目录

第一章

1. 操作系统的目标

有效性(系统管理角度)：管理和分配硬软件资源，合理地组织计算机的工作流程

方便性(用户角度)：提供良好的用户接口

可扩充性(系统角度)：OS 必须具有很好的可扩充性，方能适应发展的要求(便于增加新的功能层次和模块，并能修改老的功能层次和模块)

开放性(互操作角度)：指系统能遵循世界标准规范，特别是遵循开放系统互连 OSI 国际标准

1.3 操作系统的基本特征

并发(concurrence)：多个事件在同一时间段内发生

共享(sharing)：共享是指系统中的所有资源可供内存中的多个并发执行的进程(线程)共同使用，而这种资源共同使用称为资源共享(资源复用)。（进程用于并发，线程用于减少进程切换时的时间开销）。根据资源属性不同，可有互斥共享和同时访问两种不同的共享方式。

虚拟(virtual)：通过某种技术把一个物理实体映射为若干个对应的逻辑实体。可提高资源利用率。虚拟技术包括时分和空分

异步性(asynchronism)：也称不确定性，指进程的执行顺序和执行时间的不确定性

微内核：

第二章

程序并发执行时的特征：

间断性(异步性)

程序并发执行时，由于共享系统资源和相互合作，形成了相互制约的关系，导致“执行 暂停 执行 ”这种间断性的活动

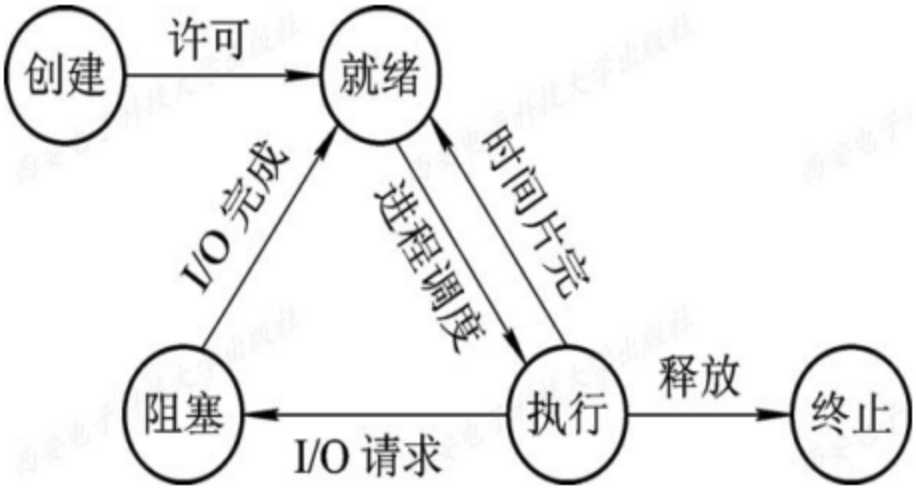
失去封闭性（无法由一个进程独占资源）

多个程序共享系统资源，因而资源的状态将由多个程序来改变

不可再现性

失去了封闭性，也将导致其失去可再现性

图 2-6 进程的五种基本状态及转换：



4. 同步应遵循的规则

空闲让进：当无进程处于临界区时，表明临界资源处于空闲状态，应允许一个请求进入临界区的进程立即进入自己的临界区，以有效地利用临界资源。

忙则等待：当已有进程进入临界区时，表明临界资源正在被访问，因而其它试图进入临界区的进程必须等待，以保证对临界资源的互斥访问。

有限等待：对要求访问临界资源的进程，应保证在有限时间内能进入自己的临界区，以免陷入“死等”状态。

让权等待：当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙等”状态。

下图给出了一个前趋图，其中S_1，S_2，S_3，…，S_6是最简单的程序段。为使各程序段能正确执行，应设置若干个初始值为“0”的信号量。

为保证$S_1 \rightarrow S_2$，$S_1 \rightarrow S_3$的前趋关系，分别设置信号量**a** 和**b**，同样，为保证$S_2 \rightarrow S_4$，$S_2 \rightarrow S_5$，$S_3 \rightarrow S_6$，$S_4 \rightarrow S_6$和$S_5 \rightarrow S_6$，设置信号量**c**，**d**，**e**，**f**，**g**。

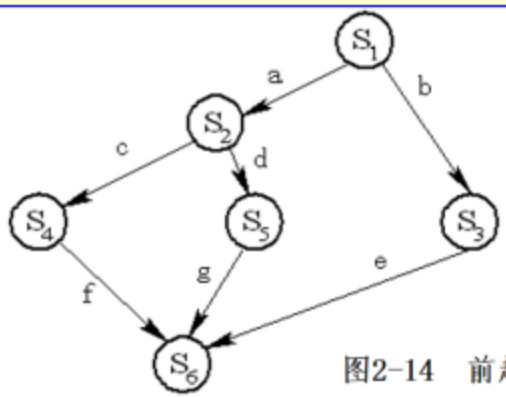


图2-14 前趋图举例

```
P1() {S1;signal(a);signal(b);}
P2(){wait(a);S2;signal(c);signal(d);}
P3() {wait(b);S3;signal(e);}
P4() {wait(c);S4;signal(f);}
P5() {wait(d);S5;signal(g)}
P6() {wait(e);wait(f);wait(g);S6}
Main() {
        semaphore a,b,c,d,e,f,g;
        a.value=b.value=c.value=0;
        d.value=e.value=0;
        f.value=g.value=0;
        cobegin
                p1(); p2(); p3(); p4(); p5(); p6();
        coend
}
```


生产者—消费者(producer_consumer)问题

生产者—消费者(producer_consumer)问题

- **问题描述**

  一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费，为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池，生产者进程将它所生产的产品放入一个缓冲区中；消费者进程可从一个缓冲区中取走产品去消费。

- 所有生产者进程和消费者进程都是以异步方式运行，但它们之间必须保持同步，即不允许消费者进程到一个空缓冲区去取产品；也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品。

进程管理

单缓冲区的生产者—消费者问题

Var empty，full: semaphore = 1, 0；

```
Proceducer：
while (true) {
    生产一个产品;
    P(empty) ;
    送产品到缓冲区;
    V(full);
 };
```

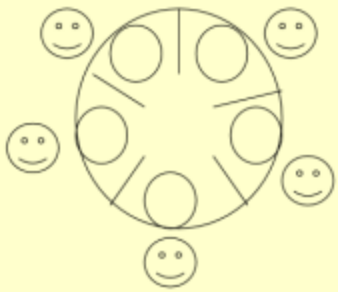
```
Consumer：
while (true) {
    P(full);
    从缓冲区取产品;
    V(empty);
    消费产品;
 };
```


2.5.2 哲学家进餐问题

进程管理

2.5.2 哲学家进餐问题

- 5个哲学家围坐在圆桌旁，每人面前有一只空盘子，每2人之间放一只筷子，如下图所示。
- 为了就餐，每个哲学家必须拿到两只筷子，并且只能直接从自己的左边或右边去取筷子。
- 放在桌子上的筷子是临界资源，可用一个信号量表示一只筷子，五个信号量构成信号量数组。



