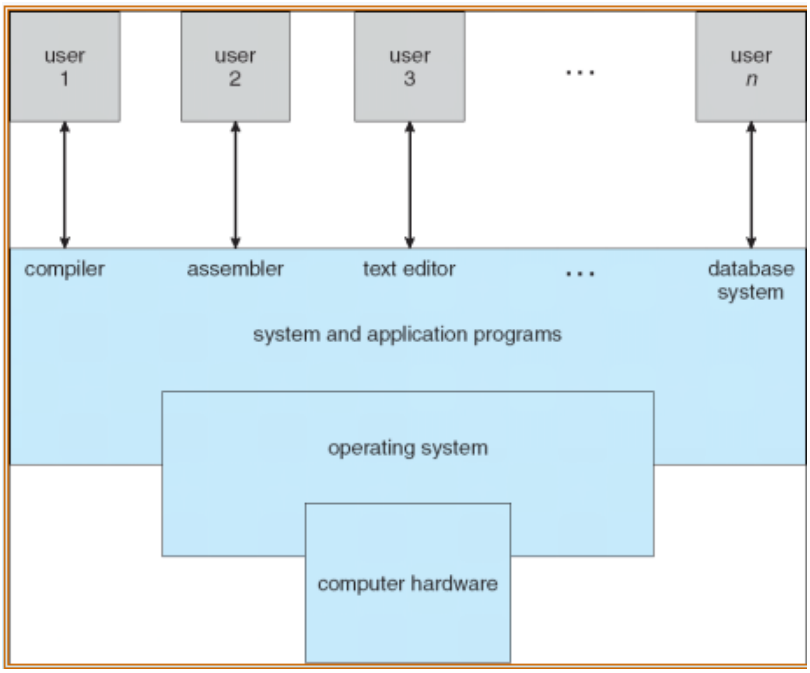
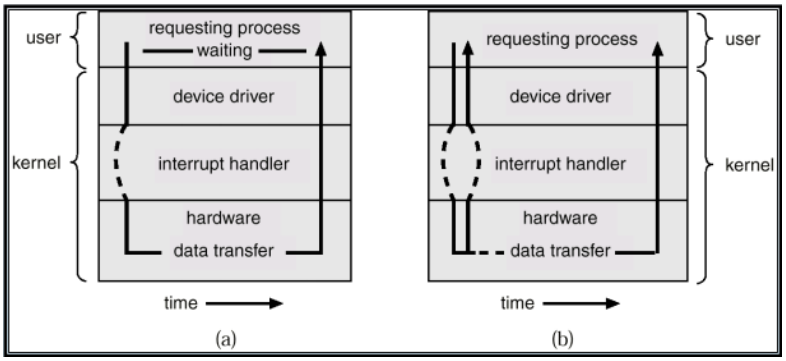
第一章

1.操作系统是管理计算机硬件的程序，应用程序提供基础，并且充当计算机硬件和计算机用户的中介。
2.计算机系统组成：计算机硬件、操作系统、系统程序与应用程序、用户



3.操作系统是一直运行在计算机上的程序（通常称为内核）　　是资源分配者，是控制程序（就是说操作系统本身也是程序）
4.计算机系统的组成：一个或多个 CPU 和若干设备控制器通过共同的总线相连，该总线提供了对共享内存的访问，CPU 与设备控制器并发工作，竞争内存周期。
5.打开电源或重起时运行时需要运行初始化程序----引导程序，引导程序必须定位操作系统内核并把它装入内存
6.事件的发生通常通过硬件或软件中断来表示，操作系统是由中断驱动的，陷阱是一种软件中断，由程序运行错误或用户请求触发
7.存储结构：
易失：寄存器，高速缓存，主存　　可易失可不易失：电子磁盘　　非易失：磁盘，光盘，磁带（速度依次减小）
8.CPU 在本地缓存与主存间转移数据
设备控制器负责在其所控制的外部设备与本地缓冲存储之间转移数据
9.内存是 CPU 可直接访问的唯一大容量存储区域
10.I/O 两种控制方式：①同步：I/O 完成后控制权返还给用户进程；②异步：不等待 I/O 完成直接把控制权返还给用户进程（下图中前者为同步，后者为异步）



11.只有一个通用 CPU 的系统成为单处理系统。
12.装入内存并执行的程序称为进程
13.程序本身不是进程，程序是被动的实体
14.多道程序设计师 CPU 总有一个作业执行，提高了 cpu 的利用率
15.双重模式操作：用户模式，内核模式
当用户应用程序需要操作系统的服务，他必须从用户模式转换到内核模式执行请求
16.当中断或错误出现，通过硬件从用户模式切换至管态

