第八章　磁盘存储器的管理

- 组成：驱动部分+存储介质
- 外存的特点
  - 容量大
  - 断电后仍可保存信息
  - 速度较慢
  - 成本较低
  - 种类很多
  - 外存空间组织与地址存取方式复杂
  - I/O过程方式复杂

- 用户对外存的要求
  - 使用：读写外存数据
  - 要求：方便、效率、安全

- 读写不涉及硬件细节，使用逻辑地址和逻辑操作
- 存取速度尽可能快，容量大且空间利用率高
- 信息安全可靠，防止硬件故障和他人侵权
- 方便共享，动态扩缩，携带拆卸
- 了解存储情况和使用情况
- 以尽可能小的代价完成上述要求

文件结构形式：

逻辑结构(File Logical Structure)

物理结构(存储结构，文件在外存上的存储组织形式)

- 文件的物理结构

  逻辑文件在存储设备（外存）上的存储组织形式，它与存储介质的存储特性有关

- 存储块（物理盘块）

  文件存储介质格式化后就分成许多大小相等的单位。每个物理块是一个磁盘的扇区，编号为物理块号

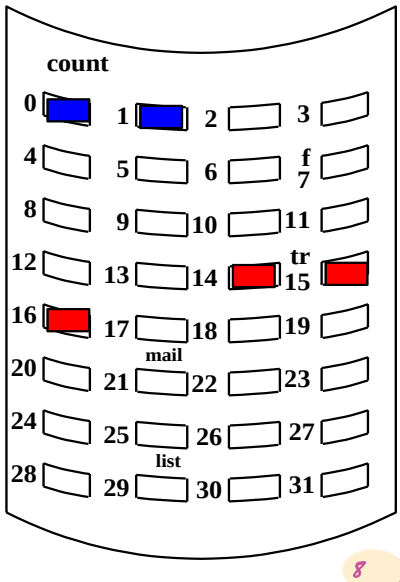
- 物理块是分配和传输信息的基本单位，其与外存设备有关，如扇区、簇，但与逻辑记录大小无关

- 文件在逻辑上可看作是连续的
- 文件在物理设备上存放方式
  - 连续结构（顺序结构）
  - 链接结构（串联结构）
  - 索引结构
  - HASH文件等

连续组织方式

- *连续分配(Continuous Allocation)*
  - 为每一个文件分配一组相邻接的盘块
  - 一组盘块定义了磁盘上的一段线性地址
- *顺序文件结构*
  - 逻辑文件中的记录顺序地存储到邻接的各物理盘块中
  - 物理文件称为*顺序文件*

连续组织方式



目 录

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

连续组织方式

- 优点
  - 结构简单，容易实现
  - 支持顺序存取和随机存取
  - 顺序存取速度快
  - 所需的磁盘寻道次数和寻道时间最少
- 缺点
  - 要求有连续的存储空间，不利于动态扩充
  - 容易形成碎片，空间利用不充分
  - 必须事先知道文件的长度，用户不方便

9

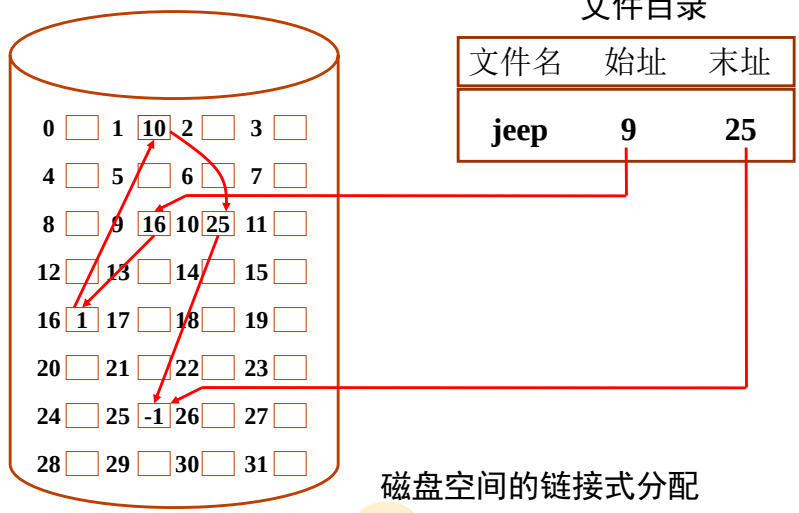
链接组织方式

- 链接分配（Chained Allocation）
  - 通过盘块上的链接指针将同属于一个文件的多个离散的盘块链接成一个链表
  - 这样形成的物理文件称为链接文件
- 不要求连续存放
- 记录式文件一块中可包含一个逻辑记录或多个逻辑记录，也可以若干物理块包含一个逻辑记录
- 链接方式
  - 隐式链接
  - 显式链接

10

链接组织方式

1. 隐式链接



磁盘空间的链接式分配

链接组织方式

1. 隐式链接

- 物理块的最末一个字(或第一个字)作为链接字，它指出后继块的物理地址。链首指针存放在该文件目录中。文件的结尾块的指针为“∧”
- 优点
  - 离散存储，空间利用率高
  - 顺序存取效率高
- 缺点
  - 随机存取效率太低
  - 若要访问第i个物理块，必须读出前i-1个

