Unix 文件系统的静态结构

同济大学计算机系　操作系统课程讲稿　邓蓉　2022-12-15

文件系统基本概念

1、**磁盘**，存储空间分配单位是数据块。一个数据块是连续的 2^n 扇区。
Unix V6/Unix V6++，1 个扇区，512 字节
Linux，连续的 8 个扇区，4096 字节

2、**文件**，按磁盘数据块大小切块，离散存放在磁盘上。每一块是一个逻辑块。同一个文件，相邻逻辑块物理上不必相邻（在磁盘上不必连续存放）。但尽量连续存放一定能提升文件读写效率。

例 1：一个 size 是 10 字节的文件。数据占用磁盘空间，

- Unix V6/Unix V6++，512 字节
- Linux，4096 字节

一个 size 是 1000 字节的文件。占用磁盘空间，

- Unix V6/Unix V6++，1024 字节
- Linux，4096 字节

3、**文件控制块**（File Control Block）

每个文件有一个文件控制块，用来登记文件的属性：
文件名，
文件大小，
地址映射表，存放文件逻辑块和物理块号的映射关系
最后一次访问时刻，
最后一次修改时刻，
文件主 uid 和 文件访问权限（ Unix 是 RWXRWXRWX，Windows 是 ACL ）。

文件存放在哪张磁盘，文件控制块就存放在这张磁盘上。
磁盘上，文件数据 和 文件控制块分开存放。

Windows 系统中，文件控制块 是 目录项。这是个单一的数据结构，存放在目录文件里。

Unix 系统，文件控制块存放在 2 处：文件名存放在目录项里，其余属性存放在索引节点（ inode，i 是 index 索引的缩写 ）里。

目录项 和 索引节点是文件系统最重要的数据结构。

4、 目录 目录项 和 文件树

每个文件系统有一棵文件树。叶子节点是文件（也可能是空目录），其余节点是目录。

4.1 目录 和 目录项

目录是存放目录项 的 特殊文件。这里好关键的是，目录也是一个文件 !!!

目录文件中的每一项是一个目录项。对应该目录下的一个文件或一个子目录。

4.1.1 以标准 Linux 系统为例，看它的根目录和根目录文件。图 1 是 Linux 文件系统的根目录。

图 1

系统中，文件 ID 编号如下：

根目录 /, 2#文件

/bin: 3#文件

/dev: 4#文件

/etc: 5#文件

/home: 6#文件

/lib: 7#文件

/sbin: 8#文件

/usr: 9#文件

/proc: 10#文件

.....

以图 1 左子图列出的 8 个子目录为例，根目录文件应该长这样，图 2.a，包括 10 个目录项。

- 特殊目录项 . 本目录
- 特殊目录项 .. 父目录
- 剩下的 8 个目录项，对应于根目录的 8 个子目录。

根目录文件是 2#文件，图 2a 所示的这 10 个目录项是 2#文件的内容。

目录项是文件系统存储在磁盘上的数据结构。

- Windows 目录项包括文件名和文件所有属性。图 2.b。
- Unix 目录项仅包含文件名 和 文件 ID，图 2.c。文件控制块中的其余属性存放在别处，是一个叫索引节点的数据结构：index node，简称 inode。文件 ID 是索引节点编号。目录项建立文件名和索引节点的映射关系。inode 也是文件系统存储在磁盘上的数据结构。

图 2

4.2 文件树

文件树，分散存放在所有目录文件中。

从根目录开始，长出来文件树。

- 根目录是文件树的根节点
- 递归，执行以下操作
  - 除去特殊目录项，其余每个目录项生出来一条边，长出来一个子节点。边，文件名。子节点，文件 或 目录文件。

所以，文件树，叶子节点是普通文件或空目录。其余节点全是目录文件。下图，一棵文件树。

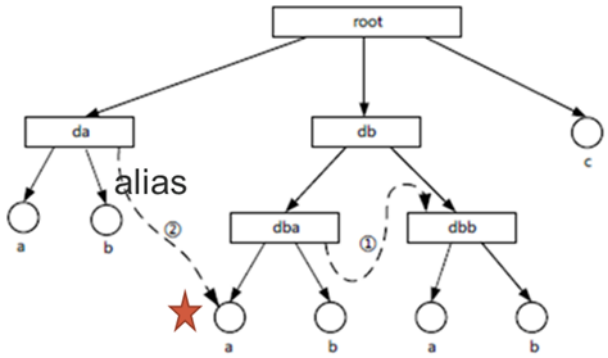


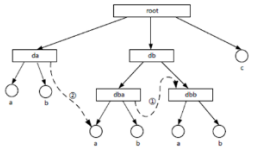
图 3

4.3 目录搜索

目录树的作用：搜索文件（按名搜索文件）。

目录树搜索，有 2 种方式。按绝对路径名搜索，按相对路径名搜索。

- **文件/目录的绝对路径名：从根目录到任何节点，有唯一路径。从树的root开始，把全部中间结点的文件名，用“/”连接起来，形成的字符串。有几条通路，这个文件就有几个文件名：/db/dba/a和/da/alias。**
- **在UNIX V6++系统中，进程的当前工作目录是u_cdir指向的那个节点，没有cd的话，就是用户的家目录。**
  - **Linux系统，用户的家目录是 /home/用户名。登记在花名册文件/etc/passwd中。简写为 ~**
- **文件/目录的相对路径名：从进程的当前工作目录开始，到树上的每个节点，也是只有唯一路径 。这条路径就是节点的相对路径名。**



