

第七章 文 件 管 理

7.1 文件和文件系统

7.2 文件的逻辑结构

7.3 文件目录

7.4 文件共享

7.5 文件保护

7.1 文件和文件系统

7.1.1 文件、记录和数据项

1. 数据项

在文件系统中，**数据项**是最低级的数据组织形式，

数据项的类型：

(1) **基本数据项**。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。

基本数据项

学号	姓名	年龄	所在班级
.....

(2) **组合数据项**。由若干个基本数据项组成的，简称**组项**。

例如，工资是个组项，它可由基本工资、工龄工资和奖励工资等基本项所组成。

组合数据项

工号	工资		
	基本工资	工龄工资	奖励工资
.....

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。

学号	姓名	年龄	所在班级
20120001	王小兵	20	信1201-1
.....

记录

在诸多记录中，为了能惟一地标识一个记录，必须在一个记录的各个数据项中，确定出一个或几个数据项，把它们的集合称为**关键字(key)**。关键字是惟一能标识一个记录的数据项。

例如，学号可用来从诸多记录中标识出惟一的一个记录。如果找不到这样的数据项，则把几个数据项定为能在诸多记录中惟一地标识出某个记录的关键字。

关键字

学号	姓名	年龄	所在班级
20120001	王小兵	20	信1201-1
.....

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**和**无结构文件**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。

文件

学号	姓名	年龄	所在班级
20120001	王小兵	20	信1201-1
20120003	李晓华	20	信1201-2
.....



文件应具有自己的**属性**，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

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

通常是用于指示文件在哪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**。

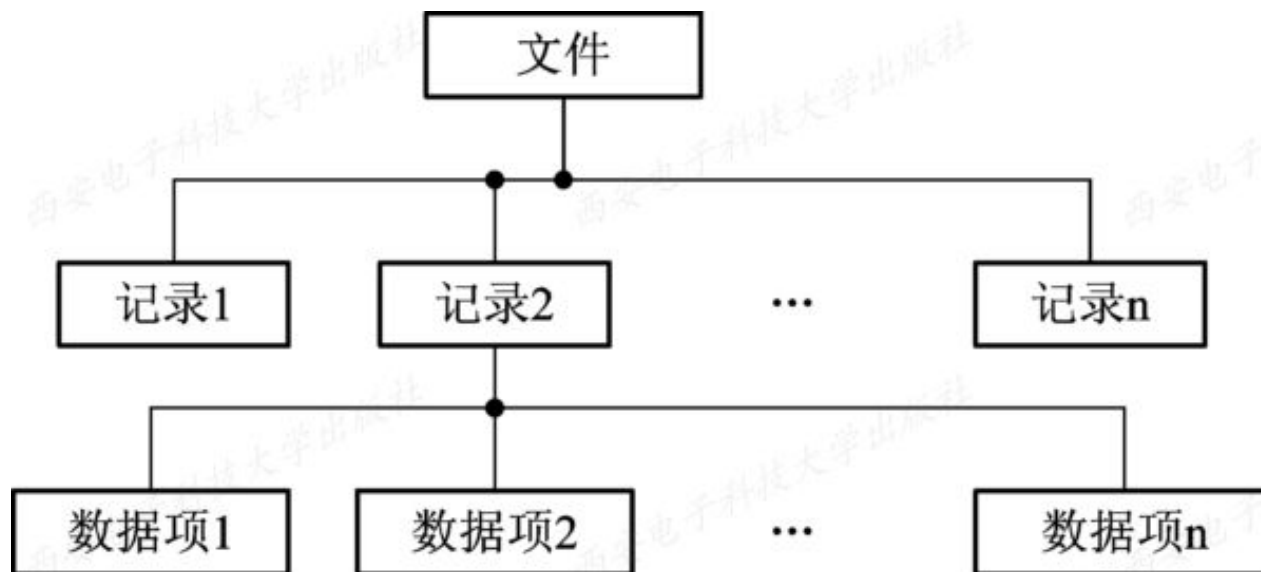


图7-1 文件、记录和数据项之间的层次关系



7.1.2 文件名和类型

1. 文件名和扩展名

(1) 文件名。

(2) 扩展名。



2. 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。



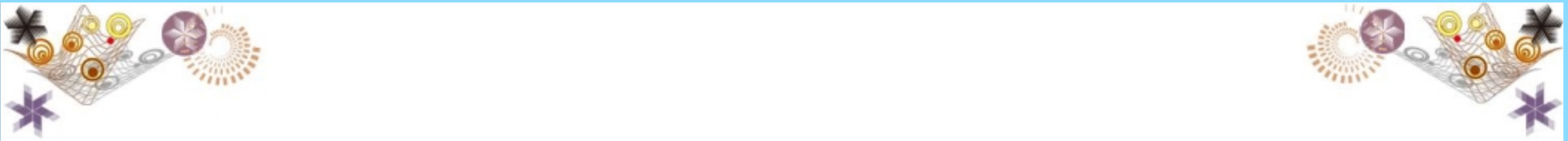
2) 按文件中数据的形式分类

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。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经过链接程序链接后所形成的文件。其后缀名是 .exe。



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，不允许读和写。

(2) **只读文件**，该类文件只允许文件主及被核准的用户去读，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处理方式的不同，可将文件分为三类：

- (1) 普通文件。
- (2) 目录文件。
- (3) 特殊文件。

4) 按组织形式和处理方式分类

(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。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各种文件操作。

(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。

根据设备数据交换单位的不同，又可将特殊文件分为

★ **块设备文件**：用于磁盘、光盘或磁带等块设备的I/O 操作。

★ **字符设备文件**：用于终端、打印机等字符设备的I/O 操作。

Linux设备文件

7.1.3 文件系统的层次结构

如图7-2所示，文件系统的模型可分为三个层次：最底层是对象及其属性，中间层是对对象进行操纵和管理的软件集合，最高层是文件系统提供给用户的接口。

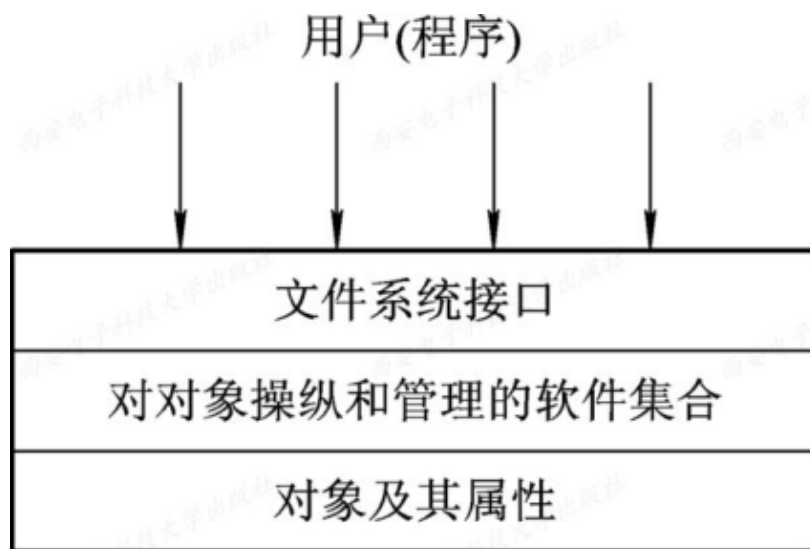


图7-2 文件系统模型



1. 对象及其属性

文件管理系统管理的对象如下：

(1) **文件**：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，每个目录项中，必须含有文件名及该文件所在的物理地址(或指针)。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

该层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，其中包括有：


- ① 对文件存储空间的管理；
- ② 对文件目录的管理；
- ③ 用于将文件的逻辑地址转换为物理地址的机制；
- ④ 对文件读和写的管理；
- ⑤ 对文件的共享与保护等功能。



一般把与文件系统有关的软件分为四个层次：

- 1) I/O控制层：文件系统的最低层
- 2) 基本文件系统层：处理内存与磁盘之间的数据块的交换
- 3) 基本I/O管理程序：用于完成与磁盘I/O有关的事务
- 4) 逻辑文件系统：用于处理与记录和文件相关的操作。





3. 文件系统的接口

(1) 命令接口，是指作为用户与文件系统直接交互的接口，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，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**取得文件系统的服务。



