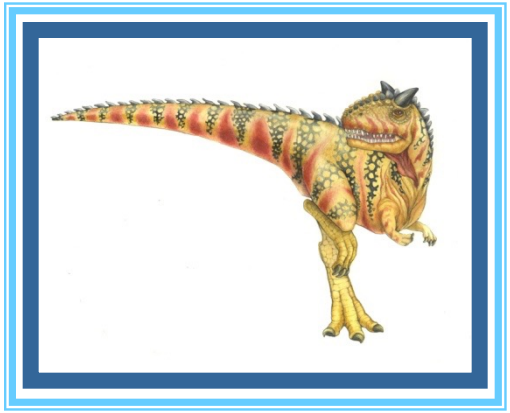
Chapter 13: File-System Interface

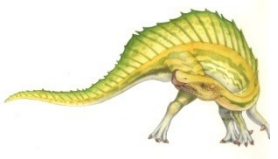


Chapter 13: File System Interface

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

Objectives

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



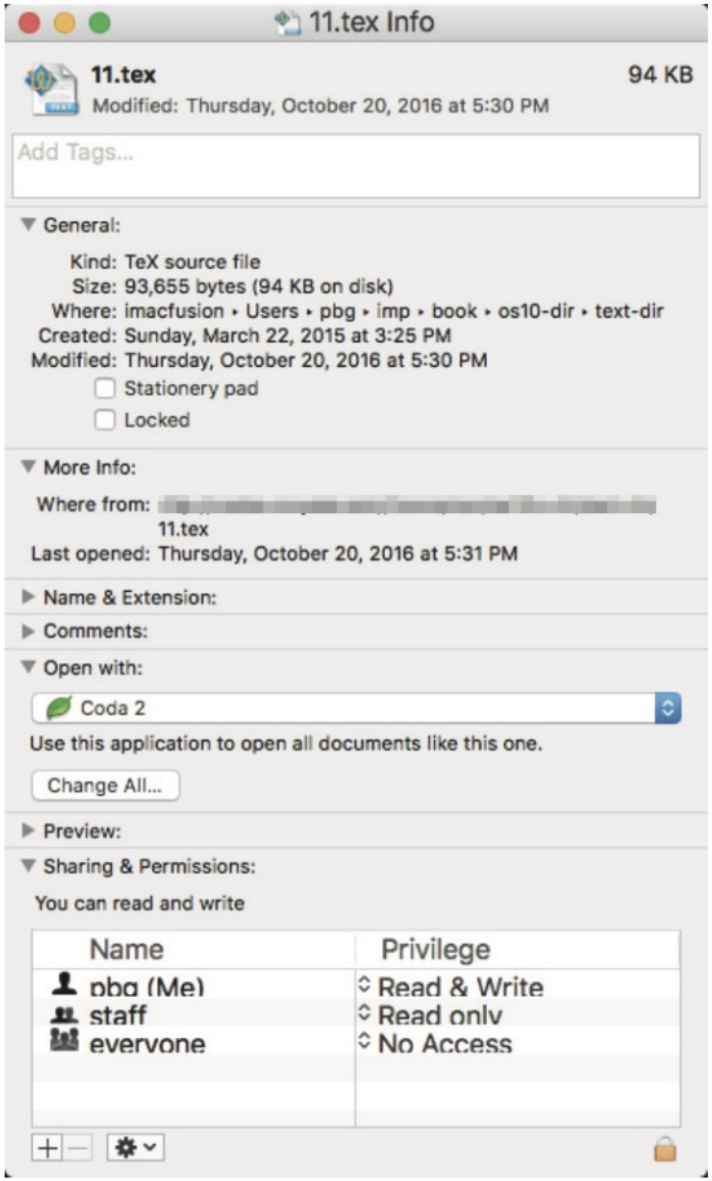
File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by the file's creator
  - Many types, consider text file, source file, executable file

File Attributes

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

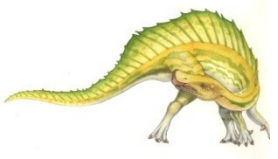
File info Window on Mac OS X



File Operations

- File is an ADT or abstract data type
- Create – create a file
- 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, preparing file for subsequent access
- Close (F_i) – move the content of entry F_i in memory to directory structure on disk

- Such operations involve the changes of various OS kernel data structures



Open Files

- Several data structures are needed to manage open files:
  - Open-file tables: tracks open files, system-wide open-file table, and per-process open-file table
  - File pointer: pointer to last read/write location, per process that has the file open
  - File-open count: counting the number of processes that the file has been opened – to allow removal of data from the open-file table when the last processes closes it (when file-open count is zero)
  - Disk location of a file: cache of data access information
  - Access rights: per-process access mode information

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, 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

Access Methods

- Sequential Access – simplest access method

```
read next
write next
reset
no read after last write
        (rewrite)
```