Semaphore chopstick[5]={1,1,1,1,1}；

```
semaphore mutex[5] = {1,1,1,1,1};        //初始化信号量
semaphore count = 4;     //控制最多允许四位哲学家同时进餐

void philosopher(int i){
  do {
    //thinking       //思考
    p(count);        //判断是否超过四人准备进餐
    P(mutex[i]);     //判断缓冲池中是否仍有空闲的缓冲区
    P(mutex[(i+1)%5]);//判断是否可以进入临界区（操作缓冲池）
    //...
    //eat            //进餐
    //...
    V(mutex[i]);//退出临界区，允许别的进程操作缓冲池
    V(mutex[(i+1)%5]);//缓冲池中非空的缓冲区数量加1，可以唤醒等待的消费者进程
    V(count);//用餐完毕，别的哲学家可以开始进餐
  }while(true);
}
```

6.解决哲学家进餐问题—方法二

第二种方法，也就是使用AND型信号量，同时对哲学家左右两边的筷子同时申请。下面是伪代码：

```
semaphore mutex[5] = {1,1,1,1,1};        //初始化信号量

void philosopher(int i){
  do {
    //thinking       //思考
    Swait(mutex[i], mutex[(i+1)%5]);//判断哲学家左边和右边的筷子是否同时可用
    //...
    //eat
    //...
    Ssignal(mutex[i], mutex[(i+1)%5]);//进餐完毕，释放哲学家占有的筷子
  }while(true);
}
```

进程管理

2.5.3 读者—写者问题

- “读者一写者(Reader—Writer Problem)问题”是指保证一个Writer进程必须与其他进程互斥地访问共享对象的同步问题。

- 读者一写者的要求：
  - 允许*多个读者同时执行读操作*
  - 不允许读者、写者同时操作
  - 不允许多个写者同时操作

读者优先

1.初始化

```
semaphore wmutex=1;//实现对文件的互斥访问，表示当前是否有进程在访问共享文件
int readcount=0;//记录当前有多少个读进程在访问文件
semaphore rmutex;//用于保证对readcount变量的互斥访问
```

2.写者

```
writer(){
    while(1){
        P(wmutex);//写之前加锁
        写文件...
        V(wmutex);//写之后解锁
        }
    }
```

3.读者

```
reader(){
    while(1){
        P(rmutex);//各读进程互斥访问readcount
        readcount++;//访问文件读进程数+1
        if(readcount==1)
            P(wmutex);//写进程加锁，不允许在读操作过程中执行写操作
        V(rmutex);
        读文件...              //上面这一部分使得多个读者能够同时访问
        P(rmutex);//各读进程互斥访问raeadcount
        readcount--;//每当一个读进程完成读操作，读者数量-1
        if(readcount==0)
            V(wmutex);//当没有读者，读操作结束后，写进程解锁
        V(rmutex);
        }
    }
```

☆第三章

作业调度、进程调度：

1、☆先来先服务(FCFS)（在进程调度中为时间片轮转调度算法）

按照作业到达的先后次序来进行调度。

时间片轮转调度算法：时间片 q 表示，给每个进程 q 秒的时间，如果完成了，就退出，如果没有完成，就把这个进程放到就绪队列的队尾。

题目给出进程到来时间表

	A	B	C	D	E
到达时间(第)	0	1	2	3	4
服务时间	4	3	4	2	4

然后算出就绪队列、运行情况

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
就绪队列	A	B	C	D	E	A	B	C	D	E	A	B	C	E	A	C	E	
		A	A	A	A	B	C	D	E	A	B	C	E	A	C	E		
			B	B	B	C	D	E	A	B	C	E	A	C	E			
				C	C	D	E	A	B	C	E	A						
					D	E	A	B	C									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
运行情况	A_0	A_1	A_1	A_1	A_1	A_1	A_2	A_2	A_2	A_2	A_2	A_3	A_3	A_3	A_3	A_4		
		B_0	B_1	B_1	B_1	B_1	B_1	B_2	B_2	B_2	B_2	B_2	B_3					
			C_0	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_3	C_3	C_3	C_4	
				D_0	D_1	D_1	D_1	D_1	D_1	D_2								
					E_0	E_1	E_1	E_1	E_1	E_1	E_2	E_2	E_2	E_2	E_3	E_3	E_3	E_4

红色:即将执行；蓝色:完成

At: A已经执行了t个时间片

说明：进程在第 i 秒到达后，加入队头/队尾（如就绪队列第 7 秒，C 是队头，B 是队尾）

然后算出其他时间（周转时间、等待时间、带权周转时间）

案例：当时间片为q=1时

	A	B	C	D	E
到达时间(第)	0	1	2	3	4
服务时间	4	3	4	2	4
完成时间(第)	15	12	16	9	17
周转时间(完成-到达)	15	11	14	6	13
等待时间(周转-服务)	11	8	10	4	9
带权周转时间(周转/服务)	15/4=3.75	11/3	14/4=3.5	6/2=3	13/4=3.25

2、短作业优先(short job first，SJF)的调度算法：
SJF 算法是以作业的长短来计算优先级，作业越短，其优先级越高。（越先进入就绪队列）

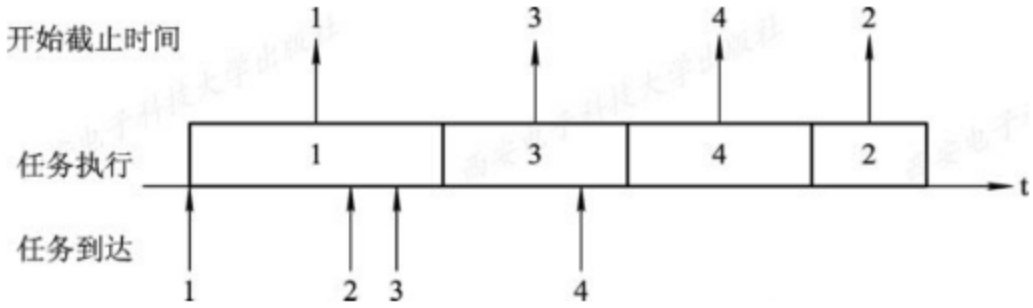
作业号	提交时间	运行时间	开始时间	等待时间	完成时间	周转时间	带权周转时间
1	8	2	8	0	10	2	1
2	8.4	1					
3	8.8	0.5					
4	9	0.2					

如图，在作业 1 完成后，时间片来到 10 秒，此时作业 2、3、4 都来了，所以选择运行时间最短的作业 4，先运行。

作业号	提交时间	运行时间	开始时间	等待时间	完成时间	周转时间	带权周转时间
1	8	2	8	0	10	2	1
2	8.4	1					
3	8.8	0.5					
4	9	0.2	10	1	10.2	1.2	6

