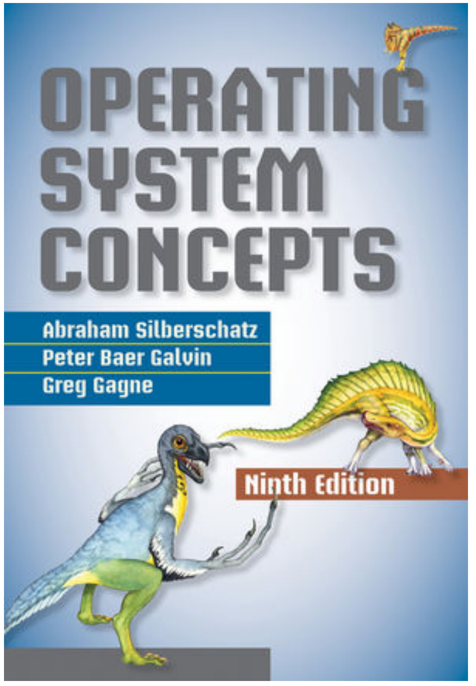
8 – File System Interface

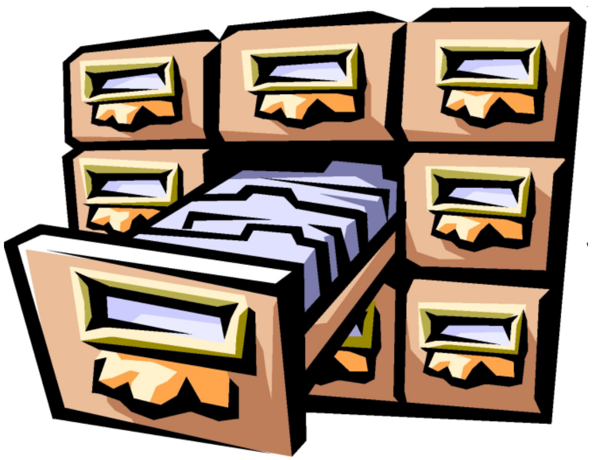
EECE 315 (101)
ECE – UBC
2013 W2



Acknowledgement: This set of slides is partly based on the PPTs provided by the Wiley's companion website (including textbook images, when not explicitly mentioned/referenced).

Overview

- **File Concept**
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection



Overview

- File system is one the most visible aspects of an OS
  - it provides the mechanism for on-line storage of and access to both data and programms

- the *file system* consists of two distinct parts:
  - a collection of *files*
    - each storing related data
  - and a *directory structure*
    - which organizes and provides information about all the files in the system

- File systems live on device (e.g. hard disk)

Concept of File

- The concept of **file** is extremely general:
  - 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
  - From a user's perspective, a ***file*** is the smallest allotment of logical secondary storage
  - A ***file*** represents programs and data:
    - Everything must be within a file to be written to the secondary storage
    - A *data file* may be numeric, alphabetic, alphanumeric, or binary
    - A *file* is a sequence of bits, bytes, lines or records, the meaning of which is defined by the file's creator and user
    - A *file* may be free form or may have a certain defined structure, which depends on its type

File Attributes

- Directory listing example: the following is the output result using the “ls –l” command in Unix/Linux (in Windows/Linux, a similar command is “dir”):

-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/

access permissions
directories
owner
group
size
date/time
file/directory name

File Attributes (cont)

- A file's attributes vary from one OS to another but typically consist of:
  - **Name**
    - a file is named, for the convenience of its human user, e.g. *myfile.c*
    - in some OS, a name is case-sensitive.
    - the name attribute is the only information kept in human-readable form
  - **Identifier** – a unique tag (number) that identifies file within the file system
  - **Type** – an info needed for systems that support different types
  - **Location** – a pointer to a device and the file location on that device
  - **Size** – the current file size (in bytes, words, or blocks)
  - **Protection** – controls who can do reading, writing, executing
  - **Time, date, and user identification** – data for protection, security, and usage monitoring

- Information about all files is kept in the directory structure, which is maintained on the disk

File Operations

- A File is an **abstract data type.** To define a file properly, we need to consider the operations that can be performed on files.
  - **Creating a file:** Two steps are necessary to create a file
    - space must be found in the file system
    - an entry must be created in the directory
  - **Writing a file:** To write to a file, we use a system call that specifies the name of the file and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place
  - **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put (in memory)
    - because a process is either reading from or writing to a file, the current operation location can be kept as a per-process current-file-position pointer

File Operations (cont)