311

图 4

PS：特殊目录项的作用

. 出现在命令行 或 PATH 环境变量，标识可执行程序 及其 输入输出文件所在的路径。

.. 目录搜索，沿目录树向上走。这个特殊目录项，用来支持相对路径。

4.4 硬链接

有多少个目录项引用同一个文件，这个文件的硬链接数就是几。上图，五星标出的叶子节点 a，2 个目录项引用它。这个文件的硬链接数就是 2。它有两个文件名：

- /da/alias
- /db/dba/a

硬链接的实现，Windows 和 Unix 不一样。

- Windows 好麻烦。有多少文件名，FCB 就要有多少个复本，每个目录项一个复本。上述节点 a，有两个目录项，分别存放在 da 目录文件和 dba 目录文件。这两个目录项，文件名不同，其余文件属性完全相同。下图，左子图所示目录项 for 文件名/da/alias；存在目录文件 da。右子图所示目录项 for 文件名/db/dba/a；存在目录文件 dba。update 文件时，所有目录项要同步修改。好麻烦。

da目录文件中的目录项
文件名　　// “alias”
文件位置信息
文件size
ACL（访问控制列表）
Owner（文件主）
修改事件
……文件的所有属性

dba目录文件中的目录项
文件名　　// “a”
文件位置信息
文件size
ACL（访问控制列表）
Owner（文件主）
修改事件
……文件的所有属性

图 5

- Unix 的设计超级好。

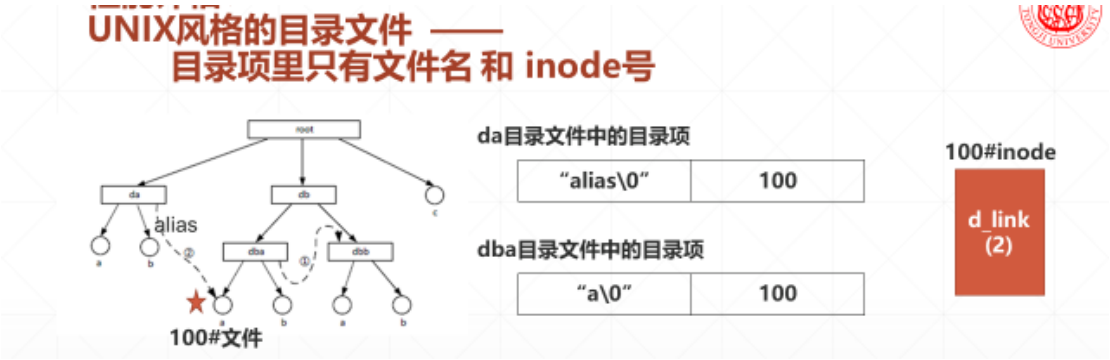


图 6

假定星标的是 100#文件。除文件名，其余所有属性登记在 100# inode 中。
update 文件，只需修改 100# inode。

概念辨析：软链接，就是 Windows 大家熟悉的快捷方式，和硬链接不同。它是一个普通的纯文本文件，里面存放的是文件的绝对路径名。

无论是硬链接 还是 软链接。可以用多个文件名引用同一个文件。方便文件数据共享。

4.5 目录结构的性能

目录结构的性能从 搜索速度 和 树结构改变 这两方面评估。待介绍完 Unix V6++文件系统实现 和 Windows 文件系统实现，再进行评估。

Unix V6 文件系统的静态结构

1、卷结构

超级块	inode区	数据区

图 7

整张磁盘分 3 个区域：

- 超级块，SuperBlock（Unix V6，filsys）。管理整张磁盘的存储资源。是一个尺寸为 1024 字节的数据结构。登记有
  - inode 区长度 s_isize

- 数据区长度 s_fsize
- 管理空闲数据块的数据结构 s_nfree，s_free 和 这个数据结构的锁 s_flock。
- 管理空闲 inode 的数据结构 s_ninode，s_inode 和 这个数据结构的锁 s_ilock。

```
11 class SuperBlock
12 {
21 public:
22     int     s_isize;        /* 外存Inode区占用的盘块数 */
23     int     s_fsize;        /* 盘块总数 */
24
25     int     s_nfree;        /* 直接管理的空闲盘块数量 */
26     int     s_free[100];    /* 直接管理的空闲盘块索引表 */
27
28     int     s_ninode;       /* 直接管理的空闲外存Inode数量 */
29     int     s_inode[100];   /* 直接管理的空闲外存Inode索引表 */
30
31     int     s_flock;        /* 封锁空闲盘块索引表标志 */
32     int     s_ilock;        /* 封锁空闲Inode表标志 */
33
34     int     s_fmod;         /* 内存中super block副本被修改标志，意味着需要更新外存对应的Super Block */
35     int     s_ronly;        /* 本文件系统只能读出 */
36     int     s_time;         /* 最近一次更新时间 */
37     int     padding[47];    /* 填充使SuperBlock块大小等于1024字节，占据2个扇区 */
38 };
```

图 8

- inode 区。尺寸固定，N 个扇区；用来存放 inode。
  inode 是一个尺寸为 64 字节的数据结构。
  每个扇区可以存放 8 个 inode，inode 区可容纳 8*N 个 inode。这张磁盘最多可以存放 8*N 个文件，含目录文件。
  该区域，资源分配单位是 inode（64 字节）。新建一个文件或子目录，用掉一个 inode；删除一个文件或子目录，回收一个 inode。
- 数据区。尺寸固定，M 个扇区；用来存放文件数据，含目录文件。
  该区域，资源分配单位是数据块（一个扇区，512 字节）。新建目录文件，用掉 1 个数据块。write 系统调用向文件写入数据，如果导致文件长度增加，为文件多分配一个数据块。

