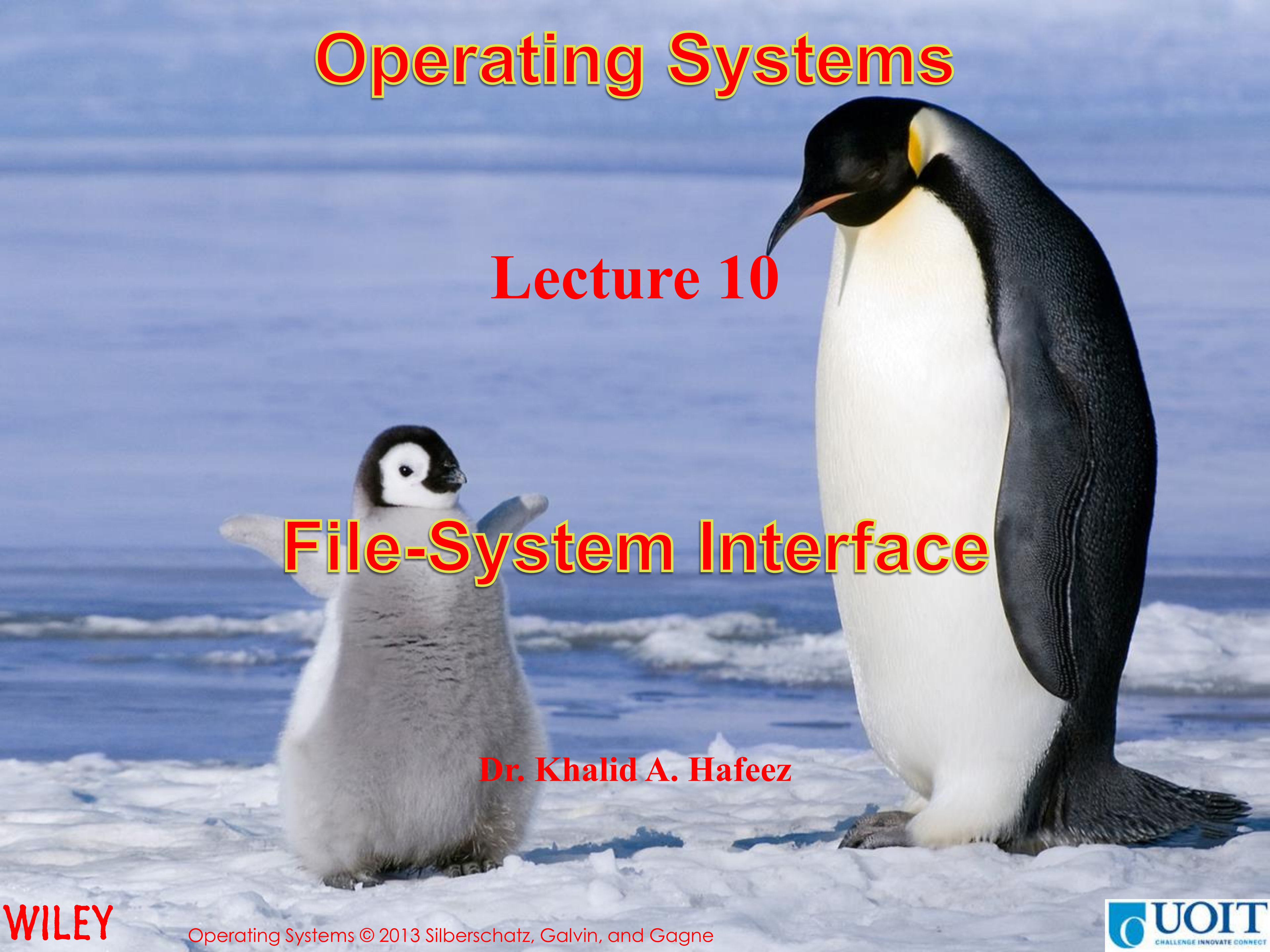


# Operating Systems

## Lecture 10

## File-System Interface

Dr. Khalid A. Hafeez

WILEY

UOIT

CHALLENGE INNOVATE CONNECT

# File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

# 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-System Interface

## File Concept

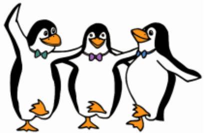
- The  file system is the most visible aspect of an operating system.
  - It provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system.
  - The file system consists of two distinct parts:
    - Collection of files, each storing related data,
    - Directory structure, which organizes and provides information about all the files in the system.

- The OS provides a uniform logical view of stored information.
  - The OS abstracts from the physical properties of its storage devices to define a logical storage unit, the file.
  - A file is a named collection of related information that is recorded on secondary storage.

# File Concept

## File Concept

- Contiguous logical address space
- Types of files:
  - Data
    - numeric
    - character
    - binary
  - Program
    - source,
    - object,
    - executable
- Content is defined by the file’s creator
  - a file is a sequence of bits, bytes, lines, or records



