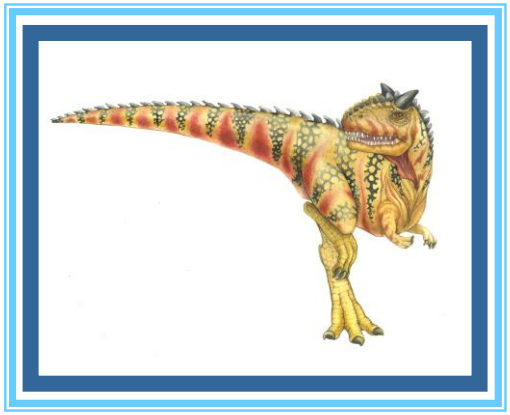
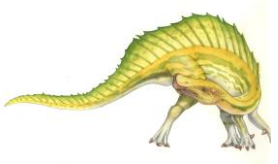
Chapter 13: File-System Interface



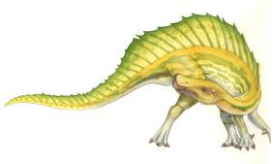
Outline

- File Concept
- Access Methods
- Disk and Directory Structure
- Protection
- Memory-Mapped Files



Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

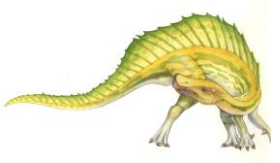


File Concept

- Contiguous logical address space
- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program
- Contents defined by file's creator
  - Many types
    - **text file,**
    - **source file,**
    - **executable file**

File Attributes

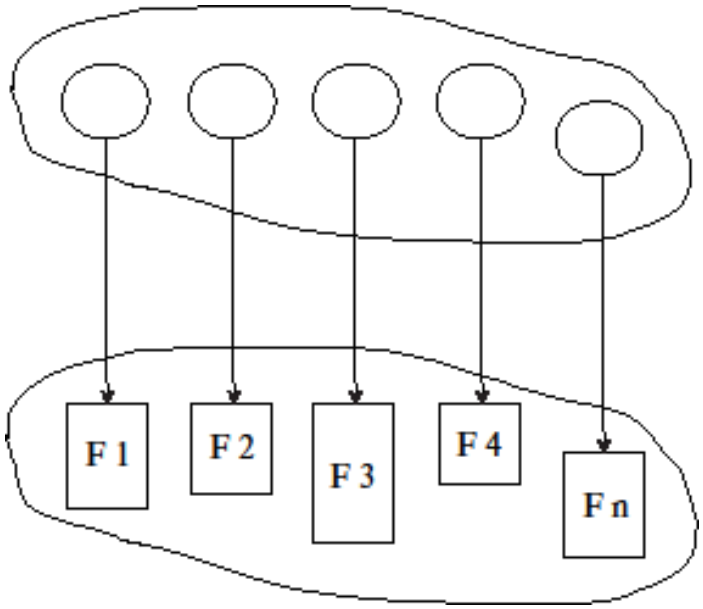
- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Many variations, including extended file attributes such as file checksum
- Information about files is kept in the directory structure, which is maintained on the disk



File info Window on Mac OS X

Directory Structure

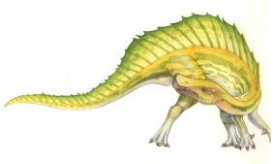
- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

File Operations

- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open*** **(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- ***Close*** **(F_i)** – move the content of entry F_i in memory to directory structure on disk



Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
    - Typically two levels: per-process table, system-wide table
  - Per-process table keeps file descriptor flags and file pointer to the system-wide table
  - System-wide table keeps the following
    - File position: points to last read/write location
    - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
    - Disk location of the file: cache of data access information
    - Access rights: per-process access mode information
  - For Linux:
    - Two file descriptors in same process can point to the same entry in the system-wide open file table, by using dup(2)
    - Two processes could have the same file descriptor pointing to the same entry in system-wide table (e.g., parent-child share the same fd, or special fd's like 0, 1, 2)
    - Two independent processes calling open() will create separate entries in the system-wide table

File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently (no one can claim exclusive lock on it)
  - **Exclusive lock** similar to writer lock (no one can claim any lock)
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested (OS enforced, example: Windows)
  - **Advisory** – processes can find status of locks and decide what to do (programmer enforced, example Unix)

