# 文件系统(一组文件 目录结构)

（用户角度 文件是逻辑外存的最小分配单元）

文件的属性

1. 文件名（同一目录没有重名文件）
2. 标识符 一个系统内各个文件的标识符位移，对用户来说没有可读性，因此标识符是操作系统区分各个文件的一个“内部名称”。
3. 文件类型 （txt.exe.mp4…） 操作系统可以默认的根据这个后缀使用相应的应用软件打开文件
4. 位置：文件存放的路径
5. 保护：决定谁能读写，或者进行访问控制的信息
6. 时间 日期 用户标识

文件操作：  
创建文件 1.找到空间 2.创建一个条目FCB

写文件 使用系统调用 指明文件名词和写入内容 写操作发生 更新指针

读文件 使用系统调用 指明名词和读入文件到内存中的位置 读指针可以被多个进程所共享，降低开销

文件内重定位 设置文件指针的给定值（文件寻址 seek） 不包含真正的IO

文件的删除 按照名称搜索，找到之后释放所有空间，并且删除条目

截短文件 该操作不改变文件的属性，但是将其长度设置为0并且释放其空间

操作系统维护一个打开文件表 open file table 当需要打开某个文件的时候，通过调用系统调用open()找到文件，对于open()的操作，可以设置一些操作模式，比如创建 只写 只读 读写 追加等等 当open()找到文件之后，会加入到打开文件表中，并且返回文件位置的指针，以免再次查找，当不需要该文件的时候，调用close()从表中移除该文件的条目（create 和 delete操作的是已经关闭的文件 ）

多进程的环境可能同时打开同一文件，操作系统创建两个表

两级内部表

单个进程的表记录了每个进程需要打开文件的表，一旦有多个进程打开同一文件，那么指针信息存到单个进程的表中

整个系统的表记录了已经打开文件的条目，如果有新加入的文件，那么单个进程的表和整个系统的表都增加记录，并且整个系统的表设置open count 每次有新加入的文件（多个进程打开同一文件）那么在该进程的表中加入指针等信息，并且在系统表中count++，同样的关闭文件可以count—

（文件指针必须与磁盘文件分开保存 ）

访问权限 记录着单进程表中

共享锁 reader 可以让多个进程并发获取

专用锁 writer 只有一个进程可以获取此锁

强制加锁&建议加锁

强制加锁（windows）一般会保证加锁的完整性，禁止其他进程同时访问

建议加锁（UNIX）一般不阻止其他进程的访问 但是不能修改（在没有获得锁的时候）

文件的分类

1. 无结构文件（流式文件） eg txt ：由一系列二进制/字符流组成
2. 有结构文件 eg excel ：由一个个记录组成，每个记录又是由一个个数据项组成

文件通过目录的方式组织起来，类似一种树形结构

操作系统应该向上提供哪些功能？

例如 System Call: create() read() write() delete() open() close()

与内存一样，外存也被分为很多的存储单元，每个存储单元代表一个地址，并且大小一般为1B。

外存分为磁盘块，每个磁盘块大小相等，并且包含2的整数次幂个地址

文件的逻辑地址也可以分为逻辑块号和块内地址，操作系统同样要将外存的逻辑地址转换为物理地址。

文件的逻辑结构

无结构文件：

有结构文件：有结构文件由一系列的记录组成，每条记录又由若干个数据项组成，一般来说每条记录的一个数据项可以用来作为关键字（比如说 姓名 学号 成绩…）。

记录各条记录的长度（占用的存储空间）是否相等，可以分为定常记录和可变长记录。

定常记录：即每个数据项的长度是固定一直的（例如学号）

可变长记录：有的数据项长度是可以变化的（例如姓名）

有结构文件的逻辑结构：  
顺序文件：

文件的记录一个接一个的存储（逻辑上），记录是可以边长的，也可以是定常的，各个记录在物理上可以顺序存储或者链式存储。

{顺序存储，逻辑上相邻，物理上相邻 / 链式存储 逻辑上相邻，物理上不一定相邻}

顺序文件分为：串结构，顺序结构

串结构：纪录之间顺序与关键字无关

顺序结构：纪录至今的顺序按照关键字顺序排列

问题：

1. 是否能够根据第一个元素得到第i个元素
   1. 链式存储 ：不论是否边长 是否与关键字有关 都不能实现随即存储
   2. 顺序存储：
      1. 可变长记录：无法实现 随机存储
      2. 不可边长记录：
         1. 可以实现随机存取 i\*r
         2. 串结构：由于不是按照关键字顺序排列的，所以每次只能从头找，无法实现
         3. 顺序结构：可以通过关键字找到