链接组织方式

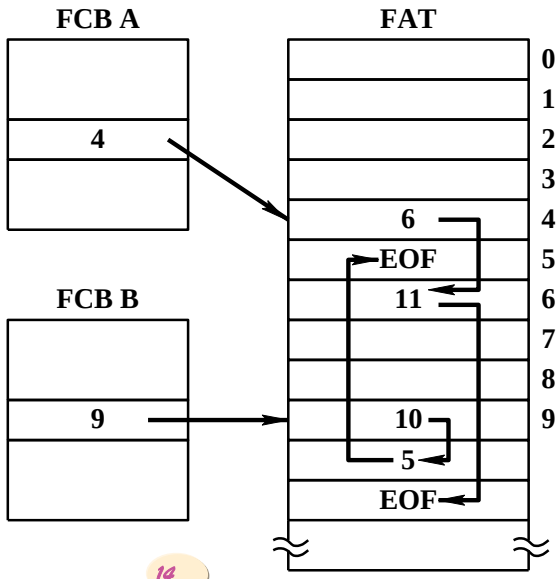
2. 显式链接

- 把用于链接文件物理块的指针存放在内存的一张链接表：文件分配表FAT(File Allocation Table)
- 整个磁盘仅设置一张文件分配表FAT
- 链接表中属于某一文件的第一个盘块号作为文件地址被填入相应文件的FCB的“物理地址”字段中。
- 优点：
  - 查找记录的过程是在内存中进行的
  - 显著地提高了检索速度
  - 减少了访问磁盘的次数

13

链接组织方式

2. 显式链接



14

链接组织方式

2. 显式链接

- 实例
  - 对于1.2M磁盘，每个物理块大小为1KB

    共有 ？ 个FAT表项
    - 1.2 K个FAT表项
  - 若每个表项占12位（1.5B）

    共需 ？ KB的空间来保存FAT
    - 1.8 KB的空间来保存FAT

15

链接组织方式

2. 显式链接

- 优点
  - 便于快速查找
- 缺点
  - FAT很大，需较大的内存空间

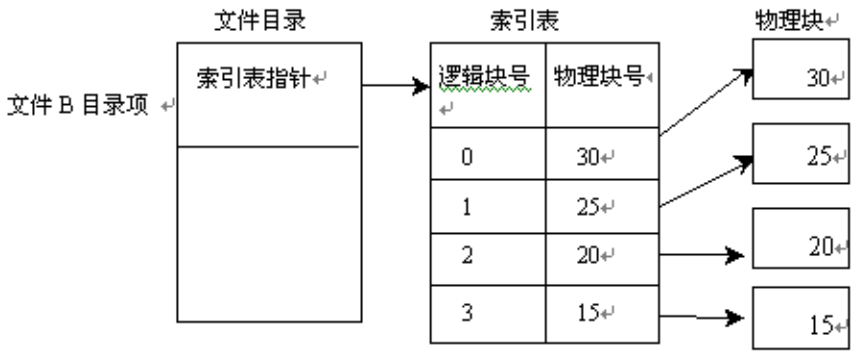
16

链接组织方式

- 优点
  - 消除了外部碎片，提高外存利用率
  - 文件动态增长时，可动态地为它分配盘块
  - 文件的增删改方便，不需事先知道文件长
- 缺点
  - 存取速度慢
  - 只适于顺序存取，不适于随机存取
  - 可靠性差，若某一块指针出错，则链断开
  - 更多的寻道次数和寻道时间
  - 链接指针占用一定的空间

17

索引组织方式



18

索引组织方式

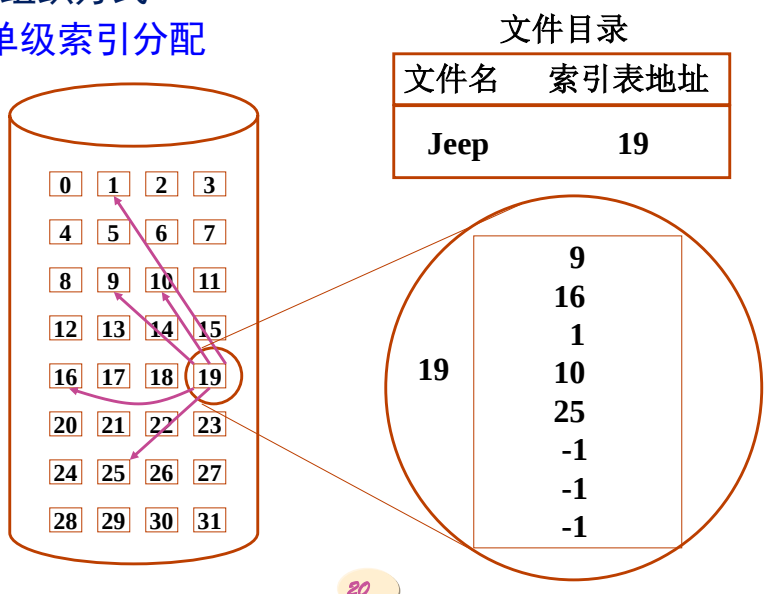
1. 单级索引分配

- 链接分配存在的问题
  - 不能支持高效的直接存取，要对一个较大的文件进行直接存取，须首先在FAT中顺序地查找许多盘块号。
  - FAT需占用较大的内存空间
- 索引分配
  - 为每个文件分配一个索引块，把分配给该文件的所有盘块号都记录在该索引块中
  - 在建立一个文件时，便为之建立的目录项中填上指向该索引块的指针
- 支持直接访问
  - 对于大文件而言，该方式优于链式分配方式