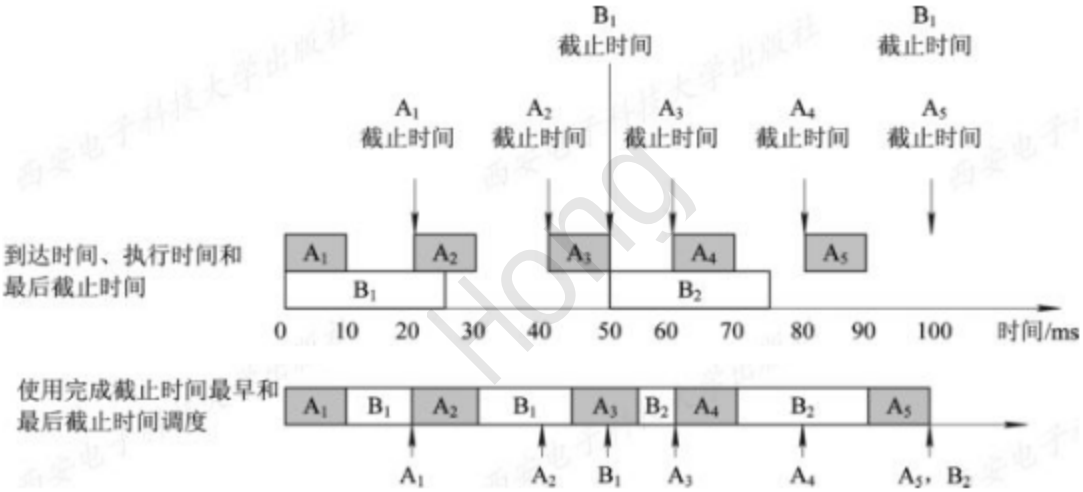
实时调度

3、EDF 算法（最早截止时间优先）



非抢占式中，1 任务到达，到执行完成。1 完成时，2 和 3 已经到达，3 的截止时间最早，所以先运行 3。

抢占式与非抢占的区别就是，在下一个任务到来时，是否需要执行完当前的任务。比如在抢占式中，1 任务到达，到执行完成的过程中，2 到达了，这时候就需要判断 1 和 2 的截止时间（周期时间）谁早，如果 2 早，则 2 要先执行。如果两者截止时间相同，则优先不切换任务。



4、LLF 算法（最低松弛度优先）

松弛度：最晚完成时间-当前进程未完成时间-当前时间

作业到来时，谁的松弛度低，优先为谁服务。

3.1.1 处理机调度的层次

1. 高级调度(High Level Scheduling)

要花费时间比较长的调度，比如作业调度

2. 低级调度(Low Level Scheduling)

对进程做相应的处理，比如进程调度

3. 中级调度(Intermediate Scheduling)

比如页面和块的调度

$$\text{CPU 的利用率} = \frac{\text{CPU 有效工作时间}}{\text{CPU 有效工作时间} + \text{CPU 空闲等待时间}}$$

产生死锁的必要条件

只要其中任一个条件不成立，死锁就不会发生：

(1) 互斥条件。

(2) 请求和保持条件。

(3) 不可抢占条件。

(4) 循环等待条件。

3. ☆处理死锁的方法

目前处理死锁的方法可归结为四种：

(1) 预防死锁。

预防死锁的方法是通过破坏产生死锁的四个必要条件中的一个或几个，以避免发生死锁。由于互斥条件是非共享设备所必须的，不仅不能改变，还应加以保证，因此主要是破坏产生死锁的后三个条件。

破坏“请求和保持”条件（1或2）

1、所有进程在开始运行之前，必须一次性地申请其在整个运行过程中所需的全部资源

2、该进程在请求新的资源时，它不能持有不可抢占资源。

破坏“不可抢占”条件

1、当一个已经保持了某些不可被抢占资源的进程，提出新的资源请求而不能得到满足时，它必须释放已经保持的所有资源

破坏“循环等待”条件

1、对系统所有资源类型进行线性排序，并赋予不同的序号。

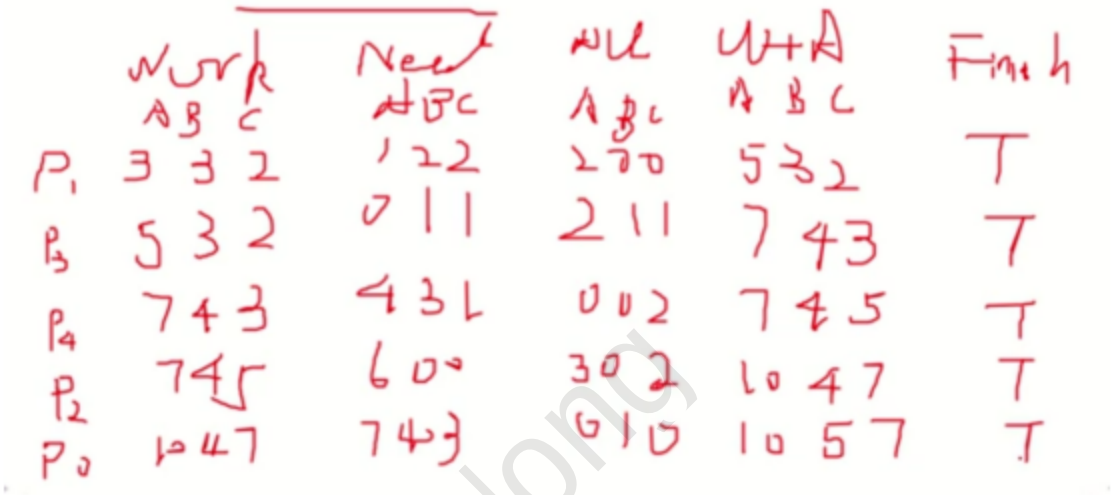
预防死锁可能需要对资源和进程进行严格的控制，增加系统开销和复杂性。

(2) 避免死锁。

在资源动态分配过程中，防止系统进入不安全状态，以避免发生死锁。

银行家算法：

资源 / 情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3 (2	3 3	2 0).
P_1	3	2	2	2 (3	0 0	0 2)	1 (0	2 2	2 0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			



T0 时刻的安全性：利用安全性算法对 T0 时刻的资源分配情况进行分析(如图 3-16 所示)可知，在 T0 时刻存在着一个安全序列{P1, P3, P4, P2, P0}，故系统是安全的。

避免死锁需要资源分配算法进行动态分析和预测，可能涉及较高的计算开销。

(3) 检测死锁。

(4) 解除死锁。

1. 终止进程的方法

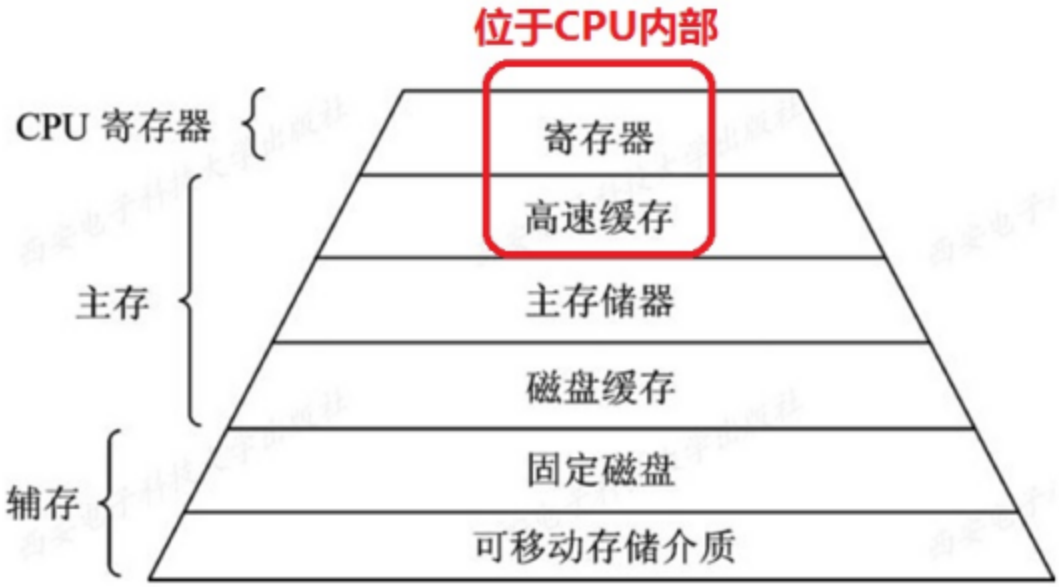
1) 终止所有死锁进程（代价大。进程可能已经运行了很长时间，已接近结束，一旦被终止真可谓“功亏一篑”）

