
河南大学
操作系统
计算机学院

第 7 章

# 文件管理

# 文件系统的引入

- 所有的计算机应用程序都需要存储和检索信息。
- 长期存储信息有三个基本要求：
  - 能够存储大量信息.
  - 使用信息的进程终止时，信息仍旧存在。
  - 必须能使多个进程并发存取有关信息。
- 解决所有这些问题的通常做法是把信息以文件的形式，存储在磁盘或其他外部介质上。
- 存储在文件中的信息必须是永久性的，不会因为创建和终止进程而受到影响。只有在文件的所有者显式地删除文件时，文件才会消失。

# 数据项

- 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。如描述一个学生的基本数据项有学号、姓名等。
- 组合数据项：由若干个基本数据项组成的，简称组项。如工资是组合数据项，包括基本工资，奖金等。
- 数据项的名字和类型两者共同定义了一个数据项的“型”（Type）。而表征一个实体在数据项上的数据则称为“值”（Value）。如：学生是一个实体，“成绩”是实体的“型”描述，具体的分数是实体的“值”。

# 记录

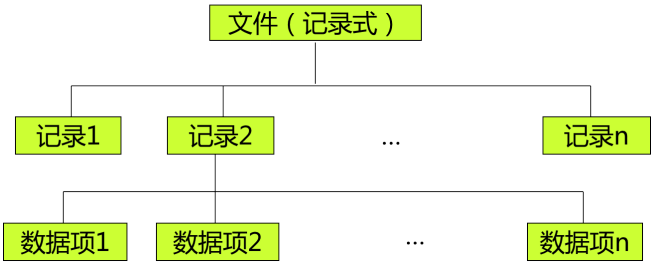
- 记录是一组相关数据项的集合，用于描述一个对象在某方面的属性。
- 为了描述对象的不同属性，一个记录可以选取不同的数据项。
  - 作为一个学生：可选取学号、姓名、成绩等数据项。
  - 作为一个病人：可选取病历号、姓名、病史等数据项。
- 关键字：能唯一地标识出记录的一个或几个基本/组合数据项。

# 文件

- 文件是由创建者所定义的、具有符号名的相关信息集合。
- 文件可分为有结构文件和无结构文件两种。有结构文件是由若干个相关记录组成，无结构文件被看成一个字符流。
- 文件是文件系统中最大的一个数据单位，它描述了一个对象集。
- 文件必须有文件名，所有文件的信息都保存在目录中。
- 文件是一个抽象机制，它提供了一种把信息保存在存储介质上，而且便于以后存取的方法，用户不必关心实现细节。

# 文件

- 文件表示的范围/包含的内容：源程序、二进制代码、文本文档、数据、表格、声音、图像和视频等。
- 文件属性：名字、标识符、类型、位置、大小、保护、时间日期和用户标识等。



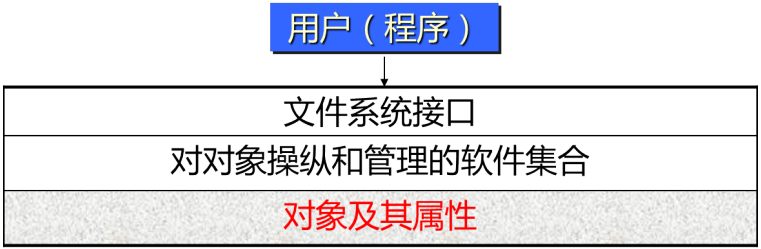
# 文件类型

- 按用途分
  - 系统文件
  - 用户文件
  - 库文件
- 按数据形式分
  - 源文件
  - 目标文件
  - 可执行文件
- 按存取控制属性
  - 只读文件
  - 读写文件
  - 只执行文件
  - 不保护文件
- 按文件的逻辑结构分
  - 有结构文件（记录式文件）
  - 无结构文件（流式文件）
- 按文件的物理结构分
  - 顺序文件（数据连续存放）
  - 链接文件
  - 索引文件
- 按组织形式和处理方式分
  - 普通文件
  - 目录文件
  - 特殊文件：特指系统中的各类I/O设备。

# 文件系统

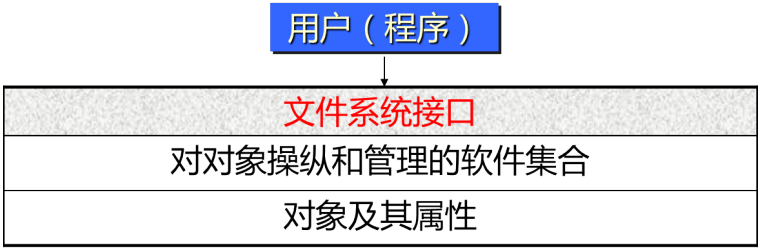
- 文件系统是指 OS 中负责操纵和管理文件的软件集合。
- 文件系统 = 文件管理程序 + 它所管理的全部文件（文件和目录的集合）。
- 文件系统为用户提供统一方法访问存储在物理介质上的信息。

# 文件系统的层次结构



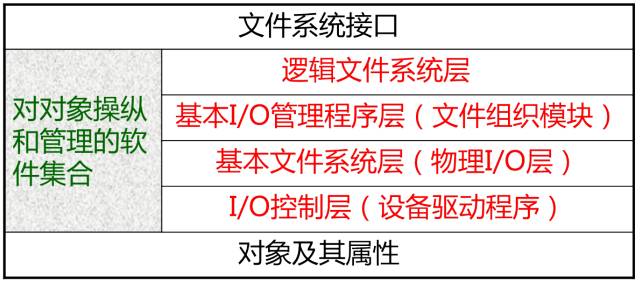

- 文件系统管理的对象
  - 文件。
  - 目录：提供文件逻辑名来访问文件；目录项含有文件名及该文件所在的物理地址（或指针）。
  - 物理存贮空间

# 文件系统的层次结构




- 文件系统的接口
  - 命令接口：用户与文件系统交互的接口。
  - 程序接口：用户程序与文件系统交互的接口。用户程序可通过系统调用来取得文件系统的服务。

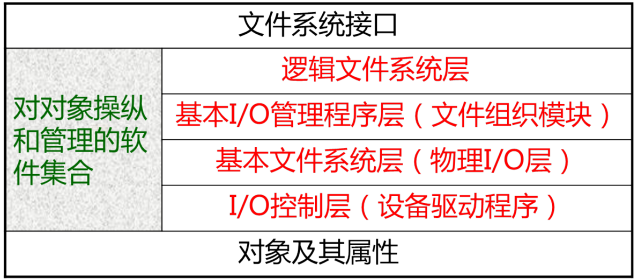
# 文件系统的层次结构




逻辑文件系统层：处理文件及记录的相关操作（目录操作及访问保护）。

接受用户程序指令，如：read，write

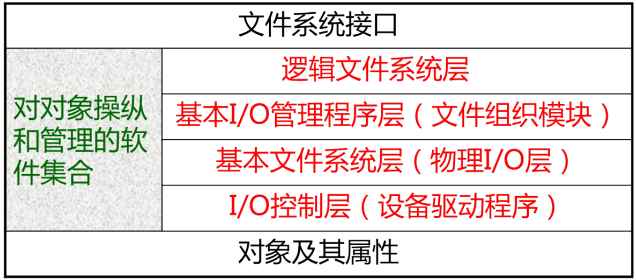
# 文件系统的层次结构




基本I/O管理程序层：选择设备，逻辑块号到物理块号的转换，空闲空间管理等。

文件的逻辑块号→物理块号

# 文件系统的层次结构




基本文件系统层：负责内存与磁盘间的数据块交换（在外存及内存缓冲区的位置）。

向driver发指令，(buf←→具体物理盘块号)