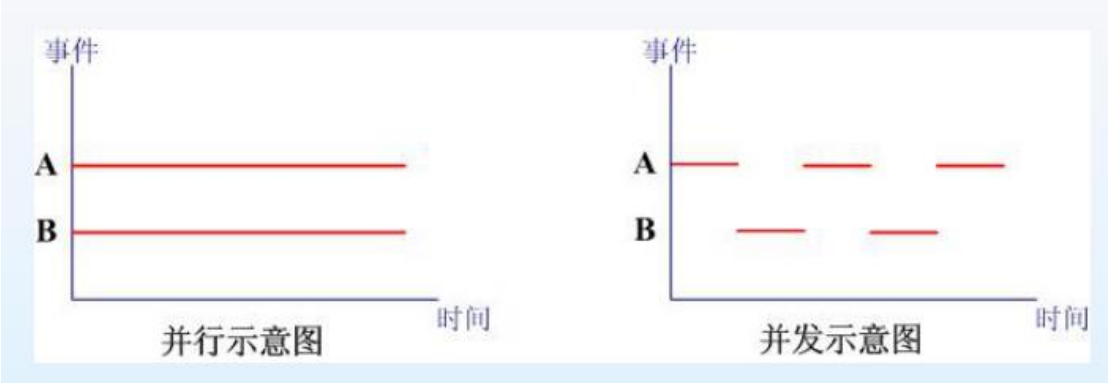
第二章

1.用户界面（用户和控制操作系统的方式）：命令行界面、批界面、图形用户界面（最为常用）
2.命令解释程序执行命令时常用的两种方法：一种方法是命令解释程序本身包含代码以执行这些命令；另一种是系统程序实现大部分命令，命令解释程序只要用命令来识别文件以装入内存并运行（第二种方法增加新的命令不需要更改 shell，如 UNIX 系统中删除文件命令 rm file.txt 会搜索名为 rm 的文件，将该文件装入内存，并用参数 file.txt 来执行，与 rm 命令相关的功能是完全有文件 rm 的代码所决定的。这样程序员能通过创建合适名称的新文件，以轻松地向系统增加新命令，这种命令解释程序可能很小，在增加新命令时不必改变）
3.应用程序接口（API）是一系列适用于程序员的函数，程序员一般根据 API 编程，而不是系统调用
4.向操作系统传参有三种方法：通过寄存器传递参数、参数存在内存的块和表中、参数通过程序放在或压入堆栈中（内存块和堆栈的方法都不限制传递参数的数量）
5.系统程序提供了一个方便的环境，以开发程序和执行程序
6.绝大多数用户所看到的操作系统是由应用和系统程序而不是系统调用所决定的
7.操作系统的结构：
简单结构：利用最小的空间提供最多的功能，但没有仔细的划分成模块，没有很好的区分接口和功能层次
分层法：操纵系统被分成若干层（级），最低层（0 层）是硬件，最高层（N 层）是用户接口，每层只能利用较低层的功能和服务（模块性）
微内核：把内核空间的东西尽可能移到用户空间，在用户模块间利用消息传递来通信。很容易扩展系统、很容易移植平台、高可靠性、高安全性，但用户空间与内核空间通信带来的性能影响
8.虚拟机（Virtual Machine）指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。操作系统造成一种幻觉，好像每个进程都有自己的（虚拟）

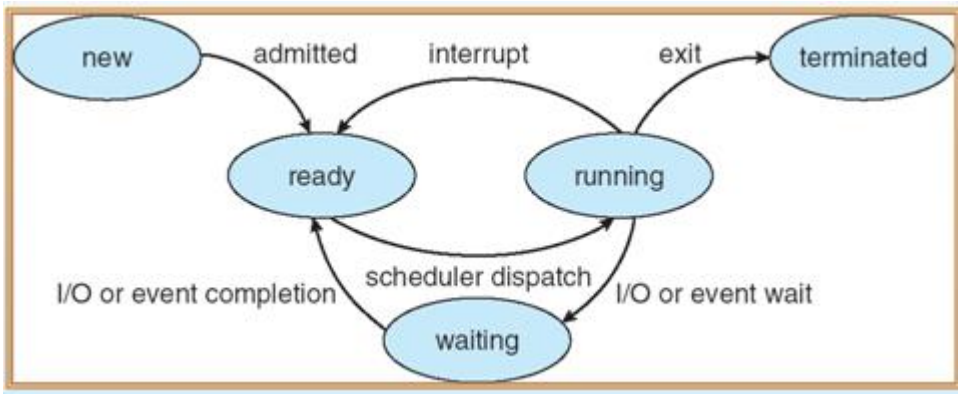
内存和自己的处理机
9.底层机器有两种模式，用户模式和内核模式。虚拟机软件可以运行在内核模式，因为它是操作系统，虚拟机本身只能运行在用户模式。

第三章

1.为了让 cpu 不空闲，应用并发

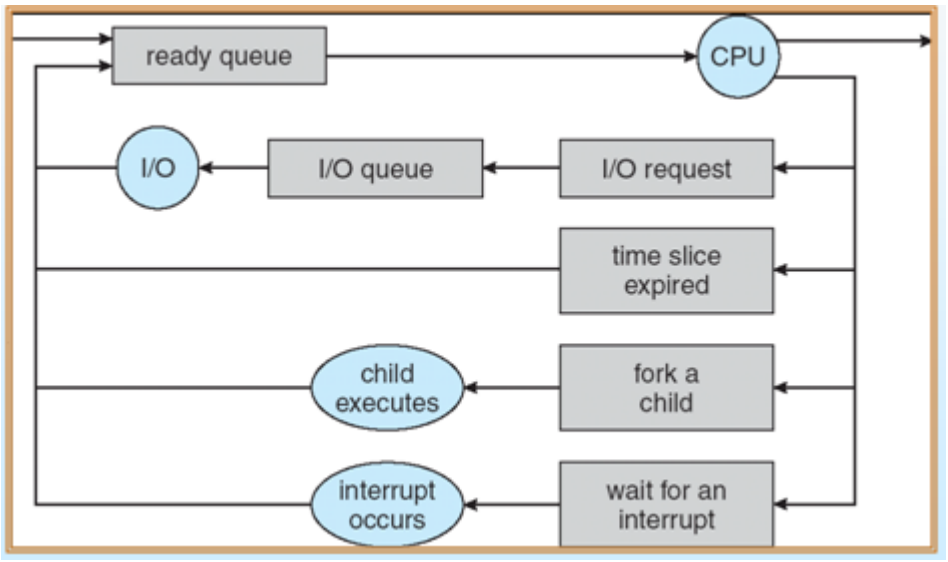


2.进程是执行中的程序
3.进程是一个动态概念，程序是一个静态概念；进程具有并发性，而程序没有
4.不同的进程可以包含同一程序，只要程序所对应的数据集不同（先后打开同一浏览器）
5.一个进程在执行时可能产生很多进程
6.进程状态图：新的、运行、等待、就绪、终止



7.（1）就绪 --> 运行：调度程序选择一个新的进程运行
（2） 运行 --> 就绪：运行进程用完了时间片；运行进程被中断，因为一高优先级进程处于就绪状态
（3）运行 --> 等待
i. OS 尚未完成服务
ii. 对一资源的访问尚不能进行
iii. 初始化 I/O 且必须等待结果
iv. 等待某一进程提供输入 (IPC)
（4） 等待 --> 就绪：当所等待的事件发生时
8. 进城控制块：进程存在的唯一标识；记录了 OS 所需的用于描述进程及控制进程所需的全部信息