索引文件：  
对于不定长的记录文件如果实现随机访问？

索引号 长度 ptr

|  |  |  |
| --- | --- | --- |
| 0 | m0 |  |
| 1 | m1 |  |
| 2 | m2 |  |
| 3 | m3 |  |

|  |
| --- |
| R0 |
| R1 |
| R2 |

索引数据文件

索引表的表项是串结构的

顺序访问 文件信息按顺序 一个记录接着一个记录地加以处理，这种访问方式最为常用

连续存储



支持随机访问 顺序访问

顺序访问速度最快 速度最快

会产生磁盘碎片

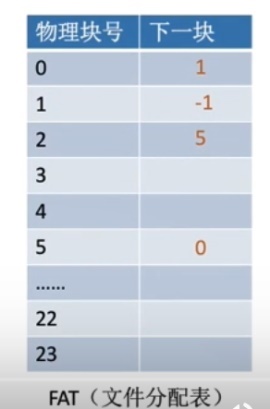
链接分配

隐式分配



支持顺序存储 不支持随机访问 查找效率低

拓展文件方便 不产生外部碎片



显式链接

设置一个FAT文件分配表

在文件分配表FCB中设置一个起始块号，往后查表，表中指针指向下一块



FAT开机的时候读入并且常驻内存 并且块号是隐藏的

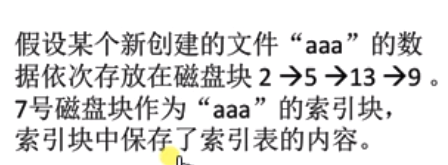
支持顺序访问和随机访问 访问速度快很多

(通过的是文件内部的地址，确定物理地址，由于常驻内存，所以查找快很多 支持随机访问)

支持快速的插入，不会产生碎片，缺点是占用空间

索引分配

索引分配允许文件离散的分配在各个磁盘块中，系统会为每个文件建立一张索引表，索引表记录文件各个逻辑快对应的物理块，索引表存放的磁盘称为索引块，文件数据存放的称为数据块