# 文件系统的层次结构

<table>
<tr><td colspan="2">文件系统接口</td></tr>
<tr><td rowspan="4">对对象操纵和管理的软件集合</td><td>逻辑文件系统层</td></tr>
<tr><td>基本I/O管理程序层（文件组织模块）</td></tr>
<tr><td>基本文件系统层（物理I/O层）</td></tr>
<tr><td>I/O控制层（设备驱动程序）</td></tr>
<tr><td colspan="2">对象及其属性</td></tr>
</table>

I/O控制层：由设备驱动程序和中断处理程序组成，实现内存和磁盘系统之间的信息传输。

启动I/O操作及处理设备发来的中断信号

# 文件系统的层次结构

- 对记录的操作：检索、插入、修改、删除。
- 对文件的操作（系统调用）
  - 最基本的文件操作：创建 分配空间 → 修改目录、删除 查目录 → 删目录 → 回收空间、截断 长度置零、读写 查目录 → 地址 → 读写指针、设置文件的读/写位置 随机存取。
  - 其它文件操作：文件属性类操作、目录类操作。
  - 组合操作：如：文件拷贝 = 创建文件 + 写入

# 文件的“打开”和“关闭”操作（系统调用）

- OS 所提供文件操作过程
  1. 第一步是检索文件目录（在外存）来找到指定文件的属性及在外存上的位置。
  2. 第二步是对文件实施相应的操作，如读或写文件等。
- 任何一个文件使用前都要先打开，即把文件控制块 FCB送到内存，以建立用户和文件的联系。
- 文件的“打开”和“关闭”操作：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）系统调用。

# 文件的“打开”和“关闭”操作（系统调用）

- 打开：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。以后若再访问此文件，则利用编号直接在内存中检索，从而节省大量的检索开销，提高了文件的操作速度。
- 关闭：当用户不再需要对该文件的操作时，利用关闭系统调用将文件的属性从内存打开文件表中删除，从而切断用户与文件间的联系。

# 文件的逻辑结构

所有文件存在着两种形式的结构：

1. 文件的逻辑结构（文件组织）

   从用户观点出发所观察到的文件组织形式，是用户可以直接处理的数据及其结构，它独立于物理特性。

   顺序文件、索引文件、索引顺序文件

2. 文件的物理结构（文件的存储结构）

   是指文件在外存上的存储组织形式，用户是看不见的，文件的物理结构不但与存储介质的存储性能有关，而且还与所采取的外存分配方式有关。

   顺序文件、链接文件、索引文件

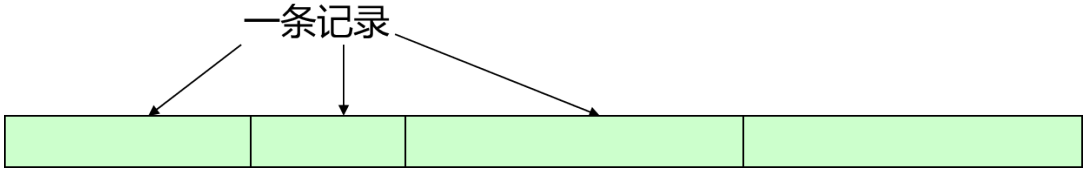
# 文件逻辑结构的类型

## 文件逻辑结构的类型

1. 按是否有结构分类
   - 有结构文件（记录式文件），如数据库文件
   - 无结构文件（流文件），如程序、文本文件
2. 按文件的组织方式分类
   - 顺序文件
   - 索引文件
   - 顺序索引文件

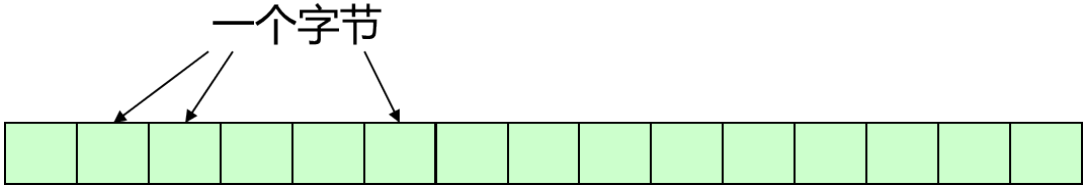
# 有结构文件（记录式文件）

- 文件构成：由一个以上的记录构成。
- 记录长度：分为定长和变长。
- 分类（按记录的组织）：顺序文件、索引文件、索引顺序文件。

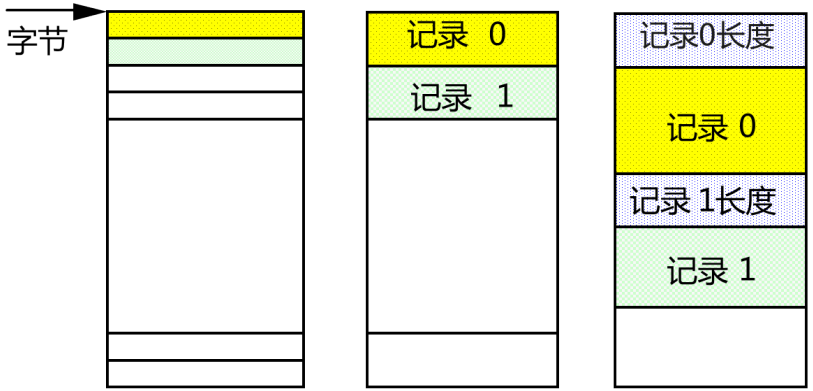


# 无结构文件（流文件）

- 文件构成：由字符流构成。
- 长度：字节为单位
- 访问：读写指针
- Unix 中把所有文件视为流式文件。



# 文件的逻辑结构



# 顺序文件

- 顺序文件记录的排序
  - 串结构：记录顺序与关键字无关，按存入时间的先后排列。
  - 顺序结构：记录顺序按关键字或按字母顺序排列。
- 对顺序文件的读、写操作
  - 记录为定长的顺序文件：易于定位，可随机读取。
  - 记录为变长的顺序文件：不易定位，只能顺序读取。

# 顺序文件的优缺点

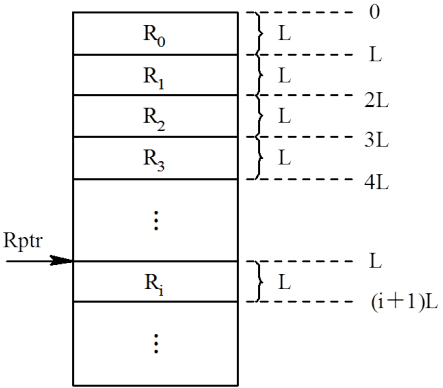
- 优点
  - 顺序存取速度快，批处理时效率是所有逻辑文件中最高的。
  - 对定长记录，还可方便实现直接存取。
- 缺点
  - 建立文件前需要能预先确定文件长度，以便分配存储空间。
  - 交互应用时“效率低”（如要查找单个记录），尤其是对变长记录的顺序文件。
  - 增加、删除记录涉及到排序问题，开销大。解决方法：事务文件 (log )，用于存放更新文件的记录。