- Cont:
  - **Repositioning within file** (file *seek*)**:** repositioning the current-file-position pointer
  - **Deleting a file:** To delete the file, we search the directory for the named file. Having found it, we release all file space, and erase the directory entry
  - **Truncating a file:** The user may want to erase the contents of a file but keep its attributes. This function allows all attributes remain unchanged except for file length
- Other operations are also possible: appending, renaming, …
- Most of the file operations mentioned involve searching the directory for the entry associated with the name file
  - to avoid this constant searching, many systems require that an **`open()`** system call be made before a file is first used actively
  - the OS keeps a small table, called the open-file table, containing information about all open files

File Operations (cont)

- When a file operation is requested, the file is specified via an index into the open-file table, so no searching is required
  - when the file is no longer being actively used, it is closed by the process and the OS removes its entry form the table
- System calls:
  - *Open()* – search the directory structure on disk to find the entry, and move the content of entry to memory
  - *Close ()* – move the content of the entry in memory to directory structure on disk
- Some systems though implicitly open a file when the first reference to it is made
  - The file is automatically closed when the job or program that opened the file terminates

Open Files

- Several pieces of data are needed to manage open files:
  - **File pointer**
    - on systems that do not include a file offset as part of the `read()` and `write()` operation, the system must track the last read/write location as a current-file-position pointer
    - this pointer is unique to each process operating on the file
  - **File-open count**: is the counter of the number of times a file is open.
    - because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry
  - **Disk location of the file**: cache of data access information. This info is needed to locate the file on disk and is kept in memory.
  - **Access rights**: each process opens a file in an access mode. This info is stored on the per-process table to allow/deny subsequent I/O

Open File Locking

- Some OS provide facilities for locking an open file (or section of a file)
- File locks allow one process to lock a file and prevent other processes from gaining access to it
  - files locks are useful for files that are shared by several processes
    - a shared lock: several processes can acquire the lock concurrently
    - an exclusive lock: only one process at a time can acquire the lock
- OS may provide either mandatory or advisory file locking mechanism:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

- If an OS recognizes the **type of a file**, it can then operate on the file in reasonable ways
- A common technique for implementing file types is to include the type as part of the file name:
  - The name is split into two parts: a name and an extension, separated by a period
- UNIX uses a crude magic number stored at the beginning of some files to indicate roughly the type of the file

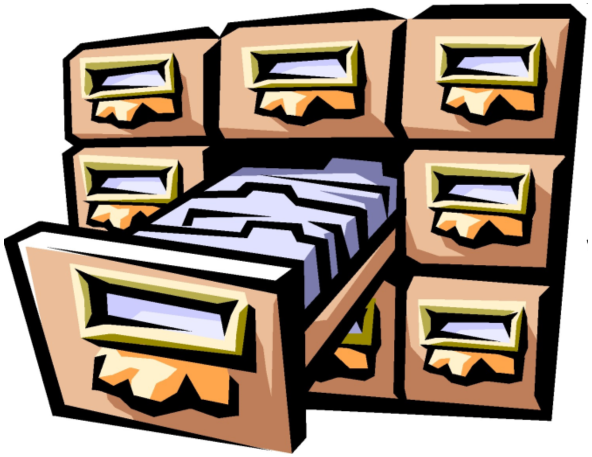
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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 Structure

- File types also can be used to indicate the internal structure of the file
  - source and object files have structures that match the expectations of the programs that read them
  - certain files must conform to a required structure that is understood by the OS
    - e.g. an executable file have a specific structure
- Most OSs (UNIX, Mac, MS-DOS, …) impose (and support) a minimal number of file structures
  - this is to reduce the size of the OS and to improve its support for different file structures
  - all OS though must support at least one structure – that is an executable file
  - e.g. UNIX considers each file to be a sequence of 8-bit bytes; no interpretation of these bits is made by the OS
    - this scheme provides maximum flexibility but little support

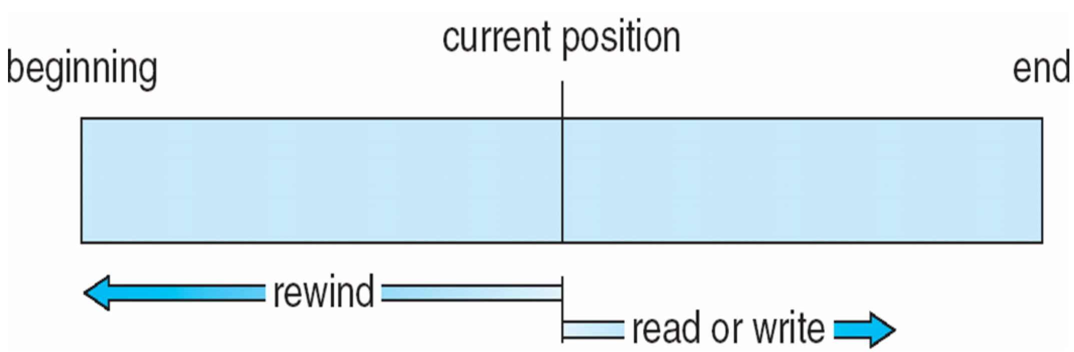
Overview

- File Concept
- **Access Methods**
- Directory Structure
- File-System Mounting
- File Sharing
- Protection