不同的是，FCB记录的不是数据块的块号，而是索引块的块号…

通过索引块找到索引表，然后在找到对应的块

索引表 逻辑块号可以隐藏

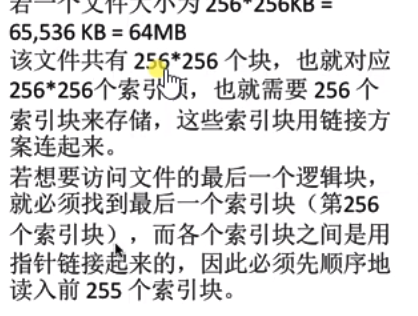
支持随机访问，方便插入 不会产生碎片 但是占用空间

对于一个1KB的索引块，一个索引表项是4B，那么最多可以存放1KB/4B

=2^10/4=2^8=256个索引表项，也就是文件大小不能超过 256\*1K个空间

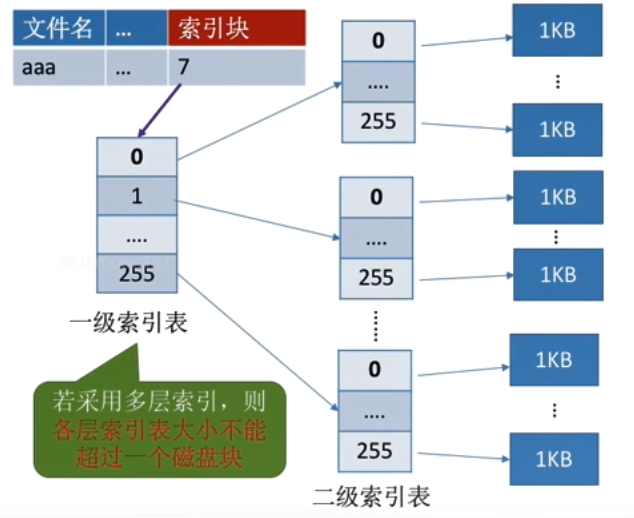
为了解决这个问题

1. 链接方案
   1. 如果一个文件过大，那么将多个索引块链接（存储指针）这个就可以将很长的索引表拆分成很多的部分。



也就是说，效率低，如果64M的文件就有256个块，那么假如想访问最后一个，那么就必须依次访问前255个，效率低下

1. 多级索引



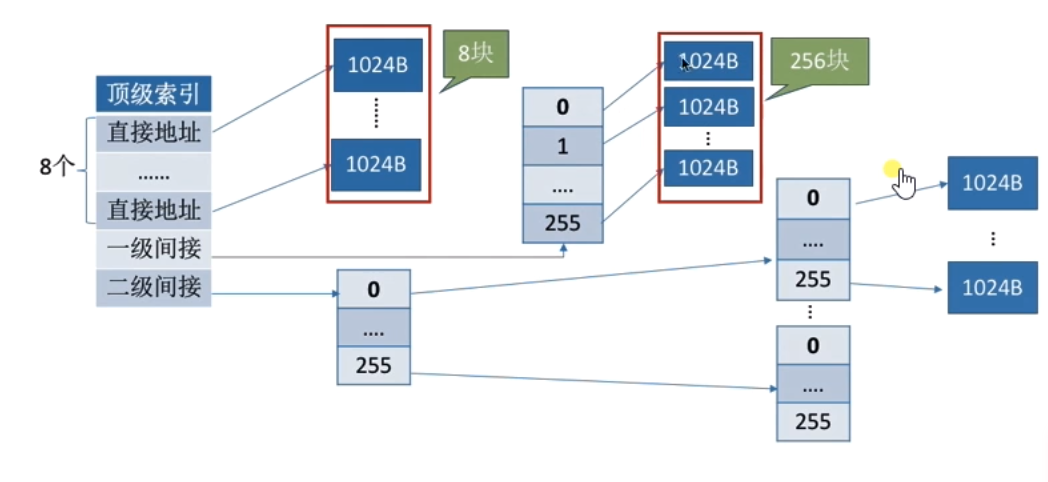
对于两层索引，那么最多可以存放 256\*256个表项，一共可以索引256\*256\*1k的空间，即64M空间

