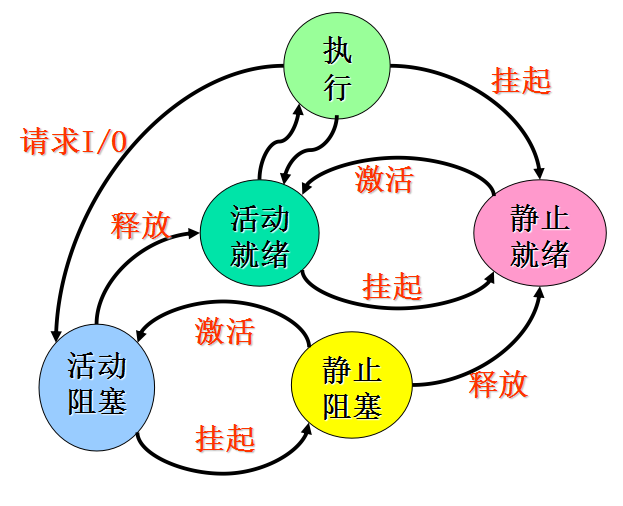
1. 操作系统的主要功能
   1. 处理机管理功能
   2. 存储器管理功能
   3. 设备管理功能
   4. 文件管理功能
   5. 向用户提供用户接口

## 第二章 进程的描述与控制

1. 进程
   1. 定义：进程是程序关于某个数据集合的一次执行过程。（进程是执行中的程序）
   2. 程序并发执行时的特征
      1. 间断性
      2. 失去封闭性
      3. 不可再现性
   3. 特征：结构特征（pcb+程序+数据=进程）、动态性（最基本）、并发性、独立性、异步性
   4. 三状态
      1. 就绪状态：进程以获得除cpu之外的所有必需资源，一旦得到cpu即可执行。
      2. 执行状态：进程已获得运行所必需的资源（只有一个）。
      3. 阻塞状态：正在执行的进程由于发生某件事而暂时无法执行。



* 1. 五状态：执行态、活动就绪、静止就绪、活动阻塞、静止阻塞



1. 进程控制块pcb
   1. PCB＋程序＋数据＝进程实体
   2. 进程管理（原语）：
      1. 创建
         1. 申请空白PCB
         2. 为 新进程分配其运行所需的资源
         3. 初始化进程控制块pcb
         4. 新进程插入就绪队列
      2. 终止
         1. 找出被终止进程的PCB
         2. 若进程状态为运行态，置CPU调度标志为真
         3. 若其有子孙进程，终止其子孙进程并回收其资源
         4. 回收终止进程的资源
         5. 回收终止进程的PCB
2. 进程同步
   1. 两种形式的制约：间接相互制约（资源共享）、直接相互制约（进程合作）
   2. 临界资源：一次仅允许一个进程访问的资源
   3. 临界区：每个进程访问临界资源的代码
   4. 同步机制应遵循的四个准则：空闲让进、忙则等待、有限等待、让权等待
   5. 同步机制:工具：信号量和管程
   6. 信号量机制
      1. 整型信号量（不可能取到负值）：缺点：不满足“让权等待”
      2. 纪录型信号量（可正可负） 共享资源，初值就是资源的数量，比如互斥初值是1，同步（合作）初值是0
      3. AND型信号量
      4. 信号量集
3. 信号量的应用
   1. 利用信号量实现进程互斥

semaphore mutex =1; //表示打印机

begin

parbegin

p1: begin

repeat

… …

wait(mutex);

使用打印机

signal(mutex);

… …

until false;

end

p2: begin

repeat

… …

wait(mutex);

使用打印机

signal(mutex);

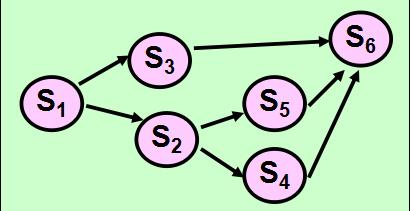
… …

until false;

end

parend

* 1. 利用信号量实现前驱关系



semaphore a, b, c, d, e, f, g = 0, 0, 0, 0, 0, 0, 0;

begin

parbegin

begin S1; signal(a); signal(b); end;

beginwait(a);S2;signal(c); signal(d); end;

begin wait(b); S3; signal(e); end;

begin wait(c); S4; signal(f); end;

begin wait(d); S5; signal(g); end;

begin wait(e); wait(f); wait(g); S6; end;

parend

end

* 1. 利用记录型信号量实现同步

p2：x=处理结果；p1：Print(x)；

semaphore empty=1; //变量x可赋值使用，即P1 的print(x)已完成

semaphore full=0; //变量x已赋值，即P1可 print(x)

begin

parbegin

p1: begin

repeat

… …

wait(full);

print(x)；

signal(empty);

… …

until false;

end

p2: begin

repeat

… …

wait(empty);

x:=处理结果；

signal(full);

… …

until false;

end

parend

1. 进程通信
   1. 低级通信：进程间仅交换一些状态和少量数据。进程之间的互斥和同步。
   2. 高级通信：进程间可交换大量数据。
2. 线程
   1. 引入线程的目的：为了减少程序在并发执行时所付出的时空开销，使OS具有更好的并发性
   2. 线程的定义：是进程的一条执行路径，每个线程共享其所属的进程的所有资源。
3. 几种同步

1）生产者-消费者问题

一般意义的“生产者—消费者”问题：N个buffer，多个生产者，多个消费者，循环存取buffer。这就是一般意义的“生产者—消费者”问题。利用记录型信号量解决一般意义的“生产者—消费者”问题算法描述，请看教材。

说明：

（1）由于buffer有N个，而且buffer又是临界资源，因此，需要增加一个信号量mutex来实现对buffer的互斥访问，其初始值为1。需要特别强调的是，这种情况下，mutex不能省略。

（2）这种情况下，只要保证为不同的进程分配不同buffer，putdata和getdata操作是可以同时进行的，因此，对buffer的互斥访问不是发生在对buffer的存取操作上，而是发生在对buffer的分配上。

Int i=0,out=0;

Item buffer[n];

Semaphore mutex=1,empty=n,full=0;

//mutex:生产者间，消费者间互斥使用缓冲区

//empty: 缓冲区的空闲容量

//full: 缓冲区的已占容量

Void producer(){

do{

生产一个产品放入nextp;

wait(empty);

wait(mutex);

buffer[in]=nextp;

in=(in+1) % n;

signal(mutex);

signal(full);

}while(TRUE)

}

Void consumer(){

do{

wait(full);

wait(mutex);

nextc =buffer[out];

out =(out+1) mod n;

signal(mutex);

signal(empty);