2) 逐个终止进程（代价可能小。要用死锁检测算法确定系统死锁是否已经被解除，若未解除还需再终止另一个进程）

检测和恢复死锁需要定期进行死锁检测，这会占用系统资源，并且在发生死锁时执行恢复操作可能会引入一定的延迟和性能影响。

第四章

存储器的层次结构



☆程序的装入和链接（简答题）

链接：由链接程序(Linker)将编译后形成的一组目标模块以及它们所需要的库函数链接在一起，形成一个完整的装入模块(Load Module)；

装入：由装入程序(Loader)将装入模块装入内存。

4.2.1 程序的装入

1. 绝对装入方式(Absolute Loading Mode)

当计算机系统很小，且仅能运行单道程序时，完全有可能知道程序将驻留在内存的什么位置。此时可以采用绝对装入方式。用户程序经编译后，将产生绝对地址(即物理地址)的目标代码。

2. 可重定位装入方式(Relocation Loading Mode)

绝对装入方式只能将目标模块装入到内存中事先指定的位置，这只适用于单道程序环境。而在多道程序环境下，编译程序不可能预知经编译后所得到的目标模块应放在内存的何处。因此，对于用户程序编译所形成的若干个目标模块，它们的起始地址通常都是从 0 开始的，程序中的其它地址也都是相对于起始地址计算的。把在装入时对目标程序中指令和数据地址的修改过程称为重定位，又因为地址变换通常是在进程装入时一次完成的，以后不再改变，故称为静态重定位。

3. 动态运行时的装入方式(Dynamic Run-time Loading)

可重定位装入方式可将装入模块装入到内存中任何允许的位置，故可用于多道程序环境。但该方式并不允许程序运行时在内存中移动位置。实际上，由于反复的对换和掉页，程序会被存放于不同的位置，此时就需要动态装入方式。动态装入时，装入内存的所有地址都仍是逻辑地址（即外存中的程序地址）为使地址转换不影响指令的执行速度，这种方式需要一个重定位寄存器的支持。把逻辑地址转换物理地址推迟到程序真正要执行时才进行。