9. 进程与 PCB 是一一对应的
10. 设备队列是由链表连接的
11. 下图很好哒解释了程序状态图



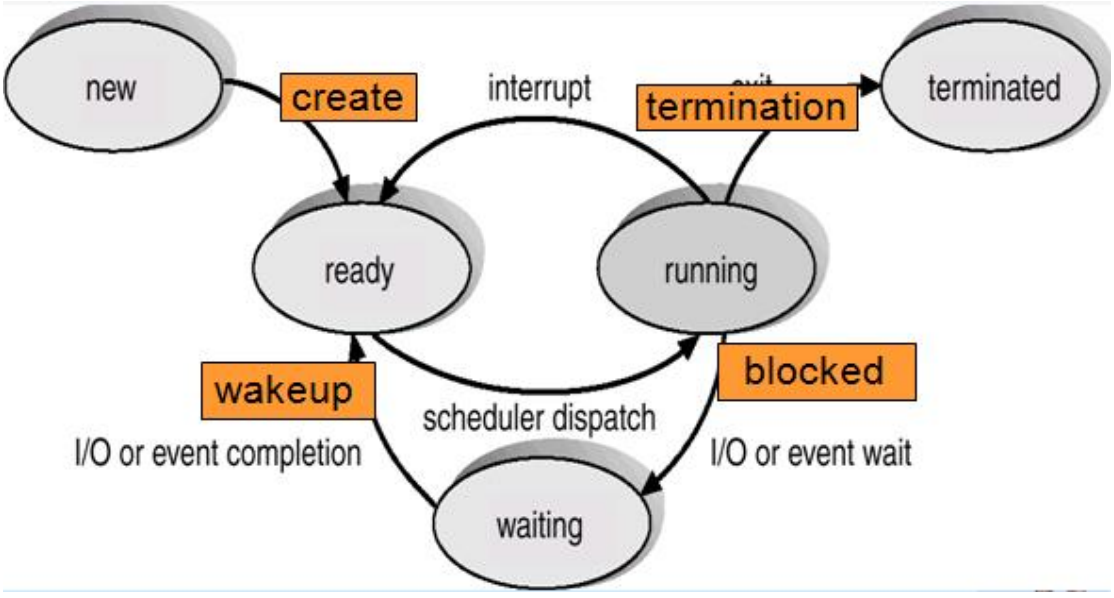
12. 调度程序

长期调度（作业调度）：选择一个进程进入内存的就绪队列，应该选择一个合理的包含 I/o 为主的和 cpu 为主的组合进程。如果进程均是 io 为主的的，那么就需队列几乎为空，从而短期调度程序几乎没有事情可做，如果所有进程均是 cpu 为主，那么 io 等待队列将几乎为空，从而几乎不使用设备，因而系统会不平衡。

短期调度（CPU 调度）：从就绪队列中选择一个进程，并为之分配 CPU（频率较高）

中期调度：核心是将进程从内存中移出

13. 原语：不允许向中断，不与许并发执行的程序段
14.



Create：1、在进程表中增加一项，并从 PCB 池中取一个空白 PCB　2、为新进程分配资源，除内存空间外，还有其他各种资源　3、初始化进程控制块，为新进程分配进程标识符，初始化 PSW 4、加入就绪进程队列
Termination;1、读取 pcb 状态 2、有子进程先终止子进程 3、释放资源 4、移出所在队列
Wakeup:将被唤醒进程置为就绪态，并置于就绪队列
Blocked：请求系统服务：1、启动某种操作 2、新数据尚未到达 3、无新工作可做

15.子进程可能从操作系统那里直接获得资源，也可能只从其父进程那里获得资源，父进程可能必须在其子进程之间分配资源或共享资源，限制子进程只能使用父进程的资源，能防止创建过多的进程带来的系统超载
16.常见的两种通信模式：信息传递和共享内存
17.管道是在内核空间的内存中创建的

第四章

1.进程的两个基本属性：资源的拥有者、调度单位
2.线程：是进程中的一个实体，是独立调度和分派的基本单位。（减少进程切换和创建开销，提高执行效率和节省资源）
3.有两种不同方法来提供线程支持，用户层的用户线程或内核层的内核线程。用户线程受内核支持而无需内核管理，而内核线程由操作系统直接支持和管理。
4.多线程模型：
多对一模型：多个用户级线程映射到一个内核线程，但一旦线程阻塞，进程就阻塞
一对一模型：每个用户级线程映射到一个内核线程，但内核负担大，切换频繁
多对多模型：允许多个用户级线程映射到多个内核线程上
二级模型：一些重要线程采用一对一模型，其他采用多对多
5.取消线程可分为两种情况：
异步取消：一个线程立即终止目标线程
延迟取消：目标线程不断检查它是否应终止，这允许目标线程有机会以有序方式来终止自己

第五章

1.CPU 调度决策可在如下四种环境下发生：
1)当一个进程从运行一个进程从运行态切换到等待态
2)一个进程从运行态到就绪态
3)一个进程从等待态到就绪态
4)一个进程终止
1)和 4)只有 CPU 调度，没有 CPU 选择，2)和 3)可以进行选择；
当调度只能发生在 1)和 4)两种情况下，称调度方案是非抢占的，否则调度方案是抢占的。

2.从进程提交到进程完成的时间成为周转时间，周转时间为所有时间段之和，包括等待进入

内存、在就绪队列中等待、在 CPU 上执行和 IO 执行。

3.等待时间为在就绪队列中等待所花费时间之和。

4. 带权周转时间是\周转时间与执行时间的比：

Wi=Ti/Tri

对于几个作业来说，其平均带权周转时间为：$W=\frac{1}{n}\sum_{i=1}^{n}W_i$

5.需要使 CPU 使用率和吞吐量最大化，而使周转时间、等待时间和响应时间最小化。

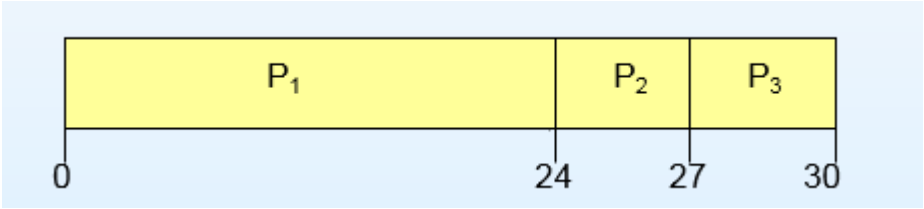
6.调度算法：

1)先到先服务调度（FCFS（非抢占））：先请求 CPU 的进程先分配到 CPU

进程	区间时间
P_1	24
P_2	3
P_3	3

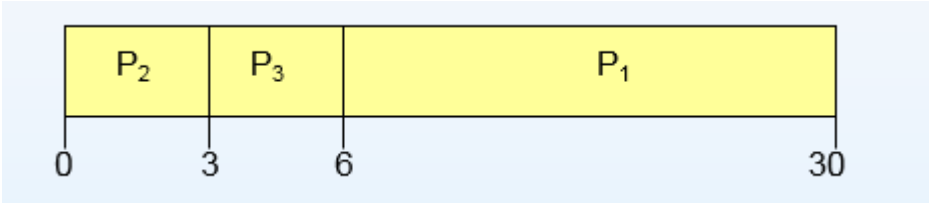
根据到达时间不同，可能有以下两种情况：

A:



平均等待时间: (0 + 24 + 27)/3 = 17

B:



平均等待时间:: (6 + 0 + 3)/3 = 3

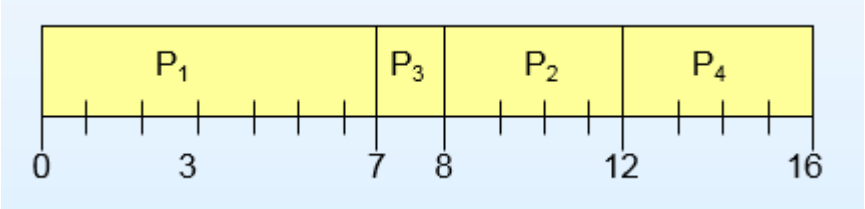
FCFS 根据进程到达时间不同，平均等待时间变化较大。

2)最短作业优先调度（SJF）

SJF 分为抢占 SIF 算法和非抢占 SJF 算法

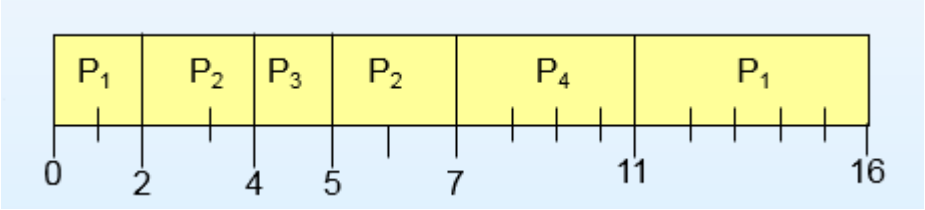
进程	到达时间	区间时间
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