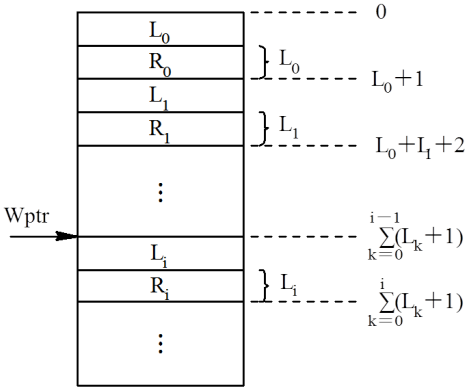
# 对顺序文件的读/写操作

- 定长记录：
  - 读写完一个记录（定长 L）后：读指针 Rptr:= Rptr + L；写指针 Wptr:= Wptr + L
  - 第 i 个记录相对于第一个记录首址的地址：Ai=i× L
- 变长记录：
  - 读写完一个记录后：Rptr:=Rptr+Li+1；Wptr:=Wptr+Li+1。Li 是刚读完或写完记录的长度。变长记录中每个记录前用一个字节指明该记录的长度。
  - 第 i 个记录的首址：须顺序地查找每个记录，获得记录长度 Li，然后才能计算出第 i 个记录的首址：

$$Ai = \sum_{i=0}^{i-1}(Li+1)$$

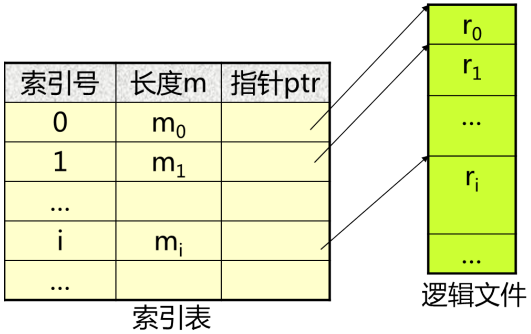
# 定长记录和变长记录文件



# 索引文件

- 为解决变长记录文件的直接存取低效问题，引入了索引文件组织方式。
- 索引文件为变长记录文件建立一张索引表，每个记录在索引表中占一个表项。
- 索引文件可以有多个索引表。如学号索引表、姓名索引表。



# 索引文件的优缺点

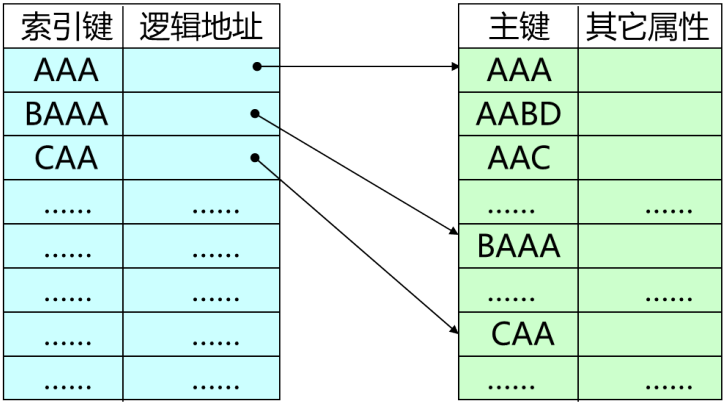
- 优点
  - 通过索引表可方便地实现直接存取，具有较快的检索速度。（索引表本身就是一个定长记录的顺序文件）
  - 易于进行文件的增删。
- 缺点
  - 索引表的使用增加了存储费用。
  - 索引表的查找策略对文件系统的效率影响很大。（折半查找法）

# 索引顺序文件

- 索引顺序文件
  - 顺序文件 + 索引文件
  - 在顺序文件（主文件）的基础上，另外建立了文件索引表和溢出文件。目的是加快顺序文件的检索速度。
  - 把关键字域中的取值分为若干个区间（如 a-z 分为 26 个区间），每个区间对应于索引表中的一个索引项（指向该区间的第一个记录）。
  - 新增记录暂时保存在溢出文件中，定期归并入主文件。

# 索引顺序文件

1. 利用关键字检索索引表，找到该记录所在组的第一个记录的地址。
2. 采用顺序查找法查找记录。



# 检索速度的比较

**例 1**：10000 个记录 ($N$)

- 顺序文件：5000 次查找。( 平均 $N/2$ 次 )
- 索引顺序文件：设 100 个记录一组，索引表查找方法设为顺序法的情况下，则查找次数为 50+50=100。( 平均 $2 \times \sqrt{N}/2 = \sqrt{N}$ 次 )

**例 2**：1000000 个纪录

- 索引顺序文件：采用多级索引表
- 低级索引 :( 100 个纪录一组 )：10000。
- 高级索引：100
- 查找次数：50+50+50=150 ( 平均 $(3/2)\sqrt[3]{N}$ 次 )

# 直接文件和哈希文件

- 直接文件
  - 传统转换：记录键值 → 线性表或链表 → 物理地址
  - 直接文件：根据给定的记录键值，直接获得指定记录的物理地址。即记录键值本身就决定了记录的物理地址。
  - 键值转换（Key to address transformation）：由记录键值直接到记录物理地址的转换。
- 哈希文件（最常用的一种直接文件）
  - K 为记录键值，A 为通过 Hash 函数转换所形成的该记录在目录表中对应表目的位置，则：A=H(k)。该表目的内容指向相应记录的物理块。

# 文件目录

- 文件目录
  - 用于检索文件的目录称为文件目录，它是由目录项所构成的有序序列。
- 对文件目录的管理要求
  - 实现“按名存取”
  - 提高对目录的检索速度
  - 文件共享（多个用户共享一个文件，只需要在外存保留一个文件）
  - 允许文件重名（不同文件使用同一个名字）

# 文件控制块

- 从文件管理角度看，文件由文件控制块 FCB 和文件体（文件本身）两部分组成。
- 文件控制块（FCB）
  - 文件控制块是操作系统为管理文件而设置的数据结构，存放了文件的有关说明信息，是文件存在的标志。
- FCB 中的三类信息
  - 基本信息类：文件名、文件物理位置（位置、长度）、文件逻辑结构（流式、记录式；定长、变长）、文件物理结构（顺序、链接、索引）。
  - 存取控制信息类：权限、用户名、口令、共享计数。
  - 使用信息类：建立日期、最后修改日期等。

# 文件控制块

- 存取文件的过程：
  - 文件控制块 FCB→ 块链
  - 文件控制块 FCB→FAT/MFT→ 块链
- 每个文件有一个 FCB，它们被保存在外存空间。当访问一个文件时，应当能够根据文件名字找到它所对应的 FCB。那么 FCB 是如何保存于外存的呢？

