File System

- Objectives
  - Function of file systems
  - Describe the interfaces to file systems
  - Discuss design trade-offs
  - Explore file system protection

File Concept

- Contiguous logical address space
- File is an abstraction by OS.
- These are mapped to physical devices

- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- 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

Common terms related to files

- Common terms
  - Field is the basic element of data
  - A record is collection of related fields
  - A file is a collection of similar records
  - A database is a collection of related data/files
- Typical operations to be supported
  - Retrieve_all: Retrieve all the records in the file
  - Retrieve_one: Retrieve one record
  - Retrieve_next: Retrieve next
  - Retrieve_previous: Retrieve previous
  - Insert_One
  - Delelete_One
  - Update_one
  - Retrieve_few

File Management Systems (FMS)

- A file management system is the set of system software that provides services and applications in the use of files. Users can access files only through FMS.
- This relieves the programmer of necessity of developing special-purpose software for each application.
- Objectives of FMS
  - To meet the data management needs and requirements of the user: storage of data and ability to perform operations
  - Optimize the performance regarding throughput and response time
  - To provide I/O support for variety of storage devices
  - To prevent the potential lost of destroyed data
  - To provide I/O support for multiple users.

Minimal et of requirements of FMS

- Each user should be able to create, delete, read, change files
- Each user may have controlled access to other user's files
- Each user should control access to his/her files
- Each user should be able to restructure the files
- Each user should be able to move data between files.
- Each user should be able to backup and recover the user's files in case of damage
- Each user should be able to access the files using symbolic names.

File Attributes

- **Name** – only information kept in human-readable form
- **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
- Information about files are kept in the directory structure, which is maintained on the disk

