第七章　文件管理

计算机中大量的文件与数据，存放在不同的存储介质中，用户和系统在使用时需要频繁地对它们进行访问。为此，在操作系统中引入并建立了文件管理系统，以完成外存上的大量文件信息的管理。

本章主要讲述文件与文件系统、文件的组织和存取，以及文件的保护。

本章主要内容

7.1.1　文件的概念

文件：具有名字的一组相关元素的集合。文件分无结构和结构：域(基本数据项）、记录（数据项组合）、数据库 。文件具有属性。

7.1.2　文件的分类

普通文件、特殊文件、目录文件

7.1 文件和文件系统

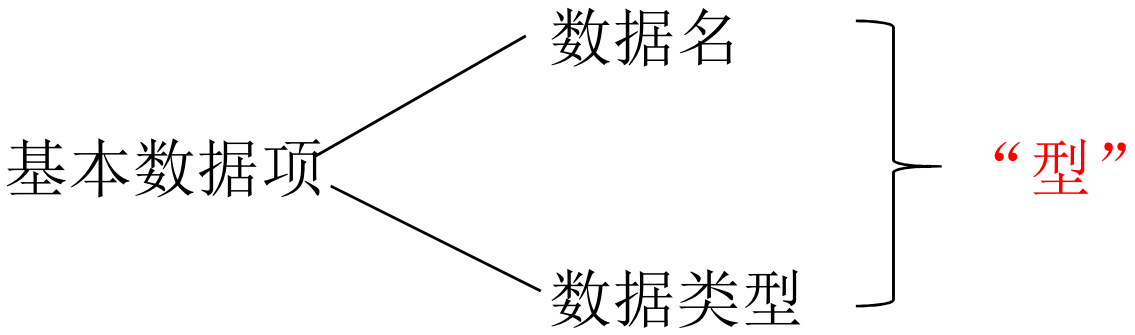
7.1.1 文件基本概念

1. 数据项（域）

在文件系统中，数据项是最低级的数据组织形式，可把它分成以下两种类型:

(1) 基本数据项。这是用于描述一个对象的某种属性的字符集，是数据组织中可以命名的最小逻辑数据单位，又称为数据元素或字段。它的命名往往与其属性一致。例如，用于描述一个学生的基本数据项有学号、姓名、年龄、所在班级等。

（2）组合数据项。它是由若干个基本数据项组成的，简称组项。例如，经理、工资



例如，学号：应使用整数；姓名：使用字符串

性别：可用逻辑变量或汉字。

而表征一个实体在数据项上的数据则称为“值”。

例如，学号/30211、姓名/王有年、性别/男等。

2．记录

记录是一组相关数据项的集合，用于描述一个对象在某方面的属性。

例如，一个学生，当把他作为班上的一名学生时，应怎样描述？

若把学生作为一个运动员时，应怎样描述？

作为一个商场顾客，应怎样描述？

在诸多记录中能惟一地标识一个记录的一个或几个数据项，把它们的集合称为关键字(key)。或者说，关键字是惟一能标识一个记录的数据项。

通常，关键字可以使一个数据项或者几个数据项。

3．文件

文件是指由创建者所定义的、具有文件名的一组相关元素的集合，可分为有结构文件(若干个相关记录组成)和无结构文件(一个字符流）两种。

文件在文件系统中是一个最大的数据单位，它描述了一个对象集。

例如，可以将一个班的学生记录作为一个文件。一个文件必须要有一个文件名，它通常是由一串ASCII码或(和)汉字构成的，名字的长度因系统不同而异。

Dos系统：8.3个字符　　windows系统：255个字符

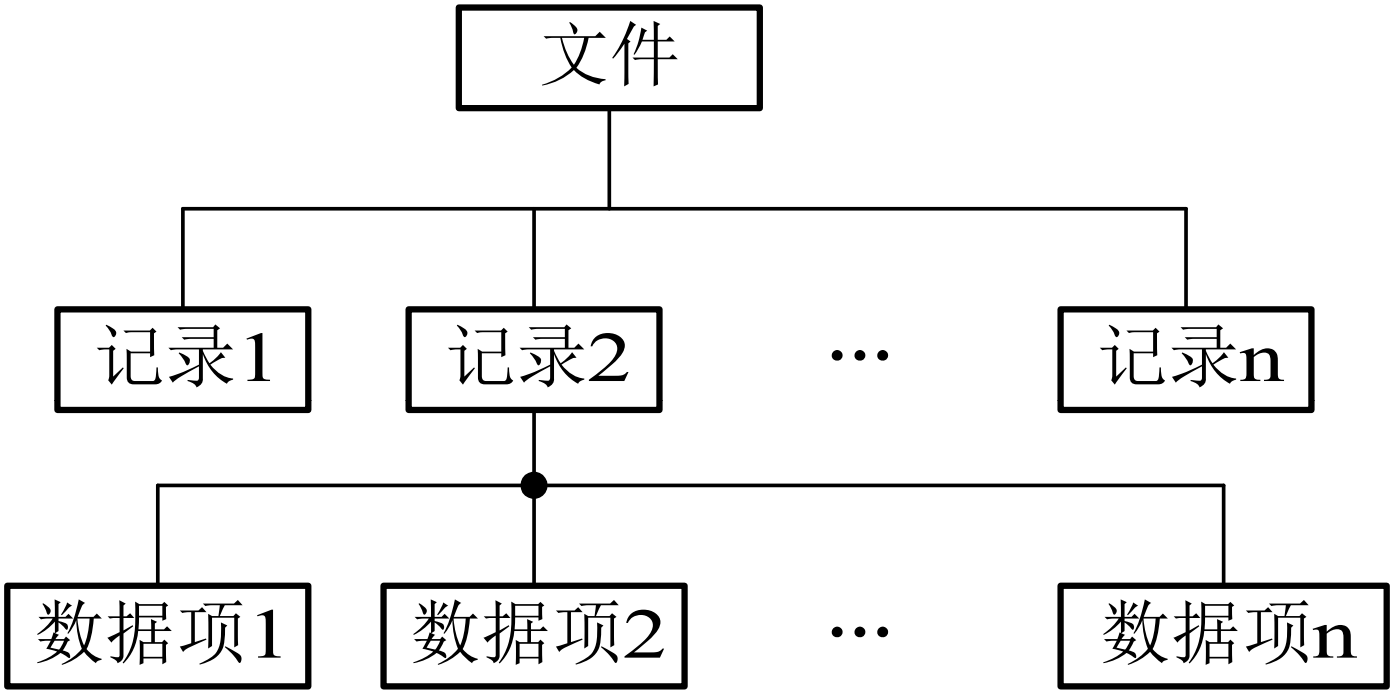
此外，文件应具有自己的属性，属性可以包括：

（1）文件类型。可以从不同的角度来规定文件的类型，如源文件、目标文件及可执行文件等。

（2）文件长度。文件长度指文件的当前长度，长度的单位可以是字节、字或块，也可能是最大允许的长度。

（3）文件的物理位置。该项属性通常是用于指示文件在哪一个设备上及在该设备的哪个位置的指针。

（4）文件的建立时间。这是指文件最后一次的修改时间等。



文件、记录和数据项之间的层次关系

7.1.2　文件类型

为了便于管理和控制文件而将文件分成若干种类型。由于不同系统对文件的管理方式不同，因而它们对文件的分类方法也有很大差异。

为了方便系统和用户了解文件的类型，在许多OS中都把文件类型作为扩展名而缀在文件名的后面，在文件名和扩展名之间用“.”号隔开。（Linux通过读取文件头来判断文件类型）

1. 按用途分类

根据文件的性质和用途的不同，可将文件分为三类：

(1) 系统文件。这是指由系统软件构成的文件。大多数的系统文件只允许用户调用，但不允许用户去读，更不允许修改；有的系统文件不直接对用户开放。

(2) 用户文件。指由用户的源代码、目标文件、可执行文件或数据等所构成的文件。用户将这些文件委托给系统保管。

(3) 库文件。这是由标准子例程及常用的例程等所构成的文件。这类文件允许用户调用，但不允许修改。

2. 按文件中数据的形式分类

按这种方式分类，也可把文件分为三类：

（1）源文件。这是指由源程序和数据构成的文件。通常由终端或输入设备输入的源程序和数据所形成的文件都属于源文件。它通常是由ASCII码或汉字所组成的。

（2）目标文件。这是指把源程序经过相应语言的编译程序编译过，但尚未经过链接程序链接的目标代码所构成的文件。它属于二进制文件。通常，目标文件所使用的后缀名是“.obj”。

（3）可执行文件。这是指把编译后所产生的目标代码再经过链接程序链接后所形成的文件。

3. 按存取控制属性分类

根据系统管理员或用户所规定的存取控制属性，可将文件分为三类：

(1) 只执行文件。该类文件只允许被核准的用户调用执行，既不允许读，更不允许写。

(2) 只读文件。该类文件只允许文件主及被核准的用户去读，但不允许写。

(3) 读写文件。这是指允许文件主和被核准的用户去读或写的文件。

4. 按组织形式和处理方式分类

根据文件的组织形式和系统对其的处理方式，可将文件分为三类：

(1)普通文件：由ASCII码或二进制码组成的字符文件。一般用户建立的源程序文件、数据文件、目标代码文件及操作系统自身代码文件、库文件、实用程序文件等都是普通文件，它们通常存储在外存储设备上。

(2) 目录文件：由文件目录组成的，用来管理和实现文件系统功能的系统文件，通过目录文件可以对其它文件的信息进行检索。

（3）特殊文件：特指系统中的各类I/O设备。为了便于统一管理，系统将所有的输入/输出设备都视为文件，按文件方式提供给用户使用，如目录的检索、权限的验证等都与普通文件相似，只是对这些文件的操作是和设备驱动程序紧密相连的，系统将这些操作转为对具体设备的操作。

7.1.3 文件系统管理

文件管理系统是用户或应用程序访问文件的唯一方式，从系统角度看，文件管理系统负责对文件存储器的存储空间进行组织、分配、存储文件，并对存入的文件进行保护检索。

1.文件系统功能

（1）文件存储空间的管理。

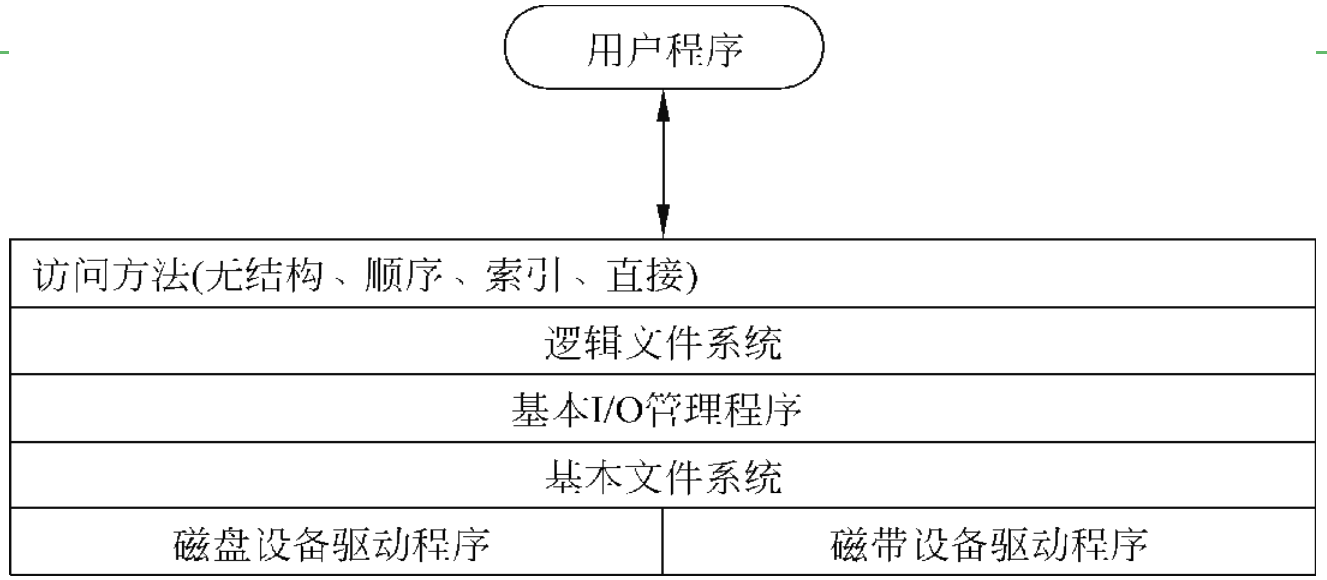
（2）实现文件名到物理地址的映射。（按名存取）

（3）实施对文件的操作：建立、删除、读写和目录操作。

（4）实现文件的共享和提供文件保护功能。

（5）提供操作文件的接口。

2. 文件系统软件结构



- 设备驱动程序直接与外围设备通信。
- 基本文件系统（物理I/O层）与计算机系统外部环境的基本接口，负责处理与磁盘交换的数据块。
- 基本I/O管理程序负责所有文件I/O的开始和终止（选择执行I/O的设备，参与调度访问磁盘，指定I/O缓冲区和外存分配）
- 逻辑文件系统使用户和应用程序能够访问到记录。（处理文件记录）
- 不同的逻辑结构对应着不同的访问方法。

3. 文件操作

最基本的文件操作

（1）创建文件。在创建一个新文件时，系统首先要为新文件分配必要的外存空间，并在文件系统的目录中，为之建立一个目录项。目录项中应记录新文件的文件名及其在外存的地址等属性。

(2) 删除文件。当已不再需要某文件时，可将它从文件系统中删除。在删除时，系统应先从目录中找到要删除文件的目录项，使之成为空项，然后回收该文件所占用的存储空间。

（3）读文件。在读一个文件时，须在相应系统调用中给出文件名和应读入的内存目标地址。此时，系统同样要查找目录，找到指定的目录项，从中得到被读文件在外存中的位置。在目录项中，还有一个指针用于对文件的读/写。

（4）写文件。在写一个文件时，须在相应系统调用中给出该文件名及该文件在内存中的(源)地址。为此，也同样须先查找目录，找到指定文件的目录项，再利用目录中的写指针进行写操作。

(5) 截断文件。如果文件名及其属性均无改变时，则可采取另一种所谓的截断文件的方法，此即将原有文件的长度设置为0，或者说是放弃原有的文件内容。

(6) 设置文件的读/写位置。用于设置文件读/写指针的位置，以便每次读/写文件时，不是从其始端而是从所设置的位置开始操作。也正因如此，才能改顺序存取为随机存取。

文件的“打开”和“关闭”操作

第一步是通过检索文件目录来找到指定文件的属性及其在外存上的位置；

第二步是对文件实施相应的操作，如读文件或写文件等。

当用户要求对一个文件实施多次读/写或其它操作时，每次都要从检索目录开始。为了避免多次重复地检索目录，在大多数OS中都引入了“打开”(open)这一文件系统调用，当用户第一次请求对某文件进行操作时，先利用open系统调用将该文件打开。

“打开”，是指系统将指名文件的属性(包括该文件在外存上的物理位置)从外存拷贝到内存打开文件表的一个表目中，并将该表目的编号(或称为索引)返回给用户。

以后，当用户再要求对该文件进行相应的操作时，便可用索引号向系统提出操作请求。系统便直接利用该索引号到打开文件表中去查找，从而避免了对该文件的再次检索。

如果用户已不再需要对该文件实施相应的操作时，可利用“关闭”(close)系统调用来关闭此文件，OS将会把该文件从打开文件表中的表目上删除掉。

其它文件操作

为了方便用户使用文件，通常，OS都提供了数条有关文件操作的系统调用，可将这些调用分成若干类:

对文件属性进行操作

改变已存文件的文件名

改变文件的拥有者(文件主)

改变对文件的访问权

查询文件的状态(包括文件类型、大小和拥有者以及对文件的访问权等)