19

索引组织方式

1. 单级索引分配



20

索引组织方式

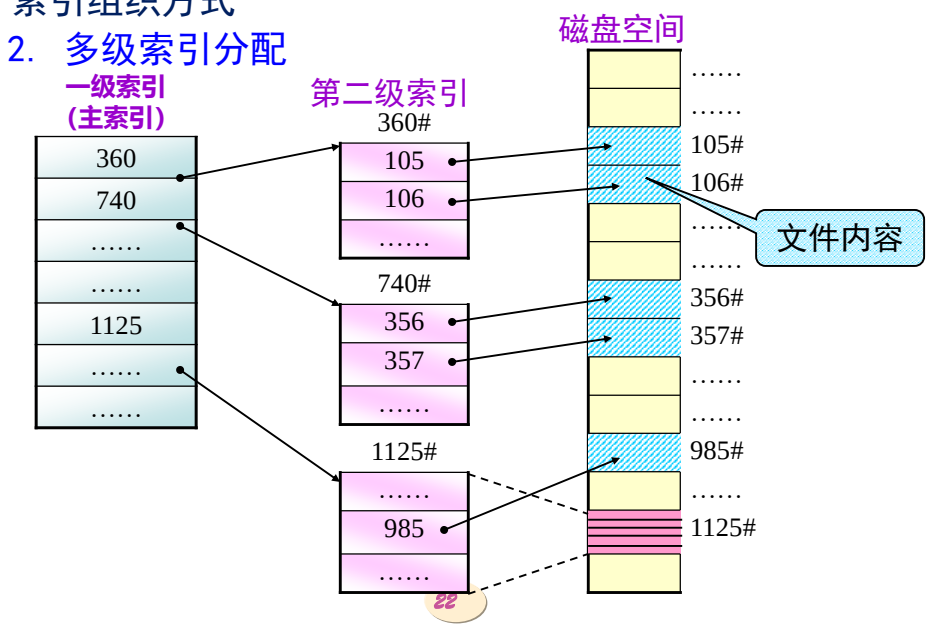
1. 单级索引分配

- 若每个盘块大小为1KB，每个盘块号占4B，则索引块中可存放256个盘块号，即采用这种索引方式时每个文件大小不能超过256KB
- 索引表组织
  - 链接模式：一个盘块一个索引表，多个索引表链接起来
  - 多级索引：将一个大文件的所有索引表（二级索引）的地址放在另一个索引表（一级索引）中

21

索引组织方式

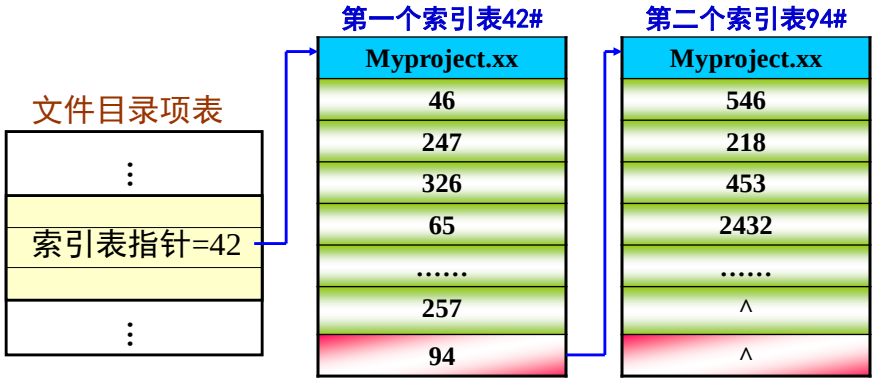
2. 多级索引分配



22

索引组织方式

2. 多级索引分配



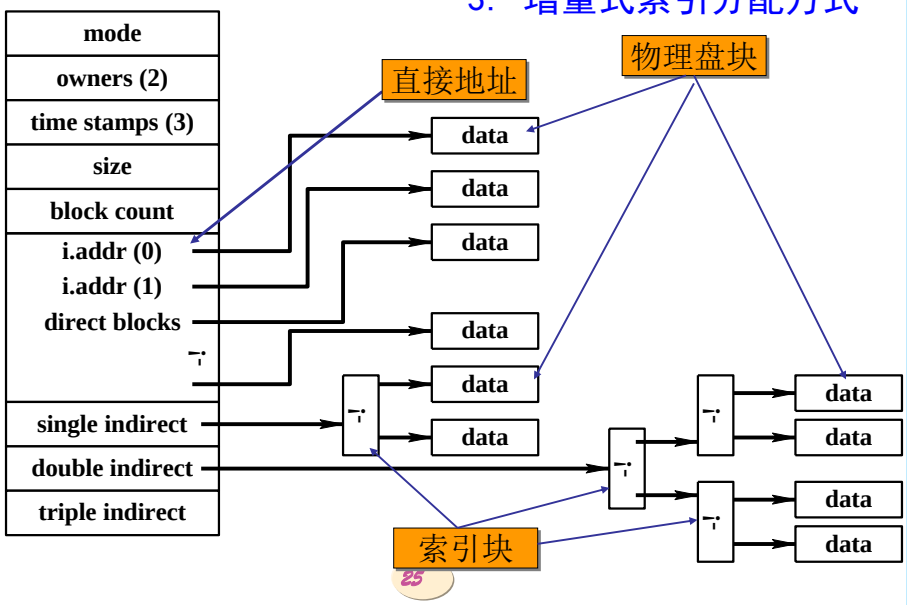
索引组织方式

2. 多级索引分配

- 若每个盘块大小为1KB，每个盘块号占4B，则一级索引块中可存放256个盘块号，即对应256个二级索引块
- 每个二级索引块可对应256个物理磁盘块，采用这种索引方式时每个文件大小不能超过256*256*1KB=64MB
- 若每个盘块大小为4K，则最大文件大小为1K*1K*4K=4GB

