

# 第七章 文件管理

任立勇

2020年6月

# ? 问题

- 什么是文件？
- 文件由什么组成？
- 文件如何命名？
- 如何保证文件数据的安全？
- 对文件可以进行哪些操作？
- 文件在磁盘上如何存储？
- 磁盘的空白存储区如何管理 ？

# 本章要点

- 文件和文件系统
- 文件的逻辑结构
- 文件目录
- 文件共享
- 文件保护

# 概述

- 内存是易失性设备，其中存储的信息会因为断电而丢失
- 将系统和用户需要用到的大量程序和数据以文件的形式存放在外存上
- 操作系统中的文件管理模块专门管理在外存上的文件，并把对文件的存取、共享和保护等手段提供给用户。

# 7.1 文件和文件系统

- 有效地管理文件的存储空间；
- 管理文件目录；
- 完成文件的读/写操作；
- 实现文件共享与保护；
- 为用户提供交互式命令接口和程序调用接口。



## 7.1.1文件、记录和数据项

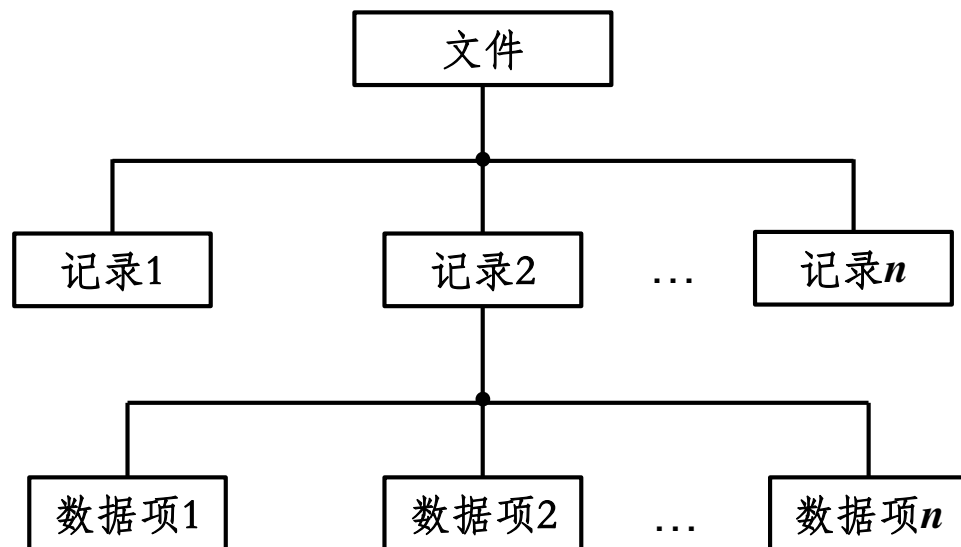


图7-1 文件、记录和数据项之间的层次关系



## 7.1.1 文件、记录和数据项

- 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，即原子数据，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，经理便是个组项，它由正经理和副经理两个基本项组成。又如，工资也是个组项，它可由基本工资、工龄工资和奖励工资等基本项所组成。
- 基本数据项除了数据名外，还应有数据类型。根据属性的不同，需要用不同的数据类型来描述。
  - 例如，在描述学生的学号时，应使用整数；描述学生的姓名则应使用字符串(含汉字)；描述性别时，可用逻辑变量或汉字。



- 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。
- 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。
  - 在有结构的文件中，文件由若干个相关记录组成；
  - 而无结构文件则被看成是一个字符流。
- 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。

# 文件的属性

- 文件的属性可以包括：
  - (1) 文件类型。
  - (2) 文件长度。
  - (3) 文件的物理位置。
  - (4) 文件的建立时间。

## 7.1.2 文件名和文件类型

### 1、文件名和扩展名

- 不同的操作系统对文件名的规定不同。例如：MS-DOS最多只允许8个字符，UNIX系统支持14个字符等。一些特殊字符也不能作为文件名
- 扩展名通常作为指示文件的类型

# 7.1.2 文件名和文件类型

## 1. 文件类型

### 1) 按用途分类

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** 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。
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

### 1. 源文件

指由源程序和数据构成的文件。

### 2. 目标文件

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。

### 3. 可执行文件

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：

- **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。
- **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。由于目录文件也是由字符序列构成，因此对其可进行与普通文件一样的种种文件操作。
- **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

## 7.1.3 文件系统的层次结构

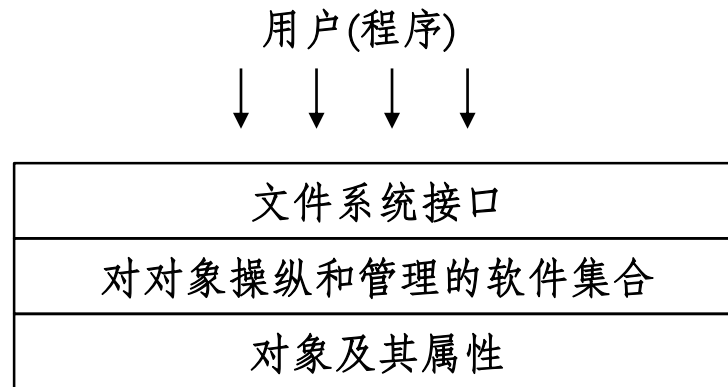


图7-2 文件系统模型

## 1) 对象及其属性

文件管理系统管理的对象有：

- ①文件。它作为文件管理的直接对象。
- ②目录。为了方便用户对文件的存取和检索，在文件系统中必须配置目录。对目录的组织和管理是方便用户和提高对文件存取速度的关键。
- ③磁盘(磁带)存储空间。文件和目录必定占用存储空间，对这部分空间的有效管理，不仅能提高外存的利用率，而且能提高对文件的存取速度。

## 2) 对对象操纵和管理的软件集合

这是文件管理系统的核心部分。文件系统的功能大多是在这一层实现的，其中包括：

- 对文件存储空间的管理
- 对文件目录的管理
- 用于将文件的逻辑地址转换为物理地址的机制
- 对文件读和写的管理
- 以及对文件的共享与保护等功能。

### 3) 文件系统的接口

为方便用户使用文件系统，文件系统通常向用户提供两种类型的接口：

(1) **命令接口**。这是指作为用户与文件系统交互的接口。 用户可通过键盘终端键入命令，取得文件系统的服务。

(2) **程序接口**。这是指作为用户程序与文件系统的接口。 用户程序可通过系统调用来取得文件系统的服务。



## 7.1.4 文件操作

- 用户通过文件系统提供的系统调用实施对文件的操作。
  1. 最基本的文件操作有：创建文件、删除文件。读文件、写文件、截断文件和设置文件的读 / 写位置。
  2. 文件的“打开”和“关闭”操作：所谓“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。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。
  3. 其它文件操作：对文件属性的操作，改变文件名、改变文件的拥有者，查询文件的状态等；



# 文件操作实例（Linux）

- open: 打开一个文件，并指定访问该文件的方式，调用成功后返回一个文件描述符。
- creat: 打开一个文件，如果该文件不存在，则创建它，调用成功后返回一个文件描述符。
- close: 关闭文件，进程对文件所加的锁全都被释放。
- read: 从文件描述符对应的文件中读取数据，调用成功后返回读出的字节数。
- write: 向文件描述符对应的文件中写入数据，调用成功后返回写入的字节数。



# 文件操作实例 (Linux)

- ```
# include <fcntl.h>
# include <unistd.h>
# include <sys/types.h>
# include <sys/stat.h>
int open(const char *pathname,int flags);
int open(const char *pathname,int flags,mode_t mode);
int close(int fd);
```

open函数有两个形式.其中pathname是我们要打开的文件名(包含路径名称,缺省是认为在当前路径下面).

- Flags指文件的打开方式:
  - O\_RDONLY:以只读的方式打开文件.
  - O\_WRONLY:以只写的方式打开文件.
  - O\_RDWR:以读写的方式打开文件.
  - O\_APPEND:以追加的方式打开文件.
  - O\_CREAT:创建一个文件.
  - O\_EXEC:如果使用了O\_CREAT而且文件已经存在,就会发生一个错误.
  - O\_NOBLOCK:以非阻塞的方式打开一个文件.
  - O\_TRUNC:如果文件已经存在,则删除文件的内容.

- `size_t write(int fildes,const void *buf,size_t nbytes);`
- 参数说明：
  - fildes: 与文件相对应的文件描述符，可通过调用open函数获取
  - buf: 存放将写入文件的数据，可以是字符串，也可是其他数据。其中buf是指向字符串的指针
  - nbytes: 需写进文件的字节数
- `size_t read(int fildes,char *buf,size_t nbytes);`
- 参数说明：
  - fildes: 文件描述符
  - buf: 存放从文件中读取的数据
  - nbytes: 希望读取的直接数

- 如果使用了O\_CREATE标志，那么要使用open的第二种形式。
- 即需要指定mode标志,用来表示文件的访问权限.mode可以是以下情况的组合.

-----  
S\_IRUSR 用户可以读 S\_IWUSR 用户可以写  
S\_IXUSR 用户可以执行 S\_IRWXU 用户可以读写执行  
-----

S\_IRGRP 组可以读 S\_IWGRP 组可以写  
S\_IXGRP 组可以执行 S\_IRWXG 组可以读写执行  
-----

S\_IROTH 其他人可以读 S\_IWOTH 其他人可以写  
S\_IXOTH 其他人可以执行 S\_IRWXO 其他人可以读写执行  
-----

S\_ISUID 设置用户执行ID S\_ISGID 设置组的执行ID

- 功能：write从buffer中写count字节到文件fd中,成功时返回实际所写的字节数.