对目录操作

创建一个目录

删除一个目录

改变当前目录和工作目录等

实现文件共享的系统调用

对文件系统进行操作的系统调用等

4. 文件的存取方法

(1)顺序存取。从头按顺序读取全部内容

(2)随机存取。按关键字存取

文件的存取方法与文件逻辑结构、物理结构及存储介质相关。文本文件源程序顺序存取，记录文件都可。

7.2 文件的逻辑结构

文件中记录的组织方式被称为文件的逻辑结构，文件的逻辑结构是从用户观点出发所看到的文件组织形式，是用户可见的并可以直接处理的数据及其结构，也被称为文件组织。

原则：

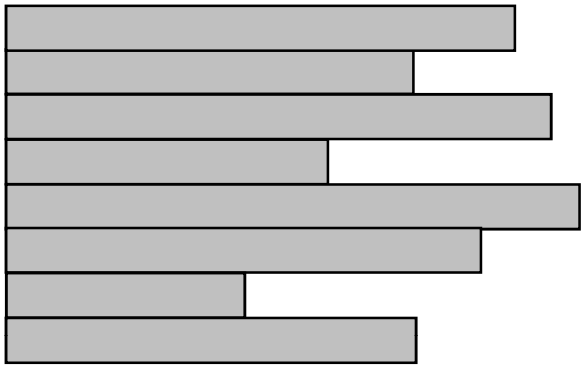
速度、维护、空间、保护

两大类：

无结构的字符流、有结构的记录型

7.2.1　无结构文件（堆文件）

以字节为单位，管理简单，节省空间。通过穷举搜索方法查找数据，访问速率慢。



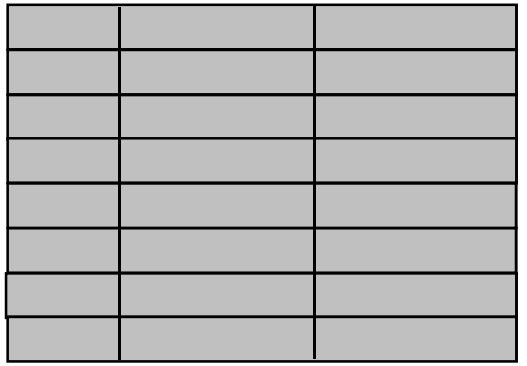
(a) 无结构文件

7.2.2 顺序文件

按记录的输入顺序排列；按某种关键字排列。利用有效的查找算法提高查找速度。

优点：适合批量存取，可存放在磁带上。定长记录可直接存取。

缺点：增加、删除记录困难。增加一个运行事务文件。



(b) 定长顺序文件

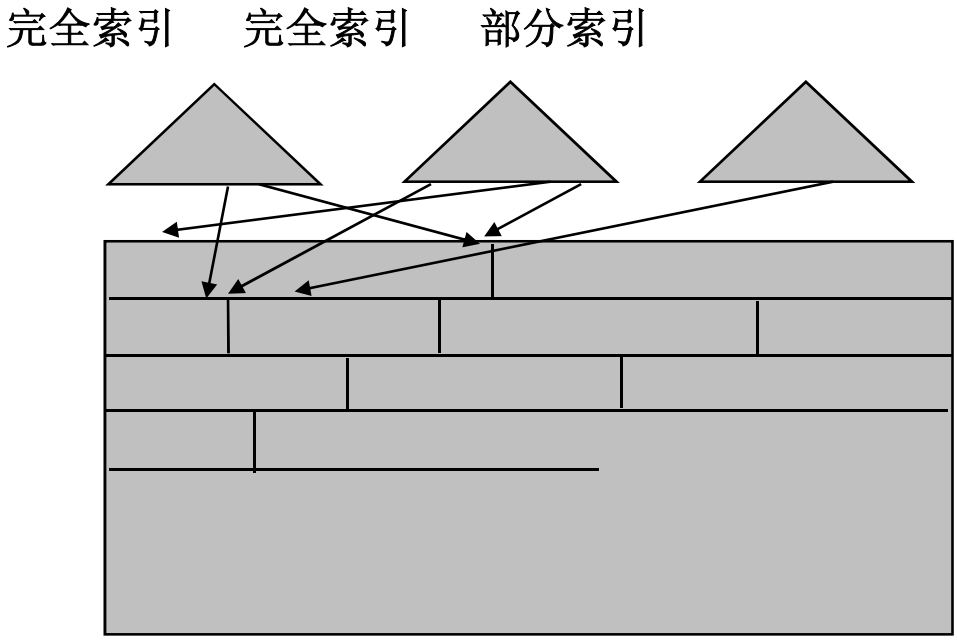
7.2.3 索引文件

为文件建立一个按记录键排序的索引文件，记录每条记录的长度和指向该记录的指针。顺序和直接存取都可。

索引分为完全索引，部分索引（分组索引）。

优点：查找快，适合很少对所有数据进行处理的应用中。

缺点：索引占用存储空间，修改文件，修改索引文件

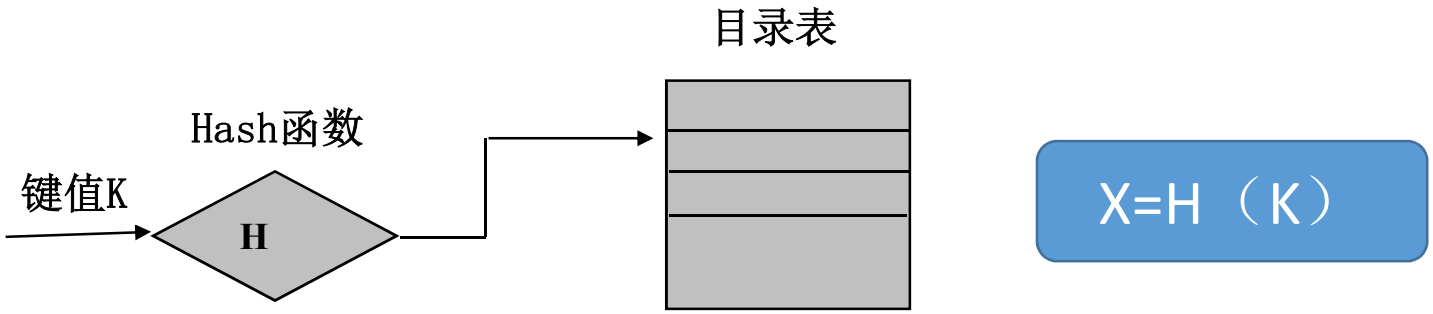


适用于对数据及时性要求比较高且很少对所有数据进行处理的应用

(c) 索引文件

7.2.4　直接文件

由Hash函数所求得的记录地址指向一个目录表的表目。该表目内容直接指向记录对应的物理地址。



（d）Hash文件的逻辑结构

7.3 文件的物理结构

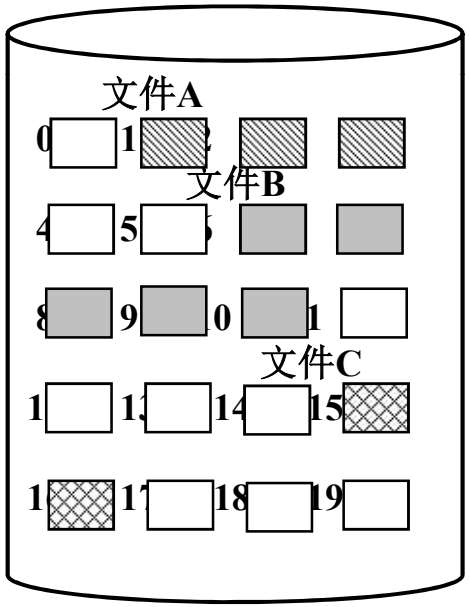
无论哪种文件组织方式，都是先搜索到记录或信息的逻辑地址，再映射到相对应的物理地址，最后对物理地址的有关信息来操作。那么逻辑地址是怎样映射到物理地址?

跟文件的物理结构密切相关，不同的外村分配方式形成不同的物理结构。

7.3.1 连续文件

文件占用连续的盘块

优缺点：对文件存取速度最快，空间限制，需知道文件的长度，存在外存碎片，需用紧缩方法。不适合用来存放经常被修改的文件。



文件目录

文件名	起始块	长度
文件A	1	3
文件B	6	5
文件C	15	2

连续文件分配

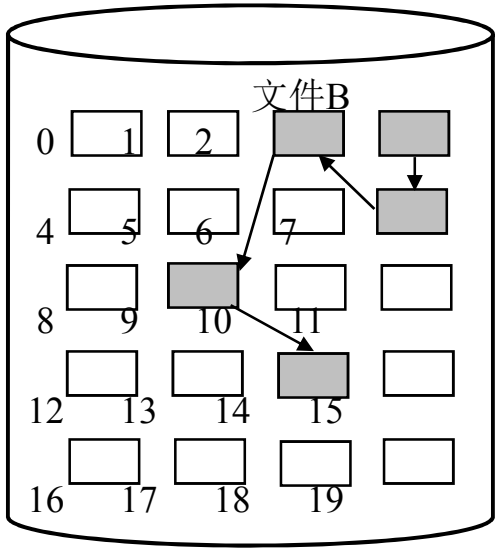
7.3.2 链接式文件

1. 隐式链接：起始块号和结束块号。簇的分配方式可改变检索速度。

优缺点：存取速度较慢、不适合直接存取、空间不连续

例7-1：最后读取号500，读取200号块，需读200块。

读的是20号块，读取21号块，只需读一块。



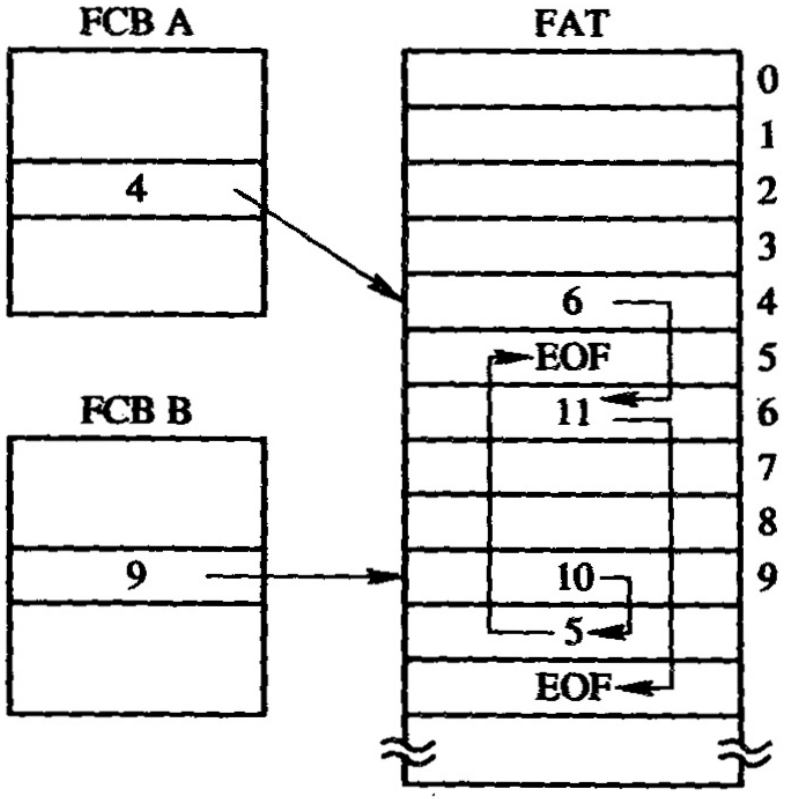
文件目录

文件名	起始块	结束块
文件B	3	15

隐式链接文件分配

2.显式链接

链接文件的各物理指针，显式的存放在内存中一张表中（FAT）。每个表项中存放一个盘块号，即下一块的盘块号。优缺点：对文件的查找在内存中，减少了访问磁盘的次数。FAT占用内存。

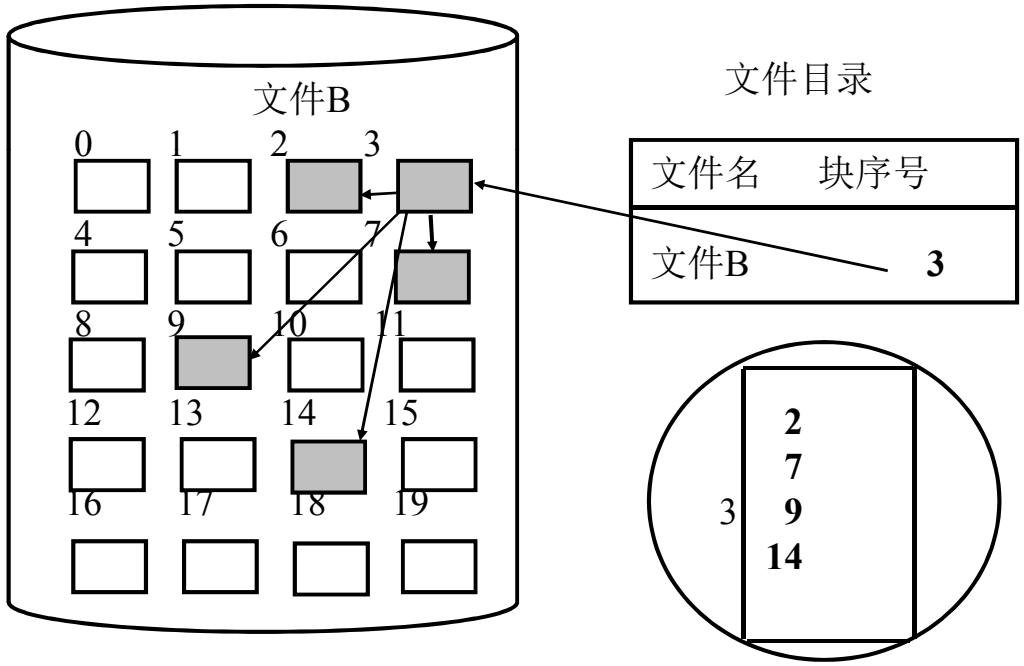


显式链接文件分配

7.3.3 索引文件

1. 单级索引分配

优缺点：适合顺序或直接存取，索引占用外存，需访问两次存储器，小文件不合适 。



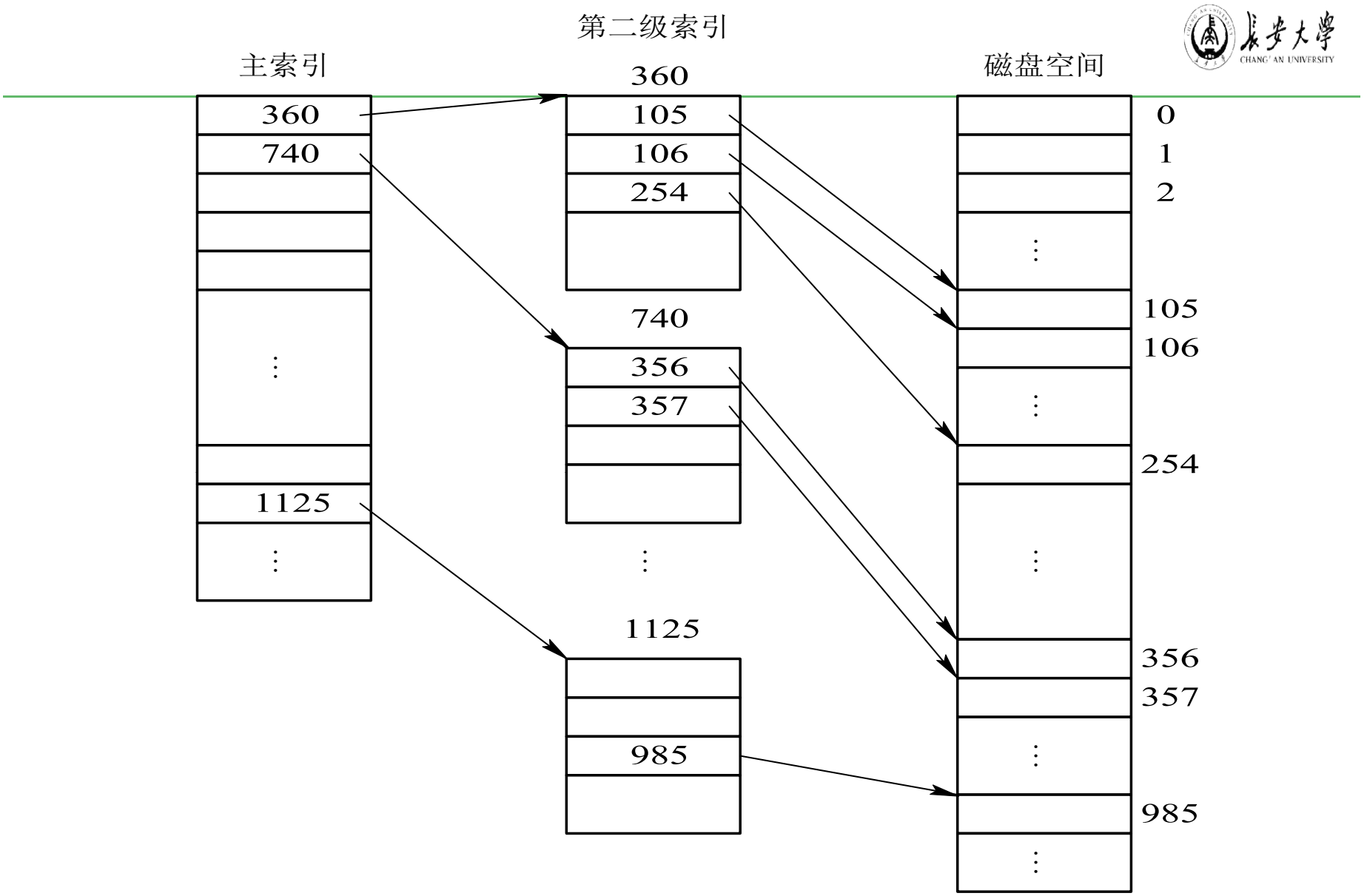
单级索引分配方式

2．多级索引分配

当OS为一个大文件分配磁盘空间时，如果所分配出去的盘块的盘块号已经装满一个索引块时，OS便为该文件分配另一个索引块，用于将以后继续为之分配的盘块号记录于其中。依此类推，再通过链指针将各索引块按序链接起来。显然，当文件太大，其索引块太多时，这种方法是低效的。