File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false; // false represents EXCLUSIVE
    public static final boolean SHARED = true;
    public static void main(String args[]) 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();
            // 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();
        } } }
```

File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - Operating system
    - Modern operating systems impose minimal number of file structures, but they must support at least one structure – that of an executable file
  - Program

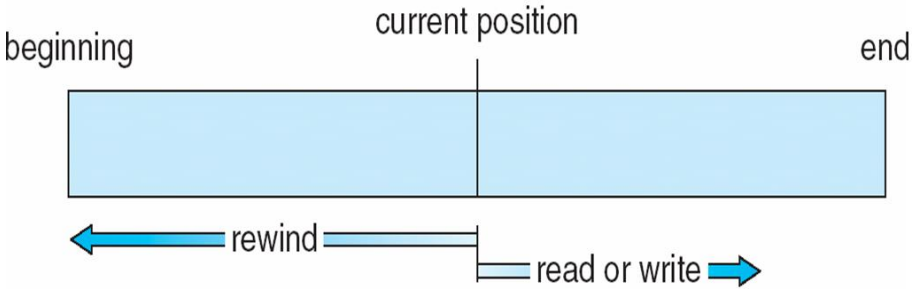
Access Methods

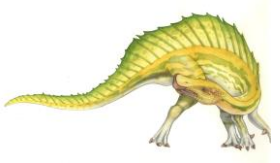
- A file can be accessed in several ways
  - **Sequential Access**
  - **Direct Access**
  - **Other Access Methods**

Sequential Access

- Operations
  - **`read next`**
  - **`write next`**
  - **`Reset`**
  - no read after last write
    - A write always appends to the end of the file and reposition pointer at the new end of file

- Figure





Direct Access

- Operations
  - **`read n`**
  - **`write n`**

  Equivalent to
  - **`position to n`**