4.2.2 程序的链接

1. 静态链接

在程序运行之前，先将各目标模块以及它们所需的库函数链接成一个完整的装配模块，以后不再拆开。

静态链接时需要完成：对相对地址进行修改，变换外部调用符号

2. 装入时动态链接

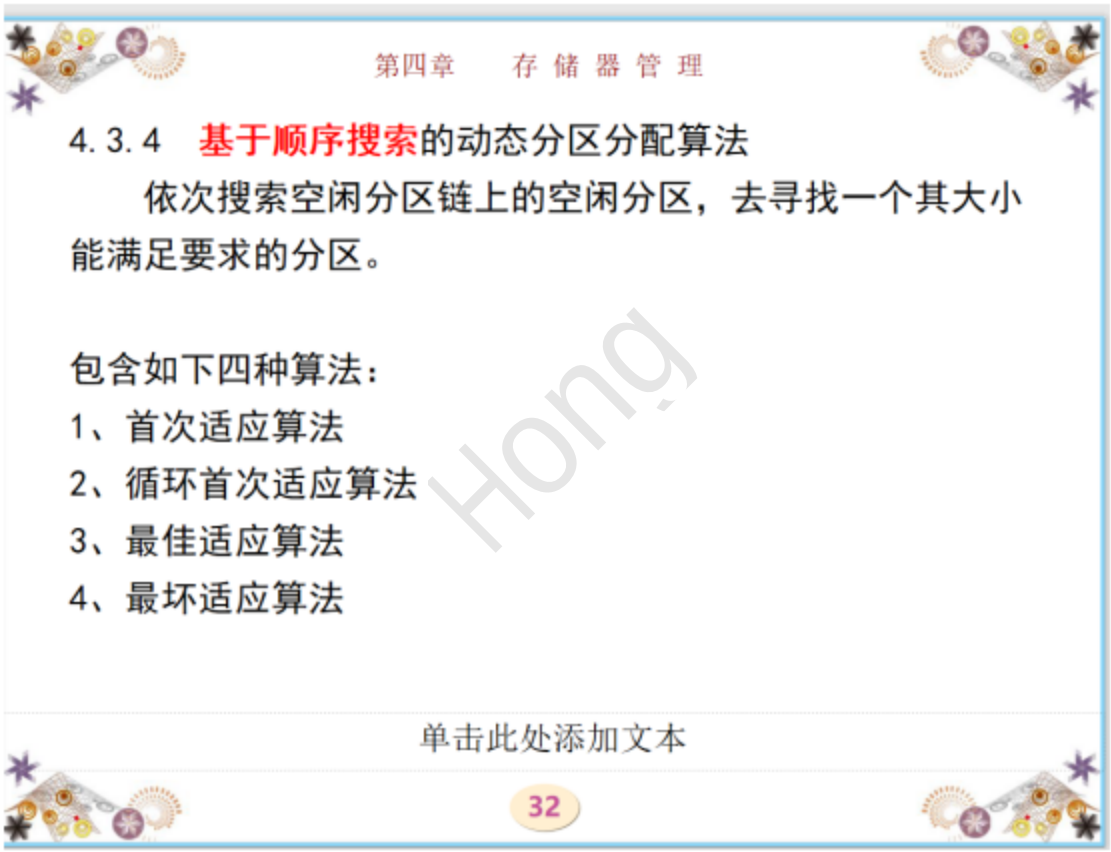
将用户源程序编译后所得到的一组目标模块，在装入内存时，采用边装入边链接的方式

优点：便于修改和更新、便于实现对目标模块的共享

3. 运行时动态链接

对某些模块的链接推迟到程序执行时才进行，即运行哪个，就链接哪个

4.3.4 基于顺序搜索的动态分区分配算法（选择）



4.3.5 基于索引搜索的动态分区分配算法（选择，伙伴系统）

4.3.5 基于索引搜索的动态分区分配算法

顺序搜索需要从头到尾扫描空闲分区表或者空闲分区链，但是一旦分区过多，查找效率会降低，故提出索引搜索。

包含三类算法

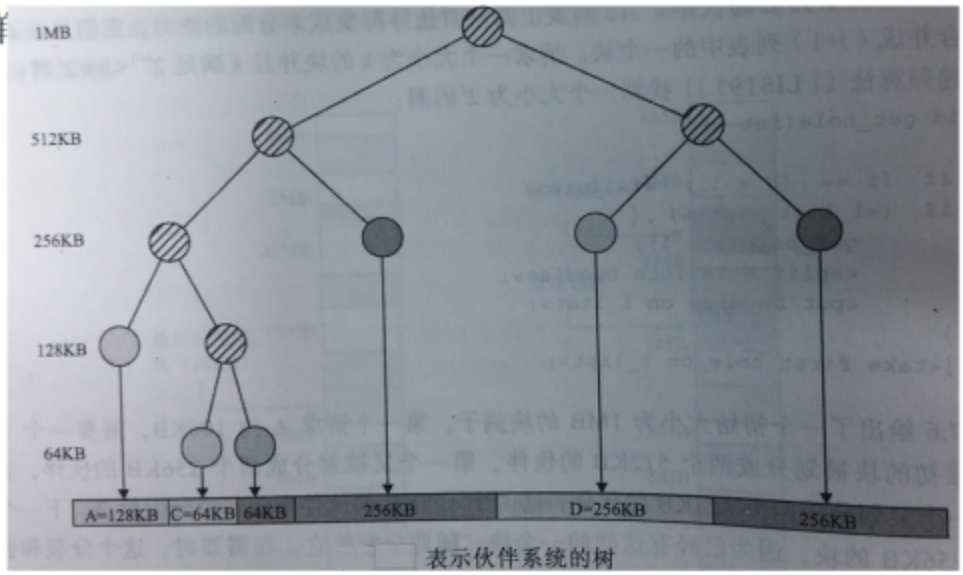
1、快速适应算法

2、伙伴系统

3、哈希算法

2. 伙伴系统(buddy system)

该算法规定，**无论已分配分区或空闲分区，其大小均为2的k次幂**(k为整数，l≤k≤m)。通常2^m是整个可分配内存的大小(也就是最大分区的大小)。假设系统的可利用空间容量为2^m个字，则系统开始运行时，整个内存区是一个大小为2^m的空闲分区。在系统运行过程中，由于不断地划分，将会形成若干个不连续的空闲分区，将这些空闲分区按分区的大小进行分类。对于具有相同大小的所有空闲分区，单独设立一个空闲分区双向链表，这样，不同大小的空闲分区形成了k个空闲分区链表。



伙伴系统示意图

伙伴系统简单而言，即把一块区域进行2的指数次进行划分或者归并，直到有一块最小的区域正好能够符合作业，则该区域的同样大小的区域（即伙伴）则被用于加入到对应的空闲分区链表中，等待下次的分配

☆4.5　分页存储管理方式

当前最多存放多少页：

机械字长 - 页内占用字长 =X，算出 X，则最多存放 2^X 页；比如 32-14=18，所以最多存放 2^18 页。

(1) 分页存储管理方式。
(2) 分段存储管理方式。
(3) 段页式存储管理方式。

☆4.6　分段存储管理方式

第五章

☆虚拟存储器的工作原理（简答题，讲清楚）

虚拟存储的基本原理是在程序装入时，不必将其全部读入到内存，而只需将当前需要执行的部分页或段读入到内存，就可让程序开始执行。在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），则由处理器通知操作系统将相应的页或段调入到内存，然后继续执行程序。

虚拟存储器的容量主要受到计算机可寻址的范围限制。

PPT 上的回答：

由局部性原理——程序在执行时将呈现出局部性规律，即在一较短的时间内，程序的执行仅局限于某个部分，相应地，它所访问的存储空间也局限于某个区域。可知，应用程序在运行之前没有必要将之全部装入内存，而仅须将那些当前要运行的少数页面或段先装入内存便可运行，其余部分暂留在盘上，随后通过对换、调页、调段等方式，把其他需要使用的部分换到内存中