# 文件控制块

- 存取文件的过程：
  - 文件控制块 FCB→ 块链
  - 文件控制块 FCB→FAT/MFT→ 块链
- 每个文件有一个 FCB，它们被保存在外存空间。当访问一个文件时，应当能够根据文件名字找到它所对应的 FCB。那么 FCB 是如何保存于外存的呢？
- 它是作为目录项存储于目录文件中的。FCB 也被称作目录项。

# 文件控制块

- 文件目录
  - 文件目录是一种数据结构，用于标识系统中的文件及其物理地址，将文件名映射到外存物理位置，供检索使用。
- 目录项
  - 构成文件目录的项目（目录项就是 FCB）
  - 文件目录：文件控制块的有序集合。
- 目录文件
  - 为了实现对文件目录的管理，通常将文件目录以文件的形式保存在外存，这个文件就叫目录文件。

文件目录和目录文件是同一事物的两种称谓。
从用途角度来看称其为文件目录，从实现角度来看称其为目录文件。

# 索引结点

- 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，先将存放目录文件的第一个盘块中的目录调入内存，然后把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，便再将下一个盘块中的目录项调入内存。
- 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。

$$1+2+3+\cdots+N=(N+1)N/2$$

平均值 (N+1)/2；近似值 N/2

# 索引结点

**例**：在某个文件系统中，每个盘块为 512 字节，文件控制块占 64 个字节。对一个存放在磁盘上的 256 个目录项的目录，为找到其中一个文件的 FCB，求平均启动磁盘的次数。

# 索引结点

**例**：在某个文件系统中，每个盘块为 512 字节，文件控制块占 64 个字节。对一个存放在磁盘上的 256 个目录项的目录，为找到其中一个文件的 FCB，求平均启动磁盘的次数。

解：每个目录项中存放的是对应文件的 FCB，故 256 个目录项的目录总共需要占用 256×64/512=32 个盘块。故在该目录中检索到一个文件平均启动磁盘次数为 (1+32)/2=16.5。

# 索引结点

- 检索目录文件的过程中，只用到了文件名，仅当文件名与指定要查找的文件名相匹配时，才需从该目录项中读出该文件的物理地址，而其它信息在检索目录时不需调入内存。
- 解决方案：引入索引结点。

| 文件名 | 索引结点编号 |
| --- | --- |
| 文件名1 | |
| 文件名2 | |
| …… | …… |

# 索引结点

- 索引结点把文件名与文件描述信息分开。
- 将 FCB 分为文件名、i 结点指针 (索引 index) 和相应的 i 结点。
- 离散存放的目录结构
- 查询时只调入文件目录（仅包括文件名和索引结点指针），找到后才调入相应结点。

# 索引结点

- 索引结点分为磁盘索引结点和内存索引结点。
- 磁盘索引结点：存放在磁盘上的索引结点。
  - 文件主标识；
  - 文件类型；
  - 文件存取权限；
  - 文件物理地址；(表达出盘块号)
  - 文件长度；
  - 连接（共享）计数；
  - 存取时间。

# 索引结点

- 内存索引结点：存放在内存中的索引结点。
- 文件打开后，将磁盘索引结点的内容部分或全部子集拷贝到内存中的索引结点，并增加以下内容：
  - 编号；
  - 状态；( 上锁、修改 )
  - 共享计数；
  - 逻辑设备号；
  - 链接指针：i 结点的组织结构。

# 索引结点

**例**：在某个文件系统中，每个盘块为 512 字节，文件控制块占 64 个字节，其中文件名占 8 个字节。如果索引结点编号占 2 个字节，对一个存放在磁盘上的 256 个目录项的目录，试比较引入索引结点前后，为找到其中一个文件的 FCB，平均启动磁盘的次数。

# 索引结点

**例**：在某个文件系统中，每个盘块为 512 字节，文件控制块占 64 个字节，其中文件名占 8 个字节。如果索引结点编号占 2 个字节，对一个存放在磁盘上的 256 个目录项的目录，试比较引入索引结点前后，为找到其中一个文件的 FCB，平均启动磁盘的次数。

解：引入索引节点前，每个目录项中存放的是对应文件的 FCB，故 256 个目录项的目录总共需要占用 256×64/512=32 个盘块。故在该目录中检索到一个文件平均启动磁盘次数为 (1+32)/2=16.5。

# 索引结点

**例**：在某个文件系统中，每个盘块为 512 字节，文件控制块占 64 个字节，其中文件名占 8 个字节。如果索引结点编号占 2 个字节，对一个存放在磁盘上的 256 个目录项的目录，试比较引入索引结点前后，为找到其中一个文件的 FCB，平均启动磁盘的次数。

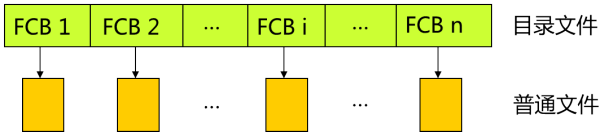
解：引入索引节点后，每个目录项中只需存放文件名和索引节点的编号，因此 256 个目录项的目录总共需要占用 256×(8+2)/512=5 个盘块。因此，找到匹配的目录项平均需要启动 3 次磁盘；而得到索引结点编号后还需启动磁盘将对应文件的索引结点读入内存，故平均需要启动磁盘 4 次。

# 目录的结构

- 单级文件目录：在整个系统中只建立一张目录表。
- 新建文件：检索目录有无同名文件，如果没有，把新建文件加入目录表（置状态位为 1）。
- 删除文件：检索目录项，回收存储空间，删除该目录项。

| 文件名 | 物理地址 | 文件说明 | 状态位 |
|---|---|---|---|
| 文件名1 | | | |
| 文件名2 | | | |
| | | | |

是否空闲

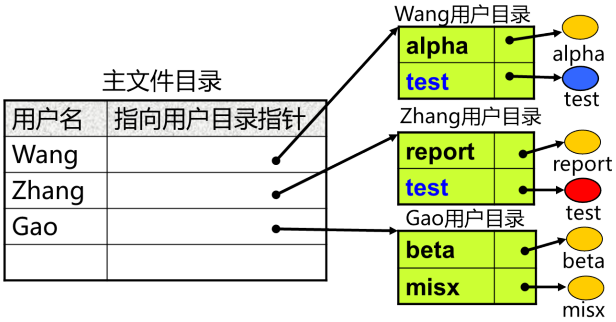


# 单级文件目录

- 优点
  - 单级文件目录管理比较简单，易于实现按名存取。
- 缺点
  - 限制了用户对文件的命名（不允许重名）。
  - 文件平均检索时间长 (平均 N/2)。
  - 不便于实现文件共享；只适用于单用户环境。