2、 文件
分配给文件的磁盘存储资源　1 个 inode + 用来存放文件数据（逻辑块）的扇区 + 一个目录项。

2.1 下图是分配给 195#文件“foo”的磁盘资源:
- 195# inode（DiskInode），在 inode 区。
- 用来存放 文件数据 和 索引信息 的若干扇区，在数据区。
- 父目录文件中占据一个目录项。

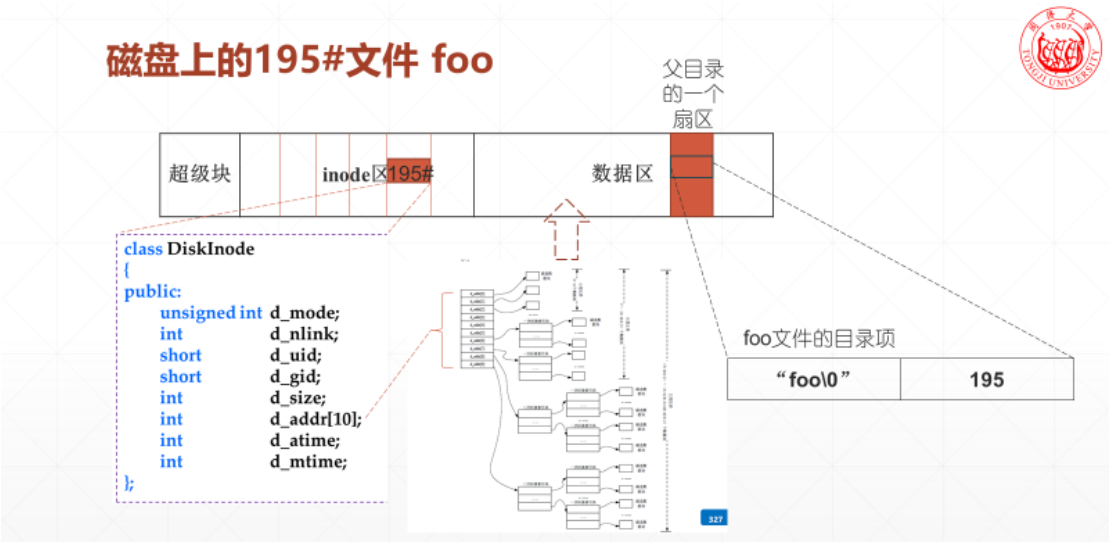


图 9

目录项 DiskInode:

d_nlink: 文件的硬链接数

d_uid: 文件主的 uid。执行 creat 系统调用创建这个文件的进程的 uid

d_size: 文件长度

d_atime: 文件的最后访问时刻

d_mtime: 文件的最后修改时刻

d_addr: 文件混合索引树的根。混合索引树是一棵不平衡多叉树，除根节点外，其余节点是分配给文件的扇区。叶子节点存放文件数据，其余节点是索引盘块，用来存放子节点扇区号。稍后以例示之。

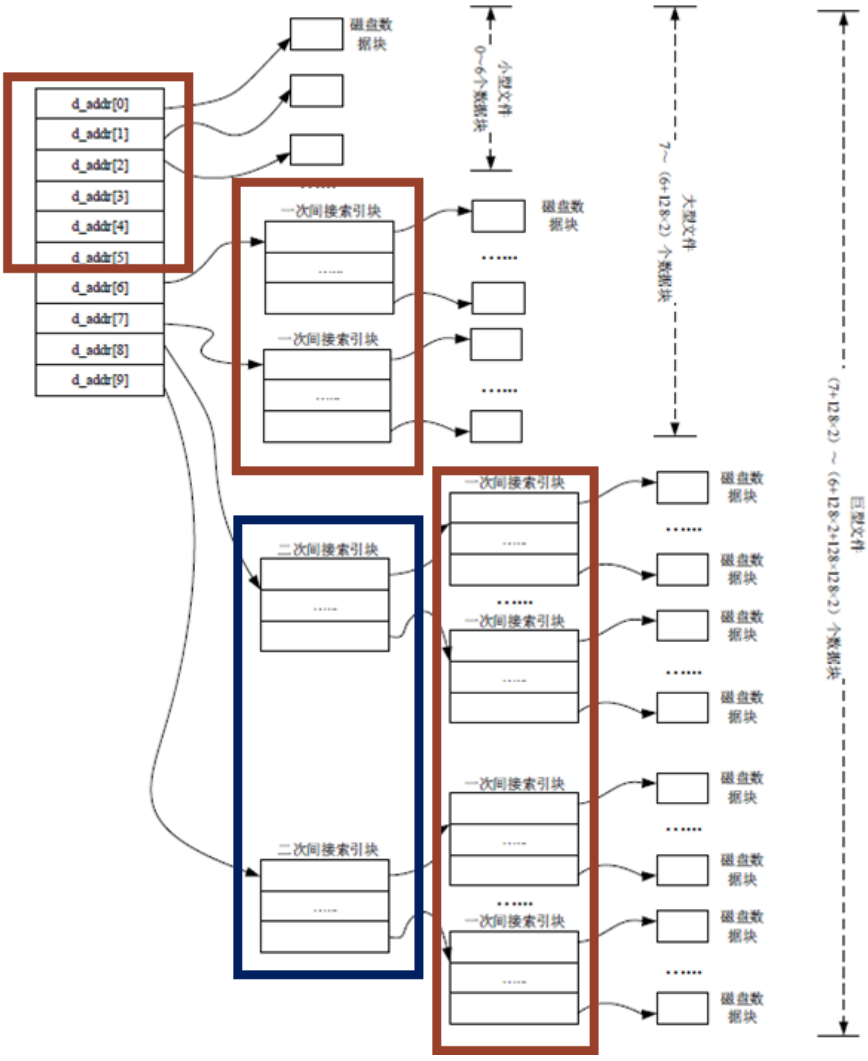
d_mode: 文件的类型 + 访问控制 bits + other flags。