Access Methods

- Files store information. When it is used, this information must be accessed and read into computer memory
- The info in the file can be accessed in several ways
  - **Sequential Access**:
    - it is the simplest method
    - in this mode, information in the file is processed in order, one record after the other
    - it is the most common method, e.g. in editors or compilers
    - *read next*: reads the next portion of file and advances a file pointer
    - *write next*: appends to the end of the file and advances to the end of the newly written material (new end of file)



Access Methods (cont)

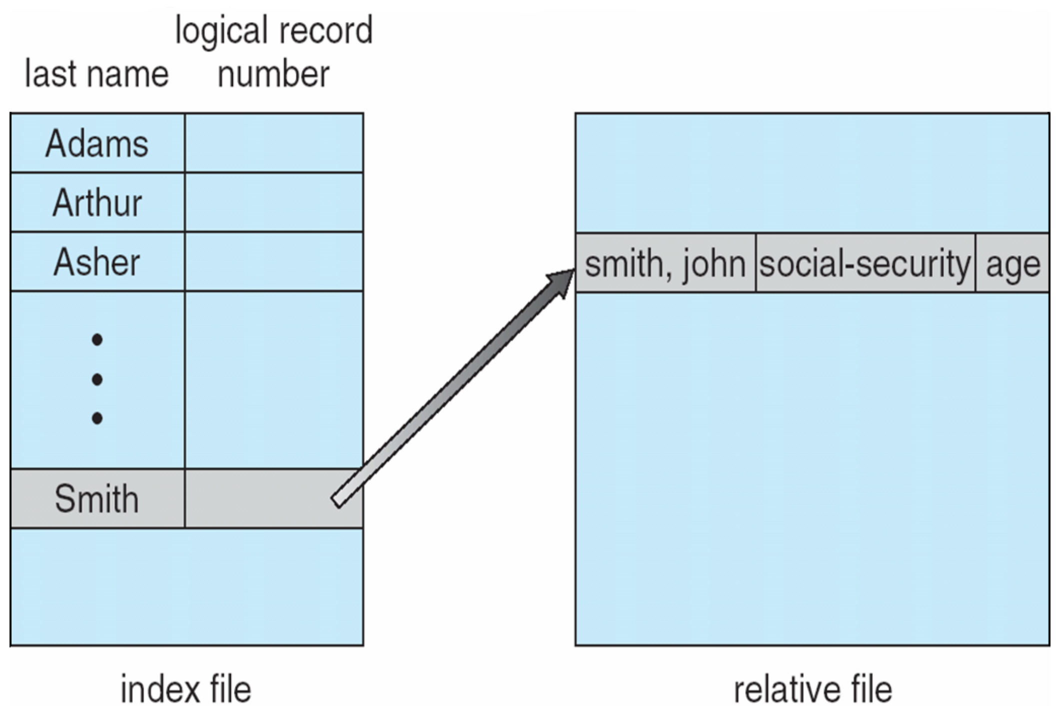
- Cont
  - **Direct Access** (relative access):
    - a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order
    - this model is based on the disk model of a file, since disks allow random access to any file block
    - the block number provided by the user to the OS for the access is a relative block number (i.e. an index relative to the beginning of the file)

Fig: simulation of sequential access on a direct-access file

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

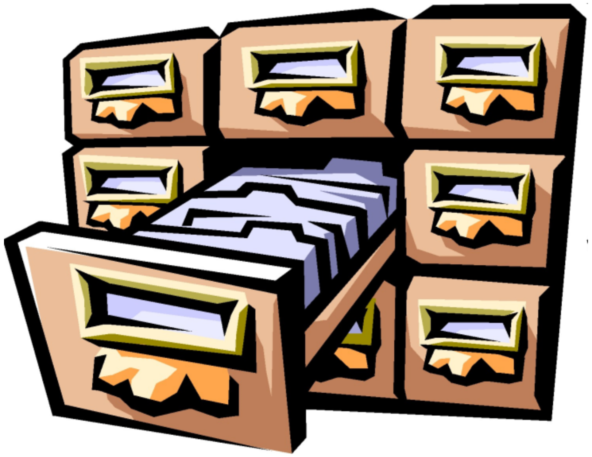
Other Access Methods

- Other access methods can be build on top of a direct-access method
  - these method generally <u>involve the construction of an index</u> for the file
  - the **index** contains pointers to various blocks
  - to find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record
- Example of Index and Relative Files