如果访问1026号 那么 1026/256=4 也就是在1级索引表找到4号二级索引表，然后1026%256=2 那么就找到二级索引表2号数据块 3次操作

如果三级索引 访问四次 采用k层索引 那么访问k+1次

混合索引

索引表中存放一些直接索引 一级索引. 二级索引…



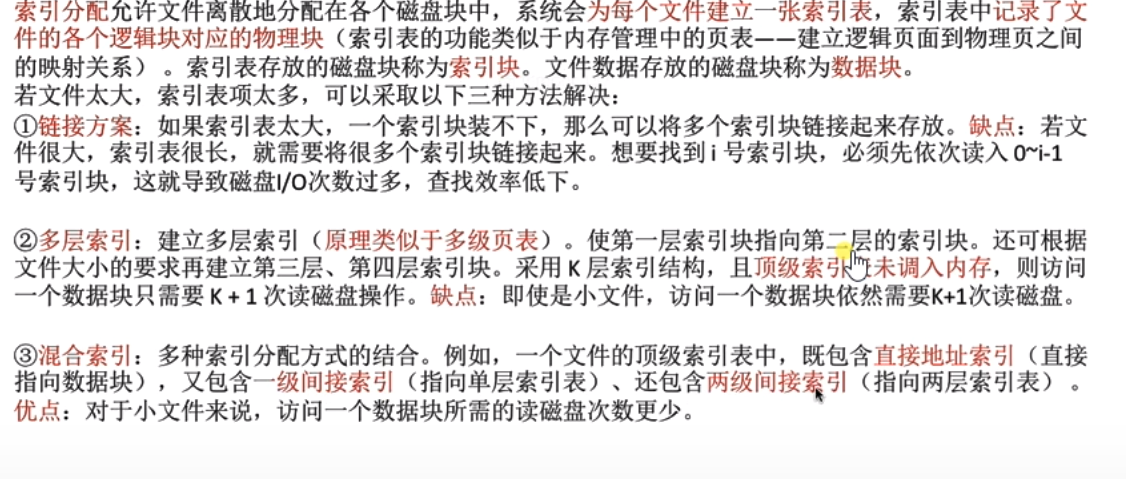
8+256+256\*256=65536个块，即存储65536KB

访问0-7号 2次操作

访问8-263 一级间接索引 3次磁盘操作

访问264-65536 二级间接索引 4次磁盘操作

小文件读取快



空闲空间管理

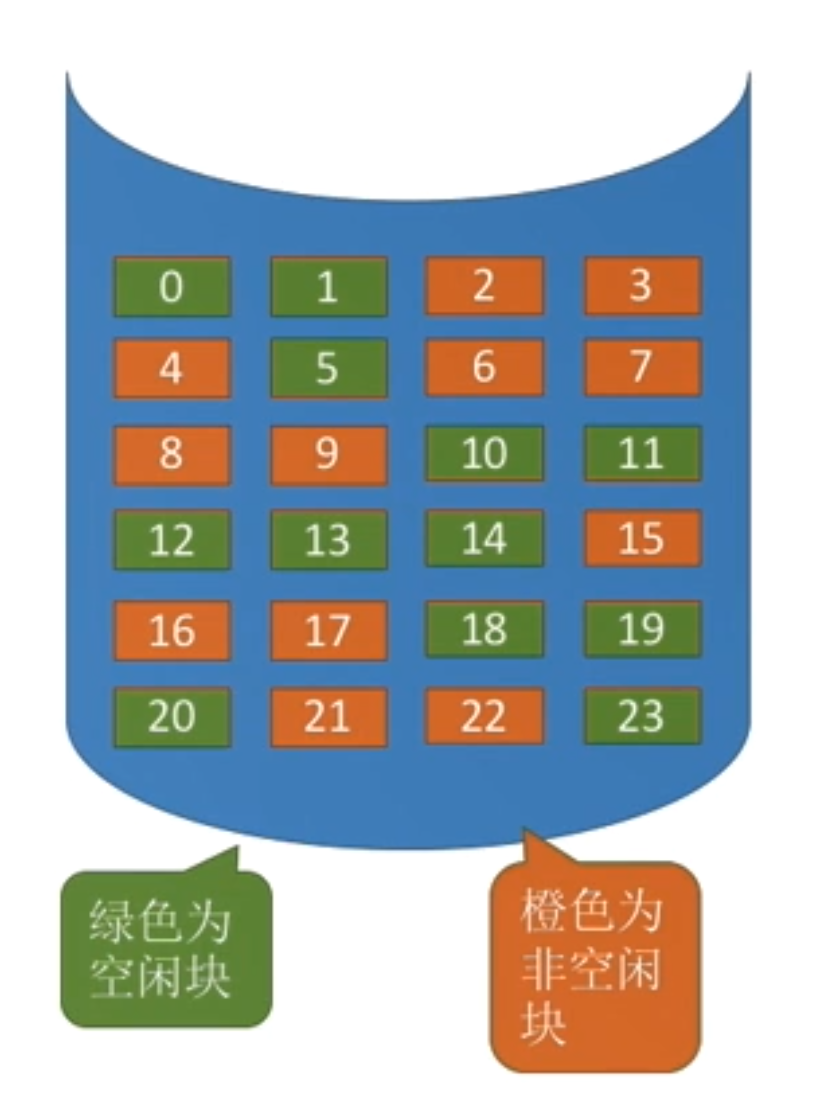
分为目录区和文件区，目录区存放FCB 或者用于存储磁盘的空间管理信息



空闲空间管理

空闲表法：

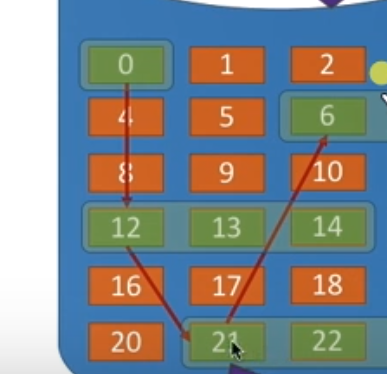
文件的物理结构是连续的。



如何分配磁盘块，为一个文件分配连续的存储空间，可以采用首次适应，最佳适应（最小），最坏（最大）适应算法来决定分配哪个空间

回收磁盘块

1. 回收区前后都没有空闲区间
2. 回收区前后都有空闲区间、（空闲表 - 1）
3. 回收区前面有空闲
4. 回收区后面有空闲 ‘

空闲链表法

空闲盘块链：每一个盘块都指向下一个空闲的盘块

空闲盘区链：有相邻的盘块可以组成一个整体的盘区

空闲盘块链中，每一个盘块存储下一个空闲盘块的指针

空闲盘区链中，每个盘区记录下一个盘区指针和这一个盘区的长度

空闲盘块链 ： 操作系统保存链头和链尾的指针，分配和回收类似队列的方式，分配的时候从对头申请k个盘块，并且将队头重置，回收的时候将盘块接到队尾，并且重置链尾