索引组织方式

3. 增量式索引分配方式



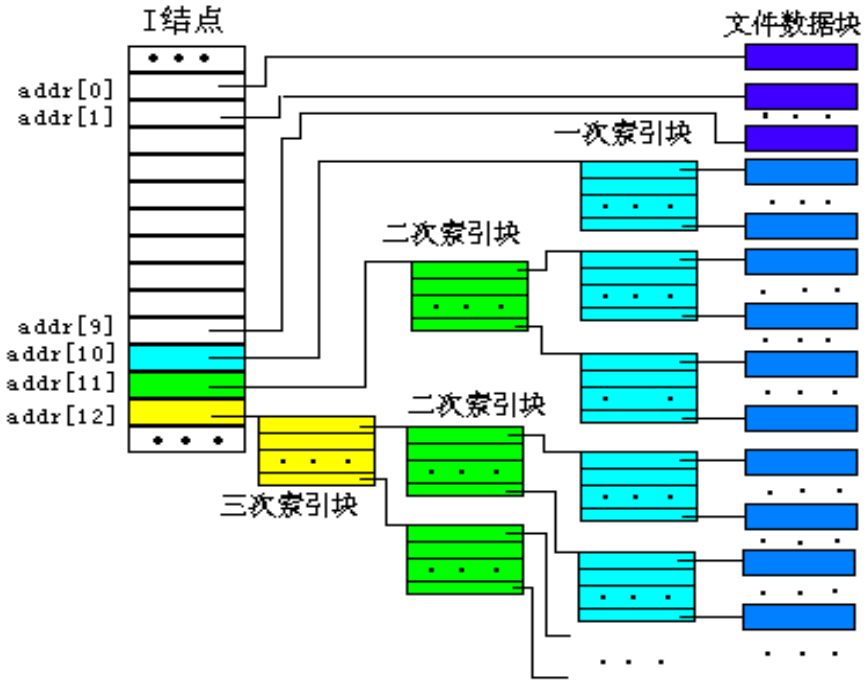
25

索引组织方式

3. 增量式索引分配方式

- 直接地址
  - 为了提高对文件的检索速度， 在索引结点中可设置10个直接地址项， 即用iaddr(0)~iaddr(9)来存放直接地址
- 一次间接地址
  - 对于大、 中型文件，可再利用索引结点中的地址项iaddr(10)来提供一次间接地址。这种方式的实质就是一级索引分配方式
- 多次间接地址
  - 当文件长度大于4 MB+40 KB时(一次间址与10个直接地址项)， 系统还须采用二次间址分配方式。这时，用地址项iaddr(11)提供二次间接地址。该方式的实质是两级索引分配方式

26



8.1 外存的组织方式 第八章 磁盘存储器的管理

索引组织方式

- 优点：
  - 保持了链接结构的优点
  - 能顺序存取和随机存取
  - 满足了文件动态增长、插入删除的要求
  - 充分利用外存空间
- 缺点：
  - 较多的寻道次数和寻道时间
  - 索引表带来系统内外存空间，存取时间开销
  - 对于小文件，空间浪费严重

28

- 连续文件：
  - 优点：不需要额外空间，在文件目录中指出文件的大小和首块的块号即可，顺序访问效率高。适于顺序存取。
  - 缺点：动态增长和缩小系统开销大；文件创建时要求用户提供文件的大小；存储空间浪费较大。
- 链式文件：
  - 优点：适应于顺序访问
  - 缺点：文件的随机访问系统开销较大
- 索引文件：
  - 优点：适应于顺序存访问，也适应于随机访问
  - 缺点：索引表的空间开销和文件索引的时间开销

例：

一个存储于磁盘上的文件系统，其中的文件由大小512B的块组成。假定每一个文件有一个文件目录项。该目录项包含此文件的名字、文件长度以及第一块（或第一索引块）和最后一块的位置，而且该目录项位于内存。对于索引结构文件，该目录项指明第一索引块、该索引块又依次指向511个文件块且有一个指向下一个索引块的指针。

针对连续、链接、索引结构的每一种，如果当前位于逻辑块10（即最后一个访问的块是逻辑块10）且希望访问逻辑块4，那么，必须分别从磁盘上读多少个物理块？

1. 采用顺序结构：

- 文件存放在连续的磁盘块中
- 可以从内存文件目录项中查找文件存放的第一块的地址
- 块号加4即得到第4个逻辑块的块号
- 将此物理块读入内存

采用连续结构时，需要从磁盘上读入1个物理块。

2. 采用链接结构

•文件以链接方式存放在磁盘上
•首先从内存的该文件目录中查找到文件存放的第一块地址
•若逻辑号从1开始从磁盘上读取逻辑块1对应的物理块
•查找逻辑块2对应的物理块地址
•从磁盘上读取逻辑块2对应的物理块
•从中查找到逻辑块3对应的物理块地址
•把逻辑块4对应的物理块读入内存

因此，采用链接结构时，
若逻辑块从1开始编号，则需要从磁盘上读4个物理块；
若逻辑块从0开始编号，则需要从磁盘上读5个物理块。

3. 采用索引结构

- 文件的存储地址在索引表中
- 每个索引块中可存放511个文件块的地址
- 由于当前位于逻辑块10与逻辑块4对应的索引块相同
- 该索引块应该已在内存
- 从内存的该索引块中查找到逻辑块4对应的物理块号
- 从磁盘上将此物理块读入内存

