（一）概论

一、操作系统的发展：

批处理系统，分时系统，实时系统

二、操作系统的特点：

并发：并发是指允许两个或者两者以上的事件在同一时间间隔内同时执行

共享：是指系统中的资源允许内存中的并发程序共同使用

虚拟：通过某种技术让物理内存映射到若干个逻辑上的对应物

异步：即不确定性，在多道程序设计中，各个程序之间存在着直接或者间接的联系，程序的推进速度受它的运行环境的影响，这时同一程序和数据的多次运行可能得到不同的结果；程序的运行时间、运行顺序也具有不确定性；外部输入的请求、运行故障发生的时间难以预测，这些都是不确定性的体现

三、四个重要概念：

1、操作系统：操作系统是一个管理计算机系统资源，控制程序运行的系统软件，它为用户提供了一个方便、安全、可靠的工作环境和界面

2、多道程序设计：多道程序设计是指允许多个作业（程序）同时进入计算机系统的内存并启动交替计算的方法

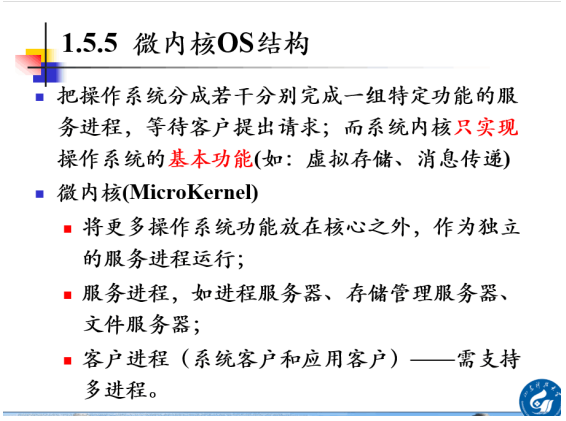
3、系统调用：系统调用是一种中介角色，把用户和硬件隔离开来，应用程序只有通过系统调用才能请求服务程序并使用系统资源

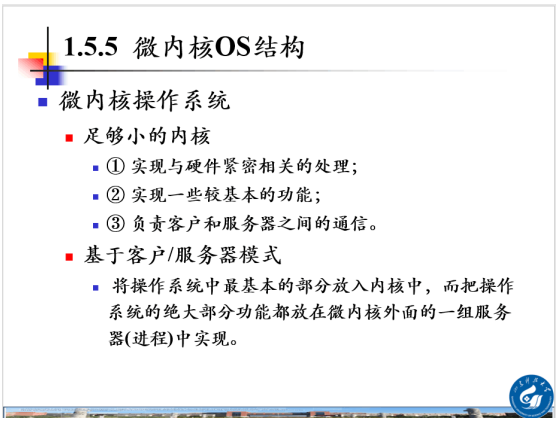
系统调用是操作系统提供给软件开发人员的唯一接口，开发人员可利用它使用系统功能。用户在程序中调用操作系统提供的子功能称为系统调用。

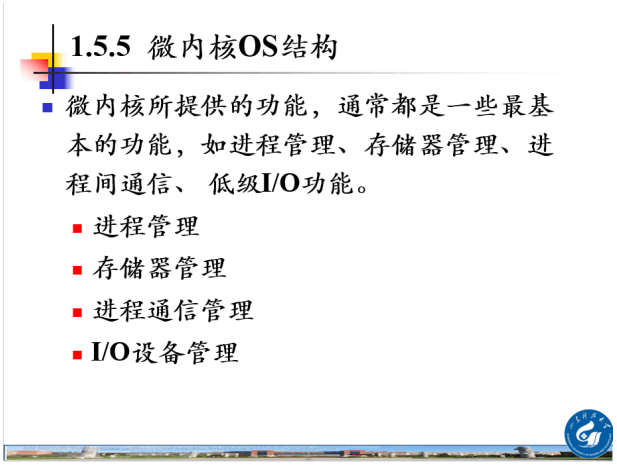
4、微内核技术：微内核就是很小的内核。它指的是必须在特权态下运行的指令尽可能少的操作系统的内核。其设计目标是使操作系统的内核尽可能小以提高操作可靠性（代码越少越容易做到可靠）和灵活性（运行在用户态的操作系统功能可以更加灵活）