2.2 文件和混合索引树

Unix V6++的混合索引树，从上到下，前 6 个叶子节点挂在 d_addr 数组上。d_addr[6]挂第一个索引盘块，这个扇区最多可以挂 128 个叶子节点（4*128=512 字节）。d_addr[7]挂第二个索引盘块，这个扇区最多可以再挂 128 个叶子节点。用满这 2 个一次索引盘块，文件长度可以达到

（6+2*128）*512 字节， 超过 128K 字节。

d_addr 数组还有 2 个 2 次索引盘块。每个 2 次索引盘块可以挂 128 个一次索引盘块，用来支持很大的文件。



例 1：文件长 1000 字节。存放文件数据，数据区要用多少扇区？

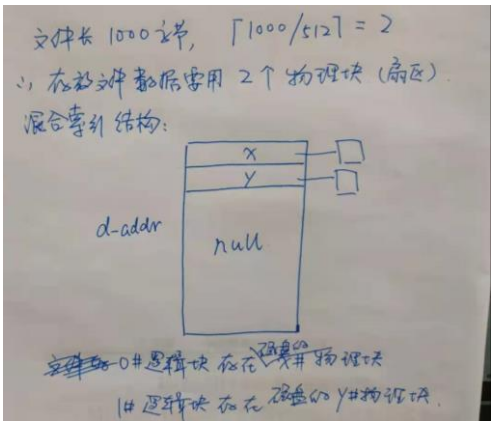


图 10

例 2：文件长 102400 字节。存放文件数据，数据区要用多少扇区？

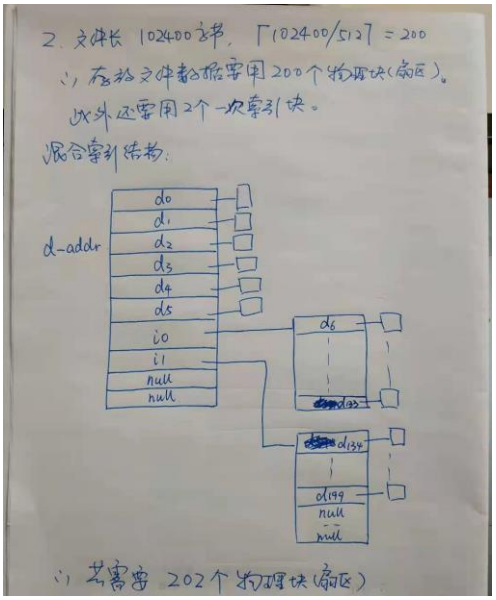


图 11。物理块 d0~d199 装文件内容。i0、i1 是分配给文件的 2 个索引盘块。

例 3:

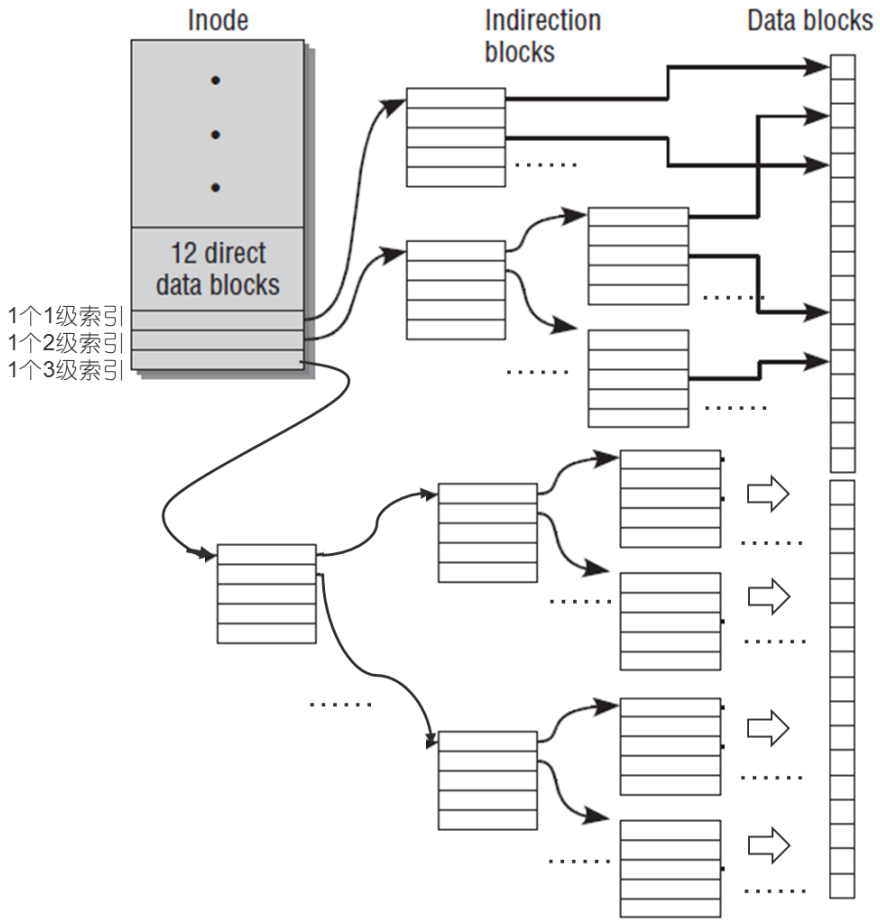


图 12

PS: 注意，混合索引树 d_addr 数组和索引盘块中每个元素的值，是分配给文件的数据块号。

没有挂子节点的元素赋 null。

Unix 混合索引树是不平衡结构。这种设计适合存放很多小文件、只有极少数大文件的磁盘，比如 70，80 年代 AT&T 实验室的那个供程序员使用的分时 Unix V6 系统。好处是显然的，

- 小文件不必使用索引块，索引结构消耗的磁盘空间少。
- 访问小文件很快，因为可以直接从 inode 中得到逻辑块的地址。无需为索引盘块分配缓存、更不需要 IO 磁盘读入索引盘块，文件读写速度快、开销小。

2.3 d_mode

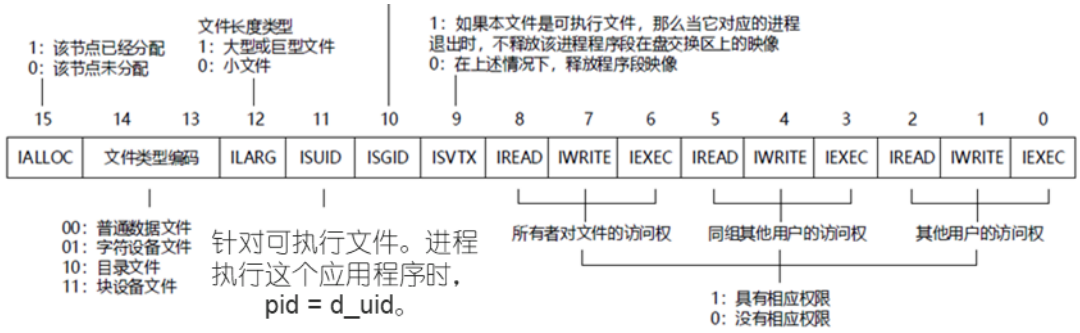


图 13

IALLOC 为 0，这是个空闲 inode。可以把它分配给新文件 或 新创建的子目录。
其余 bits，标识 文件类型 + 访问控制 bits + other flags。

2.3.1 文件类型

Unix：“一切皆文件”。所有 IO 操作都是文件读写。更精确点儿，“一切皆缓存”，不是嘛。

Unix V6++支持 4 种最基本的文件类型。

- 普通文件。 磁盘上的文件。 图 9 是普通文件的格式。inode 中文件类型 是 00。
- 目录文件。 磁盘上存放目录项的文件。 图 9 也是目录文件的格式，与普通文件的区别有 2 处：inode 中文件类型 是 01。数据块里装的不是用户数据，而是等长的目录项。
- 字符设备文件。 终端 tty，硬件设备，是特殊文件。图 14 是字符设备文件的格式，以 Unix V6++系统的控制台，tty1 文件为例。在主文件系统中，特殊文件有一个磁盘 inode，有一个目录项。文件数据来自 tty 缓冲区。d_mode，10。用 major 和 minor 查字符设备开关表，tty1 的缓冲在 cdevsw[major].m_TTY[minor]对象中。
- 块设备文件。 磁盘，是硬件设备，是特殊文件。与字符设备文件相似。