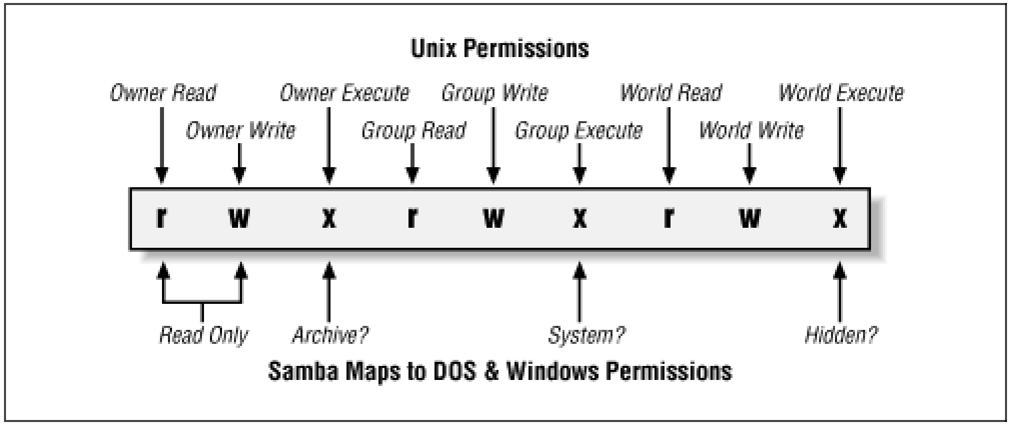
# File Concept

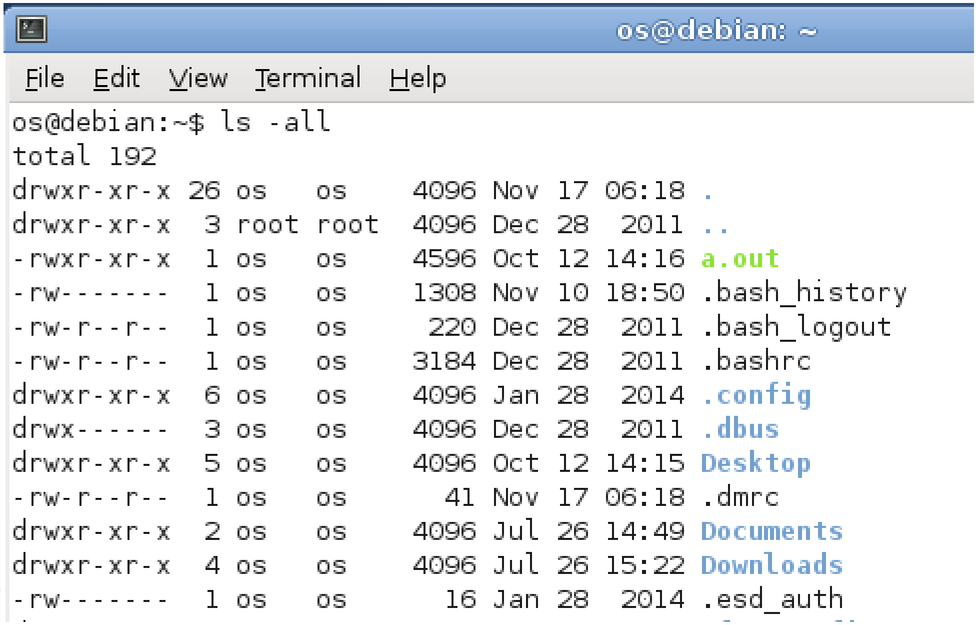
- File Attributes
  - A file’s attributes vary from one operating system to another but typically consist of these:
    - **Name**: the only information kept in human-readable form
    - **Identifier**: unique tag (number) identifies the file within the file system
    - **Type**: needed for systems that support different types of files
    - **Location**: pointer to the file location on the device
    - **Size**: current file size (in bytes, words, or blocks)
    - **Protection**: controls determine who can do reading, writing, executing
    - **Time, date, and user identification**: This information may be kept for creation, last modification, and last use. These data for protection, security, and usage monitoring
  - Information about files are kept in the directory structure, which is maintained on the disk
  - Many newer file systems support extended file attributes such as character encoding, and file checksum

# File Concept

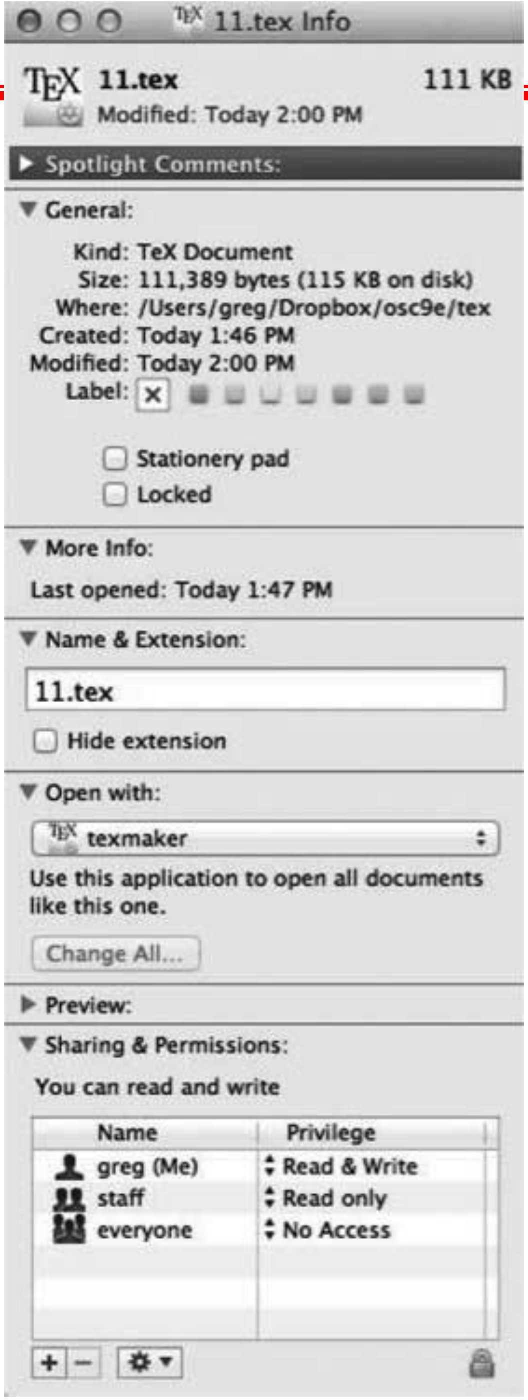
## File Attributes

- File info Window on Mac OS X





```
os@debian:~$ ls -all
total 192
drwxr-xr-x 26 os   os    4096 Nov 17 06:18 .
drwxr-xr-x  3 root root  4096 Dec 28  2011 ..
-rwxr-xr-x  1 os   os    4596 Oct 12 14:16 a.out
-rw-------  1 os   os    1308 Nov 10 18:50 .bash_history
-rw-r--r--  1 os   os     220 Dec 28  2011 .bash_logout
-rw-r--r--  1 os   os    3184 Dec 28  2011 .bashrc
drwxr-xr-x  6 os   os    4096 Jan 28  2014 .config
drwx------  3 os   os    4096 Dec 28  2011 .dbus
drwxr-xr-x  5 os   os    4096 Oct 12 14:15 Desktop
-rw-r--r--  1 os   os      41 Nov 17 06:18 .dmrc
drwxr-xr-x  2 os   os    4096 Jul 26 14:49 Documents
drwxr-xr-x  4 os   os    4096 Jul 26 15:22 Downloads
-rw-------  1 os   os      16 Jan 28  2014 .esd_auth
```



11.tex 111 KB
Modified: Today 2:00 PM

Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label: ☒

□ Stationery pad
□ Locked

Last opened: Today 1:47 PM

11.tex

□ Hide extension

texmaker

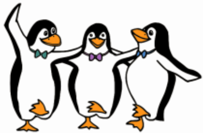
Use this application to open all documents like this one.

You can read and write

| Name | Privilege |
|---|---|
| greg (Me) | Read & Write |
| staff | Read only |
| everyone | No Access |

# File Concept

- File Operations
  - File is an **abstract data type**
  - The OS provide system calls to perform the **main six** operation on files:
    1. Create a file:
       - OS finds a space in the file system for the file.
       - OS makes an entry for the new file in the directory.
    2. Write:
       - Find the file in the directory to find its location on the disk and then start writing at write pointer location
    3. Read:
       - Find the file in the directory to find its location on the disk and then start writing at read pointer location
         - Both the read and write operations use this same pointer



# File Concept

## File Operations

4. Reposition within file: (known as file seek)
   - The current-file-position pointer is repositioned to a given value (no I/O operation)
5. Delete
   - OS searches the directory for the file, releases all file space, and erase the directory entry.
6. Truncate
   - OS will keep the file attributes but it will set its size to zero, and release all file space

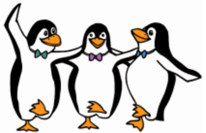
- ***Open($F_i$):*** search the directory structure on disk for entry $F_i$, and move the content of entry to memory

- ***Close ($F_i$):*** move the content of entry $F_i$ in memory to directory structure on disk

# File Concept

## File Operations

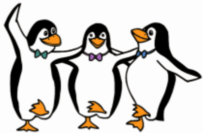
- To avoid searching the directory every time we need to access a file:
  - OS requires that an *open()* system call must be made before a file is first used.
    - This call will put the file in an open-file table that makes it easy to access the file
    - Each process that opens the same file, has this file added to its own open-file table.
    - The first time the file is opened by any process will be put in a system-wide table as well as that processes open-file table.
    - There is read-lock and an exclusive-lock (write) for ever opened file.