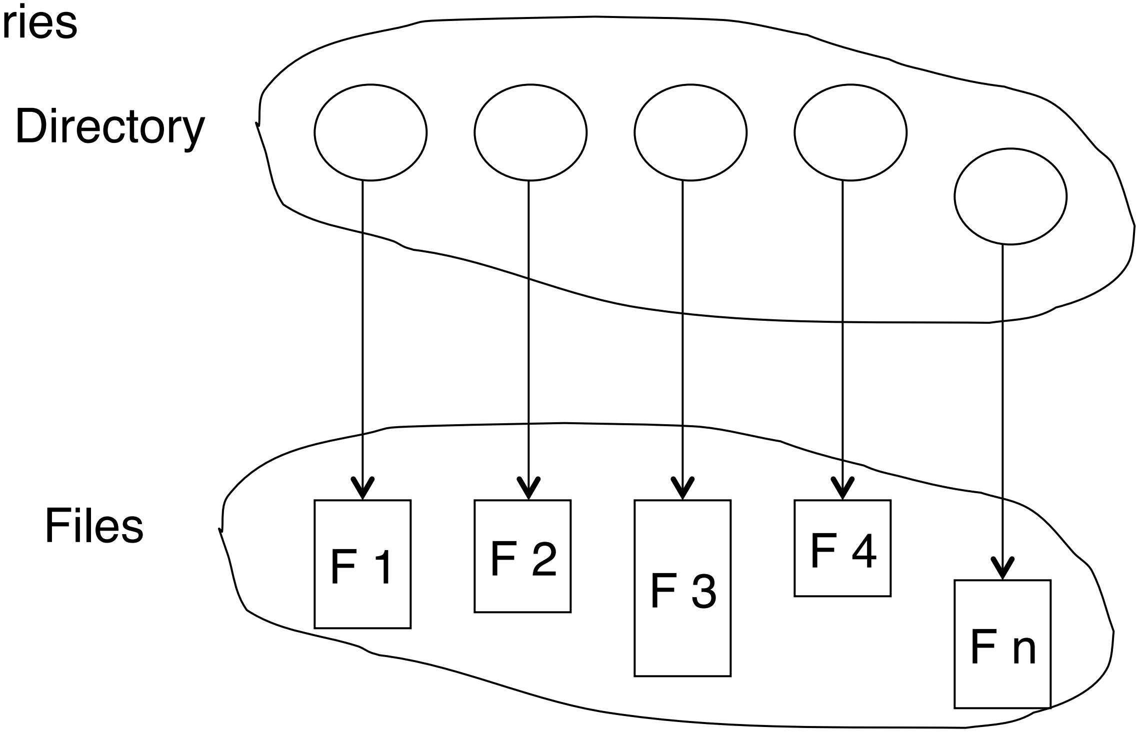
Overview

- File Concept
- Access Methods
- **Directory Structure**
- File-System Mounting
- File Sharing
- Protection



Disk and Directory Structure

- Each entity containing a file system is generally known as a **volume**
  - each volume that contains a file system must also contain information about the files in the system
  - this information is kept in entries in a device directory (directory for short) or volume table of contents
- A **directory** can be viewed as a symbol table that translates file names into their directory entries



Operations Performed on Directory

- The directory itself can be organized in many ways
  - we want to be able to insert entries, to delete entries, to reach for a named entry, …

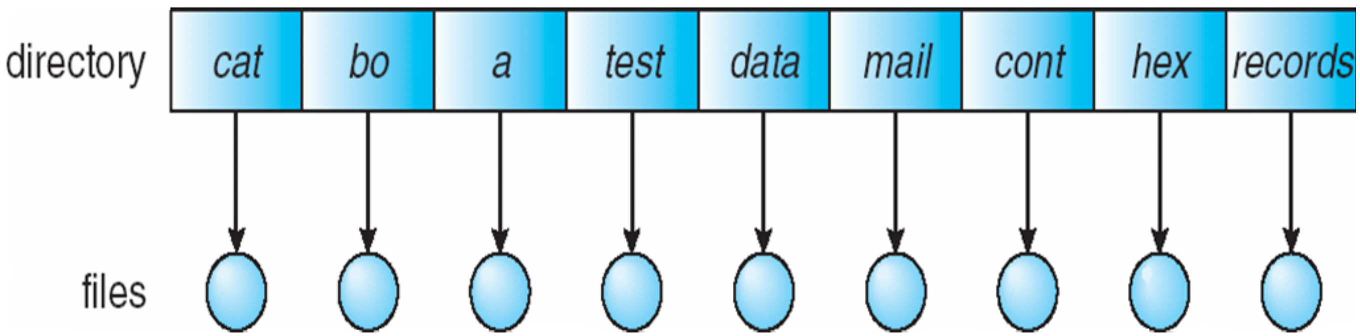
- When considering a particular directory structure, the following operations can be performed on a directory:
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system (e.g. backup copy)