- Direct Access – file is fixed length logical records

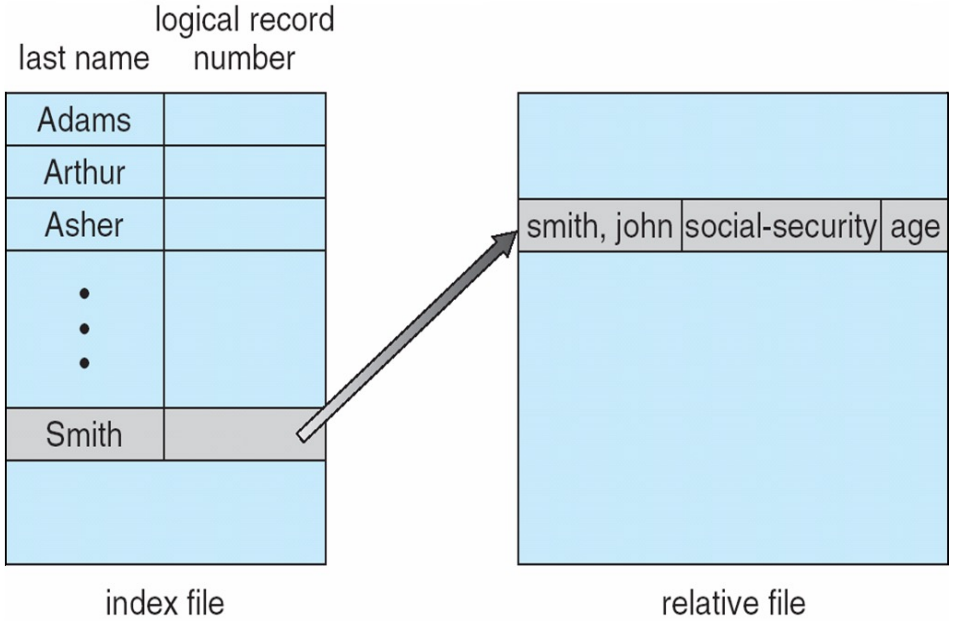
```
read n
write n
position to n
        read next
        write next
rewrite n
```

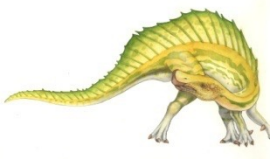
n = relative block number

- Relative block numbers allow OS to decide where file should be placed
  - See disk block allocation problem in Chapter 14

Other Access Methods

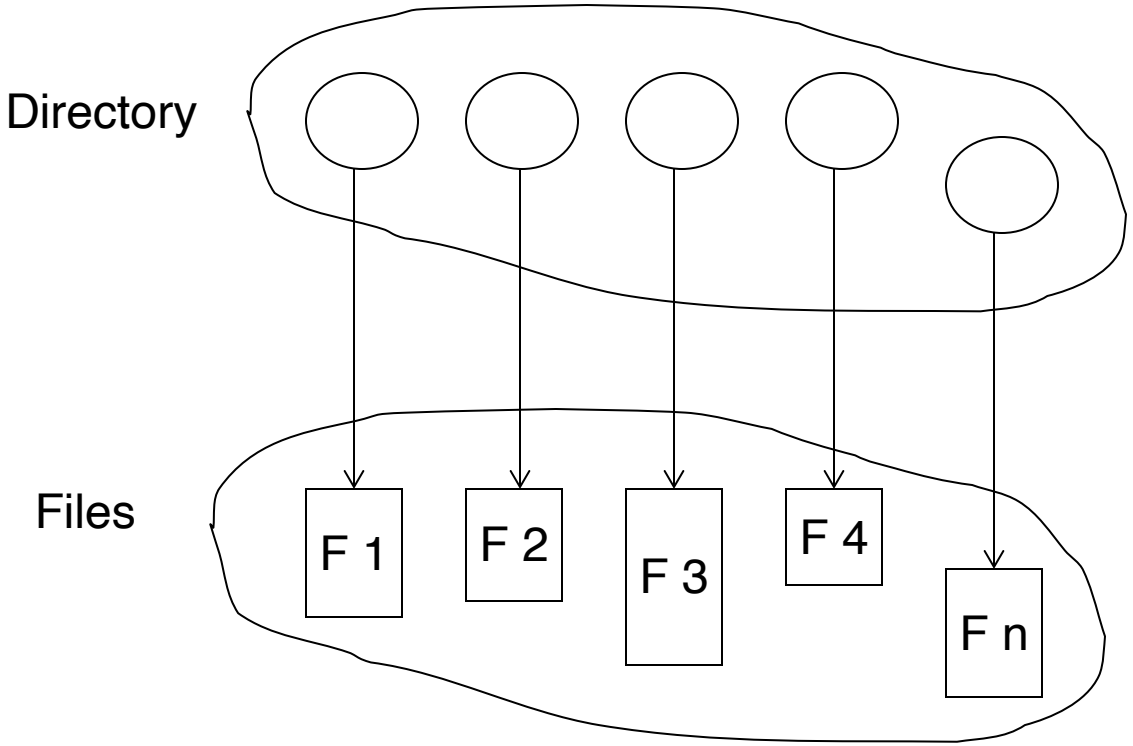
- Other file access methods can be built on top of a direct-access method
- Generally, involve creation of an **index** for a file
  - Keep index in memory for fast location of the data to be operated on
  - If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM) is an example
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example