此时，应为这些索引块再建立一级索引，称为第一级索引，即系统再分配一个索引块，作为第一级索引的索引块，将第一块、第二块……等索引块的盘块号填入到此索引表中，这样便形成了两级索引分配方式。

如果文件非常大时，还可用三级、四级索引分配方式。


主索引
第二级索引
磁盘空间
360
740
1125
360
105
106
254
740
356
357
1125
985
0
1
2
105
106
254
356
357
985

两级索引分配

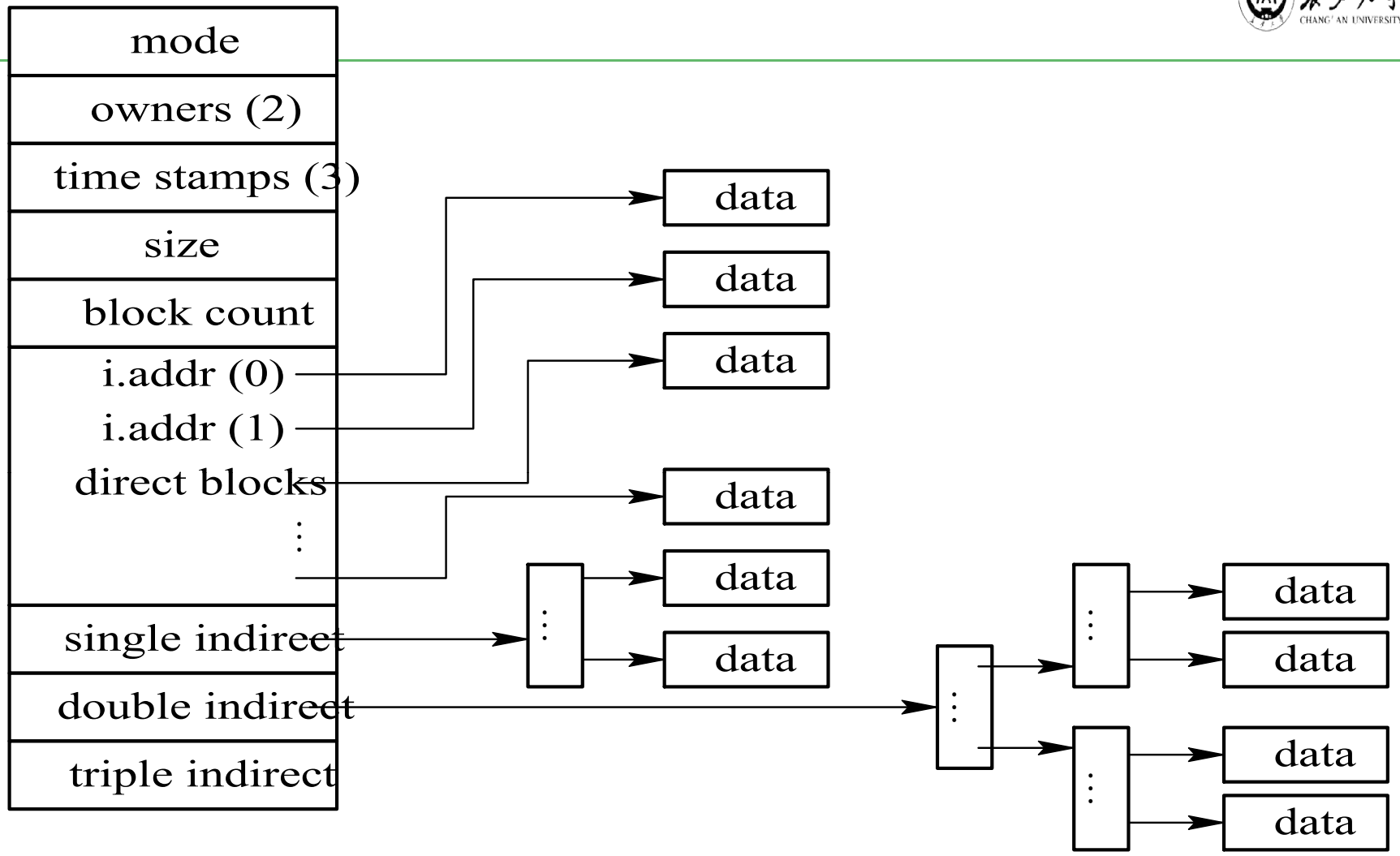
如果每个盘块的大小为1 KB，每个盘块号占4个字节，则在一个索引块中可存放256个盘块号。这样，在两级索引时， 最多可包含的存放文件的盘块的盘块号总数N = 256 × 256 = 64 K个盘块号。由此可得出结论: 采用两级索引时，所允许的文件最大长度为64 MB。倘若盘块的大小为4 KB，在采用单级索引时所允许的最大文件长度为4 MB；而在采用两级索引时所允许的最大文件长度可达4 GB。

3．混合索引分配方式

所谓混合索引分配方式，是指将多种索引分配方式相结合而形成的一种分配方式。例如，系统既采用了直接地址，又采用了一级索引分配方式，或两级索引分配方式，甚至还采用了三级索引分配方式。这种混合索引分配方式已在UNIX系统中采用。在UNIX System Ⅴ的索引结点中，共设置了13个地址项，即iaddr(0)～iaddr(12)，如图7-8所示。在BSD UNIX的索引结点中，共设置了13个地址项，它们都把所有的地址项分成两类，即直接地址和间接地址。

1）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。换言之，在这里的每项中所存放的是该文件数据所在盘块的盘块号。假如每个盘块的大小为 4 KB，当文件不大于40 KB时，便可直接从索引结点中读出该文件的全部盘块号。


mode
owners (2)
time stamps (3)
size
block count
i.addr (0)
i.addr (1)
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data

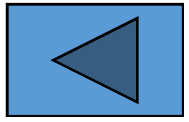
混合索引方式

2）一次间接地址

对于大、中型文件，只采用直接地址是不现实的。为此，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。图中的一次间址块也就是索引块，在一次间址块中可存放1 K个盘块号，因而允许文件长达4 MB。

3）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。系统此时是在二次间址块中记入所有一次间址块的盘号。在采用二次间址方式时，文件最大长度可达4 GB。同理，地址项iaddr(12)作为三次间接地址，其所允许的文件最大长度可达4 TB。



7.4 文件存储空间的管理

7.4.1 空闲表法和空闲链表法

1. 空闲表法

1）空闲表

空闲表法属于连续分配方式，即系统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空闲表项，其中包括表项序号、该空闲区的第一个盘块号、该区的空闲盘块数等信息。再将所有空闲区按其起始盘块号递增的次序排列，如图7-9所示。

序　　号	第一空闲盘块号	空闲盘块数
1	2	4
2	9	3
3	15	5
4	—	—

图7-9　空闲盘块表

2）存储空间的分配与回收

空闲盘区的分配与内存的动态分配类似，同样是采用首次适应算法、循环首次适应算法等。例如，在系统为某新创建的文件分配空闲盘块时，先顺序地检索空闲表的各表项，直至找到第一个其大小能满足要求的空闲区，再将该盘区分配给用户(进程)，同时修改空闲表。系统在对用户所释放的存储空间进行回收时，也采取类似于内存回收的方法，即要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。

在内存分配上，虽然很少采用连续分配方式，然而在外存的管理中，由于这种分配方式具有较高的分配速度，可减少访问磁盘的I/O频率，故它在诸多分配方式中仍占有一席之地。

例如，在前面所介绍的对换方式中，对对换空间一般都采用连续分配方式。

对于文件系统，当文件较小(1～4个盘块)时，仍采用连续分配方式，为文件分配相邻接的几个盘块；当文件较大时，便采用离散分配方式。

2．空闲链表法

空闲链表法是将所有空闲盘区拉成一条空闲链。根据构成链所用基本元素的不同，可把链表分成两种形式：空闲盘块链和空闲盘区链。

（1）空闲盘块链。这是将磁盘上的所有空闲空间，以盘块为单位拉成一条链。当用户因创建文件而请求分配存储空间时，系统从链首开始，依次摘下适当数目的空闲盘块分配给用户。当用户因删除文件而释放存储空间时，系统将回收的盘块依次插入空闲盘块链的末尾。

缺点：分配时需要多次操作。

(2) 空闲盘区链。这是将磁盘上的所有空闲盘区(每个盘区可包含若干个盘块)拉成一条链。在每个盘区上除含有用于指示下一个空闲盘区的指针外，还应有能指明本盘区大小(盘块数)的信息。分配盘区的方法与内存的动态分区分配类似，通常采用首次适应算法。在回收盘区时，同样也要将回收区与相邻接的空闲盘区相合并。在采用首次适应算法时，为了提高对空闲盘区的检索速度，可以采用显式链接方法，亦即，在内存中为空闲盘区建立一张链表。

7.4.2　位示图法

1．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。

磁盘上的所有盘块都有一个二进制位与之对应，这样，由所有盘块所对应的位构成一个集合，称为位示图。通常可用m × n个位数来构成位示图，并使m × n等于磁盘的总块数，如图7-10所示。

位示图也可描述为一个二维数组map：

```
Var map: array of bit;
```

	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																
M																
16																

位示图

2．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：

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

式中，n代表每行的位数。

（3）修改位示图，令map[i,j]=1。

3．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：

$$i = (b - 1)\text{DIV } n + 1$$

$$j = (b - 1)\text{MOD } n + 1$$

(2) 修改位示图。令map[i,j] =0。

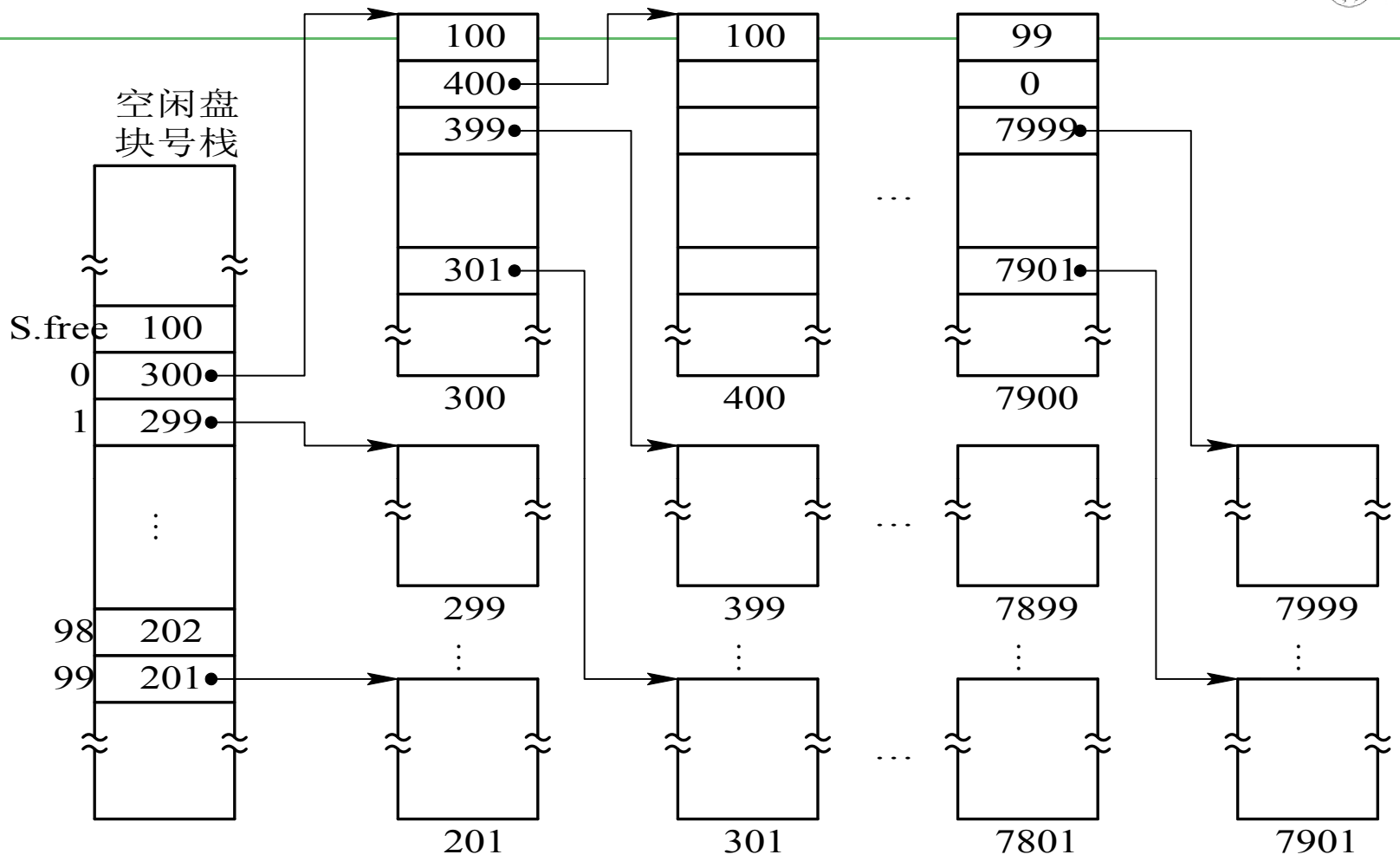
这种方法的主要优点是，从位示图中很容易找到一个或一组相邻接的空闲盘块。例如，我们需要找到6个相邻接的空闲盘块，这只需在位示图中找出6个其值连续为“0”的位即可。此外，由于位示图很小，占用空间少，因而可将它保存在内存中，进而使在每次进行盘区分配时，无需首先把盘区分配表读入内存，从而节省了许多磁盘的启动操作。因此，位示图常用于微型机和小型机中，如CP/M、Apple-DOS等OS中。