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>
#define BUFFER_SIZE 1024
int main(int argc,char **argv)
{
    int from_fd,to_fd;
    int bytes_read,bytes_write;
    char buffer[BUFFER_SIZE];
    char *ptr;
    if(argc!=3)
    {
        fprintf(stderr,"Usage:%s fromfile tofile\n\a",argv[0]);
        exit(1);
    }
}
```



- /\* 打开源文件 \*/  
if((from\_fd=open(argv[1],O\_RDONLY))== -1)  
{  
fprintf(stderr,"Open %s Error:%s\n",argv[1],strerror(errno));  
exit(1);  
}
- /\* 创建目的文件 \*/
- if((to\_fd=open(argv[2],O\_WRONLY O\_CREAT,S\_IRUSR  
S\_IWUSR))== -1)  
{  
fprintf(stderr,"Open %s Error:%s\n",argv[2],strerror(errno));  
exit(1);  
}



- /\* 以下代码是一个经典的拷贝文件的代码 \*/
- ```
while(bytes_read=read(from_fd,buffer,BUFFER_SIZE))
{
    if((bytes_read==-1)&&(errno!=EINTR)) break; /* 发生致命错误 */
    else if(bytes_read>0)
    {
        ptr=buffer;
        while(bytes_write=write(to_fd,ptr,bytes_read))
        {
            /* 一个致命错误发生了 */
            if((bytes_write==-1)&&(errno!=EINTR))break;
            else if(bytes_write==bytes_read) break; /* 写完了所有读的字节 */

            else if(bytes_write>0) /* 只写了一部分,继续写 */

            {
                ptr+=bytes_write;
                bytes_read-=bytes_write;
            }
        }
        if(bytes_write==-1)break; /* 写的时候发生的致命错误 */