    - **`read next`**
    - **`write next`**

  n = **relative block number** (index relative to beginning of file)
- Relative block numbers allow OS to decide where file should be placed

Simulation of Sequential Access on Direct-access File

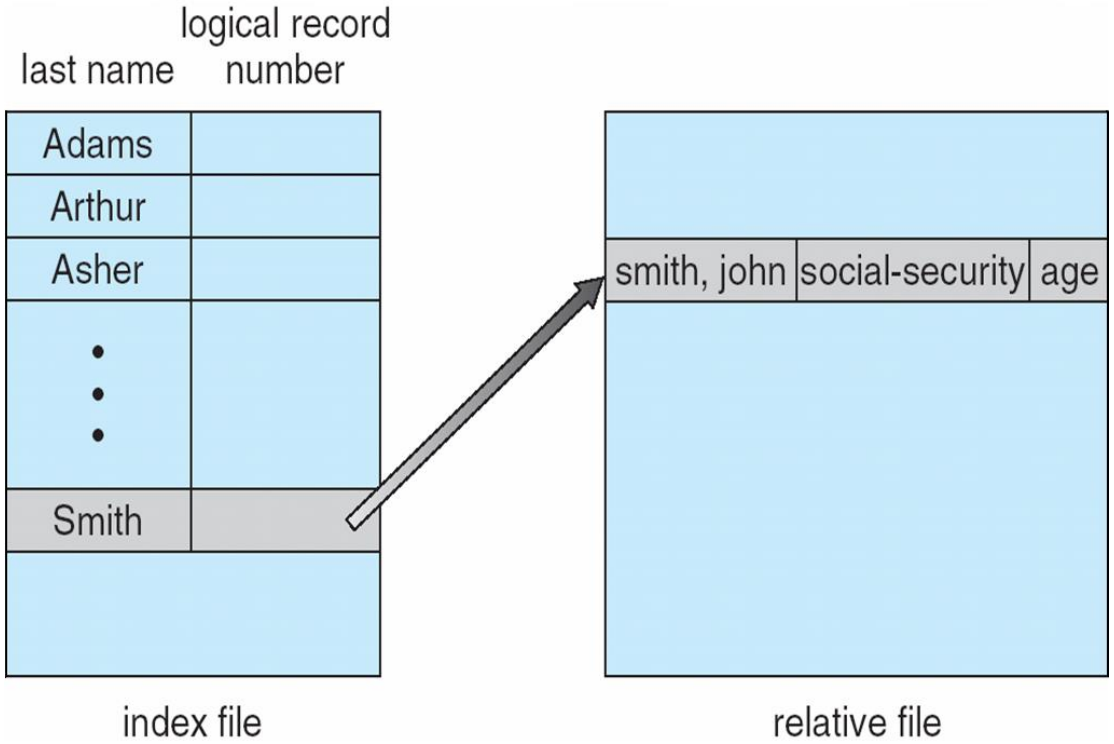
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Here cp is the current pointer
Simulation of direct-access on sequential access involves a lot of resetting, thus is very inefficient

Other Access Methods

- Other access methods can be built on top of base methods
- Generally involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on
  - The index file kept sorted on a defined key
  - Index searched on via binary search
  - All done by the OS
- VMS operating system provides index and relative files as an example (see next slide)

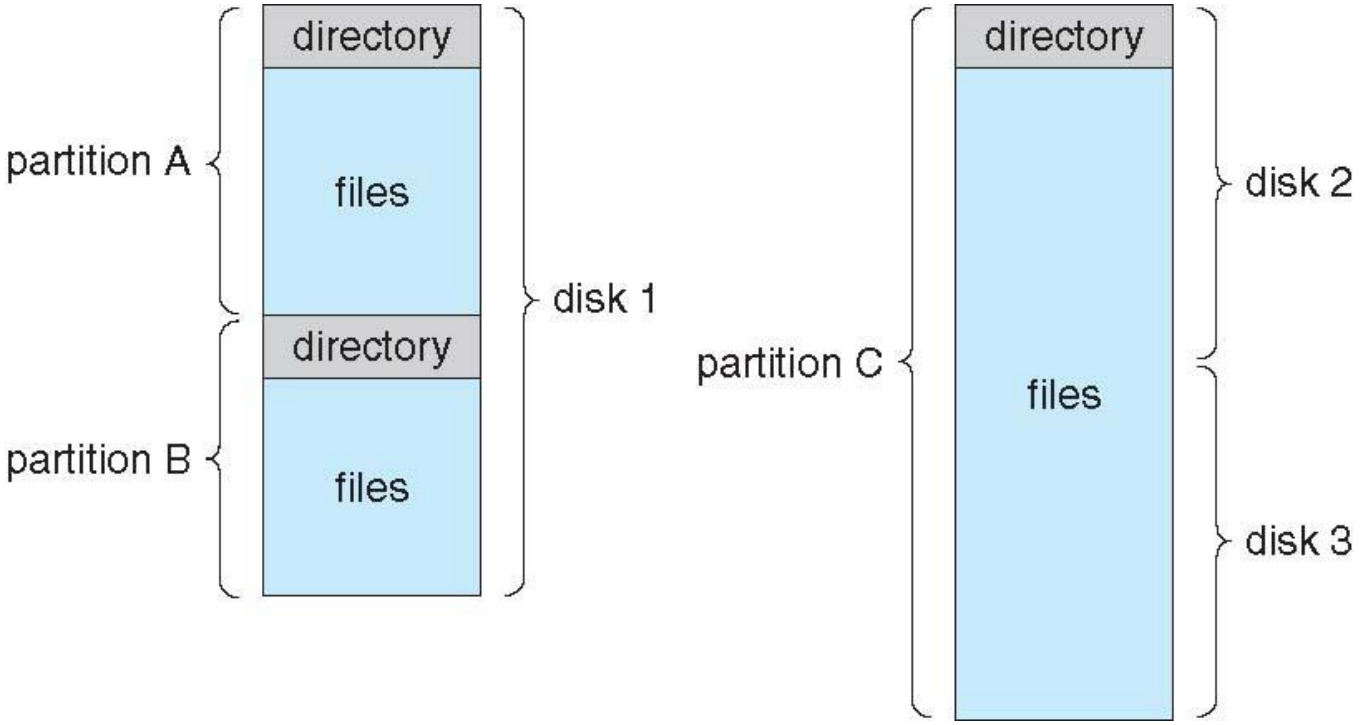
Example of Index and Relative Files



Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system is known as a **volume**
- Each volume containing a file system also tracks that file system's info in **device directory** or **volume table of contents**
