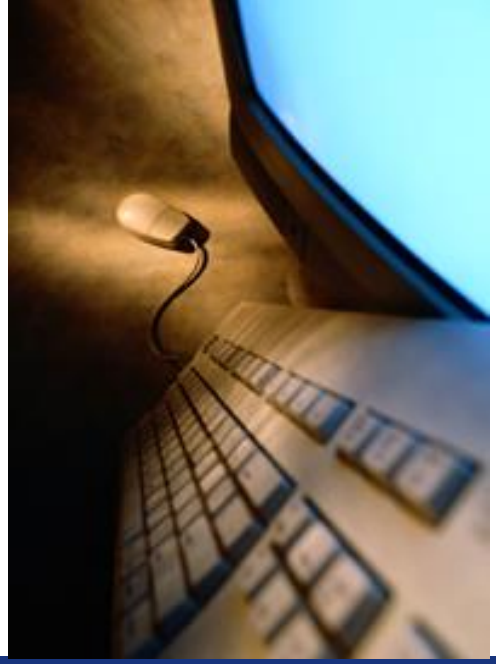
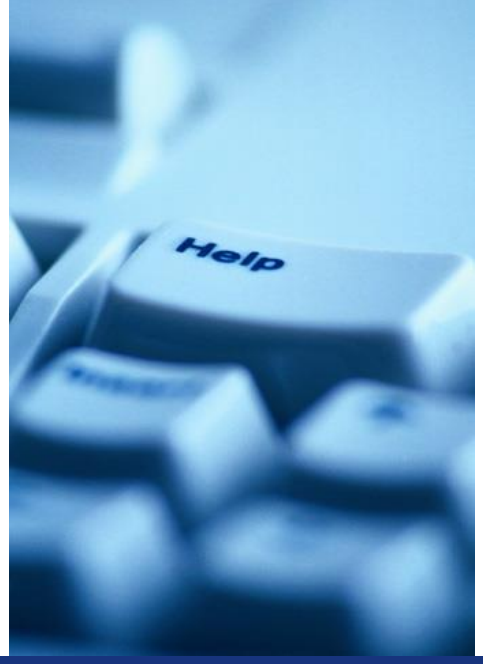



操作系统原理

Operating Systems Principles

张青
计算机学院

第十三讲 — 文件系统

目标

- 文件系统的功能；
- 文件系统接口；
- 文件系统的设计权衡，访问方法、共享、加锁以及目录结构；
- 文件系统保护；

文件

- 计算机可以在各种介质（如磁盘、磁带和光盘）上存储信息
- 为方便用户使用，操作系统提供了信息存储的统一逻辑视图
- 操作系统对存储设备的物理属性加以抽象，从而定义逻辑存储单位，即文件（**file**）。
- 文件的特点：

Long-term existence

- files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

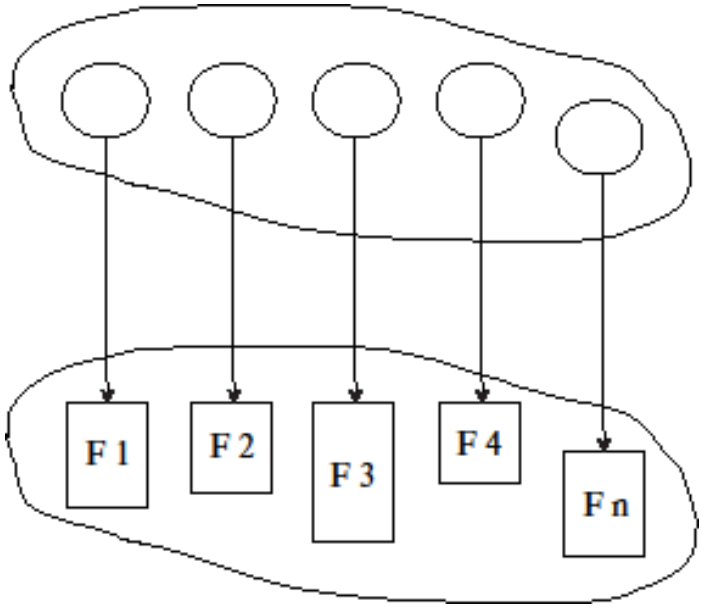
- files have names and can have associated access permissions that permit controlled sharing

Structure

- files can be organized into hierarchical or more complex structure to reflect the relationships among files