消费 nextc中的产品;

}while(TRUE)

}

Void main(){

cobegin

proceducer();

consumer();

coend

}

2）“哲学家进餐”问题

教材中解决哲学家进餐问题的算法有可能发生死锁，为避免死锁，可以采用以下三种策略：

策略一原理：至多只允许四个哲学家同时进餐，以保证至少有一个哲学家能够进餐，最终总会释放出他所使用过的两支筷子，从而可使更多的哲学家进餐。定义信号量count，只允许4个哲学家同时进餐，这样就能保证至少有一个哲学家可以就餐。

semaphore chopstick[5]={1，1，1，1，1};

semaphore count=4;

void philosopher(int i)

{while(true)

{

think();

wait(count); //请求进入房间进餐

wait(chopstick[i]); //请求左手边的筷子

wait(chopstick[(i+1)%5]); //请求右手边的筷子

eat();

signal(chopstick[(i+1)%5]); //释放右手边筷子

signal(chopstick[i]); //释放左手边的筷子

signal(count); //退出房间释放信号量

}

}

策略二原理：仅当哲学家的左右两支筷子都可用时，才允许他拿起筷子进餐。可以利用AND 型信号量机制实现，也可以利用信号量的保护机制实现。利用信号量的保护机制实现的思想是通过记录型信号量mutex对取左侧和右侧筷子的操作进行保护，使之成为一个原子操作，这样可以防止死锁的出现。描述如下：

semaphore mutex = 1 ;

semaphore chopstick[5]={1，1，1，1，1};

void philosopher(int i)

{while(true)

{

think();

wait(mutex);

wait(chopstick[(i+1)%5]);

wait(chopstick[i]);

signal(mutex);

eat();

signal(chopstick[(i+1)%5]);

signal(chopstick[i]);

}

}

策略三原理：规定奇数号的哲学家先拿起他左边的筷子，然后再去拿他右边的筷子；而偶数号的哲学家则相反。按此规定，将是1、2号哲学家竞争1号筷子，3、4号哲学家竞争3号筷子。即五个哲学家都竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一个哲学家能获得两支筷子而进餐。

semaphore chopstick[5]={1，1，1，1，1};

void philosopher(int i)

{while(true)

{

think();

if(i%2 == 0) //偶数哲学家，先右后左。

{

wait (chopstick[(i + 1)%5]) ;

wait (chopstick[i]) ;

eat();

signal (chopstick[(i + 1)%5]) ;

signal (chopstick[i]) ;

}

else //奇数哲学家，先左后右。

{

wait (chopstick[i]) ;

wait (chopstick[(i + 1)%5]) ;

eat();

signal (chopstick[i]) ;

signal (chopstick[(i + 1)%5]) ;

}

}

3）“读者—写者”问题的演变

从本质上讲，读者—写者问题要解决：读、读共享；写、写互斥；写、读互斥。有两种解决模式：

模式一，读者优先的“读者—写者”问题解决模式（算法描述请参考教材）：

①定义互斥信号量wmutex，实现写、写互斥和写、读互斥。

②定义整型变量Readcount表示正在读的进程数目。由于只要有一个Reader进程在读，便不允许Writer进程写，因此仅当Readcount=0，即无Reader进程在读时，Reader才需要执行Wait(wmutex)操作。若Wait(wmutex)操作成功，Reader进程便可去读，相应地，做Readcount+1操作。同理，仅当Reader进程在执行了Readcount减1操作后其值为0时，才需执行signal(wmutex)操作，以便让Write进程写。

③由于Readcount为多个读进程共享（修改），因此需要以互斥方式访问，为此，需要定义互斥信号量rmutex，保证读进程间互斥访问Readcount。

说明：在读者—写者问题中，实现“读、读共享”是有一定难度的，请掌握该模式（由②和③构成）。

模式二，写者优先的“读者—写者”问题解决模式，算法描述如下：

设3个信号量：

rmutex --- 读互斥信号量，初值为1；

wmutex --- 写互斥信号量，初值为1；

s --- 用于在写进程到达后封锁后续的读者，初值为1；

count --- 共享变量，用于记录当前正在读文件的读者数目，初值为0；

semaphore rmutex=1, wmutex=1, s=1;

int count=0;

main() {

Cobegin

Reader();

Writer();

Coend

}

Reader() {

While(1) {

P(s);

P(rmutex);

If (count==0) P(wmutex); /\*当第1个读者读文件时，阻止写者写\*/

count++

V(rmutex);

V(s);

读文件；

P(rmutex);

count--;

If (count==0) V(wmutex); /\*当最后1个读者读完文件时，允许写者写\*/

V(rmutex);

}

}

Writer() {

While(1) {

P(s);

P(wmutex);

写文件；

V(wmutex);

V(s);

}

}

## 第三章 处理机调度与死锁

1. 调度层次
   1. 高级调度（作业调度）
   2. 中级调度（进程调度）
   3. 低级调度
2. 作业调度
   1. FCSF先来先服务，作业等待时间得长短。比较有利于长作业（进程），而不利于短作业（进程）。
   2. SJF短作业优先，作业的运行时间。
      1. 优点：能有效的降低作业的平均等待事件，提高系统吞吐量。
      2. 缺点：对长作业不利；该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业（进程）会被及时处理；由于作业（进程）的长短含主观因素，不一定能真正做到短作业优先。