Directory (cont)

- Directories are used to organize files in a file system.
  - to improve efficiency: locating a file quickly
  - for naming: convenient to users
    - two users can have the same name for different files
    - the same file can have several different names
  - to group files: logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

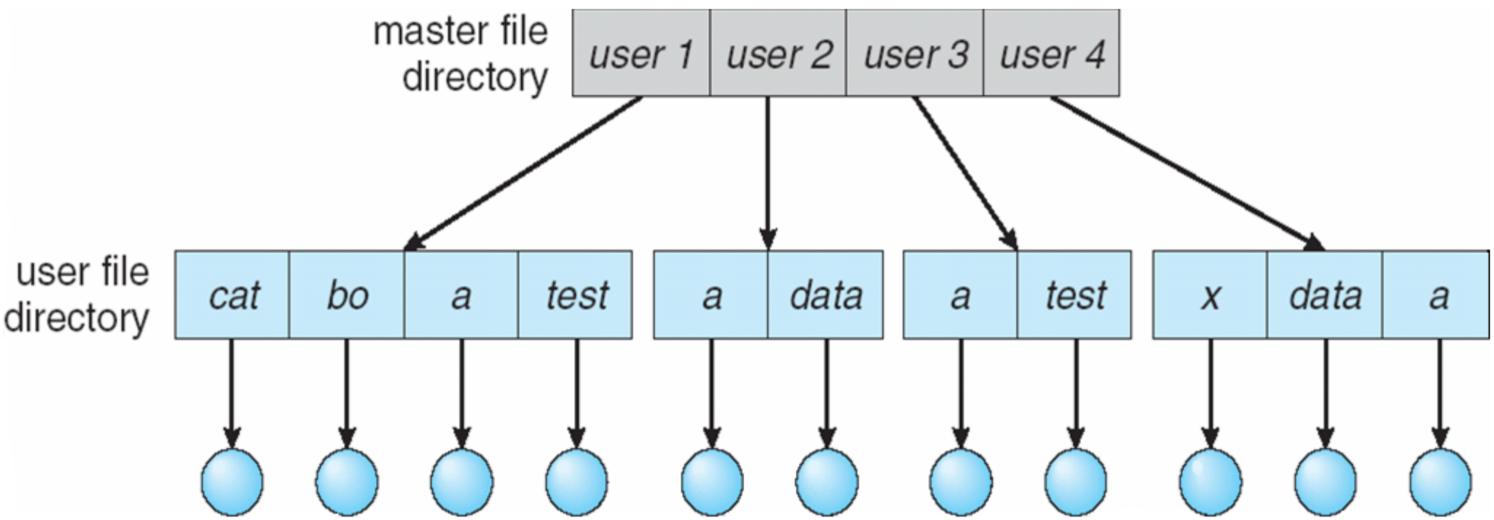
- Now we look at the most common schemes for defining the logical structure of a directory
- The simplest directory structure is single-level directory:
  - all files are contained in the same directory for all users



- Single-level directory has significant limitations:
  - Naming
  - Grouping

Two-Level Directory

- In the two-level directory structure, each user has his/her user file directory (UFD)
  - the UFDs have similar structures but each lists only the files of a single user
  - when a user logs on (or user job starts), the system's master file directory (MFD) is search which is indexed by user name or account number
  - when a user refers to a particular file, only his won UFD is searched