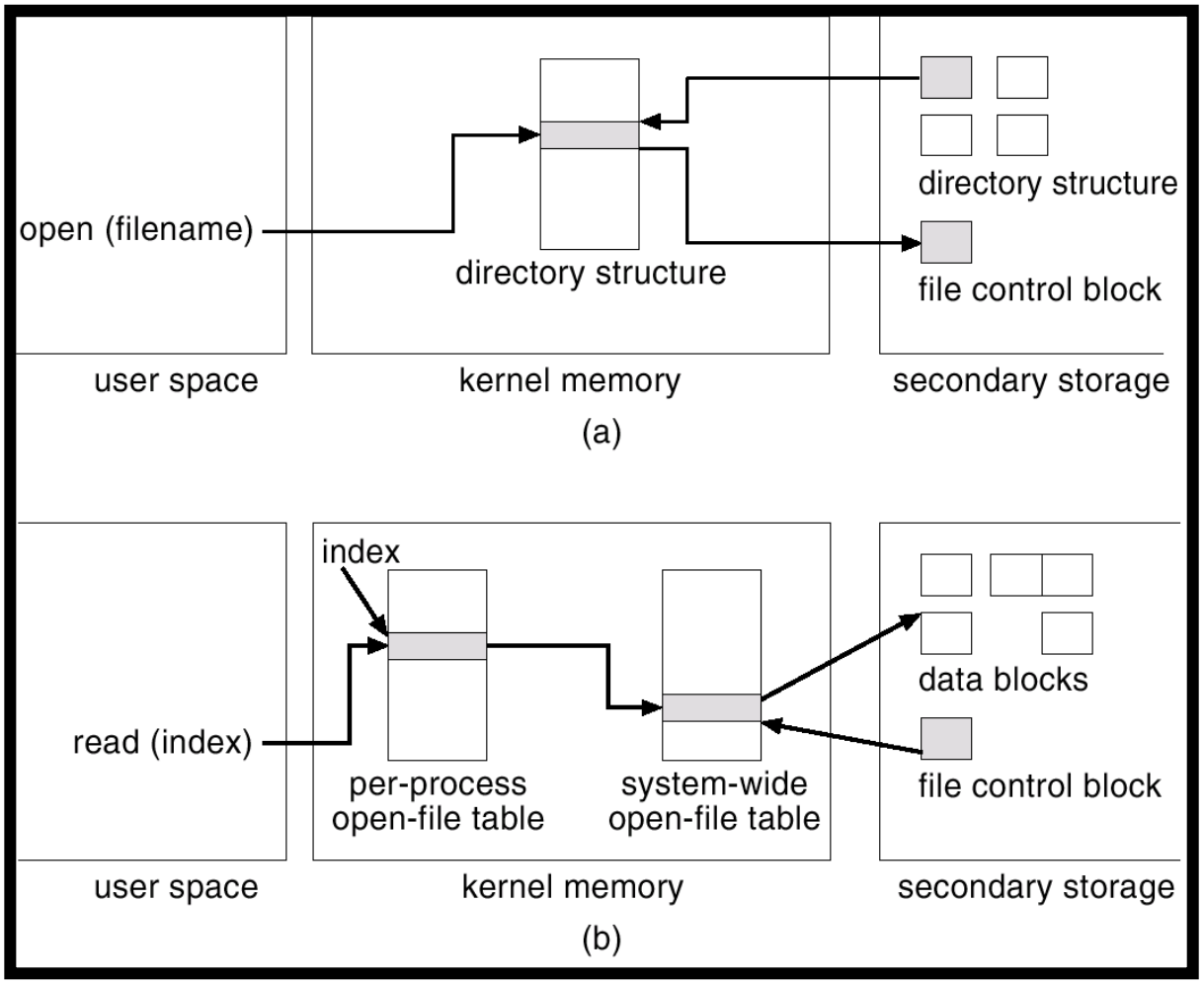
File Operations

- A file is a abstract data type.
- File operations are as follows
  - Create
  - Write : File position pointer
  - Read
  - file seek – reposition the pointer within file
  - Delete: Search a directory
  - Truncate: Keep the attributes and delete the content.
  - 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

- The OS keeps a small table “Open-file table” containing information about all open files
- The open() system call returns a pointer to the entry in Open-file table.
- OS maintains two kinds of internal tables
  - A per-process table: tracks all the files the process has open
    - Current pointer is found here.
  - A system-wide table: An entry in per-process tables points to a system-wide open-file table.
    - The system-wide table contains process-independent information
      - Location of file, access dates and file size.
- Several pieces of data are needed to manage open files:
  - 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

In-Memory File System Structures



(a) File open (b) File read

Open File Locking

- Provided by some operating systems and file systems