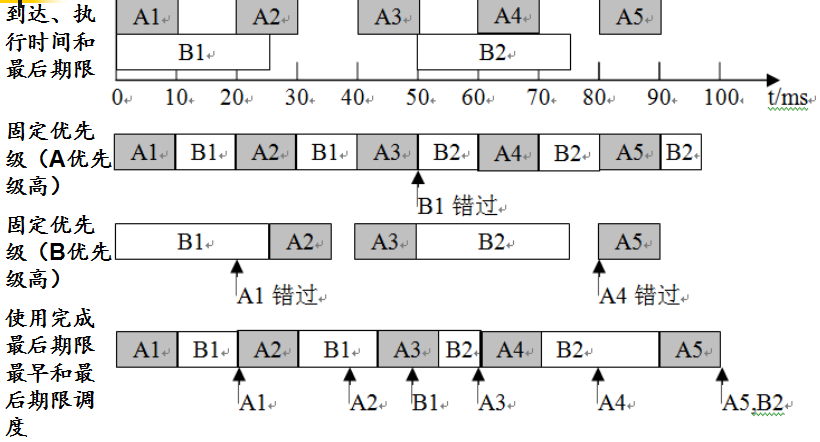


* 1. 高响应比优先

优先权=等待时间+要求服务时间/要求服务时间

* 1. RR轮转调度算法，时间片的确定要适中
  2. 多级反馈队列
  3. EDF最早截至时间优先

下图中有两个周期性任务，任务A的周期时间为20ms，每个周期的处理时间为10ms；任务B的周期时间为50ms，每个周期的处理时间为25ms



* 1. LLF最低松弛度优先

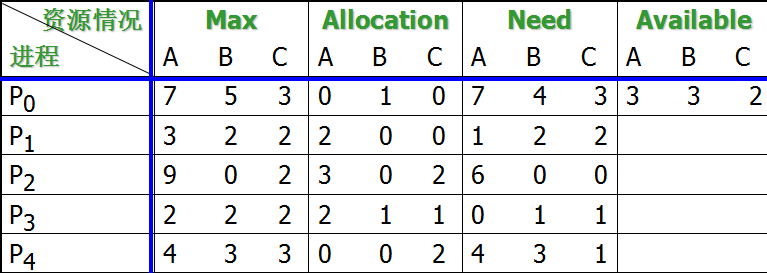
松弛度=必须完成时间-其本身的运行时间-当前时间

进程切换条件：有任务完成；有任务松弛度降到0。

1. 死锁
   1. 定义：是指多个进程在运行过程中因为争夺资源而造成的一种僵局，当进程处于这种状态时，若无外力作用，他们都将无法再向前推进
   2. 原因：竞争资源（不可抢占资源，可消耗资源），进程间推进顺序非法。
   3. 产生死锁得必要条件：互斥条件、请求和保持条件、不可抢占（不可剥夺）条件、环路等待条件
   4. 处理死锁的基本方法：
      1. 预防死锁：破坏产生死锁得必要条件，其中破坏互斥条件是最不实际的
         1. 破坏“请求和保持”条件：系统规定所有进程在开始运行之前，都必须一次性的申请其在整个运行过程所需的全部资源
         2. 破坏“不剥夺”条件
         3. 破坏“环路等待”条件：所有进程对资源的请求必须严格按照资源序号递增的次序提出
      2. 预防死锁：银行家算法、安全性算法
      3. 检测死锁：资源分配图，死锁定理
      4. 解除死锁：剥夺资源（从其他进程剥夺足够数量的资源给死锁进程以解除死锁状态。），撤销进程（最简单的是让全部进程都死掉；温和一点的是按照某种顺序逐个撤销进程，直至有足够的资源可用，使死锁状态消除为止。）
2. 银行家算法
   1. 安全状态



* 1. 银行家算法



T0时刻的安全性：用安全性算法对T0时刻的资源分配情况进行分析，存在着一个安全序列{P1,P3,P4,P2,P0}，故系统是安全的

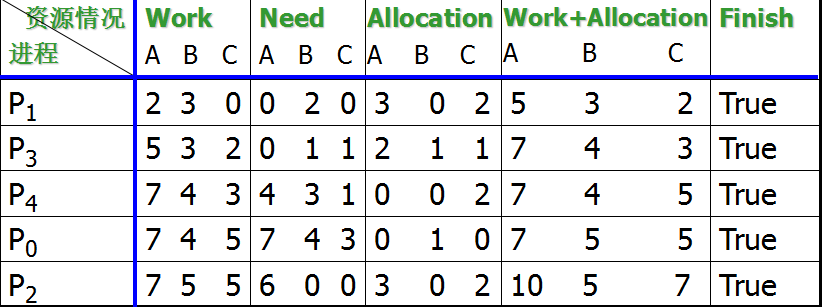
P1发出资源请求向量Request1(1,0,2)，系统按银行家算法检查:

（1）Request1(1,0,2)<=Need1(1,2,2)

（2）Request1(1,0,2)<=Available1(3,3,2)

（3）系统先假定可为P1分配资源，并修改向量Available，Allocation1，Need1

（4）再利用安全性算法检查此时系统是否安全。如下表:



由安全性检查得知，能找到一个安全序列{P1,P3,P4,P0,P2}，因此，系统是安全的，可以立即将P1所申请的资源分配给它。

1. 死锁定理：S状态为死锁状态的充分条件是当且仅当S状态的资源分配图是不可完全简化的。<死锁定理>  
   第四章 存储器管理
2. 重定位
3. 程序的装入
   1. 绝对装入方式
   2. 可重定位的装入方式（静态）
   3. 动态运行时装入方式（动态）重定位寄存器
4. 内存的分配方式（简答题）
   1. 连续的分配方式
   2. 离散的分配方式
5. 连续的分配方式（优缺点）
   1. 单一连续分配：一道程序在内存中，内存浪费严重
   2. 固定分区分配：程序变多缺点：会产生内碎片
   3. 动态分区分配（重点）不能消除外碎片
      1. 思想：按需分配
      2. 实现：分配算法（3种）（计算题）
         1. 首次适应算法FF

FF算法要求空闲分区表以地址递增的次序排列。在分配内存时，从表首开始顺序查找，直至找到一个大小能满足要求的空闲分区为止；然后按照作业的大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍留在空闲分区表中。若从头到尾不存在满足要求的分区，则分配失败。

优点：优先利用内存低址部分的内存空间

缺点：低址部分不断划分，产生小碎片（内存碎块、内 存碎片、零头）；每次查找从低址部分开始，增 加了查找的开销

* + - 1. 循环首次适应算法NF

在分配内存空间时，从上次找到的空闲分区的下一个空闲分区开始查找，直到找到一个能满足要求的空闲分区，从中划出一块与请求大小相等的内存空间分配给作业。

优点：使内存空闲分区分布均匀，减少查找的开销

缺点：缺乏大的空闲分区

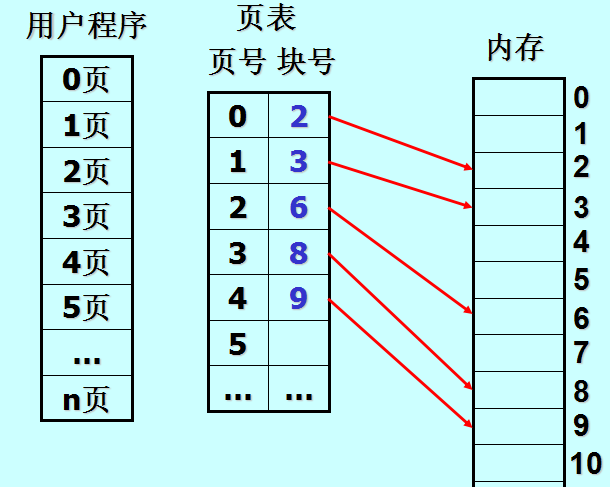
* + - 1. 最佳适应算法BF

所谓“最佳”是指每次为作业分配内存时，总是把能满足要求、又是最小的空闲分区分配给作业，避免“大材小用”。要求将所有的空闲分区按其容量以从小到大的顺序形成一空闲分区链。