    }
}
close(from_fd);
close(to_fd);
exit(0);
}
```

## 7.2文件的逻辑结构

- 文件是由一系列的记录组成的。
- 对于任何一个文件，都存在着以下两种形式的结构：

### (1) 文件的逻辑结构

从用户观点出发所观察到的文件组织形式。

### (2) 文件的物理结构

指文件在外存上的存储组织形式。

## 7.2 文件逻辑结构

- 文件的逻辑结构可分为两大类：

- (1) 有结构文件：是指由一个以上的记录构成的文件，故又把它称为记录式文件；

- 记录的长度可分为定长和不定长两类。

- 可采用多种方式组织记录，形成不同的文件：

- ①顺序文件：是由一系列记录按某种顺序排列所形成的文件。

- ②索引文件：当记录为可变长度时，通常为之建立一张索引表。

- ③索引顺序文件：它为文件建立一张索引表，为每一组记录中的第一个记录设置一个表项。

## 2. 无结构文件

大量的源程序、可执行文件、库函数等，所采用的就是无结构的文件形式，即流式文件。其长度以字节为单位。

对流式文件的访问，则是采用读写指针来指出下一个要访问的字符。可以把流式文件看作是记录式文件的一个特例。

在UNIX系统中，所有的文件都被看作是流式文件；即使是有结构文件，也被视为流式文件；系统不对文件进行格式处理。

## 7.2.2 顺序文件

### 1. 逻辑记录的排序

- 文件中的记录排列可归纳为以下两种情况：

- ①串结构，各记录之间的顺序与关键字无关。通常的办法是由时间来决定，即按存入时间的先后排列。
- ②顺序结构，指文件中的所有记录按关键字排列。



## 2. 对顺序文件的读 / 写操作

- 对于定长记录的顺序文件，如果已知当前记录的逻辑地址，便很容易确定下一个记录的逻辑地址。在读一个文件时，可设置一个读指针Rptr（见图7-3）。令它指向下一个记录的首地址，每当读完一个记录时，便执行：

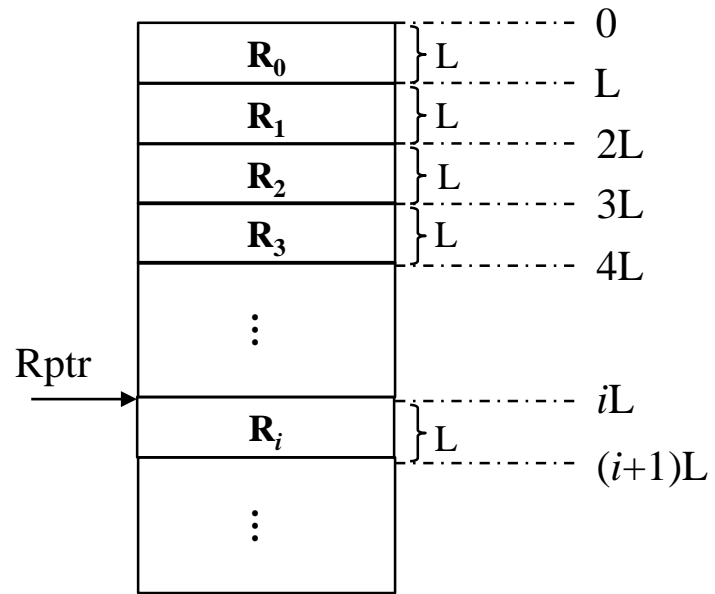
$$Rptr: = Rptr + L$$

- 对于变长记录的顺序文件，在每次读或写完一个记录后，须将读或写指针加上Li。

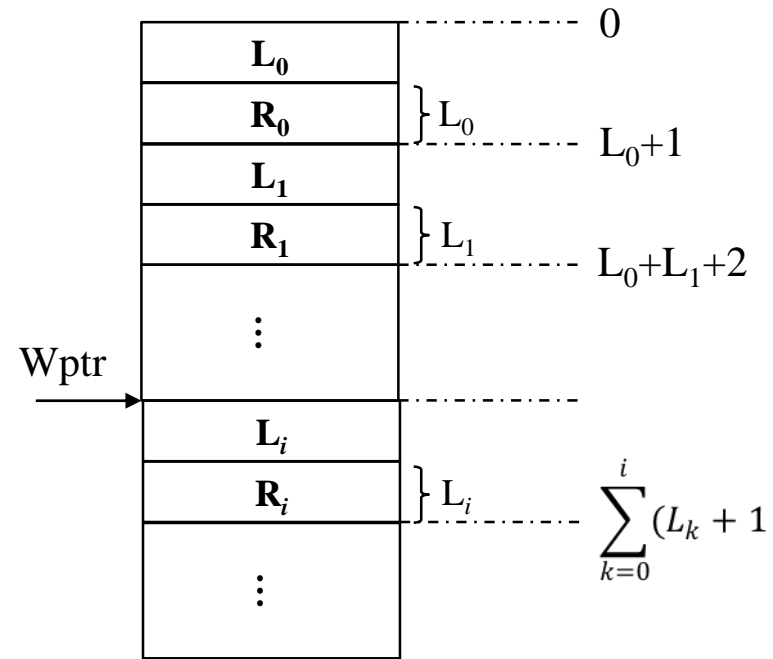
$$Wptr: = Wptr + Li$$



# 定长和变长记录文件



(a) 定长记录文件



(b) 变长记录文件

图7-3 定长和变长记录文件

### 3. 顺序文件的优缺点：

优点：

- (1)对顺序文件的存取效率是所有逻辑文件中最高的.
- (2)只有顺序文件才能存储在磁带上，并能有效地工作。

缺点：

- (1)在交互应用的场合，如果用户（程序）要求查找或修改单个记录，为此系统便要去逐个地查找诸记录。
- (2)如果想增加或删除一个记录，都比较困难；

## 7.2.3 记录寻址

- 对于定长记录文件，如果要查找第*i*个记录，第*i*个记录相对于第一个记录首址的地址：

$$A_i = i \times L$$

- 对于可变长度记录的文件，要查找其第*i*个记录时，须顺序地查找每个记录，从中获得相应记录的长度*L<sub>i</sub>*，然后才能按下式计算出第*i*个记录的首址。