因此，采用索引结构时，需要从磁盘上读1个物理块。

存储设备、文件物理结构、存取方法的关系

存储介质	磁带	磁盘		
物理结构	连续结构	连续	链接	索引
存取方式	顺序存取	顺序	顺序	顺序
		随机		随机

- 解决的问题：如何为新创建的文件分配存储空间？
- 解决的方法：

1、分配方式：

（1）连续分配：访问速度高，但会产生外存零头。

（2）离散分配：访问速度慢，但能有效利用外存空间。

2、分配时数据结构

3、分配回收算法

（分配的基本单位都是磁盘块）

空闲表法和空闲链表法

文件存储空间管理的基本分配单位是盘块

空闲表法和空闲链表法

1. 空闲表法

- 空闲表属于连续分配方式，与内存的动态分配方式相同，为每个文件分配一个连续的存储空间
- 为外存上的所有空闲区建立一张空闲表，每个空闲区对应于一个闲表项，将所有空闲区按起始盘块号递增的顺序排列
- 存储空间的分配与回收可采用 *首次适应算法*、*循环首次适应算法*等
- 对*对换空间*的分配采用连续分配提高速度
- 系统中的较小文件也采用连续分配方式

空闲表法和空闲链表法

1. 空闲表法

- 将存储空间中，各个空闲分区登记在一张表中。一个分区对应一个表项，并将所有空闲分区按其起始存储块号递增的次序排列。

空闲区序号	第一空闲盘块号	空闲盘块数
0	10a8	12
1	9002	98
2	a6002	4096
……	……	……

空闲表法和空闲链表法

1. 空闲表法

- 分配
  - 系统为某新创建的文件分配空闲盘块
  - 先顺序地检索空闲表的各表项
  - 找到第一个其大小能满足要求的空闲区
  - 将该盘区分配给用户(进程)
  - 修改空闲表
- 回收
  - 系统在对用户所释放的存储空间进行回收
  - 考虑回收区是否与空闲表中插入点的前区和后区相邻接
  - 对相邻接者应予以合并

空闲表法和空闲链表法

2. 空闲链表法

将所有空闲盘区，拉成一条空闲链

空闲表法和空闲链表法

2. 空闲链表法

（1）空闲盘块链

- 将磁盘上所有空闲区空间，为盘块为单位拉成一条链
- 创建文件请求分配存储空间：

系统从链首开始，依次摘下适当数目的空闲盘块链给用户

- 删除文件释放存储空间：

系统将回收的盘块依次插入空闲盘块链的末尾

- 优点：分配和回收一个盘块的过程非常简单
- 为一个文件分配盘块时可能要重复多次操作

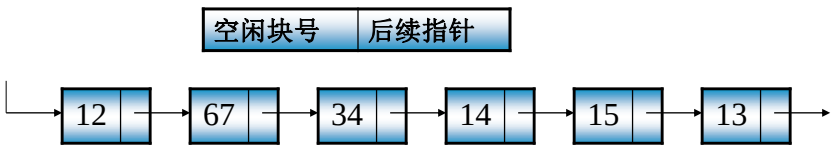
41

空闲表法和空闲链表法

2. 空闲链表法

（1）空闲盘块链

- 将所有空闲盘块形成一个链表
- 链首分配+链尾回收



- 难于找到连续的空闲盘块，重复操作多次，效率低

42

空闲表法和空闲链表法

2. 空闲链表法

（2）空闲盘区链

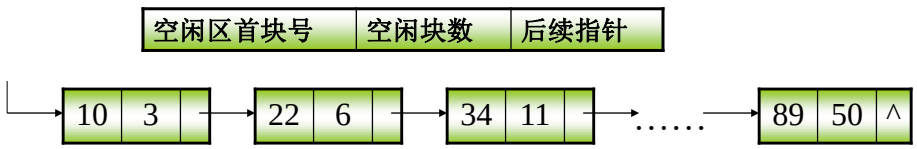
- 将磁盘上所有空闲盘区拉成一条链
  - 每个盘区上包含若干用于指示下一个空闲盘区的指针
  - 指明盘区大小的信息
- 分配盘块通常采用首次适应算法（显式链接法）
- 回收时将回收区与空闲盘区相合并

空闲表法和空闲链表法

2. 空闲链表法

（2）空闲盘区链

- 将所有空闲盘区形成一个链表。



位示图

- 用二进制的一位表示磁盘盘块的使用情况
- "0"表示盘块空闲，"1"表示盘块已分配
- 位示图:

所有盘块所对应的二进制位构成的一个集合

- 通常可用m*n个位数来构成位示图

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

位示图

盘块的分配

- 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)
- 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

$$b = n(i - 1) + j$$

- 修改位示图， 令map[i,j]=1

位示图

❑ 盘块的回收

- 将回收盘块的盘块号转换成位示图中的行号和列号
  - i = (b - 1) DIV n + 1
  - j = (b - 1) MOD n + 1
- 修改位示图，令map [i,j]=0 （当“0”表示盘块空闲时）