文件目录结构

- A collection of nodes containing information about all files

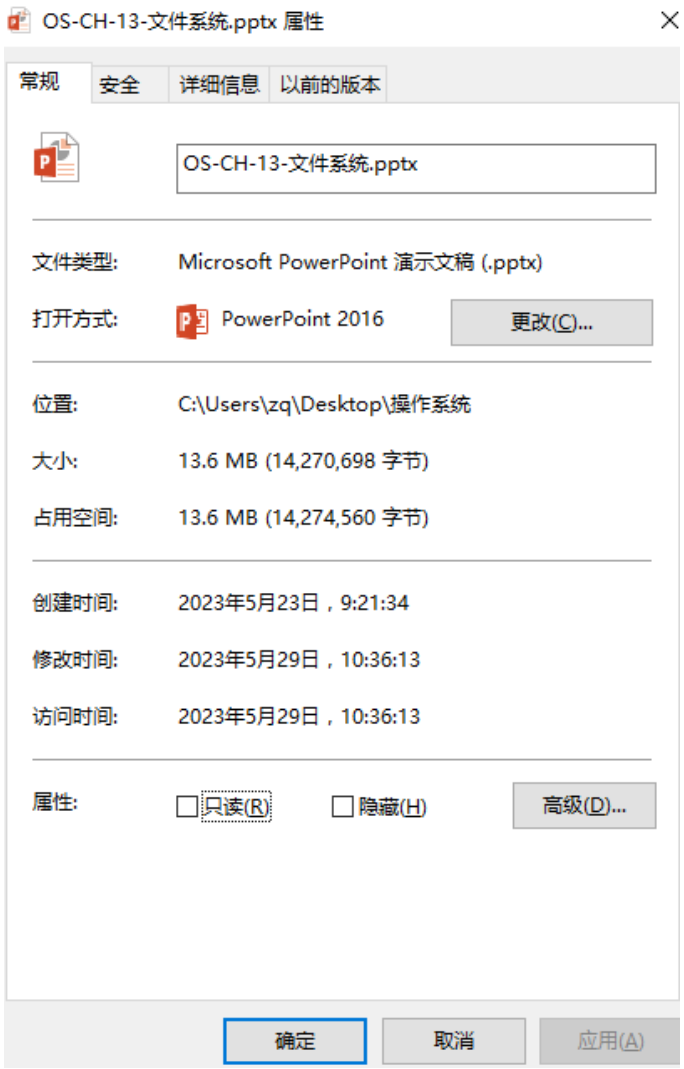


- Both the directory structure and the files reside on disk

文件属性

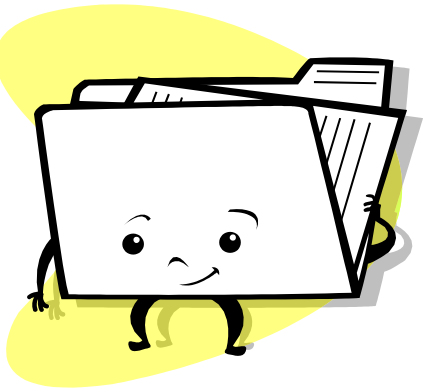
- 文件可存储不同类型的信息，包括源程序或可执行程序、数字或文本数据、照片、音乐、视频等等
- 文件的属性通常包括：
  - ✓ 名称
  - ✓ 标识符
  - ✓ 类型
  - ✓ 位置
  - ✓ 尺寸
  - ✓ 保护
  - ✓ 时间、日期和用户标识
  - ✓ 扩展文件属性（如文件的字符编码和安全功能）

Windows和Mac OS上的文件属性



文件操作

- 文件为抽象数据类型。为正确定义文件，操作系统提供了系统调用来对文件执行操作
- 最小的文件操作集合：
  - 创建文件
  - 删除文件
  - 读文件
  - 写文件
  - 重新定位文件：搜索目录寻找适当条目
  - 截断文件：删除文件内容，但保留它的基本属性



文件操作

- 操作系统有一个打开文件表（open-file table），用以维护所有打开文件的信息
- 每个打开文件具有如下关联信息：
  - ✓ 文件指针：跟踪读写位置，作为当前文件位置指针。对操作文件的每个进程都是唯一的，因而与磁盘文件属性分开保存
  - ✓ 文件打开计数：多个进程可能打开同一文件。文件打开计数跟踪打开和关闭的次数，在最后一个进程关闭这个文件时为0。然后，系统可以删除这个条目
  - ✓ 文件的磁盘位置
  - ✓ 访问权限

文件操作

- 文件锁：有的操作系统提供功能，用于锁定打开的文件。它允许一个进程锁定文件，以防止其他进程访问它
- 文件锁对于多个进程共享的文件很有用。例如，系统中的多个进程可以访问和修改的系统日志文件
- 文件锁提供类似于读者-写着锁
  - ✓ 共享锁类似于读者锁，以便多个进程可以并发获取它
  - ✓ 独占锁类似于写者锁，一次只有一个进程可以获取这样的锁
- 有的操作系统仅提供文件独占锁
- 操作系统可以提供强制（独占访问）或建议（非独占访问）文件锁定机制
- Java的文件锁定方法lock(),它用于获取锁：

```
FileLock lock(long begin, long end, Boolean shared)
```

其中begin和end是锁定区域的开始和结束位置。对于共享锁，shared为true时为独占锁，为false时释放锁

文件锁案例——Java

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
            RandomAccessFile raf = new RandomAccessFile("file.txt",
    "rw");
            // get the channel for the file
            FileChannel ch = raf.getChannel();
            // this locks the first half of the file - exclusive
            exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
            /** Now modify the data . . . */
            // release the lock
            exclusiveLock.release();
```

文件锁案例——Java