7.4.3 成组链接法

在Unix系统中采用的是成组链接法。结合了空闲表法和空闲链表法的特点。

1．空闲盘块的组织

(1) 空闲盘块号栈用来存放当前可用的一组空闲盘块的盘块号(最多含100个号)，以及栈中尚有的空闲盘块号数N。顺便指出，N还兼作栈顶指针用。例如，当N=100时，它指向S.free(99)。由于栈是临界资源，每次只允许一个进程去访问，故系统为栈设置了一把锁。图6-23左部示出了空闲盘块号栈的结构。其中，S.free(0)是栈底，栈满时的栈顶为S.free(99)。


空闲盘
块号栈
S.free 100
0 300
1 299
98 202
99 201
100
400
399
301
99
0
7999
7901
300
400
7900
299
399
7899
7999
201
301
7801
7901

空闲盘块的成组链接法

(2) 文件区中的所有空闲盘块被分成若干个组，比如，将每100个盘块作为一组。假定盘上共有10 000个盘块，每块大小为1 KB，其中第201～7999号盘块用于存放文件，即作为文件区，这样，该区的最末一组盘块号应为7901～7999；次末组为7801～7900.....；第二组的盘块号为301～400；第一组为201～300。

(3) 将每一组含有的盘块总数N和该组所有的盘块号记入其前一组的第一个盘块的S.free(0)～S.free(99)中。这样，由各组的第一个盘块可链成一条链。

(4) 将第一组的盘块总数和所有的盘块号记入空闲盘块号栈中，作为当前可供分配的空闲盘块号。

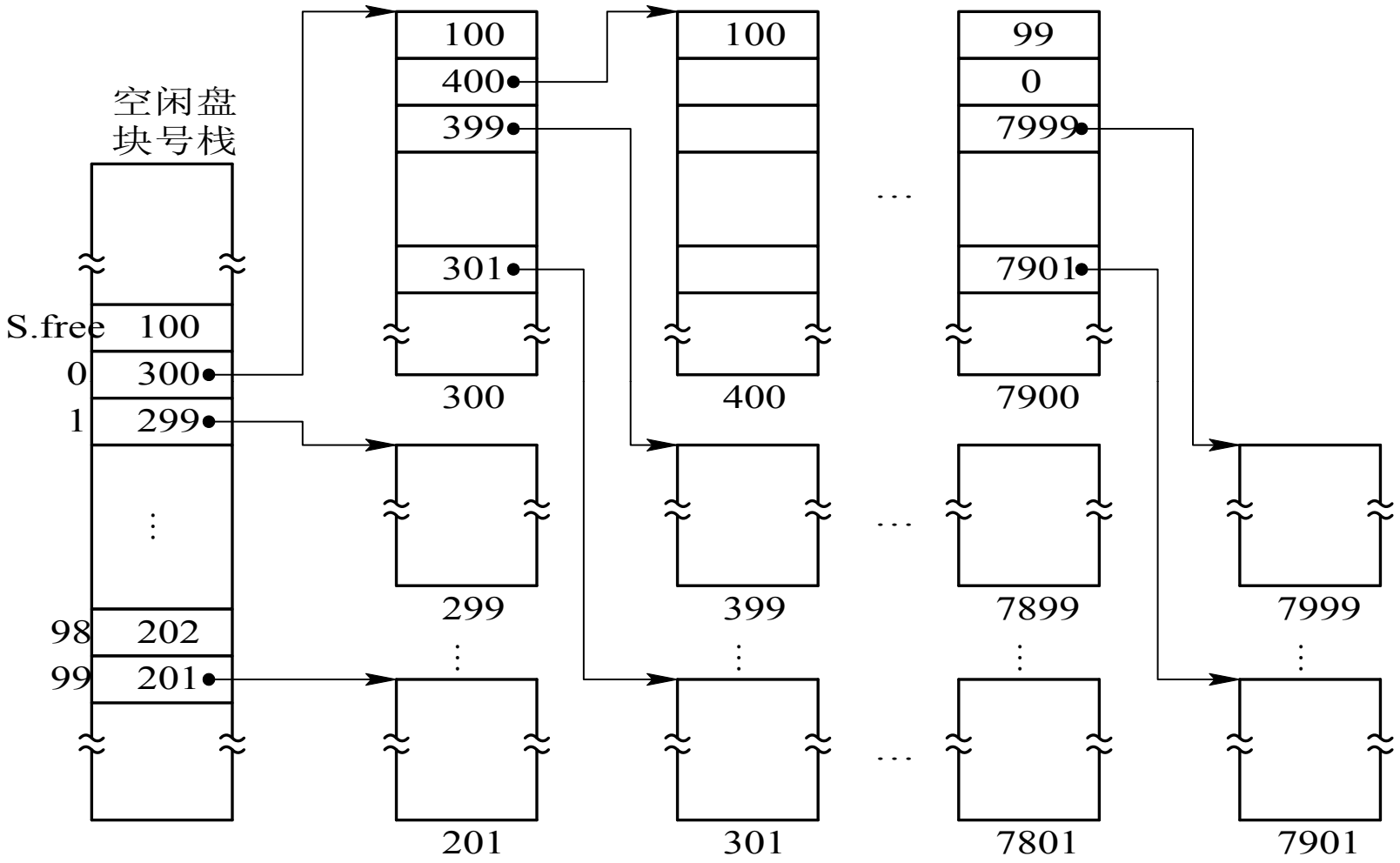
(5) 最末一组只有99个盘块，其盘块号分别记入其前一组的S.free(1) ～S.free(99)中，而在S.free(0)中则存放 “0”，作为空闲盘块链的结束标志。(注：最后一组的盘块数应为99，不应是100，因为这是指可供使用的空闲盘块，其编号应为(1～99)，0号中放空闲盘块链的结尾标志。)

2．空闲盘块的分配与回收

当系统要为用户分配文件所需的盘块时，须调用盘块分配过程来完成。

该过程首先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，然后将栈顶指针下移一格。

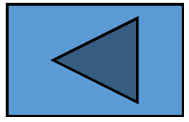
201	202	…	300	301	…	400	…	7900	7901	7902	…	7999



空闲盘块的成组链接法

若该盘块号已是栈底，即S.free(0)，这是当前栈中最后一个可分配的盘块号。由于在该盘块号所对应的盘块中记有下一组可用的盘块号，因此，须调用磁盘读过程，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(其中的有用数据已读入栈中)。然后，再分配一相应的缓冲区(作为该盘块的缓冲区)。最后，把栈中的空闲盘块数减1并返回。

在系统回收空闲盘块时，须调用盘块回收过程进行回收。它是将回收盘块的盘块号记入空闲盘块号栈的顶部，并执行空闲盘块数加1操作。当栈中空闲盘块号数目已达100时，表示栈已满，便将现有栈中的100个盘块号记入新回收的盘块中，再将其盘块号作为新栈底。



7.5.1　文件控制块（FCB）

FCB：基本信息、地址信息、访问控制信息、使用信息；

文件目录，目录文件

7.5.2　目录结构

单级结构；两级结构；

多级目录结构：根、结点、树叶。

路径名：相对、绝对、当前工作目录

多级目录层次更清楚，更有效的管理进行文件的管理和保护。逐层访问增加了磁盘的访问次数。

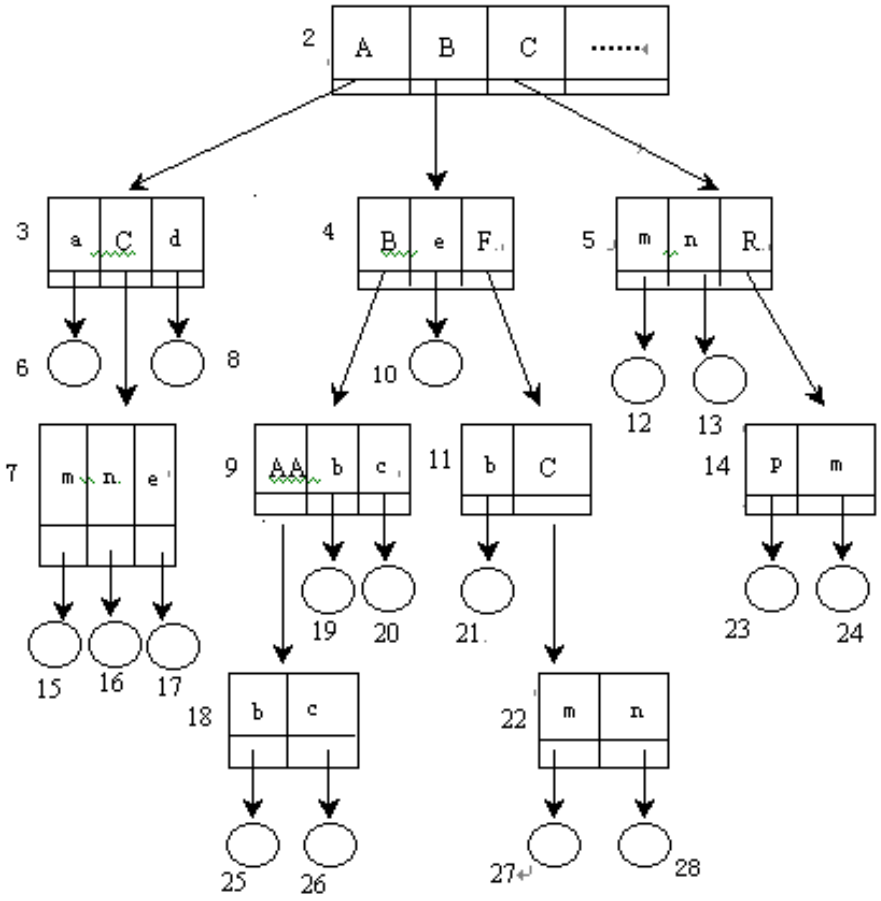


图7-15　多级目录结构

7.5 文件目录管理

通常，在现代计算机系统中，都要存储大量的文件。为了能对这些文件实施有效的管理，必须对它们加以妥善组织，这主要是通过文件目录实现的。文件目录也是一种数据结构，用于标识系统中的文件及其物理地址，供检索时使用。对目录管理的要求如下：

(1) 实现“按名存取”，这是目录管理中最基本的功能，也是文件系统向用户提供的最基本的服务。

(2) 提高对目录的检索速度。通过合理地组织目录结构的方法，可加快对目录的检索速度，从而提高对文件的存取速度。这是在设计一个大、中型文件系统时所追求的主要目标。

(3) 文件共享。在多用户系统中，应允许多个用户共享一个文件。这样就须在外存中只保留一份该文件的副本，供不同用户使用，以节省大量的存储空间。

(4) 允许文件重名。系统应允许不同用户对不同文件采用相同的名字，以便于用户按照自己的习惯给文件命名和使用文件。

7.5.1　文件控制块和索引结点

为了能对一个文件进行正确的存取必须为文件设置用于描述和控制文件的数据结构，称为文件控制块（ＦＣＢ）。一个文件控制块就是一个目录项。

1．文件控制块

为了能对系统中的大量文件施以有效的管理，在文件控制块中，通常应含有三类信息，即基本信息、存取控制信息及使用信息。

1）基本信息类

基本信息类包括:

① 文件名，在每个系统中，每一个文件都必须有惟一的名字，用户利用该名字进行存取。

② 文件物理位置，指文件在外存上的存储位置，它包括存放文件的设备名、文件在外存上的起始盘块号、指示文件所占用的盘块数或字节数的文件长度。

③ 文件逻辑结构，指示文件是流式文件还是记录式文件、记录数；文件是定长记录还是变长记录等。

④ 文件的物理结构，指示文件是顺序文件，还是链接式文件或索引文件。

2）存取控制信息类

存取控制信息类包括：文件主的存取权限、核准用户的存取权限以及一般用户的存取权限。

3）使用信息类

使用信息类包括：文件的建立日期和时间、文件上一次修改的日期和时间及当前使用信息(这项信息包括当前已打开该文件的进程数、是否被其它进程锁住、文件在内存中是否已被修改但尚未拷贝到盘上)。

下图示出了MS-DOS中的文件控制块。

文件名	扩展名	属性	备用	时间	日期	第一块号	盘块数

FCB的长度为32个字节，对于360 KB的软盘，总共可包含112个FCB，共占4 KB的存储空间。

MS-DOS的文件控制块

7.5.2　目录结构

1．单级目录结构

这是最简单的目录结构。在整个文件系统中只建立一张目录表，每个文件占一个目录项，目录项中含文件名、文件扩展名、文件长度、文件类型、文件物理地址以及其它文件属性。此外，为表明每个目录项是否空闲，又设置了一个状态位。单级目录如下图所示。

文件名	物理地址	文件说明	状态位
文件名 1			
文件名 2			
…			

单级目录

每当要建立一个新文件时，

必须先检索所有的目录项，以保证新文件名在目录中是惟一的。

然后再从目录表中找出一个空白目录项，填入新文件的文件名及其他说明信息，并置状态位为1。

删除文件时，先从目录中找到该文件的目录项，回收该文件所占用的存储空间，然后再清除该目录项。

单级目录的优点是简单且能实现目录管理的基本功能—按名存取，但却存在下述一些缺点：

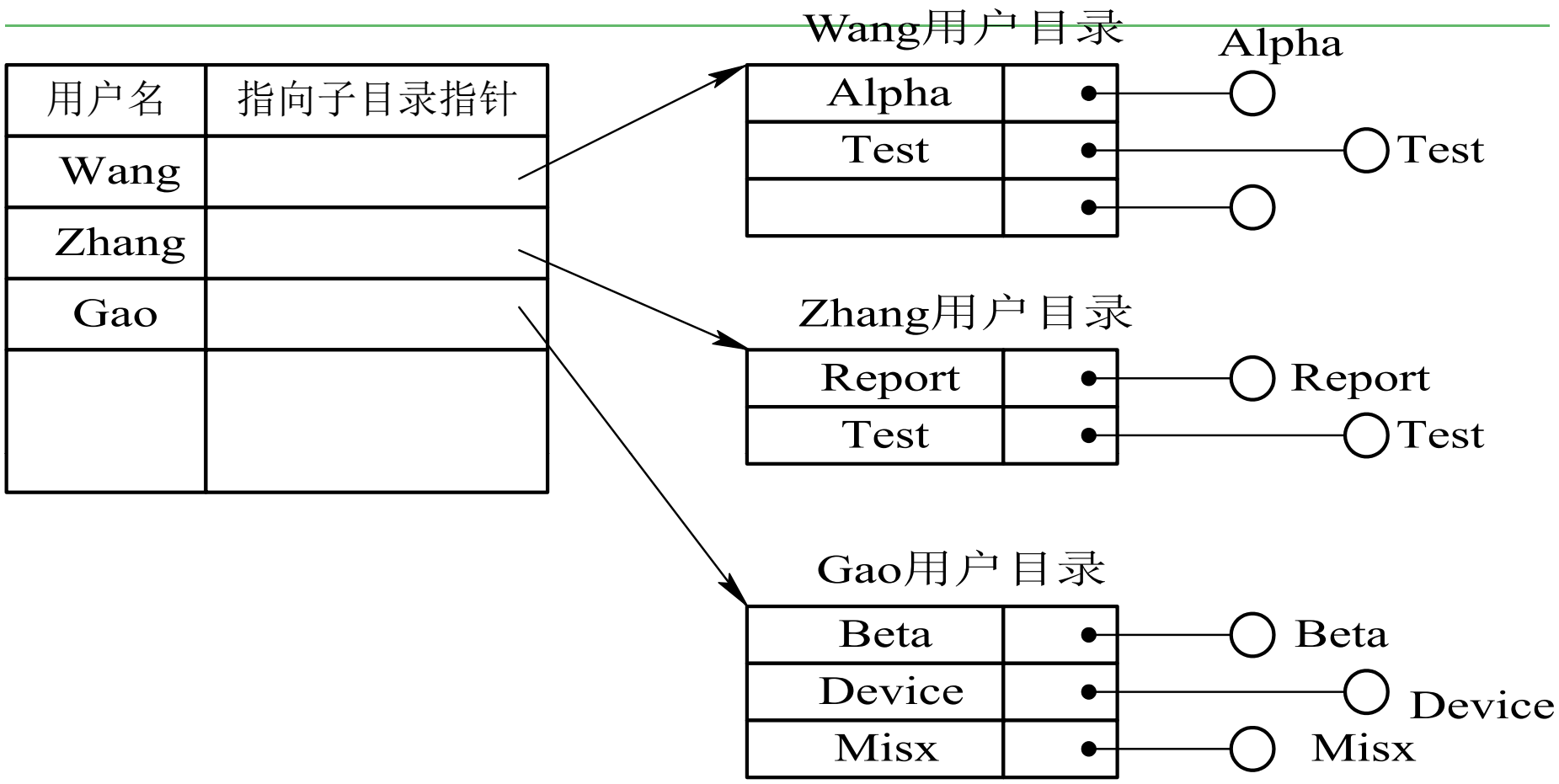
(1) 查找速度慢。对于一个具有N个目录项的单级目录，为检索出一个目录项，平均需查找N/2个目录项。