$$A_i = \sum_{j=0}^{i-1} L_j + 1$$

- 可见，对于定长记录，除了可以方便地实现顺序存取外，还可较方便地实现直接存取。然而，对于变长记录就较难实现直接存取了。

## 7.2.4 索引文件

- 为变长记录文件建立一张索引表，对主文件中的每个记录，在索引表中设有一个相应的表项，用于记录该记录的长度 $L$ 及指向该记录的指针(指向该记录在逻辑地址空间的首址)。
- 由于索引表是按关键字排序的，因此，索引表本身是一个定长记录的顺序文件，从而也就可以方便地实现直接存取。

# 3.索引文件

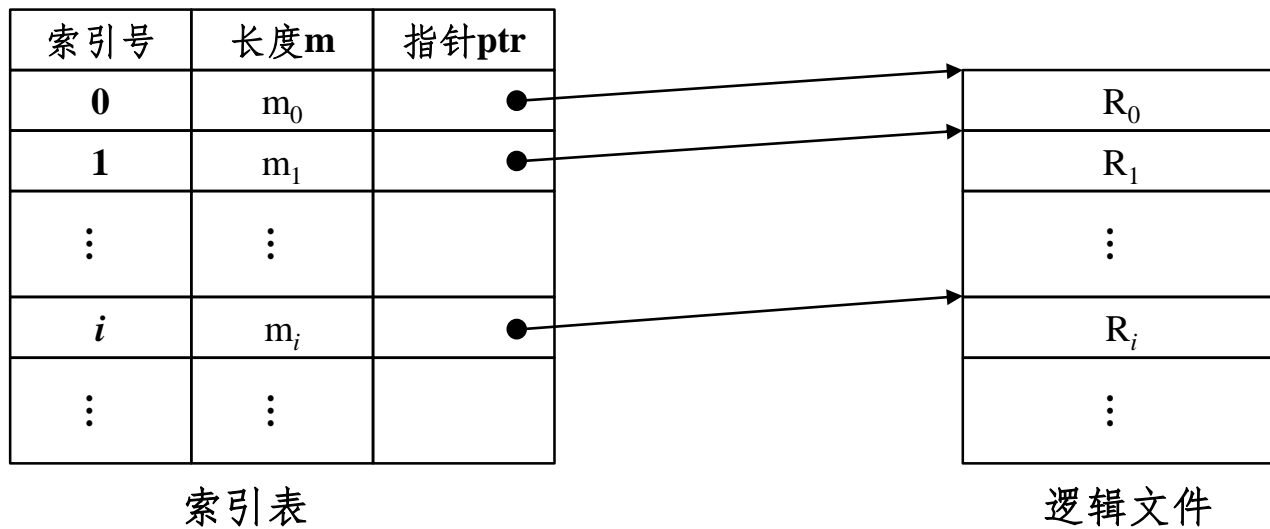


图7-4 索引文件的组织

### 3.索引文件的检索

- 利用折半查找法去检索索引表，从中找到相应的表项；
- 再利用该表项中给出的指向记录的指针值，去访问所需的记录。
- 而每当要向索引文件中增加一个新记录时，便须对索引表进行修改。
- 由于索引文件可有较快的检索速度，故它主要用于对信息处理的及时性要求较高的场合.例如，飞机订票系统。
- 主要问题：它除了有主文件外，还须配置一张索引表，而且每个记录都要有一个索引项，因此提高了存储费用。



## 7.3 文件目录

- 对目录管理的要求如下：
  - (1) 实现“按名存取”。
  - (2) 提高对目录的检索速度。
  - (3) 文件共享。
  - (4) 允许文件重名。
- 文件控制块（FCB）：用于描述和控制文件的数据结构
- 文件目录：文件控制块的有序集合。

## 7.3.1 文件控制块FCB

- 基本信息：文件名、文件类型等；
- 地址信息：**卷**（存储文件的设备）、**起始地址**（起始物理地址）、**文件长度**（以字节、字或块为单位）等。
- 访问控制信息：文件所有者、访问信息（用户名和口令等）、合法操作等；
- 使用信息：创建时间、创建者身份、当前状态、最近修改时间、最近访问时间等。

# FCB

文件名	文件标识符
文件结构	文件类型
文件组织	记录长度
当前文件大小	最大文件尺寸
文件设备	物理位置
存取控制	口令
文件建立时间	最近存取时间
最近修改时间	当前存取方式
当前的共享状态	共享访问时的等待状态
进程访问文件所用的逻辑单元号	当前的逻辑位置
访问元素的当前物理位置	下一个元素的物理位置
缓冲区大小	缓冲区地址
指向下一个FCB的指针	文件创建者
临时/永久文件	文件拥有者

# 目录内容的组织方式及分析

- 目录项的两种组织方式：
  - 1. FCB存储全部目录内容
  - 2. 存储部分目录信息，如文件名、索引节点指针等，其余部分保存在索引节点（ $i$ 节点）。打开文件时将索引节点从磁盘读到内存中。

# 目录文件及操作

- 目录文件：一个文件目录也被看做是一个文件，即目录文件。是多个文件的目录项构成的一种特殊文件。
- 目录的操作
  - 搜索目录
  - 创建目录
  - 删除目录
  - 显示目录
  - 修改目录

# 目录结构

- 单级目录结构
- 两级目录结构
- 层次目录结构：树型目录、无循环图



## 7.3.2 单级目录结构

- 所有用户的全部文件目录保存在一张目录表中，每个文件的目录项占用一个表项。