# 两级文件目录

- 两级文件目录：在整个系统中建立两级目录。
- 为每个用户建立一个单独的用户文件目录（UFD）。
- 系统中为所有用户建立一个主文件目录（MFD）。

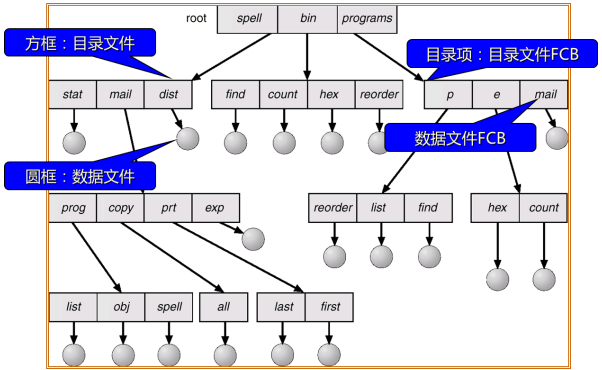


# 两级文件目录

- 优点
  - 两级文件目录提高了检索目录的速度。例如：如果有 n 个用户，每用户最多 m 个文件，则最多检索 n+m 个目录项而非单级目录结构的 n×m 项。
  - 不同用户目录中可重名。
  - 不同用户可用不同文件名来访问系统中一共享文件。
- 缺点
  - 对文件的共享不方便（用户隔离）。
  - 增加了系统开销，缺乏灵活性，无法反映真实世界复杂的文件结构形式。

# 树型结构目录

- 树型结构目录只有一个根目录，而且除根目录外，其余每个目录或者文件都有唯一的一个上级目录。
- 每一级目录可以包含文件，也可以包含下一级目录。



# 树型结构目录

- 最上一级目录称为根目录，其它称为子目录。
- 操作系统为每个目录建立一个目录文件，其内容就是该目录下包含的文件的 FCB 集合。

根目录文件的内容：

| 文件名 | 文件类型 | 外存地址 | … |
|---|---|---|---|
| 作业 | 目录文件 | | |
| 软件 | 目录文件 | | |
| 娱乐 | 目录文件 | | |
| F1 | 数据文件 | | |

作业目录文件的内容：

| 文件名 | 文件类型 | 外存地址 | … |
|---|---|---|---|
| C | 目录文件 | | |
| OS | 目录文件 | | |
| F1.C | 数据文件 | | |
| F2.C | 数据文件 | | |
| OS1 | 数据文件 | | |

# 树型结构目录

- 优点
  - 层次结构清晰，便于管理和保护。
  - 解决重名问题。
  - 查找速度加快，因为每个目录下的文件数目较少。
- 缺点
  - 查找一个文件，需要按路径名逐级访问中间节点，增加了磁盘访问次数。

# 路径名和当前目录

- 路径名
  - 在树形目录结构中，从根目录到任何数据文件，都只有一条唯一的通路。
  - 这一系列目录名和最后达到的文件名自身依次地用“/”连接起来组成了该文件的路径名。
  - 多个文件可以同名，只要保证它们的路径名唯一即可。
- 当前目录
  - 为了提高文件检索速度，文件系统向用户提供了一个当前正在使用的目录，称为当前目录（工作目录）。
  - 当前目录一般存放在内存。

# 路径名和当前目录

- 绝对路径名
  - 绝对路径名是从根目录开始到达所要查找文件的路径名，以“/”打头。
  - /root/usr/m1/prog/f1.c
- 相对路径名
  - 相对路径名从当前目录（工作目录）的下级开始书写。
  - 例如：当前目录是/usr/m1，则有相对于当前目录的相对路径名：prog/f1.c
  - . 当前目录：./prog/f1.c
  - .. 上级目录：../ml/prog/f2.c

# 目录操作

- 创建目录
  - 用户可以为自己建立用户文件目录（UFD），并创建子目录。创建一个新文件时，只需查看 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。
- 删除目录
  - 不删除非空目录。必须先删除所有文件，变成空目录后才能删除。
  - 可删除非空目录。

# 目录操作

- 移动目录：文件或子目录移动后（剪切、复制），文件的路径名将随之改变。
- 链接操作：对于树型结构目录，每个文件或子目录只允许一个父目录。但是可以通过链接操作让指定文件有多个父目录，从而方便共享。（网状结构）
- 目录查询：操作系统支持多种目录查找方式。

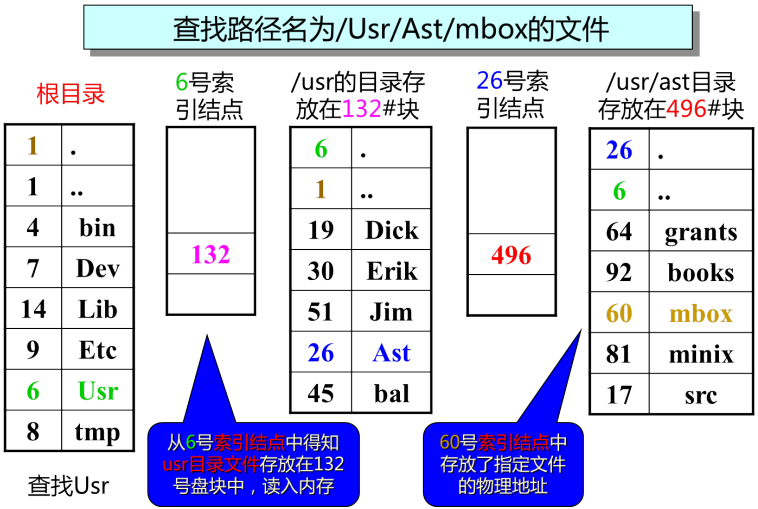
# 目录查询技术

- 数据文件（按名存取）的查询步骤
  - 文件名 → 目录项（FCB）或索引结点 → 盘块号 → 驱动程序 → 启动磁盘 → 读数据文件至内存
- 对目录进行查询的方式
  1. 线性检索法（顺序检索法）
  2. Hash 方法

# 线性检索法

- 顺序查找（线性查找）
  - 查找过程：从表的一端开始查找，顺序用各记录的关键字进行比较，若找到与其值相等的元素，则查找成功；若直到最后一个记录仍未找到，则查找失败。
  - 线性表可以是顺序存储结构或链式存储结构，可以是有序表或无序表。
- 二分查找（折半查找）
  1. 线性表必须是顺序存储结构，并且是有序表。

# 线性检索法


查找路径名为/Usr/Ast/mbox的文件
根目录
1 .
1 ..
4 bin
7 Dev
14 Lib
9 Etc
6 Usr
8 tmp
查找Usr
6号索引结点
132
从6号索引结点中得知usr目录文件存放在132号盘块中，读入内存
/usr的目录存放在132#块
6 .
1 ..
19 Dick
30 Erik
51 Jim
26 Ast
45 bal
26号索引结点
496
60号索引结点中存放了指定文件的物理地址
/usr/ast目录存放在496#块
26 .
6 ..
64 grants
92 books
60 mbox
81 minix
17 src

# Hash 方法

- Hash 表（散列表）的查找采用计算寻址的方式进行查找，查找效率与比较次数无关或关系较小，所以查找的效率较高。
- Hash 方法目录查询
  - 构造 Hash 表：以文件名 k 为自变量，通过一个确定的函数关系 f 计算出 Hash 函数值 f(k)，把这个值（Hash 地址）存储为一个线性表，称为散列表或哈希表。
  - 查找 Hash 表：根据给定值 k（文件名），计算 Hash 函数值 f(k)，若系统中存在文件名与 k 相等的记录，则必定存储在哈希表中 f(k) 的位置上。因此不需比较便可直接在哈希表中取得所查目录项。然后在目录项中得文件的物理地址。