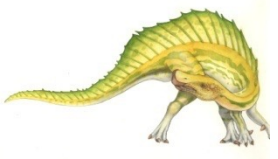


Directory Structure

- A collection of nodes containing information about all files

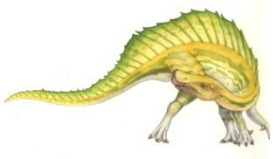


Both the directory structure and files reside on disk

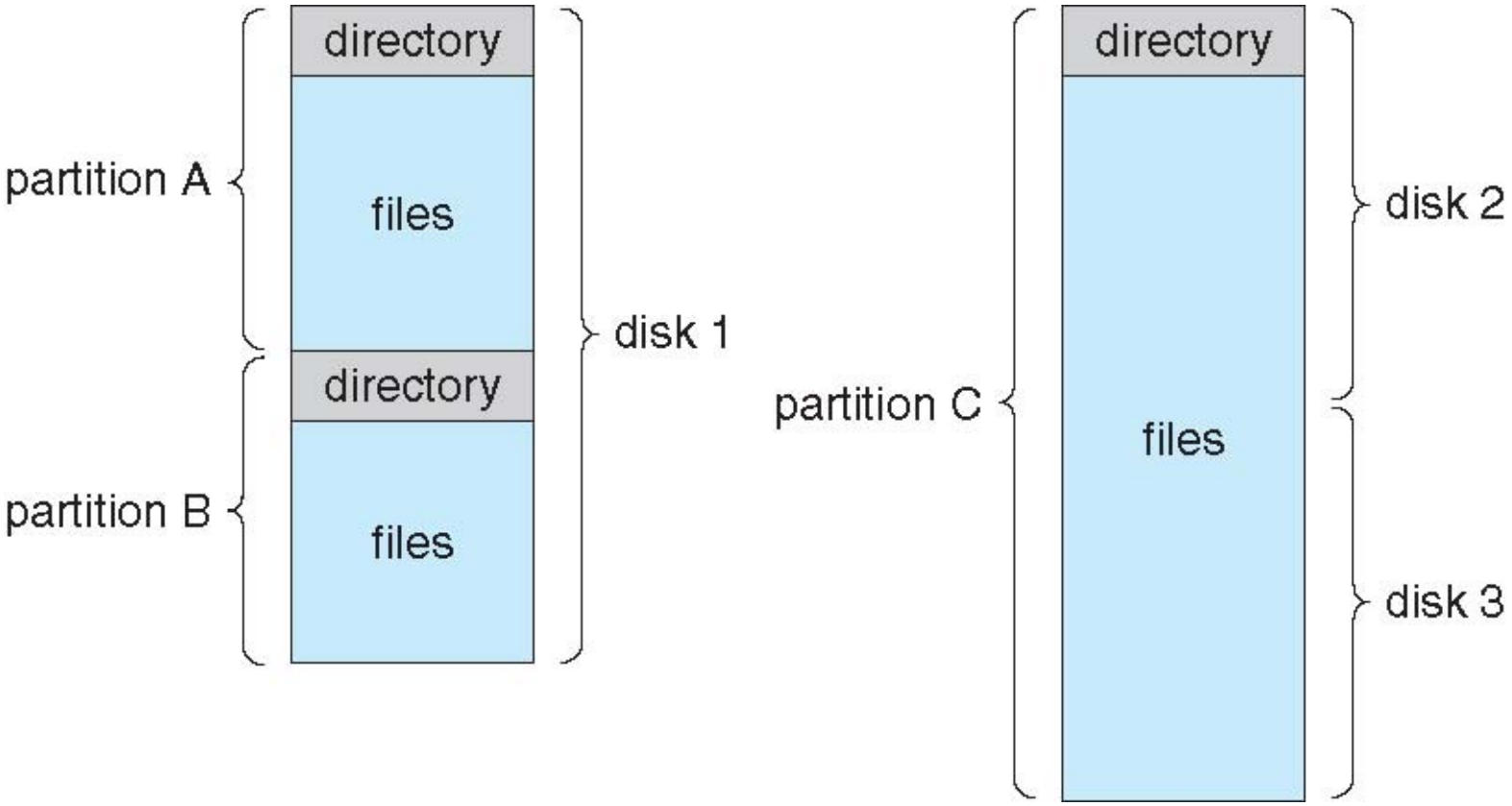


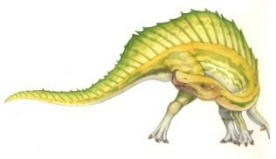
Disk Structure

- 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 are also known as minidisks, slices
- An entity on a disk containing a file system known as a volume
- Each volume containing a file system also keeps track of the file system info in device directory or volume table of contents
- Other than general-purpose file systems, there are many special-purpose file systems, frequently within the same operating system or computing systems