# File Concept

## Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
    - It includes file pointer: pointer to last read/write location, per process that has the file open
  - **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

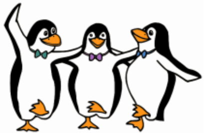


# File Concept

- Open 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
    - **Exclusive lock** similar to writer lock
  - Mediates access to a file
  - They can be Mandatory or advisory:
    - **Mandatory** – access is denied depending on locks held and requested
    - **Advisory** – processes can find status of locks and decide what to do

# File Concept

- Linux System
  - Some Standard Directories
    - / The root of the hierarchy
    - /bin Most essential Linux commands: ls, rm, …
    - /boot Linux kernel, all files needed to boot
    - /dev Those special device files
    - /etc/ System configurations, Contains no binary
    - /lib Library Object files for C/C++ and Fortran
    - /home Users' directories
    - /sbin Administration Tools, need special permission
    - /etc/passwd
    - /usr The largest part of the linux file system;
      - Contains general purpose programs

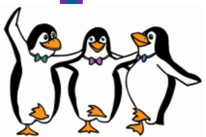


# File Concept

## Linux System

- Directory Operations

| | |
|---|---|
| mkdir dirname | creates a directory with name dirname |
| rmdir dirname | removes a directory dirname. |
| mv data1 newdata/ | moves the file data1 to the folder newdata and deletes the old one. |
| cp data1 newdata/ | will copy the file data1 to the directory newdata (assuming it has already been created) |
| ls | lists files |
| pwd | shows what directory (folder) you are in. |
| cd | changes directories |
| ls -all \|more | shows one screen of file names at a time. |
| rm data1 | deletes the file data1 in the current directory |



# File Concept

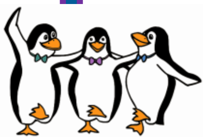
- Linux System
  - Processing Files

| | |
|---|---|
| Creating: | vi, emacs, pico, vim, … |
| Displaying: | less, more |
| Determining file type: | file “filename.ext” |
| File Size: | wc (wc test.c) |
| Compressing Files: | zip, gzip |

# File Concept

## Linux System

- Processing Files
  - find: search in the specified path to locate files that match the pattern
    - find . -name *bash –print
    - find /usr/include –name socket.h –print
  - whereis: locate the binary, source, and manual page files for a command
    - wheris –b cat
    - whereis ifconfig
  - grep: to search text or searches the given file for lines containing a match to the given strings or words
    - grep “printf” /usr/include/*
    - grep –n include *.c
    - grep ‘^[A-H]’ test.c



# File Concept

- ## File Locking Example – Java API

  This program acquires two locks on the file file.txt. The first half of the file is acquire as an exclusive lock; the lock for the second half is a shared lock.

```
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();
```