- In addition to **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization

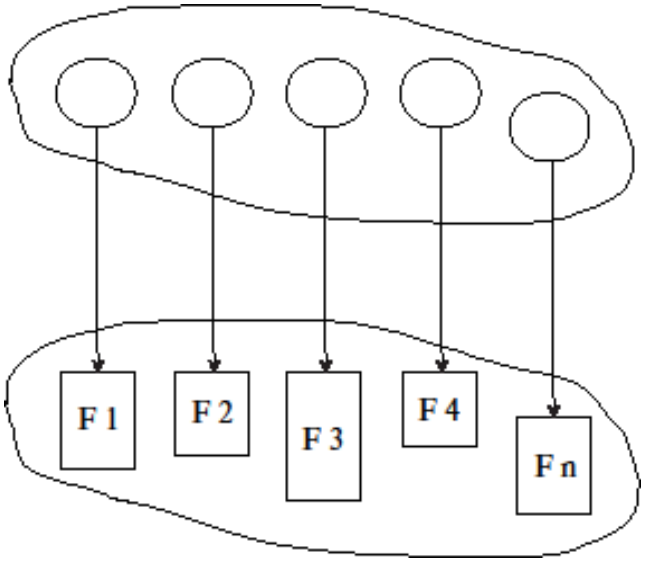


Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have many file systems, some general- and some special- purpose
- Consider Solaris, it has
  - **tmpfs** – memory-based volatile FS for fast, temporary I/O
  - **objfs** – interface into kernel memory to get kernel symbols for debugging
  - **ctfs** – contract file system for managing daemons (startups when booting)
  - **lofs** – loopback file system allows one FS to be accessed in place of another (ex: using an .iso file and mounting it as a FS without a CD drive)
  - **procfs** – kernel interface to process structures
  - **ufs, zfs** – general purpose file systems

Directory Structure

- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

Operations Performed on Directory

- Search for a file
  - Able to find all files whose names match a particular pattern
- Create a file
- Delete a file
  - File system may need a method to defragment the directory structure
- List a directory
- Rename a file
- Traverse the file system