# 文件共享形式与目的

- 文件共享: 一个文件被多个用户或程序使用。
- 单机共享 → 多机共享 → 网络共享
- 共享形式：
  1. 文件被多个用户使用，由存取权限控制。
  2. 文件被多个程序使用，但各使用自己的读写指针。
  3. 文件被多个程序使用，但共享读写指针。
  4. 多个用户用相同或不同的名字来访问同一文件。
- 共享目的:
  - 共享节省用户时间和存储空间。
  - 进程间通过文件交换信息（比如 Unix 中的管道）。

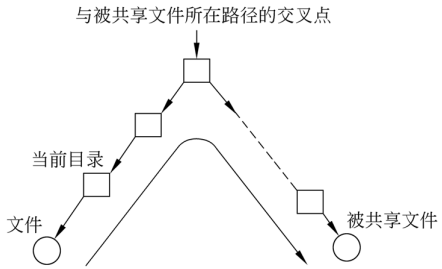
# 文件共享

## 文件共享的方法

- 早期实现文件共享的方法
  - 绕弯路法
  - 连访法
  - 利用基本文件目录实现文件共享
- 基于索引结点的共享方式
- 利用符号链实现文件共享

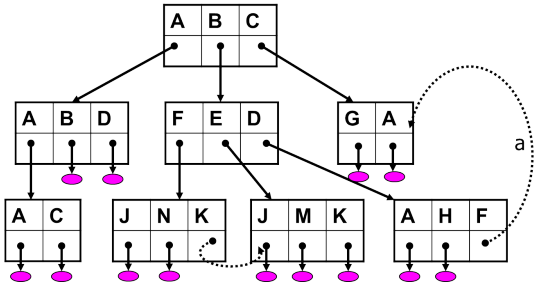
# 绕弯路法

- 绕弯路法：允许每个用户获得一个“当前目录”，用户所访问的所有文件均相对于当前目录。如果共享文件不在当前目录，则用户从当前目录出发向上返回到共享文件所在路径的交叉点，再顺序向下访问共享文件。
- 绕弯路法共享的效率比较低。



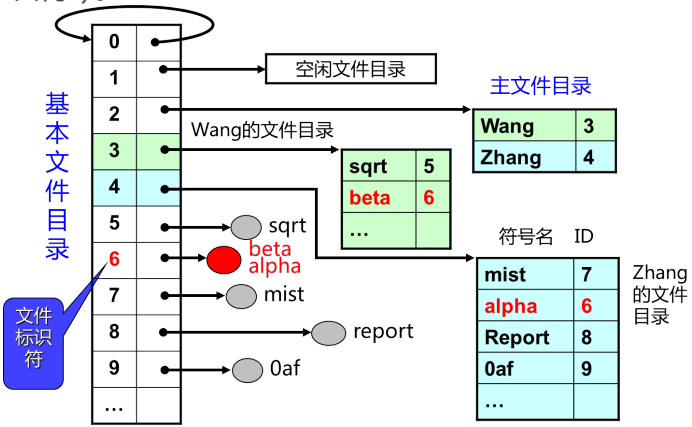
# 连访法

- 为了提高共享访问速度，在相应的目录项之间进行链接，使一个目录中的目录项指向另一目录中的目录项。
- 链接后已不是树形结构，成为网状结构，路径不唯一。
- 在删除目录树的分支时，要考虑是否有链接，否则有些链接指针可能指向一个已被删除的目录表目。



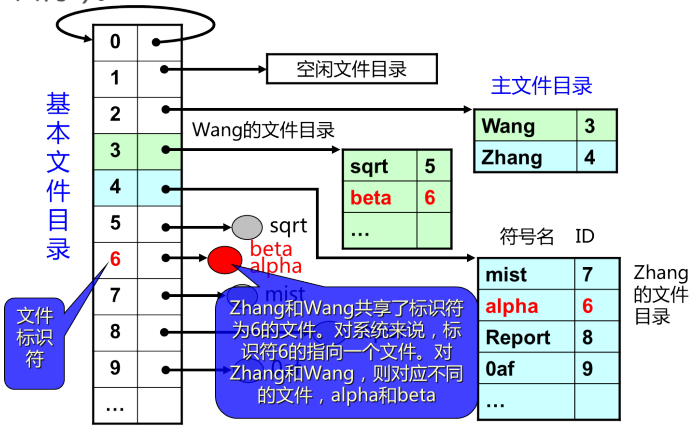
# 利用基本文件目录实现文件共享

文件系统中设置一个基本文件目录 BFD，每个文件在该目录中均占有一个目录项（唯一标识符）。



# 利用基本文件目录实现文件共享

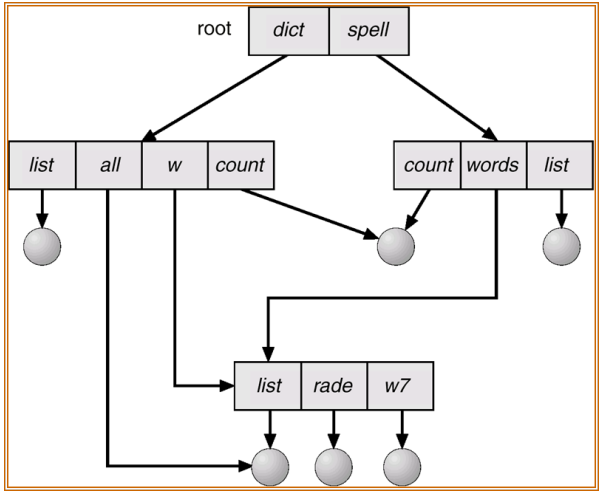
文件系统中设置一个基本文件目录 BFD，每个文件在该目录中均占有一个目录项（唯一标识符）。



# 基于索引结点的共享方式

- 如果采用连访法共享文件，就形成一种有向无循环图目录结构（带链接的树形目录）。它允许目录含有共享子目录和文件，但不构成环路。
- 同一个文件或子目录可能出现在两个不同目录中，通过指针与文件相连。同一个文件可以有多个路径名。有向无循环图目录结构不再是一颗树，而成为网状结构。
- 在有向无循环图目录结构中，如果有用户要共享一个文件或子目录，必须把共享文件或子目录拷贝或链接到用户的目录中。
- 存在的问题：新增内容不能被共享。