# File Concept

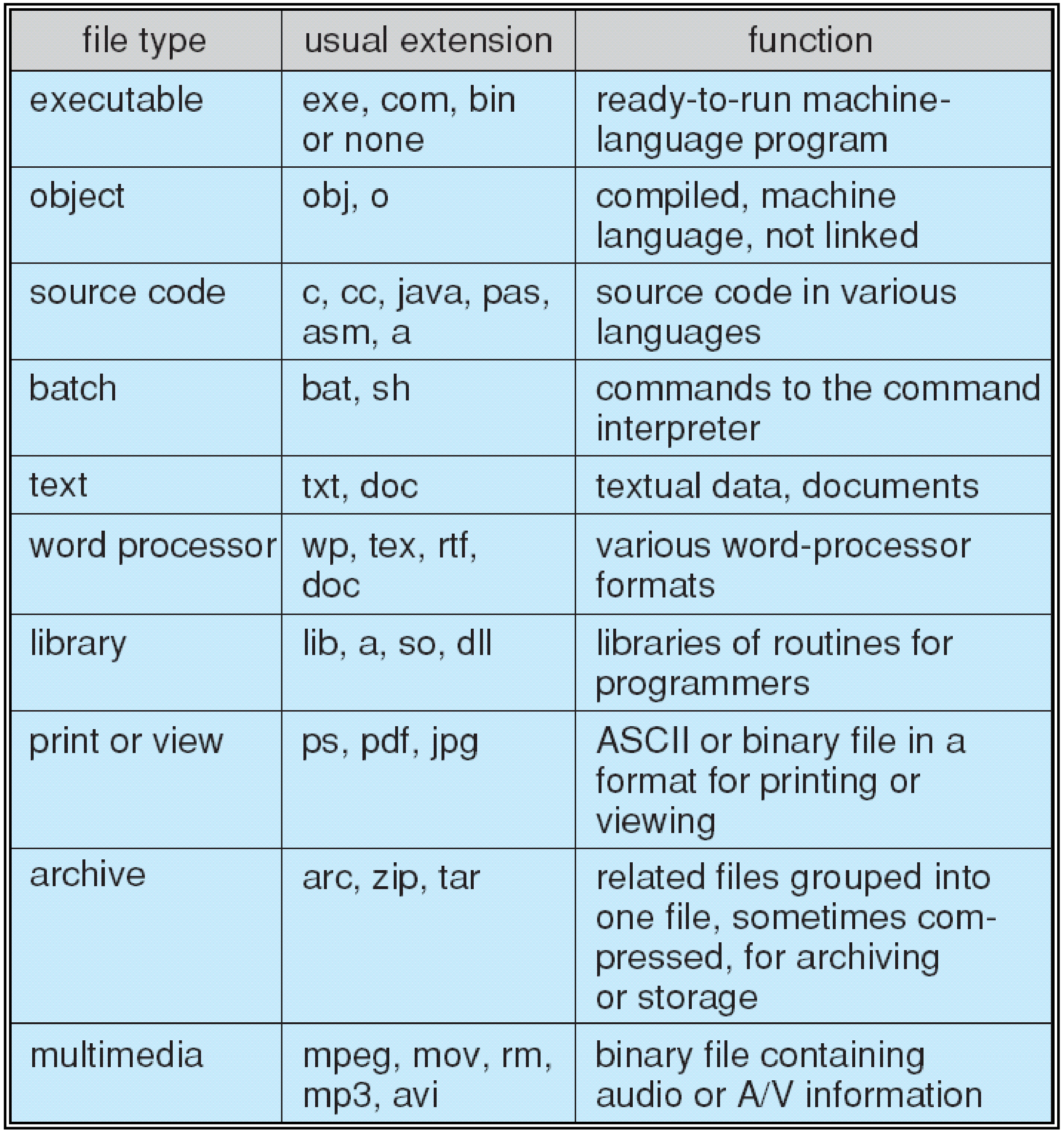
- File Locking Example – Java API

```
                        // 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 Concept

- File Types – Name, Extension

Common file types.



| 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, pas, asm, a | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| text | txt, doc | textual data, documents |
| word processor | wp, tex, rtf, doc | various word-processor formats |
| library | lib, a, so, dll | libraries of routines for programmers |
| print or view | ps, pdf, jpg | ASCII or binary file in a format for printing or viewing |
| archive | arc, zip, tar | related files grouped into one file, sometimes compressed, for archiving or storage |
| multimedia | mpeg, mov, rm, mp3, avi | binary file containing audio or A/V information |

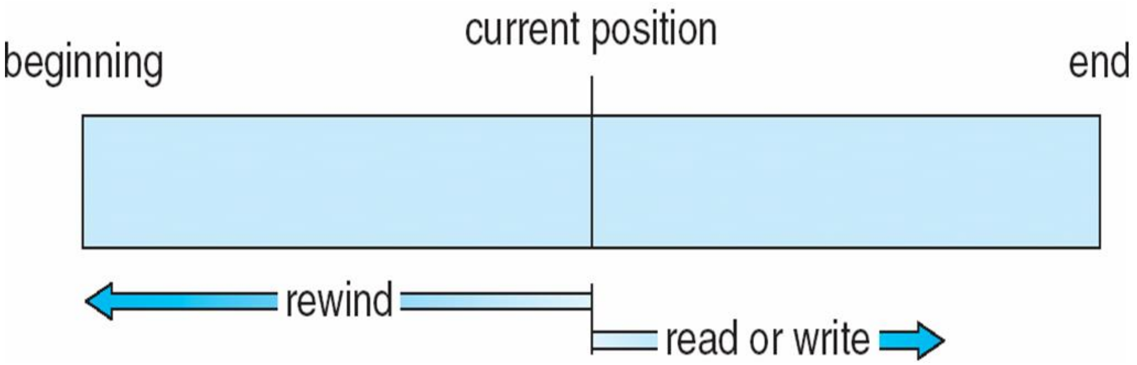
# File Concept

## File Structure

- File types also can be used to indicate the internal structure of the file.
  - None - sequence of words, bytes
  - Simple record structure
    - Lines
    - Fixed length
    - Variable length
  - Complex Structures
    - Formatted document
    - Relocatable load file
  - Can simulate last two with first method by inserting appropriate control characters
  - Who decides:
    - Operating system
    - Program

# Access Methods

- The information in the file can be accessed in several ways:
  - Sequential Access
    - Information in the file is processed in order, one record after the other.
    - It is based on a tape-drive model
      - **read next**
      - **write next**
      - **Reset**
      - no read after last write  (rewrite)



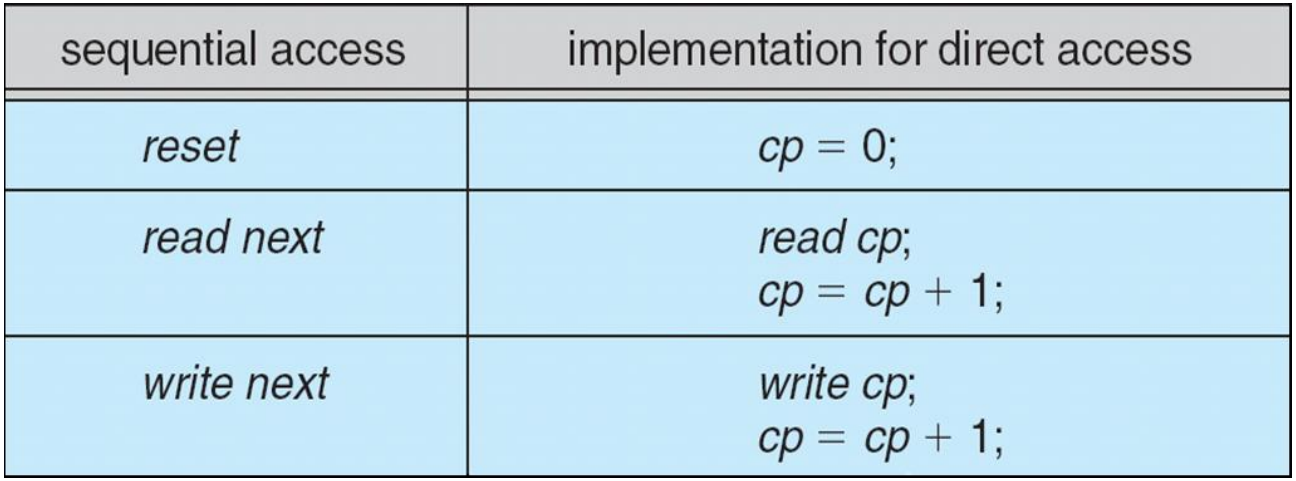
# Access Methods

- The information in the file can be accessed in several ways:
  - Direct Access (or relative access)
    - File is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order.
    - It is based on the disk model of a file
      - **read *n***
      - **write *n***
      - **position to *n***
        - **Read-next**
        - **Write-next**
      - **rewrite *n***
      - *n* = relative block number from the start of the file
  - Relative block numbers allow OS to decide where file should be placed