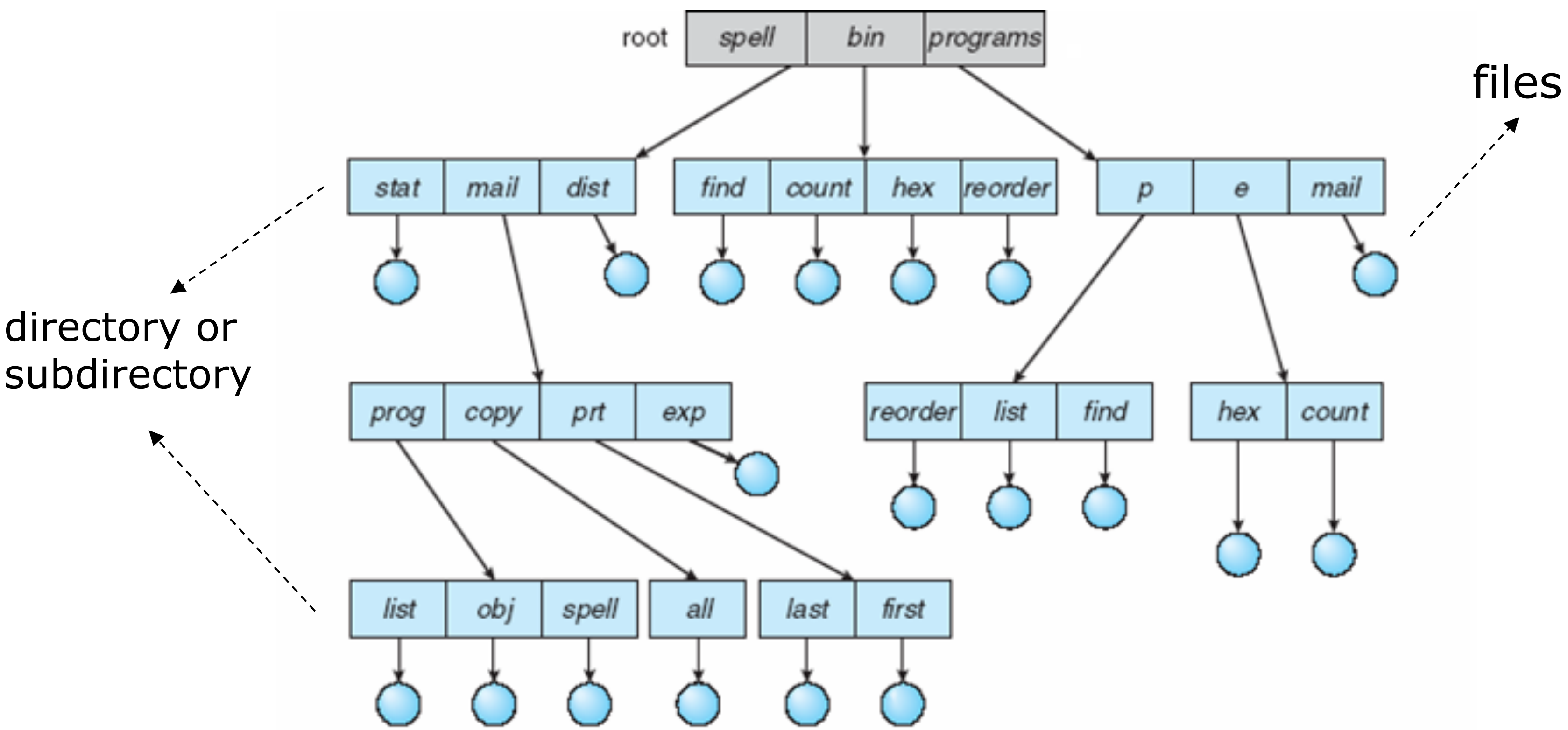


Two-Level Directory (cont)

- The two-level directory solves the name-collision problem
  - this structure isolates one user from another,
  - if access is permitted, then one user must have the ability to name a file in another user's directory
- A two-level directory can be thought of as a tree of height 2:
  - the root is the MFD and the UFDs are its direct descendants
- Every file in the system has a **path name**
  - a *user name* and a *file name* define the path name
- For example, to access file named *test* of *userB*, it can be referred to as */userB/test*
  - *additional syntax may be used to specify the volume:*
    - *e.g. C:\userB\test (using a letter: in MS-DOS)*
    - *or the volume can be treated as a part of the directory name*
- Efficient searching: the sequence of directories searched when a file is named is called the search path
- Still, this method does not have grouping capability

Tree-Structured Directories

- A natural generalization is to extend the directory structure to a tree of arbitrary height
  - a tree is the most common directory structure
- This generalization would allow users to create their own subdirectories and to organize their files accordingly.



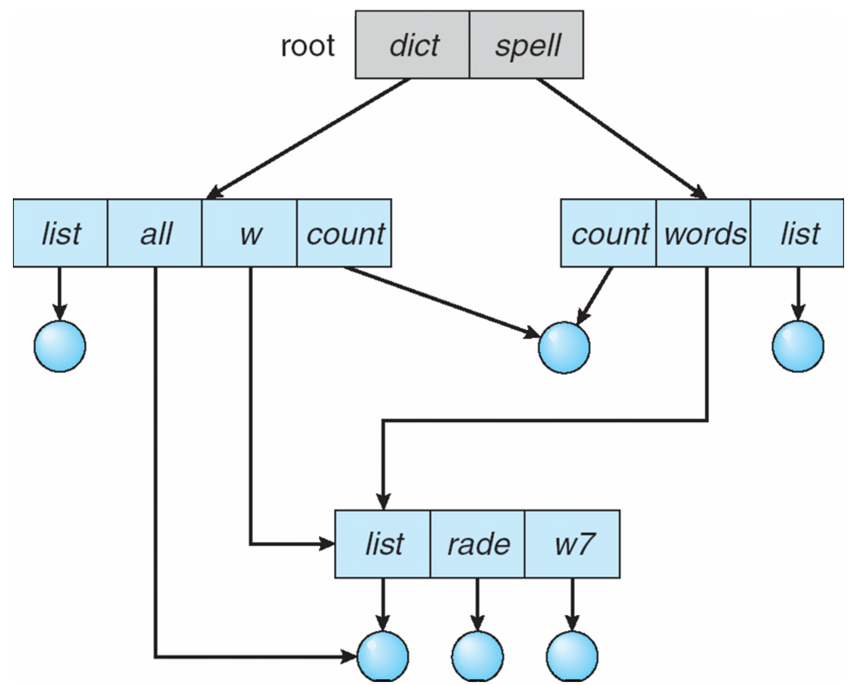
Tree-Structured Directories (Cont)