微内核(Microkernel)就是很小的内核。它指的是必须在特权态下运行的指令尽可能少的操作系统的内核。微内核结构由美国卡内基梅隆大学所提出，其设计目标是使操作系统的内核尽可能小以提高操作可靠性(代码越少越容易做到可靠)和灵活性(运行在用户态的操作系统功能可以更加灵活)。在微内核组织下，所有操作系统服务都放在核外用户态完成。微内核仅提供以下四种内核态操作系统服务：进程间通信、某些存储管理、进程调度和派遣、低级I/O。与微内核对应的操作系统内核组织形式是强内核(Monolithic Kernel)。此外，现代的商业操作系统采取的内核组织方式是介于强内核和微内核之间的一种中间结构。

微内核的定义







（二）进程管理

进程的典型定义：

- 进程是程序的一次执行。
- 进程是程序及其数据在处理机上顺序执行时发生的活动。
- 进程是一个程序在数据集合上运行的过程，它是系统进行资源分配和调度的独立单位。

- 在引入程序实体的传统OS中又可定义为：进程是进程实体的运行过程，是系统进行资源分配和调度的独立单位。
- 每个进程都有它专属的地址空间，其他的进程不能直接访问。

一、进程和程序的联系：

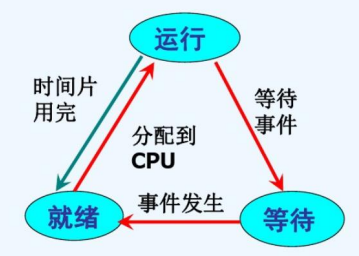
- 进程是动态的，程序是静态的：程序是有序代码的集合；进程是程序的执行。通常进程不可在计算机之间迁移；而程序通常对应着文件、静态和可以复制。
- 进程是暂时的，程序是永久的：进程是一个状态变化的过程，程序可长久保存。
- 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块，进程实体能和其它进程实体并发执行；而程序(没有建立PCB)是不能并发执行的。
- 进程与程序的对应关系：通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可包括多个程序

二、进程最基本的状态有哪些？哪些事件可能引起不同状态间的转换？

运行态：进程占有处理器正在运行的状态。

就绪态：进程具备运行条件，等待系统分配处理器以便运行的状态。

等待态：又称**阻塞**态或睡眠态，指进程不具备运行条件，正在等待某个事件完成的状态。



进程状态转换图

而负责进程在 新建，就绪，运行，等待（阻塞）这几个状态转变的一个程序就是操作系统中的 scheduler （调度程序）。

短程调度（进程调度），它是距离硬件最近的一级调度，是按照某种算法从就绪队列中选择一个（或多个）进程，使其获得；

长程调度（作业调度），它是批处理系统中使用的一种调度。其主要任务是按照某种算法从外存的后备队列上选择一个或者多个作业调入内存，并为其创建进程、分配必要的资源，然后再将其创建的进程控制块插入就绪队列中；

中程调度 负责从内存中移除进程（等待一段时间，内存里面不爆满的时候再调入），降低**程序多道度**（内存中程序数量）。

几个概念：

I / O型进程：I / O 操作多于计算操作

CPU型进程：计算操作多于 I / O 操作

上下文切换：有时也称做进程切换或任务切换，是指 CPU 从一个进程或线程切换到另一个进程或线程。

三、什么是PCB？它包含哪些基本信息？

每个进程有且仅有一个进程控制块，或称进程描述符，它是进程存在的唯一标识，是操作系统用来记录和刻画进程状态和环境信息的数据结构，是进程动态特征的集合，也是操作系统掌握进程的唯一资料结构和所处的状态的主要依据。