（适用于离散分配的物理结构 ）

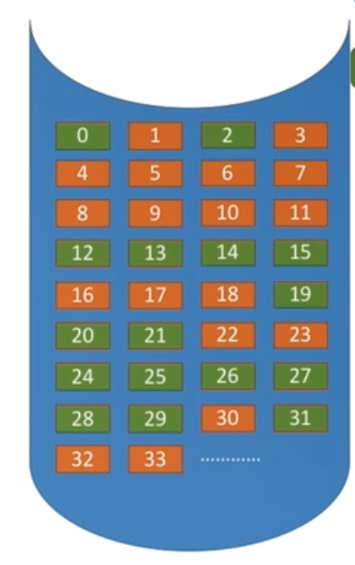
空闲盘区链：系统保存链头尾指针

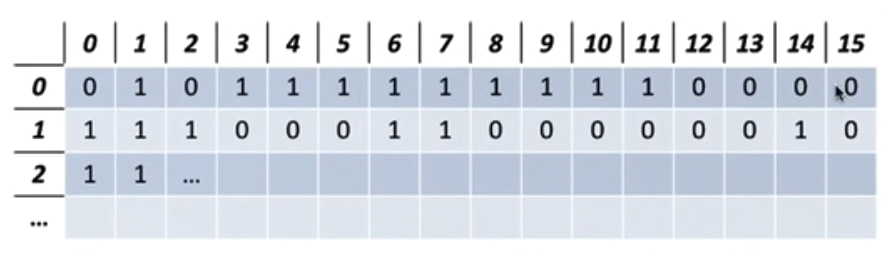
按照首次分配/最佳分配/最坏分配算法检索，找到合适的空闲盘区

如果找不到，那么就将多个盘区分配。

回收的时候注意上下有没有可以合并的，有的话则加入盘块，挂到队尾。

（区分空闲表 空闲表的回收也要合并，而空闲盘块链则不需要）

位示图



每一个二进制位对应一个盘块。0空闲 1 分配

用连续的字表示 一个字长16为 可以用字号 位号的二元组对应盘块号



字号为1 位号为10 b=10+16\*1=26

B块对应的字号是 b/16 = i 位号 b%16 = j

分配： 扫描k个块，并且通过b算出位号和字号，并且将空闲快分配给文件，然后相应的位示图 置1

回收，字号和位号 并且置位0

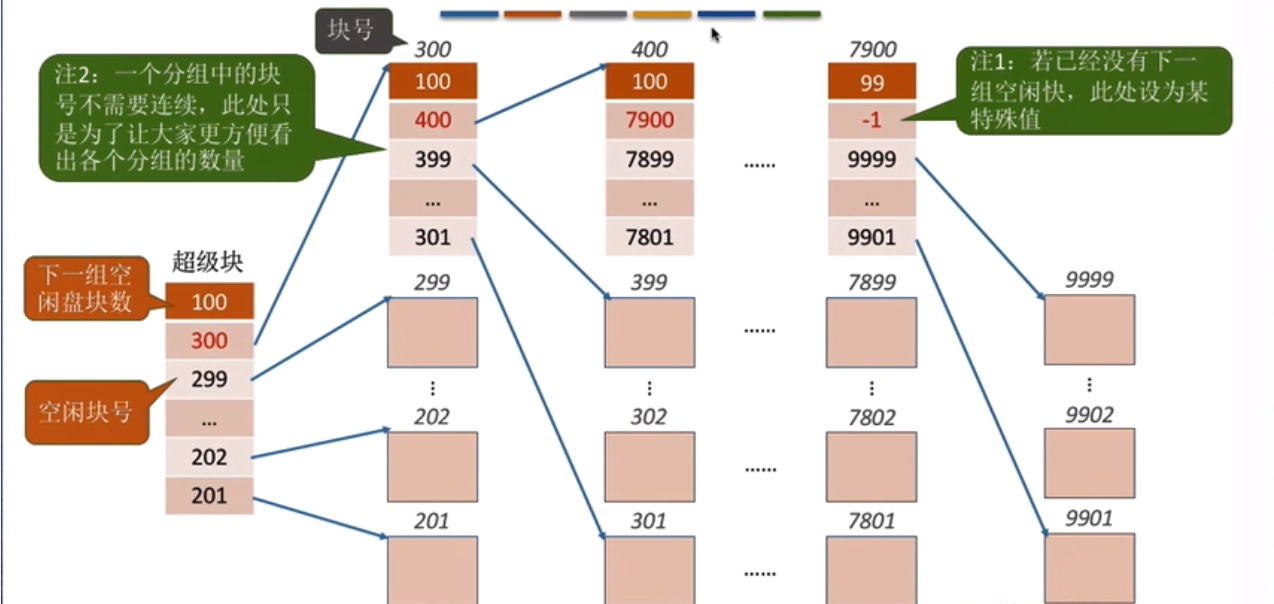
成组链接法（UNIX）

在目录区设置一个超级块

系统启动的时候，需要将超级块读入内存



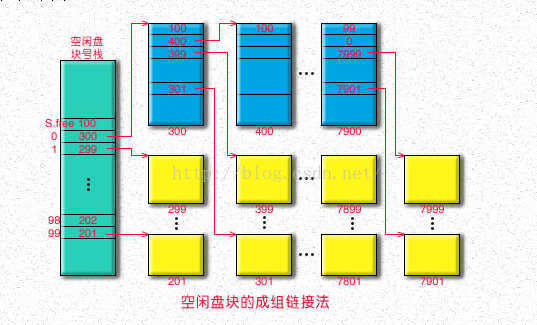
第一个分组300，需要记录下一个分组的空闲盘块信息



成组链接法作为文件存储空间管理方法之一(主要是空闲盘区的管理)，还有其他三种管理方法分别是：空闲表法、空闲链表法和位示图法，它克服了空闲链表法表太长的缺点，但是保持了其优点，即分配和回收一个盘块比较简单。

这么好的文件存储空间管理方法，我们当然要认真学咯。。

首先来看文字解释：成组链接法是Unix系统中常见的管理空闲盘区的方法，它把空闲块分为若干组，每100个空闲块为一组，每组的第一个空闲块记录了空闲块总数和下一组物理空闲块的物理盘块号。理解这一点很重要。特别是对于看下面这张神图。