```
                // this locks the second half of the file - shared
                sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                        SHARED);
                /** Now read the data . . . */
                // release the lock
                sharedLock.release();
            } catch (java.io.IOException ioe) {
                System.err.println(ioe);
            }finally {
                if (exclusiveLock != null)
                exclusiveLock.release();
                if (sharedLock != null)
                sharedLock.release();
            }
        }
    }
```

这个程序获取文件file.txt的两个锁。文件锁的前半部分获取独占锁，后半部分的锁为共享锁

文件类型

- 操作系统使用扩展名来指示文件类型和可用于文件的操作类型

文件类型	常用扩展名	功能
可执行文件	exe,com,binor none	可运行的机器语言程序
目标文件	obj,o	已编译的、尚未链接的机器语言
源代码文件	c,cc,java,perl,asm	各种语言的源代码
批处理文件	bat,sh	命令解释程序的命令
标记文件	xml,html,tex	文本数据、文档
文字处理文件	xml,rtf,docx	各种文字处理程序的格式
库文件	lib,a,so,dll	为程序员提供的程序库
打印或可视文件	gif,pdf,jpg,png	打印或图形格式的ASCII或二进制文件
档案文件	rar,zip,tar	文件的集合，用于归档或存储
多媒体文件	mp3,mp4,avi	包含音视频等信息的二进制文件

Windows通过扩展名识别文件类型，Unix系统通过文件开始部分的magic数字来识别类型

文件结构

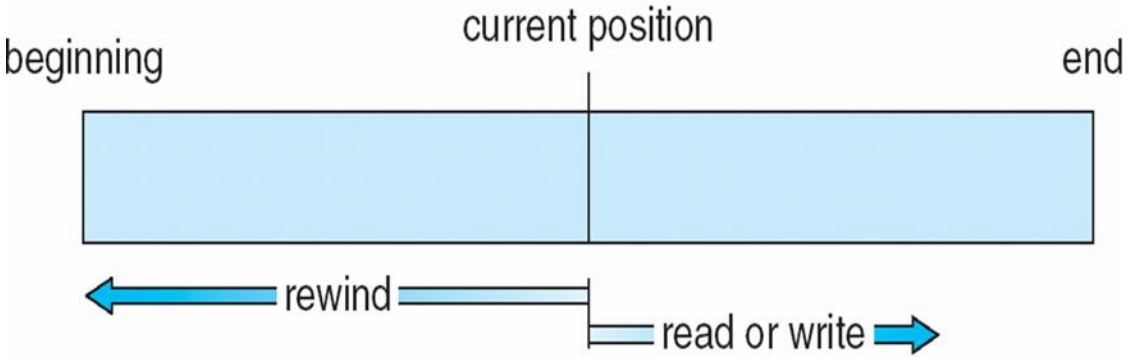
- 文件类型也可用于指示文件的内部结构
  - 如源文件和目标文件具有一定结构，以便匹配读取他们的程序的期望