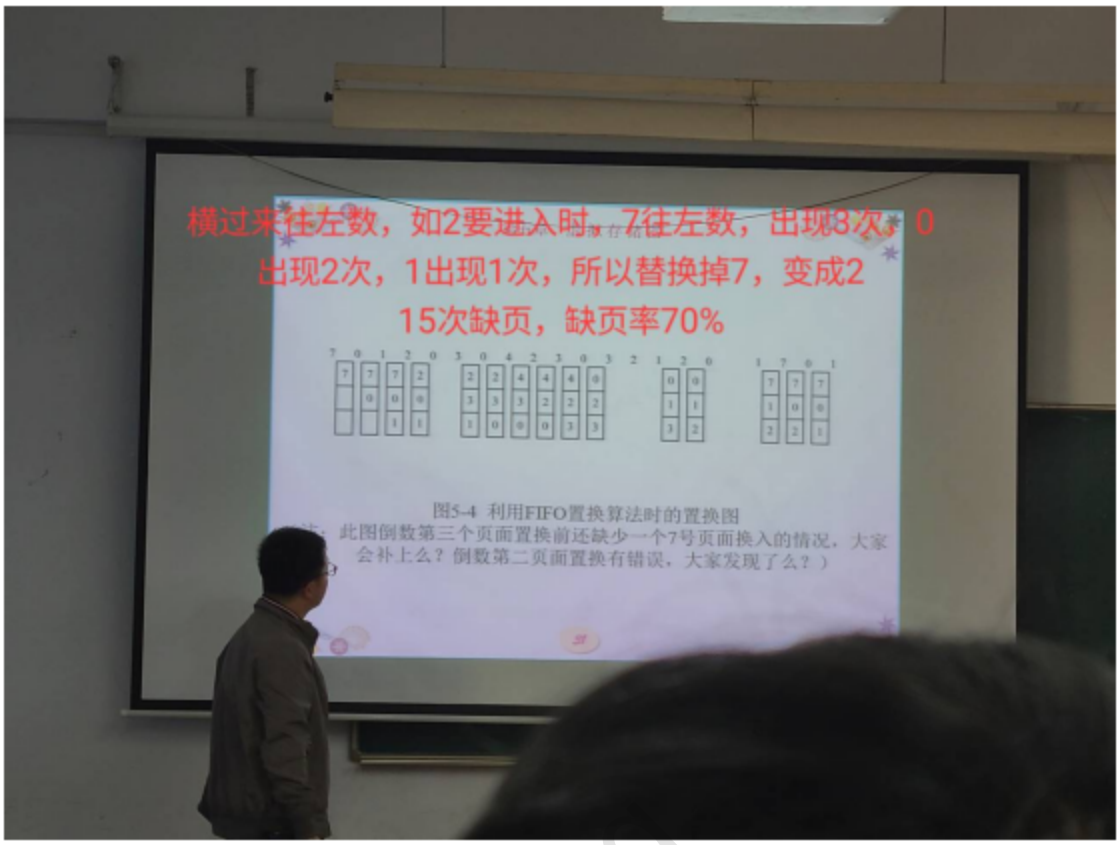
页面置换算法：

最佳页面置换算法（OPT）

柱子横着往右看，页面中已有的内存号中，谁最晚用到替换谁

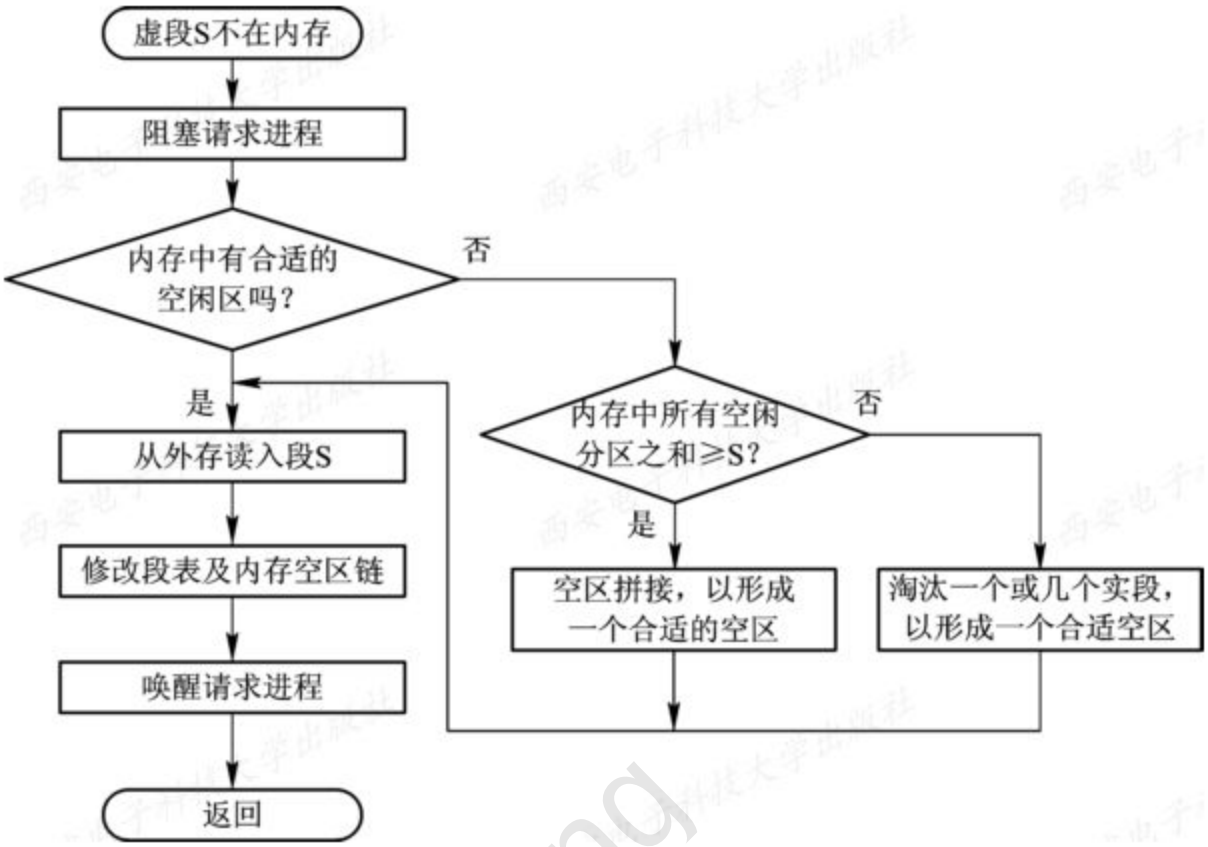
先进先出(FIFO)页面置换算法

柱子横着往左看，页面中已有的内存号中，谁最先出现（到目前为止连续出现最多）替换谁



最近最久未使用置换算法（LRU）
看前一个柱子+序列往左看，页面中已有的内存号中，谁最早未使用（序列越往左的），替换掉谁

☆请求分段系统中的中断处理过程



第六章

☆I/O 系统的基本功能

1. 隐藏物理设备的细节

I/O 设备的类型非常多，且彼此间在多方面都有差异，诸如它们接收和产生数据的速度，传输方向、粒度、数据的表示形式及可靠性等方面。

2. 与设备的无关性

隐藏物理设备的细节，在早期的 OS 中就已实现，它可方便用户对设备的使用。与设备的无关性是在较晚时才实现的，这是在隐藏物理设备细节的基础上实现的。

3. 提高处理机和 I/O 设备的利用率

在一般的系统中，许多 I/O 设备间是相互独立的，能够并行操作，在处理机与设备之间也能并行操作。因此，I/O 系统的第三个功能是要尽可能地让处理机和 I/O 设备并行操作，以提高它们的利用率。为此，一方面要求处理机能快速响应用户的 I/O 请求，使 I/O 设备尽快地运行起来；另一方面也应尽量减少在每个 I/O 设备运行时处理机的干预时间。

4. 对 I/O 设备进行控制

对 I/O 设备进行控制是驱动程序的功能。目前对 I/O 设备有四种控制方式：① 采用轮询的可编程 I/O 方式；② 采用中断的可编程 I/O 方式；③ 直接存储器访问方式；④ I/O 通道方式。

5. 确保对设备的正确共享

从设备的共享属性上，可将系统中的设备分为如下两类：

(1) 独占设备，进程应互斥地访问这类设备，即系统一旦把这类设备分配给

了某进程后，便由该进程独占，直至用完释放。典型的独占设备有打印机、磁带机等。系统在对独占设备进行分配时，还应考虑到分配的安全性。

(2) 共享设备，是指在一段时间内允许多个进程同时访问的设备。典型的共享设备是磁盘，当有多个进程需对磁盘执行读、写操作时，可以交叉进行，不会影响到读、写的正确性。

6. 错误处理

大多数的设备都包括了较多的机械和电气部分，运行时容易出现错误和故障。从处理的角度，可将错误分为临时性错误和持久性错误。对于临时性错误，可通过重试操作来纠正，只有在发生了持久性错误时，才需要向上层报告。