a）标识信息：标识信息用于唯一地标识一个进程，分为用户使用的外部标识和系统使用的内部标识符。

b）现场信息：现场信息用于保留进程在运行时存放在处理器现场中的各种信息。

c）控制信息：控制信息用于管理和进程调度。

四、进程通信

由于进程的互斥与同步，需要在进程间交换一定的信息，被归为进程通信（低级进程通信，例如：只能传输少量数据的信号量，管程等）。

进程的互斥是指当多个进程共享数据块或其他排它性使用资源时，不能在同一时刻执行，但执行的次序可以是任意的，这种制约关系称为互斥。

进程的同步是指进程间完成一项任务，而需要互相等待和互相交换信息的相互制约关系称为同步。

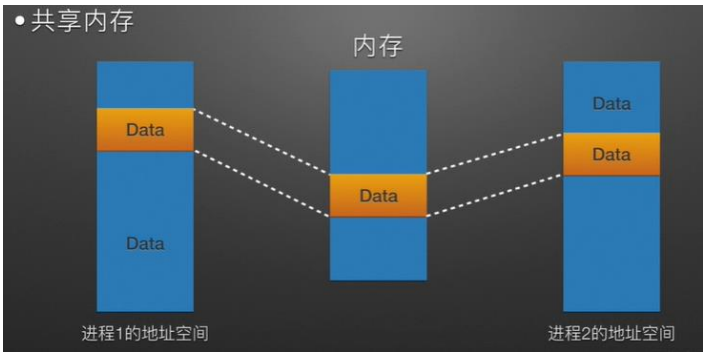
进程的互斥与同步

1、高级通信机制分为四大类：

① 共享存储器系统：

1）在内存中找一片空间，专门为两个进程共享，也就是两个进程都可以操作的一片存储空间。实现方法就是让同一块物理内存空间映射到给两个进程的地址空间中去。

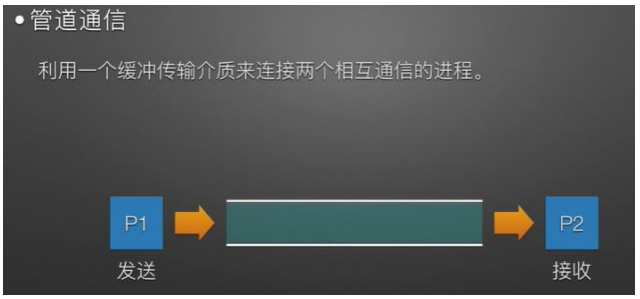
2）共享内存区域需要解决互斥问题，解决方法和读者写者问题是一样的，因为共享内存中不能多个进程同时进行写操作，但是可以同时进行读操作。



② 管道通信系统：

2）FIFO

3）管道提供简单的协调能力，只需要判断双方进程是否存在和判断管道内是否有数据。因读和写在不同的位置进行，所以无需复杂的互斥管理。



③ 消息传递系统：

通过两个原子操作实现

send(p1，message)、receive(p2，message)

或者邮箱的方式：

send(mailbox，message)、receive(mailbox，message)