- 让操作系统支持多个文件结构会使其变得复杂
  - 例如，假设有个系统支持两种类型的文件：文本文件和可执行的二进制文件。如果需要定义一个加密的文件，则上述两种类型的文件都不合适。加密文件不是ASCII文本行，而是随机位。另外，加密文件看起来是二进制文件，但不是可执行的。此时，要么绕过或滥用操作系统文件类型机制，要么放弃加密文件
- Unix、Windows采用强加最小数量的文件结构
  - Unix认为每个文件为8字节序列；操作系统并不对这些位做出解释。这种方案提供了最大灵活性，但支持的也很少。每个应用程序必须包含自己的代码，以便按适当结构来解释输入文件
- 系统必须至少支持可执行文件结构，以便能加载和运行程序

内部文件结构

- 在内部定位文件的偏移对操作系统来说是比较复杂的。因为物理记录大小不太可能刚好匹配期望的逻辑记录的长度
- 上述问题的常见解决方案：将多个逻辑记录包装到物理块中
- Unix操作系统将所有文件定义为简单的字节流
  - 每个字节通过距文件的开始（或结束）的偏移来单独寻址。此时，逻辑记录大小为1字节。根据需要，文件系统通常会自动将字节打包以存入物理磁盘块，或从磁盘块中解包得到字节
- 逻辑记录大小、物理块大小和打包技术确定了每个物理块可有多少个逻辑记录。打包可由用户应用程序或操作系统来完成
- 文件可以看作块的序列。所有基本I/O功能都以块为单位进行
- 由于磁盘空间总是按块为单位进行分配，因此每个文件的最后一块的某些部分通常会被浪费，产生内部碎片
  - 例如，每个块是512字节，则1949字节的文件将分得4个块（2048字节），最后的99个字节就浪费了

顺序访问

- 文件存储信息。使用文件时，必须访问这些信息，并将其读到计算机内存。
- 文件信息可按多种方式来访问：顺序访问、直接访问、其他访问方法
- 顺序访问：文件信息按顺序（即一个记录接着一个记录的形式）加以处理。这种访问模式是目前最常见的
  - 例如，编辑器和编译器通常以这种方式访问文件
  - 读和写构成文件的大部分操作，涉及read_next()—读取文件的下一部分，并自动前移文件指针以跟踪I/O位置，write_next()—对文件结尾附加内容，并前移到新写材料的末尾（文件的新结尾）



直接访问

- 直接访问或相对访问，文件由固定长度的逻辑记录组成，以允许程序按任意顺序进行快速读取和写入记录
- 对于直接访问文件的读取或写入的顺序没有限制
  - 直接访问方法基于文件的磁盘模型，因为磁盘允许对任何文件块的随机访问。此时可以实现诸如先读取块14，再读取块53，最后再写入块7的文件访问
  - 对于大量信息的立即访问，直接访问文件极为有用。数据库通常是这种类型的。当需要查询特定主题时，首先计算哪个块包含答案，然后直接读取相应块以提供需要的信息