7.1.4 文件操作

1. 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

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

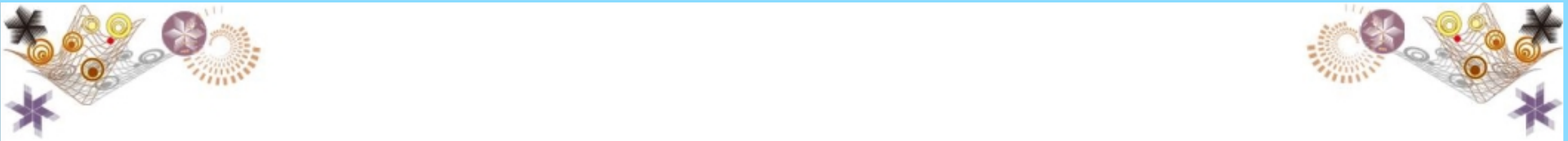
在读一个文件时，须在相应系统调用中给出文件名和应读入的内存目标地址。此时，系统同样要查找目录，找到指定的目录项，从中得到被读文件在外存中的位置。在目录项中，还有一个指针用于对文件的读/写。

(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) 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。改顺序存取为随机存取。



2. 文件的“打开”和“关闭”操作



当前OS所提供的大多数对文件的操作，其过程大致都是这样两步：

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，便可利用系统所返回的索引号向系统提出操作请求。

系统可**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。这样不仅节省了大量的检索开销，也显著地提高了对文件的操作速度。



◆ 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。



3. 其它文件操作

OS都提供了有关文件操作的系统调用，这些调用分成：

◆ **有关对文件属性进行操作的**，即允许用户直接设置和获得文件的属性，如：

- 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
- 以及查询文件的状态(包括文件类型、大小和拥有者以及对文件的访问权等)；



◆ 有关目录的：

- 创建一个目录，
- 删除一个目录，
- 改变当前目录和工作目录等；

此外，还有用于实现文件共享的系统调用和用于对文件系统进行操作的系统调用等。

7.2 文件的逻辑结构

- (1) 文件的逻辑结构(File Logical Structure)。
- (2) 文件的物理结构，又称为文件的存储结构。



7.2.1 文件逻辑结构的类型

1. 按文件是否有结构分类

1) 有结构文件

(1) 定长记录：文件中所有记录的长度都是相同的。

(2) 变长记录：文件中各记录的长度不相同。

★大量的信息管理系统和数据库是采用有结构的文件形式

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。采用读/写指针来指出下一个要访问的字符。




2. 按文件的组织方式分类

根据文件的组织方式，可把有结构文件分为三类：

(1) 顺序文件：由一系列记录按照某种**顺序排列**所形成的文件。

(2) 索引文件：为可变长记录文件**建立一张索引表**，为每个记录设置一个表项，以加速对记录的检索速度。

(3) 索引顺序文件：顺序文件和索引文件相结合的产物。为每个文件建立一张索引表时，**只为一组记录中的第一个记录**建立一个索引项。



7.2.2 顺序文件(Sequential File)

1. 顺序文件的排列方式

在顺序文件中的记录，可以按照各种不同的顺序进行排列。一般地，可分为两种情况：

(1) 串结构。

各记录之间的顺序与关键字无关。**由时间来决定记录之间的顺序**，即按存入时间的先后排列，最先存入的记录作为第一个记录，其次存入的为第二个记录.....，依此类推。

(2) 顺序结构。

指**文件中的所有记录按关键字排列**。

对顺序结构文件，则可**利用某种有效的查找算法**，如折半查找法、插值查找法、跳步查找法等方法来提高检索效率。



2. 顺序文件的优缺点

顺序文件的最佳应用场合是在对文件中的记录进行批量存取时(即每次要读或写一大批记录)。所有逻辑文件中顺序文件的存取效率是最高的。此外, 对于顺序存储设备(如磁带), 也只有顺序文件才能被存储并能有效地工作。

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。这时, 顺序文件所表现出来的性能就可能很差, 尤其是当文件较大时, 情况更为严重。

顺序文件的另一个缺点是, 如果想增加或删除一个记录都比较困难。

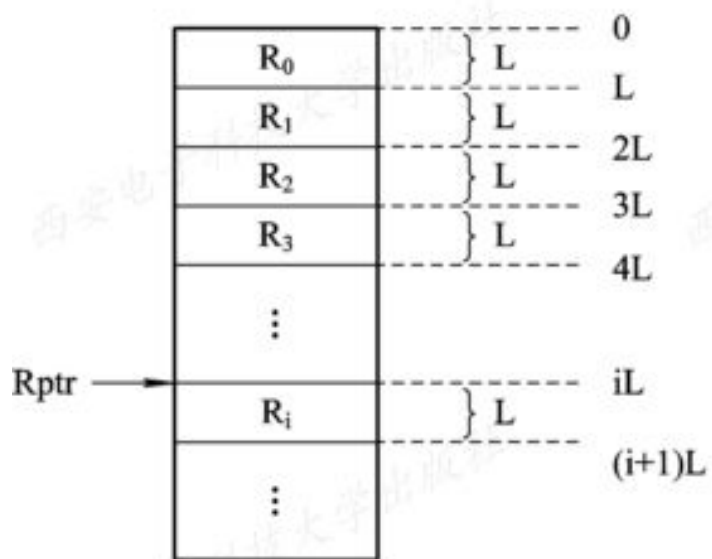


7.2.3 记录寻址

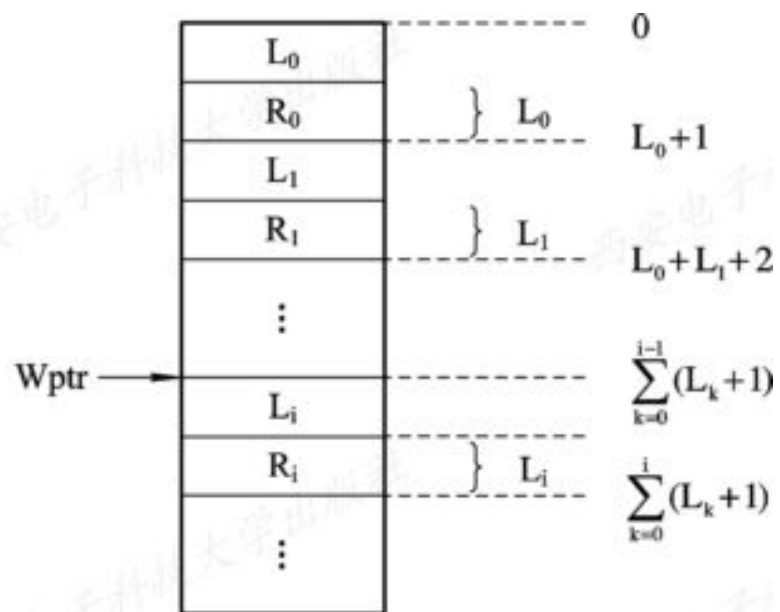
1. 隐式寻址方式

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，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 $Rptr := Rptr + L$ 。L为记录长度。

对于变长记录的顺序文件，在顺序读或写时的情况相似，但应分别为它们设置读或写指针，在每次读或写完一个记录后，须将读或写指针加上 L_i 。 L_i 是刚读或刚写完的记录的长度。



(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



2. 显式寻址方式

该方式可用于对定长记录的文件实现**直接或随机访问**。因为任何记录的位置都很容易通过记录长度计算出来。而对于可变长度记录的文件则不能利用显式寻址方式实现直接或随机访问，必须增加适当的支持机构方能实现。下面我们通过两种方式对定长记录实现随机访问：

- (1) 通过文件中记录的位置。
- (2) 利用关键字。



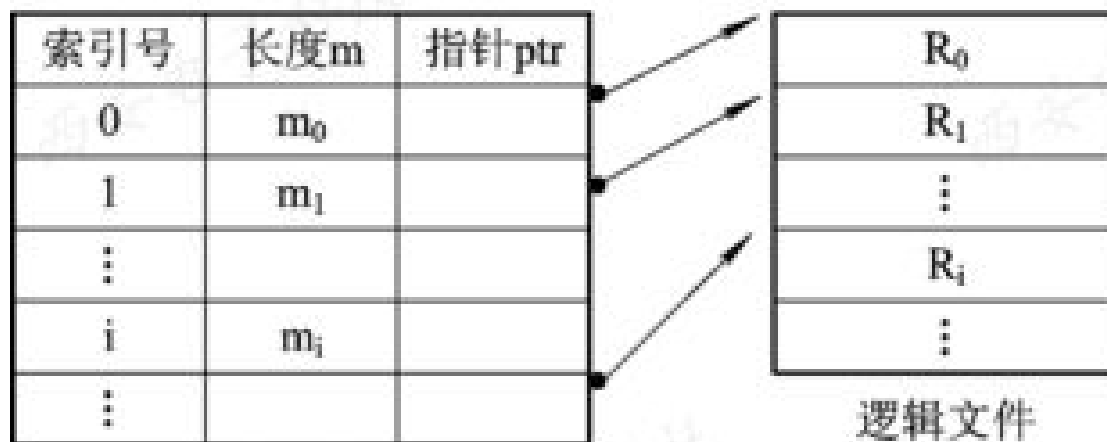
7.2.4 索引文件(Index File)