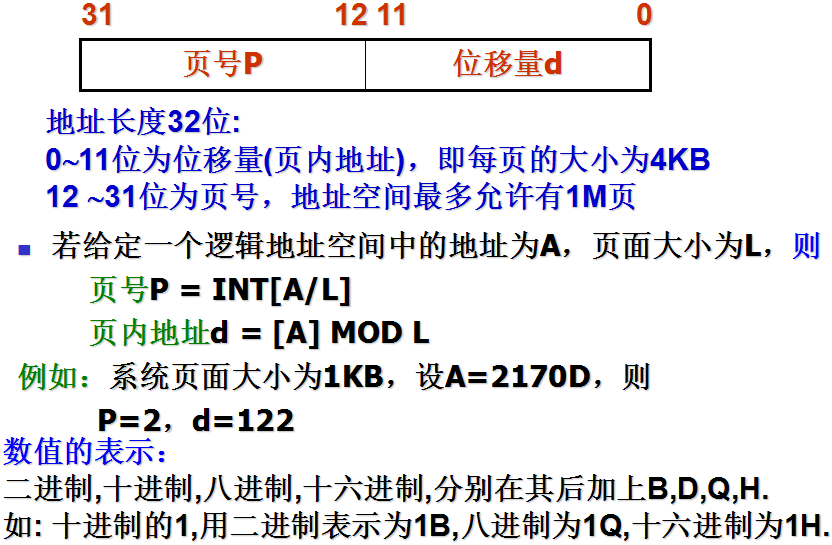
缺点：产生许多难以利用的小空闲区

* 1. 可重定位的分区分配
     1. 以动态运行时装入方式为前提
     2. 在动态分区分配方式上增加一个紧凑（拼接碎片）功能

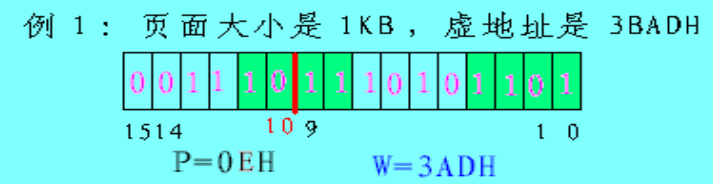
1. 离散的分配方式
   1. 分页存储管理方式（可能在最后一页产生内碎片）进程分页，内存分块（页和块等大小）
      1. 页面与页表
         1. 页表（计算题）



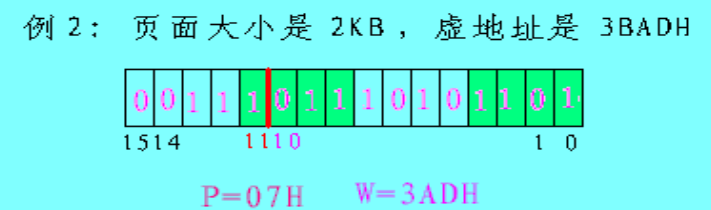
* + - 1. 虚地址结构（逻辑地址）



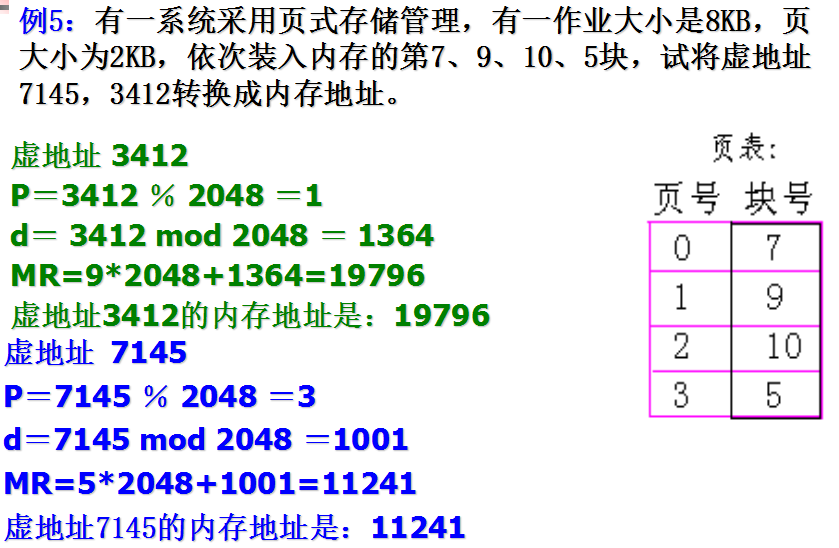
例1. 设页面大小为1KB，将逻辑地址3BADH划分为页号和页内偏移量两部分。用16进制表示。