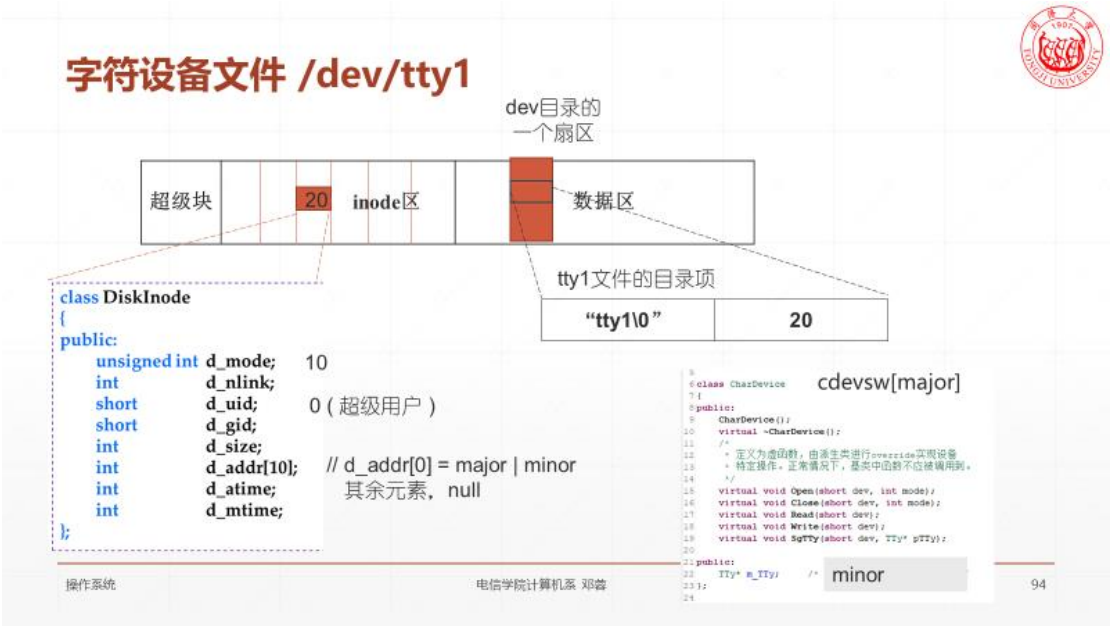


图 14

2.3.2 RWXRWXRWX

Unix 文件的访问控制 bit。R，W，X 为 1 分别表示读权限、写权限和执行权限。

从左至右，第一组、第二组和第三组 RWX 分别是 文件主、同组用户和其它用户 对文件的访问权限。

create 系统调用创建文件时，确定文件主 和 访问控制 bit。

open 系统调用打开文件时，比对 进程控制块 和 文件 inode，用访问控制 bit 确定进程有否相应的文件访问权限。

文件的访问控制 bit 以及其它 bit 的用法，介绍系统调用实现时具体介绍。

3、DiskInode 的编号 & 文件系统获取 DiskInode 的方法

磁盘上每个文件有且仅有一个 DiskInode。DiskInode 的编号是文件 ID。DiskInode，64 字节。一个扇区正好装 8 个 inode。inode 起始于 2#扇区，故，2#扇区：0# inode ~ 7# inode；3#扇区：8# inode ~ 15# inode …… n#扇区：[(n-2)*8, (n-1)*8)。如图：

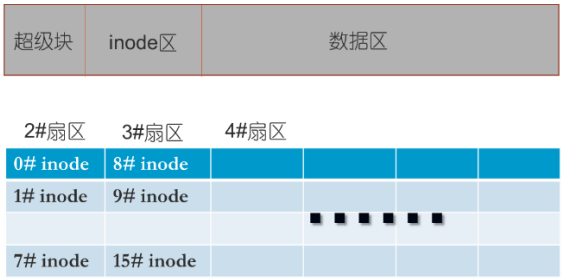


图 15，inode 位置和编号

文件系统为应用程序提供文件读写服务时，需要文件的混合索引树，它的树根在 DiskInode 里。所以，使用文件前我们需要打开文件。打开文件和我们现在讨论的问题相

关的是 2 个步骤：1、根据文件名，得到文件 ID；2、根据 ID，将磁盘上指定位置的 DiskInode 取入内存，供文件系统享用。步骤 2，一个具体的例子，读取 100#inode 的细节：100 除以 8，商 12，余 4。100#inode 在 14#扇区，是这个扇区的 4#inode。

- bp=Bread（0，14）；
- IOmove（bp->b_addr + 256，分配给这个文件的内存 inode，64）。

如图：

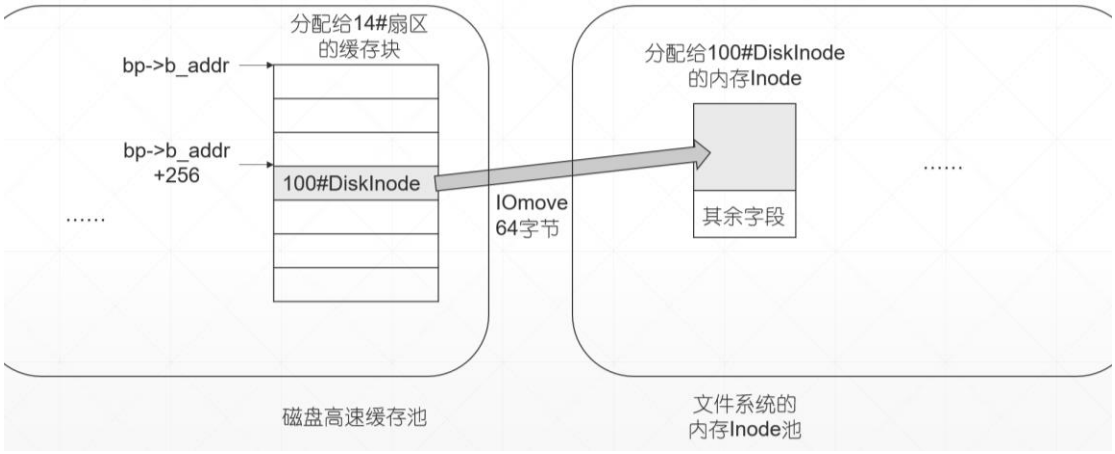


图 16，IOmove 缓存块中的 DiskInode 至内存 Inode

创建文件（子目录）时，需要为文件分配一个空闲 DiskInode。删除文件（空目录）时，DiskInode 置空闲。空闲 DiskInode 的标识，d_mode 中的 IALLOC 比特为 0 & 内存 Inode 池中没有编号相同的 Inode。

4、磁盘空闲资源管理

Unix V6 文件卷，空闲 inode 和 空闲数据块栈式管理，栈顶在 SuperBlock。

```
public:
    int     s_isize;        /* 外存Inode区占用的盘块数 */
    int     s_fsize;        /* 盘块总数 */

    int     s_nfree;        /* 直接管理的空闲盘块数量 */
    int     s_free[100];    /* 直接管理的空闲盘块索引表 */

    int     s_ninode;       /* 直接管理的空闲外存Inode数量 */
    int     s_inode[100];   /* 直接管理的空闲外存Inode索引表 */

    int     s_flock;        /* 封锁空闲盘块索引表标志 */
    int     s_ilock;        /* 封锁空闲Inode表标志 */

    int     s_fmod;         /* 内存中super block副本被修改标志，意味着需要更新外存对应的Super Block */
    int     s_ronly;        /* 本文件系统只能读出 */
    int     s_time;         /* 最近一次更新时间 */
    int     padding[47];    /* 填充使SuperBlock块大小等于1024字节，占据2个扇区 */
```