I/O 设备的类型

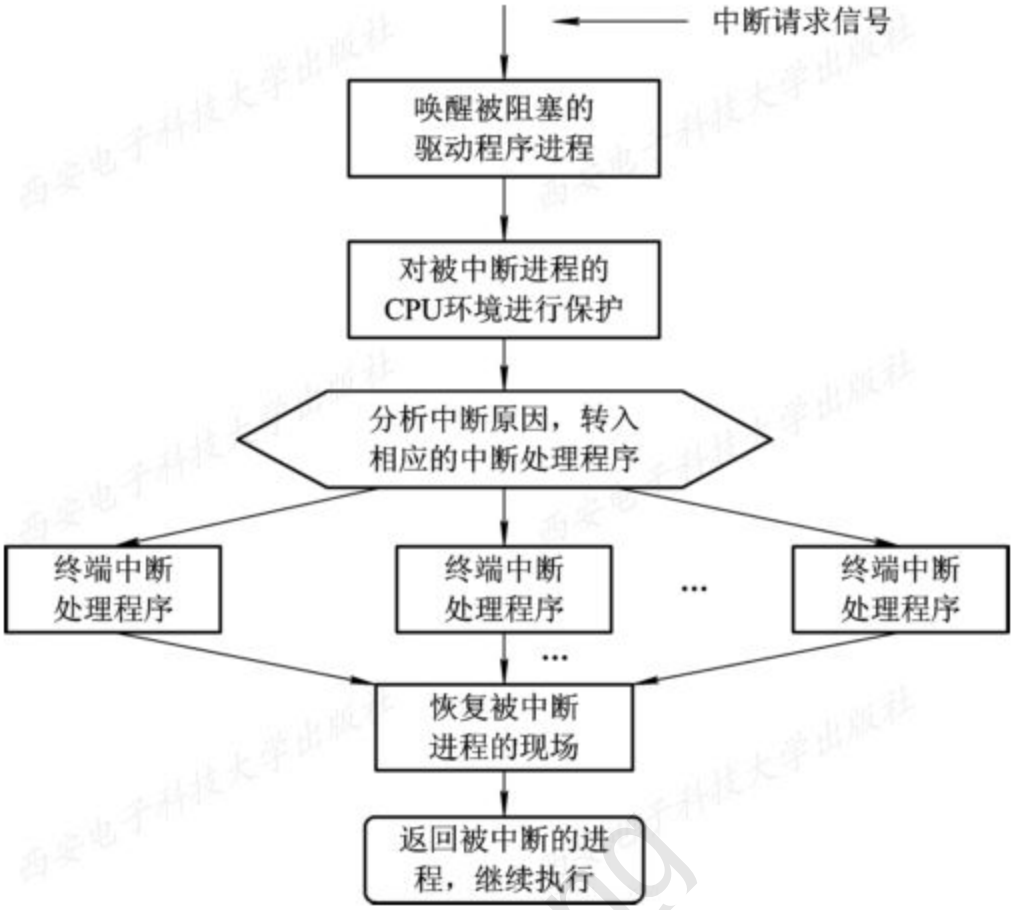
1) 按使用特性分类

存储设备：外存、辅存
I/O 设备：
输入设备：键盘、鼠标、扫描仪、视频摄像
输出设备：打印机、绘图仪
交互式设备：显示器

2) 按传输速率分类

低速设备：键盘、鼠标器
中速设备：行式打印机、激光打印机
高速设备：磁带机、磁盘集、光盘机

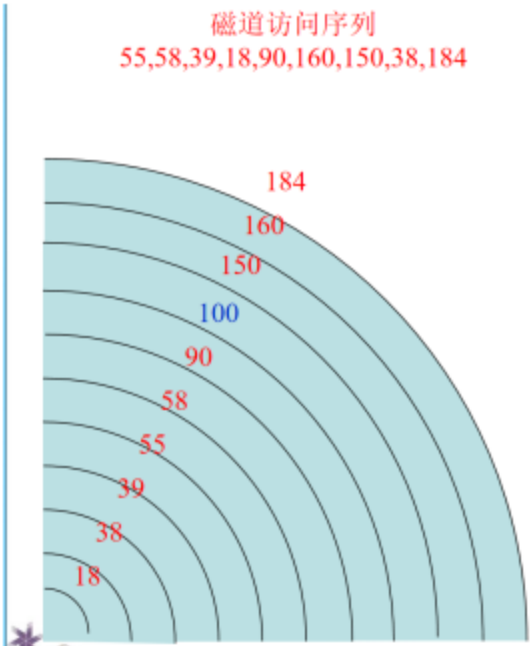
中断处理流程



☆磁盘调度算法

先来先服务(FCFS)

根据进程请求访问磁盘的先后次序进行调度。

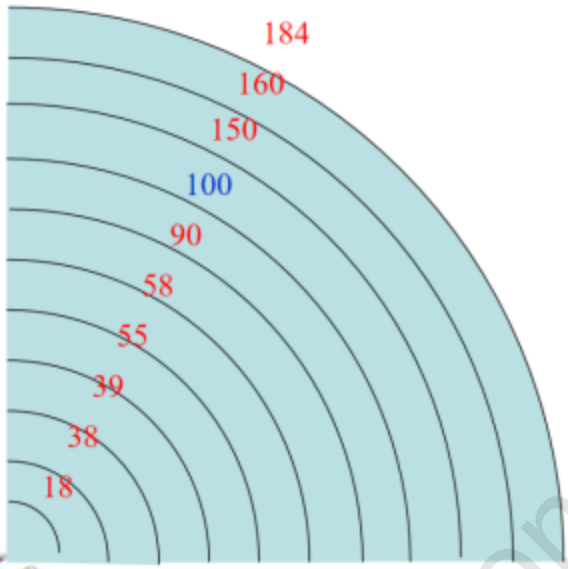


(从100号磁道开始)	
被访问的 下一个磁道号	移动距离 (磁道数)
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度：55.3	

图6-30 FCFS调度算法

最短寻道时间优先(SSTF)
访问的磁道与当前磁头所在的磁道距离最近

磁道访问序列
55,58,39,18,90,160,150,38,184



(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	

图6-31 SSTF调度算法

SCAN 调度算法
从当前磁道号开始，先访问大/小的，在访问小/大的(类似从当前磁道号向两侧扩散)

(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	

CSCAN 调度算法

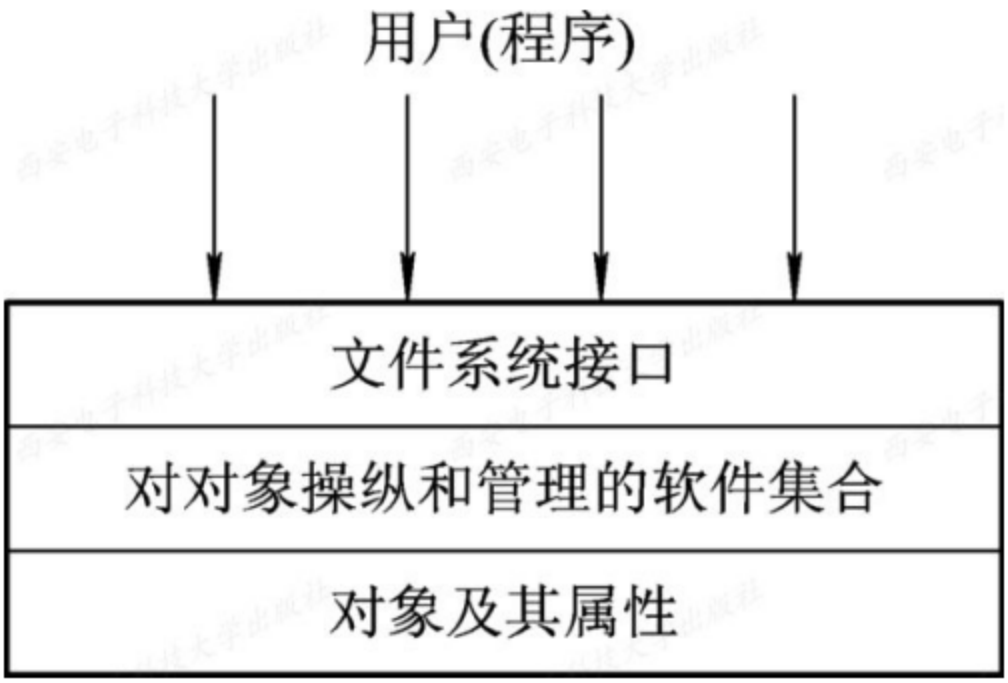
从当前磁道号开始，先访问大/小的，再从另一端开始访问（相当于都是朝着一个方向访问）