④ 客户机-服务器系统：

五、进程调度时间计算问题：

周转时间 = 完成时间 - 到达时间

平均周转时间 = 周转时间总和 / 总的任务个数

等待时间 = 周转时间 - 运行时间

带权周转时间 = 周转时间 / 运行时间

优先权 = （等待时间 + 运行时间）/ 运行时间

六、线程

用户线程： 操作系统看不到的线程叫做用户（由用户线程库管理），操作系统看不到TCB；

内核线程：操作系统管理的线程叫内核线程，操作系统之间管理TCB

（补充：关于内核 https://blog.csdn.net/AlbenXie/article/details/80436095）

用户线程与内核线程的对应关系：

多对一（n个 用户线程对应一个内核线程）、一对一（...）、多对多（...）

七、死锁

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象

出现的四个必要条件（不是充要条件）：

1、互斥：在一个时间只能有一个进程使用资源；

2、持有并等待：进程保持至少一个资源正在等待其他进程持有的资源；

3、不可抢占：一个资源只能被进程资源释放，进程已经完成了它的任务之后；

4、循环等待：存在等待进程集合{P0,P1...Pn}；P0正在等待P1占有的资源...Pn正在等待P0占有的资源。

死锁的解决：

1、确保系统永远不会处于死锁状态

2、运行系统进入死锁状态，然后恢复

3、忽略这个问题，假装系统中从来没有发生死锁，用于大多数操作系统，如：unix

（1）死锁预防（限制申请方式）：

1、互斥：共享资源

2、占有并等待：要么hold所有资源，要么不持有任何资源

3、不可抢夺：

4、循环等待：对所有资源进行排序，并要求每个进程按照资源的顺序进行申请

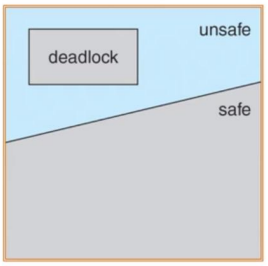
（2）死锁避免：确保系统永远不会进入不安全状态

当进程运行过程申请资源时，判断会不会出现死锁。

1、要求每个进程声明它可能需要的每个类型资源的最大数目

2、资源的分配状态时通过限定提供与分配的资源数量，和进程的最大需求

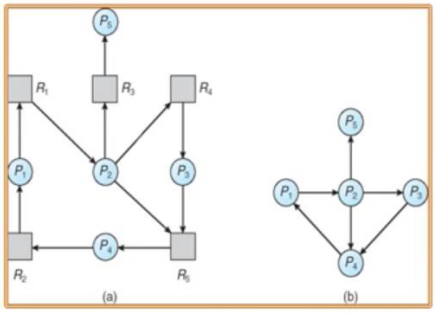
3、死锁避免算法动态检查资源分配状态，以确保永远不会有一个环形等待状态（不安全状态）（不安全状态包括死锁状态，即检测会不会变成不安全状态，而不是检测是否变成死锁状态）



不安全状态的地位

（3）死锁检测算法：

1、将资源分配图（图a）简化为进程等待图（图b），将所有资源的结点去掉，只留下进程的结点，然后判断只有存在环：



Resource Allocation Graph wait-for Graph

2、直接在资源分配图中判断：Available：代表每种类型可用资源的数量

Allocation：定义了当前分配给各个进程每种类型资源的数量

死锁检测算法

1. *Work* 和 *Finish* 分别是长度为m与n的向量，初始化：
   (a) *Work* = *Available* //work为当前空闲资源量
   (b) For $i = 1, 2, \cdots, n$, if $Allocation_i > 0$, then *Finish*[i] = false;otherwise, *Finish*[i] = *true*. //Finish为线程是否可结束
2. 找出这样的索引i：
   (a) *Finish*[*i*] == *false*
   (b) $Request_i =< Work$
   如果没有找到，转到4. //线程没有结束的线程，且此线程将需要的资源量小于当前空闲资源量
3. $Work = Work + Allocation_i$
   Finish[*i*] = *true*
   转到2. //把找到的线程拥有的资源释放回当前空闲资源中
4. If *Finish*[*i*] == false, for some *i*, $1 \leq i \leq n$, 系统处于死锁状态。此外，if *Finish*[*i*] == *false*, P_i 死锁。 //如果有Finish[i]等于false，这表示系统处于死锁状态

算法需要$O(m \times n^2)$ 操作检测是否系统处于死锁状态

定期执行该算法

（4）死锁恢复：

八、进程同步

竞争条件：像这样的情况，其中几个进程访问和操作相同的数据并发和执行结果的特定顺序发生取决于访问，被称为竞争条件

临界资源：一次仅允许一个进程使用的资源

临界区：临界区是指进程中的一段需要访问共享资源并且当另一个进程处于相应代码区域时便不会被执行的代码区域

互斥：当一个进程处于临界并访问共享资源时，没有其他进程会处于临界区并且访问任何相同的共享资源

（同一时间临界区最多存在一个线程）

（三）内存管理

一、页面置换算法：

1、先进先出 2、最优置换 3、clock算法

4、二次机会法（访问位A，脏位W）：

a） 都是0的优先级最高，其次是访问位为0脏位为1

再者是访问位是1脏位是0，最后是都是1的

b） 扫描过程中将只有一个是1的，将其置零；

对于两个都是1的，将访问位置零，脏位不变，同时指针向下走。

因此，01和10的有一次机会被替换出去，11的有两次机会，因此称为二次机会法

二、抖动问题（thrashing）

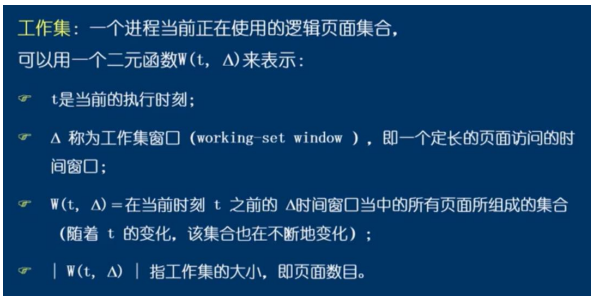
如果分配给一个进程的物理页面太少，不能包含整个的工作集，即使常驻集包含于工作集，那么进程将会造成更多的缺页中断，需要频繁的地在内存和外存之间替换页面从而使进程的运行速度变得很慢，我们把这种现象成为”抖动”