4.1 空闲 inode 栈

- s_inode 数组，空闲 inode 栈。最多保存 100 个空闲 inode 号。

  s_ninode 是栈中空闲 inode 的数量，兼做栈顶指针。

  栈顶 s_inode[s_ninode-1] 是 待分配的下个空闲 DiskInode（的号码）。这里，DiskInode 和 inode 称谓混用。

分配空闲 inode FileSystem::IAlloc(dev)

create 系统调用需要为新文件申请一个空闲 inode（新建子目录是一样的），Unix V6++弹出栈顶 inode 分配给这个新文件。

- 若栈为空，去磁盘找 100 个空闲 DiskInode，填充 s_inode。
- 弹出栈顶：return（s_inode[--s_ninode]）

```
分配i节点：ialloc( dev)，取空闲i节点栈的栈顶节点
again: if (s_ninode)
            return(s_inode[--s_ninode]);
       else {
          get next 100 free inodes from disk;
          goto again; }
```

回收空闲 inode FileSystem::IFree(dev, num)

删除文件或子目录，回收 DiskInode。

- 若栈已满，什么也不做，返回。　　这种设计是合理的。理由见下图后面的 PS1。
- 将 inode 压栈。

```
回收num#的i节点: ifree(dev,num)
if (s_ninode！=100)
       s_inode[s_ninode++]=num;
```

PS1：　创建文件并不是经常的操作，100 个空闲 inode 可以用很久。若栈已满，这个空闲 inode 就不收了。未来，栈空的时候再去磁盘把它找回来。不费事的。

4.2 空闲盘块号栈

- s_free，空闲盘块号栈。最多保存 100 个空闲 inode 号。
  s_nfree 是池中空闲盘块的数量，兼做栈顶指针。
  栈顶 s_inode[s_nfree-1] 是 待分配的下个空闲盘块（的号码）。

分配空闲块 FileSystem::Alloc(dev)

与空闲 inode 分配相似的出栈操作。与之不同的是，分配最后一个空闲盘块（队长盘块）之前，需要将其中保存的 s_nfree 和 s_free（下一组 100 个空闲盘块的号码）复制到 SuperBlock。

```
1、n = s_ free[--s_nfree];      // pop
2、if ( s_nfree==0 )
        将n#扇区开头的404字节复制到 SuperBlock,
        覆盖  s_nfree和s_free；
3、return(n)；
```

回收空闲块 FileSystem::Free(dev, num)

与空闲 inode 分配相似的入栈操作。与之不同的是，回收盘块时，如果 s_free 数组已满，我们不能对这个空闲块置之不理；而是需要将 SuperBlock 中的 100 个空闲盘块 s_nfree 和 s_free（上一组 100 个空闲盘块的号码）复制进这个新回收的空闲盘块，清空 SuperBlock 中的栈之后，压栈这个新的空闲块。

```
if (s_nfree==100) //新组长空闲块num，装入SuperBlock中的链接信息
{
   将 SuperBlock 中, s_nfree 和 s_free、s_nfree复制 到num#扇区；
   s_nfree=0；
}
 s_ free[s_nfree++] = num;    // pop
```

PS：系统没有简单的办法判断一个盘块空闲还是已分配。所以，必须登记回收的每个空闲盘块。

4.3 空闲资源管理策略

早期的 Unix 系统和 Windows 系统均采用栈式方式管理空闲资源。

图 17

4.3.1 Unix V6 用的是链式管理法，用一根链管理所有空闲盘块。如下图，一块有 449 个空闲块的硬盘。

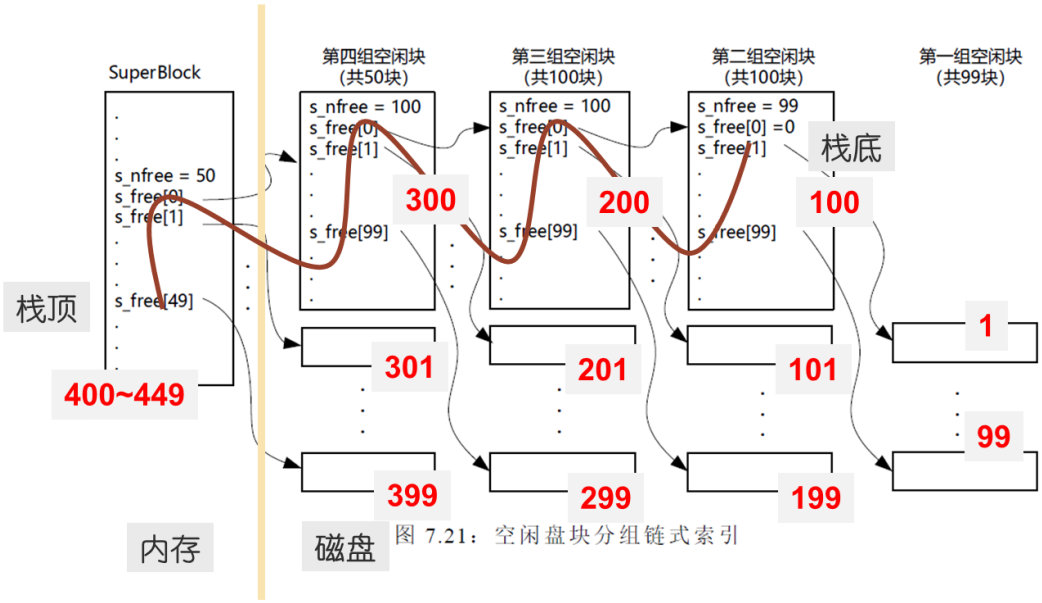


图 7.21：空闲盘块分组链式索引

图 18

链，存放在 SuperBlock 和空闲盘块（队长盘块）中，不需要消耗额外磁盘空间。这种方法适合小容量磁盘或非常昂贵的存储介质。但缺点也是非常明显的，栈式管理，后进先出：