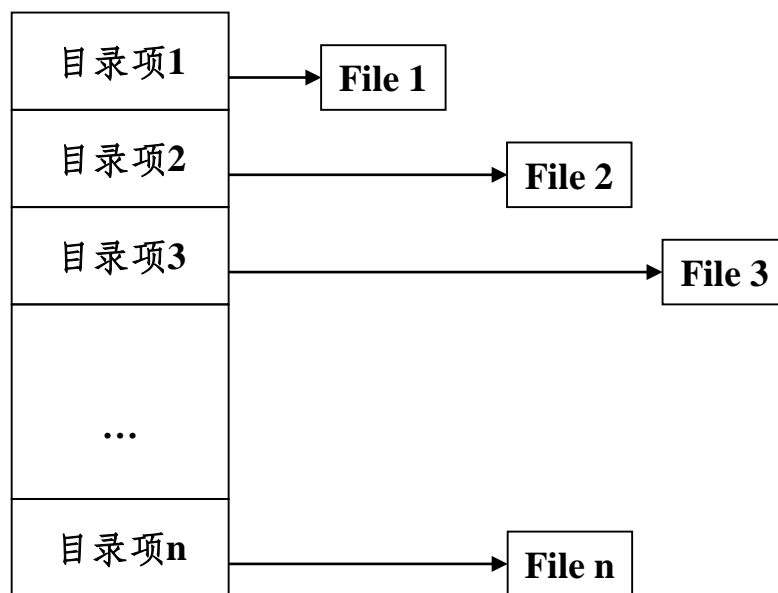


图7-9 单级目录结构

# 单级目录结构分析

- 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
- 存在下述一些缺点：
  - (1) 查找速度慢
  - (2) 不允许重名
  - (3) 不便于实现文件共享

# 两级目录结构

- 主文件目录MFD、用户文件目录UFD

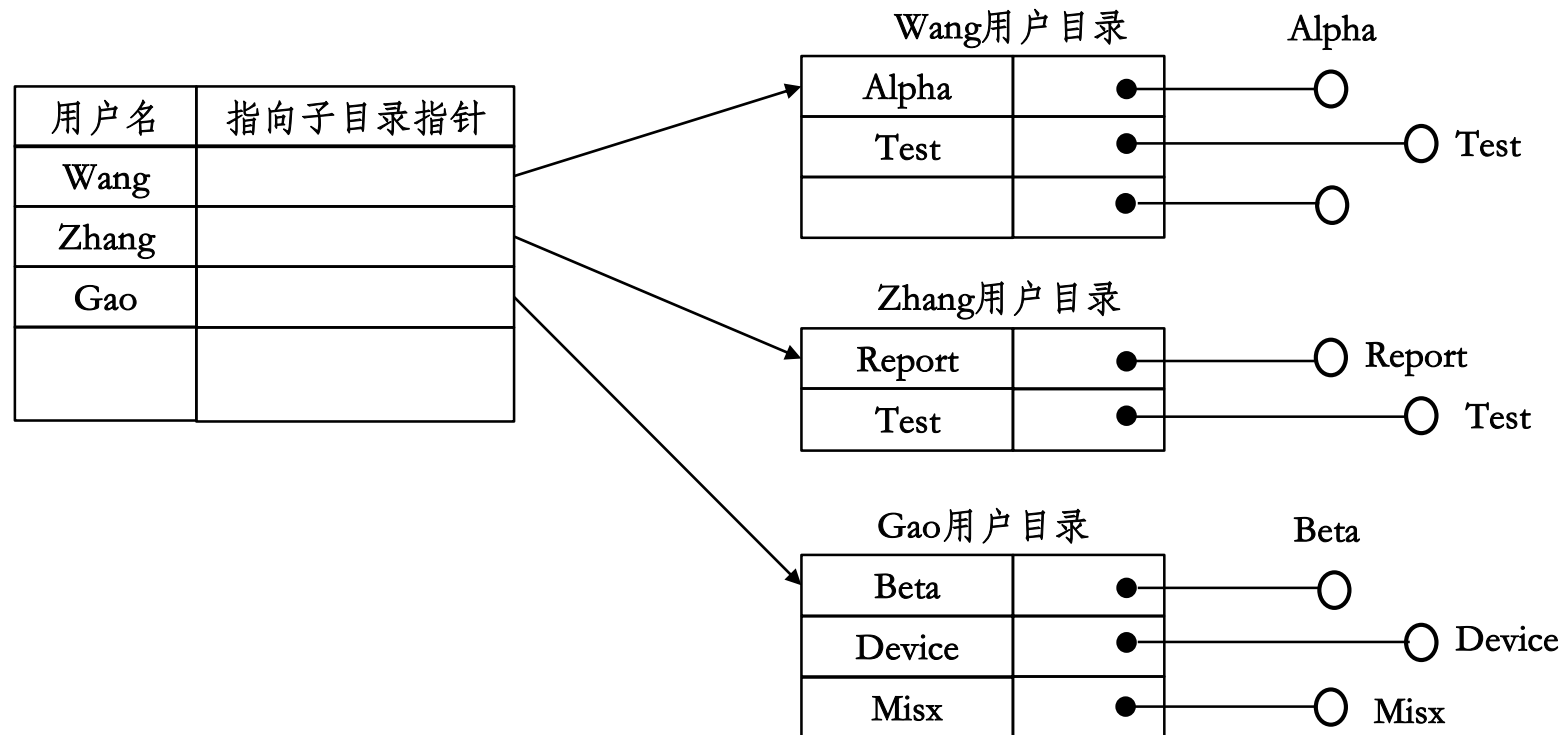


图7-10 两级目录结构

# 两级目录结构分析

- 一定程度解决了重名问题
- 提高了文件目录检索效率
- 简单的文件共享
- **问题：**不便用户文件的逻辑分类；进一步解决重名、共享、检索效率等问题

## 3. 多级目录结构

(1) 目录结构：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。（图7-11）

## 7.3.3 树形结构目录

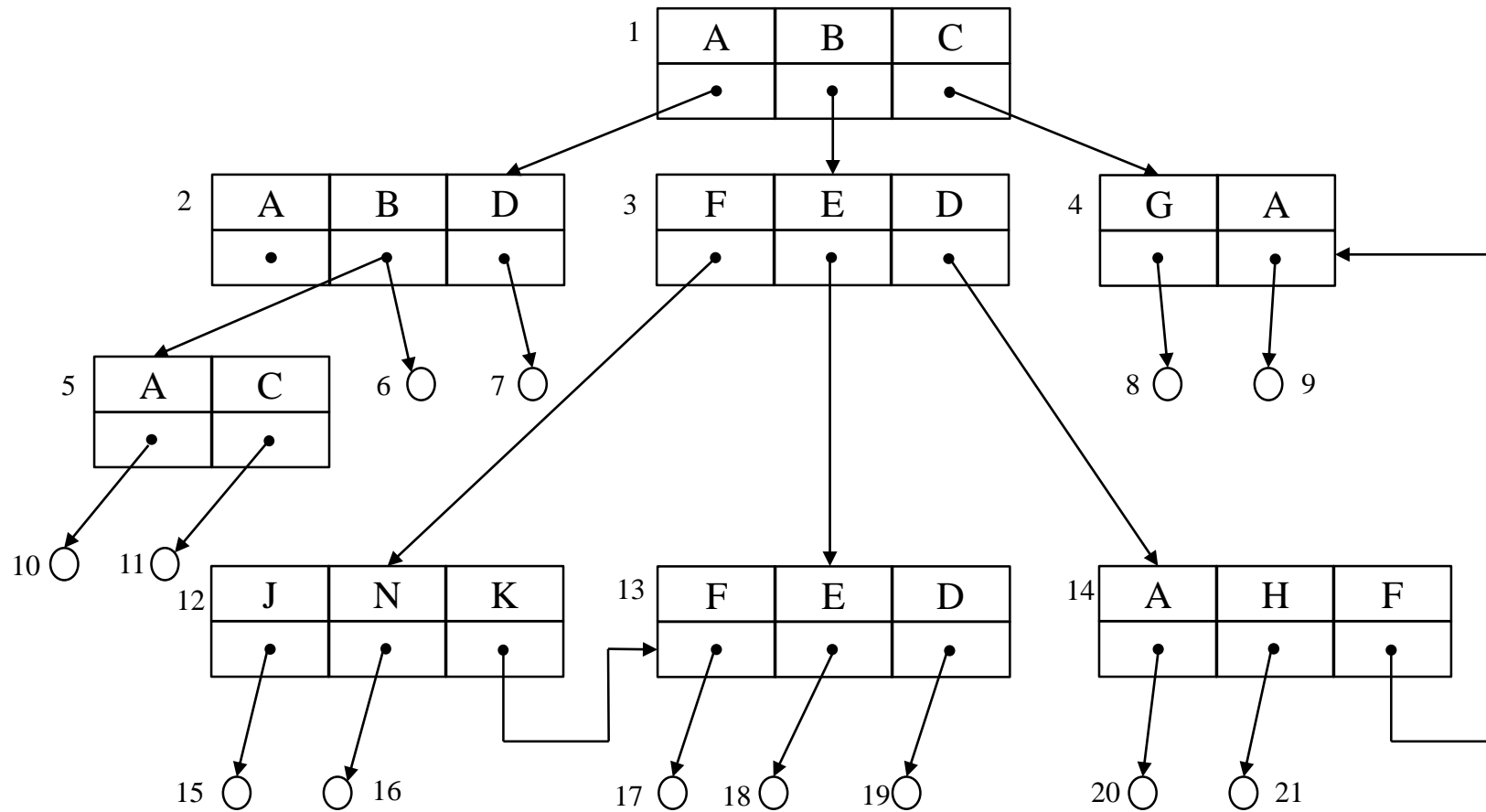


图7-11 多级目录结构

- （2）路径名：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：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。
- 相对路径名
- 绝对路径名



## 4. 增加和删除目录

- **增加目录**：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。
- **目录删除**采用下述两种方法处理：
  - (1) 不删除非空目录。
  - (2) 可删除非空目录。

## 7.3.4 目录查询技术

★对目录进行查询的方式有两种：线性检索法和Hash方法：

### 1. 线性检索法

- 线性检索法又称为顺序检索法。

- ①在单级目录中，利用用户提供的文件名，用顺序查找法直接从文件目录中找到指名文件的目录项。
- ②在树型目录中，用户提供的文件名是由多个文件分量名组成的路径名，此时须对多级目录进行查找。

# 线性检索法

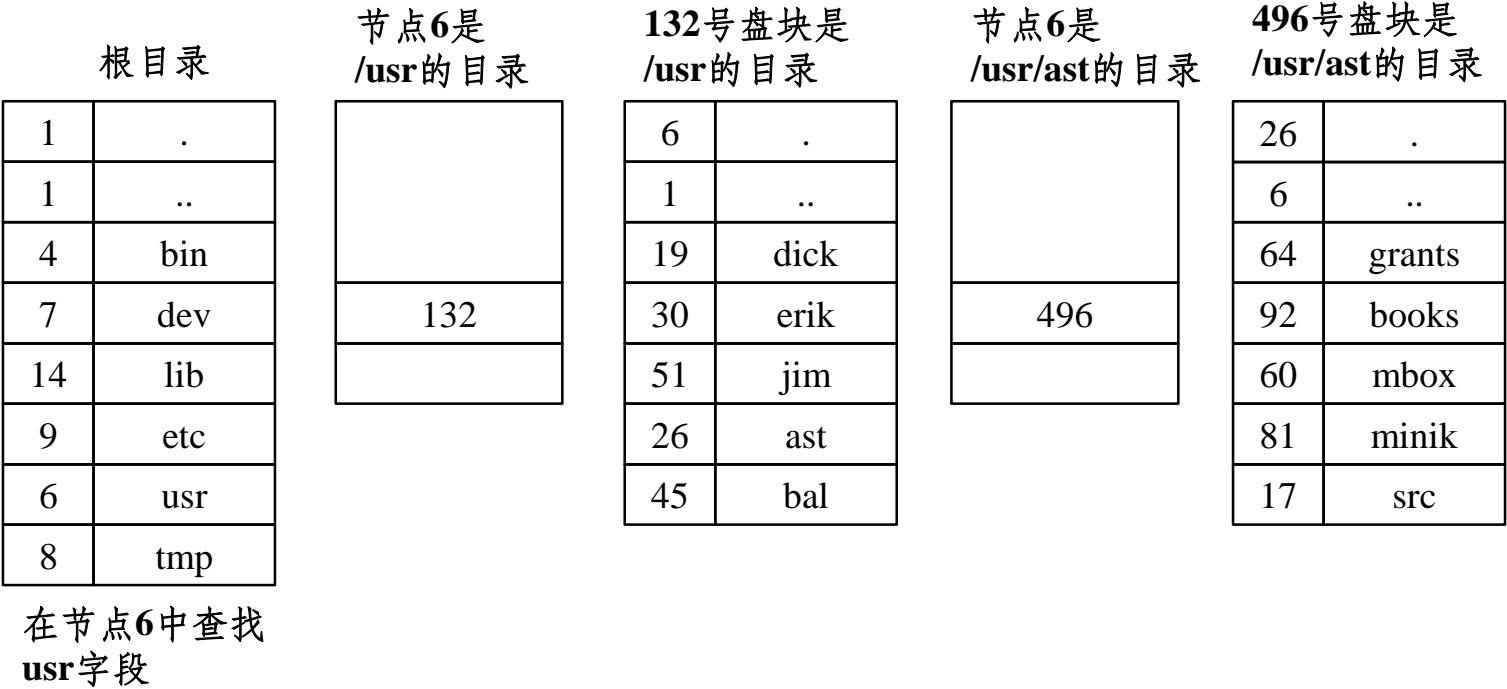


图7-12 查找/usr/ast/mbox的步骤

## 2. Hash方法

- **Hash方法**: 建立了一张Hash索引文件目录，系统利用用户提供的文件名并将它变换为文件目录的索引值，再利用该索引值到目录中去查找。
- Hash方法将显著地提高检索速度。
- 在文件名中使用了通配符“\*”、“?”等，系统便无法利用Hash法检索目录，因此，需要利用线性查找法查找目录。
- 在进行文件名的转换时，有可能把”个不同的文件名转换为相同的Hash值，所谓的“**Hash冲突**”。

## 7.4 文件共享

文件共享的有效控制涉及两个方面：

- 同时存取（Simultaneous Access）
- 存取权限（Access Rights）

- 在树型结构的目录中，当有两个(或多个)用户要共享一个子目录或文件时，必须将共享文件或子目录链接到两个(或多个)用户的目录中，才能方便地找到该文件。此时该文件系统的目录结构已不再是树型结构，而是个有向非循环图。