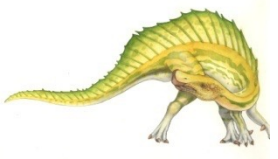
A Typical File-system Organization





Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

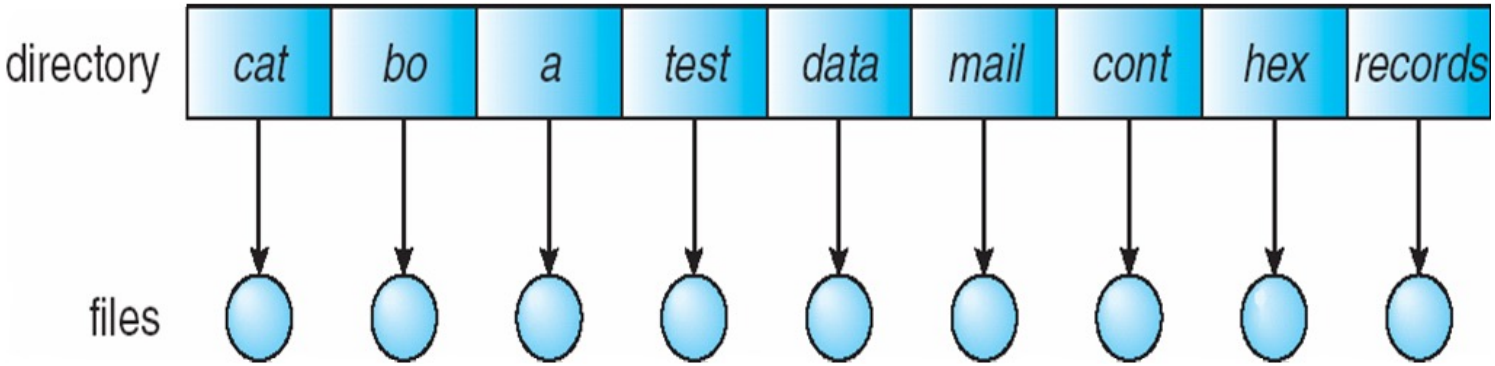


Organize the Directory (Logically) to Obtain

- **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, my comp3511, …)