- 很难为文件分配连续磁盘空间。这是非常致命的缺点，会极大影响文件系统的 IO 吞吐率。
- 最近释放的盘块或 inode，会立即分配给需要它们的文件。栈底资源得不到利用。数据块使用不均衡，容易损坏扇区。

4.3.2　Windows 的栈式管理法

所有空闲盘块 组成空白文件。起始盘块为 firstFreeBLK。用 FAT 表登记下个空闲盘块。

分配空闲扇区（最直接的方法）。介绍 Windows 操作系统时详述。

```
retBLK = firstFreeBLK;
firstFreeBLK  = FAT[firstFreeBLK] ;
return retBLK;
```

4.3.3　现在标准的方式是用位示图管理空闲资源。

Windows 卷，磁盘有多少个数据块（Windows 叫做簇），位示图就有多少个 bit，1 标识已分配，0 标识空闲。位示图可以位于磁盘固定位置，如下图，超级块之后；也可以是 Windows 文件卷根目录下的一个隐藏文件。实现方式不影响设计的实质。

Unix 卷，磁盘上的存储资源有两种，inode 区的 DiskInode 和 数据区中的数据块。

- inode 区配有一个位示图，inode bitmap，简称 IB；每个 bit 管理一个 DiskInode，共有 M 个 bit，M 是 inode 区 DiskInode 的总数。
- 数据区配有一个位示图，data bitmap，简称 DB；每个 bit 管理一个数据块，为了计算、管理方便，DB 共有 N 个 bit，N 是磁盘数据块的总数。

图 19

用位示图管理磁盘存储资源，优势是明显的。系统可以扫描位示图，寻找连续的多个 0 bit，为文件分配连续磁盘数据块 或者 为属于同一个目录的多个文件分配连续的 DiskInode；这会提升文件系统访问效率。付出的代价是，需要在磁盘上单独辟出一块空间用来存放位示图，为每个数据块分配 1 个 bit。有一定的开销，但微乎其微。

5、文件系统格式化

这是 Unix V6 文件卷。它不是启动盘[注]。

图 20

[注]启动盘（主硬盘）上还要包括引导扇区，内核映像 和交换区。现代系统，内核映像是根目录下的一个普通文件。系统启动时，引导程序（引导扇区中装的代码）目录搜索找到内核映像文件，将其复制进内存。

图 21

Unix V6++内核映像不是文件系统的一个文件。它存放在硬盘固定位置。Unix V6++的主硬盘 c.img 如下格式。扇区 200 ~ s_fsize-1 是文件区。

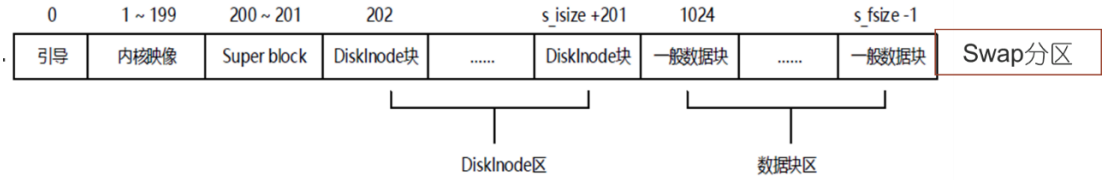


图 22

下面我们格式化图 20 中，Unix V6++的主硬盘 c.img。假设，格式化成功后磁盘上有一棵最简 Unix 文件树，如图。盘里有 7 个文件：5 个目录，1 个文件 shell 程序（8 个逻辑块） 和 1 个终端 tty1。

注意，0# inode 不用，数字 0 用来标识空目录项。

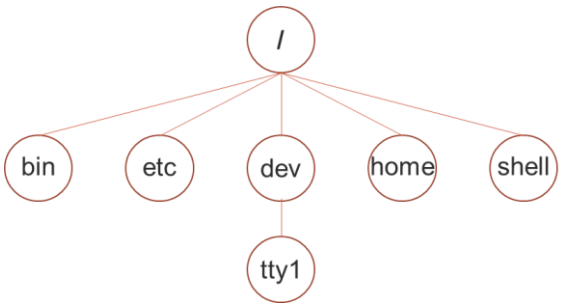


图 23

总结：SuperBlock、DiskInode 和 目录项 Directory 是 Unix V6++文件系统记录在磁盘上的

数据结构，是文件系统的元数据（MetaData）。

- SuperBlock 记录存储磁盘资源的使用情况；
- 存放在目录文件里的所有目录项 Directory，构成一棵文件树，是文件系统实现按名查找的关键数据结构；（文件搜索、创建和删除；子目录搜索、创建和删除）
- DiskInode 管理一个具体文件，是文件控制块。索引块是 DiskInode 的辅助数据结构；对于大文件，索引块用来记录逻辑块和数据块的地址映射关系。