(2) 不允许重名。

(3) 不便于实现文件共享。应当允许不同用户使用不同的文件名来访问同一个文件。

2．两级目录

为了克服单级目录所存在的缺点，可以为每一个用户建立一个单独的用户文件目录UFD(User File Directory)。此外，在系统中再建立一个主文件目录MFD(Master File Directory)；在主文件目录中，每个用户目录文件都占有一个目录项，其目录项中包括用户名和指向该用户目录文件的指针。如下图所示，图中的主目录中示出了三个用户名，即Wang、Zhang和Gao。


用户名
指向子目录指针
Wang
Zhang
Gao
Wang用户目录
Alpha
Test
Alpha
Test
Zhang用户目录
Report
Test
Report
Test
Gao用户目录
Beta
Device
Misx
Beta
Device
Misx

两级目录结构

在两级目录结构中，如果用户希望有自己的用户文件目录UFD，可以请求系统为自己建立一个用户文件目录；如果自己不再需要UFD，也可以请求系统管理员将它撤消。

OS只需检查该用户的UFD，判定在该UFD中是否已有同名的另一个文件。若有，用户必须为新文件重新命名；若无，便在UFD中建立一个新目录项，将新文件名及其有关属性填入目录项中，并置其状态位为“1”。

当用户要删除一个文件时，OS也只需查找该用户的UFD，从中找出指定文件的目录项， 在回收该文件所占用的存储空间后，将该目录项删除。

两级目录结构基本上克服了单级目录的缺点，并具有以下优点：

（1）提高了检索目录的速度。如果在主目录中有n个子目录，每个用户目录最多为m个目录项，

单级目录：检索n × m个目录项；

两级目录：检索n + m个目录项。

假定n = m，可以看出，采用两级目录可使检索效率提高n/2倍。

(2) 在不同的用户目录中，可以使用相同的文件名。

(3) 不同用户还可使用不同的文件名来访问系统中的同一个共享文件。

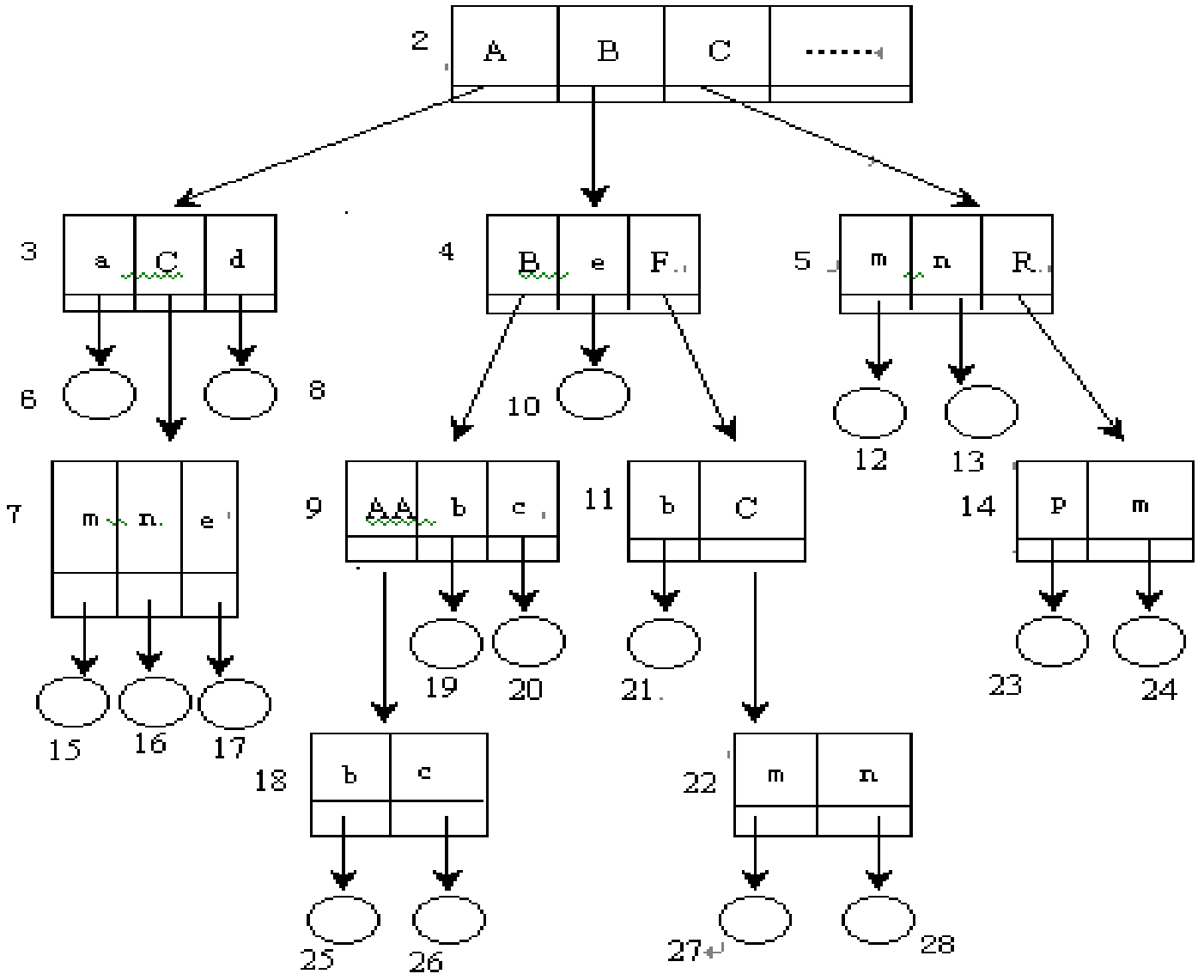
问题：

该结构虽然能有效地将多个用户隔开，当多个用户之间要相互合作去完成一个大任务，且一用户又需去访问其他用户的文件时，这种隔离便会使诸用户之间不便于共享文件。

3．多级目录结构

1） 目录结构

对于大型文件系统，通常采用三级或三级以上的目录结构，以提高对目录的检索速度和文件系统的性能。多级目录结构又称为树型目录结构，主目录在这里被称为根目录，把数据文件称为树叶，其它的目录均作为树的结点。


2
A
B
C
3
a
C
d
4
B
e
F
5
m
n
R
6
8
10
12
13
7
m
n
e
9
A
b
c
11
b
C
14
P
m
15
16
17
18
b
c
19
20
21
22
m
n
23
24
25
26
27
28

多级目录结构

2）路径名

在树形目录结构中，从根目录到任何数据文件，都只有一条惟一的通路。在该路径上从树的根(即主目录)开始，把全部目录文件名与数据文件名依次地用“/”连接起来，即构成该数据文件的路径名(path name)。系统中的每一个文件都有惟一的路径名。

3）当前目录(Current Directory)

为了访问方便，可为每个进程设置一个“当前目录”，又称为“工作目录”。进程对各文件的访问都相对于“当前目录”而进行。

这样，把从当前目录开始直到数据文件为止所构成的路径名，称为相对路径名(relative path name)；而把从树根开始的路径名称为绝对路径名(absolute path name)。

就多级目录较两级目录而言，其查询速度更快，同时层次结构更加清晰，能够更加有效地进行文件的管理和保护。

但是在多级目录中查找一个文件，需要按路径名逐级访问中间节点，这就增加了磁盘访问次数，无疑将影响查询速度。

目前，大多数操作系统如UNIX、Linux和Windows系列都采用了多级目录结构。

4．增加和删除目录

在树型目录结构中，用户可为自己建立UFD，并可再创建子目录。在用户要创建一个新文件时，只需查看在自己的UFD及其子目录中有无与新建文件相同的文件名。若无，便可在UFD或其某个子目录中增加一个新目录项。

在树型目录中，对于一个已不再需要的目录，应如何删除其目录项，须视情况而定。这时，如果所要删除的目录是空的，就可简单地将该目录项删除，如果要删除的目录不空，即其中尚有几个文件或子目录，则可采用下述两种方法处理：

(1) 不删除非空目录。MS-DOS中就是采用这种删除方式。

(2) 可删除非空目录。当要删除一个目录时，如果在该目录中还包含有文件，则目录中的所有文件和子目录也同时被删除。

上述两种方法实现起来都比较容易，第二种方法则更为方便，但比较危险。因为整个目录结构虽然用一条命令即能删除，但如果是一条错误命令，其后果则可能很严重。

7.5.3 目录管理

1. 索引结点的引入

1）索引结点的引入

文件目录通常是存放在磁盘上的，当文件很多时，文件目录可能要占用大量的盘块。

在查找目录的过程中，要将存放目录文件的盘块逐一调入内存查找指定文件。

设目录文件所占用的盘块数为N，按此方法查找，则查找一个目录项平均需要调入盘块(N+1)/2次。假如一个FCB为64 B，盘块大小为1 KB，则每个盘块中只能存放16个FCB；若一个文件目录中共有640个FCB，需占用40个盘块，故平均查找一个文件需启动磁盘20次。

可以发现，在检索目录文件的过程中，只需用到文件名。在许多系统中，便采用了把文件名与文件描述信息分开的办法，亦即，使文件描述信息单独形成一个称为索引结点的数据结构，简称为i结点。

在文件目录中的每个目录项仅由文件名和指向该文件所对应的i结点的指针所构成。在UNIX系统中一个目录仅占16个字节，其中14个字节是文件名，2个字节为i结点指针。在1 KB的盘块中可做64个目录项，这样，为找到一个文件，可使平均启动磁盘次数减少到原来的1/4，大大节省了系统开销。下图示出了UNIX的文件目录项。

文件名	索引结点编号
文件名 1	
文件名 2	
…	…

0 13 14 15

UNIX的文件目录

2）磁盘索引结点

这是存放在磁盘上的索引结点。每个文件有惟一的一个磁盘索引结点，它主要包括以下内容：

（1）文件主标识符，即拥有该文件的个人或小组的标识符。

（2）文件类型，包括正规文件、目录文件或特别文件。

（3）文件存取权限，指各类用户对该文件的存取权限。

（4）文件物理地址，每一个索引结点中含有13个地址项，即iaddr(0)～iaddr(12)，它们以直接或间接方式给出数据文件所在盘块的编号。

(5) 文件长度，指以字节为单位的文件长度。

(6) 文件连接计数，表明在本文件系统中所有指向该(文件的)文件名的指针计数。

(7) 文件存取时间，指本文件最近被进程存取的时间、最近被修改的时间及索引结点最近被修改的时间。

3）内存索引结点

当文件被打开时，要将磁盘索引结点拷贝到内存的索引结点中，在内存索引结点中又增加了以下内容：

(1) 索引结点编号，用于标识内存索引结点。

(2) 状态，指示i结点是否上锁或被修改。

(3) 访问计数，每当有一进程要访问此i结点时，将该访问计数加1，访问完再减1。

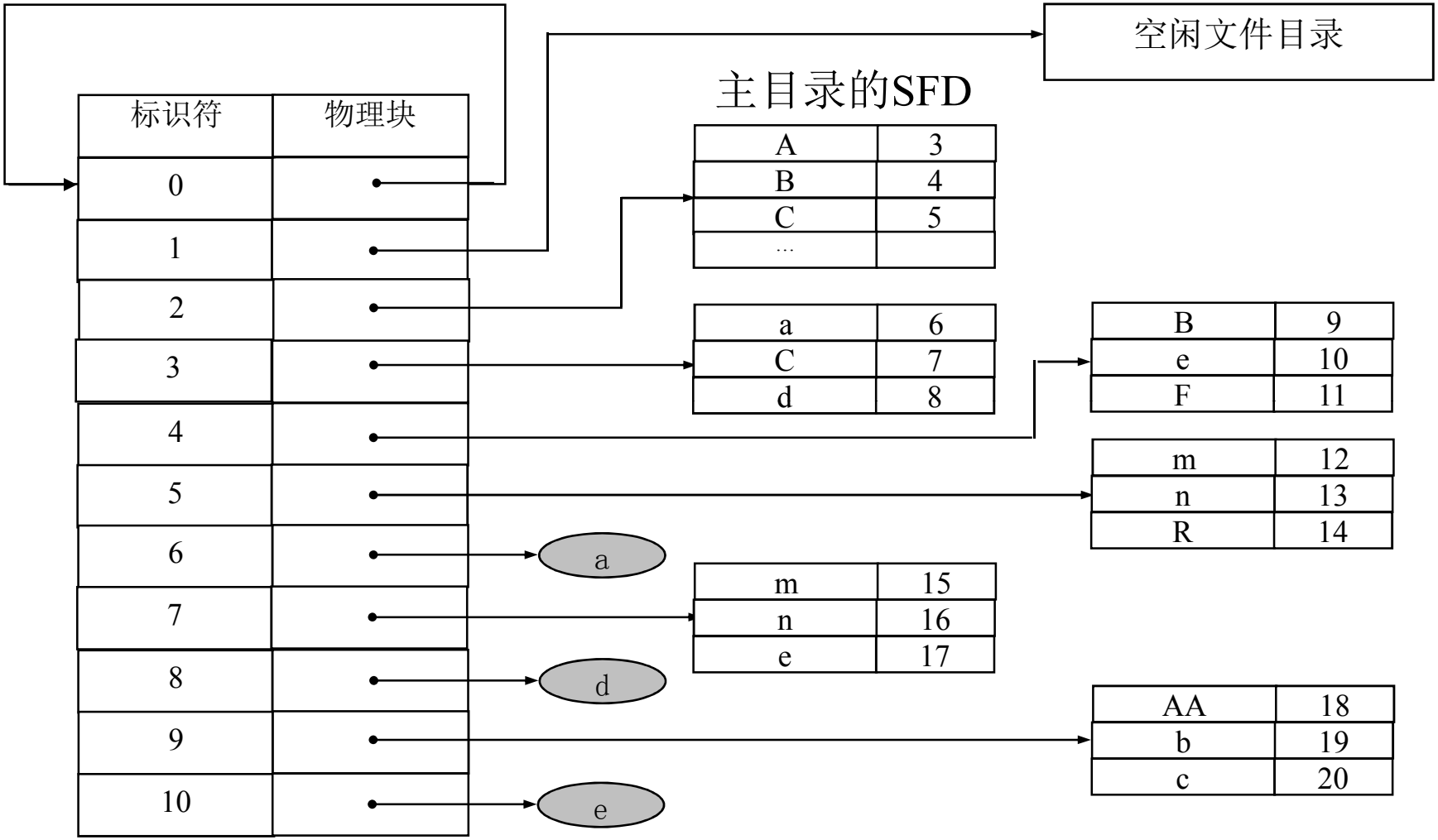
(4) 文件所属文件系统的逻辑设备号。

(5) 链接指针。设置有分别指向空闲链表和散列队列的指针。

索引结点的内容

磁盘索引结点	
文件主标识符	拥有该文件的个人或小组的标识符
文件类型	包括正规文件、目录文件和特殊文件
文件存取权限	指各类用户对文件的存取权限
文件物理地址	给出数据文件所在盘块的编号
文件长度	以字节为单位的文件长度
文件连接计数	表明在本文件系统中所有指向该（文件的）文件名的指针计数
文件存取时间	指出本文件最近被进程存取的时间、最近被修改的时间及索引结点最近被修改的时间
内存索引结点（增加的内容）	
索引结点编号	用于标识内存索引结点
状态	指示i结点是否上锁或被修改
访问计数	每当有一进程要访问此i节点时，将该访问计数加1，访问完再减1
链接指针	设置有分别指向空闲链表和散列队列的指针