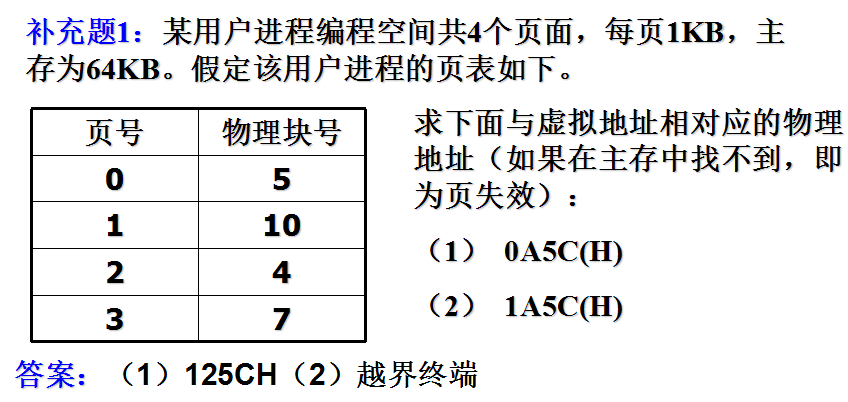


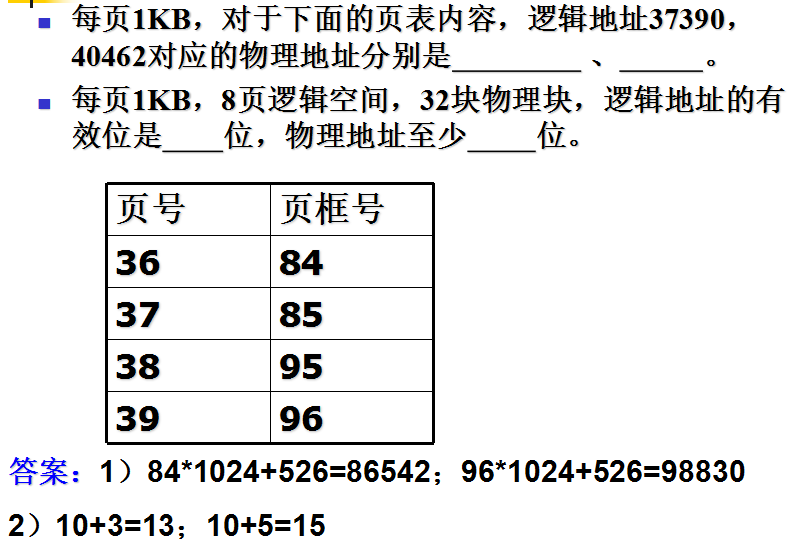
例2. 设页面大小为2KB，将逻辑地址3BADH划分为页号和页内偏移量两部分。用16进制表示。



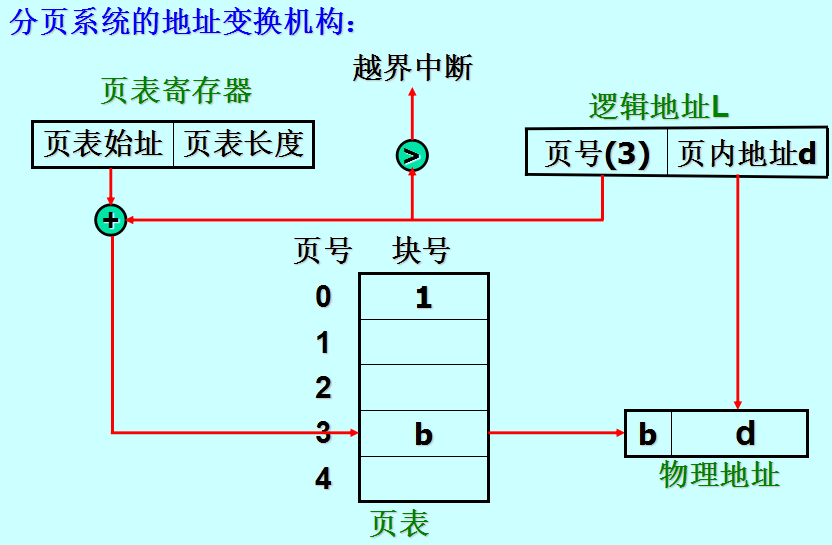
* + - 1. 页式地址映射（逻辑地址转换城物理地址）（考点）



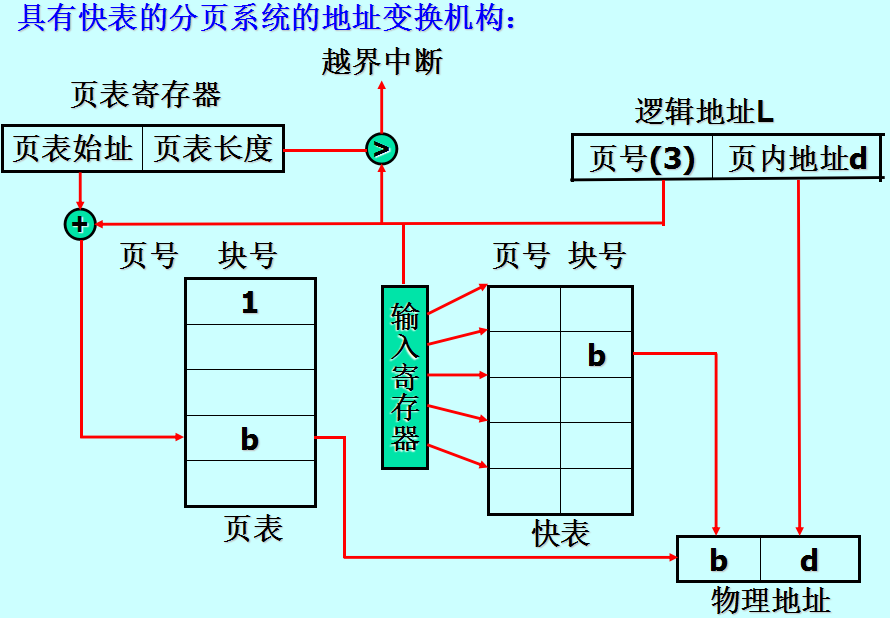




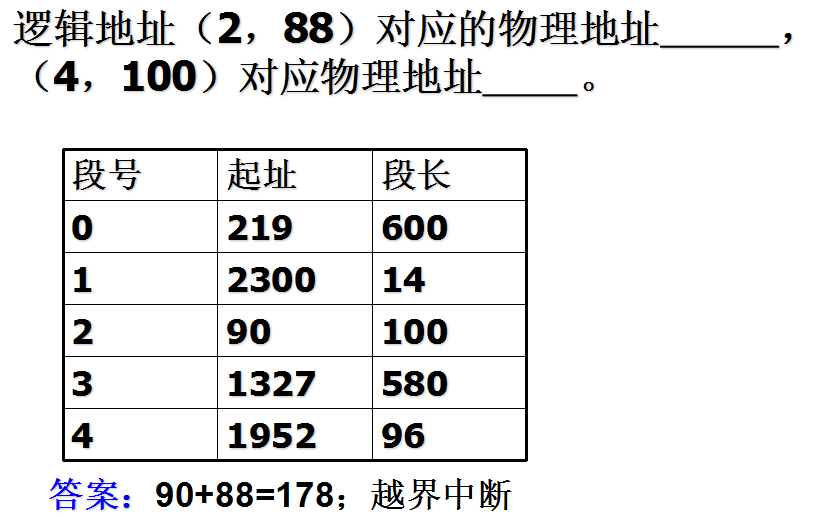
* + 1. 地址转换机构（页表在内存，页表寄存器快表。页表很庞大时采取两级或多级页表）



有快表有页表，先查快表；没有快表查页表



* 1. 分段存储管理方式（不可能有内碎片产生，外碎片不可避免）进程分段，各段在内存。段与段之间离散分配，某一段在内存中连续分配  
     计算：给定逻辑地址合成物理地址段表