- Mediates access to a file
- Similar to reader-writer locks
- A **shared lock** is similar to reader lock
- An **exclusive lock** is similar to writer lock.
- Programmers should be careful in holding exclusive locks for a longer times.
- 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 Locking Example – Java API

```
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 Locking Example – Java API (cont)

```
                    // 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
                    exclusiveLock.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	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	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	binary file containing audio or A/V information

File structures

- **How many file structures should be supported by OS ?**
  - More file structures implies more complexity
- Suppose if OS supports five file structures, code should be provided for all.
- UNIX considers file as a sequence of 8-bit bytes.

Access Methods

- **Sequential Access**

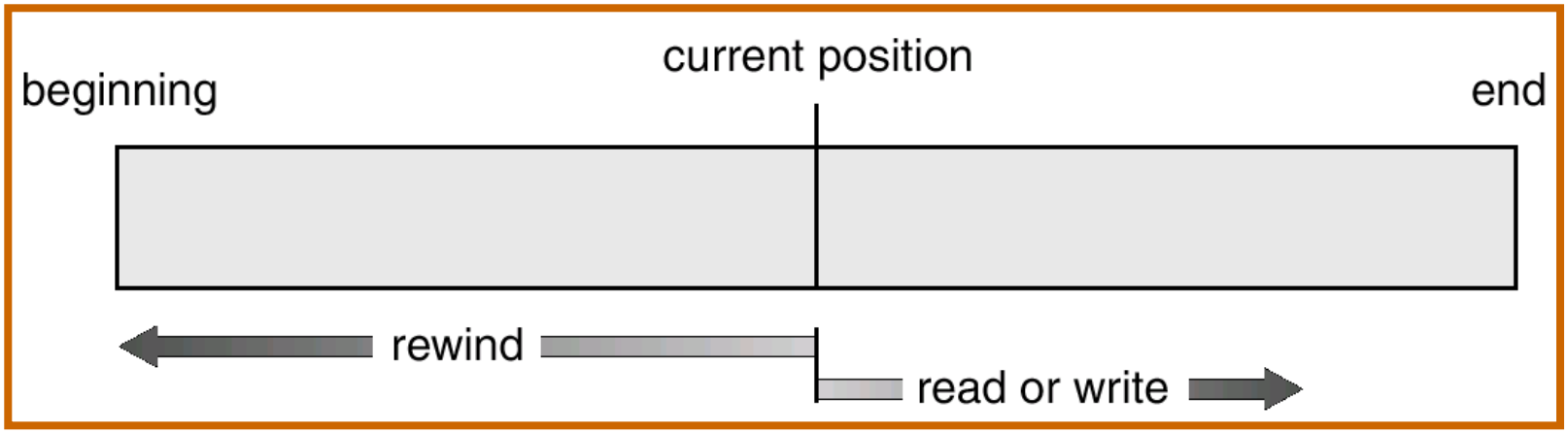
  read next
  write next
  reset
  no read after last write
  (rewrite)