这张图我得好好解释一下，首先来看左边绿色的空闲盘块号栈，这是第一组(唯一进入内存的一组，只有它会占据存储空间)。看到S.free = 100了没，这表示该组有100个空闲块数目，再往下看，第0号对应的是300，表示下一组物理空闲块的物理盘块号为300，你看它指向的是不是300号对应的磁盘块。再看黄色的块，这些块里保存的才是真正的可用的空闲块，也就是说每组中只有99个块可用。尽管如此，每组还是有100个块的。特别要注意的是，最后一组的下一组盘块号不是没有么，我们这里采用的是结束标记“0”，也就是最右边一个蓝色块的第二项为0。

磁盘块分配和回收的过程。

首先检查超级空闲盘块号(第一组)栈是否已上锁，(提示：这是临界资源)，若已上锁则进程睡眠等待；否则，给空闲磁盘块号栈上锁后，将S.free减1，若S.free仍大于0，即第一组不止一个空闲盘块，则将s\_free[s\_nfree]中登记的空闲盘块分配出去；若S.free为0，即当前空闲盘块号栈中只剩下最后一个空闲盘块，必须先将该盘块的内容读入超级块的空闲盘块号栈中，然后再将该盘块分配出去。若s\_nfree为0，而且栈底登记的盘块号为0，则表示系统中无空闲盘块可分配。

磁盘回收过程：

系统回收空闲盘块时，若第一组不满100块，则只需将回收块的块号填入超级块的空闲盘块号栈栈顶，并将其中的空闲盘块数加1；若第一组已有100块，则必须先将超级块中的空闲盘块数和空闲盘块号写入回收块中，然后将盘块数1和回收块号记入超级块中。

记住一点的是，分配过程是从前往后分配，先分配第一组，然后分配第二组……

回收过程是正好相反，从后往前分配，先将释放的空闲块放入第一组，第一组满了，再开辟一组，之前的第一组变为第二组……

文件系统的层次结构