  - Commonly needed in backing up all files

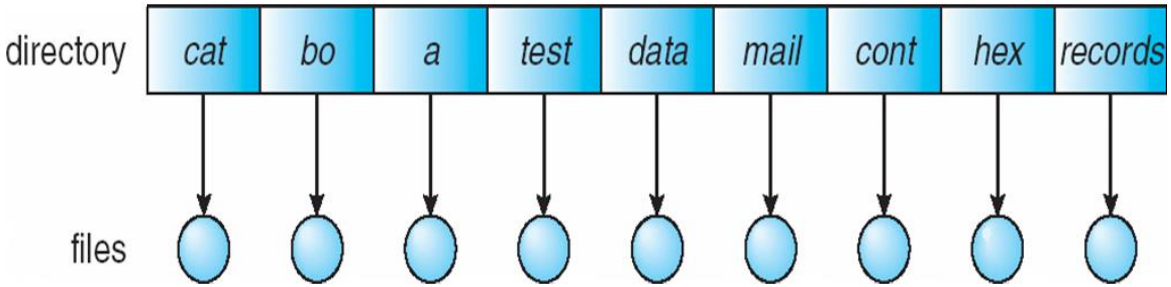
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

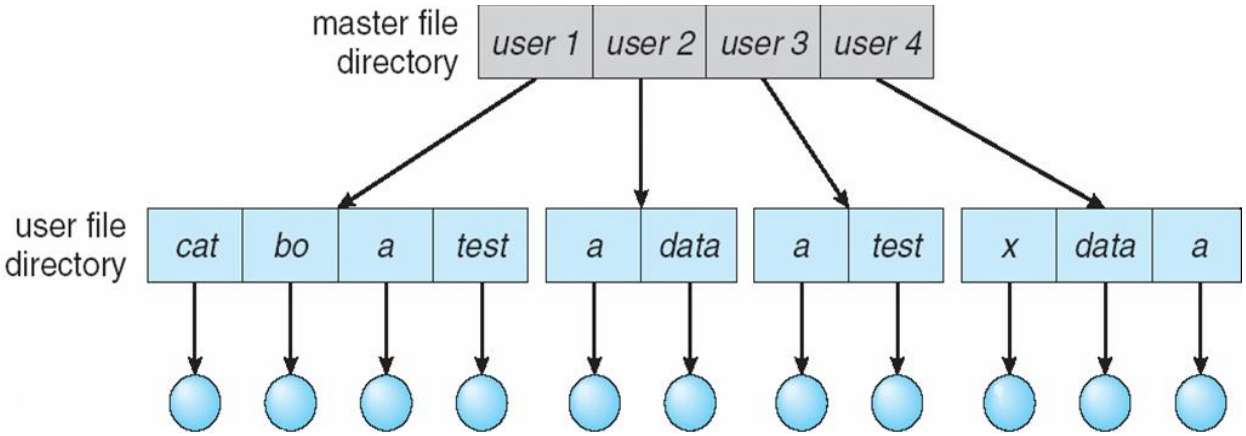
- A single directory for all users



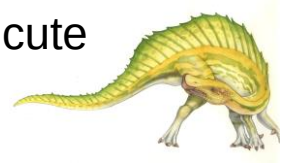
- Naming problem
  - Must have unique names
  - Use long file names (up to 255 characters)
  - Even a single user may find it difficult to remember the names of all files.

Two-Level Directory

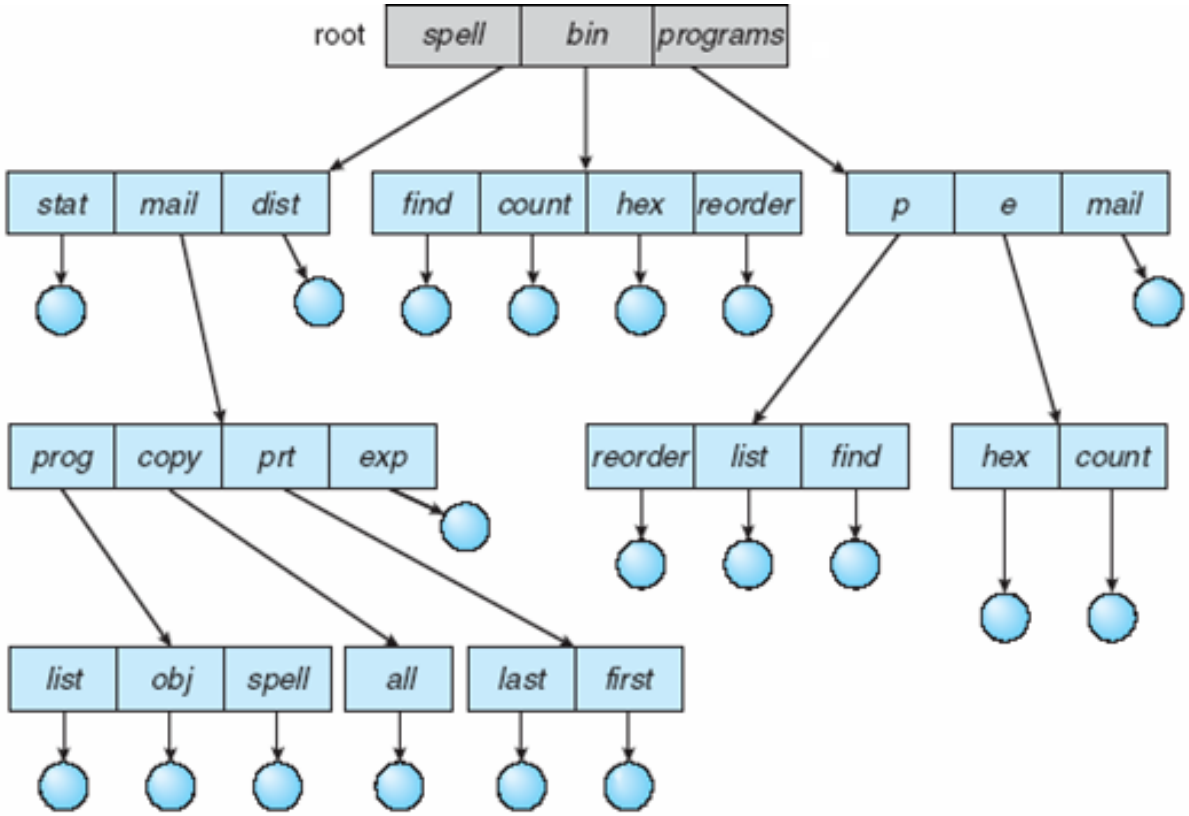
- Separate directory for each user



- A user's reference to a particular file is only searched in his own directory
- Can have the same file name for different user
- Efficient searching
- Can't have user groups accessing each other's files
- Use a path to differentiate files of same names
- Use a special user directory for system files
  - Use a search path to define order of which program to load/execute

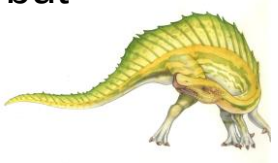


Tree-Structured Directories



Tree-Structured Directories (cont.)

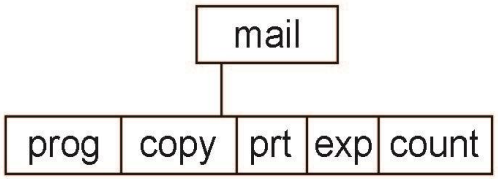
- Most common directory structure
- Every file in the system has a unique path name