# Access Methods

The information in the file can be accessed in several ways:

- Direct Access (or relative access)
  - Simulation of Sequential Access on Direct-access File

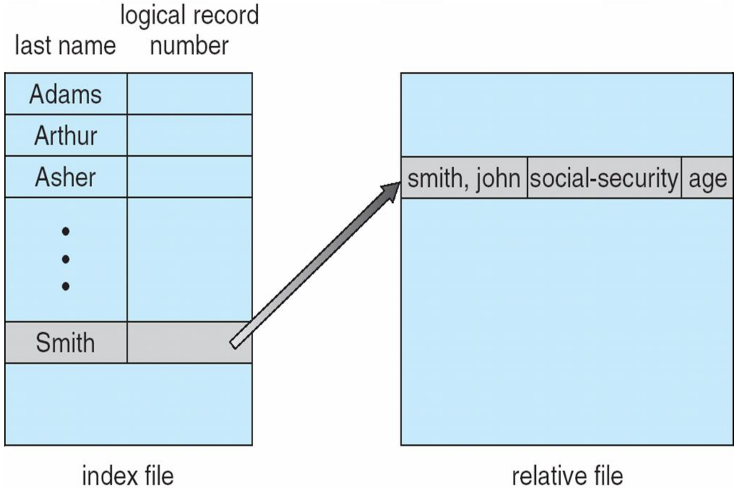


| sequential access | implementation for direct access |
|---|---|
| *reset* | *cp* = 0; |
| *read next* | *read cp*;<br>*cp* = *cp* + 1; |
| *write next* | *write cp*;<br>*cp* = *cp* + 1; |

# 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 (consider UPC (universal product code) code plus record of data about that item)
- If the index file is too large, then create an index for the index
  - One index (in memory) of the second index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key



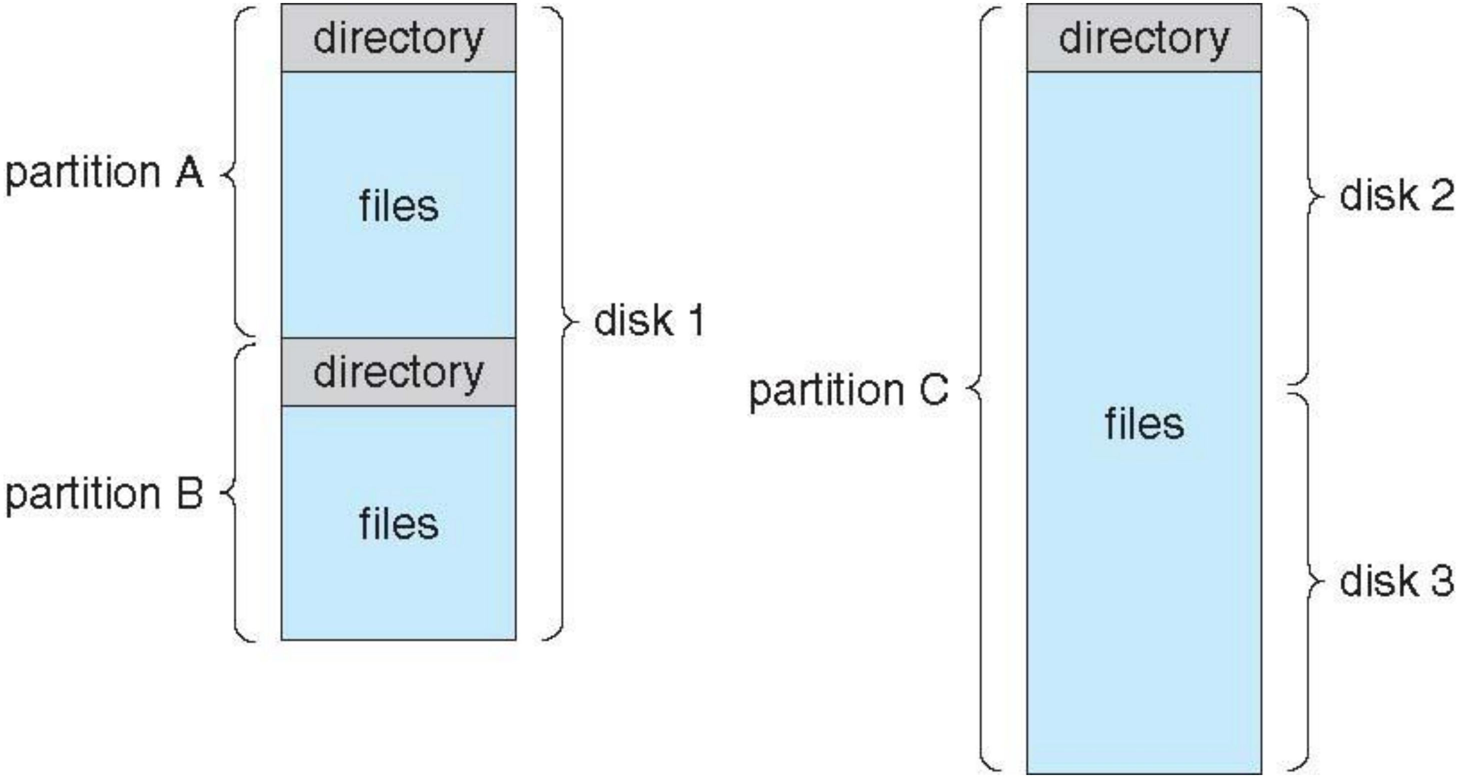
# Directory Structure

## 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
- An entity containing file system is known as a **volume**
  - The volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set.
- Each volume containing file system also tracks that file system′s info in **device directory** or **volume table of contents** (or just **directory**)
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

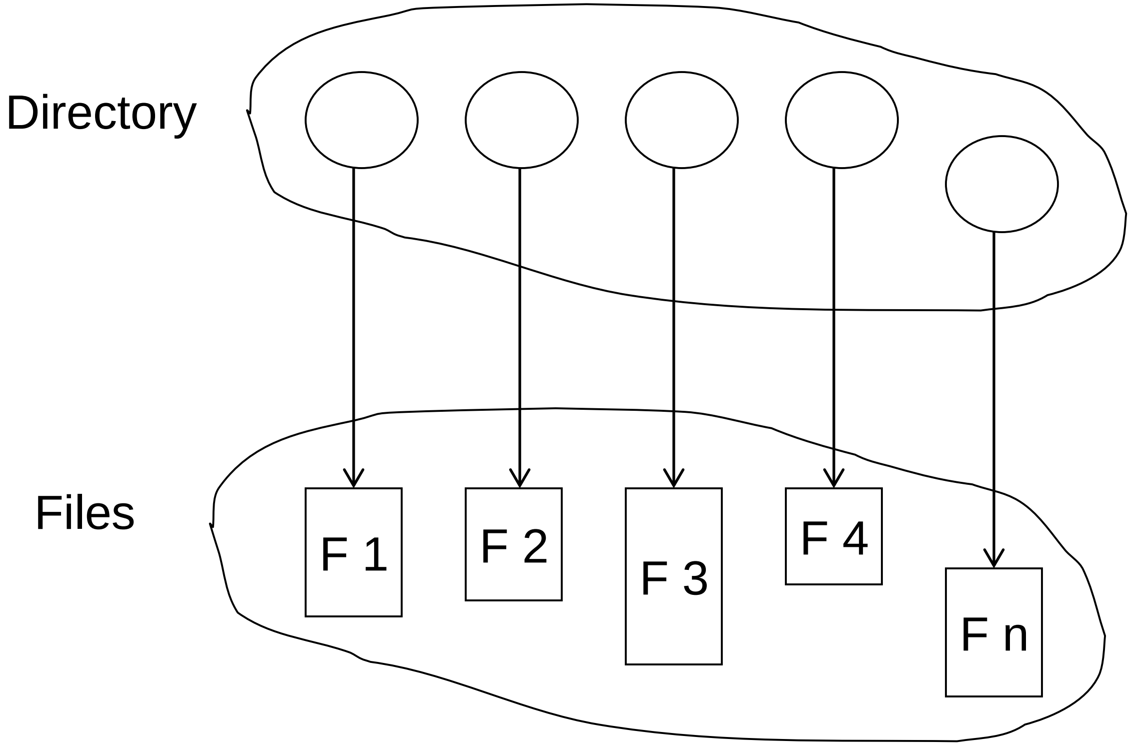
# Directory Structure

- A Typical File-system Organization



# Directory Structure

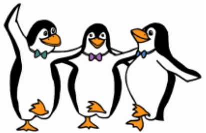
- A **directory** is a collection of nodes containing information about all files
  - The directory has the name, location, size, and type for all files on that volume.



Both the directory structure and the files reside on disk

# Directory Structure

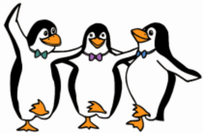
- Types of File Systems
  - We consider only general-purpose file systems
  - But systems frequently have may file systems, some general- and some special- purpose
  - Consider Solaris has
    - tmpfs – temporary memory-based volatile FS for fast, temporary I/O
    - objfs – interface into kernel memory to get kernel symbols for debugging
    - ctfs – contract file system for managing daemons
    - lofs – loopback file system allows one FS to be accessed in place of another
    - procfs – kernel interface to process structures
    - ufs, zfs – general purpose file systems