产生抖动的原因：随着驻留内存的进程数目增加，分配给每个进程的物理页数不断减少，缺页率不断上升，所以OS要选择一个适当的进程数目h和进程需要的帧数，以便在并发水平和缺页率之间达到一个平衡。

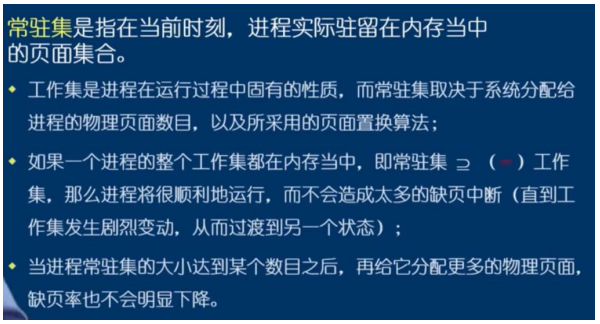
三、工作集替换算法（work set）

1、工作集：一个进程当前正在使用（一个时间段）的逻辑页面集合，可以用一个二元函数W(t, A)

2、工作集和常驻集模型：

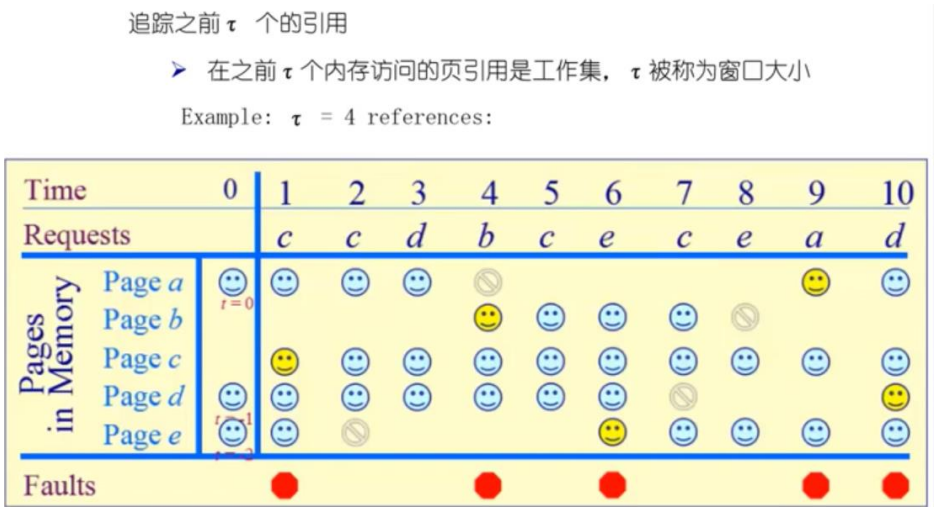


工作集模型



常驻集

3、eg：



工作集算法举例

四、缺页率算法（Page fault frequency）：动态调整 W(t, A) 中 A 的大小

1、缺页率 = 缺页次数 / 内存访问时间（比率）

2、影响缺页率的因素：

（1）页面替换算法

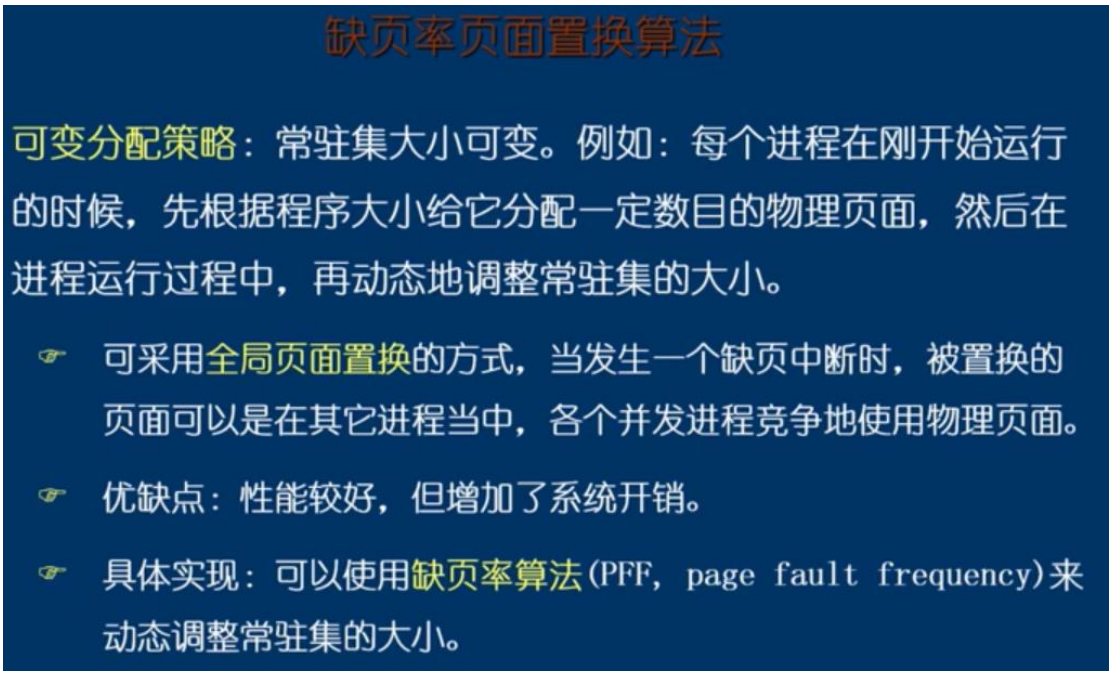
（2）分配给页面的物理页面数目

（3）页面本身的大小

（4）程序的编写方法（局部性是否良好）

3、算法：

每次发生缺页时记录时间，如果发生缺页的两次时间间隔大于某个阈值，则减少工作集，反之，则增大工作集。



若运行程序的缺页率过高，则通过增加工作集来分配更多的物理页面，若运行程序的缺页率过低，则通过减少工作集来减少它的物理页面数量，力图使运行的程序的缺页率保持在一个合理的范围内。

4、eg：

Example: window size = 2

- 如果 $t_{current} - t_{last} > 2$，从工作集中移除没有在[$t_{last}$，$t_{current}$]被引用的页面
- 如果 $t_{current} - t_{last} < 2$，仅增加缺失页到工作集中

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	☺	☺	☺	☺	⊘					☺	☺
Page b					☺	☺	☺	☺	☺	⊘	
Page c		☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
Page d	☺	☺	☺	☺	☺	☺	☺	☺	☺	⊘	☺
Page e	☺	☺	☺	☺	⊘		☺	☺	☺	☺	☺
Faults		●			●		●			●	●
$t_{cur} - t_{last}$		1			3		2			3	1