- **Direct Access**

  read *n*
  write *n*
  position to *n*
  read next
  write next
  rewrite *n*

  n = relative block number

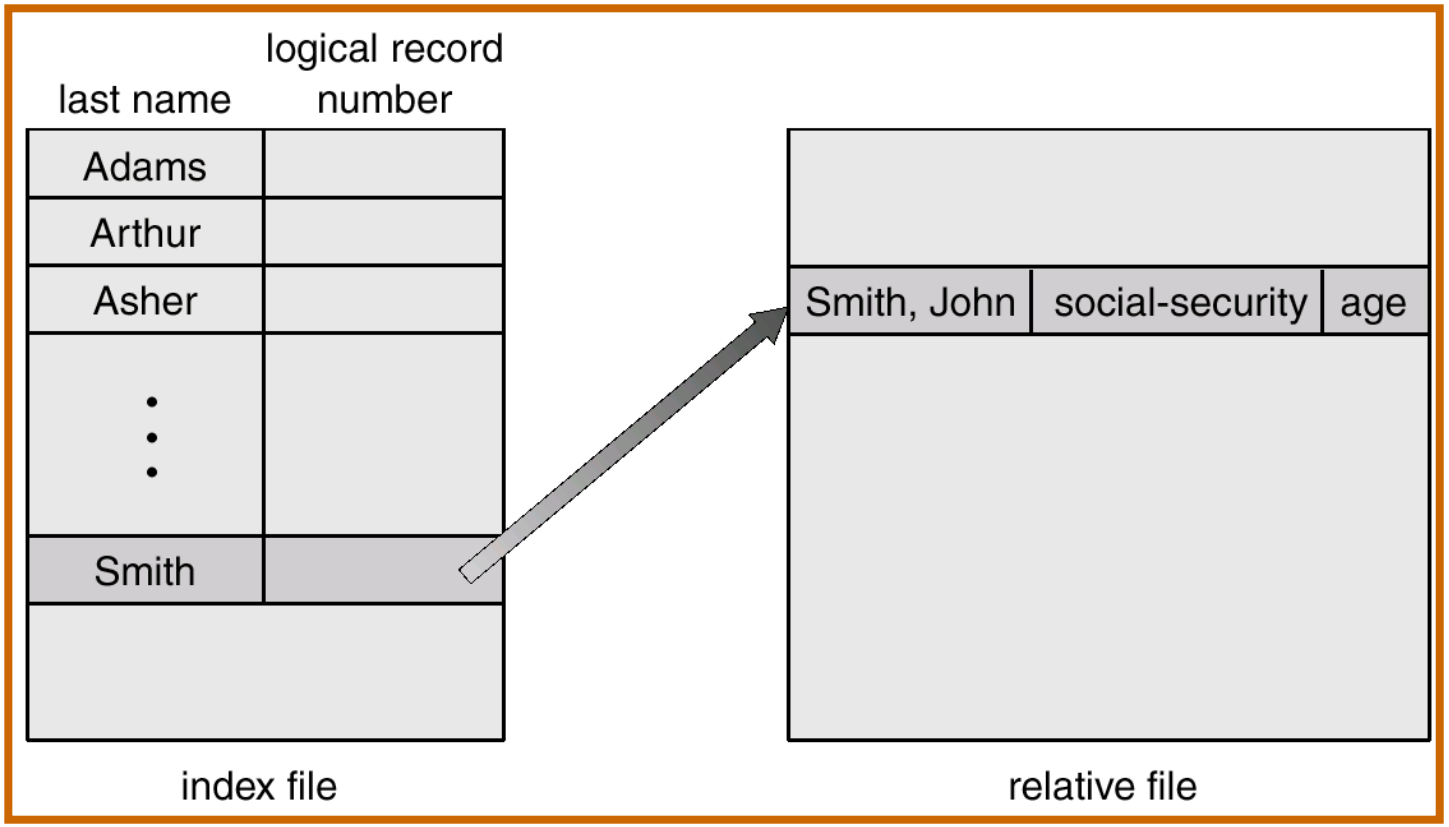
Sequential-access File



Simulation of Sequential Access on a Direct-access File

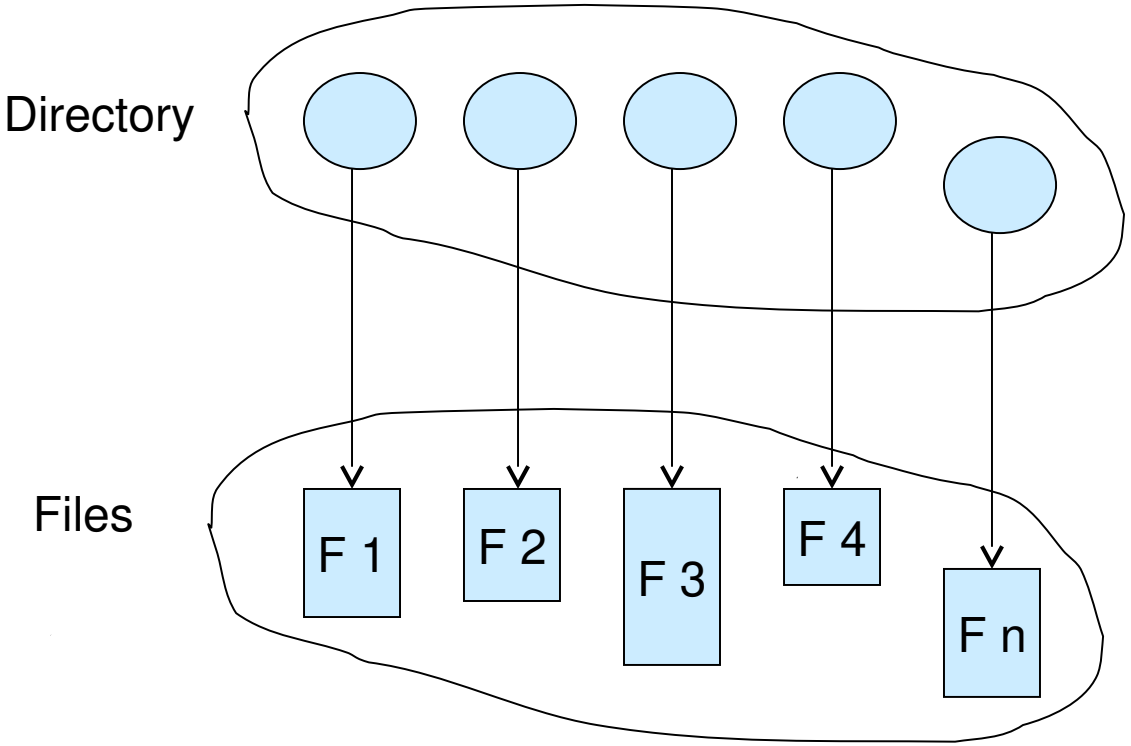
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;