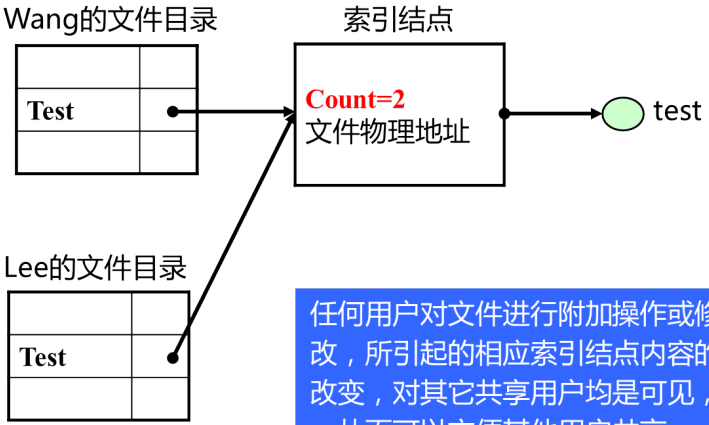
# 有向无循环图目录结构


root
dict
spell
list
all
w
count
count
words
list
list
rade
w7

# 基于索引结点的共享方式

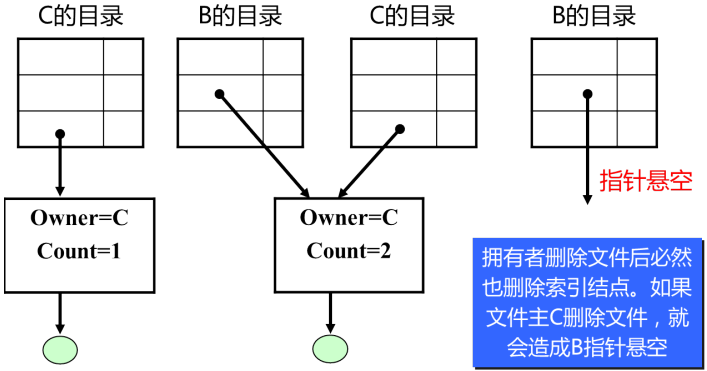
- 文件目录：由文件名和指向索引结点的指针组成。
- 索引节点（i 结点）：存放除文件名以外的文件属性。包括文件的结构信息、物理块号、存取控制、管理信息等。
- 基于索引结点的共享方式：共享索引结点
- 在索引结点中增加链接计数 count，表示共享的用户数。
- 删除文件时必须 count=0 方可。当 count>1 时，文件主也不能删除文件，否则指针悬空。

# 基于索引结点的共享方式

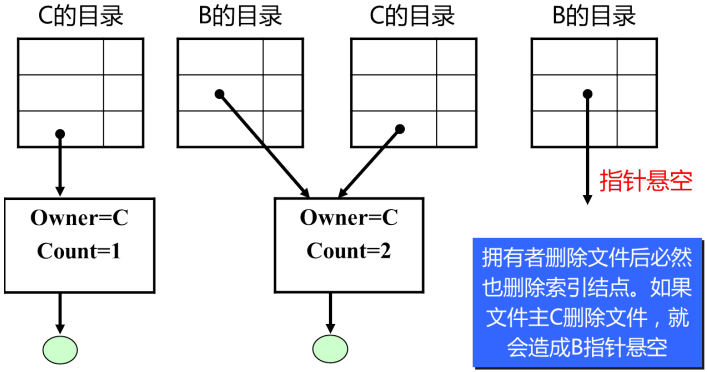



任何用户对文件进行附加操作或修改，所引起的相应索引结点内容的改变，对其它共享用户均是可见，从而可以方便其他用户共享。

# 基于索引结点的共享方式



# 基于索引结点的共享方式




要解决基于索引结点的共享方式可能发生的指针悬空问题，可以利用符号链实现文件共享。

# 利用符号链实现文件共享

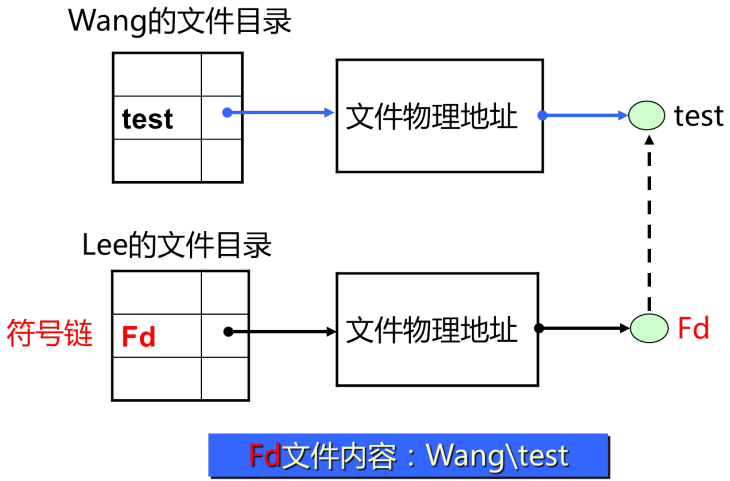
- 共享某文件时，创建一个“符号链 LINK”类型的新文件，并加到用户目录中，该文件仅包含被链接/共享文件的路径名，称这种链接方法为符号链接（软链接）。
- 符号链方式中，只有文件主才拥有指向其索引结点的指针，其它共享的用户只有该文件的路径名，不拥有指向其索引结点的指针。
- 文件主可对原文件删除。其他用户试图通过符号链去访问一个已被删除的共享文件时，会因系统找不到该文件而使访问失败，于是将符号链删除，而不会发生指针悬空现象。

# 利用符号链实现文件共享

- 共享某文件时，创建一个“符号链 LINK”类型的新文件，并加到用户目录中，该文件仅包含被链接/共享文件的路径名，称这种链接方法为符号链接（软链接）。
- 符号链方式中，只有文件主才拥有指向其索引结点的指针，其它共享的用户只有该文件的路径名，不拥有指向其索引结点的指针。
- 文件主可对原文件删除。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，于是将符号链删除，而不会发生指针悬空现象。

Windows 中的快捷方式就是符号链文件

# 利用符号链实现文件共享



# 利用符号链实现文件共享

- 优点
  - 当文件主删除一个共享文件时，其它共享文件的用户不会留下一个悬空指针。
  - 利用符号链可链接到世界上任何地方的机器中的文件。
- 缺点
  - 基于索引结点的共享方式可以直接从索引节点得到文件的物理地址，而利用符号链实现文件共享则需要根据给定的文件路径名去查找目录，启动磁盘的频率较高，使访问文件的开销增加。
  - 符号链本身也是一个文件，虽然很简单，但也需要配索引结点，耗费磁盘空间。

# 影响文件安全性的主要因素

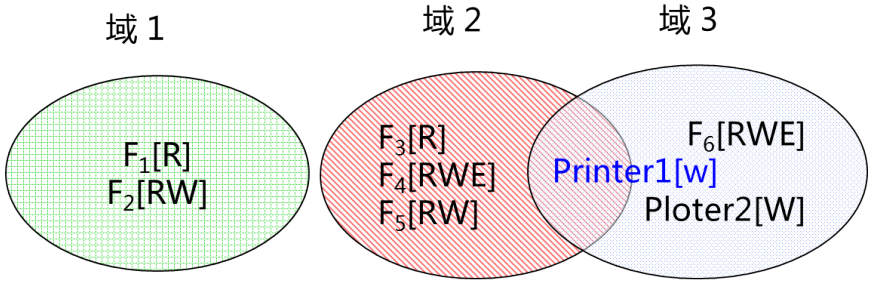
- 影响文件安全性的主要因素
  1. 人为因素
  2. 系统因素
  3. 自然因素
- 确保文件系统安全性的措施
  1. 存取控制机制 ------人为因素
     - 设置存取权限、文件属性、网络访问控制（过滤技术）
  2. 系统容错技术 ------系统因素
  3. 后备系统 ----------系统因素、自然因素