page fault frequency举例

易混知识点：

（四）存储管理

一、磁盘

1、磁盘的访问时间：

a、寻道时间：定位到期望的磁道所花费的时间

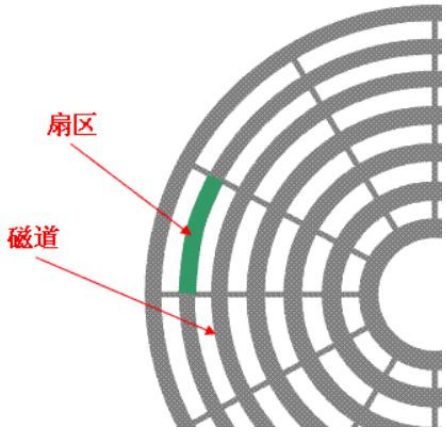
b、旋转延迟时间：从扇区的开始处到达目的处所花费的时间

平均旋转延迟时间 = 磁盘旋转一周时间的一半

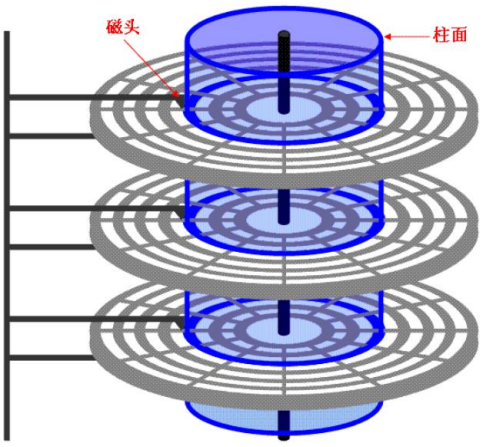
c、传输时间：指把数据从磁盘读出或向磁盘写入数据所经历的时间

若每次读/写的字节数为b，磁盘每秒钟的转速为r，一条磁道上的字节数为N

则 t = b / (r * N)



磁道和扇区



柱头和柱面

2、磁盘调度

1）FCFS

算法思想非常简单，就是不论初始磁头在什么位置，都是按照服务队列的先后顺序依次处理进程，可以类比队列的先进先出。优点

是进程处理起来非常简单，但缺点显而易见，就是平均寻道长度会很长。

2）SSTF

最短寻道时间算法，算法本质是贪心，已知磁头的初始位置，则最先被处理就是距离磁头位置最近的进程，处理完成后再处理距离当前磁道最近的进程，直到所有的进程被处理。该算法的优点是平均寻道长度会大大减少，缺点是距离初始磁头较远的服务长期得不到处理，产生“饥饿”现象。具体的思路是：通过循环寻找与初始磁头最近的进程，将进程处理，然后将该进程标记为-1，将初始磁头移动到该进程所在的磁道。然后依次类推，标记为-1的进程不再参与，知道所有的进程都被标记为-1，磁盘调度完成。

3）SCAN

磁头仅沿一个方向进行扫描，在扫描途中完成所有没有完成的请求，直到磁头到达磁盘在这个方向上的最后一个磁道或者这个方向上最后一个请求所在的磁道。利用数组存储进程和磁道编号，依据给定的初始磁头，先找到初始磁头在哪两个进程之间，然后向内扫描。当磁头扫描到磁盘最内层即磁道0且进程还没有全部被处理，磁头开始向外扫描，直到所有的进程都完成。

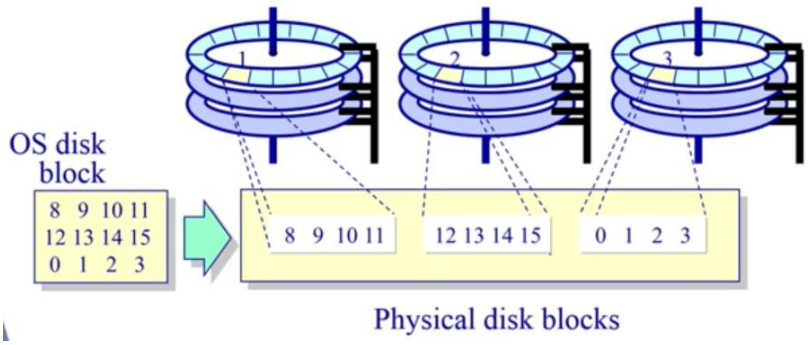
4）C-SCAN

在磁盘扫描算法的基础上改变磁头的扫描路径：扫描到最内层之后从最外层向内继续扫描，即扫描方向一致。该算法的思路与扫描算法基本一致，也使用二维数组进行进程编号和进程所在磁道号的存储，算法的不同之处在于当磁头扫描到磁盘的最内层时，磁头跳转到磁盘最外层重新向内扫描，这样就可以有效的避免将已经扫描过的磁道重新扫描一次，降低了平均寻到距离。