(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：35.8	

第七章

文件系统层次结构，文件系统模型

文件系统的模型可分为三个层次：最底层是对象及其属性，中间层是对对象进行操纵和管理的软件集合，最高层是文件系统提供给用户的接口。



文件的逻辑结构

1) 无结构文件

2) 有结构文件

有结构文件再分为：

(1) 定长记录（文件的每一条记录长度一致，其含有的数据项的数目、顺序也一致）。

(2) 变长记录（文件的每一条记录长度不同，其含有的数据项的数目、顺序也不同）。

其中，如果根据文件的组织方式，可把有结构文件分为三类：

(1) 顺序文件（一系列记录按顺序排列形成文件）。

(2) 索引文件（为记录建立一张索引表，各个记录设置一个表项，再用索引表来对应）。

(3) 索引顺序文件（为每个文件建立一张索引表，再把记录分成多组，每组设定一个表项存于索引表）。

第八章

文件的物理结构

文件的物理结构直接与外存的组织方式有关。对于不同的外存组织方式，将形成不同的文件物理结构。目前常用的外存组织方式有：

(1) 连续组织方式。

(2) 链接组织方式。

(3) 索引组织方式。

☆位示图

位示图是利用二进制的一位来表示磁盘中一个盘块的使用情况。当其值为“0”时，表示对应的盘块空闲；为“1”时，表示已分配。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0	0	0	1	1	1	0	0	1	0	0	1	1	0
2	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1
3	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0
4																
…																
16																

注意：该图中的行和列的编号都是从 1 开始的，盘块号也是从 1 开始编号的。（此处字长 n 为 16）

盘块的分配

根据位示图进行盘块分配时，可分三步进行：

(1) 顺序扫描位示图，从中找出一个或一组其值为“0”的二进制位(“0”表示空闲时)。

(2) 将所找到的一个或一组二进制位转换成与之相应的盘块号。假定找到的其值为“0”的二进制位位于位示图的第 i 行、第 j 列，则其相应的盘块号应按下式计算：

b = n(i - 1) + j（计算出 i 行 j 列是几号盘块，这个公式要自己推）

式中，n 代表每行的位数。

(3) 修改位示图，令 map[i, j] = 1。

盘块的回收

盘块的回收分两步：

(1) 将回收盘块的盘块号转换成位示图中的行号和列号。转换公式为：

i = (b - 1)DIV n + 1 （DIV 取整）

j = (b - 1)MOD n + 1 （MOD 取余）

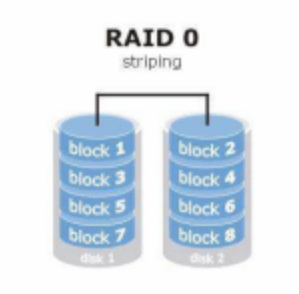
(2) 修改位示图。令 map[i, j] = 0。

riad 分类

RAID分为 8个级别，分别如下：

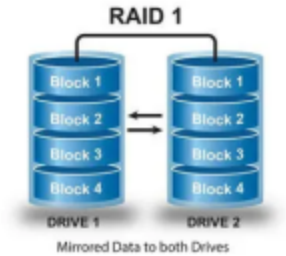
1、RAID 0

RAID 0 是组建磁盘阵列中最简单的一种形式，只需要 2 块以上的硬盘即可，成本低，可以提高整个磁盘的性能和吞吐量。RAID 0 没有提供冗余或错误修复能力，但实现成本是最低的。



2、RAID 1

RAID 1 主要是通过二次读写实现磁盘镜像，所以磁盘控制器的负载也相当大，尤其是在需要频繁写入数据的环境中。为了避免出现性能瓶颈，使用多个磁盘控制器就显得很有必要。



3、RAID 0+1

RAID 0+1 是把 RAID0 和 RAID1 技术结合起来，数据除分布在多个盘上外，每个盘都有其物理镜像盘，提供全冗余能力，允许一个以下磁盘故障，而不影响数据可用性，并具有快速读/写能力。

4、RAID: LSI MegaRAID、Nytro 和 Syncro

MegaRAID、Nytro 和 Syncro 都是 LSI 针对 RAID 而推出的解决方案，并且一直在创造更新。LSI 通过 MegaRAID 提供基本的可靠性保障;通过 Nytro 实现加速;通过 Syncro 突破容量瓶颈，让价格低廉的存储解决方案可以大规模扩展，并且进一步提高可靠性。

5、RAID2：带海明码校验

RAID 2 同 RAID 3 类似， 两者都是将数据条块化分布于不同的硬盘上， 条块单位为位或字节。然而 RAID 2 使用一定的编码技术来提供错误检查及恢复。

6、RAID3：带奇偶校验码的并行传送

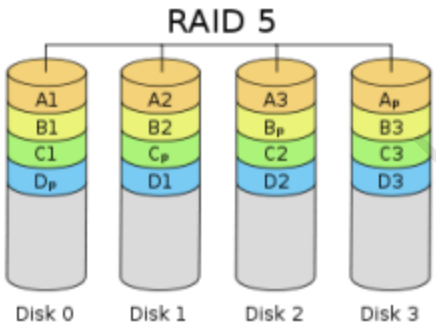
RAID3 访问数据时一次处理一个带区，这样可以提高读取和写入速度。校验码在写入数据时产生并保存在另一个磁盘上。

7、RAID4：带奇偶校验码的独立磁盘结构

RAID4 和 RAID3 很象，不同的是，它对数据的访问是按数据块进行的，也就是按磁盘进行的，每次是一个盘。

8、RAID5：分布式奇偶校验的独立磁盘结构

RAID5 的奇偶校验码存在于所有磁盘上，其中的 p0 代表第 0 带区的奇偶校验值，其它的意思也相同。RAID5 的读出效率很高，写入效率一般，块式的集体访问效率不错。



9、RAID6：带有两种分布存储的奇偶校验码的独立磁盘结构

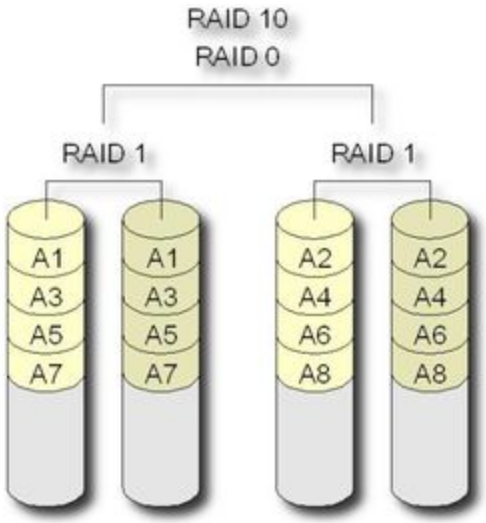
RAID6 是对 RAID5 的扩展，主要是用于要求数据绝对不能出错的场合。

10、RAID7：优化的高速数据传送磁盘结构

RAID7 所有的 I/O 传送均是同步进行的，可以分别控制，这样提高了系统的并行性，提高系统访问数据的速度；每个磁盘都带有高速缓冲存储器，实时操作系统可以使用任何实时操作芯片，达到不同实时系统的需要。

11、RAID10：高可靠性与高效磁盘结构

RAID10 是一个带区结构加一个镜象结构，新结构的价格高，可扩充性不好。主要用于数据容量不大，但要求速度和差错控制的数据库中。



12、RAID53：高效数据传送磁盘结构

RAID53 就是 RAID3 和带区结构的统一，因此它速度比较快，也有容错功能。但价格十分高，不易于实现。