A 非抢占 SJF 算法



平均等待时间：(0 + 6 + 3 + 7)/4 = 4

B 抢占 SIF 算法（最短剩余时间优先调度算法）



平均等待时间：(9 + 1 + 0 +2)/4 = 3

缺点：

a 对长作业不利

b 短作业时间估计的不准确性,影响调度的公平性

3)优先级调度算法

每个进程都有一个优先级与其关联，具有最高优先级的进程会分配到 CPU。具有相同优先级的进程按 FCFS 顺序调度。

SJF 算法属于简单优先级算法。

4)轮转算法（RR）

定义一个较小时间单元为时间片，将就绪队列作为循环队列。CPU 调度程序循环就绪队列，为每个进程分配不超过一个时间片的 CPU。

可能发生两种情况：

A. 进程可能需要小于时间片的 CPU 区间，进程本身会自动释放 CPU，调度程序接着处理就绪队列的下一个进程；

B. 否则会产生操作系统中断，进行上下文切换，将进程加入到就绪队列的尾部，接着 CPU 调度程序，会选择就绪队列中的下一个进程。

5) 最高响应比优先法（HRN）

a.每当要进行作业调度时，系统计算每个作业的响应比，选择其中 R 最大者投入执行。

b.长作业随着它等待时间的增加，W/T 也就随着增加，也有机会获得调度执行。

c.同一时间内，处理作业数：HRN < SJF 从而导致吞吐量 HRN<SJF

d.由于每次调度前都要计算响应比，系统开销也相应增加。

e.响应比 R 定义如下：

R = (W+T)/T = 1+ W/T

其中 T 为作业估计需要的执行时间，W 为作业在后备状态队列中的等待时间。

6)多级调度队列算法

将就绪队列分为多个独立队列，根据进程属性，一个进程被永久地分配到一个队列。每个队列有自己的调度算法。前台队列可能采用 RR 算法调度，而后台队列可能采用 FCFS 算法调度。

第六章

1.独立进程：不能影响或被在系统内执行的其他进程所影响（在逻辑上无任何联系的进程）
2.协作进程：影响或被在系统内执行的其他进程所影响（多个并发进程在逻辑上有某种联系）
3.count++；这行代码由计算机执行时，真正的操作：

```
register1 = count;
register1 = register1 + 1;
count = register1;
```

count；这行代码由计算机执行时，真正的操作：

```
register2 = count
register2 = register2 - 1
count = register2
```

然而“count = 5”，执行 count++；count--；时 count 本应该还是 5， 如果”count++”和”count--”不能分别被保护，那么可能出现以下情况：

S0: producer execute register1 = count {register1 = 5}
S1: producer execute register1 = register1 + 1 {register1 = 6}
S2: consumer execute register2 = count {register2 = 5}
S3: consumer execute register2 = register2 - 1 {register2 = 4}
S4: producer execute count = register1 {count = 6 }
S5: consumer execute count = register2 {count = 4}

竞争条件：多个进程并发访问和操作同一数据且执行结果与访问发生的特定顺序有关
4.临近资源：两个或两个以上的进程不能同时使用的资源
5.临界区：每个进程中访问临界资源的那段代码
6.临界区问题的解答必须满足以下几个要求：
1）互斥：一个进程在临界区执行，其他进程不得进入临界区(无空等待)
2）前进：有空让进，多中选一
3）有限等待：从一个进程做出进入临界区的请求，到该请求允许为止，其他进程允许进入临界区的次数有上界
7.几种有瑕疵的算法：
1）让两个进程共享一个普通整数变量 turn，其初值为 0(或 1)
turn==i 那么 Pi 允许在临界区内执行

```
do{
    while(turn!=i);
    <临界区>
    turn = j;
      <剩余区>
}while(1);
```

但是该算法违背了空闲让进的精神：如当 turn 初值为零时，此时 p0 不申请临界区，但是此时 p1 申请，就造成了，p0 不想进，p1 进不去的状态。
2）用 flag[2]替换 turn

```
do{
      flag[ i ]=true;
      while(flag[ j ]);
```

```
        <临界区>
          flag[ i ]=false;

}
```
但是，当两个进程同时申请临界区时，两个进程都进不去

8.Peterson 算法

```
while (true) {
                    flag[i] = TRUE;
                    turn = j;
                    while ( flag[j] && turn == j);
                        <临界区>
                    flag[i] = FALSE;
                        <剩余区>
          }
```
同时应用了 flag[2]和 turn

9.多进程算法（面包店算法）

```
do{
        choosing[i]=1;//调为取号状态
        number[i]=max(number[0],…,number[n-1])+1;//分派号码
        choosing[i]=0;//取消取号状态
         for(j=0;j<n;j++){
                while(choosing[ j]);//确定所有进程不在取号状态
                while((number[ j]!=0) &&(number[ j],j)<(number[ i],i));如果存在 j 进程申请了临界
区，并且 number[j]<number[i]那么一直循环（意思就是有优先级更高的进程申请了临界区），
        }
         <临界区>//当没有比当前 i 进程优先级更高的进程，那么跳出循环，进入临界区
         number[i]=0;//从临界区出来，变为未取号状态
         <剩余区>
}while(1);
```