* 1. 段页式存储管理方式

进程分段，段内分页，内存分块

* 1. 分页和分段的主要区别

相似点：采用离散分配方式，通过地址映射机构实现地 址变换

不同点：页是信息的物理单位，分页是为了满足系统的 需要；段是信息的逻辑单位，含有一组意义相对完 整的信息，分段是为了满足用户的需要。

页的大小固定且由系统确定，由系统把逻辑地址分为页号和页内地址，由机器硬件实现；段的长度不固定，取决于用户程序，编译程序对源程序编译时根据信息的性质划分。

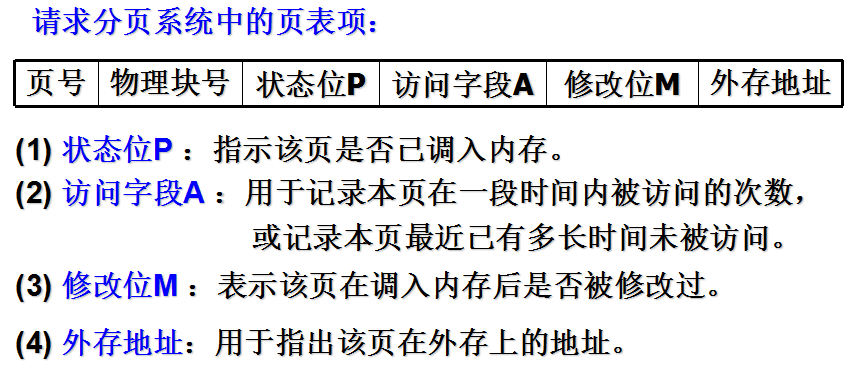
分页的作业地址空间是一维的；分段的作业地址空间是二维的。

1. 虚拟存储器
2. 虚拟存储器的定义：是指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。其逻辑容量由内存容量和外存容量之和所决定，其运行速度接近于内存速度而每位的成本却接近于外存。
3. 常规存储管理的特征
   1. 一次性:作业在运行前一次性地全部装入内存
   2. 驻留性:作业装入内存后，便一直驻留在内存中,直至作业运行结束。
4. 虚拟存储器的特征
   1. 多次性：一个作业被分成多次调入内存运行
   2. 对换性：允许在作业的运行过程中进行换进、换出。
   3. 虚拟性：能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。

虚拟性以多次性和对换性为基础；

多次性和对换性又必须建立在离散分配的基础上。

1. 虚拟存储器的实现
   1. 请求分页存储管理方式
      1. 硬件的支持：页表机制、缺页中断机制、地址变换机构



* + 1. 内存分配策略
       1. 最小物理块数的确定（和硬件有关）
       2. 物理块的分配策略
          1. 固定分配局部置换
          2. 可变分配全局置换
          3. 可变分配局部置换
       3. 物理块的分配算法（保证最小块数）
  1. 请求分段存储管理方式  
     以分段为单位进行调入调出  
     特别有利于实现分段共享

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **段名** | **段长** | **段的基址** | **存取方式** | **访问字段A** | **修改位M** | **存在位P** | **增补位** | **外存始址** |

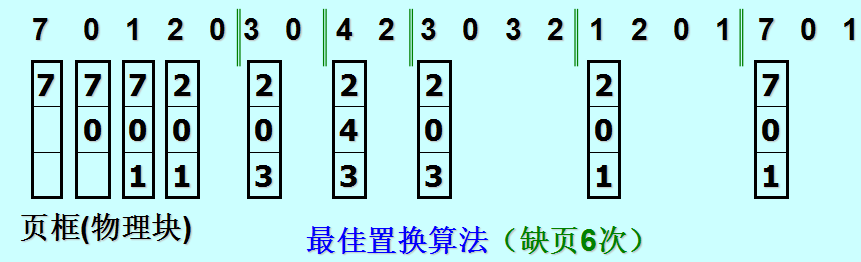
* 页面调入过程

1. 每当程序要访问的页面不在内存的时候，便向CPU发出一条缺页中断，然后保护CPU环境，分析中断原因，转向中断处理程序。
2. 中断处理程序根据页表找出该页在外存的物理块后，如果内存未满，则将将该页调入，否则按照相应的置换算法从内存中选择一页然后换出。
3. 如果该页未被修改，则不必将其写回磁盘，直接将新页调入即可，然后修改相关的表目，否则先将该页写回磁盘
4. 页面置换算法（计算，简答）
   1. 最佳置换算法

其所选择的被淘汰页面，将是以后永不再用的，或许是在最长(未来)时间内不再被访问的页面。

优点：保证获得最低的缺页率

缺点：无法预知一个进程在内存的若干个页面，哪个在未来最长时间内不再被访问。算法无法实现，但可评价其他算法。



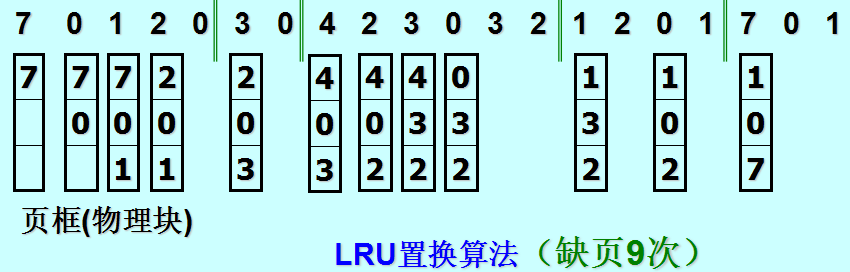
* 1. 先进先出置换算法

算法与进程的实际运行规律不相适应，因为进程中的某些页面经常被访问，但先进先出置换算法不能保证这些页面不被淘汰。

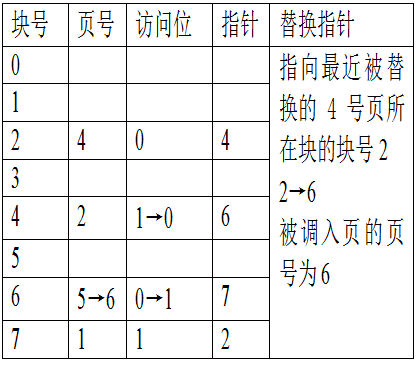


* 1. 最近最久未使用（LRU）算法

LRU置换算法虽然较好，但需较多的硬件支持



* 1. Clock置换算法（最近未使用算法）



* 1. 页面缓冲算法

**补充题.** 已知一个有快表的请求分页系统，内存读写周期为Tr(1us)，内外存之间传送一个页面的时间为Tt(5ms)，快表的命中率为p (80%)，页面失效率为f (10%)。忽略快表存取时间。

请计算快表命中时的存取时间；快表不命中、内存中页表能查到时的存取时间；页面失效时的存取时间。并计算系统的内存有效存取周期EAT(平均存取时间)。

答：快表命中时的存取时间：1μs；快表不命中、内存中页表能查到时的存取时间：1μs+1μs=2μs；（快表不命中、）页面失效时的存取时间：1μs+5ms +1μs=5002μs