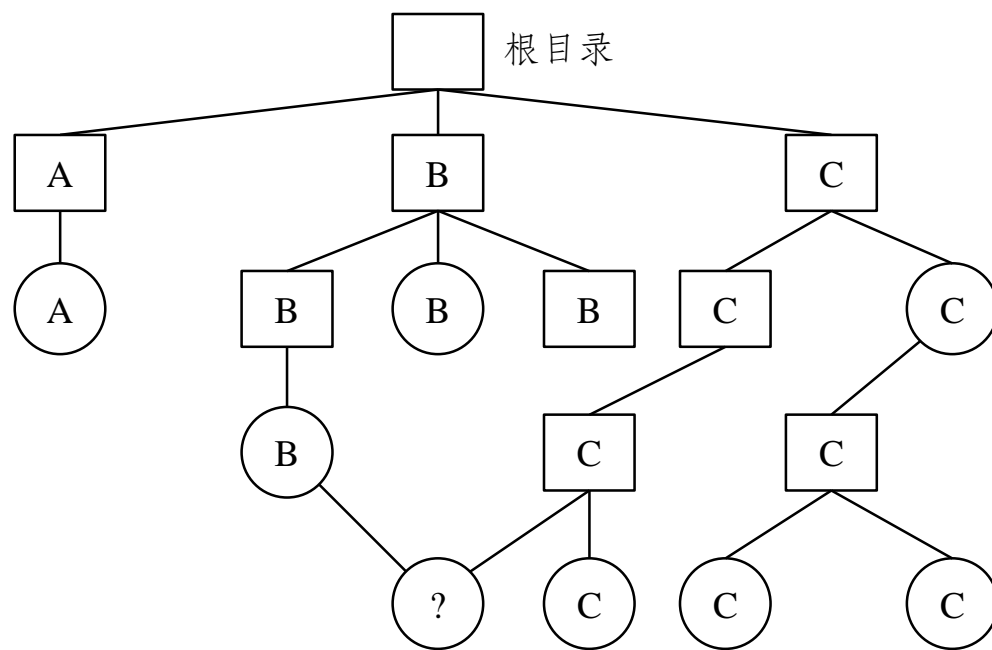


图7-13 包含共享文件的文件系统



# 文件共享的实现

- 实现文件共享的实质就是可以从不同地方打开同一个文件
- 打开文件的首要步骤就是找到文件的目录项，读取文件在外存的起始地址。
- 实现文件共享的方式：
  - 利用链接目录项实现法
  - 利用索引节点实现法
  - 利用符号链实现法等。

# 1.链接目录项实现文件共享

- 文件目录项中设置一个链接指针，用于指向共享文件的目录项。
- 访问文件时，根据链接指针内容找到共享文件的目录项，读取该目录项中文件起始位置等信息，操作该文件。
- 每当有用户（进程）共享文件时，共享文件目录项中的“共享计数”加1；当用户不再共享该文件，撤消链接指针时，“共享计数”减1。
- 只有当共享文件用户数为1时，才能删除共享文件。

# 基于索引节点的共享方式

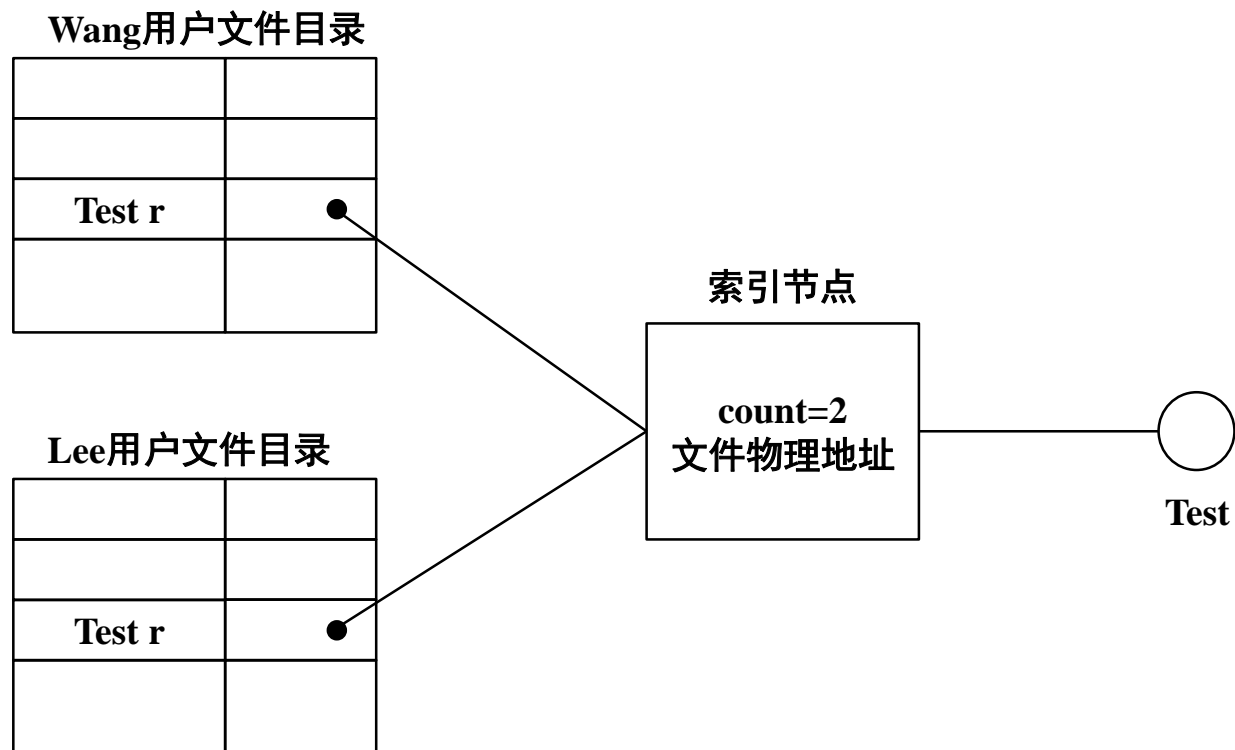


图7-14 基于索引节点的共享方式

## 2.利用索引节点实现文件共享

- 文件的物理地址及其它的文件属性等信息，不再是放在目录项中，而是放在索引结点中。在文件目录中只设置文件名及指向相应索引结点的指针。
- 由任何用户对文件进行Append 操作或修改，所引起的相应结点内容的改变(例如，增加了新的盘块号和文件长度等)，都是其他用户可见的，从而也就能提供给其他用户来共享。



## 2.利用索引节点实现文件共享

- UNIX操作系统的文件目录项中只包含文件名和指向索引节点的指针，文件的物理地址及其它说明信息保存在索引节点中。
- 可以通过共享文件索引节点来共享文件，即当用户需要共享文件时，在自己的文件目录中新建一个目录项，为共享文件命名(也可用原名)，并将索引节点指针指向共享文件的索引节点。

- ❑ 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。

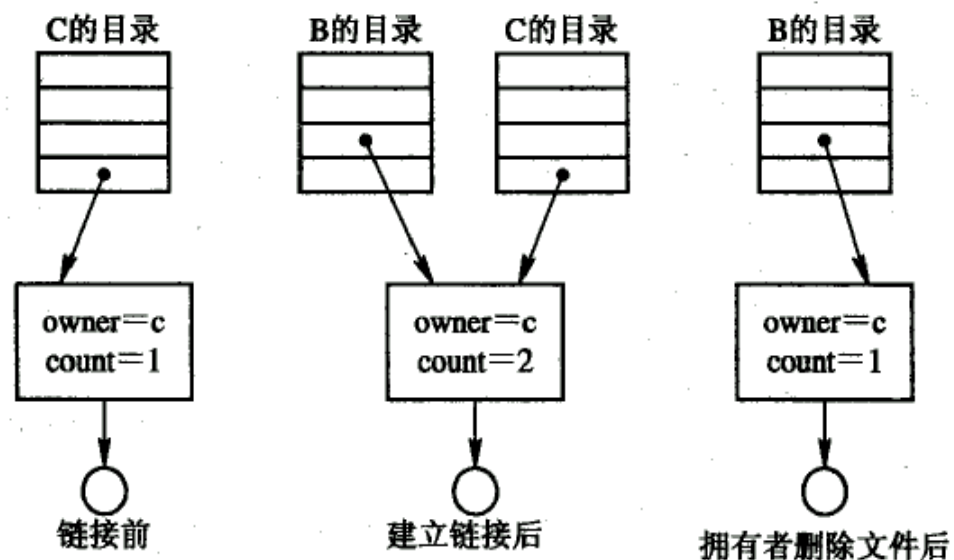


图 6 - 25 进程 B 链接前后的情况



## 7.4.2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，以实现B的目录与文件F的链接；在新文件中只包含被创文件F的路径名。这样的链接方法被称为符号链接。
- 新文件中的路径名，则只被看作是符号链。当B要访问被链接的文件F且正要读LINK类新文件时，将被OS截获，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。

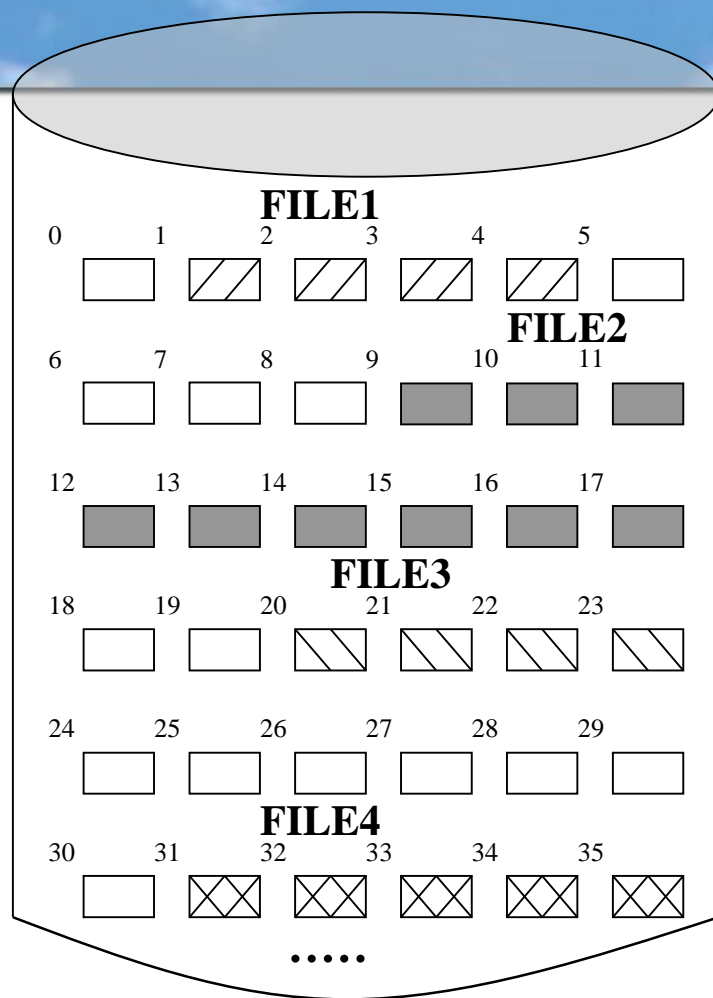
- 在利用符号链方式实现文件共享时，只是文件主才拥有指向其索引结点的指针，而共享该文件的其它用户，则只有该文件的路径名，并不拥有指向其索引结点的指针。
- 符号链方式优点：能连接任何机器上的文件。
- 每增加一个连接，就增加一个文件名，各用户使用自己的名字去共享文件。
- 缺点：备份时可能会产生多个拷贝。