符号文件目录（SFD）基本文件目录（BFD）



采用基本文件目录表的多级目录结构

2. 文件存取

当用户要打开一个文件时，系统首先根据用户给定的文件名，把主目录MFD中与待打开文件相联系的有关表目复制到内存，得到该文件对应的索引结点的指针，然后将该文件对应的索引结点拷贝到内存中成为内存索引结点，再根据索引结点中记录的文件的物理地址，换算出文件在磁盘上的物理位置；最后再通过磁盘驱动程序，将所需文件读入内存。这时文件已被打开，称这样的文件为打开的文件或**活动文件**。

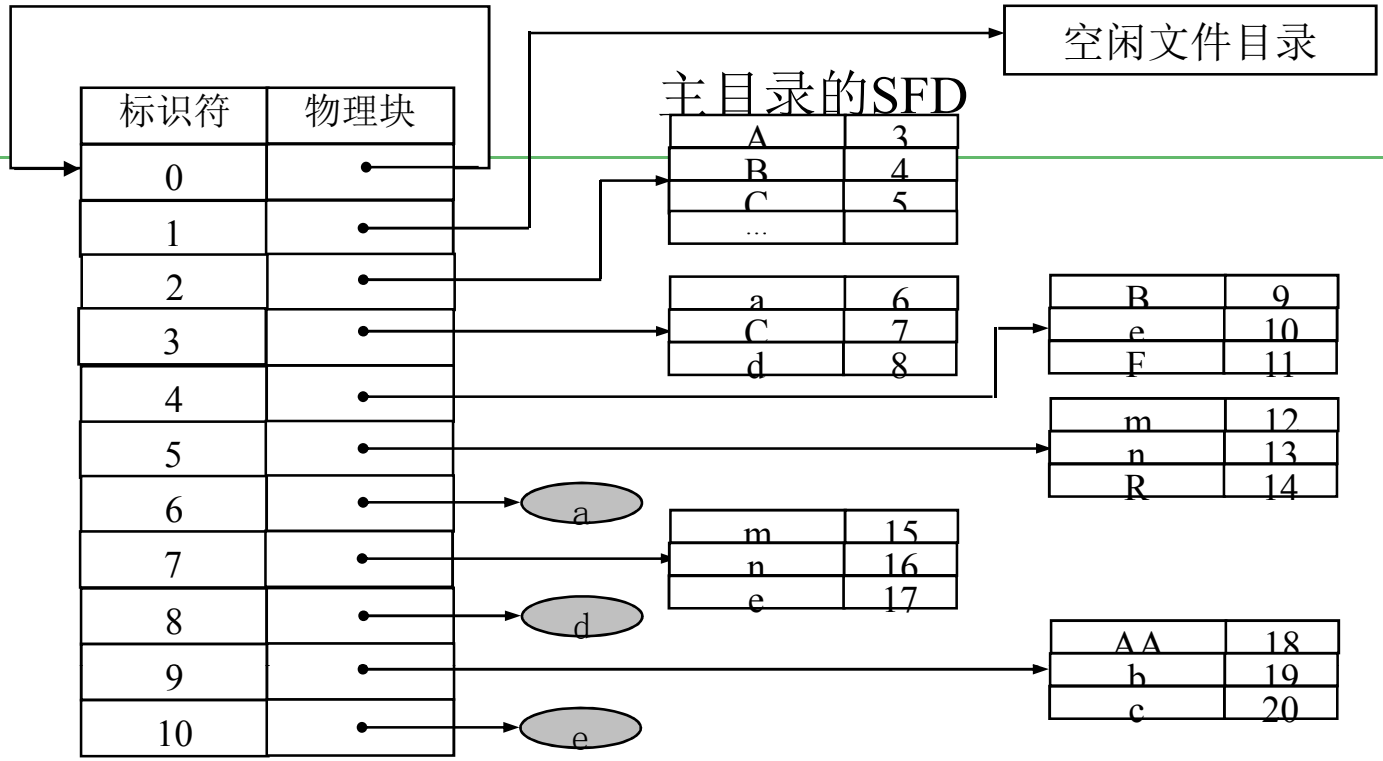
例：打开文件/A/a

用户/A/a的SFD

A	3
a	6

内存的BFD

0	…
1	…
2	…
3	…
6	…



（1）首先把主目录文件中相应的表目，A复制到内存。

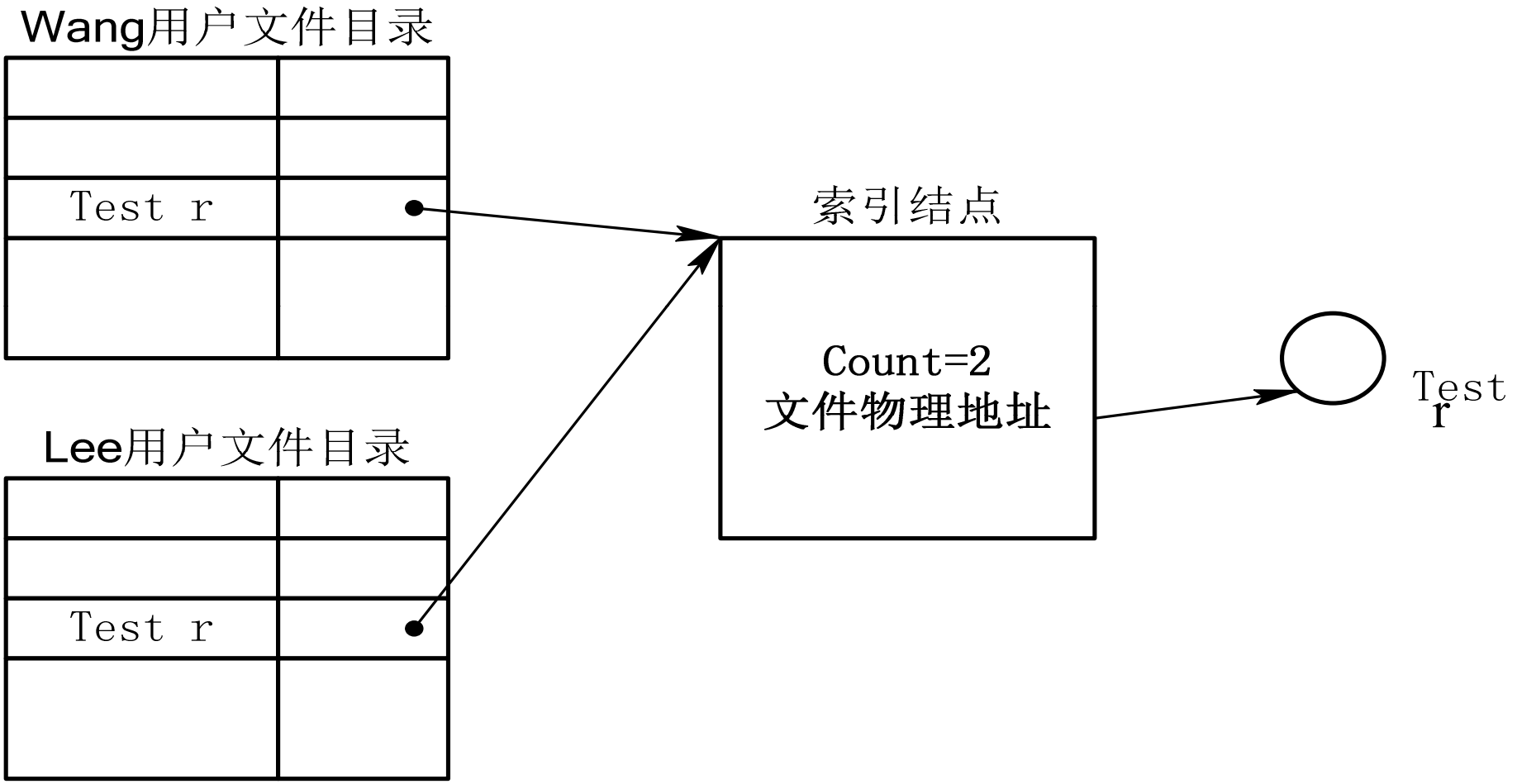
（2）根据（1）得到的标识符，再复制此标识符所指明的BFD中的有关表目，图中id=3的BFD项。

（3）根据（2）得到的子目录搜索SFD，找到与打开文件相对应的目录表项，级索引结点的指针。a对应的id=6。

（4）根据（3）所搜索到的文件名所对应的标识符id，把相应的BFD的表目即索引节点复制到内存。此例复制文件a的其他信息。

7.6 文件共享和保护

7.6.1 基于索引结点的共享方式



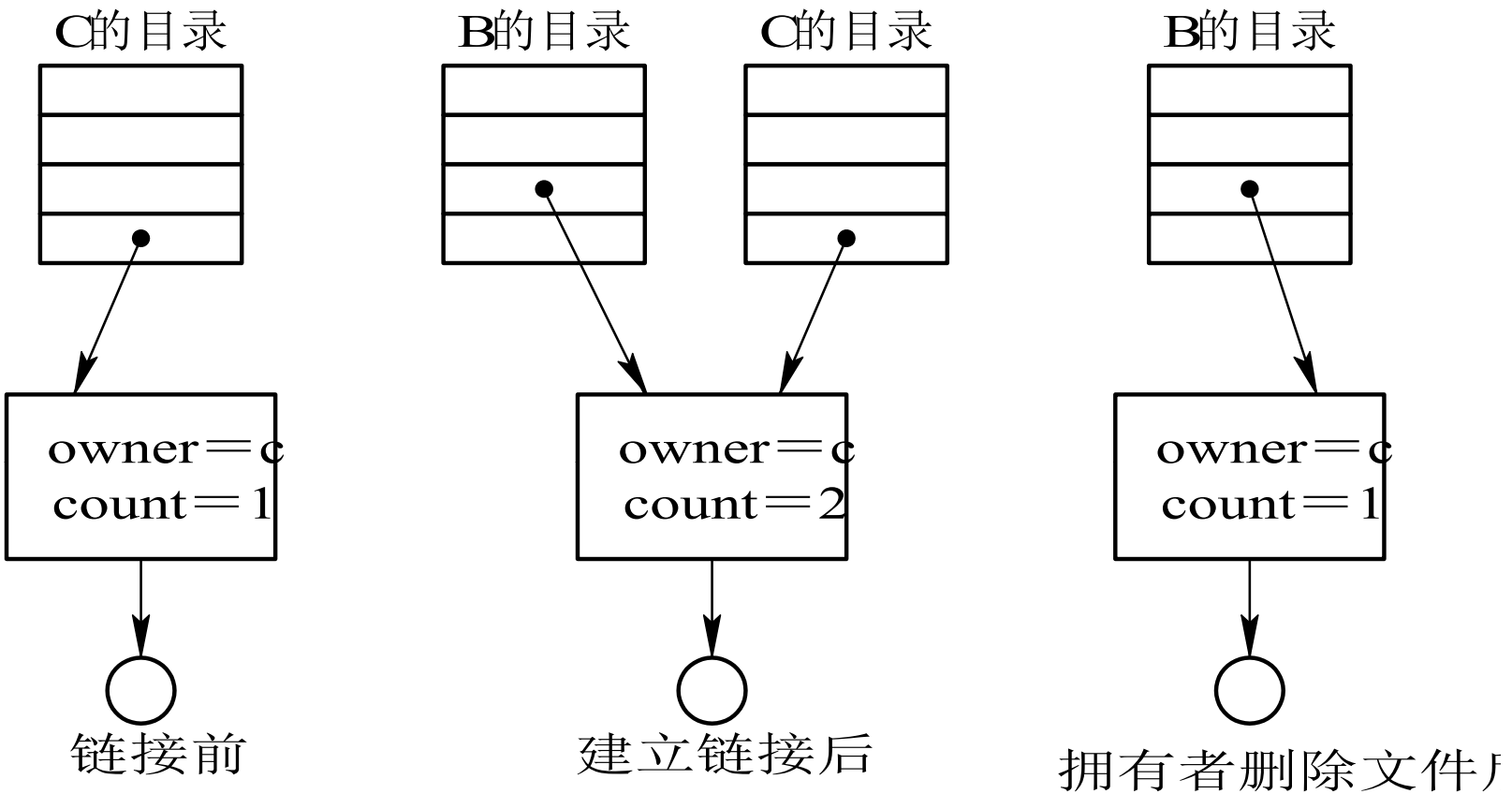
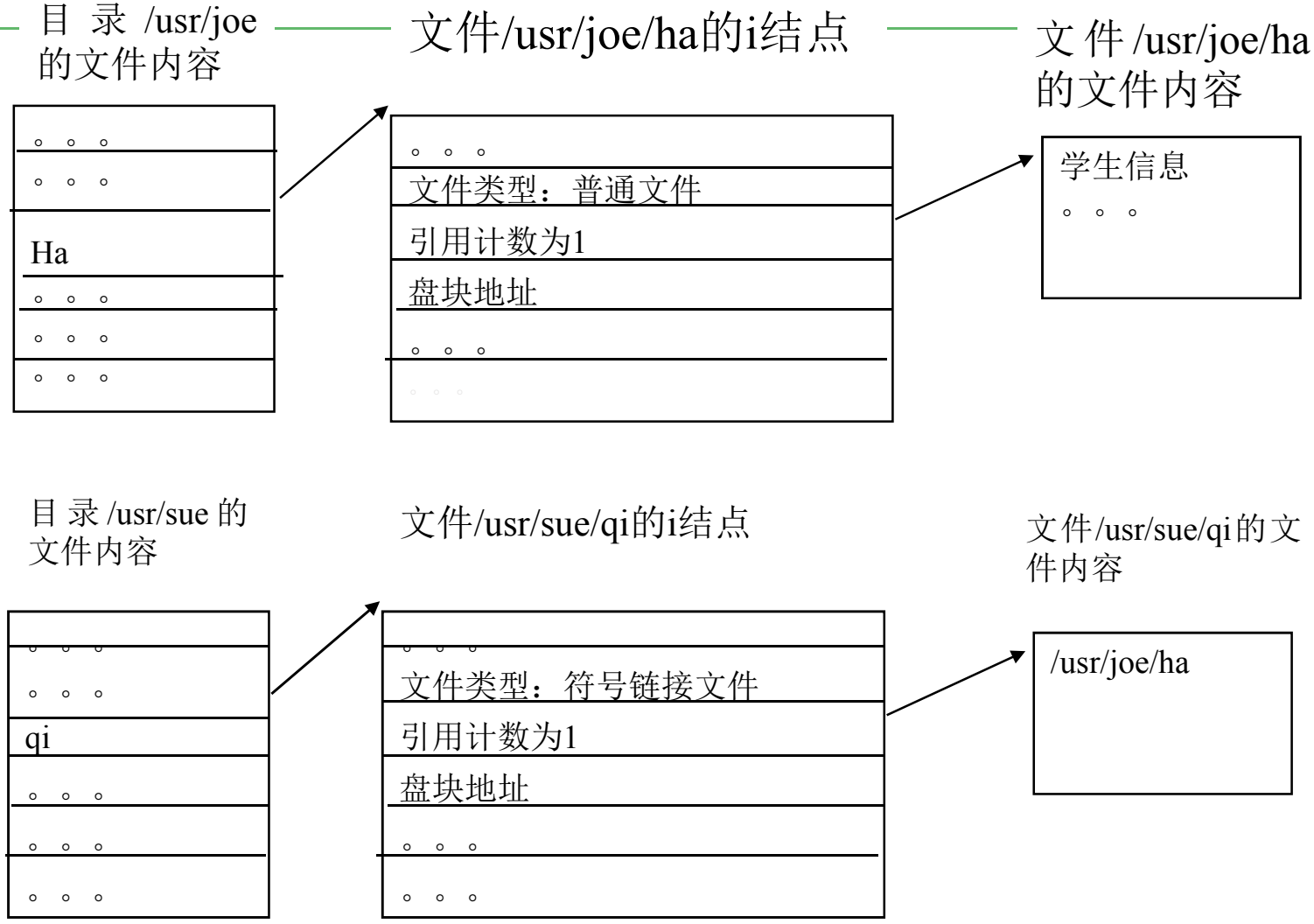