系统的内存有效存取周期EAT：1μs\*0.8+2μs\*(1-0.8)\*(1-0.1)+5002μs\*(1-0.8)\*0.1=101.2μs

13. 在一个请求分页系统中，假如一个作业的页面走向为4、3、2、1、4、3、5、4、3、2、1、5，当分配给该作业的物理块数M分别是3和4时，采用FIFO面替换算法，计算访问过程中所发生的缺页次数和缺页率，比较所得结果。

答：设开始时系统中未装入页面。

(1) 当M=3时，FIFO算法的缺页情况：

4 3 2 1 4 3 5 4 3 2 1 5 页面调用序列

4 4 4 1 1 1 5 5 5 5 5 5

3 3 3 4 4 4 4 4 2 2 2

2 2 2 3 3 3 3 3 1 1

缺 缺 缺 缺 缺 缺缺 中 中 缺 缺 中

缺页次数为9次。缺页率为：(9/12)×100%=75%。

(2) 当M=4时，FIFO算法的缺页情况:

4 3 2 1 4 3 5 4 3 2 1 5页面调用序列

4 4 4 4 4 4 5 5 5 5 1 1

3 3 3 3 3 3 4 4 4 4 5

2 2 2 2 2 2 3 3 3 3

1 1 1 1 1 1 2 2 2

缺 缺 缺 缺 中 中 缺 缺 缺 缺 缺 缺

缺页次数为10次。缺页率为：(10/12)×100%=83%。

通过以上缺页次数和缺页率的分析计算，可以看出：对于FIFO算法，增加物理块数，不一定能减少缺页次数。

1. 抖动和工作集
   1. 产生“抖动”的原因是同时在系统中运行的进程太多，而分配给每一个进程的物理块太少，不能满足进程正常运行的基本要求，致使每个进程在运行时，频繁的出现缺页，必须请求系统将所缺之页调入内存。
   2. 工作集是指在某段时间间隔里，进程实际所要访问页面的集合。

7.共享段表

1）共享段的分配与回收

2）分段保护：

> 越界检查

> 存取控制检查

> 环保护机构：

一个程序可以访问驻留在相同环或者低特权环中的数据

一个程序可以调用驻留在相同环或者高特权环中的服务  
第六章 输入输出系统

1. I/O设备：输入输出和存储功能的设备
2. I/O的的基本功能：
3. I/O设备的分类
   1. 按传输的速度：

低速设备（如键盘、鼠标、语音输入输出设备） 中速设备（如行式打印机、激光打印机等）

高速设备（如磁带机、磁盘机、光盘机等）。

* 1. 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。

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。

* 1. 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。

* 1. 设备按其使用特性分类：存储设备、输入＼输出设备

1. I/O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。

控制功能：CPU与设备控制器

数据传输：内存与外设

1. I/O控制方式
   1. 程序I/O方式，使用轮询的可编程I/O方式。CPU浪费
   2. 终端驱动I/O方式，使用中断的可编程I/O方式。CPU用较短的时间进行中断处理。
   3. 直接存储器访问方式(MDA),以数据块为单位，高效。缺点：不连续的数据块，不能一次处理
   4. I/O通道控制方式，通道时硬件，配合着通道程序
2. 设备分配
   1. 前提：大中型计算机
   2. DS：设备控制表、控制器控制表、通道控制表、系统设备表
   3. 独占设备分配步骤：分配设备、分配控制器、分配通道
3. SPOOLing技术（假脱机）
   1. 定义

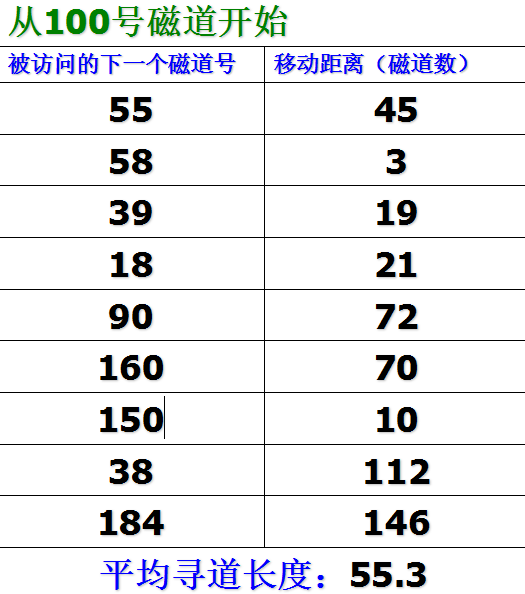
为缓和CPU的高速性与I/O设备低速性间的矛盾而引入了脱机输入、脱机输出技术。该技术是利用专门的外围控制机，将低速设备上的数据传送到高速磁盘上；或者相反。这样就可以在主机的直接控制下实现脱机输入输出。此时外围操作与CPU对数据的处理同时进行，我们把这种在联机情况下实现的同时外围操作称为SPOOLing（Simultaneaus Periphernal Operating On—Line），或称为假脱机操作。

* 1. 组成
     1. 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。
     2. 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。
     3. 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。
     4. 井管理程序。用于控制作业与磁盘井之间信息的交换。
  2. 特点
     1. 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。
     2. 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。
     3. 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。
  3. 打印机是独占设备，通过虚拟技术实现“共享”的模拟

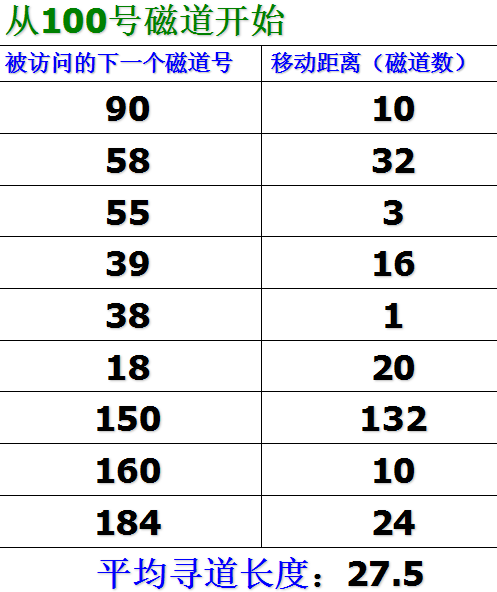
1. 缓冲区管理
   1. 引入
      1. 缓和CPU与I/O设备间速度不匹配矛盾。
      2. 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
      3. 提高CPU和I/O设备之间的并行性。
   2. 方法
      1. 单缓冲（效率低）
      2. 双缓冲区（效率比较高，当输入输出速度不匹配时效率受影响）
      3. 循环缓冲区（解决输入和输出速度相差甚远的影响）
      4. 缓冲池（解决多进程缓冲过程中内存利用率的问题）
2. 磁盘管理