10.信号量：P、V 操作

s 是信号量，初值是个整数

P 操作：wait()(s--)　申请一个资源

v 操作: signal()(s++)　释放一个资源

信号量的物理意义：

S>0 表示有 S 个资源可用

S=0 表示无资源可用

S<0，|S|表示 S 等待队列中的进程个数

11.什么时候需要用信号量：

1）需要互斥时

2）保护原语时 （原语：OS 以关中断形式实现的不可打断操作）

12.生产者和消费者

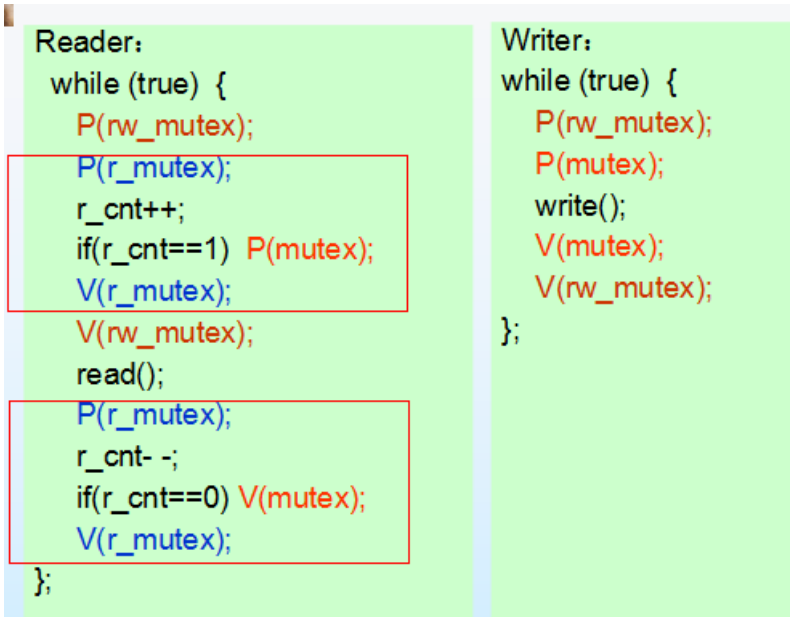
```
P:                                          Q:
While(true){                                While(true){
```

```
        生产一个产品；                          wait(S2);
        wait(S1);                              从仓库中取产品；
        送产品到仓库；                          signal(S1);
        signal(S2);                            消费产品；
};                                      };
```

S1 初值为 1，S2 初值为 0

送产品到仓库和从仓库中取产品，不可能同时发生

13.读写问题：



```
Reader：
 while (true) {
    P(rw_mutex);
    P(r_mutex);
    r_cnt++;
    if(r_cnt==1)  P(mutex);
    V(r_mutex);
    V(rw_mutex);
    read();
    P(r_mutex);
    r_cnt- -;
    if(r_cnt==0) V(mutex);
    V(r_mutex);
 };
```

```
Writer：
while (true) {
    P(rw_mutex);
    P(mutex);
    write();
    V(mutex);
    V(rw_mutex);
};
```

r_cnt:读申请数量

rw_mutex：防止读者或写者饥饿的情况

mutex:保护临界区，读写互斥

r_mutex:保护 r_cnt++--的原语操作

14.管程结构，确保一次只有一个进程能在管程内活动，不存在同步问题

第七章

1.死锁：如果所申请的资源被其他等待资源占有，那么该等待进程再也无法改变其状态。产生的原因是资源不足。

2.死锁发生的必要条件：

（1）互斥：资源不能共享

（2）占有并等待：一个进程必须占有一个资源，并等待另一个资源

（3）非抢占：一个资源只有当持有它的进程完成任务后，自由的释放，中途不可被抢占

（4）循环等待：等待资源的进程之间存在环

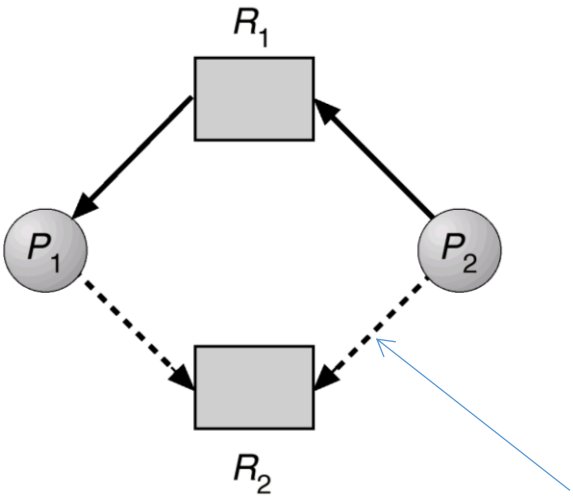
3.没环：不死锁

有环：a.（每类资源单个时死锁）

b.每类资源多个时可能死锁）

4.死锁预防：（中心思想是破坏四个必要条件之一）

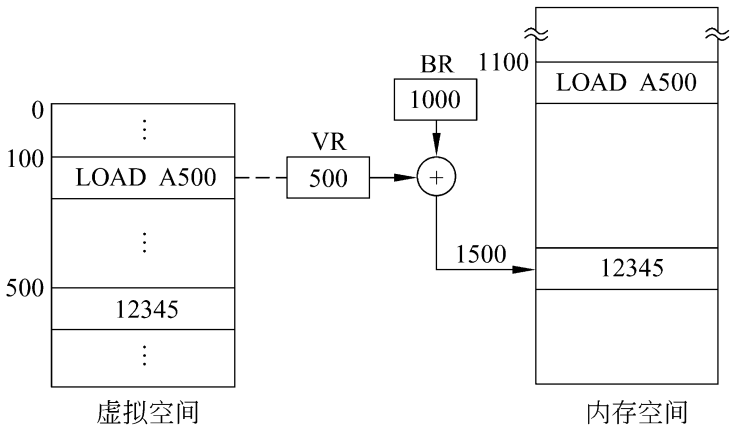
处理互斥：对于可共享资源不存在互斥，然而非共享资源必须保持互斥（通常不能通过互斥条件来预防死锁，有的资源本身就是非共享的）
处理占有并等待：当一个进程申请一个资源是，它不可以占有其他资源
处理非抢占：如果一个进程占有资源，并申请一个另一个不能立即分配的资源，那么其现已分配的资源都可被抢占。换句话说，这些资源都被隐式的释放了
处理循环等待：对所有资源类型进行完全排序，且要求每个进程按递增顺序申请资源
5.死锁避免：每次申请，要求系统考虑现有可用资源，现已分配给每一个进程的资源，和每个进程将来申请与释放的资源，以决定当前申请是否满足或必须等待，从而避免死锁发生的可能性。（当一个进程申请资源时，系统要考虑，如果给分配该资源是否还存在安全序列，如存在，则分配，如不存在，则不分配。）
6.死锁避免算法：动态监测资源分配状态，以确保循环等待条件不可能成立（保证没有环出现）
7.如果存在一个安全序列，那么系统处于安全态
8.两种死锁避免的算法：
a.