1. 按关键字建立索引

定长记录的文件可以通过简单的计算，很容易地实现随机查找。但**变长记录文件**查找一个记录必须从第一个记录查起，一直顺序查找到目标记录为止，耗时很长。

可以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。**

由于索引表是按记录键排序的，**索引表本身是一个定长记录的顺序文件**，可以方便地实现直接存取。



索引表

逻辑文件

(a) 具有单个索引表的索引文件

图7-4 具有单个和多个索引表的索引文件



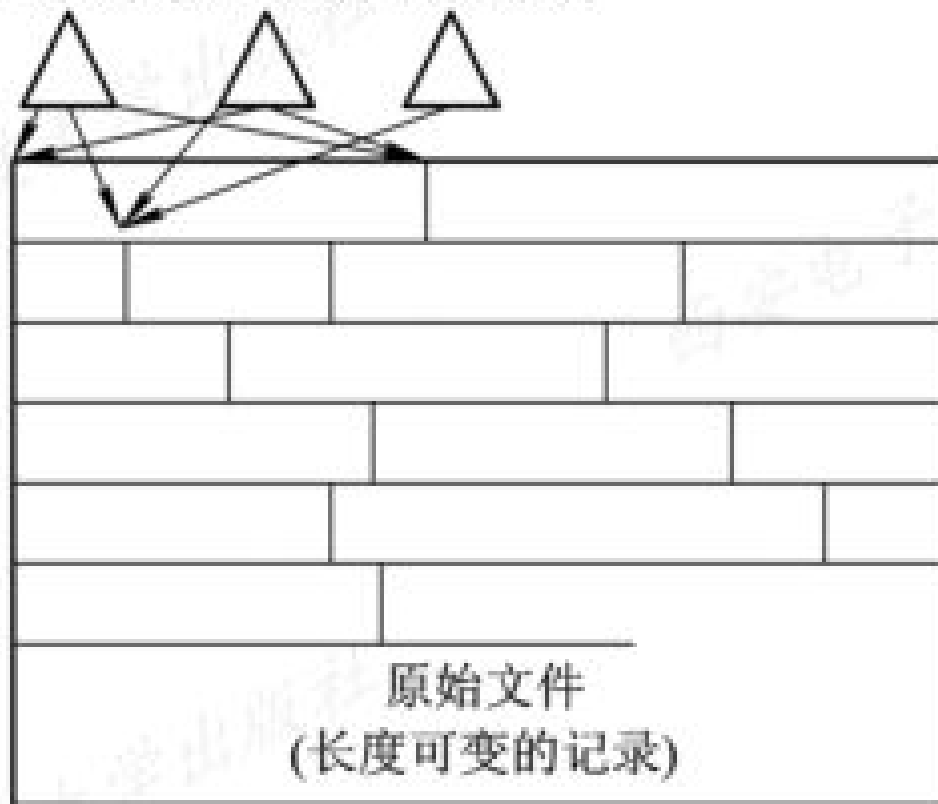
2. 具有多个索引表的索引文件

使用按关键字建立索引表的索引文件与顺序文件一样，都只能按该关键字进行检索。而实际应用情况往往是：不同的用户，为了不同的目的，希望能按不同的属性(或不同的关键字)来检索一条记录。

为实现此要求，需要**为顺序文件建立多个索引表**，即为每一种可能成为检索条件的域(属性或关键字)都配置一张索引表。在每一个索引表中，都按相应的一种属性或关键字进行排序。



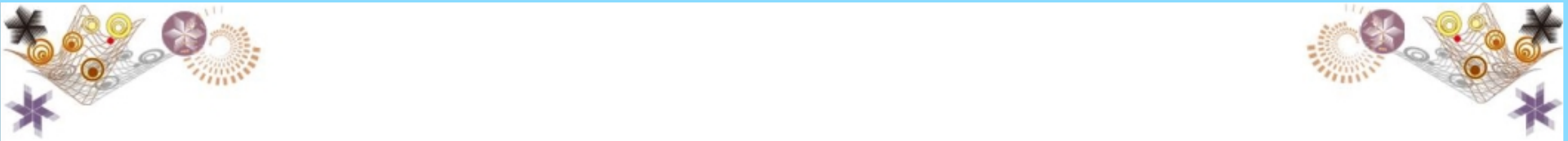
完全索引 完全索引 部分索引



(b) 具有多个索引表的索引文件

图7-4 具有单个和多个索引表的索引文件





7.2.5 索引顺序文件 (Index Sequential File)

1. 索引顺序文件的特征

索引顺序文件是对顺序文件的一种改进，它基本上克服了变长记录的顺序文件不能随机访问，以及不便于记录的删除和插入的缺点。但它仍保留了顺序文件的关键特征，即记录是按关键字的顺序组织起来的。

它又增加了两个新特征：一个是**引入了文件索引表**，通过该表可以实现对索引顺序文件的随机访问；另一个是**增加了溢出(overflow)文件**，用它来记录新增加的、删除的和修改的记录。

2. 一级索引顺序文件

最简单的索引顺序文件只使用了一级索引。其具体的建立方法是，首先将变长记录顺序文件中的所有记录分为若干个组，如50个记录为一个组。然后为顺序文件建立一张索引表，并为每组中的第一个记录在索引表中建立一个索引项，其中含有该记录的关键字和指向该记录的指针。索引顺序文件是最常见的一种逻辑文件形式，如图7-5所示。

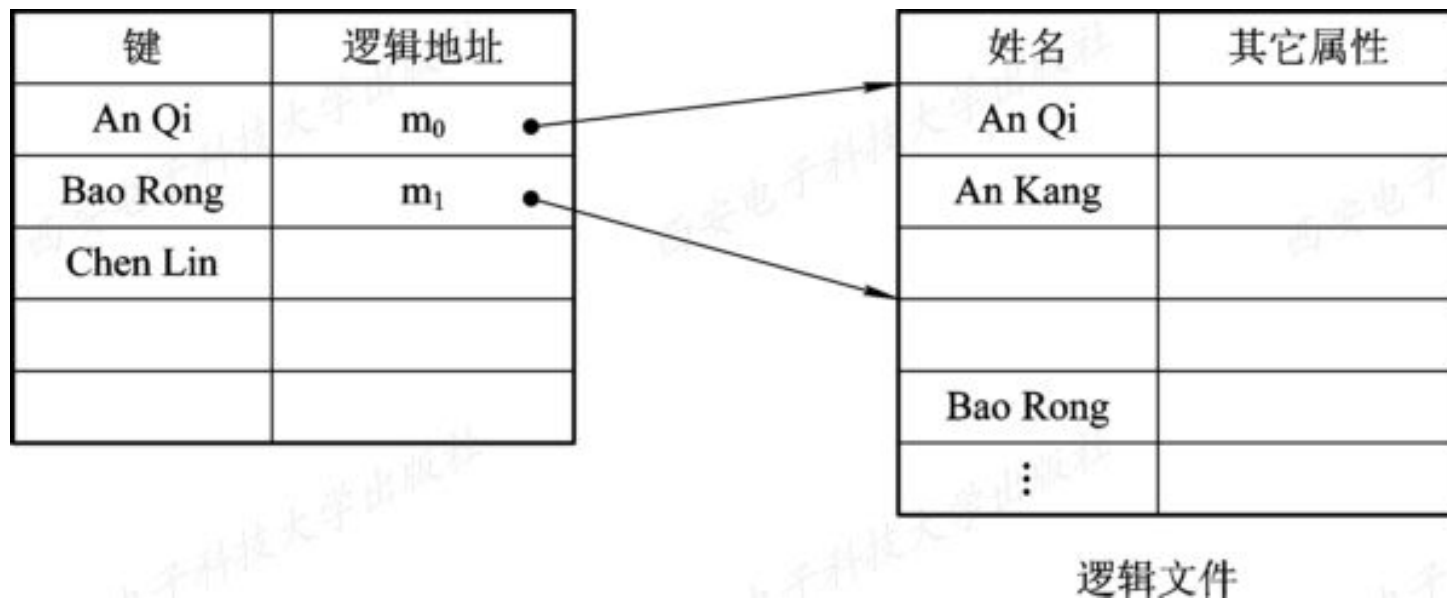


图7-5 索引顺序文件



3. 两级索引顺序文件



对于一个非常大的文件，为找到一个记录而须查找的记录数目仍然很多，例如，对于一个含有 10^6 个记录的顺序文件，当把它作为索引顺序文件时，为找到一个记录，平均须查找1000个记录。为了进一步提高检索效率，可以**为顺序文件建立多级索引**，即**为索引文件再建立一张索引表**，从而形成**两级索引表**。