# 保护域

- 访问控制技术：现代操作系统中，几乎都配置了对系统资源进行保护的机制，引入了保护域和访问权的感念。
- 访问权
  - 为了对系统中的对象（包括硬件对象和软件对象）加以保护，由系统来控制进程对对象的访问。
  - 我们把一个进程能对某对象 (Object) 执行操作的权力称为访问权(Access Right)。每个访问权可以用一个有序对（对象名，权集）来表示。
  - 例如，某进程有对文件 F1 执行读和写操作的权力，这时可将该进程的访问权表示成 (F1，[R/W])。

# 保护域

- 保护域
  - 保护域是进程对一组对象的访问权的集合。
  - 进程仅在保护域内执行操作，保护域规定了进程所能访问的对象和能执行的操作。



# 进程与域之间的联系

- 静态联系
  - 静态联系是指进程的可用资源集在进程的整个生命中是固定的。进程和域之间一一对应，即一个进程只联系着一个域，这种域称为“静态域”。
- 动态联系
  - 动态联系是指进程的可用资源集在进程的整个生命中是变化的。进程和域是一对多的关系，即一个进程可联系着多个域。
  - 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。

# 访问矩阵 (Access Matrix)

- 访问矩阵：用以描述系统存取控制的矩阵。
- 访问矩阵中的行代表同一域，列代表同一对象，矩阵中每一项由一组访问权组成。
- R 读　W 写　E 执行
- 访问权 access(i,j) 定义了在域 Di中执行的进程能对对象 j施加的操作集。
- 访问权通常由资源的拥有者或管理者所决定。

| | 文件A | 文件B | 文件C | 打印机 |
|---|---|---|---|---|
| 域D1 | R,W | R | W | |
| 域D2 | R,W | E | | |
| 域D3 | | R,W | | W |

只有当进程在域D3中运行时才可以使用打印机

# 切换权 (switch)

- 为了能对进程进行控制，需要把进程从一个保护域到另一个保护域。
- 进程在保护域之间切换的权力，称为切换权 (switch)。
- 进程拥有切换权时，可把进程从一个保护域切换到另一保护域，以实现进程和域之间的动态联系。
- 切换权 switch∈access(i,j) 时，表示允许进程从域 i切换到域 j。

| | 文件A | 文件B | 文件C | 打印机 | 域D1 | 域D2 | 域D3 |
|---|---|---|---|---|---|---|---|
| 域D1 | R,W | R | W | | | S | |
| 域D2 | R,W | E | | | | | S |
| 域D3 | | R,W | | W | | | |

# 访问矩阵的修改：拷贝权

- 拷贝权 (Copy Right)
  - 拷贝权指进程在某个域中对某对象拥有的访问权可通过拷贝将访问权扩展到同一列（即同一对象）的其它域中。
  - 访问权access(i,j) 上加*者表示在域 i运行的进程可以把对对象 j的访问权复制到同一对象的任何域中。
  - 限制拷贝：拷贝后拷贝权*不能扩散。

| | F1 | F2 | F3 |
|---|---|---|---|
| D1 | E | | W* |
| D2 | E | R* | E |
| D3 | E | | |

→

| | F1 | F2 | F3 |
|---|---|---|---|
| D1 | E | | W* |
| D2 | E | R* | E |
| D3 | E | R | W |

# 访问矩阵的修改：所有权

- 所有权 (Owner Right))
  - 除了可以利用拷贝权把访问权进行有控制的扩散以外，系统还需要能删除某些控制权，我们可以利用所有权实现这些操作。
  - 如果在访问权access(i,j) 中包含所有权 (o)，则在域Di上运行的进程，可以增加或删除其在j 列上任何项中的访问权。换言之，进程可以增加或删除在任何其它域中运行的进程对对象 j 的访问权。

| | F1 | F2 | F3 |
|---|---|---|---|
| D1 | O,E | | W |
| D2 | | R*,O | R*,O,W |
| D3 | E | | |

→

| | F1 | F2 | F3 |
|---|---|---|---|
| D1 | O,E | | ~~W~~ |
| D2 | | R*,O, W* | R*,O,W |
| D3 | ~~E~~ | W | W |

# 访问矩阵的修改：控制权

- 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访问权。控制权则可用于改变矩阵内同一行 (域)中的访问权，即用于改变在某个域中运行进程对不同对象的访问权。
  - 如果在访问权access(i,j) 中包含了控制权 Control，则在域 Di中运行的进程可以删除在域 Dj中运行进程对各对象的任何访问权。

| | F1 | F2 | F3 | F4 | F5 | 打印机1 | 绘图仪2 | D1 | D2 | D3 |
|---|---|---|---|---|---|---|---|---|---|---|
| D1 | R | R,W | | | | | | | | |
| D2 | | | R,E | R,W | | W | | | | Control |
| D3 | | | | | R,W,E | W | W | | | |

# 访问矩阵的实现

- 在较大的系统中，访问矩阵将变得非常巨大，而且矩阵中的许多格可能都为空（稀疏矩阵），造成很大的存储空间浪费，因此在实际应用中访问控制很少利用访问矩阵的方式实现。
- 例如：如果系统中有 100 个域，$10^6$ 个对象，则访问矩阵将有 $10^8$ 个表项。如果每个表项占一个字节，访问矩阵的大小约为 100MB。
- 目前使用的访问控制实现技术不是保存整个访问矩阵，而是基于访问矩阵的列或者行来保存信息，分别形成访问控制表和访问权限表。

# 访问控制表 (Access Control List)

- 访问控制表：把访问矩阵按列（对象）划分，为每一列建立一张访问控制表 ACL。
- 访问控制表 ACL 由有序对（域，权集）组成。在该表中无原矩阵中的空项（被删除），所以 ACL 比较小，查找速度快。
- 当对象为文件时，常把访问控制表 ACL 存放于该文件的 FCB 或索引结点中，作为存取控制信息。
- 访问控制表可用于定义整个系统的缺省的访问权集。

# 访问权限表 (Capabilities List)

- 访问权限表：把访问矩阵按行（域）划分，每行形成一张访问权限表。
- 表中的每一项为该域对某对象的访问权限。
- 访问权限表不允许直接被用户 (进程) 所访问。访问权限表被存储到专用系统区内，只允许专用于进行访问合法性检查的程序访问，以实现对访问权限表的保护。

| 类型 | 权力 | 对象 |
|---|---|---|
| 文件3 | R - - | 指向文件3的指针 |
| 文件4 | RWE | 指向文件4的指针 |
| 文件5 | RW - | 指向文件5的指针 |
| 打印机1 | - W - | 指向打印机1的指针 |

# 访问矩阵的实现

- 大多数系统都同时采用访问控制表和访问权限表。
- 系统中为每个对象配置一张访问控制表。当一个进程第一次试图去访问一个对象时，必须先检查访问控制表，检查进程是否具有对该对象的访问权。
- 若有权访问，根据访问权限表为该进程建立一个访问权限，以后该进程便可直接利用这一返回的权限去访问该对象。
- 利用访问控制表和访问权限表可以快速地验证访问的合法性。当进程不再需要对该对象进行访问时，便可撤消该访问权限。

# 本章作业

# 谢 谢

*School of Computer & Information Engineering*
*Henan University*
*Kaifeng, Henan Province*
*475001*
*China*