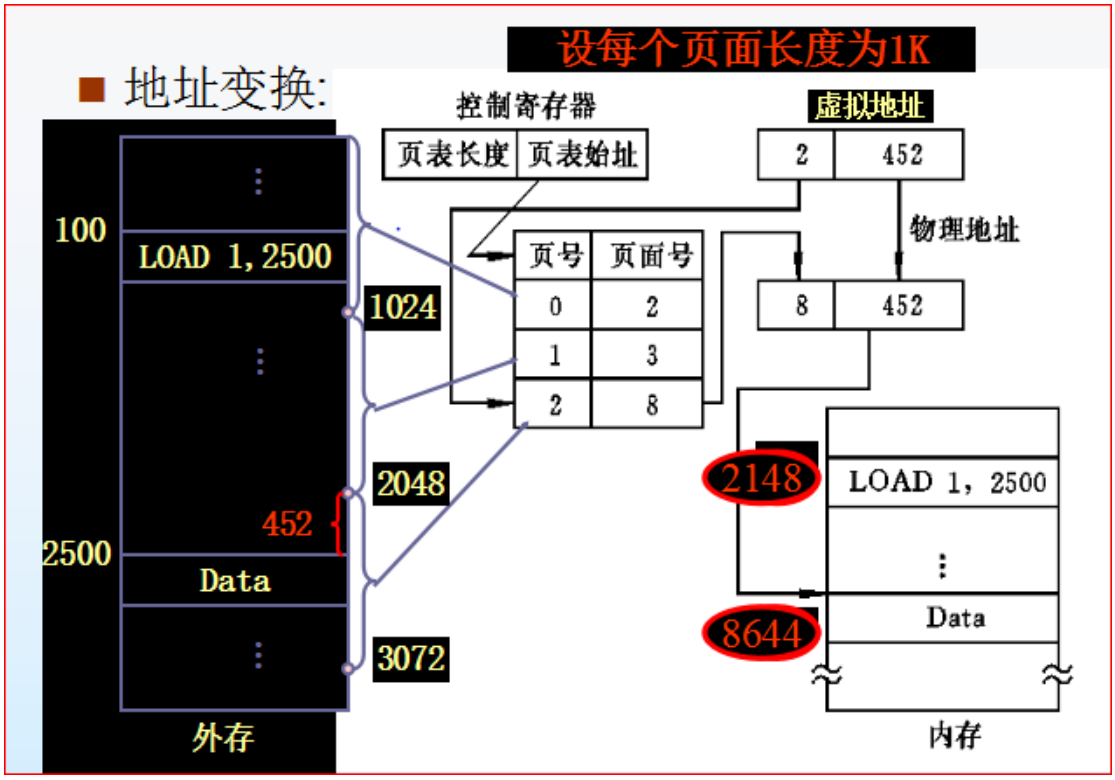
假设进程 p2 申请资源 r2，虽然 r2 现在可用，但是不能将它分配给 p2,因为会创建一个环，环表示系统处于不安全状态。如果 p1 申请 r2，且 p2 申请 r1 会发生死锁。
b.银行家算法：即使资源可以满足某一进程的当前请求，也不一定分给它，关键是看分给它后系统是否还是安全状态
详解见操作系统教材 223 页
9.如果一个操作系统既不采用死锁预防算法，也不采用死锁避免算法，就需要系统提供检测和恢复的算法，但死锁的检验和恢复会有额外开销
10.死锁检测：测试是否存在环的算法
11.死锁恢复：
通过取消进程来取消死锁：
（1）终止所有进程：代价大
（2）一次终止一个进程知道取消死锁循环为止：开销也大
通过资源抢占来取消死锁：
逐步从进程中抢占资源给其它进程使用，直到死锁环被打破为止

第八章

1.cpu 所能直接访问的存储器只有内存和处理器内的寄存器。
2.运用基地址寄存器和界限地址寄存器来寻址，来防止用户程序球修改系统或其他用户的代码或数据结构。（实现保护作用）
3.只有操作系统可以听过特殊的特权指令来加在基地址寄存器和界限地址寄存器
4.程序必须放入内存中的进程空间才能被执行
5.指令和数据结合到内存地址（地址绑定）可以在三个不同的阶段发生：
（1）编译时期：如果内存位置已知，可生成绝对代码；如果开始位置改变，需要重新编译代码
（2）加载时期：如果存储位置在编译时不知道，则必须生成可重定位代码
（3）执行时期：如果进程在执行时可以在内存中移动，则地址绑定要延迟到运行时。需要硬件对地址映射的支持，例如基址和限长寄存器
6.动态地址重定位



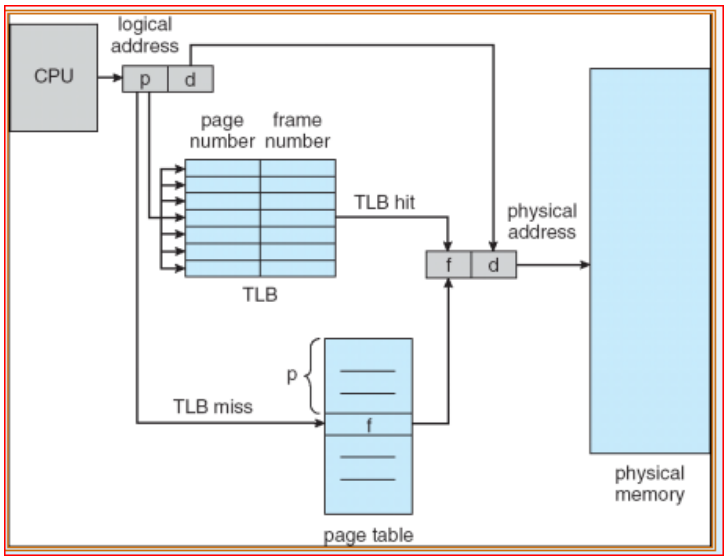
在它执行前，由操作系统将 BR 置为 1000。当程序执行到 100 号单元处的指令时，CPU 给出的取数地址为 500（load a500 是一条取数指令），而希望访问的数据存放在 1500 号单元内，即经动态重定位地址变换为 1500，以它作为访问主存的物理地址。
7.分区管理的基本原理：
a.固定分区法:划分的原则由系统操作员或操作系统决定。分区一旦划分结束，在整个执行过程中每个分区的长度和内存的总分区个数将保持不变
缺点：管理简单，系统开销小；不灵活、大程序可能无法装入；利用率低，浪费，内部碎片大
b.动态存储分配：
首次适应，最佳适应，最差适应
（详见教材 245 页）
8.分页：
将物理内存分为固定大小的块（称为帧）
将逻辑内存分成同样大小的块（称为页）
执行一个大小为 n 页的进程，要发现 n 个空闲帧并把程序装入其中
利用页表进行逻辑到物理地址的映射
9.例题



将 load 1,2500 加载到内存（注：页面长度为 1k=1024）：通过页表可知 load 1,2500 语句将存在第二帧内，又因为偏移量为 100，所以所存的物理地址为 2*1024+100=2148