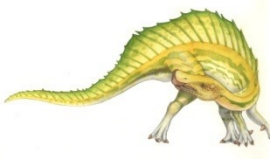
Single-Level Directory

- A single directory for all users



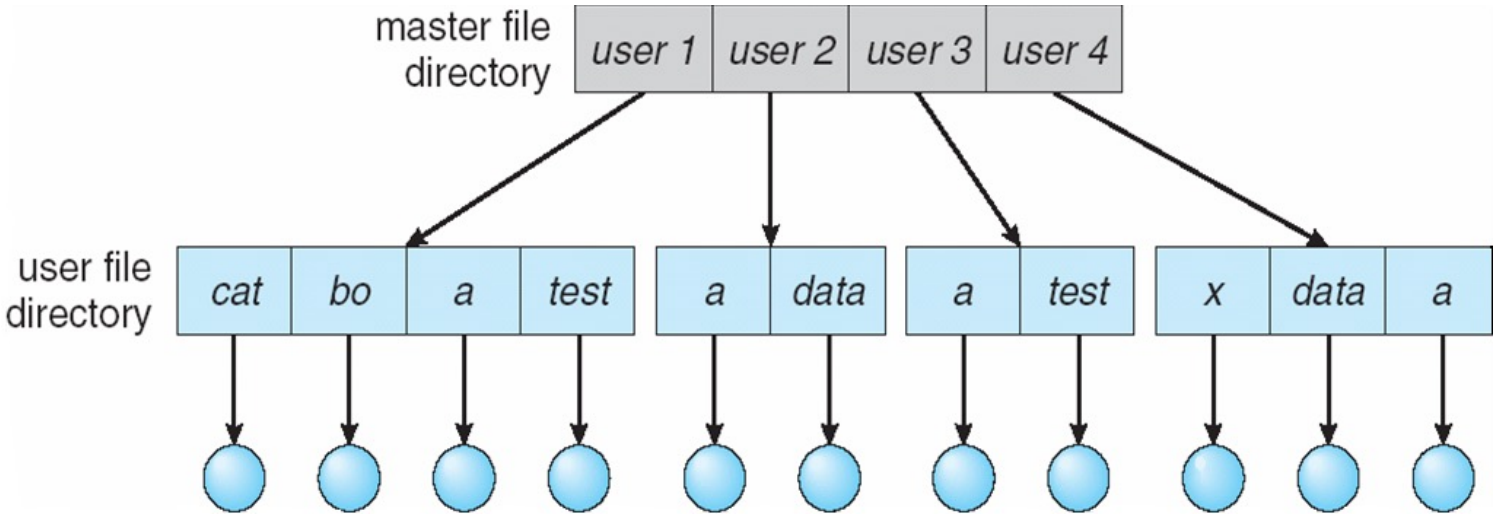
Naming problem

Grouping problem



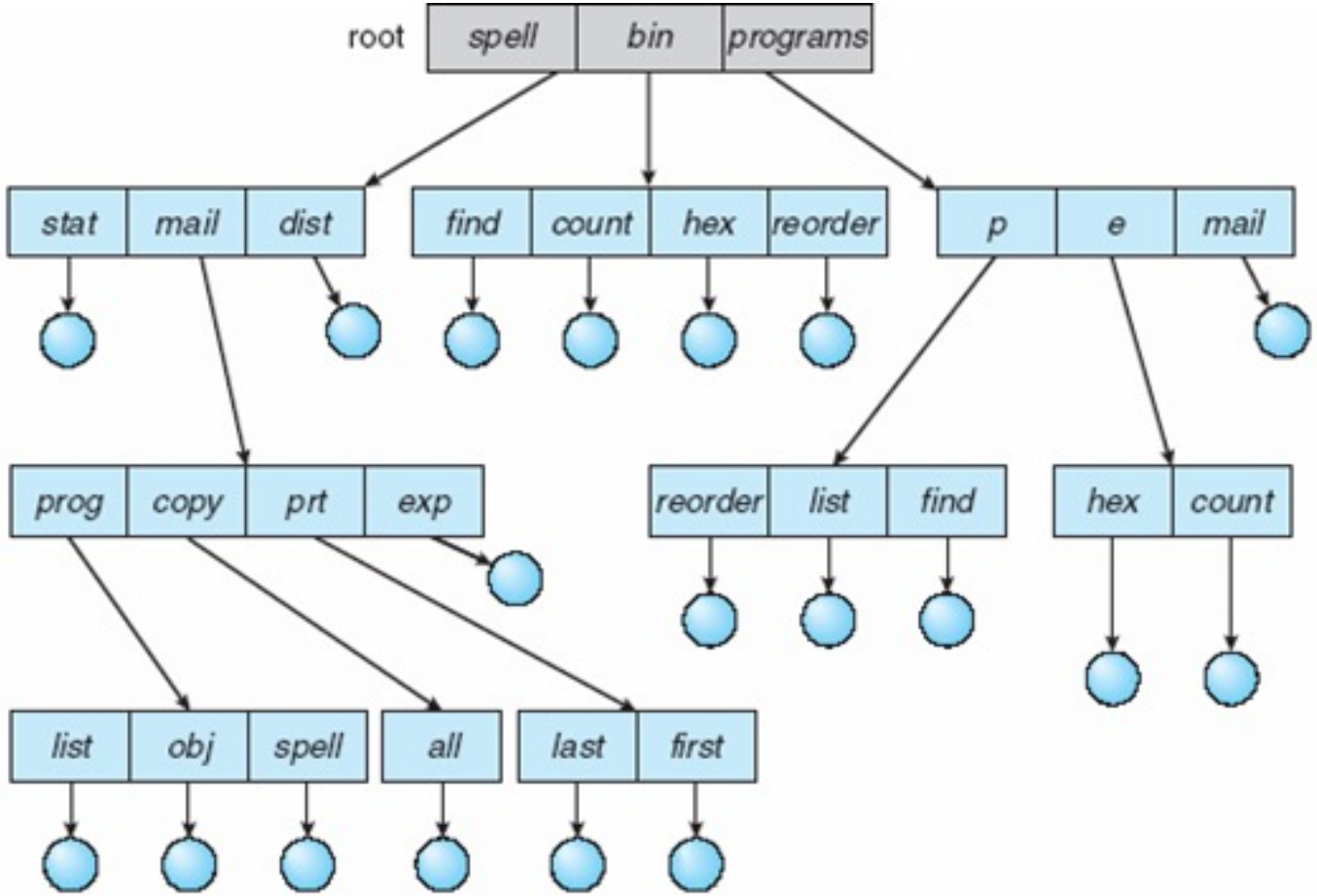
Two-Level Directory

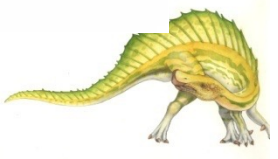
- Separate directory for each user



- Path name – need a pathname to identify a file/dir, e.g., /user1/cat
- Can have the same file name under different users (paths)
- More efficient searching than single-level directory
- No grouping capability

Tree-Structured Directories





Tree-Structured Directories (Cont.)

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

Tree-Structured Directories (Cont)

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

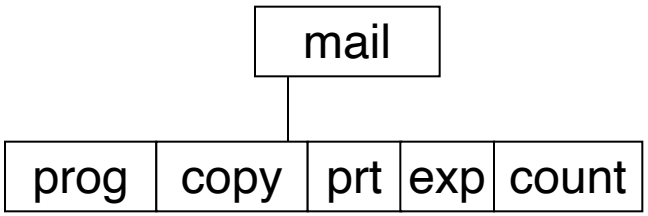
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