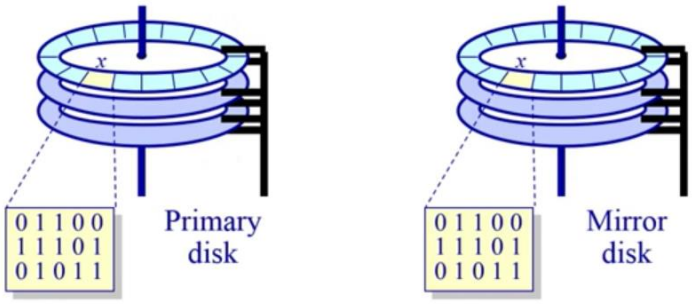
3、RAID （冗余磁盘阵列）

1、数据块分成多个子块，存储在独立的磁盘中：和内存交叉类似

通过更大的有效块大小来提供更大的磁盘带宽（提高吞吐率）



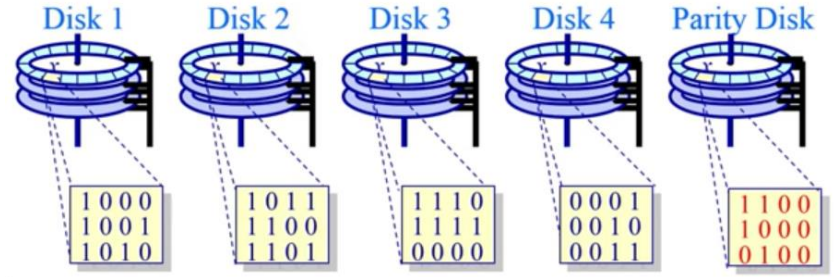
2、可靠性成倍增长，读取性能线性增加：向两个磁盘写入，从任何一个读取



镜像

3、数据块级磁盘带配有专用奇偶校验磁盘

允许从任何一个故障磁盘中恢复



二、文件系统接口

1、文件：

文件系统：一种用于持久性存储的系统抽象

文件：文件系统中一个单元的相关数据在操作系统中的抽象

文件属性：名称，类型，位置，大小，保护，创建者，创建时间，最近修改时间

文件头：

2、目录结构

文件以目录的方式组织起来，目录是一类特殊的文件

3、划分

4、访问方法：

顺序访问

随机访问

基于内容访问

三、文件系统的实现

1、FCB（文件控制块）

描述和控制文件信息的数据结构，包括文件的基本信息，使用信息和存取控制信息，它实现了文件名和文件物理地址的映射

2、目录实现

文件目录是一种数据结构，用于标识系统中的文件及其地址，供检索时使用。

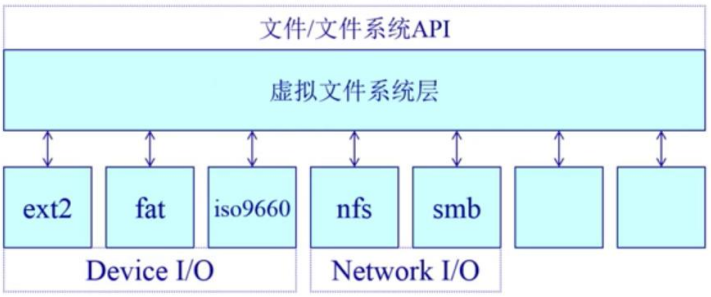
1）hash表

2）线性检索法

3、虚拟文件系统的实现

分层结构：　　上层：虚拟（逻辑）文件系统

底层：特定文件系统模块



通过虚拟内存的抽象之后形成统一的接口

目的：对所有不同文件系统的抽象

功能：　提供相同的文件h和文件系统接口

管理所有文件和文件系统关联的数据结构

高效查询例程，遍历文件系统

与特定文件系统模块的交互

4、文件分配

分配方式：

1）连续分配：

a、文件指定 起始地址 和 长度

b、优点：文件读取表现良好

高效的顺序和随机访问

缺点：无法灵活地扩展文件（适合只读方式）

2）链式分配：

a、数据块以数据链表方式存储，文件头包含了第一块和最后一块指针

b、优点：创建、增大、缩小很容易，没有碎片

缺点：不可能实现随机访问

可靠性问题：破坏一个链接后整个文件崩溃

3）索引分配：

a、为每个文件创建一个名为索引块的非数据数据块，文件头包含了索引数据块

b、优点：创建、增大、减少很容易，没有碎片，支持直接访问

缺点：当文件很小时，索引的开销比例大

当文件很大时，一个索引块可能无法囊括所有的索引

（多级索引块，来实现对大文件的支持）

5、空闲空间管理

1、bit vector（bit map）位向量（位图）

指向空闲列表的指针

必须保存到磁盘上

在内存磁盘拷贝可能有所不同

2、链式列表

快速查找空闲块

3、分组列表