7.2.6 直接文件和哈希文件

1. 直接文件

对于直接文件，则可根据给定的记录键值，直接获得指定记录的物理地址。换言之，记录键值本身就决定了记录的物理地址。这种由记录键值到记录物理地址的转换被称为键值转换(Key to address transformation)。组织直接文件的关键，在于用什么方法进行从记录值到物理地址的转换。



2. 哈希 (Hash) 文件

这是目前应用最为广泛的一种直接文件。它利用**Hash**函数(或称散列函数)可将关键字转换为相应记录的地址。但为了能够实现文件存储空间的动态分配,通常由Hash函数所求得的并非是相应记录的地址,而是**指向某一目录表相应表目的指针**,该表目的内容指向相应记录所在的物理块,如图7-6所示。

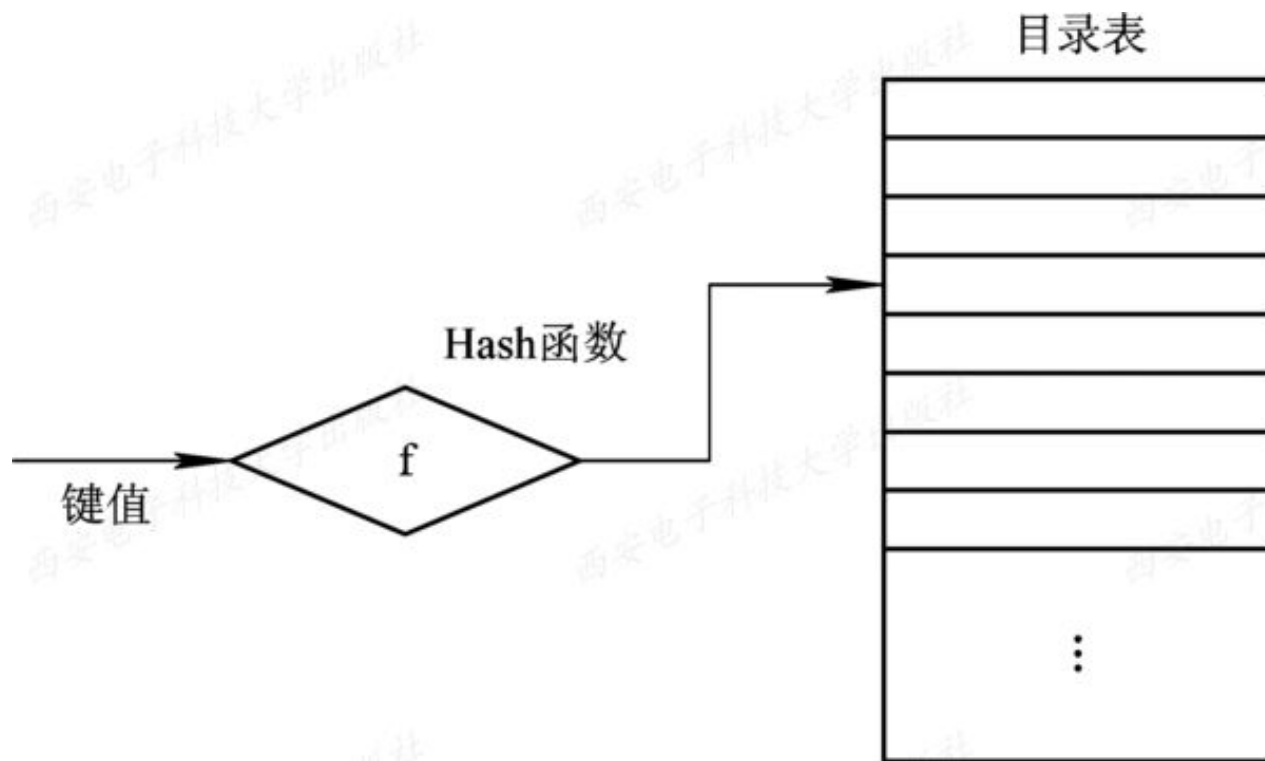


图7-6 Hash文件的逻辑结构

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。



7.3.1 文件控制块和索引结点

为文件设置用于描述和控制文件的数据结构——FCB

文件与文件控制块一一对应，文件控制块的有序集合称为文件目录。

1. 文件控制块FCB (File Control Block)

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

- ① **文件名**：指用于标识一个文件的符号名。在每个系统中，每一个文件都必须有惟一的名称，用户利用该名称进行存取。
- 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

包括：

- 文件主的存取权限
- 核准用户的存取权限
- 一般用户的存取权限



3) 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
- 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)。

对于不同OS的文件系统，由于功能不同，可能只含有上述信息中的某些部分。



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。

图7-7 MS-DOS的文件控制块



2. 索引结点

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，必须先将存放目录文件的第一个盘块中的目录调入内存，然后将用户所给定的文件名，与目录项中的文件名逐一比较。若未找到指定文件，还需要将下一盘块的目录项调入内存。

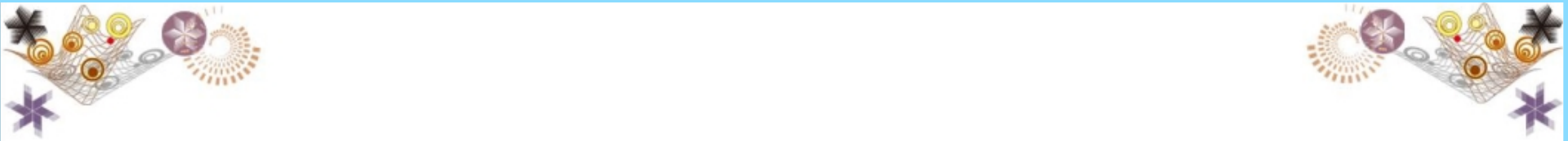
在UNIX系统，使文件描述信息单独形成一个称为索引结点的数据结构，简称为i结点。

指向i结点

文件名	索引结点编号
文件名 1	
文件名 2	
...	...

0 13 14 15

图7-8 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。每个文件有唯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 $iaddr(0) \sim iaddr(12)$ ，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。

3) 内存索引结点

这是存放在内存中的索引结点。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，便于以后使用。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。



7.3.2 简单的文件目录

1. 单级文件目录

这是最简单的文件目录。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。单级文件目录如图7-9所示。



文件名	扩展名	文件长度	物理地址	文件类型	文件说明	状态位	
文件名 1							
文件名 2							
文件名 3							

图7-9 单级文件目录







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它说明信息，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，