Example of Index and Relative Files



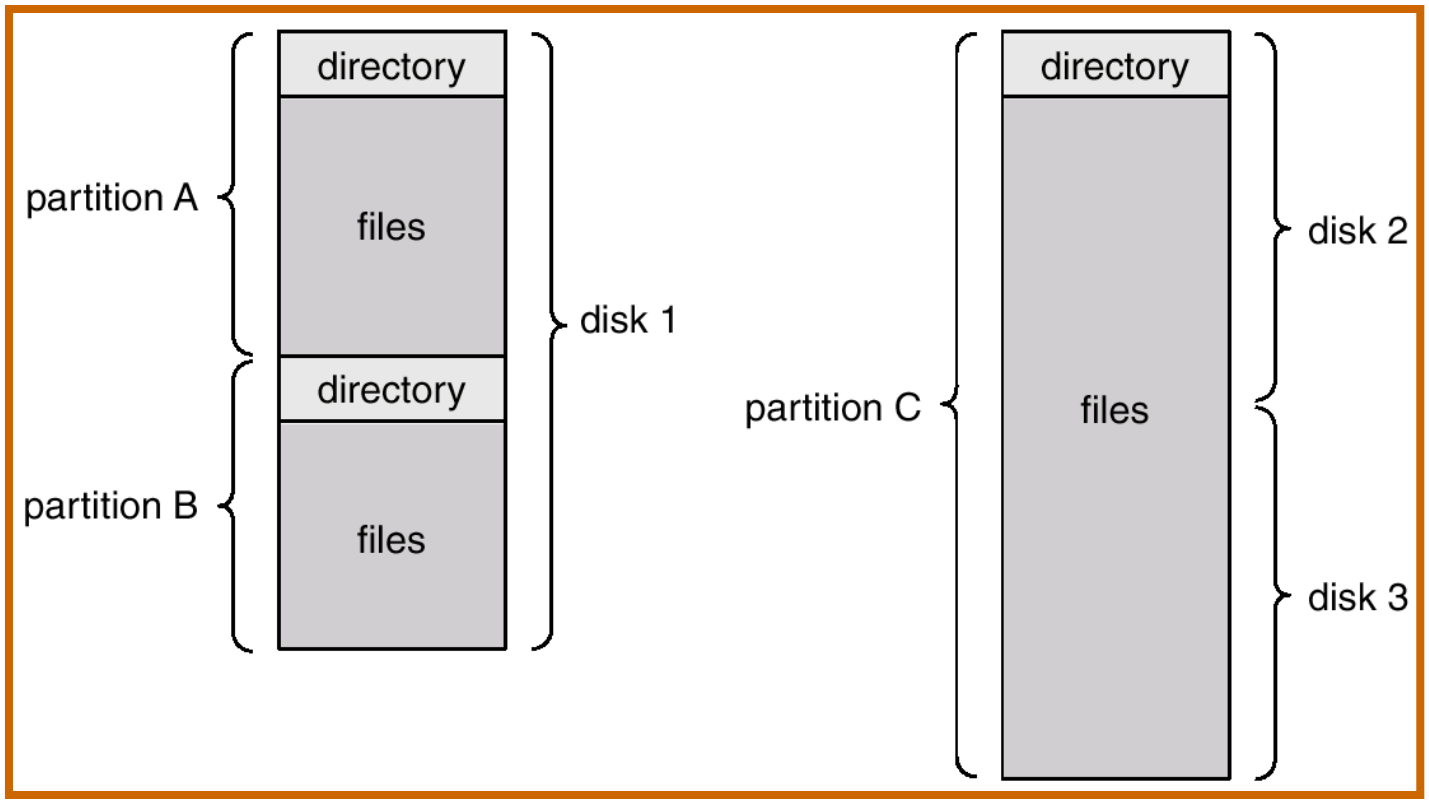
Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

A Typical File-system Organization



Information in a Device Directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID
- Protection information (discuss later)

Operations Performed on Directory

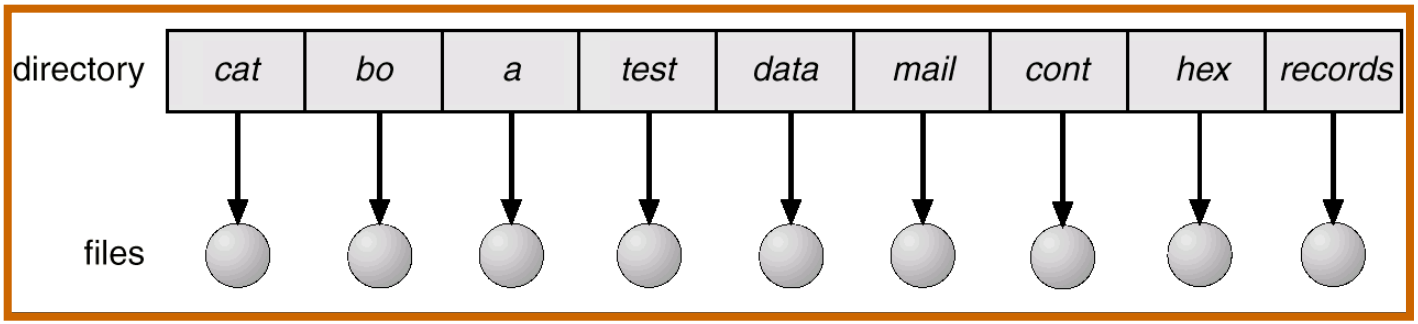
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

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, …)