# Directory Structure

## Operations Performed on Directory

- Search for a file ls test*.txt
- Create a file
  - touch testFile.txt or >> testFile.txt or > testFile.txt
- Delete a file
  - rm testFile.txt or rm SOFE3950/ *.txt
- List a directory ls -al
- Rename a file or move a file
  - mv test.txt newtest.txt or mv test.txt ~/myDir/
- Traverse the file system
  - To combine multiple files and/or directories into a single file
    - tar -cvf file.tar inputfile1 inputfile2
    - tar -xvf file.tar
  - To create compressed archives
    - tar -cvzf file.tar.gz inputfile1 inputfile2
    - tar -xvzf file.tar.gz



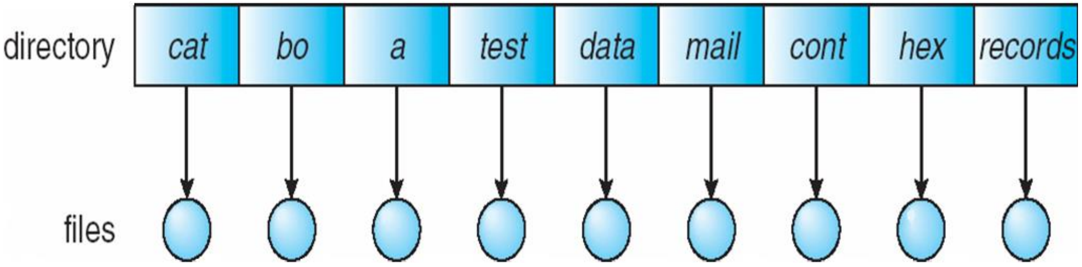
# Directory Structure

- 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, …)

# Directory Structure

## Single-Level Directory

- A single directory for all users

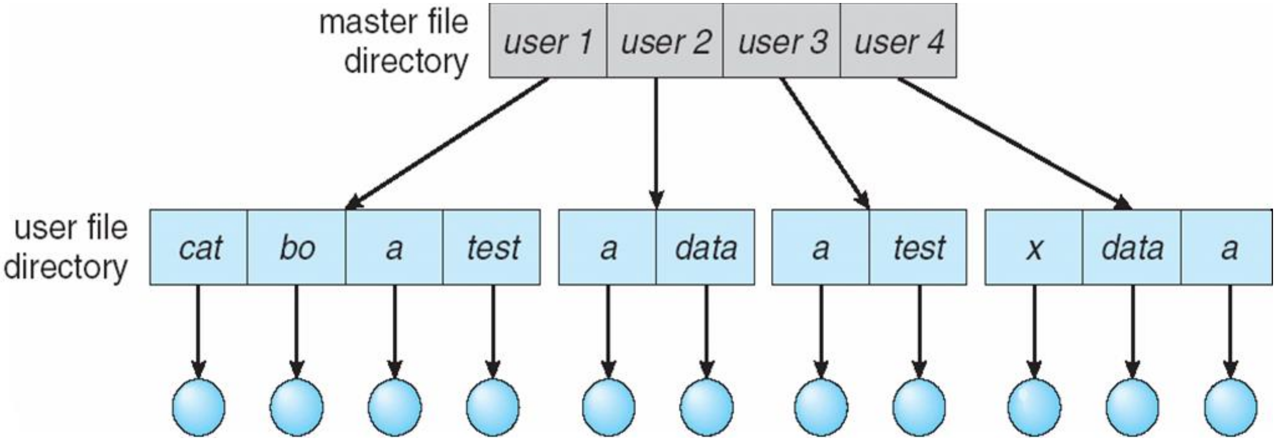


- Naming problem
  - Since all files are in the same directory, they must have unique names.
- Grouping problem

# Directory Structure

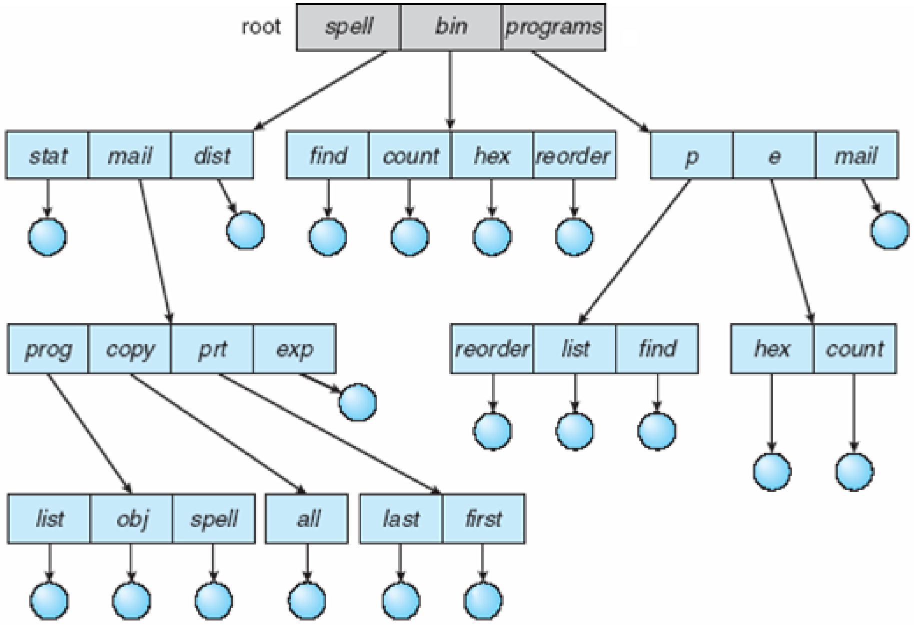
## Two-Level Directory

- To create a separate directory for each user
  - Each user has his/her own user file directory (UFD).
  - When a user logs in, the system's master file directory (MFD) is searched.
    - The MFD is indexed by user name or account number
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability



# Directory Structure

- Tree-Structured Directories
  - It allows users to create their own subdirectories and to organize their files accordingly.
  - The tree has a root directory, and every file has a unique path name.
  - A directory (or subdirectory) contains a set of files or subdirectories
  - A directory is simply another file, but it is treated in a special way.
  - All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).



# Directory Structure

## Tree-Structured Directories

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd /spell/mail/prog**
  - **type list**

# Directory Structure

## Tree-Structured Directories

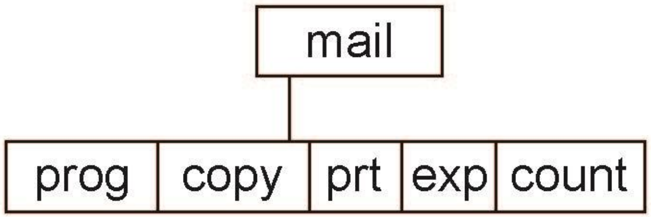
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

  **rm <file-name>**
- Creating a new subdirectory is done in current directory