成组链接法

1. 空闲盘块的组织

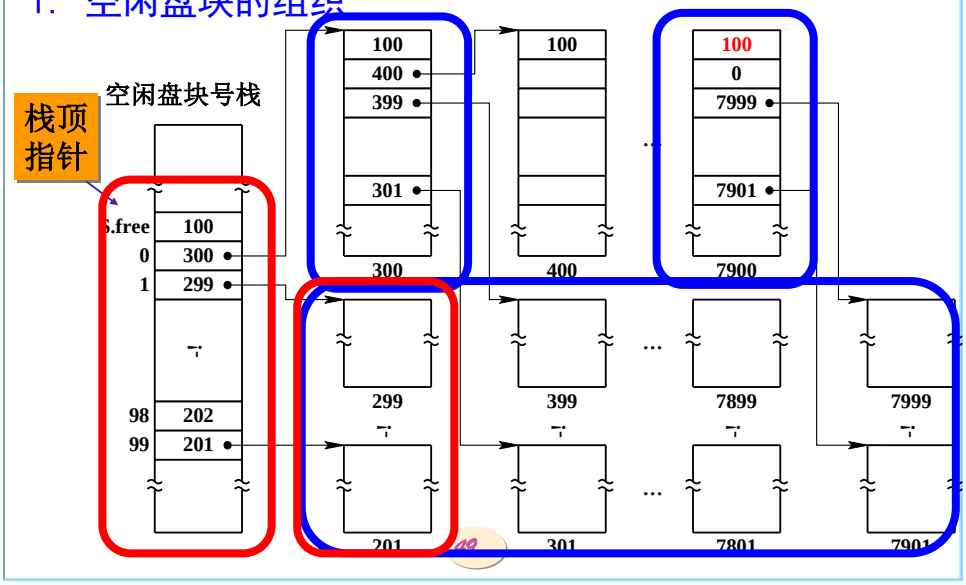
- 将空闲表和空闲链表结合形成的空闲盘块管理方法
- 空闲盘块号栈

存放一组空闲盘块号以及栈中尚有的空闲盘块数N

- 文件区中空闲盘块被分成若干个组，如100块/组
- 每组盘块数和盘块号记入前一组第一个盘块
- 第一组空闲盘块数和所有盘块号记入空闲盘块号栈

成组链接法

1. 空闲盘块的组织



成组链接法

1. 空闲盘块的组织

- 设每100盘块为1组，系统共10000个盘块，从201至7999用于文件区，则第1组为盘块号201-300，第2组为301-400，. . . ，最后一组为7901-7999
- 每一组的盘块总数N和盘块号记入前一组的第一个盘块S.free(0)~S.free(99)
- 将第一组盘块总数和盘块号记入空闲盘块号栈
- 最末一组S.free(0)为“0”，表示空闲盘块链结束

成组链接法

2. 空闲盘块的分配

- 检查空闲盘块号栈是否上锁：
  - 如未上锁，便从栈顶取出一空闲盘块号
  - 将与之对应的盘块分配给用户
  - 将栈顶指针下移一格
- 若S.free(0)，当前栈最后一个可分配的盘块号
- 调用磁盘读，将栈底盘块号所对应盘块的内容读入栈中，作为新的盘块号栈的内容，并把原栈底对应的盘块分配出去
- 分配相应缓冲区
- 把栈中的空闲盘块数减1并返回

51

成组链接法

3. 空闲盘块的回收

- 将回收盘块的盘块号记入空闲盘块号栈的顶部
- 执行空闲盘块数加1操作
- 栈已满
  - 记入新回收的盘块中
  - 将其盘块号作为新栈底

52

文件系统性能

(1) 改进文件的目录结构以及检索目录的方法来减少对目录的查找时间

(2) 选取好的文件存储结构，以提高对文件的访问速度

(3) 提高磁盘的I/O速度，能将文件中的数据快速地从磁盘传送到内存中，或者相反

块高速缓存

磁盘高速缓存(Disk Cache)

(1) 如何将磁盘高速缓存中的数据传送给请求进程?

(2) 采用什么样的置换策略?

(3) 已修改的盘块数据在何时被写回磁盘?

块高速缓存

1. 数据交付(Data Delivery)方式

* 问题引出：

如果I/O请求所需要的数据能从磁盘高速缓存中获取

需要将磁盘高速缓存中的数据传送给请求进程

* 数据交付：将磁盘高速缓存中的数据传送给请求者进程

* 数据交付给请求进程的两种方式：

(1)数据交付 (2)指针交付

块高速缓存

2. 置换算法

除了考虑到最近最久未使用原则外，还考虑：

(1) 访问频率

(2) 可预见性

(3) 数据的一致性

块高速缓存

3. 周期性地写回磁盘

根据LRU算法，经常被访问盘块数据一直保留在高速缓存中

廉价磁盘冗余阵列(RAID)

并行交叉存取

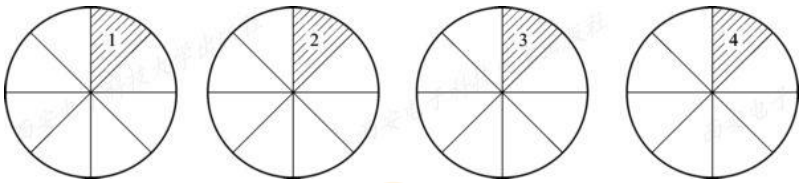
(1) 并行交叉存取技术应用到磁盘存储系统中

(2) 将多台磁盘驱动器每一盘块中数据分为若干子盘块数据

(3) 把每一个子盘块的数据存储到各个不同磁盘的相同位置

(3) 采取并行传输方式将一个盘块的数据传送到内存

(4) 各个盘块中的子盘块数据同时向内存中传输



廉价磁盘冗余阵列(RAID)

RAID的分级

(1) RAID 0级

(2) RAID 1级

(3) RAID 3级

(4) RAID 5级

(5) RAID 6级和RAID 7级

廉价磁盘冗余阵列(RAID)

RAID的优点

(1) 可靠性高，除了RAID 0级外，其余各级都采用了容错技术。当阵列中某一磁盘损坏时，并不会造成数据的丢失。此时可根据其它未损坏磁盘中的信息来恢复已损坏的盘中的信息。其a可靠性比单台磁盘机高出一个数量级。

(2) 磁盘I/O速度高，由于采取了并行交叉存取方式，可使磁盘I/O速度提高N-1倍。

(3) 性能/价格比高，RAID的体积与具有相同容量和速度的大型磁盘系统相比，只是后者的1/3，价格也只是后者的1/3，且可靠性高。以牺牲1/N的容量为代价，换取了高可靠性。

文件系统性能

提高磁盘I/O速度的其他方法

- 提前读
- 延迟写
- 优化物理块的分布
- 虚拟盘:

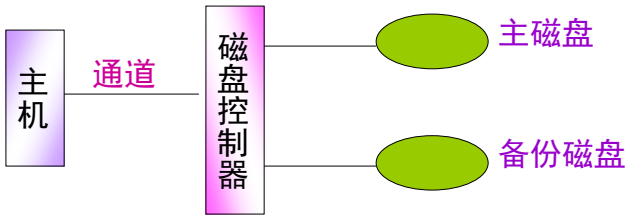
利用内存空间去仿真磁盘，形成所谓虚拟盘，又称为RAM盘。该盘的设备驱动程序也可以接受所有标准的磁盘操作，但这些操作的执行不是在磁盘上而是在内存中,对用户透明

第一级容错技术SFT-Ⅰ

- 写后读校验：
  - 将刚写入磁盘的数据读出来，与内存源数据进行比较
  - 若两者一致，便认为此次写入成功
  - 否则重写
  - 若重写后两者仍不一致，则认为该盘块有缺陷，此时，便将应写入该盘块的数据，写入到热修复重定向区中

第二级容错技术SFT-II

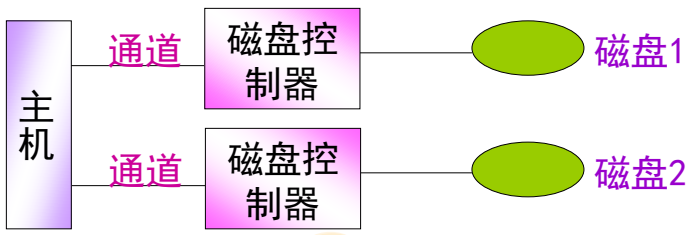
- 磁盘镜像(Disk Mirroring)
  - 同一磁盘控制器下再增设一个完全相同的磁盘驱动器
  - 写入时同时写到主盘和镜像盘，读出时只从主盘读
  - 可以解决磁盘驱动器故障的问题
  - 磁盘利用率为50%



63

第二级容错技术SFT-II

- 磁盘双工(Disk Duplexing)
  - 两块硬盘各配备自己的磁盘控制器和通道
  - 写入时将数据写到两个处于不同控制器和通道下的磁盘上
  - 可以解决磁盘控制器故障和通道故障的问题
  - 代价高



64

基于集群技术的容错功能

- SMP服务器（Symmetrical Multi-Processing,意指“对称多处理”技术）来实现集群系统服务器
- 所谓集群：一组互连自主计算机组成的计算机系统
- 集群系统工作模式有三种：
  - ① 热备份模式
  - ② 互为备份模式
  - ③ 公用磁盘模式

基于集群技术的容错功能

双机热备份模式

- 备两台服务器
- 两者的处理能力通常是完全相同的
- 一台作为主服务器，另一台作为备份服务器
- 主服务器运行，备份服务器则时刻监视着主服务器的运行
- 主服务器出现故障，备份服务器便立即接替主服务器的工作
- 备份服务器成为主服务器，修复后服务器作为备份服务器

基于集群技术的容错功能

双机互为备份模式

- 两台服务器均为在线服务器，各自完成自己的任务
- 在两台服务器之间，应通过某种专线连接起来互为备
- 每台服务器内都配置两台硬盘
- 一个用于装载系统程序和应用程序
- 另一个用于接收由另一服务器备份数据作为镜像盘
- 服务器发生故障则通过服务器切换来保障服务

基于集群技术的容错功能

公用磁盘模式

- 将多台计算机连接到公共磁盘系统上以减少信息复制开销
- 公共磁盘划分为若干个卷，每台计算机使用一个卷
- 如果某台计算机发生故障，系统重新进行配置
- 根据某种调度策略来选择另一台替代机器
- 替代机器对发生故障的机器的卷拥有所有权
- 替代机器接替故障计算机所承担的任务
- 优点：消除了信息的复制时间，减少了网络和服务器开销

事务

1. 事务的定义

用于访问和修改各种数据项的一个程序单位

一系列相关读和写操作

- 对分布在不同位置的同一数据所进行的读和写操作全部完成时，以托付操作(Commit Operation)终止事务
- 读、写或修改操作失败，执行夭折操作(Abort Operation)
- 事务“退回”(rolled back)

为确保夭折事务不会引起数据的不一致性，须将该事务内刚被修改的数据项恢复成原来的情况

- 事务具有典型的“原子性”

69

事务

2. 事务记录(Transaction Record)

- 事务名：用于标识该事务的惟一名字
- 数据项名：指被修改数据项的唯一名字
- 旧值：修改前数据项的值
- 新值：修改后数据项将具有的值

70

事务

2. 事务记录(Transaction Record)

- 〈 T_i开始〉记录被写入事务记录表中
- T_i写(修改)操作前，写一适当的新记录到事务记录表
- T_i托付时一个〈 T_i托付〉记录写入事务记录表

71

事务

3. 恢复算法

- undo〈T_i〉

把所有被事务T_i修改过的数据恢复为修改前的值

- redo〈T_i〉

把所有被事务T_i修改过的数据设置为新值

处理任何故障而不致使故障造成非易失性存储器中信息的丢失

72

事务

- 如果系统发生故障，系统应对以前所发生的事务进行清理
- 通过查找事务记录表，可以把尚未清理的事务分成两类：
  - 所包含的各类操作都已完成的事务：
    - 既包含了〈T_i开始〉记录，又包含了〈T_i托付〉记录
    - 系统利用redo〈T_i〉过程，把所有已被修改的数据设置成新值
  - 所包含的各个操作并未全部完成的事务：
    - 只有〈T_i开始〉记录而无〈T_i托付〉记录
    - 系统利用undo〈T_i〉过程，将所有已修改数据恢复为修改前的值