同理 8644=8*1024+452

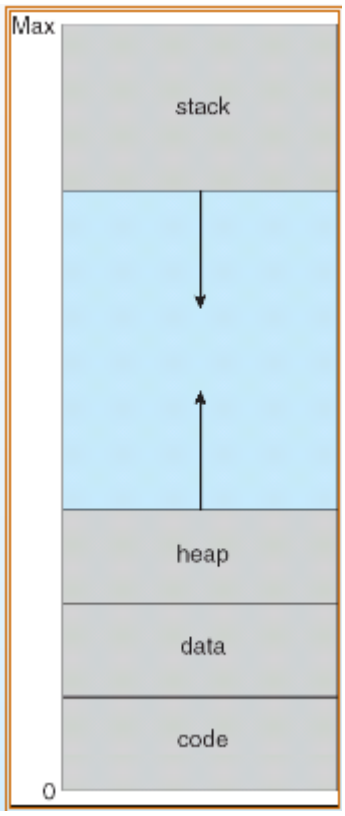
10.转换表缓冲器：（TLB）包括了页表中的一小部分条目，当 cpu 产生逻辑地址后，首先将页号提交给 TLB，如果该页号在页表中，那么命中，如果不在，再访问页表



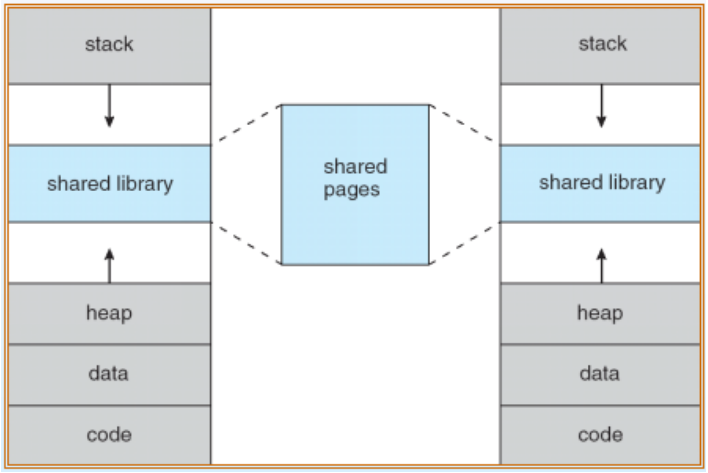
11.页表和段表区别在于分块方式不同，页表是把程序直接分成快，段表是把把程序按内容或者过程关系分成段，每段有自己的名字。

第九章

虚拟内存、物理内存-hfm_honey-ChinaUnix 博客
http://blog.chinaunix.net/uid-26983585-id-3364091.html（大家可以看一看这篇文章）
1.虚拟内存：将用户逻辑内存和物理内存区分开， 执行时只有部分程序需要在内存中，不执行的存在外存中（匀出一部分外存空间来当内存使用）
2.虚拟地址空间：

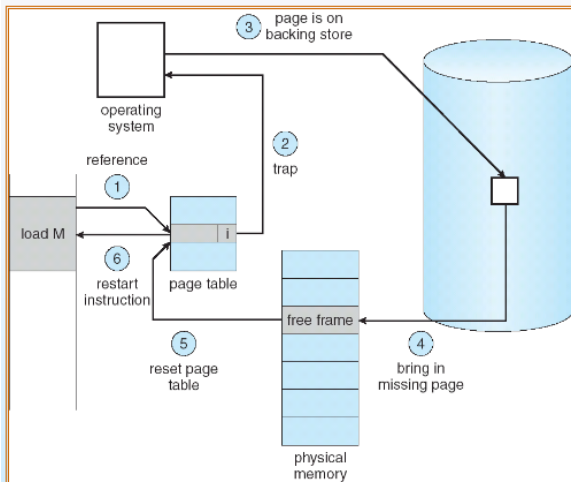


这是一个进程的私有虚拟内存空间，其中包括栈，堆，数据等参数
3.中间的 shared page（共享页）就是两个进程共享的资源

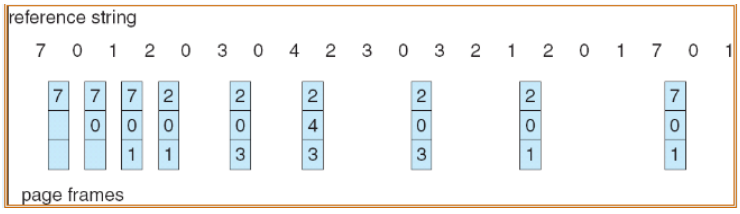


4.实现虚拟存储技术应注意
(1)需要有相当容量的辅存以便于存放多用户作业的地址空间。

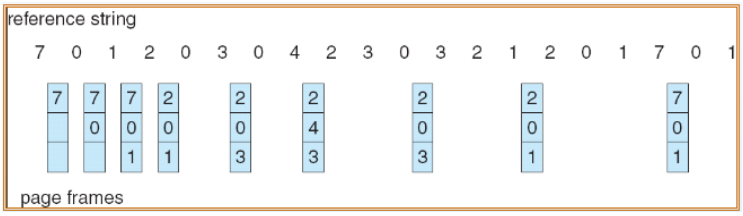
(2)要有一定容量的主存。
(3)地址变换机构
5.动态页式管理在作业或进程开始执行之前，不把作业或进程的程序段和数据段一次性地全部装入内存，而只装入经常反复执行和调用的部分。（需要哪个就将哪个调入内存）
6.懒惰交换：只在需要页时，才被调入内存
（1）交换程序：针对进程
（2）调页程序：针对单个页
7.有效-无效位：区分页是否在内存里
当访问标记位为无效的页时，会产生页错误陷阱
8.



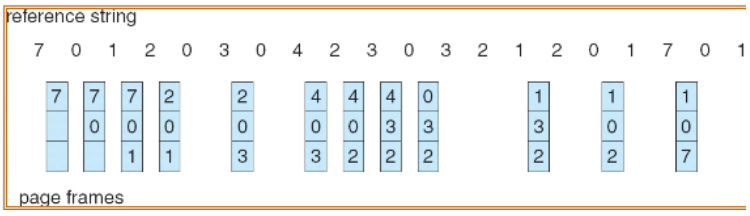
①②当访问无效页时，会陷入 OS。
③④操作系统将所需页从外训调入内从中
⑤修改页表信息
⑥返回中断
9.按需调页的性能：p 为页错误率，（0<=p<=1）(希望 p 接近于 0)
若设内存访问时间为 ma
则，有效访问时间=(1-p)*ma+p*也错误时间
10.写时复制（图片详见教材 279 页）
两个进程有共享的页面，这些页面标记为写时复制页，如果有进程要对这些页进行改写，那么就创建一个该共享页的副本
11.没有空闲页了，要换出某些用不到的页，这种情况下如何选择被替换的页，有如下几种算法：
（1）FIFO:
最早来的先走