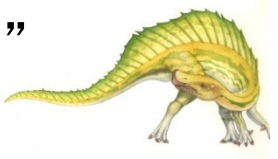
```
mkdir <dir-name>
```

Example: if in current directory `/mail`

```
mkdir count
```

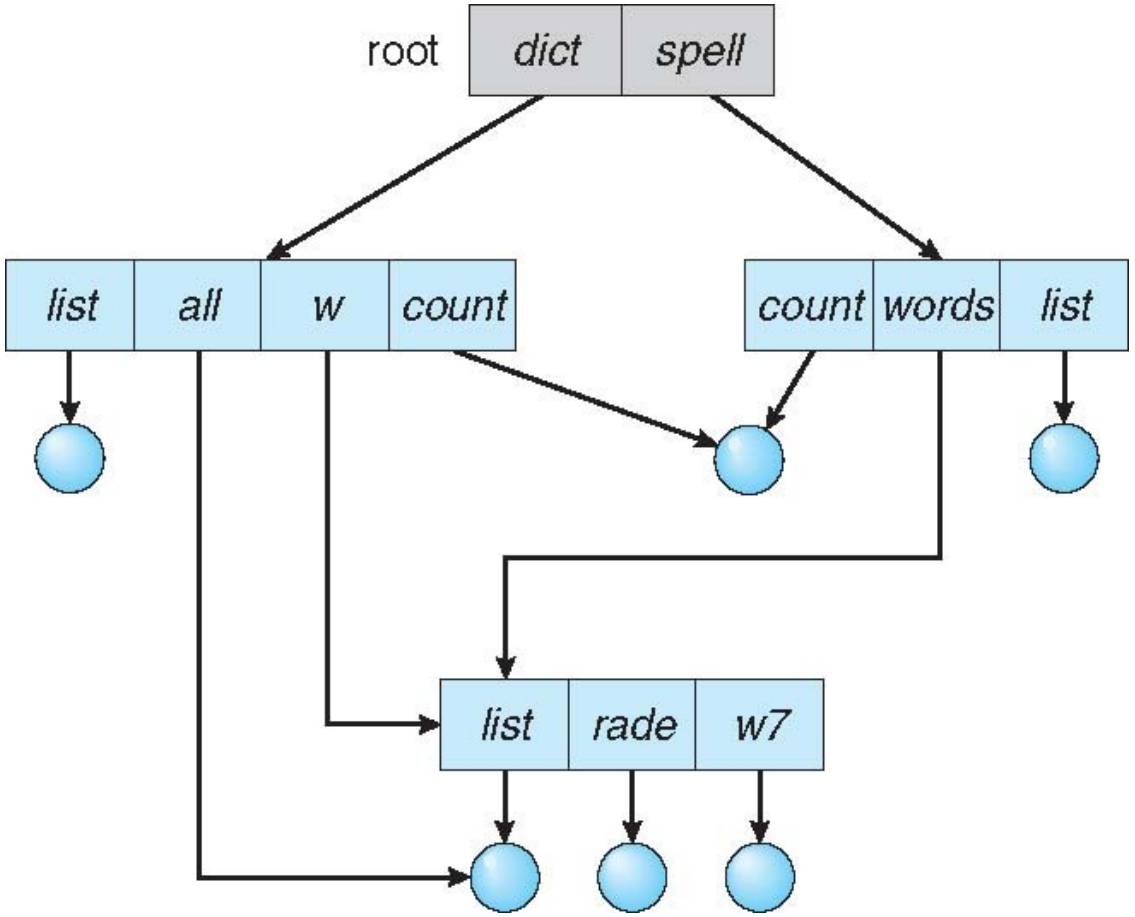


Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”



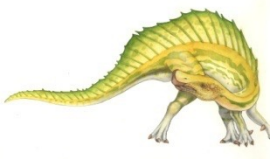
Acyclic-Graph Directories

- Have shared subdirectories and files – more flexible and complex

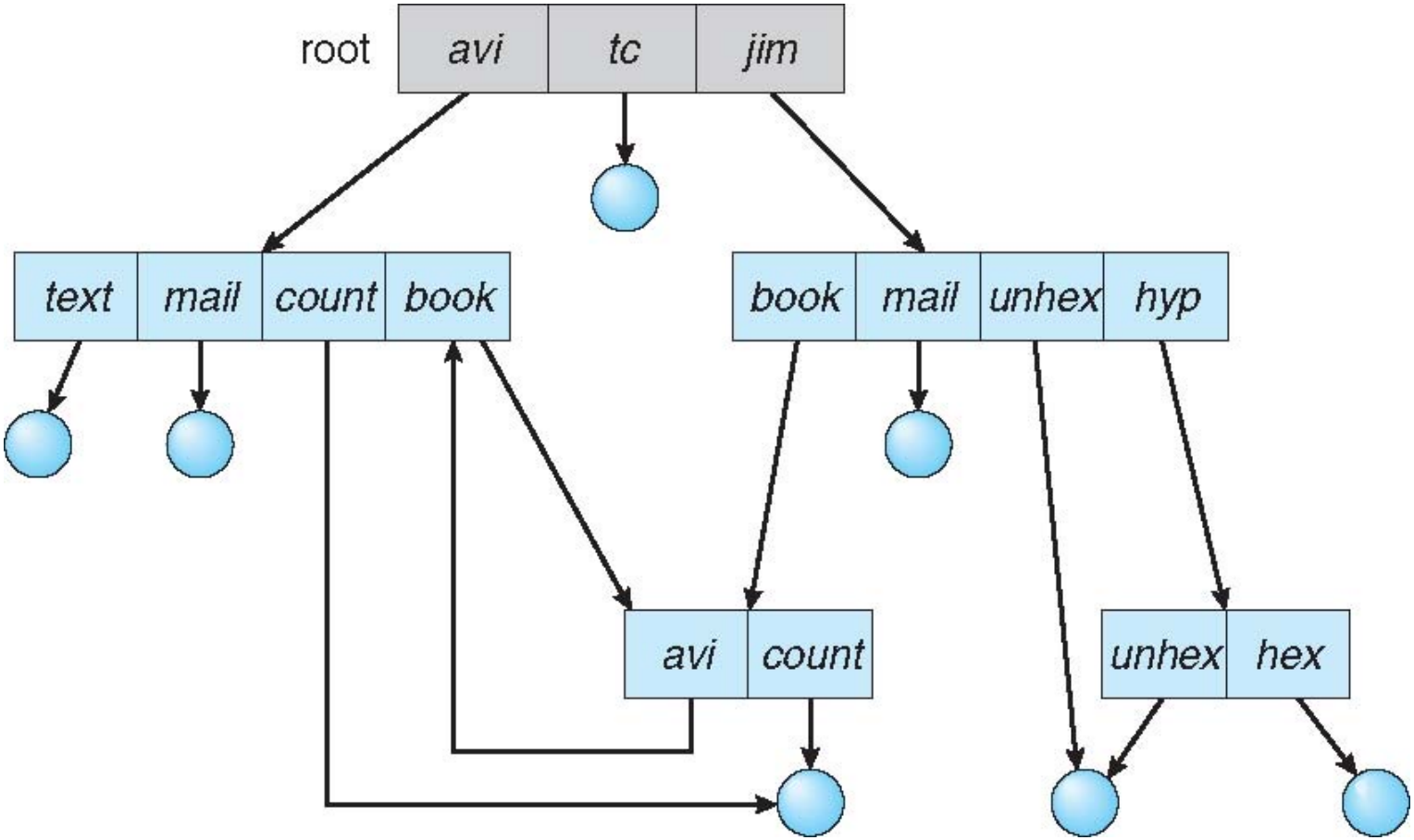


Acyclic-Graph Directories (Cont.)