- 对于直接访问，需修改文件操作以包含块号作为参数
  - 有read(n),而不是read_next()；有write(n),而不是write_next()
  - 用户提供给操作系统的块号，通常为相对块号。相对块号是相对文件开头的索引，这种设计是防止用户访问不属于文件的其他文件的块

直接访问

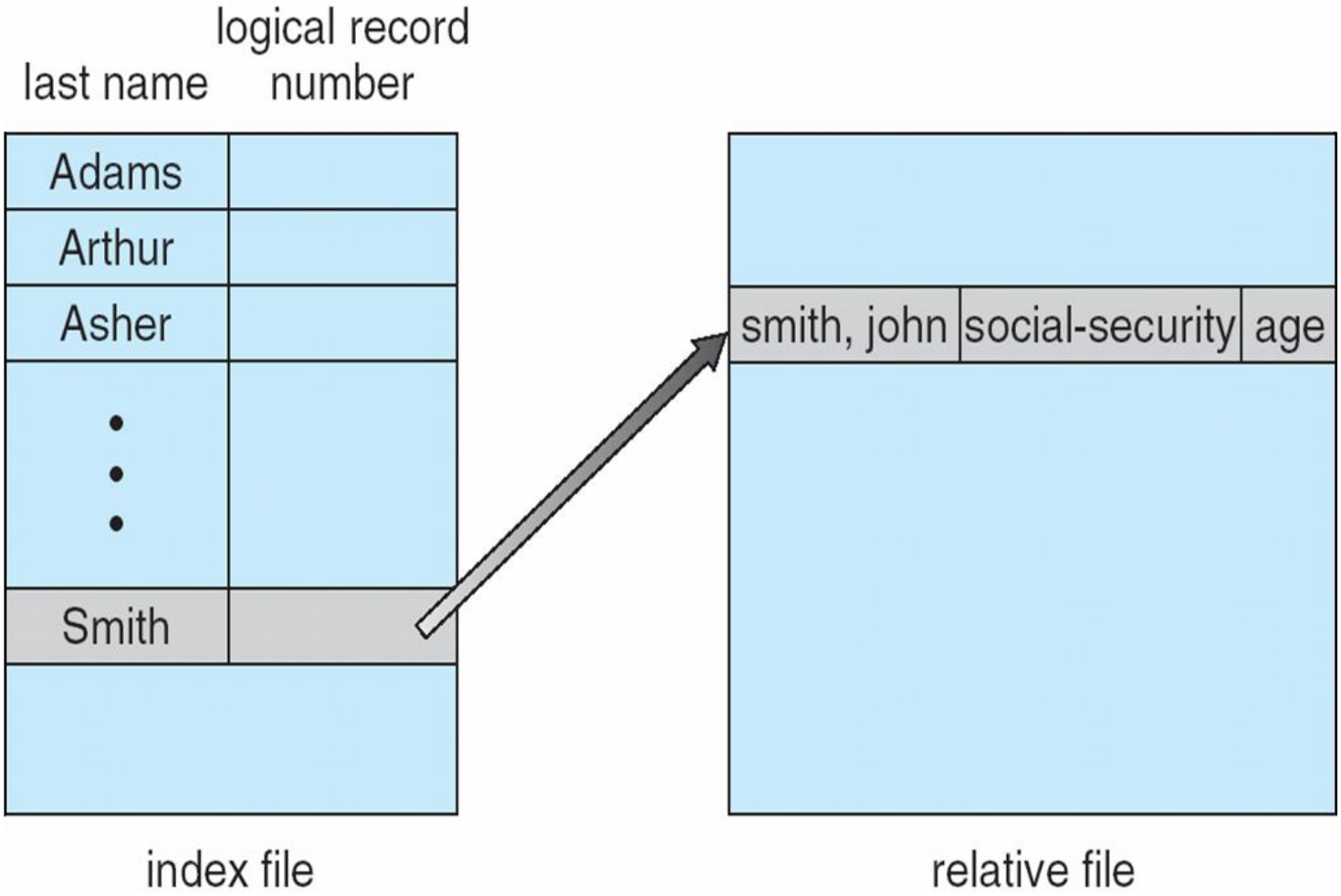
- 并非所有操作系统都支持文件的顺序和直接访问，有的只允许顺序文件访问，有的只允许直接访问
- 对于支持直接访问的系统，可通过简单保持当前位置的变量cp，轻松模拟顺序访问
- 基于顺序访问模拟直接访问是非常低效和笨拙的

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

其他访问方法

- 其他访问方法可以建立在直接访问之上。这些访问通常涉及创建文件索引，可以实现高效搜索巨大文件
- 对于大文件，索引文件本身可能变得太大而无法保存在内存中。一种解决方案是为索引文件创建索引
  - 此时主索引文件包含指针，以指向辅助索引文件；而辅助索引文件包括指针，以指向实际的数据项
  - 例如，IBM的索引顺序访问方法采用小主索引，以指向辅助索引的磁盘块；辅助索引块指向实际文件块；而文件按定义的键来排序。为了找到特定的数据项，首先二分搜索主索引，以得到辅助索引的块号。读入该块，再次通过二分搜索找到包含所需记录的块。最后，按顺序搜索该块。这样，通过记录的键通过至多两次的直接访问就可定位记录

索引文件和相关文件的例子

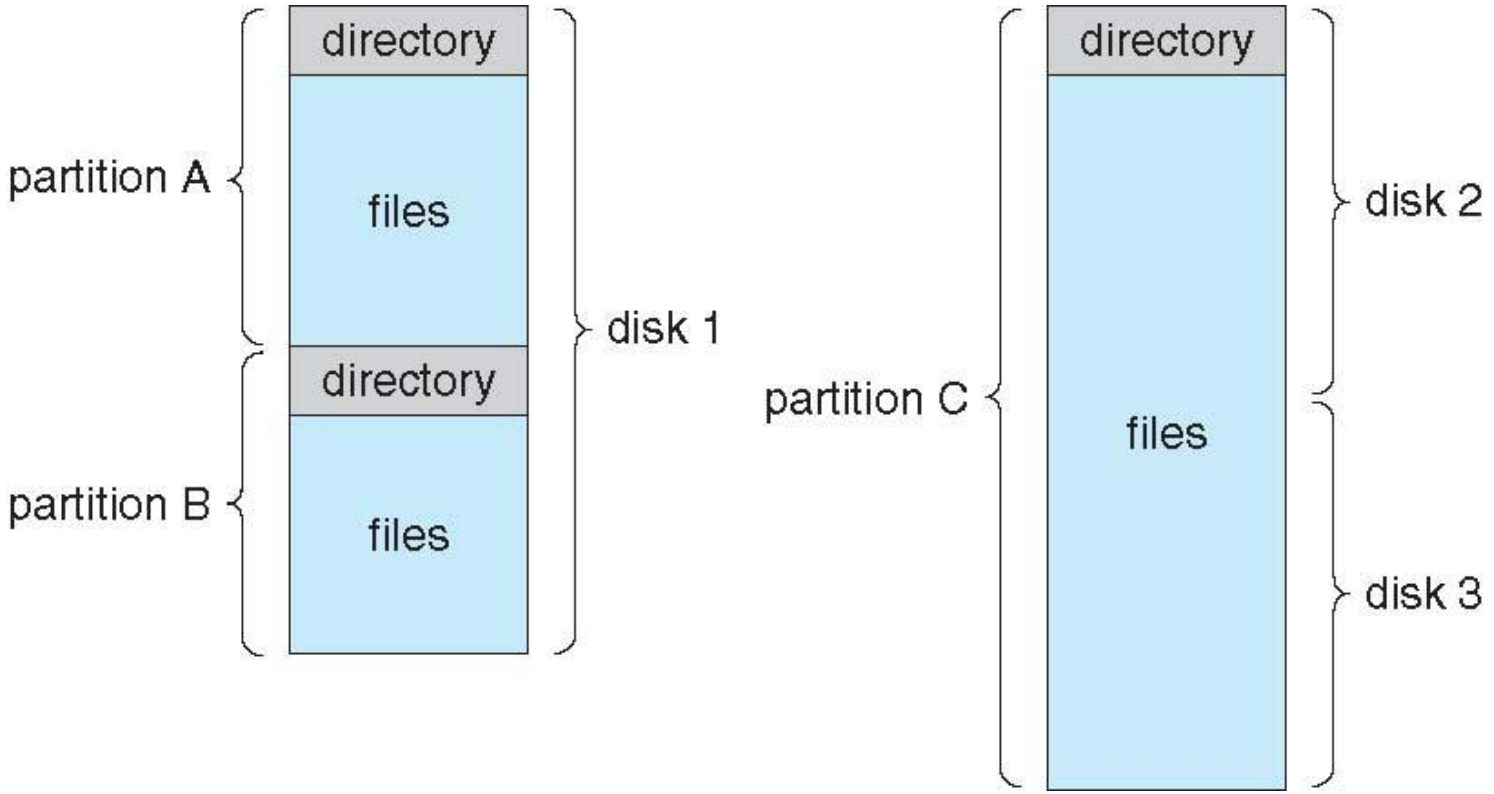


目录与磁盘的结构

- 每台计算机通常存有数千、数百万，甚至数十亿个文件。文件存储在随机存取存储设备上，包括硬盘、光盘和固态盘
- 一个存储设备可以按整体来用于文件系统；它也可以细分，以提供更细粒度的控制
  - 例如。一个磁盘可以划分为4个分区，每个分区可以有单独的文件系统。存储设备还可以组成RAID集，一起提供保护以免受单个磁盘故障
- 分区可用于限制单个文件系统的大小
  - 可将多个类型的文件系统放在同一设备上，或留下设备的一部分以为他用，例如交换空间或未格式化（原始）的磁盘空间
- 硬盘的每个分区都可用于创建文件系统
- 包含文件系统的分区通常称为卷
  - 卷可以是设备的一部分，或整个设备，或由多个设备组成的RAID集。每个卷可以作为虚拟磁盘。卷还可以存储多个操作系统，以允许引导和运行多个操作系统

典型的文件系统结构

- 包含文件系统的每个卷应包含有关系统内的文件信息。这些信息保存在设备目录或卷目录表中，设备目录（更常称为目录）记录卷上的所有文件的信息，如名称、位置、大小和类型等