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。

* 1. 先来先服务（FCFS）
     1. 优点：公平、简单
     2. 缺点：未对寻道进行优化



* 1. 最短寻道时间优先（SSTF）
     1. 优点：寻道优化
     2. 缺点：可能导致某些进程发生“饥饿”。



* 1. 扫描SCAN算法
     1. 优点：较好的寻道性能
     2. 缺点：“不巧”的进程严重推迟



* 1. 循环扫描算法CSCAN
     1. 优点：进程的延迟变小了



* 1. FSCAN算法本算法是N-Step-SCAN算法的简化。

## 第七章 文件管理

1. 文件类型
   1. 按用途分类：系统文件、用户文件和库文件。
   2. 按文件中数据的形式分类：源文件、目标文件和可执行文件
   3. 按存取控制属性分类：只执行文件、只读文件和读写文件。
2. 文件操作

最基本的文件操作有：创建文件、删除文件、读文件、写文件、截断文件和设置文件的读/写位置。

1. 文件的逻辑结构。从用户观点出发所观察到的文件组织形式，是用户可以直接处理的数据及其结构，它独立于文件的物理特性，又称为文件组织

文件的物理结构。又称为文件的存储结构，是指文件在外存上的存储组织形式。与存储介质的存储性能和采用的外存分配方式有关

1. 文件的逻辑结构（简答）
   1. 有结构的文件（记录式文件）
      1. 按长度分类：定长记录和不定长分类
      2. 按组织形式分类
         1. 顺序文件

串结构：各记录之间的顺序与关键字无关。通常由时间来决定。

顺序结构：文件中的所有记录按关键字排列。可以按关键字的长短或英文字母顺序排序。顺序结构的检索效率更高

优点：顺序文件的最佳应用场合是在对诸记录进行批量存取时，即每次操作一大批记录。只有顺序文件才能存储在磁带上，并能有效的工作。

缺点：在交互应用的场合，如果进程操作对象是单个记录，顺序文件的性能就可能很差；当文件较大时更差，如想增加或删除一个记录都比较困难。

* + - 1. 索引文件
      2. 索引顺序文件
      3. 直接文件和哈希文件
  1. 无结构的文件（流式文件）

1. 文件的目录结构
   1. 文件目录的要求：实现“按名存取”（最基本的功能，也是引入文件系统主要目的）；提高对目录的检索速度；文件共享；允许文件重名
   2. 文件目录类型（优缺点）
      1. 单级目录（不允许两个文件重名）
      2. 两级目录（缺点：完全隔离）
      3. 多级目录（树型目录）
2. 文件目录的查询技术:线性检索法和Hash方法。

## 第八章 磁盘存储器的管理

1. 文件得物理结构和外设的分配方法

磁盘的分配方式以及对应的物理结构

* 1. 连续组织方式->顺序文件(产生外存碎片)

优点：顺序访问容易；顺序访问速度快。

缺点：要求有连续存储空间;必须事先知道文件的长度。

* 1. 链接组织方式->链接式文件

优点：由于链式分配是采取离散分配方式，消除了外部碎片；因为是根据文件的当前需要，为它分配必须的盘块，当文件动态增长时，可动态的再为它分配盘块，故而无需事先知道文件的大小；对文件的增、删、改也十分方便。

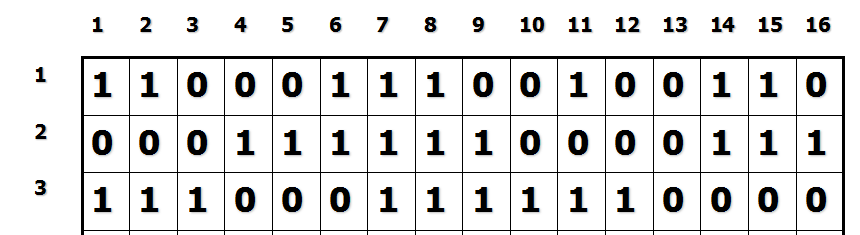
缺点：不能支持高效的直接存取。要对一个文件进行直接存取，需首先在FAT中顺序的查找许多盘块号。

FAT需占用较大的内存空间。当磁盘容量较大时，FAT可能要占用数MB以上的内存空间。这是令人难以忍受的。

* 1. 索引组织方式->索引式文件

可能要花费较多的外存空间。每当建立一个文件时，便须为之分配一个索引块，将分配给该文件的所有盘块号记录于其中。

1. 盘块存储空间的管理
   1. 空闲表法和空闲链表法（和内存分配雷同）
   2. 位式图法（二维数组 下标与盘块号的关系）



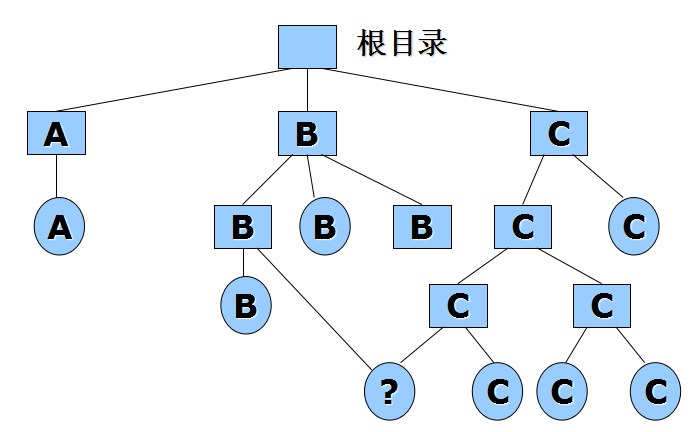
分配： b=列数n\*（i-1）+j;

回收：i=(b-1)DIV n+1 j=(b-1)MOD n+1

优点：从位示图中很容易找到一个或一组相邻接的空闲盘块。此外，由于位示图很小，占用空间少，因而可将它保存在内存中，从而在每次进行盘区分配时，无需首先把磁盘分配表读入内存，从而省掉许多磁盘的启动操作。

* 1. 成组链接法（指针的作用）

1. 文件的共享（保护）
   1. 基于索引结点的共享方式



* 1. 利用符号链实现文件共享

## 接口

1. 用户接口
   1. 脱机用户：批处理、作业控制语言
   2. 联机用户：字符显示式联机用户接口、图形化联机用户接口
2. 程序接口

程序获得OS服务的唯一途径