检查点

1. 检查点(Check Points)的作用

- Log表中记录的数据也会愈来愈多
- 系统发生故障，事务记录表中的记录清理费时
- 事务在检查点前就做了托付，则在事务记录表中出现检查点记录前的托付记录
- 系统故障时，被T_i修改过的数据，不必再执行redo操作

检查点

2. 新的恢复算法

- 如果把所有在事务T_i以后开始执行的事务表示为事务集T
- 新的恢复操作
  - 对所有在T中的事务T_K，如果在事务记录表中出现了〈T_K托付〉记录，则执行redo〈T_K〉操作
  - 反之，如果在事务记录表中并未出现〈T_K托付〉记录，则执行undo〈T_K〉操作。

75

检查点

2. 新的恢复算法

- 在引入检查点后，可以大大减少恢复处理的开销
- 故障后不需要对事务记录表中的所有事务记录进行处理
- 故障后只需对最后一个检查点之后的事务记录进行处理
- 恢复例程首先查找事务记录表
- 确定在最近检查点以前开始执行的最后的事务

76

并发控制

1．利用互斥锁实现“顺序性”

- 每一个共享对象设置一把互斥锁
- 当一事务T_i要去访问某对象时
  - 应先获得该对象的互斥锁
  - 若成功，将该对象锁住，可对该对象执行读或写操作
  - 其它事务未能获得锁而不能访问该对象

77

并发控制

1．利用互斥锁实现“顺序性”

- 如果T_i需要对一批对象进行访问
  - T_i应先获得这一批对象的互斥锁
  - 将这些对象全部锁住
  - 如果成功，便可对这一批对象执行读或写操作
  - 操作完成后又将所有这些锁释放
  - 但如果在这一批对象中的某一个对象已被其它事物锁住，则此时T_i应对此前已被T_i锁住的其它对象进行开锁
  - 宣布此次事务运行失败，不引起数据的变化

78

并发控制

2. 利用互斥锁和共享锁实现顺序性

- 互斥锁实现顺序性存在效率问题：

  只允许一个事务去读

- 共享锁(Shared Lock)
  - 允许多个事务对同一对象执行读操作
  - 不允许其中任何一个事务对对象执行写操作

79

并发控制

2. 利用互斥锁和共享锁实现顺序性

- 如果事务T_i要对Q执行读操作，则只需去获得对象Q的共享锁
  - 如果对象Q已被互斥锁锁住，则T_i必须等待；
  - 否则，便可获得共享锁而对Q执行读操作
- 如果T_i要对Q执行写操作，则T_i还须去获得Q的互斥锁
  - 若失败，须等待
  - 否则，可获得互斥锁而对Q执行写操作

80

重复数据一致性

1. 重复文件的一致性

- UNIX文件目录目录项
  - ASCII码的文件名
  - 索引结点号(指向一个索引结点)

文件名	i 结点
文件 1	17
文件 2	22
文件 3	12
文件 4	84

重复数据一致性

2. 盘块号一致性的检查

- 利用软件方法构成一个计数器表，每个盘块号占一个表项。
- 每一个表项中包含两个计数器
  - 空闲盘块号计数器
  - 数据盘块号计数器
- 对盘块的数据结构进行检查
  - 先将计数器表中的所有表项初始化为0
  - 用N个空闲盘块号计数器所组成的第一组计数器来对从空闲盘块表(链)中读出的盘块号进行计数；
  - 用N个数据盘块号计数器所组成的第二组计数器去对从文件分配表中读出的、已分配给文件使用的盘块号进行计数
  - 如果情况正常，则上述两组计数器中对应的一对(计数器中的)数据应该互补，反之则说明发生了某种错误。

重复数据一致性

3. 链接数一致性检查

- 在UNIX类型的文件目录中，其每个目录项内都含有一个索引结点号，用于指向该文件的索引结点
- 对于一个共享文件，其索引结点号会在目录中出现多次

例如，当有5个用户(进程)共享某文件时，其索引结点号会在目录中出现5次

- 在该共享文件的索引结点中有一个链接计数count，用来指出共享本文件的用户(进程)数
- 正常情况下两个数据应该一致，否则就会出现链接数不一致性差错

重复数据一致性

3. 链接数一致性检查

- 配置一张计数器表
- 为每个文件建立一个表项含有该索引结点号的计数值
- 检查
  - 从根目录开始查找
  - 在目录中遇到该索引结点号便计数器表相应表项上加1
  - 目录检查后，将计数器表表项中的索引结点号计数值与文件索引结点中的链接计数count值比较
    - 如果两者一致，表示是正确的
    - 否则，便是产生了链接数据不一致的错误

重复数据一致性

3. 链接数一致性检查

- 如果索引结点中的链接计数count值大于计数器表中相应索引结点号的计数值，则即使在所有共享此文件的用户都不再使用此文件时，其count值仍不为0，因而该文件不会被删除。
- 错误的后果：
  - 一些已无用户需要的文件仍驻留在磁盘上
  - 浪费存储空间
- 解决方法：用计数器表中的正确计数值为count重新赋值

85

重复数据一致性

3. 链接数一致性检查

- 出现count值小于计数器表中索引结点号计数值，危险

假如有两个用户共享某文件，但count值为1

  - 其中有一个用户不再需要此文件，count值就会减为0
  - 系统会将此文件删除，并释放其索引结点及盘块
  - 另一共享用户对应的目录项悬空，无法再访问此文件
- 如果该索引结点分配给其它文件，带来非法访问的危险
- 解决方法：将count值置为正确值

86