  **mkdir <dir-name>**

Example: if in current directory **/mail**

**mkdir count**



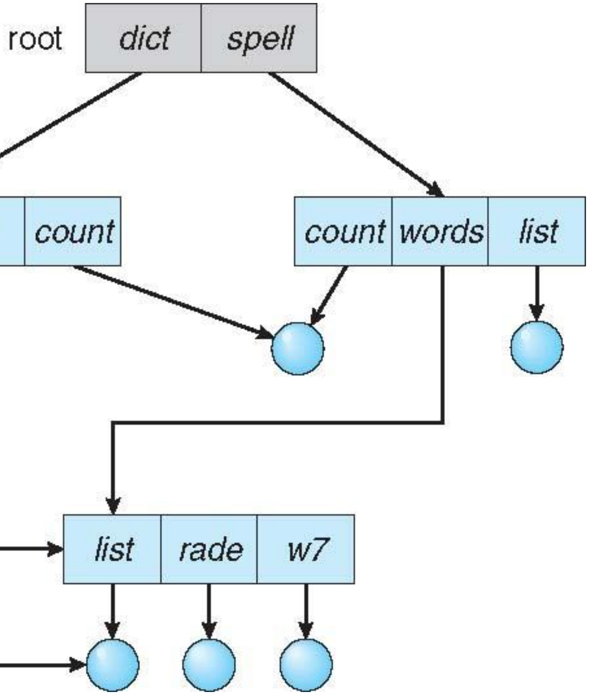
Deleting “mail” → deleting the entire subtree rooted by “mail”

# Directory Structure

## Acyclic-Graph Directories

- Have shared subdirectories and files
- Two different names (aliasing)
- **Problem**: If **dict** deletes **count** ⇒ there will be a dangling pointer

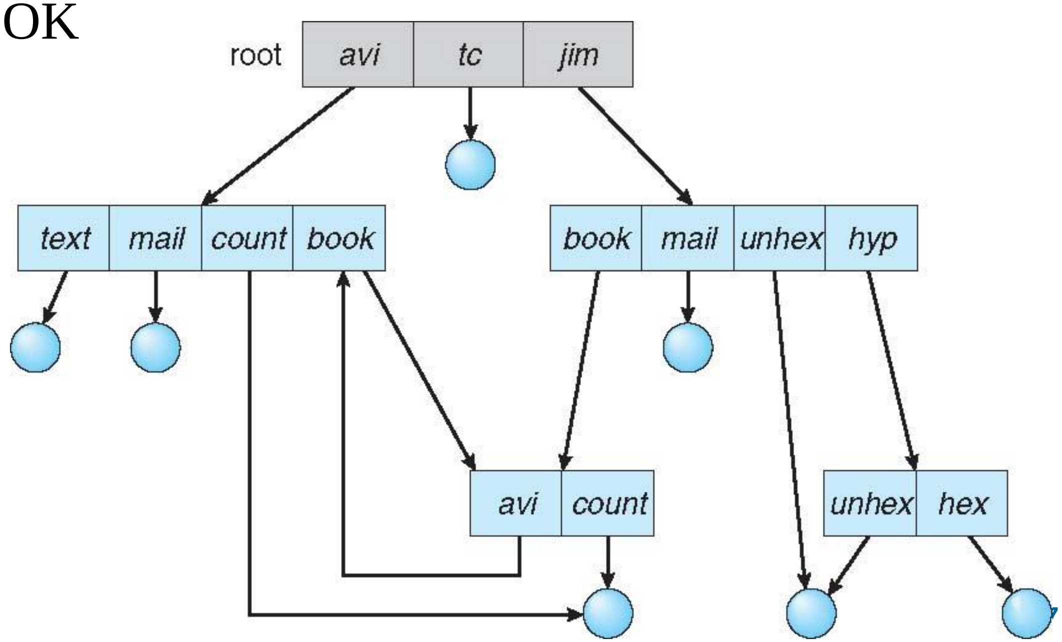
  Solutions:
  - Backpointers, so we can delete all pointers
  - Entry-hold-count solution: holds how many pointers to this file



# Directory Structure

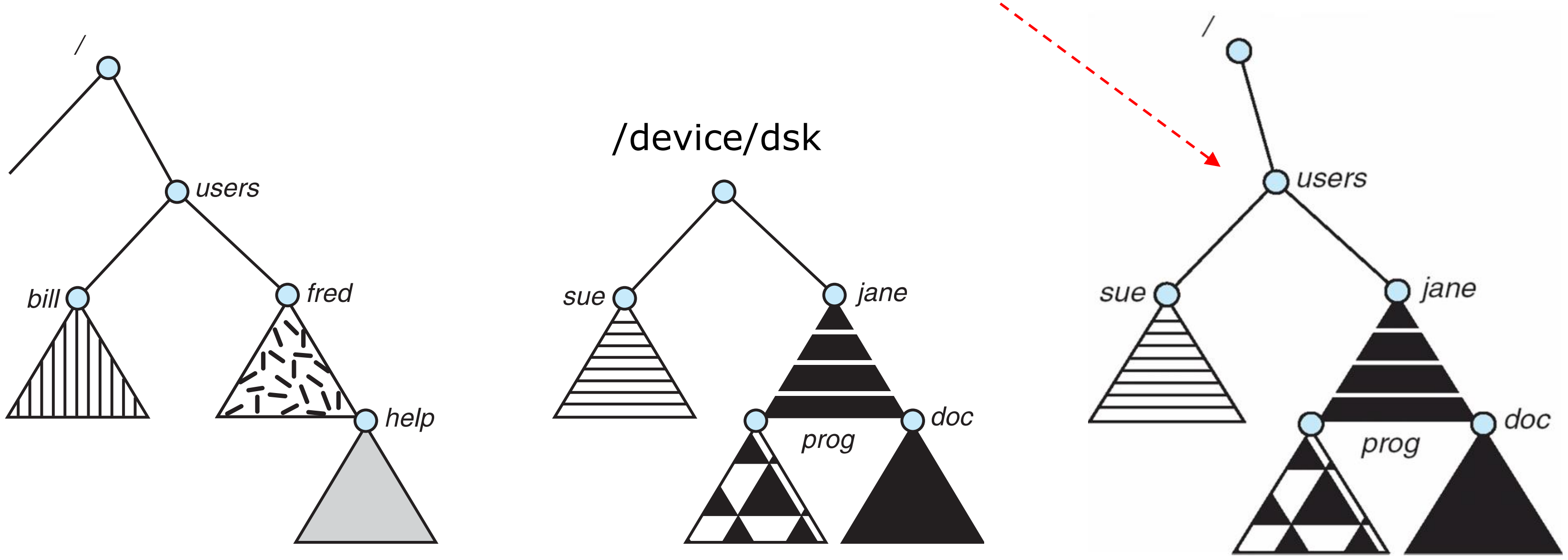
## General Graph Directory

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
    - First pass: Traverse the whole directory structure and mark every directory or file that has been visited
    - Second pass: Delete all files / directories that are not marked.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK



# File System Mounting

- A file system must be **mounted** before it can be accessed by processes
- A unmounted file system (Fig.b) is mounted at a **mount point**
  - unmounted volume residing on /device/dsk
    - In Fig.c, mount the volume residing on /device/dsk over /users
      - mount /device/dsk /users
      - unmount /device/dsk



(a)

an existing file system

(b)

unmounted volume residing on /device/dsk

(c)

# File System Mounting

- All operating systems first read the directory structure of the disk into the memory to check if the disk has correct directory structure or not; then OS **mounts** that disk to be accessible.
  - Unix:
    - All unmounted partitions (file systems) are mounted into a directory tree rooted by “/”; for example, a hard disk can contain the home directory of all users of the computer which can be mounted in an empty directory /users/.
    - Needs explicit command “mount” for mounting"
  - Macintosh:
    - Mac OS X operating system searches for a file system on the device. If it finds one, it automatically mounts the file system under the **/Volumes** directory, adding a folder icon labeled with the name of the file system
  - MS Windows:
    - Has extended two-level directory structure, where, devices and partitions are assigned a drive letter: A:\, C:\ , D:\