- In many implementations, a directory is just another file, but treated specially
- One bit in each directory entry define it as a file (0) or directory (1)
- Each process has a current directory which can be changed programmatically
- The initial current directory of a user's login shell is used when the user job starts or the user logs in, and is stored in a predefined location
  - The child process inherits the current directory of the parent
- Path names can be absolute or relative
  - Absolute path starts with '/' (Unix and Linux)
  - Relative path starts with the current directory
- Policies on deleting a directory
  - Not allowed if a directory is not empty - safer but inconvenient
  - Use a special option on the command (Unix: rm -r) – dangerous but convenient

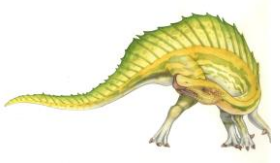


Current Directory

- Can designate one of the directories as the current (working) directory
  - **`cd /spell/mail/prog`**
  - **`type list`**
- Creating and deleting a file is done in current directory
- Example of creating a new directory
  - If the current directory is **`/mail`**
  - The command

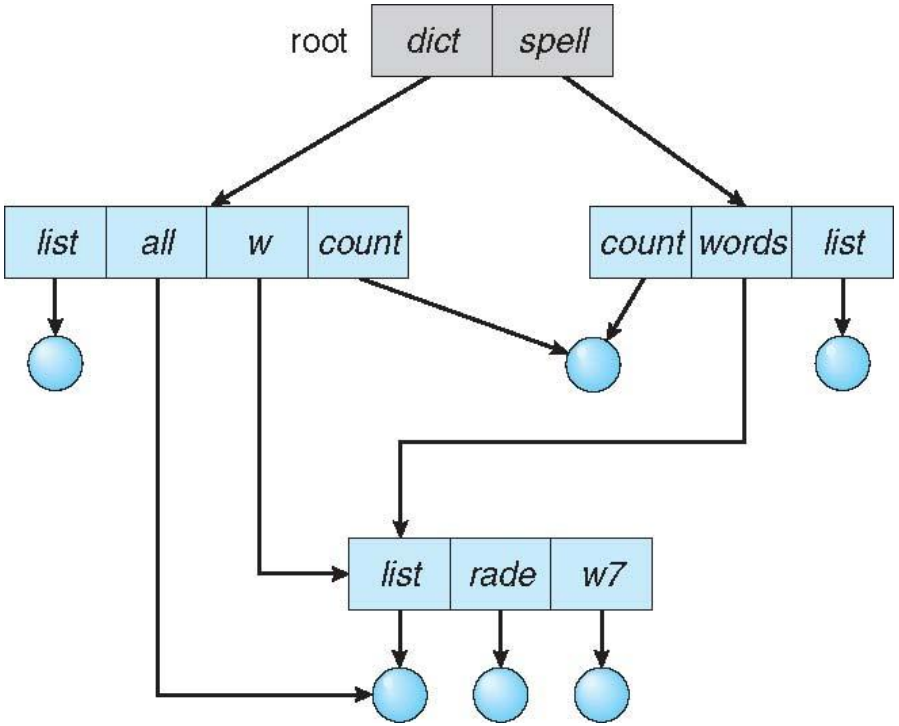
    mkdir <dir-name>
  - Results in:

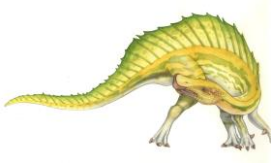




Acyclic-Graph Directories

- Have shared subdirectories and files
- Example



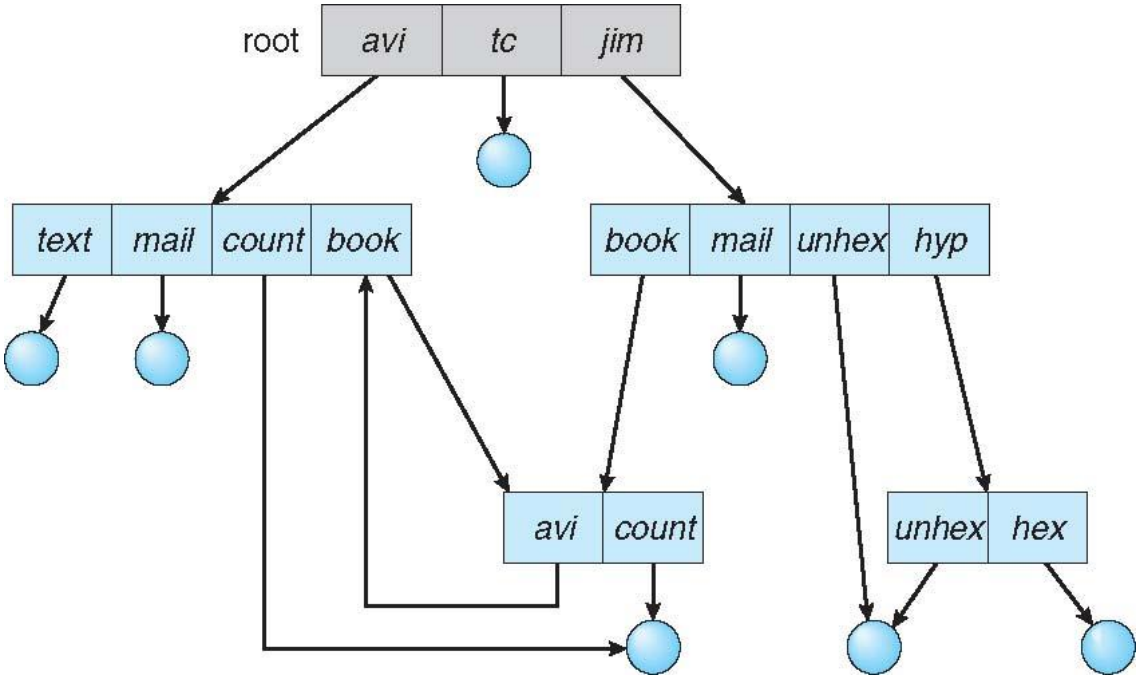


Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file
  - This is better than duplicating all information – major problem in maintaining consistency when a file is modified
  - Must make sure not to traverse shared structures more than once
- If ***dict*** deletes **w/*list*** ⇒ dangling pointer

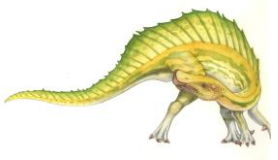
  Solutions:
  - Backpointers, so we can delete all pointers.
    - Variable size records a problem
  - Leave it as is until an attempt is made to use it (Unix's symbolic links)
  - Entry-hold-count solution (Unix's hard links also use reference count)

General Graph Directory



General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to files not subdirectories
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK
  - Or simply bypass links to directories during directory traversal.