- New directory entry type
  - Link – another name (pointer) to an existing file
  - Resolve the link – follow pointer to locate the file
- Two different (path) names (aliasing)
  - Ensure not traversing shared structures more than once
- Deletion might lead to that dangling pointers that point to empty files or even wrong files
- There is also difficulty ensuring there is no cycles in a graph – complexity associated with it



General Graph Directory

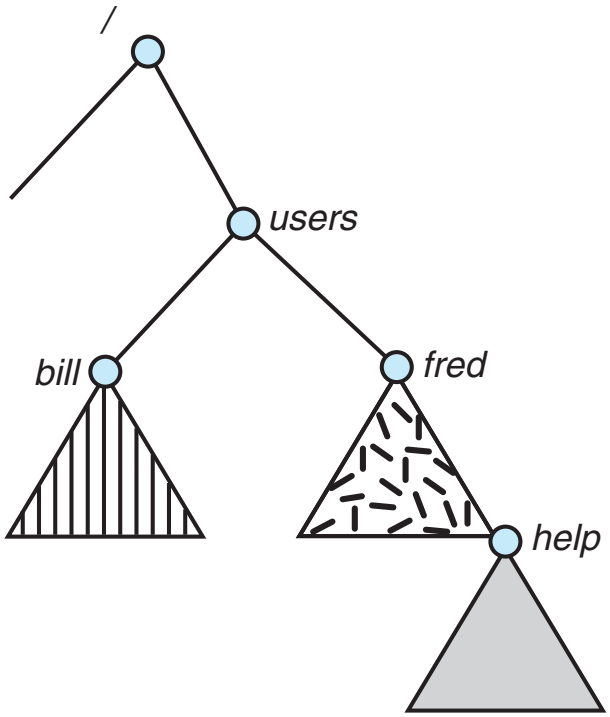


General Graph Directory (Cont.)

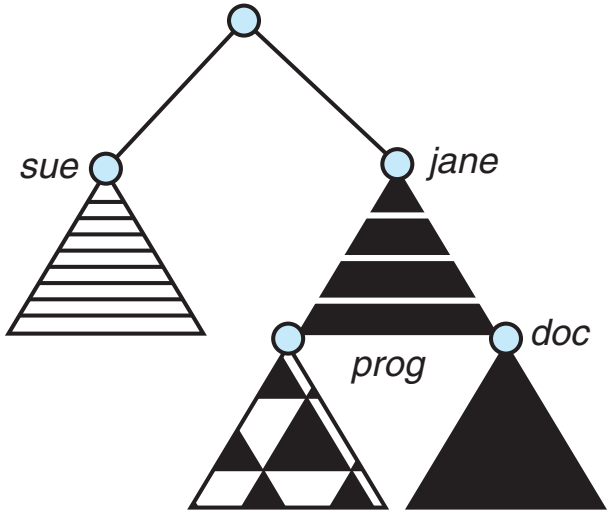
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories – sometime not convenient
  - Every time a new link is added use a cycle detection algorithm to determine whether there is a cycle or not – time consuming

File System Mounting

- A file system must be mounted before it can be accessed – just like a file must be opened before it is used
- A unmounted file system (i.e., Fig. (b)), to be mounted at a mount point



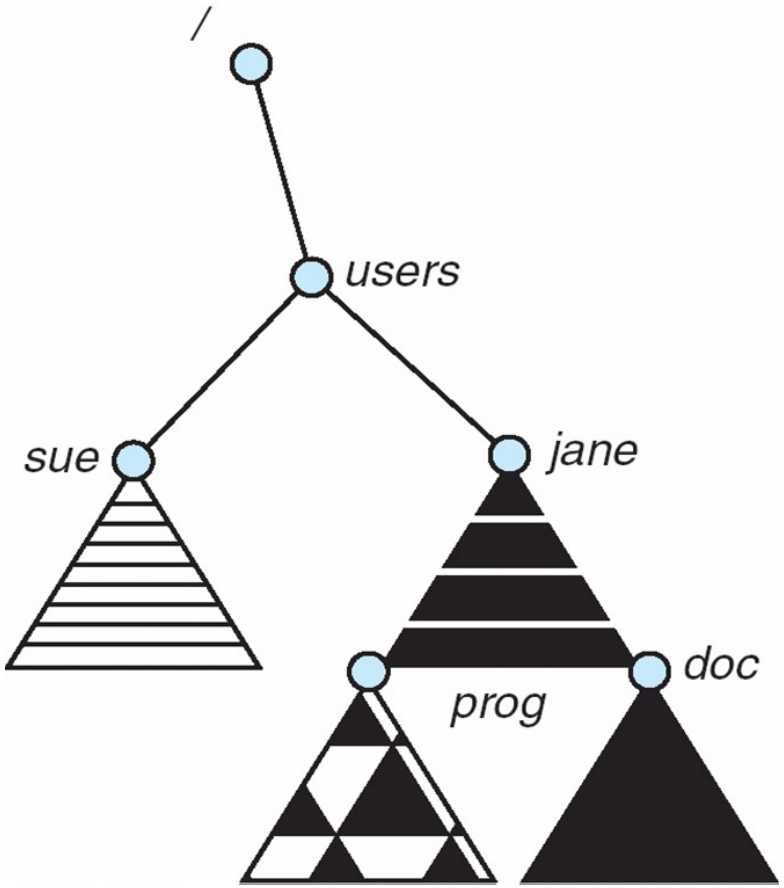
(a)