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。

重名问题在多道程序环境下却又是难以避免的；即使在单用户环境下，当文件数超过数百个时，也难于记忆。



(3) 不便于实现文件共享。

通常，每个用户都有自己的名字空间或命名习惯。因此，应当允许不同用户使用不同的文件名来访问同一个文件。然而，单级目录却要求所有用户都用同一个名字来访问同一文件。

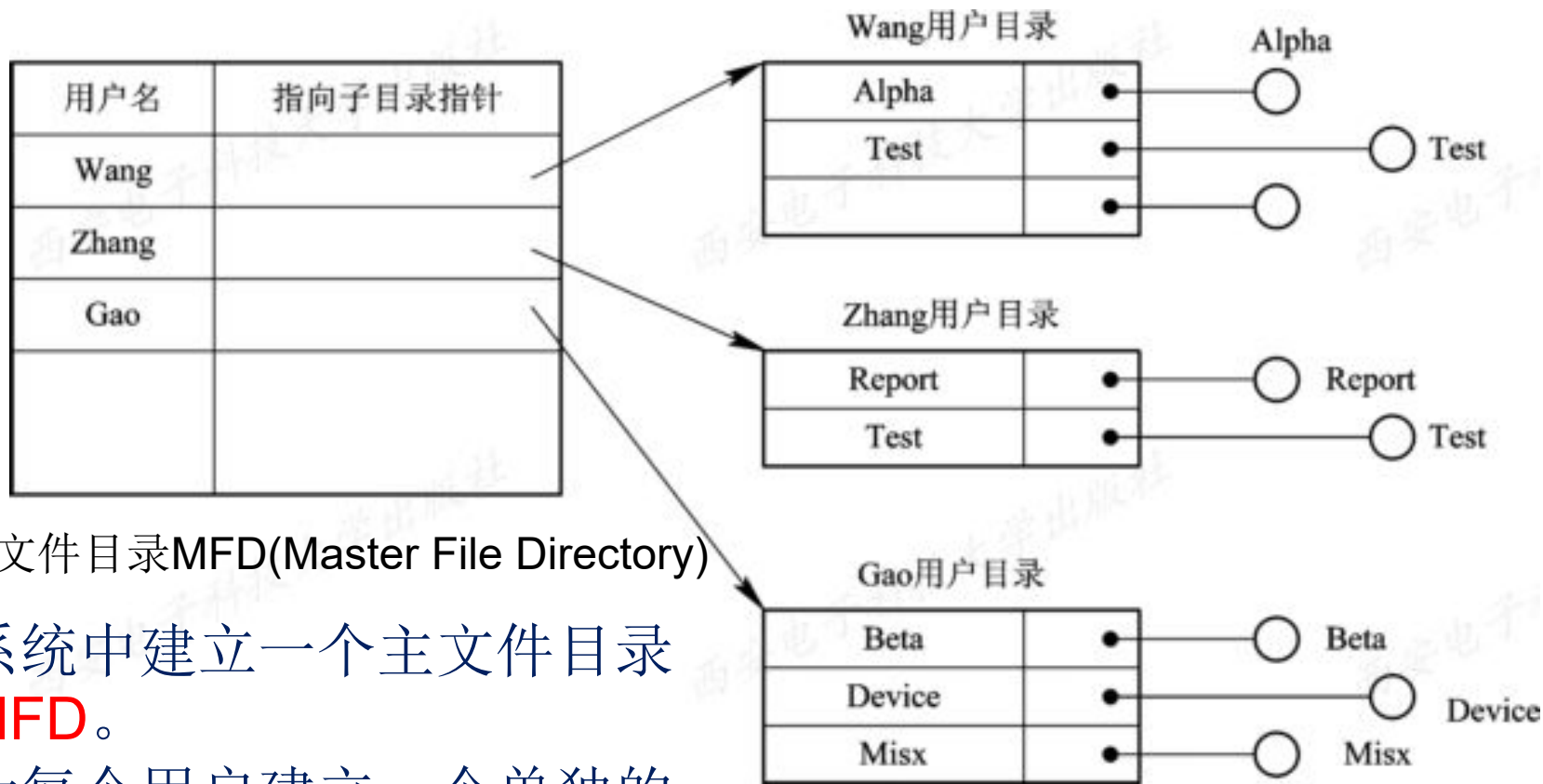
单级目录只能满足对目录管理的四点要求中的第一点，因而，它只能适用于单用户环境。

2. 两级文件目录

为了克服单级文件目录所存在的缺点，可以为**每一个用户再建立一个单独的用户文件目录UFD(User File Directory)**。这些文件目录具有相似的结构，它由用户所有文件的文件控制块组成。

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。

用户文件目录UFD(User File Directory)





主文件目录MFD(Master File Directory)

系统中建立一个主文件目录
MFD。

为每个用户建立一个单独的
用户文件目录**UFD**。

图7-10 两级文件目录



两级目录结构优点：

(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，则为查找一指定的目录项，最多只需检索 $n + m$ 个目录项。

但如果是采用单级目录结构，则最多需检索 $n \times m$ 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。

采用两级目录结构也存在一些问题。该结构虽然能有效地将多个用户隔开，但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，这种隔离便成为一个缺点，因为这种隔离会使诸用户之间不便于共享文件。





7.3.3 树形结构目录(Tree-Structured Directory)

1. 树形目录

在现代OS中，最通用且实用的文件目录无疑是树形结构目录。它可以明显地提高对目录的检索速度和文件系统的性能。主目录在这里被称为**根目录**，在每个文件目录中，只能有一个根目录，每个文件和每个目录都只能有一个父目录。把**数据文件称为树叶**，其它的目录均作为树的**结点**，或称为子目录。

用方框代表目录文件，圆圈代表数据文件。

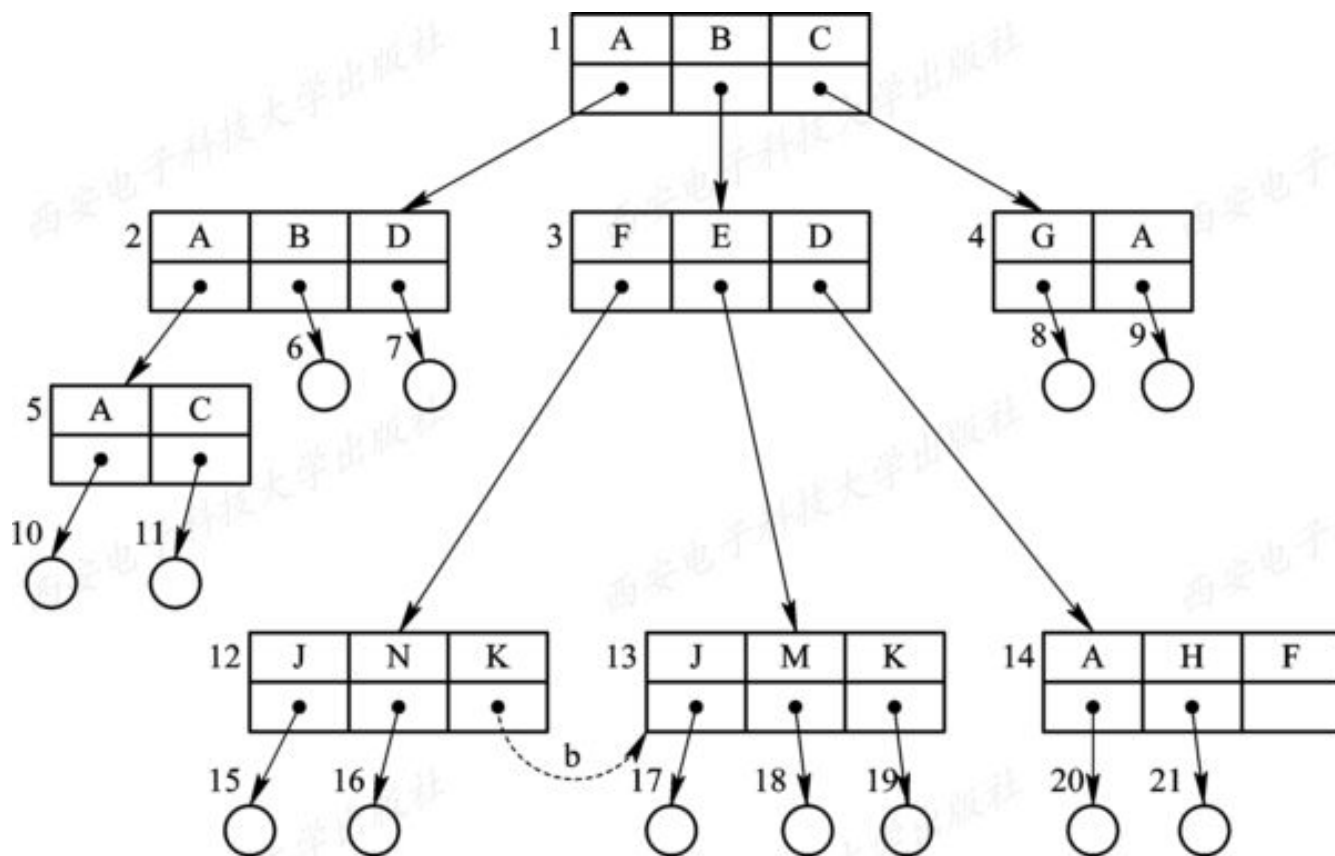


图7-11 多级目录结构



2. 路径名和当前目录

1) 路径名(path name)

在树形结构目录中，从根目录到任何数据文件都只有一条唯一的通路。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，即构成该数据文件唯一的路径名。



2) 当前目录(Current Directory)