- In this structure, we achieve:
  - Efficient searching
  - Grouping Capability
- Each process has a **current directory**
  - the current directory should have most of the files that are of current interest or
  - the user should specify a path name or change the current directory
- The initial current directory of the login shell is designated when the user logs in
- Path names can be absolute or relative
  - an **absolute path name** begins at the root and follows a path down to the specified file
    - e.g. C:\users\userB\documents\myfile.c
  - a **relative path name** defines a path from the current directory
    - e.g. ..\documents\myfile.c

Acyclic-Graph Directories

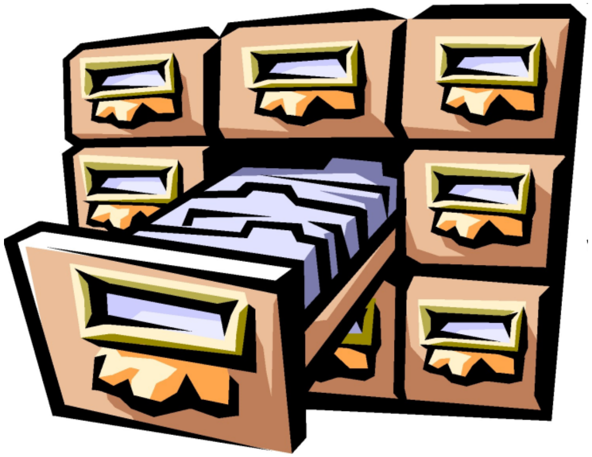
- A tree structure prohibits sharing of files or directories.
- An acyclic graph (i.e. a graph with no cycles) allows directories to **share** subdirectories and files
  - a shared directory or file will exist in file system in two (or more) places at once

- A common way (e.g. in UNIX) is to create a new directory entry called a link to implement shared files and subdirectories
  - a link is effectively a pointer to another file or subdirectory
  - a link is resolved by using the path name to locate the real file



Overview

- File Concept
- Access Methods
- Directory Structure
- **File-System Mounting**
- File Sharing
- Protection