存储结构

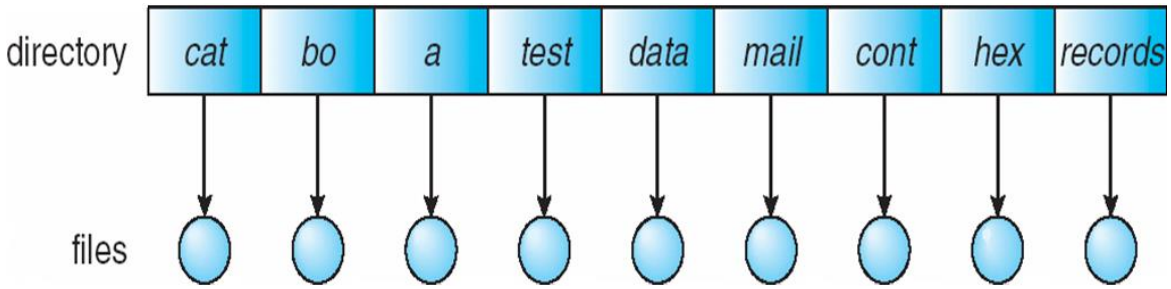
- 计算机系统可能没有文件系统，也可能有多个文件系统，而且文件系统的类型可能不同
- Solaris中常用的文件系统类型：
  - tmpfs：“临时”文件系统。它是在易失性内存中创建的，当系统重启或崩溃时，它的内容会被擦除
  - objfs:“虚拟”文件系统。本质上，这是一个内核接口，但看起来像一个文件系统；它让调试器访问内核符号
  - ctfs：维护“合同”信息的虚拟文件系统，以管理哪些进程在系统引导时启动并且在运行时必须继续运行
  - lofs：“环回”文件系统，以允许一个文件系统代替另一个来被访问
  - procfs：虚拟文件系统，将所有进程信息作为文件系统来呈现
  - ufs,zfs：通用文件系统
- 计算机的文件系统可以极大。在单个文件系统中，将文件分成组并且管理和操作这些组，是很有用的。这种组织使用目录

目录

- 目录可视为符号表，可将文件名称转成目录条目。如采取这种观点，则可按许多方式来组织目录。这种组织方式运行插入条目、删除条目、搜索命名条目以及列出所有目录条目等
- 对于目录结构，有如下操作：
  - 搜索文件：搜索目录结构，查找特定文件的条目
  - 创建文件：创建新的文件，并添加到目录
  - 删除文件：从目录中删除文件
  - 遍历目录：遍历目录内的文件，及其目录内每个文件的目录条目的内容
  - 重命名文件
  - 遍历文件系统：访问每个目录和目录结构内的每个文件
- 目录逻辑结构常见方案：单级目录、两级目录、树形目录、无环图目录、通用图目录

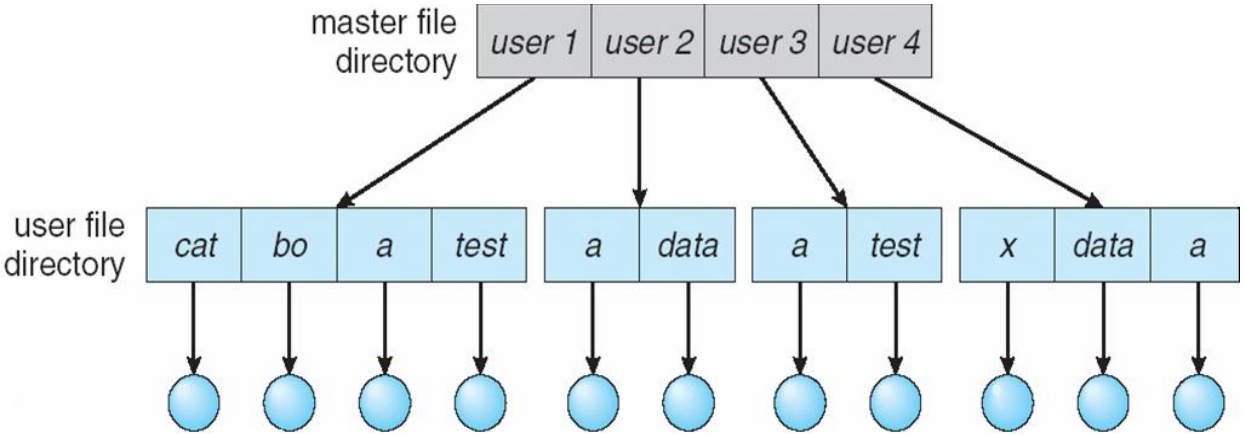
单级目录

- 最简单的目录结构是单级目录。此时所有文件都包含在一个目录中，这很容易支持和理解
- 当文件数量增加或系统有多个用户时，单级目录会因为所有文件处于同一目录导致难以选择唯一的文件名称，或因为文件太多导致用户难以记住所有文件的名称，使得使用难度提升



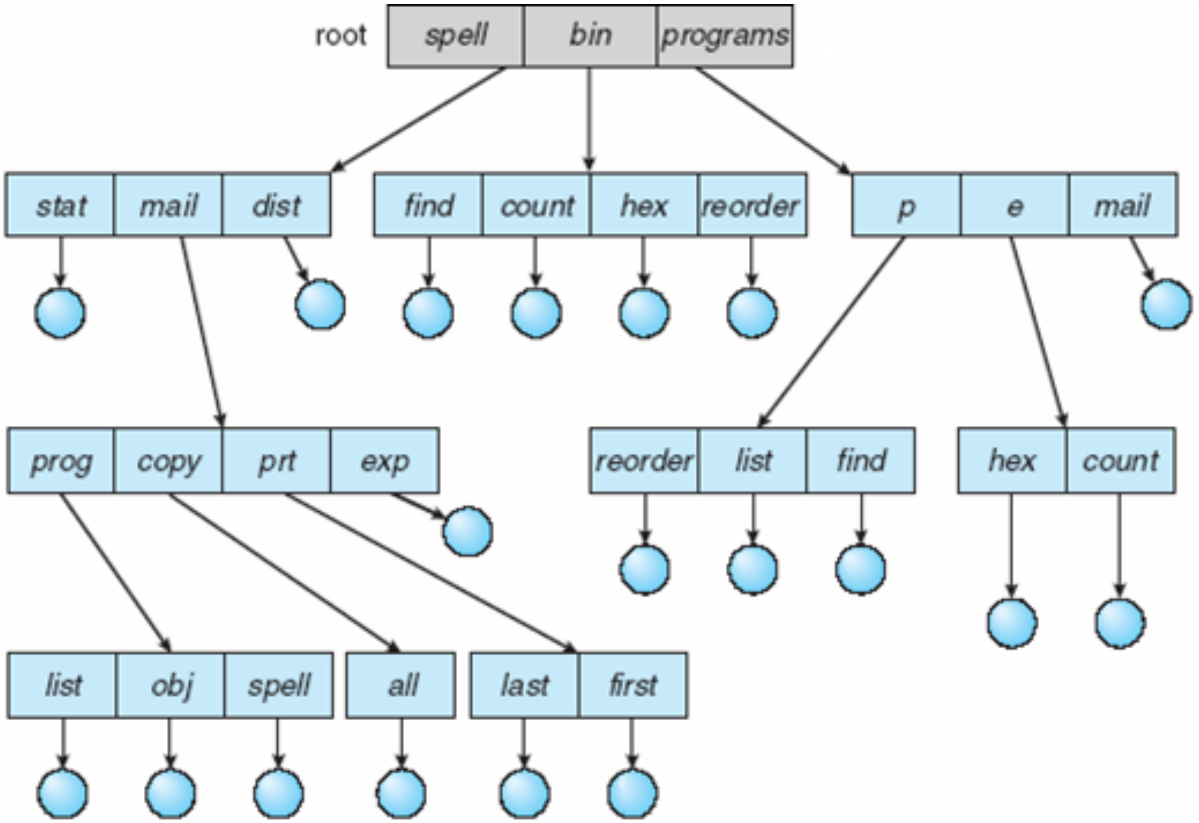
两级目录

- 单级目录常常导致混乱的文件名
- 标准的解决方案是为每个用户创建一个单独的目录
- 两级目录中，每个用户都有自己的用户文件目录（UFD）。这些UFD具有类似的额结构，但是只列出了单个用户的文件
  - 当用户作业开始或用户登录时，搜索系统的主文件目录（MFD）。通过用户名或账户可索引MFD，每个条目指向该用户的UFD
  - 两级目录结构通过隔离用户解决了名称碰撞问题，但当用户需要在某个任务上进行合作并且访问彼此的文件时，隔离变为缺点