当一个文件系统含有许多级时，每访问一个文件，都要使用从树根开始，直到树叶(数据文件)为止的、包括各中间节点(目录)名的全路径名。

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);

把从树根开始的路径名称为**绝对路径名**(absolute path name)。



树形结构目录较两级目录而言，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。

在多级目录中，不同性质、不同用户的文件可以构成不同的目录子树，不同层次、不同用户的文件分别呈现在系统目录树中的不同层次或不同子树中，可以容易地赋予不同的存取权限。

但是在树形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系列都采用了树形文件目录。





3. 目录操作

(1) 创建目录

(2) 删除目录

① 不删除非空目录

② 可删除非空目录

(3) 改变目录

(4) 移动目录

(5) 链接(Link)操作

(6) 查找



7.3.4 目录查询技术

1. 线性检索法

线性检索法又称为顺序检索法。

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。

在树形目录中，用户提供的文件名是由多个文件分量名组成的路径名，此时需对多级目录进行查找。

假定用户给定的文件路径名是 `/usr/ast/mbox`，则查找 `/usr/ast/mbox` 文件的过程如图7-12所示。



根目录

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

在结点6中查找
usr字段

结点是
/usr的目录

132

132号盘块是
/usr的目录

6	.
1	..
19	dick
30	erik
51	jim
26	ast
45	bal

结点26是
/usr/ast的目录

496

496号盘块是
/usr/ast的目录

26	.
6	..
64	grants
92	books
60	mbox
81	minik
17	src

图7-12 查找/usr/ast/mbox的步骤







2. Hash方法

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，这将显著地提高检索速度。

在现代操作系统中，通常都提供了模式匹配功能，即在文件名中使用了通配符“*”、“？”等。对于使用了通配符的文件名，系统此时便无法利用Hash方法检索目录，因此，这时系统还是需要利用线性查找法查找目录。



7.4 文件共享

在现代计算机系统中，必须提供**文件共享手段**，即**指系统应允许多个用户(进程)共享同一份文件**。这样，在系统中只需保留该共享文件的一份副本。

如果系统不能提供文件共享功能，就意味着凡是需要该文件的用户，都须各自备有此文件的副本，显然这会造成对存储空间的极大浪费。



7.4.1 基于有向无循环图实现文件共享

1. 有向无循环图DAG(Directed Acyclic Graph)

在严格的树形结构目录中，每个文件只允许有一个父目录，父目录可以有效地拥有该文件，其它用户要想访问它，必须经过其属主目录来访问该文件。这就是说，对文件的共享是不对称的，或者说，**树形结构目录是不适合文件共享**的。

如果允许一个文件可以有多个父目录，即有多个属于不同用户的多个目录，同时指向同一个文件，这样虽会破坏树的特性，但这些用户可用对称的方式实现文件共享，而不必再通过其属主目录来访问。

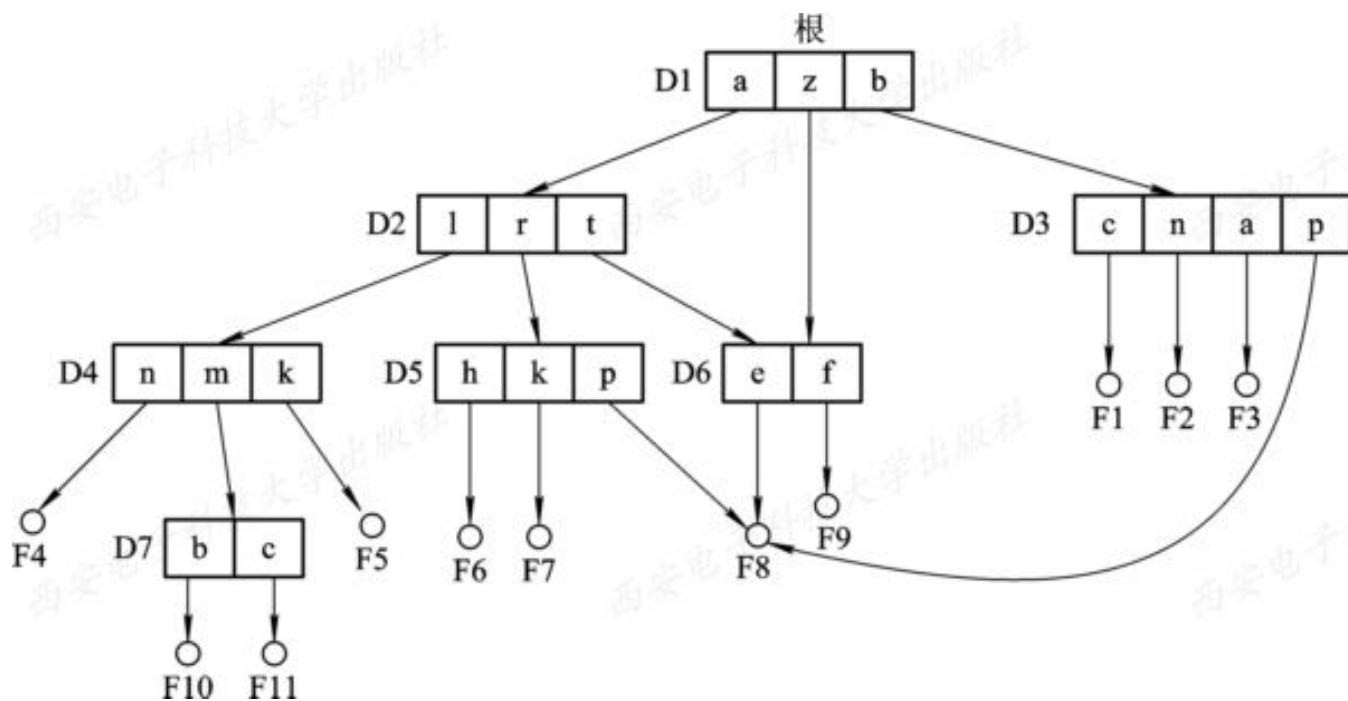
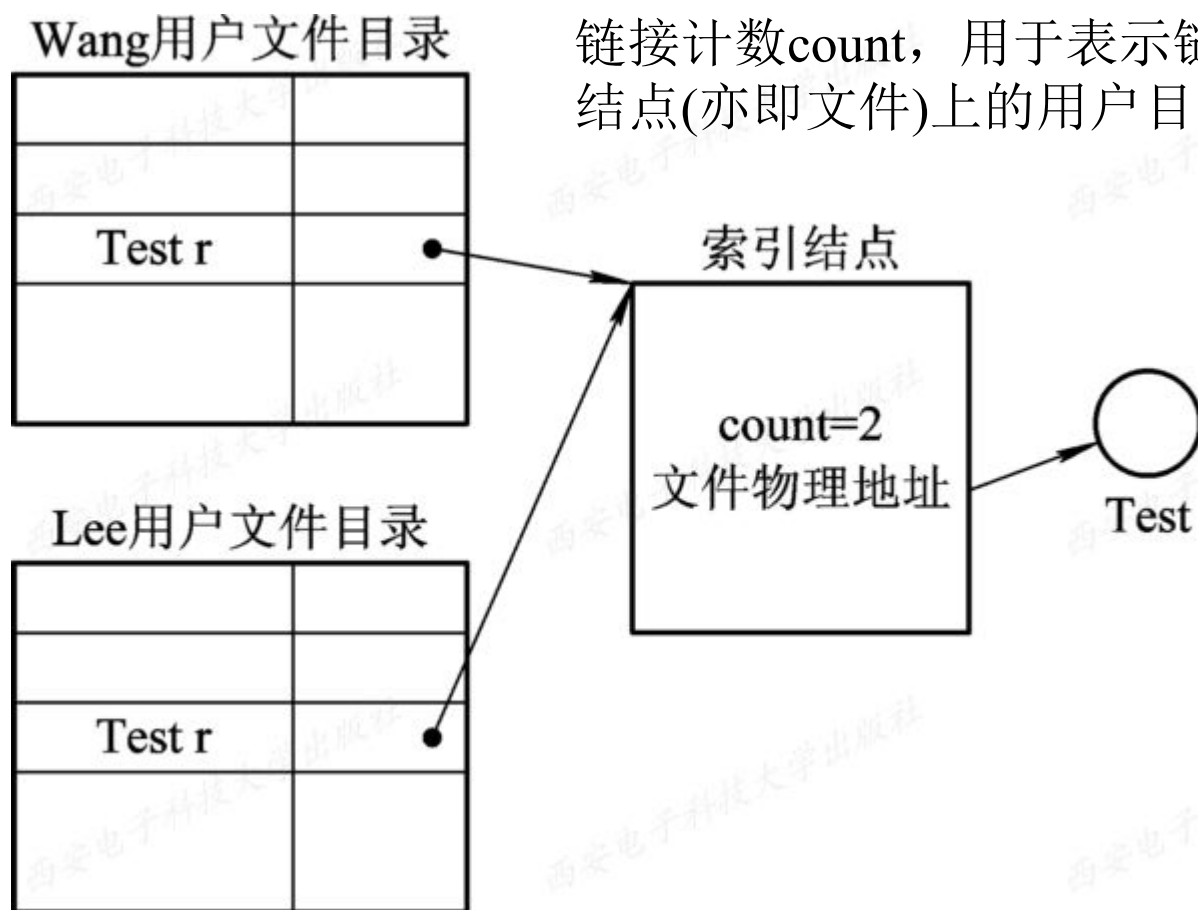