# 8.1 外存分配方式

- 在为文件分配外存空间时所要考虑的主要问题是：怎样才能有效地利用外存空间和如何提高对文件的访问速度。
- 目前，常用的外存分配方法有：
  - 连续分配
  - 链接分配
  - 索引分配

# 1.连续分配

- 连续分配 (Continuous Allocation) 要求为每一个文件分配一组相邻接的盘块。一组盘块的地址定义了磁盘上的一段线性地址。
- 把逻辑文件中的数据顺序地存储到物理上邻接的各个数据块中，这样形成的物理文件可以进行顺序存取。
- 文件目录中为每个文件建立一个表项，其中记载文件的第一个数据块地址及文件长度。
- 对于顺序文件，连续读/写多个数据块内容时，性能较好。



目录表

文件名	起始块号	文件长度
FILE1	1	4
FILE2	9	9
FILE3	20	4
FILE4	31	5
...	...	...

图 磁盘空间的连续分配



- 连续分配的主要优点：

- (1) 顺序访问容易。

- 能很快检索文件中的一个数据块。
- 例如，如果一个文件的第一个数据块的序号为 $x$ ，需要检索文件的第 $y$ 块，则该数据块在外存中的位置为 $x+y-1$ 。

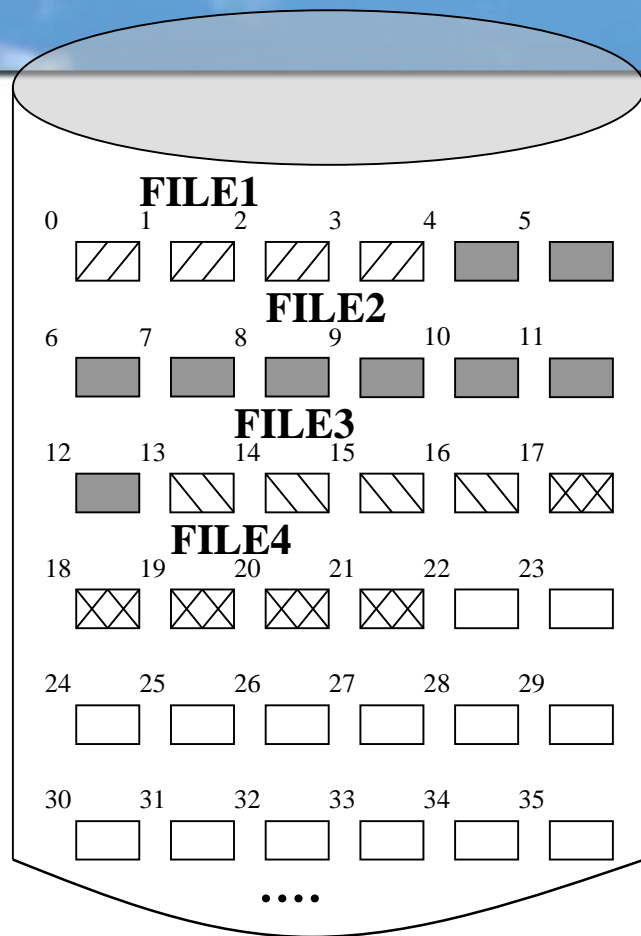
- (2) 顺序访问速度快。

- 磁头移动距离短，效率最高



# 连续分配存在的问题

- (1) 要求有连续的存储空间。
  - 该分配方案可能会导致磁盘碎片，严重降低外存空间的利用率。
  - 解决方法之一，系统定期或不定期采用紧凑技术，将小分区合并为大的、连续分区，**将文件占用空间合并在一起**。
- (2) 必须事先知道文件的长度。
  - 空间利用率不高；
  - 不利于文件尺寸的动态增长。



目录表		
文件名	起始块号	文件长度
FILE1	0	4
FILE2	4	9
FILE3	13	4
FILE4	17	5
...	...	...

图 磁盘 连续分配（紧凑以后）

## 2. 链接分配

- 连续分配的文件分区太大，不利于存储空间的有效利用。
- 如果在将一个逻辑文件存储到外存上时，可以考虑将文件装到多个离散的盘块中。
- **链接文件**：采用链接分配方式时，可通过在每个盘块上的链接指针，将同属于一个文件的多个离散的盘块链接成一个链表，把这样形成的物理文件称为**链接文件**。

- 1) 隐式链接

- 在采用隐式链接分配方式时，在文件目录的每个目录项中，都须含有指向链接文件第一个盘块和最后一个盘块的指针。
- 每个盘块中都含有一个指向下一个盘块的指针。

# 目录

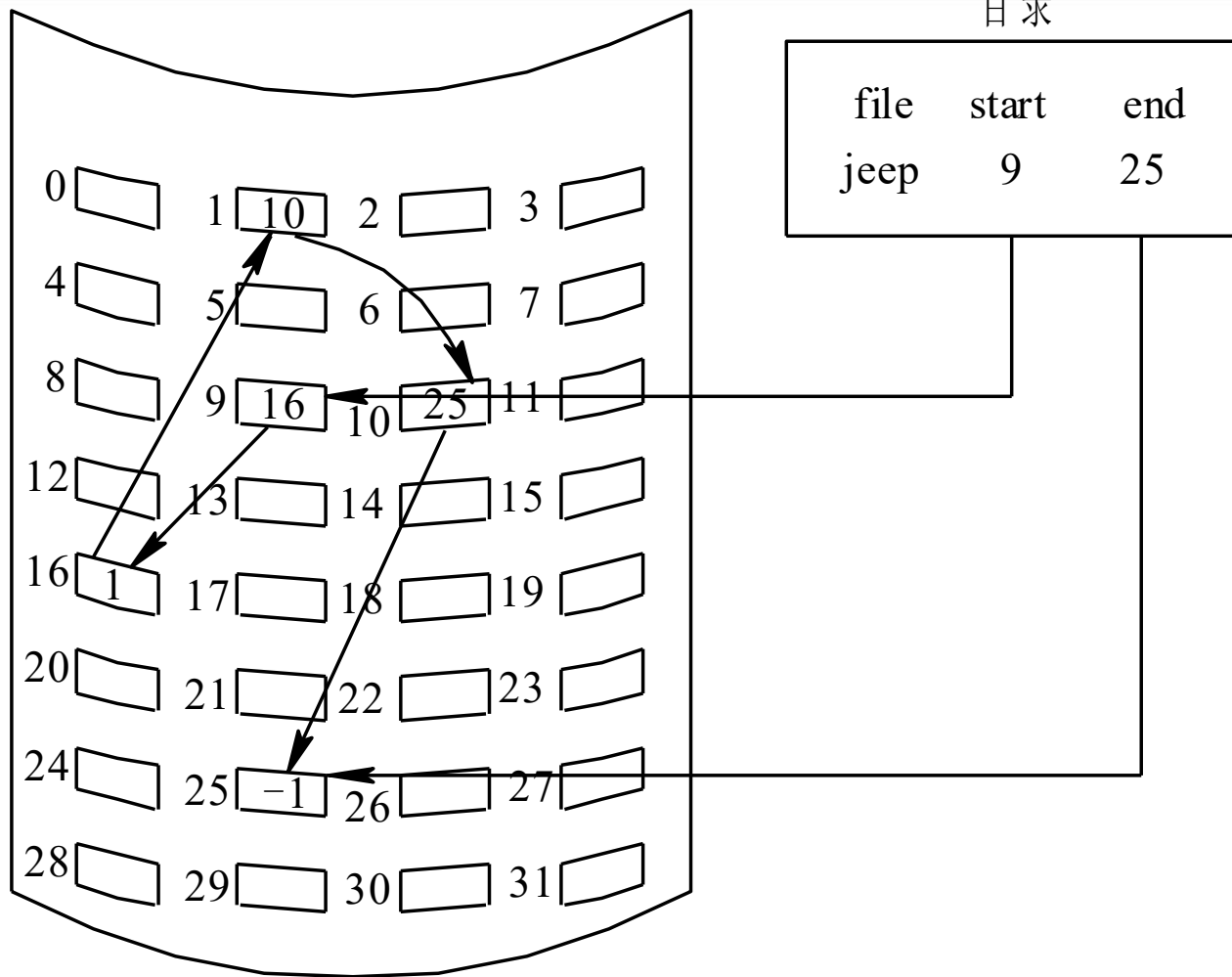


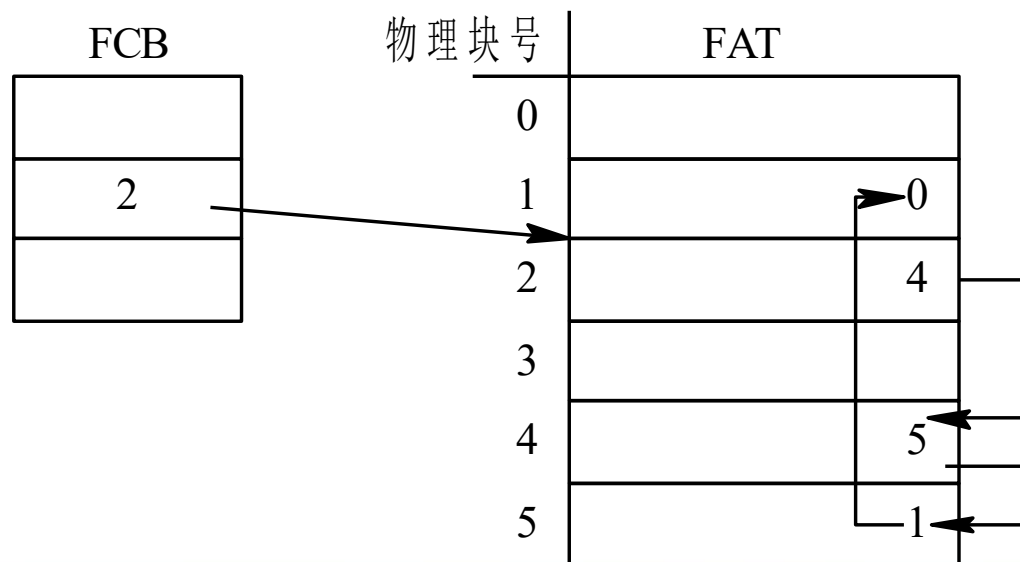
图 磁盘空间的隐式链接式分配

- 隐式链接分配方式的主要问题在于：它只适合于顺序访问，它对随机访问是极其低效的。
  - 如果要访问文件所在的第 $i$ 个盘块，则必须先读出文件的第一个盘块.....，就这样顺序地查找直至第 $i$ 块。
- 为了提高检索速度和减小指针所占用的存储空间，可以将几个盘块组成一个簇(cluster)。
  - 比如，一个簇可包含4个盘块，在进行盘块分配时，是以簇为单位进行的。在链接文件中的每个元素也是以簇为单位的。