Protection

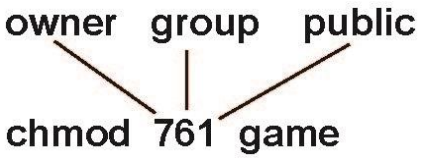
- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups in Unix

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a file (say *game*) or subdirectory, define an appropriate access.



- Attach a group to a file

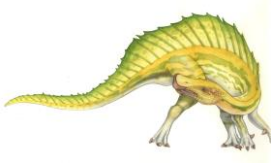
```
chgrp    G    game
```

For a directory, ‘r’ is for listing the directory, ‘x’ is for accessing any file/subdirectory under that directory

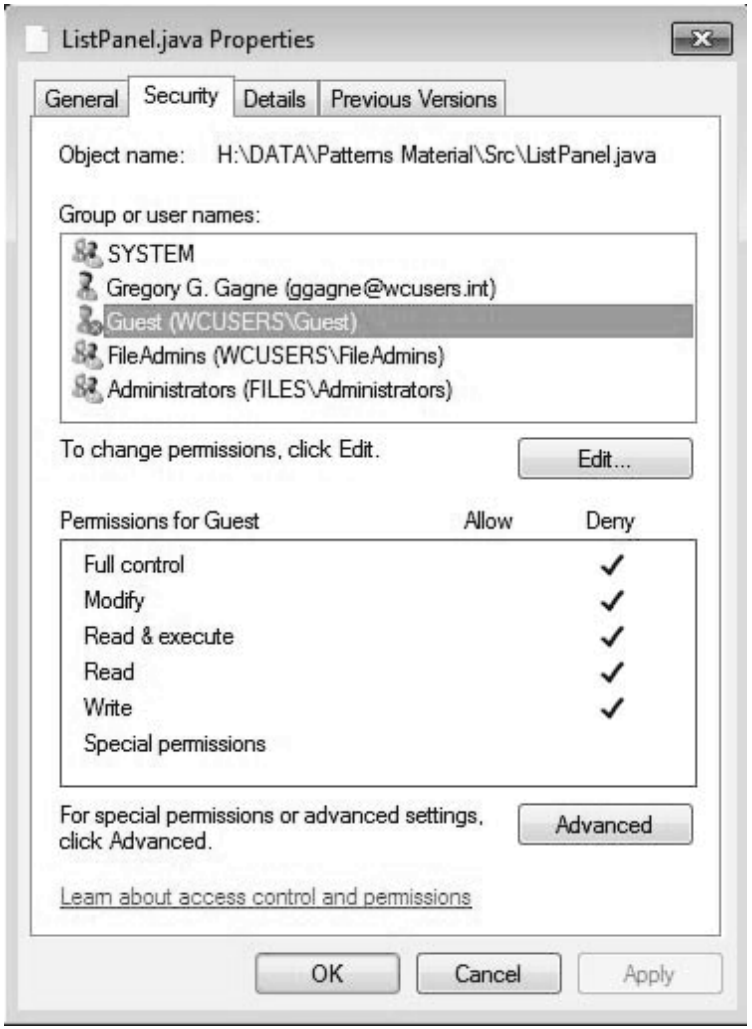
A Sample UNIX Directory Listing

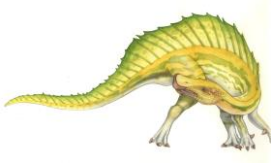
-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

^ # of hard links for file
  # of subdirectories for a directory



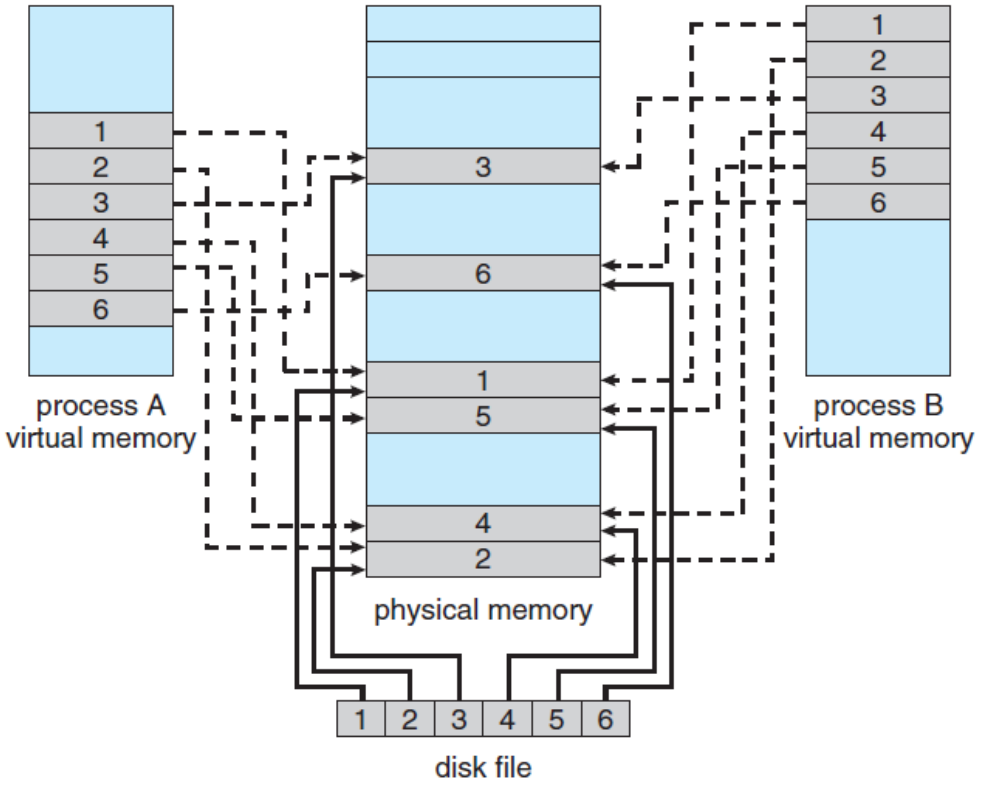
Windows 7 Access-Control List Management





Memory-Mapped Files

- Instead of accessing a file on disk via open/read/write, we can use the virtual memory techniques to treat file I/O as routine memory accesses
- A memory-mapped file allows a part of the virtual address space to be logically associated with the file, which leads to significant performance increases



End of Chapter 13