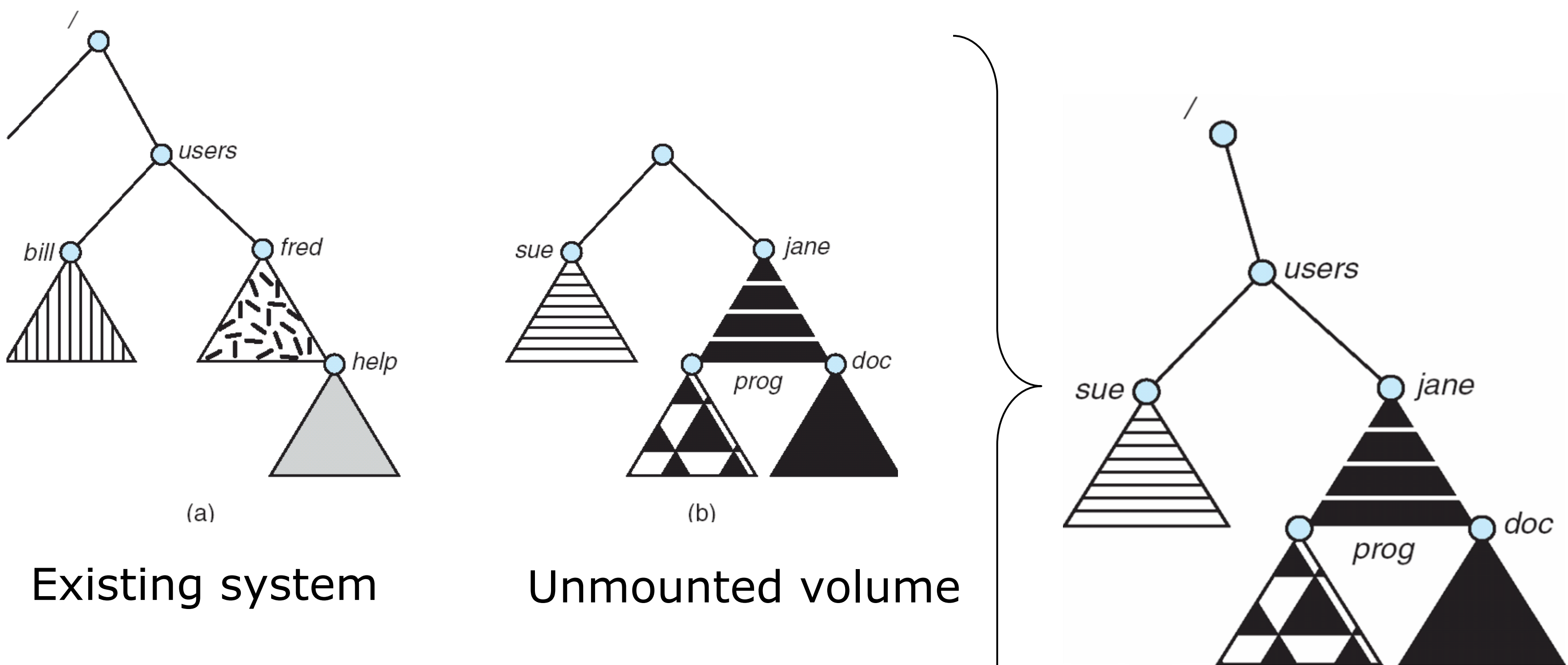


File System Mounting

- A file system must be **mounted** before it can be accessed
  - an analogy is a file that must be opened before it is used
- An unmounted file system is mounted at a **mount point**
  - typically a mount point is an empty directory
- The mount procedure is straightforward:
  - the operating system is given the name of the device and the mount point
    - the file system type is either provided, or the OS inspect the structure and determines it
  - next, the OS verifies that the device contains a valid file system
  - finally, the OS notes in its directory structure that a file system is mounted at the specified mount point

File System Mounting (cont)

- For example:



Existing system

Unmounted volume

mount point

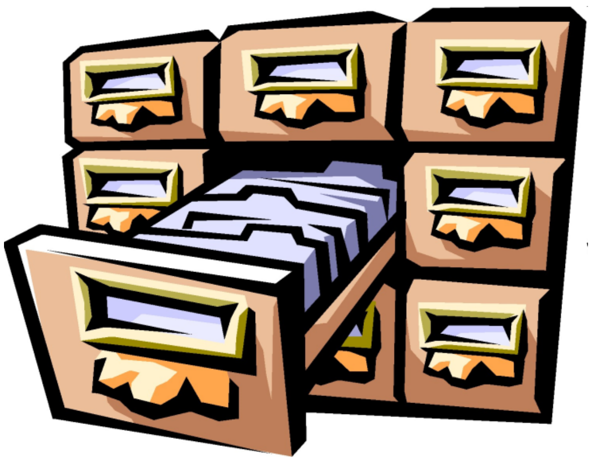
File System Examples

- MS Windows maintains an extended two-level directory structure, with devices and volumes assigned drive letter.
  - the path to a specific file takes the form of

    drive-letter:\path\to\file
  - A file system may be mounted anywhere in the directory tree, just as UNIX does
  - Windows automatically discover all devices and mount all located file system at boot time.
- In UNIX the mount commands are explicit
  - a system configuration file contains a list of devices and mount points for automatic mounting at boot time
    - other mounts may be executed manually
  - Mac OS X behaves much like BSD UNIX: all file systems are automatically mounted under /Volumes directory
    - The GUI though shows the file systems as if they were all mounted at the root level

Overview

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- **Protection**



Protection

- When information is stored in a computer system, we want to keep it safe
  - from physical damage (the issue of reliability) and
  - improper access (the issue of protection)
- Reliability is generally provided by duplicate copies of files (backup)
- The need to protect files is a direct result of the ability to access files
  - Protection mechanisms provide controlled access by limiting the types of file access that can be made. Several types of operations may be controlled:
    - **Read:** read from the file
    - **Write:** write or rewrite the file
    - **Execute:** load the file into memory and execute it
    - **Append:** write new information at the end of the file
    - **Delete:** delete the file and free its space for possible reuse
    - **List:** list the name and attribute of the file

Access Lists and Groups

- The most common approach to the protection problem is to make access dependent of the identity of the user
  - The most general scheme to implement identity dependent access is to associate with each file and directory an **access-control list** (ACL)
- Since constructing such a list is tedious, many systems use a condensed version of the access list, based on the following three classification
  - owner: the user who created the file
  - group: a set of users who may need to share the file
  - universe: all other users
- In the Unix system, these three classes of users are defined by three fields of 3 bits each, *rwx*

examples:

-rw-rw-r--
drwx------
drwxrwxr-x
drwxrwx---
-rw-r--r--

a) **owner access**	e.g. 7	⇒	rwx 111
b) **group access**	e.g. 6	⇒	rwx 110
c) **public access**	e.g. 1	⇒	rwx 001

r: read access
w: write access
x: execution access

Overview

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