  - 减少查找时间和指针所占空间，但增大了内部碎片
- 这种改进也是非常有限的。



- 2) . 显式链接

- 这是指把用于 链接文件各物理块的指针，显式地存放在内存的一张链接表中。
- 整个磁盘仅设置一张文件分配表（FAT）。



- 在该表中，凡是属于某一文件的第一个盘块号，均作为文件地址被填入相应文件的FCB 的“物理地址”字段中。
- 查找记录的过程是在内存中进行的，因而不仅显著地提高了检索速度，而且大大减少了访问磁盘的次数。
- 由于分配给文件的所有盘块号都放在该表中，故把该表称为文件分配表FAT (File Allocation Table)。

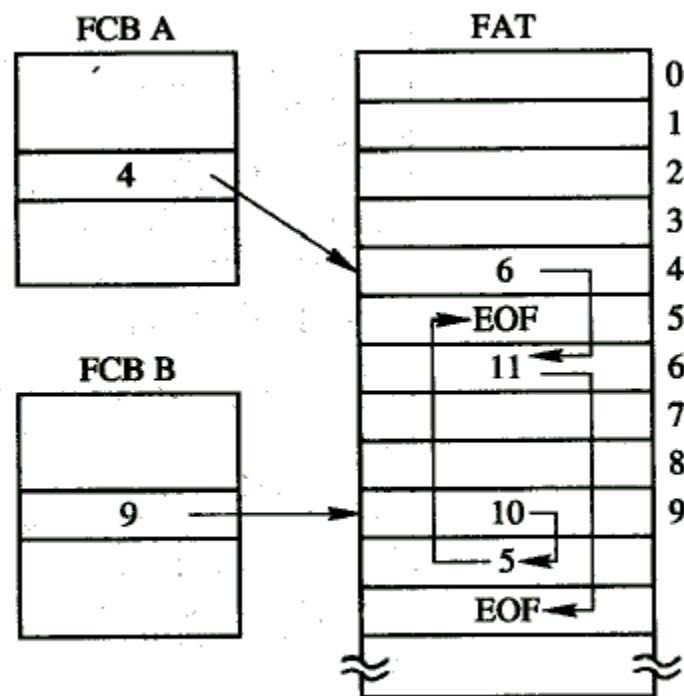


图 6 - 10 MS - DOS 的文件物理结构

□ 自学8.1.3, 8.1.4节, FAT、NTFS技术的链接方法。

## 8.1.5 索引分配

- 链接分配方式虽然解决了连续分配方式所存在的问题，但又出现了另外两个问题，即：
  - (1) 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。
  - (2) FAT需占用较大的内存空间。由于一个文件所占用盘块的盘块号是随机地分布在FAT中的，因而只有将整个FAT调入内存，才能保证在FAT中找到一个文件的所有盘块号。

# 3.索引分配

- 1) 单级索引分配
- 索引分配能解决连续分配和链接分配存在的诸多问题。
- **原理：**为每个文件分配一个索引块(表)，再把分配给该文件的所有盘块号都记录在该索引块中，因而该索引块就是一个含有许多盘块号的数组。
- 在建立一个文件时，只需在为之建立的目录项中填上指向该索引块的指针。

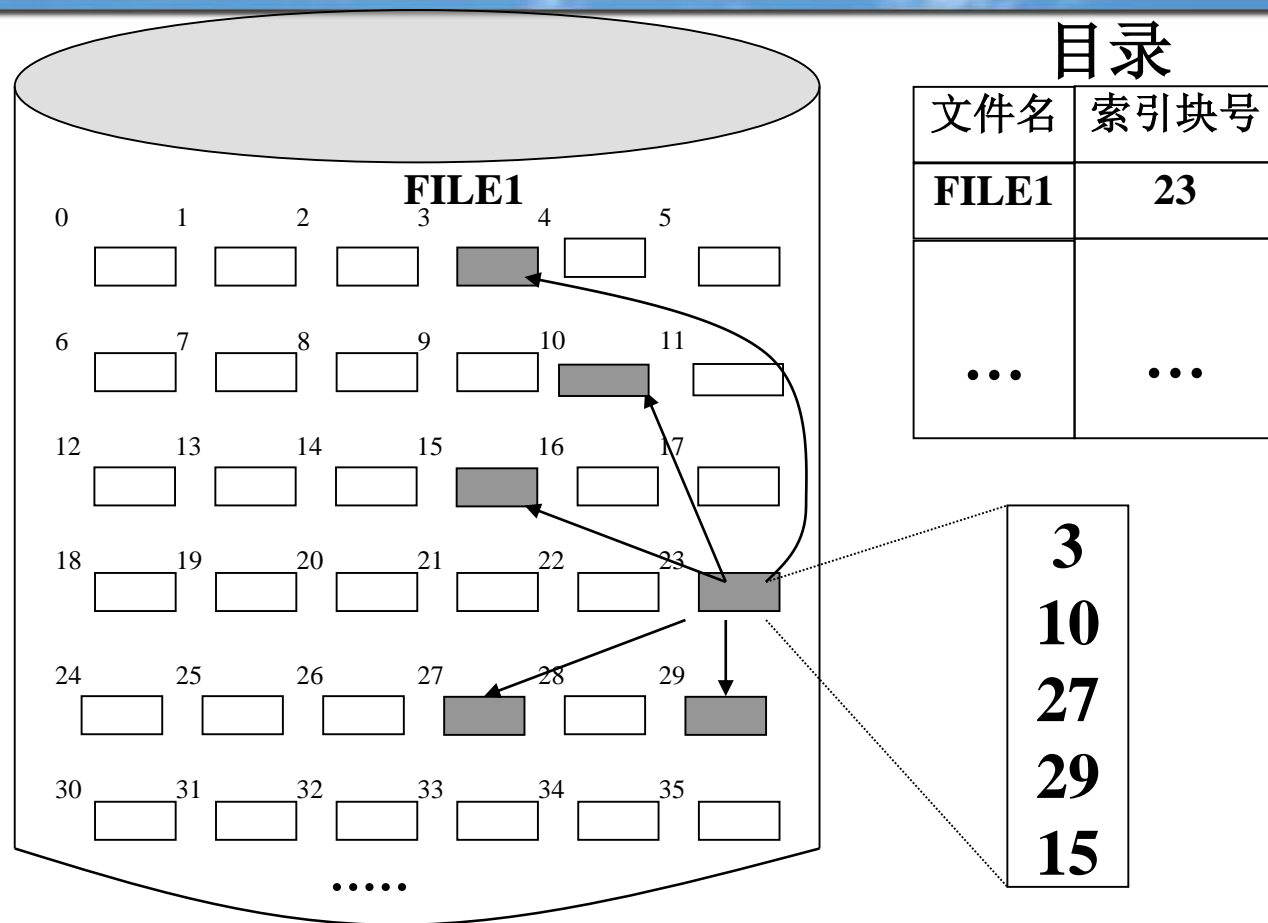


图 基于数据块分区的索引分配



# 索引分配

- 优点：
  - 索引分配方式支持直接访问。当要读文件的第 $i$ 个盘块时，可以方便地直接从索引块中找到第 $i$ 个盘块的盘块号；
  - 基于数据块的分区能消除外部碎片
- 缺点：
  - 大文件索引项较多，可能使一个数据块容纳不了一个文件的所有分区的索引。
  - 索引块可能要花费较多的外存空间。每当建立一个文件时，便须为之分配一个专门的索引块，将分配给该文件的所有盘块号记录于其中。对于小文件如果采用这种方式，索引块的利用率将是极低的。

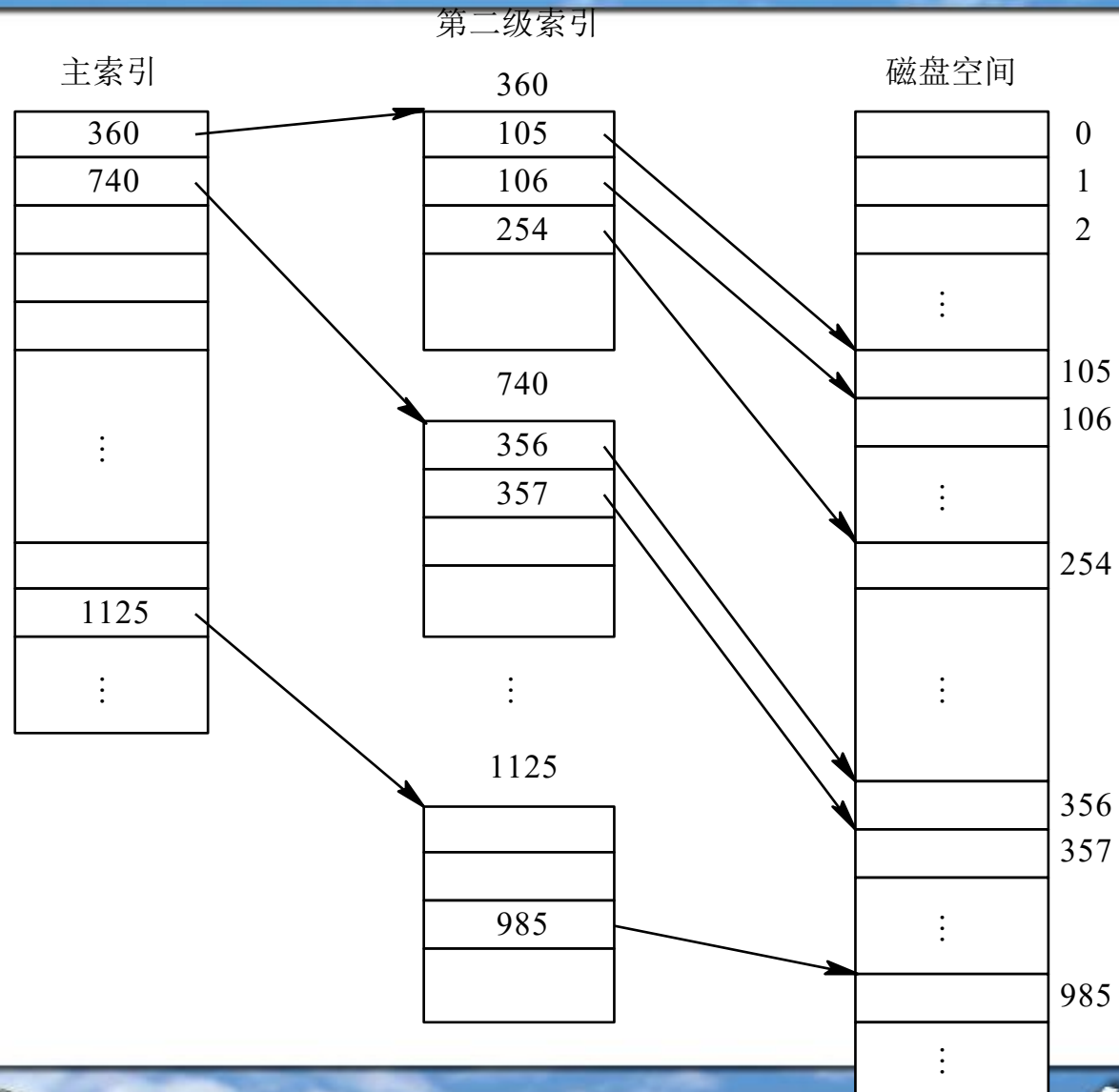
- 2. 两级索引分配

- 当文件太大，其一级索引块太多时，这种方法是低效的。

□ 此时，应为这些索引块再建立一级索引，形成两级索引分配方式。

- 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

# 两级索引分配



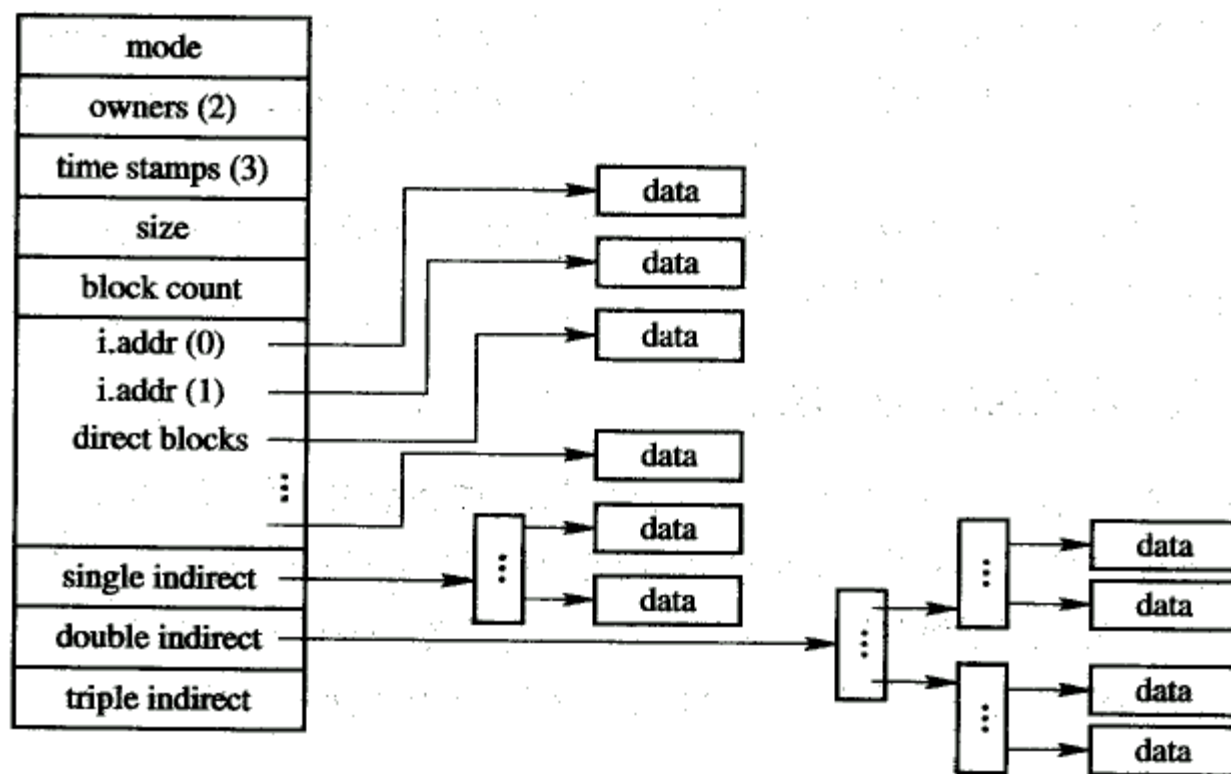


图 6 - 13 混合索引方式

# 第七章 作业

1. 在OS中，引入“打开”这一文件系统调用，“打开”的含义是什么？
2. 试说明对索引文件的检索方法。
3. 基于索引节点的文件共享方式有何优点？
4. 基于符号链的文件共享有何优点？