图7-13 有向无循环图目录层次



2. 利用索引结点

为了解决这个问题，可以引用索引结点，即诸如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，如图7-14所示。



链接计数count, 用于表示链接到本索引结点(亦即文件)上的用户目录项的数目

图7-14 基于索引结点的共享方式



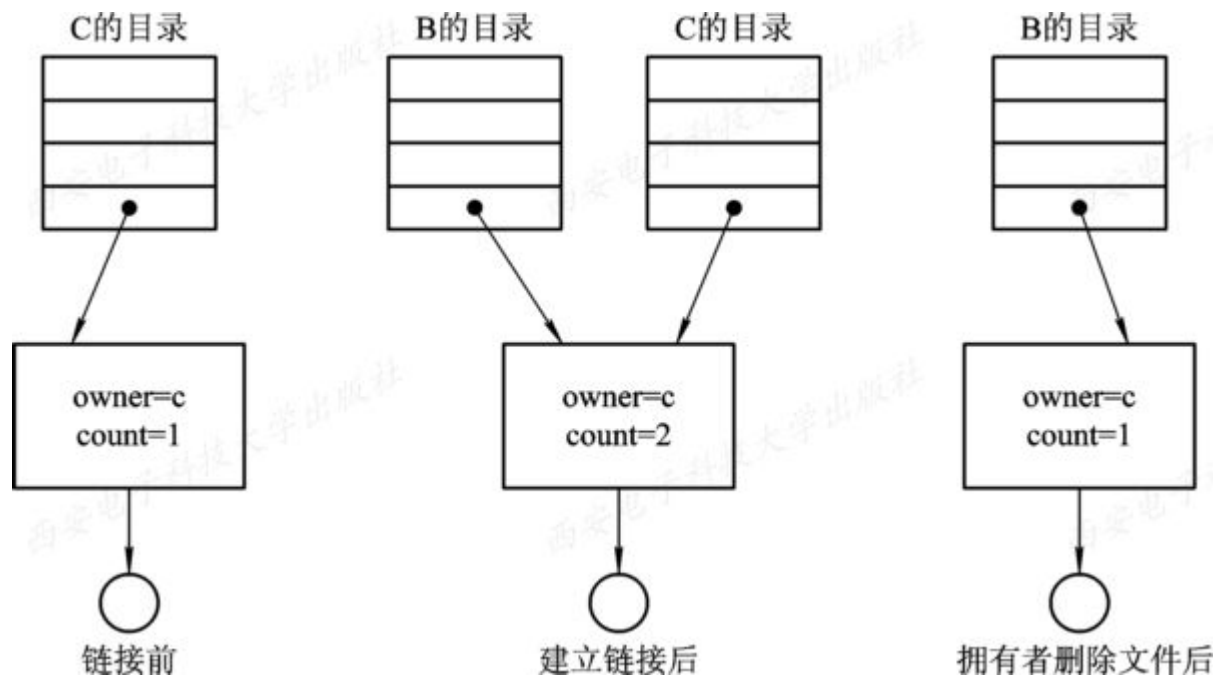




图7-15 进程B链接前后的情况



7.4.2 利用符号链接实现文件共享

1. 利用符号链接(Symbolic Linking)的基本思想

利用符号链接实现文件共享的基本思想，是允许一个文件或子目录有多个父目录，但其中仅有一个作为主(属主)父目录，其它的几个父目录都是通过符号链接方式与之相链接的(简称链接父目录)。

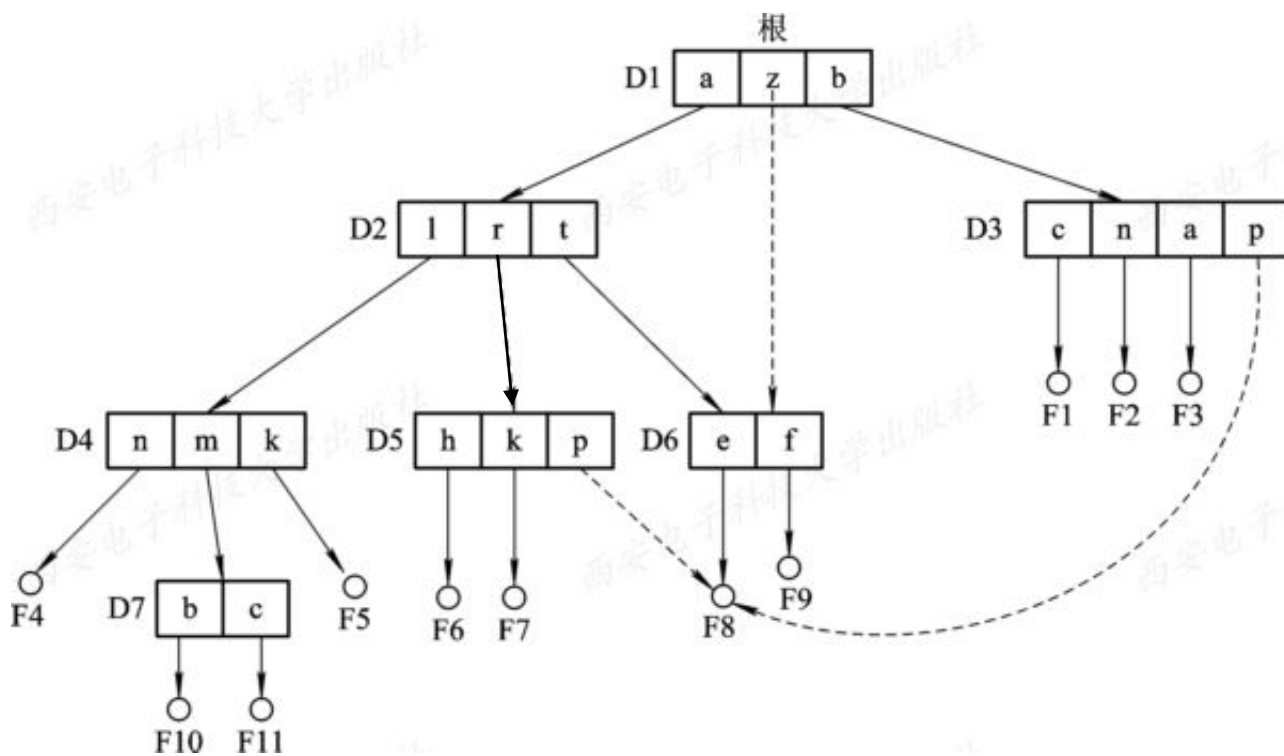


图7-16 使用符号链接的目录层次

2. 如何利用符号链实现共享

为使链接父目录D5能共享文件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写入链接父目录D5中，以实现D5与文件F8的链接。在新文件F中只包含被链接文件F8的路径名。**这样的链接方法被称为符号链接。**

新文件F中的路径名则只被看做是符号链。当用户通过D5访问被链接的文件F8，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找到文件F8，然后对它进行读(写)，这样就实现了用户B对文件F的共享。

3. 利用符号链实现共享的优点

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，如果其他用户又试图通过符号链去访问一个已被删除的共享文件，则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。

★ 举例：Web网页中的超链接，Windows系统中的快捷方式

4. 利用符号链的共享方式存在的问题

利用符号链的共享方式也存在着一些问题：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，而由于链本身实际上是一个文件，尽管该文件非常简单，却仍要为其配置一个索引结点，这也要耗费一定的磁盘空间。