图6-15　进程B链接前后的情况

当用户C创建一个新文件时，他便是该文件的所有者，此时将count置1。当有用户B要共享此文件时，在用户B的目录中增加一目录项，并设置一指针指向该文件的索引结点，此时，文件主仍然是C，count=2。如果用户C不再需要此文件，是否能将此文件删除呢？回答是否定的。因为，若删除了该文件，也必然删除了该文件的索引结点，这样便会使B的指针悬空，而B则可能正在此文件上执行写操作，此时将因此半途而废。但如果C不删除此文件而等待B继续使用，这样，由于文件主是C，如果系统要记账收费，则C必须为B使用此共享文件而付账，直至B不再需要。

7.6.2 利用符号链实现文件共享

为使B能共享C的一个文件F，可以由系统创建一个LINK类型的新文件，也取名为F，并将F写入B的目录中，以实现B的目录与文件F的链接。在新文件中只包含被链接文件F的路径名。这样的链接方法被称为符号链接(Symbolic Linking)。新文件中的路径名则只被看作是符号链(Symbolic Link)，当B要访问被链接的文件F且正要读LINK类新文件时，此要求将被OS截获，OS根据新文件中的路径名去读该文件，于是就实现了用户B对文件F的共享。



基于符号链接的共享方式

在利用符号链方式实现文件共享时，只是文件主才拥有指向其索引结点的指针；而共享该文件的其他用户则只有该文件的路径名，并不拥有指向其索引结点的指针。这样，也就不会发生在文件主删除一共享文件后留下一悬空指针的情况。当文件的拥有者把一个共享文件删除后，其他用户试图通过符号链去访问一个已被删除的共享文件时，会因系统找不到该文件而使访问失败，于是再将符号链删除，此时不会产生任何影响。

符号链方式有一个很大的优点，是它能够用于链接(通过计算机网络)世界上任何地方的计算机中的文件，此时只需提供该文件所在机器的网络地址以及该机器中的文件路径即可。

然而符号链的共享方式也存在自己的问题：当其他用户去读共享文件时，系统是根据给定的文件路径名，逐个分量(名)地去查找目录，直至找到该文件的索引结点。因此，在每次访问共享文件时，都可能要多次地读盘。这使每次访问文件的开销甚大，且增加了启动磁盘的频率。此外，要为每个共享用户建立一条符号链，而由于该链实际上是一个文件，尽管该文件非常简单，却仍要为它配置一个索引结点，这也要耗费一定的磁盘空间。

7.6.3 文件的保护

1. 存取控制矩阵

2. 存取控制表：系统为每个文件在其FCB中设置一个存取控制表，表目内容包括用户身份识别以及所具有的存取权限。

用户类	存取权限
C1	RWE
C2	RE
C3	E
C4	None

图7－20 存取控制表

3．口令

4．密码方式：加密和解密都需要耗费处理机的时间。

5. 其它方式

7.7 并发控制

在多用户系统和计算机网络环境下，可能有多个用户在同时执行事务。由于事务具有原子性，这使各个事物的执行必然是按照某种顺序一次执行的，即各事务对数据项的修改是互斥的。人们把这种特性称为顺序性（Serializablity)。把用于实现事务顺序性的技术称为并发控制（Concurrent Control)。该技术在应用数据库系统中已被广泛采用，现也广泛应用于OS中。

1．利用互斥锁实现“顺序性”

实现顺序性的一种最简单的方法是，设置一种用于实现互斥的锁，简称为互斥锁(Exclusive Lock)。

在利用互斥锁实现顺序性时，应为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，便用该锁将该对象锁住，于是事务T_i便可对该对象执行读或写操作；而其它事务由于未能获得该锁而不能访问该对象。如果T_i需要对一批对象进行访问，则为了保证事务操作的原子性，T_i应先获得这一批对象的互斥锁，以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，但不致引起数据的变化。

2．利用互斥锁和共享锁实现顺序性

利用互斥锁实现顺序性的方法简单易行。目前有不少系统都是采用这种方法来保证事务操作的顺序性，但这却存在着效率不高的问题。因为一个共享文件虽然只允许一个事务去写，但却允许多个事务同时去读；而在利用互斥锁来锁住文件后，则只允许一个事务去读。为了提高运行效率而又引入了另一种形式的锁——共享锁(Shared Lock)。共享锁与互斥锁的区别在于: 互斥锁仅允许一个事务对相应对象执行读或写操作，而共享锁则允许多个事务对相应对象执行读操作，不允许其中任何一个事务对对象执行写操作。

在为一个对象设置了互斥锁和共享锁的情况下，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。利用共享锁和互斥锁来实现顺序性的方法，非常类似于我们在第二章中所介绍的读者—写者问题的解法。

2. 重复数据的数据一致性问题

（1）重复文件的一致性

我们以UNIX类型的文件系统为例，来说明如何保证重复文件的一致性问题。对于通常的UNIX文件目录，其每个目录项中含有一个ASCII码的文件名和一个索引结点号，后者指向一个索引结点。当有重复文件时，一个目录项可由一个文件名和若干个索引结点号组成，每个索引结点号都是指向各自的索引结点。下图示出了UNIX类型的目录和具有重复文件的目录。

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

文件名	i 结点		
文件 1	17	19	40
文件 2	22	72	91
文件 3	12	30	29
文件 4	84	15	66

图6-31　UNIX类型的目录

在有重复文件时，如果一个文件拷贝被修改，则必须也同时修改其它几个文件拷贝，以保证各相应文件中数据的一致性。这可采用两种方法来实现：第一种方法是当一个文件被修改后，可查找文件目录，以得到其它几个拷贝的索引结点号，再从这些索引结点中找到各拷贝的物理位置，然后对这些拷贝做同样的修改；第二种方法是为新修改的文件建立几个拷贝，并用新拷贝去取代原来的文件拷贝。

2．盘块号一致性的检查

为了描述盘块的使用情况，通常利用空闲盘块表(链)来记录所有尚未使用的空闲盘块的编号。文件分配表FAT则是用于记录已分配盘块的使用情况。由于OS经常访问这些数据结构，也对它们进行修改，而如果正在修改时，机器突然发生故障，此时也会使盘块数据结构中的数据产生不一致性现象。因此，在每次启动机器时，都应该检查相应的多个数据结构，看它们之间是否保持了数据的一致性。

为了保证盘块数据结构(中数据)的一致性，可利用软件方法构成一个计数器表，每个盘块号占一个表项，可有0，…，*N*-1项，*N*为盘块总数。每一个表项中包含两个计数器，分别用作空闲盘块号计数器和数据盘块号计数器。计数器表中的表项数目等于盘块数N。

在对盘块的数据结构进行检查时，应该先将计数器表中的所有表项初始化为0，然后，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；再用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，亦即，若某个盘块号在被第一组计数器进行计数后，使该盘块号计数器为1，则在第二组计数器中相应盘块号计数器中的计数必为0；反之亦然。但如果情况并非如此，则说明发生了某种错误。

图(a)示出了在正常情况下，在第一组计数器和第二组计数器中的盘块号计数值是互补的；而图 (b)示出的则是一种不正常的情况，对盘块号2的计数值在两组计数器中都未出现(即均为0)。当检查出这种情况时，应向系统报告。该错误的影响并不大，只是盘块2未被利用。其解决方法也较简单，只需在空闲盘块表(链)中增加一个盘块号2。图 (c)中示出了另一种错误，即盘块号4在空闲盘块表(链)中出现了两次，其解决方法是从空闲盘块表(链)中删除一个空闲盘块号4。图(d)中所示出的情况是相同的数据盘块号5出现了两次(或多次)，此种情况影响较严重，必须立即报告。

盘块号 / 计数器组	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
空闲盘块号计数器组	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
数据盘块号计数器组	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(a) 正常情况盘块号

盘块号 / 计数器组	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
空闲盘块号计数器组	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
数据盘块号计数器组	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1

(b) 丢失了盘块盘块号

计数器组 \ 盘块号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
空闲盘块号计数器组	1	1	0	1	2	1	1	1	1	0	0	1	1	1	0	0
数据盘块号计数器组	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0

(c) 空闲盘块号重复出现盘块号

计数器组 \ 盘块号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
空闲盘块号计数器组	1	1	0	1	1	0	1	1	1	0	0	1	1	1	0	0
数据盘块号计数器组	0	0	1	0	0	2	0	0	0	1	1	0	0	0	1	1

(d) 数据盘块号重复出现

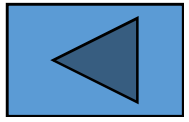
（3）链接数一致性检查

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；另一方面，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。在正常情况下这两个数据应该一致，否则就会出现数据不一致性差错。

为了检查这种数据不一致性差错，同样要配置一张计数器表，此时应是为每个文件而不是为每个盘块建立一个表项，其中含有该索引结点号的计数值。在进行检查时，从根目录开始查找，每当在目录中遇到该索引结点号时，便在该计数器表中相应文件的表项上加1。当把所有目录都检查完后，便可将该计数器表中每个表项中的索引结点号计数值与该文件索引结点中的链接计数count值加以比较，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。

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。这种错误的后果是使一些已无用户需要的文件仍驻留在磁盘上，浪费了存储空间。当然这种错误的性质并不严重。解决的方法是用计数器表中的正确的计数值去为count重新赋值。

反之，如果出现count值小于计数器表中索引结点号计数值的情况时，就有潜在的危险。假如有两个用户共享一个文件，但是count值仍为1，这样，只要其中有一个用户不再需要此文件时，count值就会减为0，从而使系统将此文件删除，并释放其索引结点及文件所占用的盘块，导致另一共享此文件的用户所对应的目录项指向了一个空索引结点，最终是使该用户再无法访问此文件。如果该索引结点很快又被分配给其它文件，则又会带来潜在的危险。解决的方法是将count值置为正确值。