Single-Level Directory

- A single directory for all users

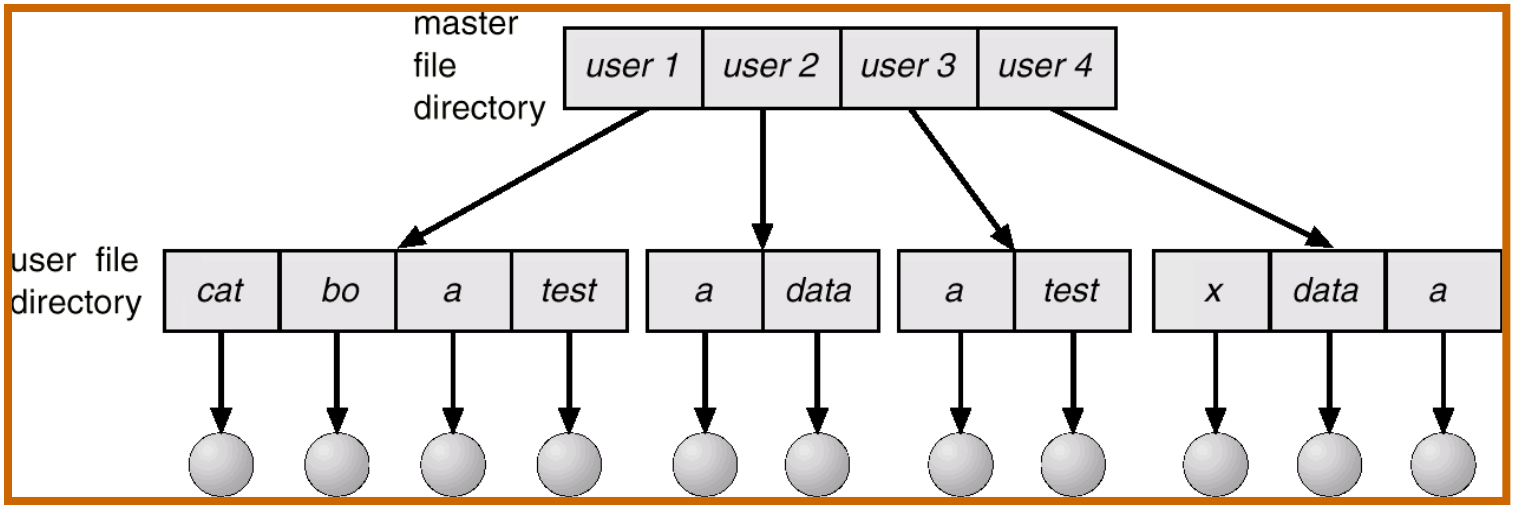


Naming problem

Grouping problem

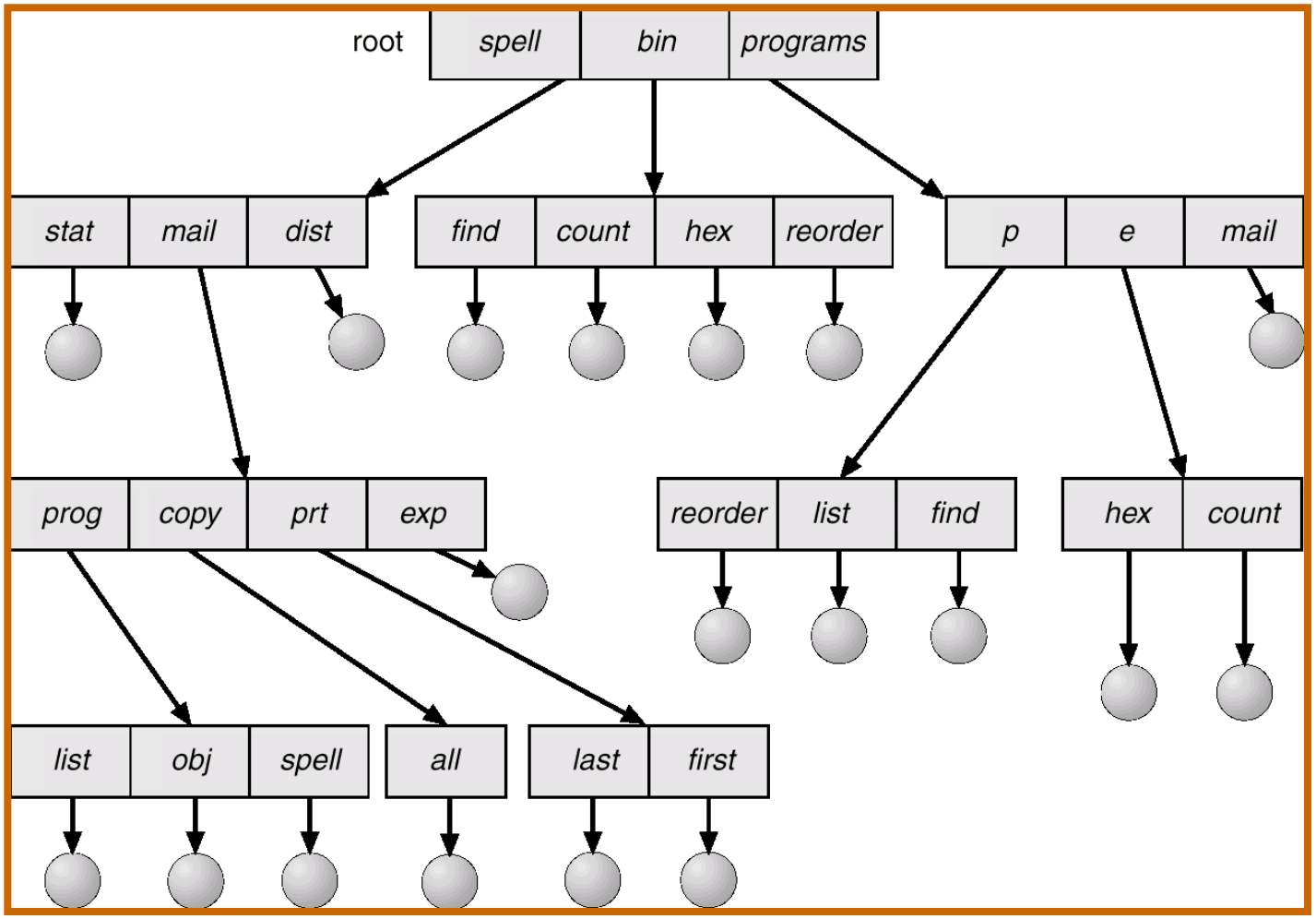
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- 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)

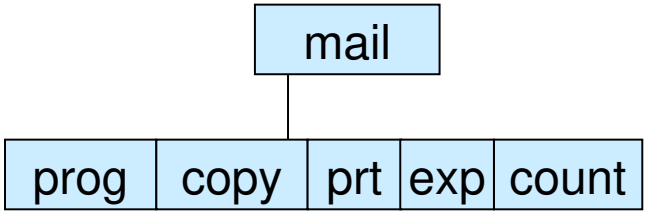
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