树形目录

- 通过将两级目录视为两级的树，自然的推广是将目录结构扩展到任意高度的树

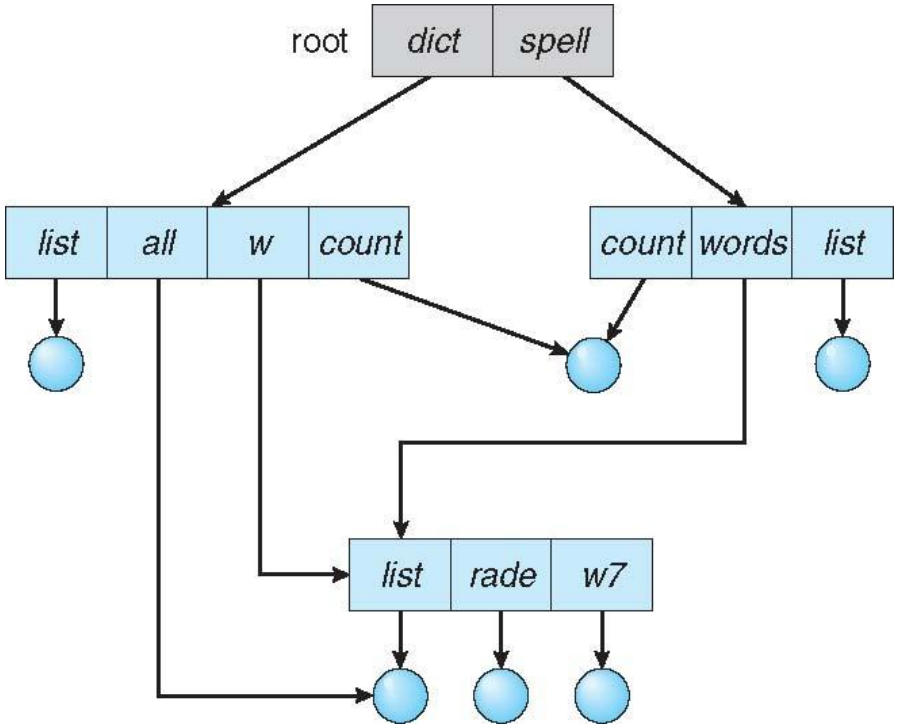


树形目录

- 允许用户创建子目录并相应地组织文件。树是最常见的目录结构，有一个根目录，系统内的每个文件都有唯一的路径名
- 每个进程都有一个当前目录（current directory），它包含进程当前感兴趣的大多数文件
- 路径名有两种形式：绝对路径名和相对路径名
  - 绝对路径名从根开始，它给出路径上的所有目录名
  - 相对路径名从当前目录开始，定义一个路径
- 树形目录需要关注如何处理删除目录：目录为空时才允许删除或直接删除目录文件及其子目录文件，后者更危险
- 采用树形目录，用户除了可以访问自己的文件外，还可以访问其他用户的文件
  - 用户B可以通过指定用户A的路径名，来访问用户A的文件。用户B可以使用绝对路径名或相对路径名。或者，用户B将其当前目录改变为用户A的目录，进而直接采用文件名来访问文件

无环图目录

- 树结构禁止共享文件或目录
- 无环图，即没有循环的图，允许目录共享子目录和文件。同一文件或子目录可出现在两个不同目录中
- 一个用户所做的任何更改都会立即为其他用户看到



无环图目录

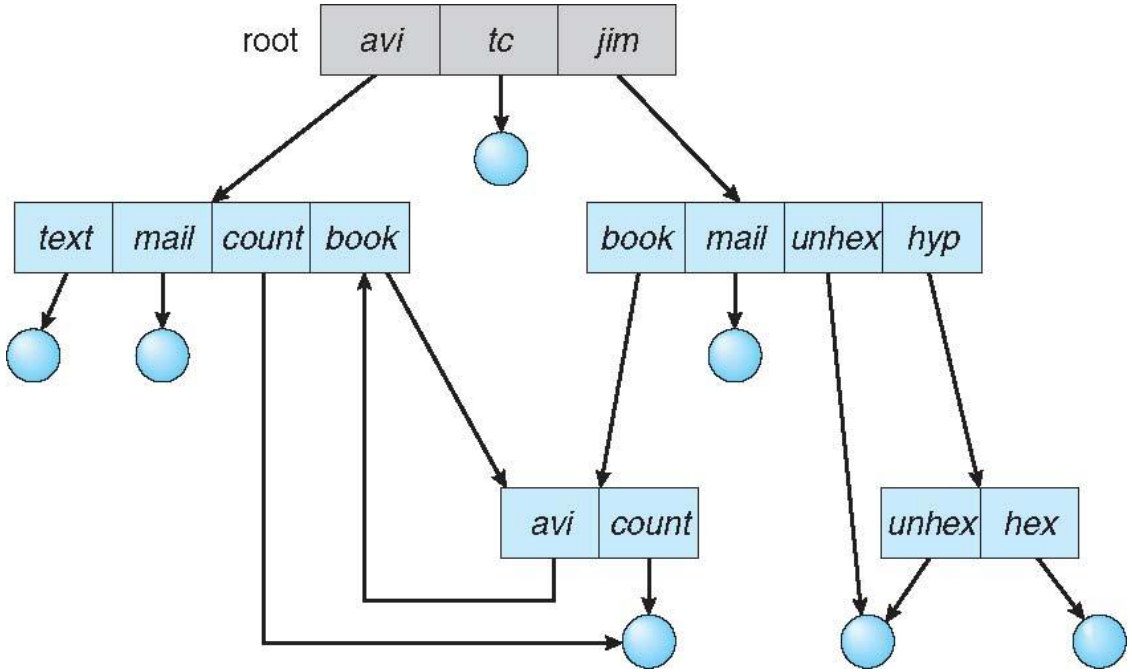
- **Two different names (aliasing)**
- **If *dict* deletes w/*list* ⇒ dangling pointer**

  Solutions:
  - Backpointers, so we can delete all pointers.
    - Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- **New directory entry type**
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

软链接、硬链接

通用图目录

- 采用无环图结构的一个严重问题是确保没有环，以避免重复遍历无环图的共享部分
  - 只允许链接到文件而不是子目录
  - 垃圾收集，以确定什么时候可删除某个文件
  - 每次添加新链接使用循环检测算法判断是否OK



文件系统安装

- 文件系统在用于系统的进程之前必须先安装
- 目录结构可以构建在多个卷上，这些卷必须先安装才能使其可用于文件系统名称空间
- 安装过程如下：
  - 操作系统要知道设备的名称和安装点（mount point）。通常安装点是空目录。如，在Unix系统中，包含用户主目录的文件系统可能按照到/home