7.8 Linux文件管理

7.8.1 Linux文件系统概论

1.Linux文件系统的树形结构

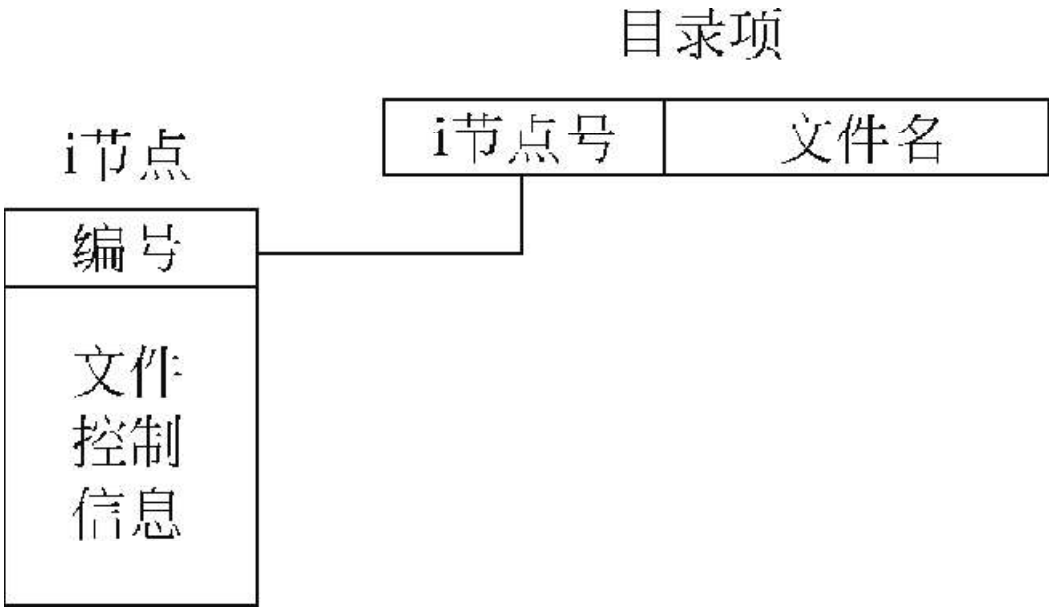


图7-27 Linux文件系统的目录项

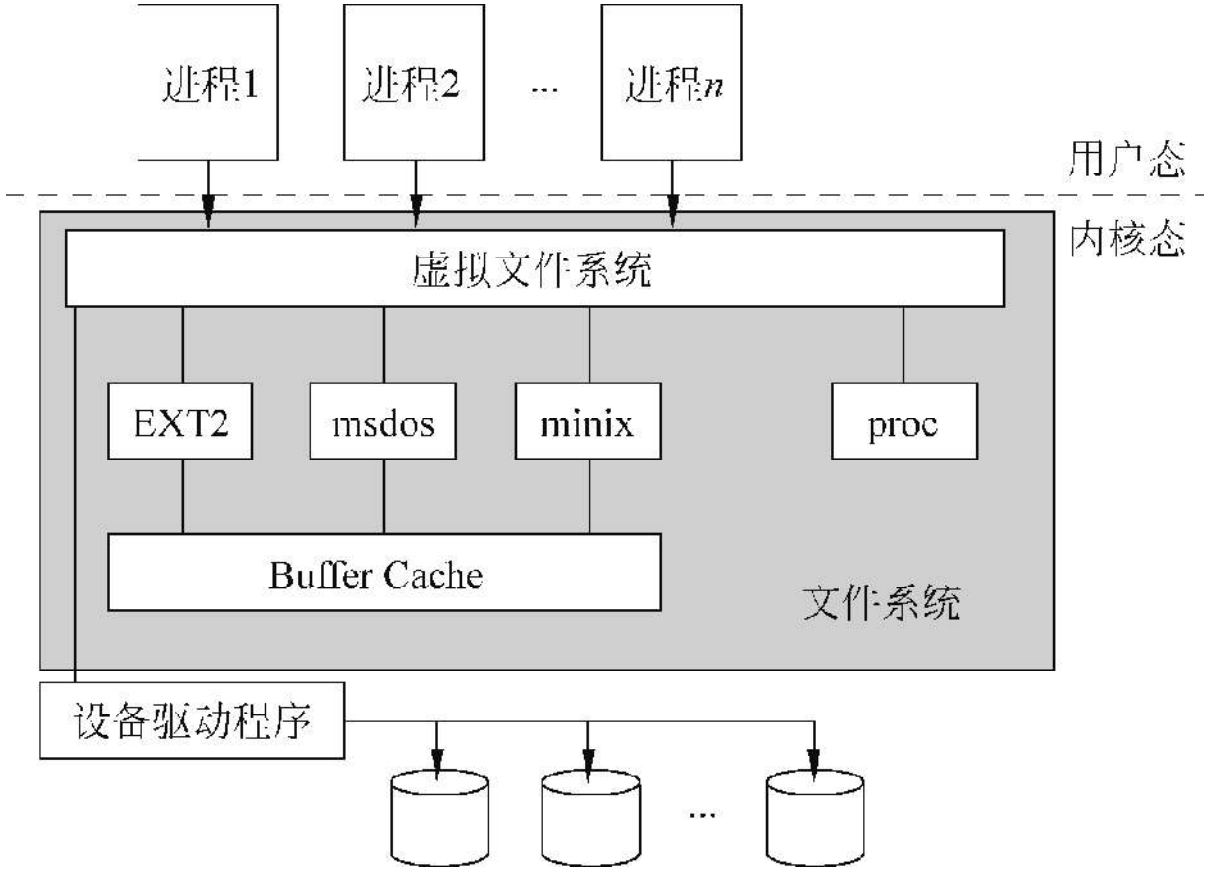
2.Linux文件的类型

1）普通文件
2）目录文件
3）设备文件
4）管道文件
5）链接文件

3.Linux文件的访问权限

7.8.2 虚拟文件系统

1. VFS的工作原理



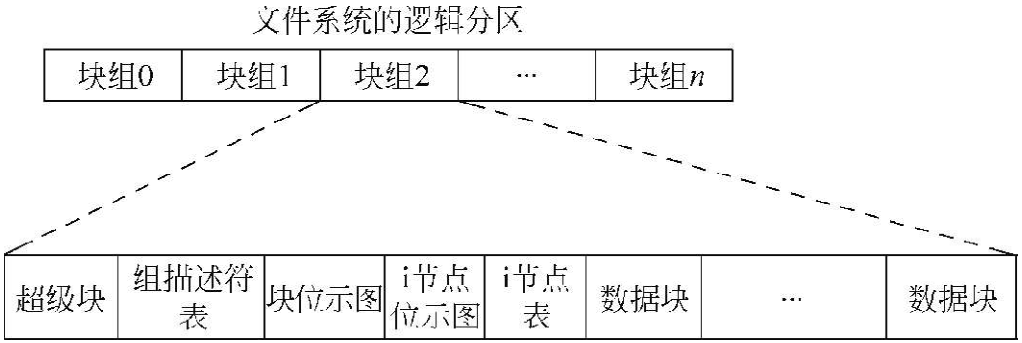
2. 文件系统的注册

3. 文件系统的安装

7.8.3 EXT文件系统

1. 文件系统的构造

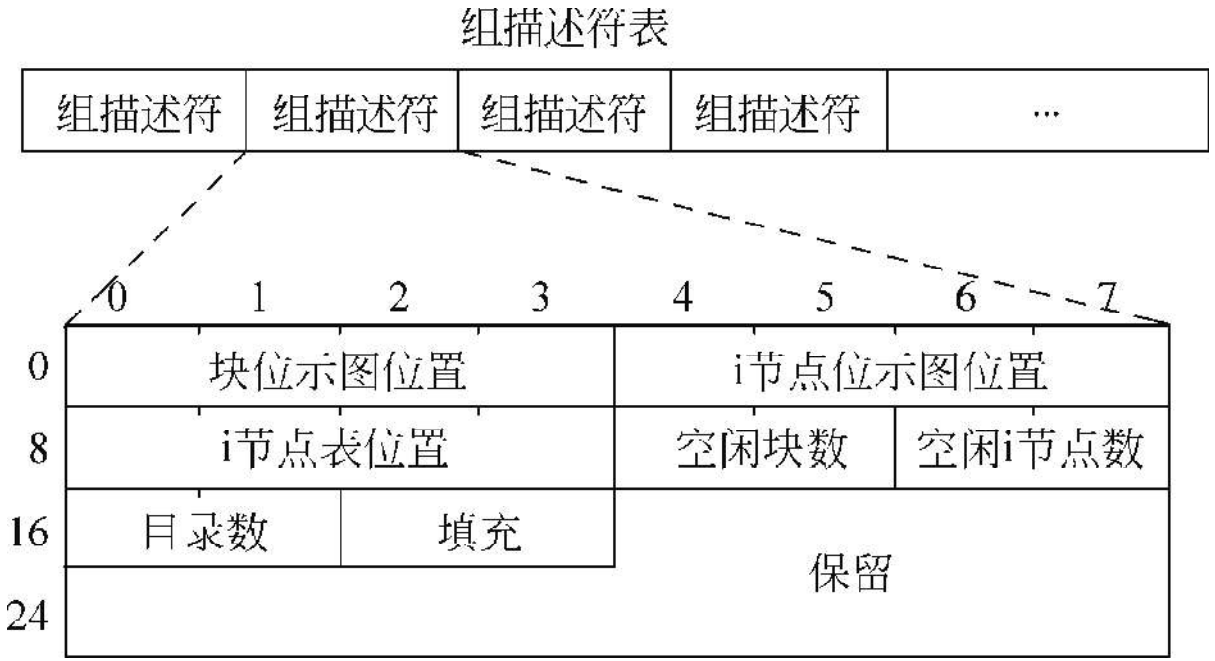
Linux文件系统把逻辑分区划分成块组，并从0开始依次编号。每个块组中包含若干数据块，数据块中就是目录或文件内容。块组中包含着几个用于管理和控制的信息块：
超级块、组描述符表、块位示图、i节点位示图和i节点表。



2. 超级块

超级块是用来描述Linux文件系统整体信息的数据结构，主要描述文件系统的目录和文件的静态分布情况，以及描述文件系统的各种组成结构的尺寸、数量等。

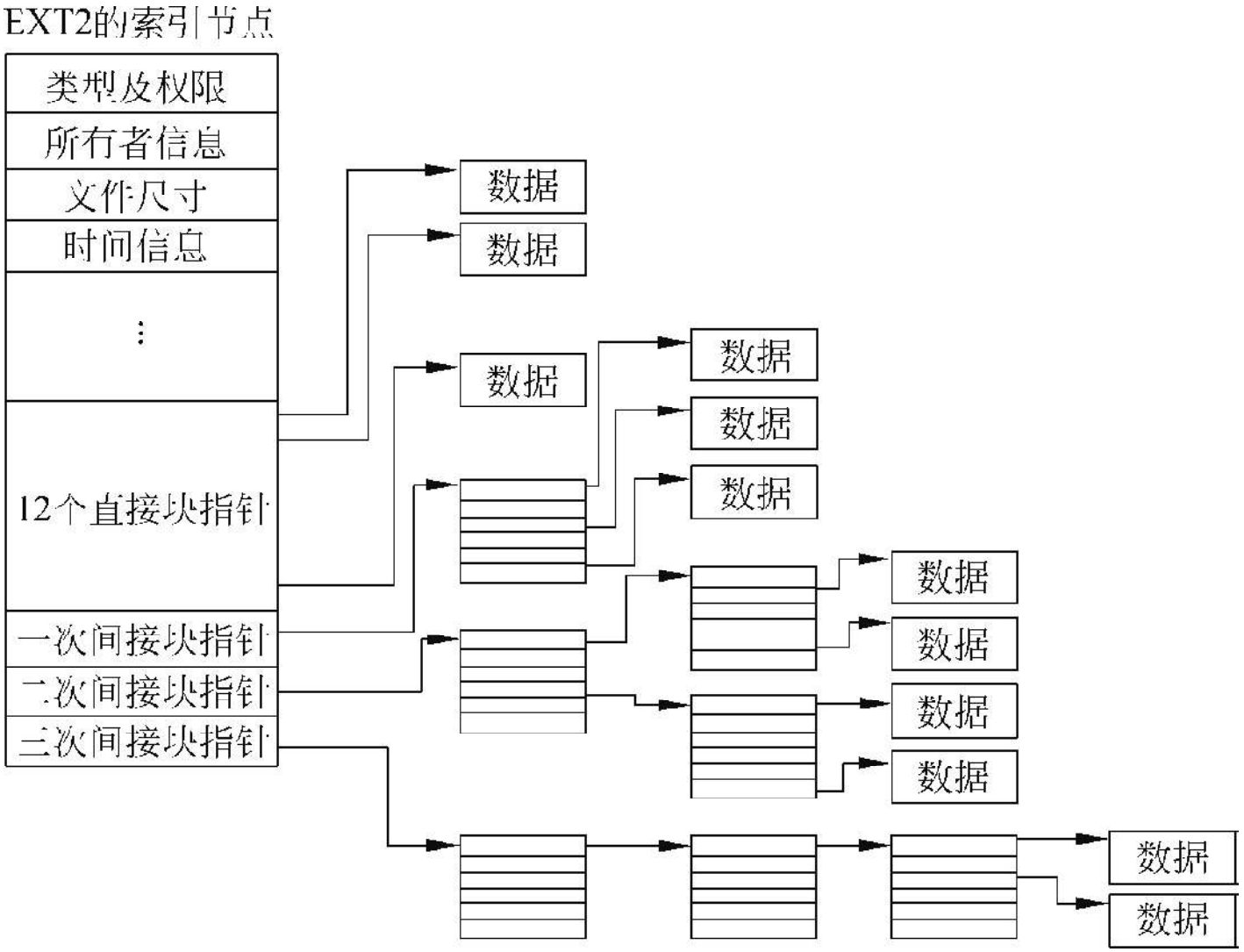
3. 组描述符表



4. 块位示图

Linux文件系统中数据块的使用状况由块位示图来描述。每个块组都有一个块位示图,位于组描述符表之后,用来描述本块组中数据块的使用状况。

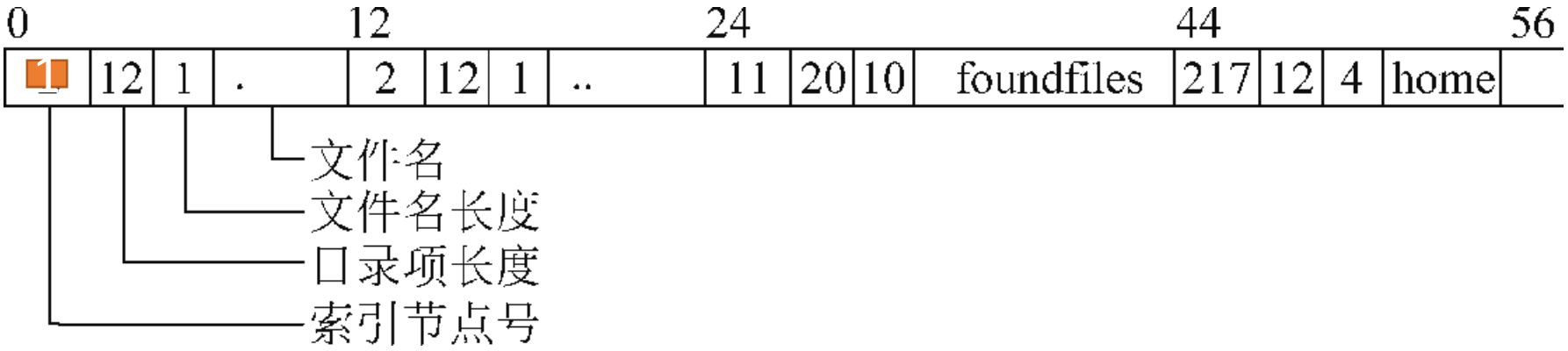
5. 索引节点


EXT2的索引节点
类型及权限
所有者信息
文件尺寸
时间信息
12个直接块指针
一次间接块指针
二次间接块指针
三次间接块指针
数据
数据
数据
数据
数据
数据
数据
数据
数据
数据
数据

6. 索引节点表和索引节点位图

一个块组中所有文件的索引节点形成了索引节点表。表项的序号就是索引节点号。索引节点位示图反映了索引节点表中各个表项的使用情况。

7. EXT的目录结构

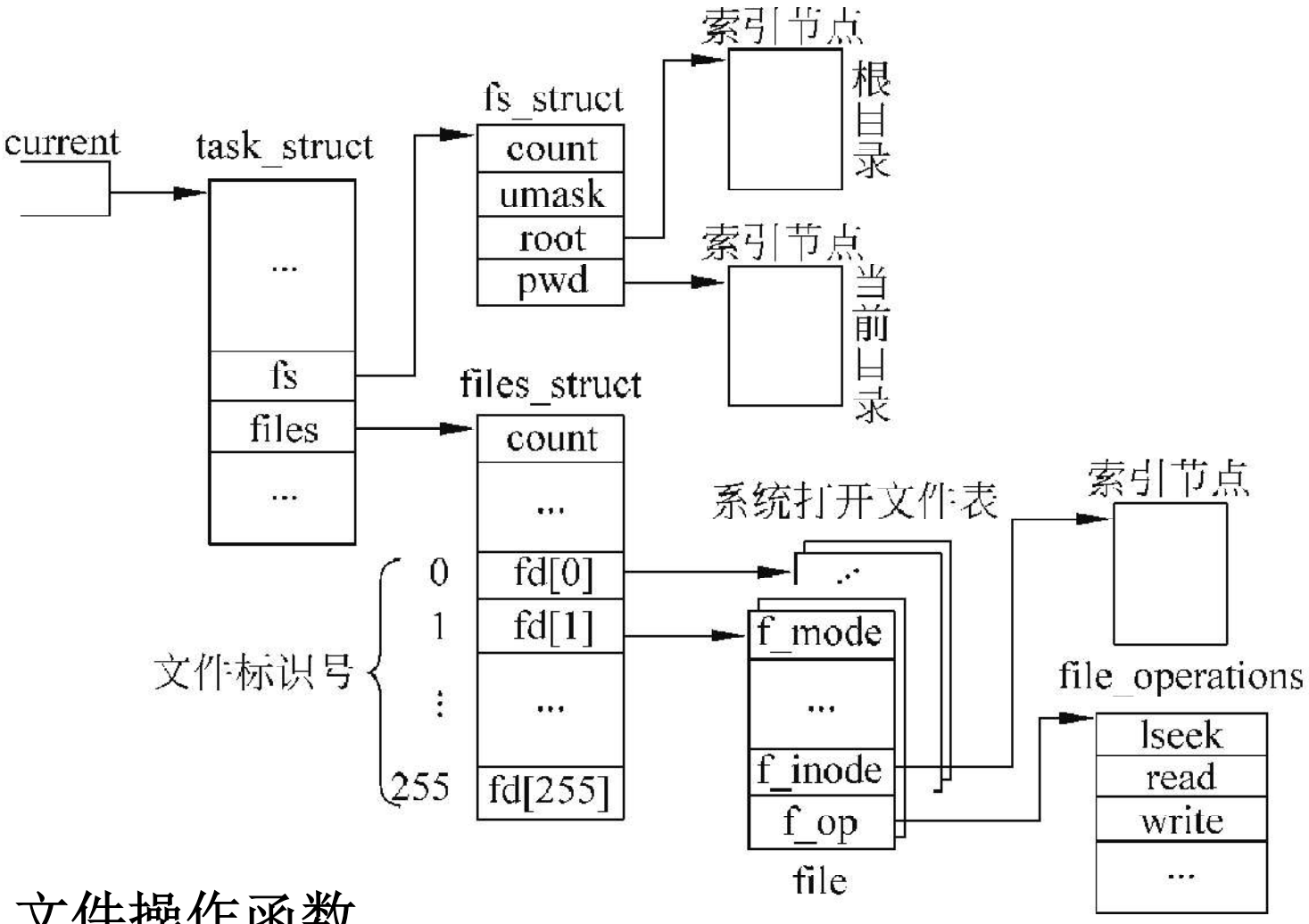


7.8.4 文件管理和操作

1. 系统打开文件表

```
Struct_file{
        mode_tf_mode;                                   /*文件的打开模式*/
        loff_tf_pos;        /*文件的当前读写位置*/
        unsignedshortf_flags; /*文件操作标志*/
        unsignedshortf_count; /*共享该结构体的计数值*/
        unsignedlongf_reada,f_ramax,f_raend,f_ralen,f_rawin;
        structfile *f_next,*f_prev; /*链接前后节点的指针*/
        structfown_structf_owner;    /*SIGIO用PID*/
        structinode *f_inode;           /*指向文件对应的索引节点*/
        structfile_operations *f_op;    /*指向文件操作结构体的指针*/
        unsignedlongf_version;           /*文件版本*/
        void *private_data;
}
```

2. 进程的文件管理



3. 文件操作函数