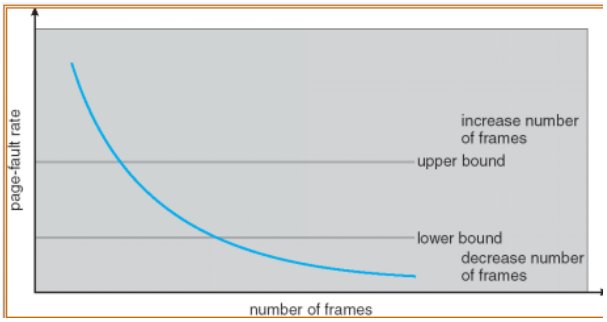
Belady 异常：如果增加页帧中的帧数，可能会增加错误率
（2）OPT 最优置换算法：
置换最长时间不会使用的页（引用串已知）



（3）LRU（最近最少使用算法）：
替换最长时间没有使用过的页，每个页关联该页上次使用的时间



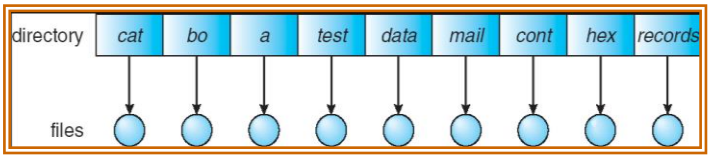
（2）（3）不会出现 belady 异常，栈算法是绝对不会出现 belady 异常的
12.如果一个进程没有足够的帧，缺页率会很高。颠簸就是一个进程频繁的换入换出页
13.产生页错误的进程，必须使用调页设备以换进和换出页，随着他们排队等待换页设备，就绪队列会变空，而进程等待调页设备 cpu 使用率就会降低。
14.不出现 belady 异常的正常状况，帧数越大，错误率越低



15.系统颠簸：
一个进程进入一个新的执行阶段，需要更多的帧，----->出现也错误，并从其他进程中拿到帧------->这些进程也需要这些页，也会出现也错误------>这些页错误进程必须使用调页设备以换进和换出页------->就绪队列变空，cpu 使用率降低------>增加多道程序的程度------->新进程试图从其他运行进程中拿到帧------>从而一起更多页错误，形成更长的调页设备的队列------->cpu 使用率进一步降低
16.局部模型：固定的程序会在，固定的内存区内执行

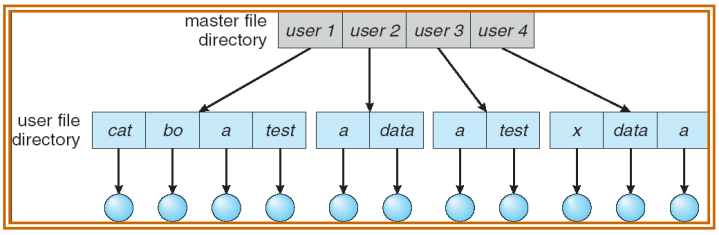
第十章

1.顺序访问：文件按信息顺序，一个记录接着一个记录的加以处理。
2.直接访问：磁盘允许对任意文件块进行随机读和写
3．单层目录结构：所有文件都包含在同一目录中，其特点是便于理解和支持

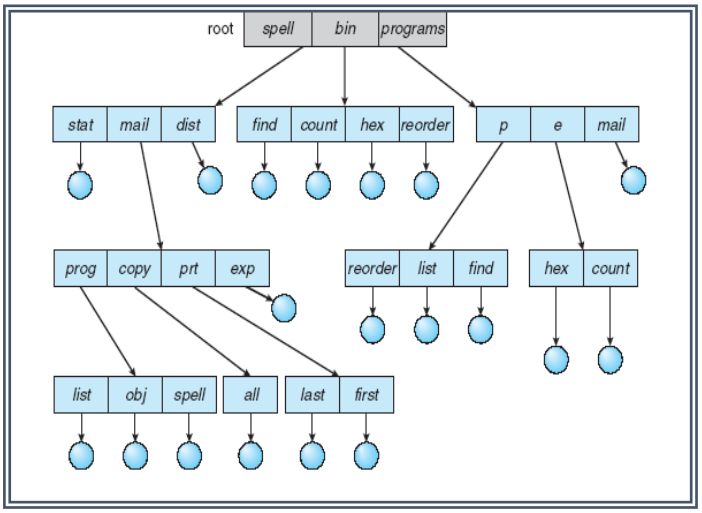


缺点：因为在同意目录下，他们必须有唯一名称

4.双层目录结构：不同用户可以为文件取相同名字，



5.树状结构目录：



绝对路径名：从根开始并给出路径上的目录名，知道所指定的文件。

相对路径名：从当前目录开始定义路径。

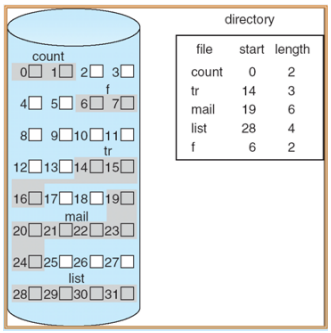
如果当前目录是 root/spell/mail, 那么相对路径名 prt/first 与绝对路径名 root/spell/mail/prt/first 指定同一文件

6. 如何保证无环：只允许链接向文件，不允许向子目录；每当创建一个新的链接时，调用环路检测算法判断是否有环

第十一章

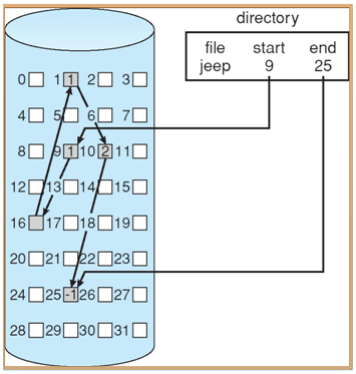
1. 每创建一个文件，就会分配一个新的 fcb（文件控制块，包含文件的各种信息）
2. 分配方式：

   （1）连续分配：要求每个文件在磁盘上占有一组连续的块（寻道数最小），困难是，很难为新文件找到空间。

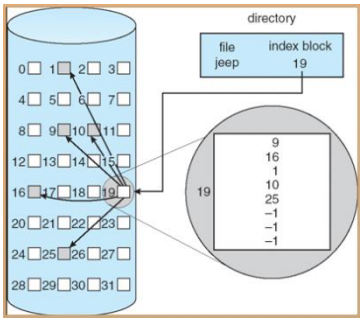


每个文件存储的空间是连续的

（2）连接分配：解决了连续分配的所有问题，目录包含第一块和最后一块的指针
每一块都有指向下一个块地指针



（3）索引分配：找出一个块作为索引块保存所有块的指针，及顺序



3．空闲空间管理：

（1）位向量：设立标志位，0 代表已分配，1 代表空闲

（2）链表：空闲块用链表连接起来，并将指向第一空闲快的指针保存在磁盘的特殊位置，同时也缓存在内存中，第一块包含下一空闲块的指针

（3）组：将 n 个空闲块的地址存在第一个空闲块中，这些块中的前 n-1 个确实为空，而最后一块包含另外 n 个空闲快的地址。