# File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If a multi-user system, we need:
  - **User IDs** to identify users, allowing permissions and protections to be per-user
    **Group IDs** to allow users to be in groups, permitting group access rights

# File Sharing

## Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems (DFS)**
  - Semi automatically via the **world wide web (WWW)**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

# File Sharing

- Consistency Semantics
  - Specify how multiple users are to access a shared file simultaneously
    - Similar to process synchronization algorithms
      - Tend to be less complex due to disk I/O and network latency (for remote file systems
    - Andrew File System (**OpenAFS**) implemented complex remote file sharing semantics
      - Writes only visible to sessions starting after the file is closed
    - Unix file system (UFS) implements:
      - Writes to an open file by a user are visible immediately to other users who have this file open.
      - Sharing file pointer to allow multiple users to read and write concurrently

# Protection

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

# Protection

## Access Lists and Groups

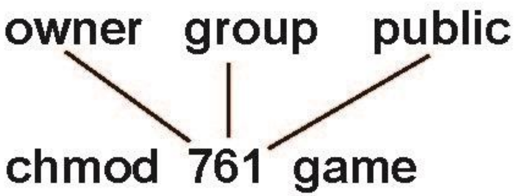
- Mode of access:  read, write, execute
- Three classes of users on Unix / Linux

a) **owner access** 7 $\Rightarrow$ rwx = 1 1 1

b) **group access** 6 $\Rightarrow$ rwx = 1 1 0

c) **public access** 1 $\Rightarrow$ rwx = 0 0 1

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

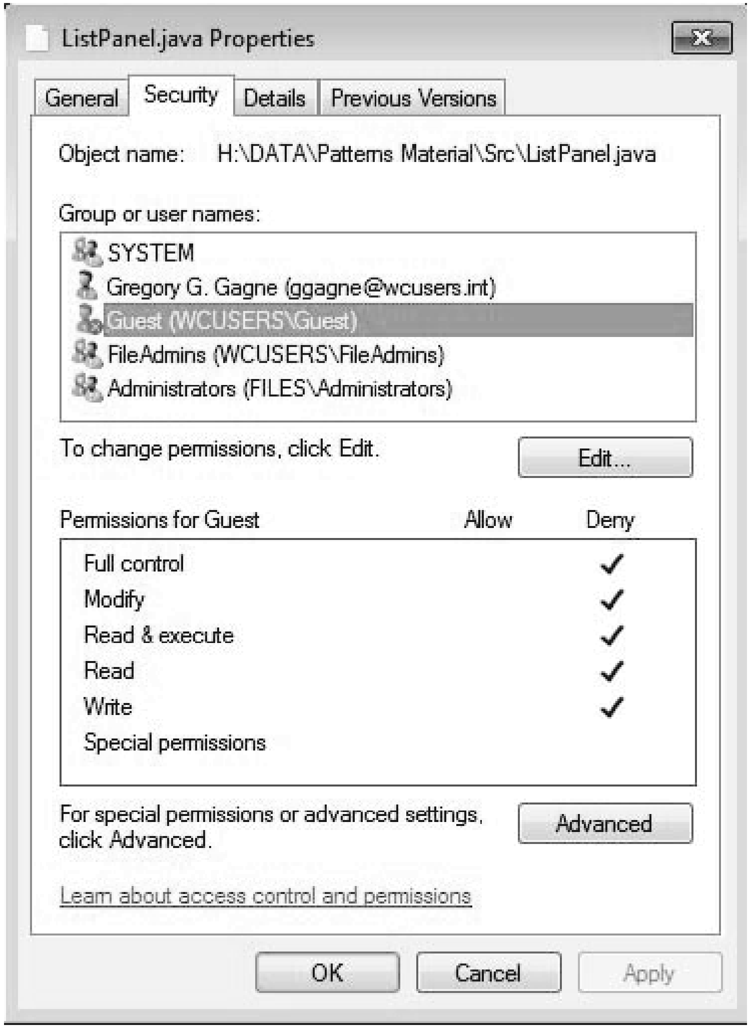


- Attach a group to a file

chgrp G game

# Protection

- Windows 7 Access-Control List Management



Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:

- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

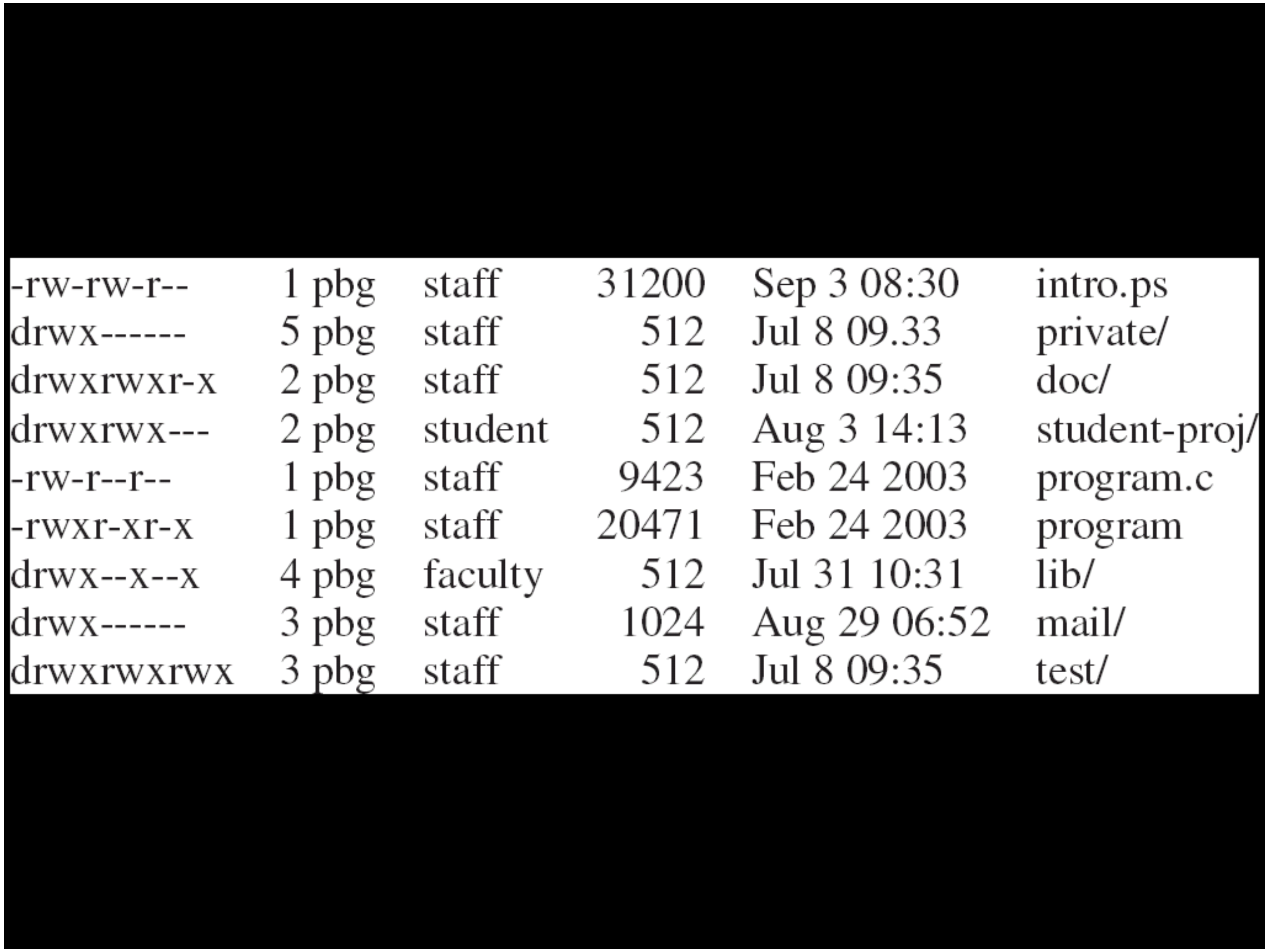
To change permissions, click Edit.

| Permissions for Guest | Allow | Deny |
|---|---|---|
| Full control | | ✓ |
| Modify | | ✓ |
| Read & execute | | ✓ |
| Read | | ✓ |
| Write | | ✓ |
| Special permissions | | |

For special permissions or advanced settings, click Advanced.

# Protection

- 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/
```