7.5 文件保护

影响文件安全性的主要因素有：

(1) 人为因素，即由于人们有意或无意的行为，而使文件系统中的数据遭到破坏或丢失。

(2) 系统因素，即由于系统的某部分出现异常情况，而造成对数据的破坏或丢失。特别是作为数据存储介质的磁盘，在出现故障或损坏时，会对文件系统的安全性造成影响；

(3) 自然因素，即存放在磁盘上的数据，随着时间的推移将可能发生溢出或逐渐消失。



为了确保文件系统的安全性，可针对上述原因而采取三方面的措施：

- (1) 通过**存取控制机制**，防止由人为因素所造成的文件不安全性。
- (2) 采取**系统容错技术**，防止系统部分的故障所造成的文件的不安全性。
- (3) 建立**后备系统**，防止由自然因素所造成的不安全性。






7.5.1 保护域(Protection Domain)

1. 访问权

为了对系统中的对象加以保护，应由系统来控制进程对对象的访问。对象可以是硬件对象，如磁盘驱动器、打印机；也可以是软件对象，如文件、程序。对对象所施加的操作也有所不同，如对文件可以是读，也可以是写或执行操作。我们把一个进程能对某对象执行操作的权力，称为**访问权** (Access right)。



2. 保护域

为了对系统中的资源进行保护而引入了保护域的概念，保护域简称为“域”。“域”是**进程对一组对象访问权的集合**，进程只能在指定域内执行操作。这样，“域”也就规定了进程所能访问的对象和能执行的操作。

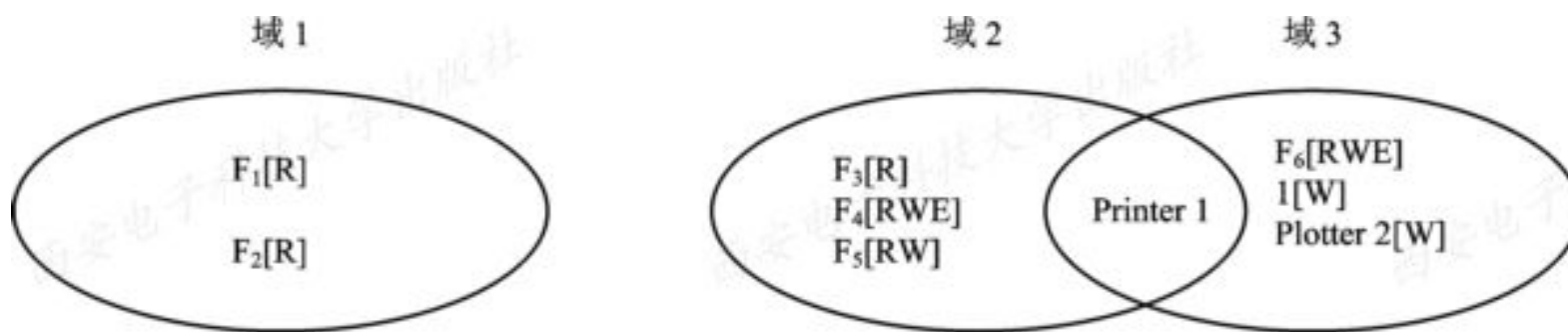


图7-17 三个保护域

3. 进程和域间的静态联系

在进程和域之间可以一一对应，即**一个进程只联系着一个域**。这意味着，在进程的整个生命期中，其可用资源是固定的，我们把这种域称为“**静态域**”。在这种情况下，进程运行的全过程都是受限于同一个域，这将会使赋予进程的访问权超过了实际需要。

4. 进程和域间的动态联系方式

在进程和域之间，也可以是一对多的关系，即**一个进程可以联系着多个域**。在此情况下，可将**进程的**运行分为若干个阶段，其每个阶段联系着一个域，这样便可根据运行的实际需要来规定在进程运行的每个阶段中所能访问的对象。



7.5.2 访问矩阵

1. 基本的访问矩阵

一个矩阵来描述系统的访问控制，并把该矩阵称为**访问矩阵**(Access Matrix)。

访问矩阵中的行代表域，列代表对象，矩阵中的每一项是由一组访问权组成的。

因为对象已由列显式地定义，故可以只写出访问权而不必写出是对哪个对象的访问权，每一项访问权 $\text{access}(i, j)$ 定义了域 D_i 中执行的进程能对对象 Q_j 所施加的操作集。

域 \ 对象	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	Printer 1	Plotter 2
D ₁	R	R, W						
D ₂			R	R, W, E	R, W		W	
D ₃						R, W, E	W	W

图7-18 一个访问矩阵

2. 具有域切换权的访问矩阵

为了实现在进程和域之间的动态联系，应能够将进程从一个保护域切换到另一个保护域。

为了能对进程进行控制，同样应将切换作为一种权力，仅当进程有切换权时，才能进行这种切换。为此，在访问矩阵中又增加了几个对象，分别把它们作为访问矩阵中的几个域；当且仅当 $\text{switch} \in \text{access}(i, j)$ 时，才允许进程从域 i 切换到域 j 。

对象 域	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	Printer 1	Plotter 2	域 D ₁	域 D ₂	域 D ₃
域D ₁	R	R, W								S	
域D ₂			R	R, W, E	R, W		W				S
域D ₃						R, W, E	W	W			

图7-19 具有切换权的访问控制矩阵



7.5.3 访问矩阵的修改

1. 拷贝权(Copy Right)

我们可利用拷贝权将在某个域中所拥有的访问权($\text{access}(i, j)$)扩展到同一列的其它域中，亦即，为进程在其它的域中也赋予对同一对象的访问权($\text{access}(k, j)$)，如图7-20所示。

*表示在i域中运行的进程能将其对对象j的访问权复制成在任何域中对同一对象的访问权

域 \ 对象	F ₁	F ₂	F ₃
D ₁	E		W*
D ₂	E	R*	E
D ₃	E		

(a)

域 \ 对象	F ₁	F ₂	F ₃
D ₁	E		W*
D ₂	E	R*	E
D ₃	E	R	W

(b)

图7-20 具有拷贝权的访问控制矩阵



2. 所有权 (Owner Right)

人们不仅要求能将已有的访问权进行有控制的扩散，而且同样需要能增加某种访问权，或者能删除某种访问权。此时，可利用所有权(O)来实现这些操作。

O表示该进程可以增加或删除任何其它域中运行的对象对对象j的访问权

域 \ 对象	F ₁	F ₂	F ₃
D ₁	O, E		W
D ₂		R*, O	R*, O, W
D ₃	E		

(a)

域 \ 对象	F ₁	F ₂	F ₃
D ₁	O, E		
D ₂		O, R*, W*	R*, O, W
D ₃		W	W

(b)

图7-21 带所有权的访问矩阵

3. 控制权(Control Right)

拷贝权和所有权都是用于改变矩阵内同一列的各项访问权的，或者说，是用于改变在不同域中运行的进程对同一对象的访问权的。控制权则可用于改变矩阵内同一行中(域中)的各项访问权，亦即，用于改变在某个域中运行的进程对不同对象的访问权的。

如果在 $\text{access}(i, j)$ 中包含了控制权，则在域 D_i 中运行的进程可以删除在域 D_j 中运行的进程对各对象的任何访问权。

如果在 $\text{access}(i, j)$ 中包含了控制权，则在域 D_i 中运行的进程可以删除在域 D_j 中运行的进程对各对象的任何访问权。

对象 域	F_1	F_2	F_3	F_4	F_5	F_6	Printer 1	Plotter 2	域 D_1	域 D_2	域 D_3
域 D_1	R	R, W									
域 D_2			R	R, W, E	R, W		W				Control
域 D_3						R, E	W	W			

图7-22 具有控制权的访问矩阵



7.5.4 访问矩阵的实现

1. 访问控制表 (Access Control List)

这是指对访问矩阵按列(对象)划分, 为每一列建立一张访问控制表ACL。在该表中, 已把矩阵中属于该列的所有空项删除, 此时的访问控制表是由一有序对(域, 权集)所组成的。

由于在大多数情况下, 矩阵中的空项远多于非空项, 因而使用访问控制表可以显著地减少所占用的存储空间, 并能提高查找速度。



2. 访问权限 (Capabilities) 表

如果把访问矩阵按行(即域)划分, 便可由每一行构成一张访问权限表。换言之, 这是由一个域对每一个对象可以执行的一组操作所构成的表。表中的每一项即为该域对某对象的访问权限。当域为用户(进程)、对象为文件时, 访问权限表便可用来描述一个用户(进程)对每一个文件所能执行的一组操作。



图7-23 访问权限表