  - 然后，当访问该文件系统的目录结构时，只要在目录名称之前加上/home，如/home/jane。当该文件系统按照在/users下时，通过路径名/users/jane可以使用同一个目录
  - 接下来，操作系统验证设备包含一个有效的文件系统。验证方式如下：通过设备驱动程序读入设备目录，并验证目录具有预期格式。最后，操作系统在其目录结构中记录一个文件系统已安装在给定安装点上

文件共享

- 多用户：当操作系统支持多个用户时，文件共享、文件命名和文件保护等问题就尤其突出
- 为了实现共享与保护，多用户系统必须要比单用户系统维护更多的文件和目录属性
- 大多数系统都采用了文件所有者（或用户）和组的概念
- 远程文件系统：利用计算机网络通信来进行文件共享，包括ftp（用于匿名和认证的访问）、分布式文件系统（DFS）、万维网（完全采用匿名文件交换）
  - 客户机-服务器模型
  - 分布式信息系统
  - 故障模式

一致性语义

- 一致性语义是个重要准则，用于评估支持文件共享的文件系统
- 这些语义规定系统的多个用户如何访问共享文件。如，一个用户的数据修改何时为另一用户可见
- Unix语义
  - 一个用户对已打开文件的写入，对于打开同一文件的其他用户立即可见
  - 一种共享模式允许用户共享文件的当前位置指针。因此，一个用户前移指针就回影响所有共享用户。这里，一个文件具有单个图像，允许来自不同用户的交替访问
- Andrew文件系统采用以下一致性会话语义：
  - 一个用户对已打开文件的写入，对于打开同一文件的其他用户，不是立即可见
  - 一旦文件关闭，对齐所做的更改只能被后来打开的会话可见。已打开的文件实例并不反映这些变化

保护

- 当信息存储在计算机系统中时，需要保护它的安全，以便避免物理损坏和非法访问
- 访问类型：通过限制可以进行的文件访问类型，保护机制提供受控访问。可以控制多个不同的操作类型：读、写、执行、附加、删除、列表
- 访问控制：根据用户身份控制访问，不同用户可以需要不同类型的文件或目录访问。
  - 每个文件和目录关联一个访问控制列表，涉及三种用户类型：所有者、组、其他用户
- 必须严格控制许可和访问的列表。比如，对于Unix系统，只有管理员和超级用户可以创建和修改组
- 其他保护方式：为每个文件加上一个密码。缺点是用户需要记住的密码数量可能太多

小结

- 文件概念、属性、操作、类型、结构
- 文件访问方法：顺序访问、直接访问、其他访问方法
- 存储结构：单级目录、两级目录、树形目录、无环图目录、通用图目录
- 文件系统安装
- 文件共享：多用户、远程文件系统、一致性语义
- 保护：访问类型、访问控制

谢谢