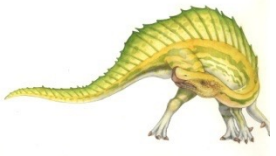


(b)

Mount Point

- Volume is mounted at /users

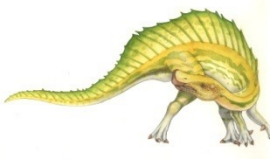




File Sharing

- Sharing of files in multi-user systems is desirable
- Sharing may be done through a protection scheme
- In distributed systems, files may be shared across a network
  - Network File System (NFS) is a common distributed file-sharing method

- With a multi-user system
  - User IDs identify users, allowing permissions and protections to be per-user
  - Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory



Protection

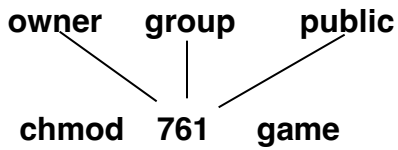
- File owner/creator of the file should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

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

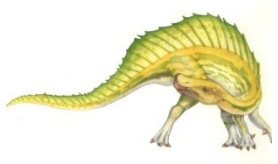
			RWX
a) owner access	7	⇒	1 1 1
b) group access	6	⇒	RWX 1 1 0
c) public access	1	⇒	RWX 0 0 1

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

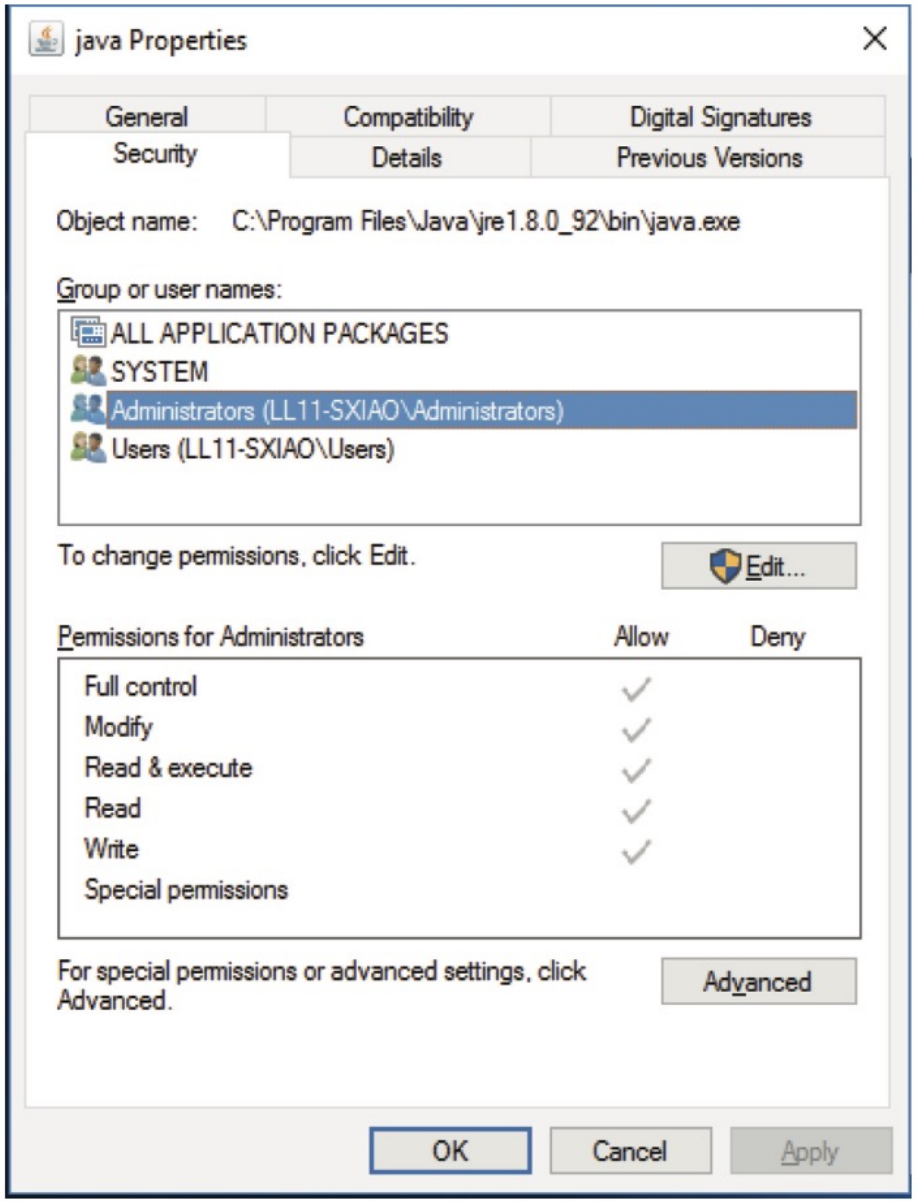


Attach a group to a file

```
chgrp       G       game
```



Windows 10 Access-Control List Management



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/

End of Chapter 13