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

  mkdir <dir-name>

  Example: if in current directory /mail

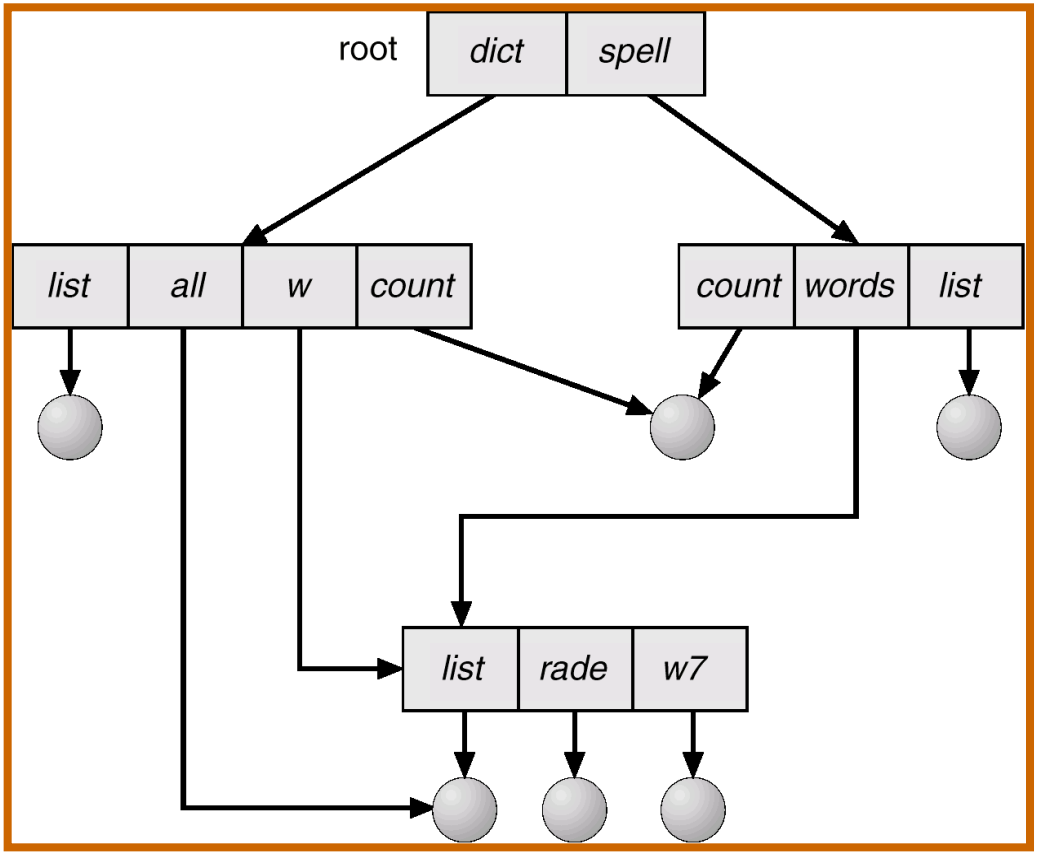
  mkdir count



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

Acyclic-Graph Directories

- Have shared subdirectories and files

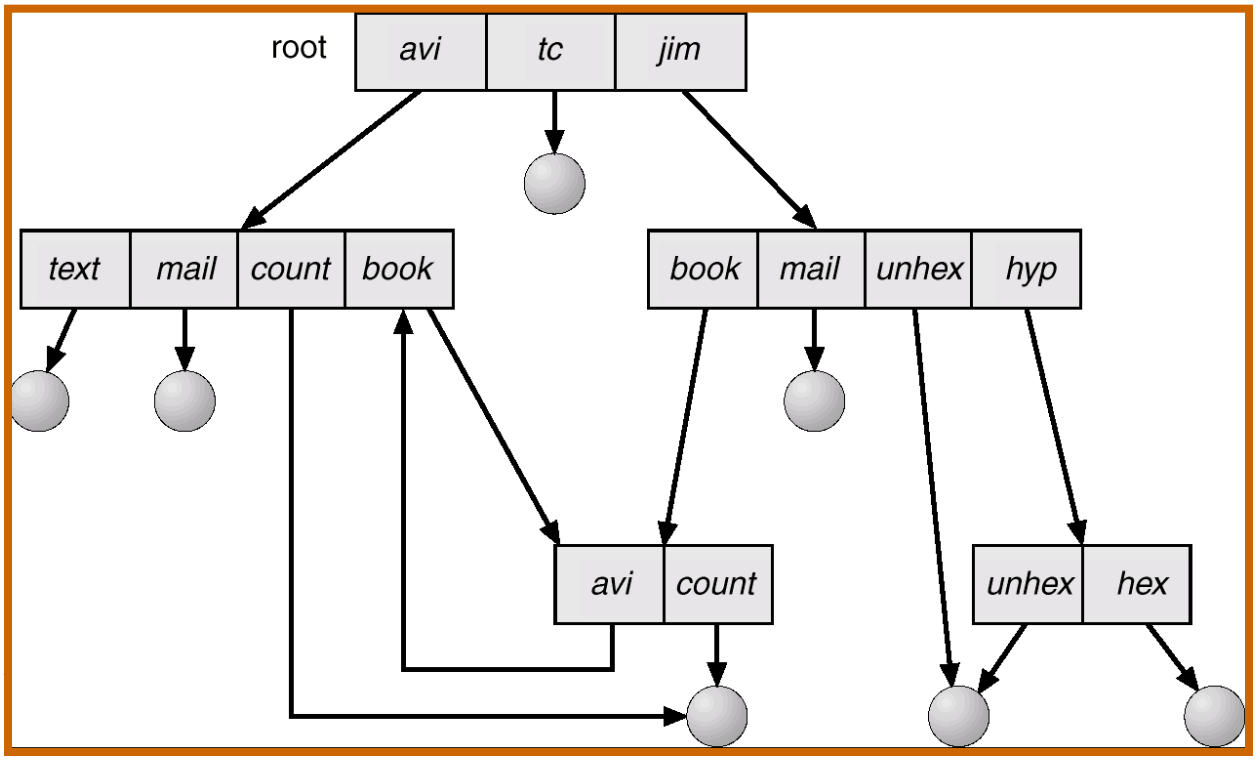


Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *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

General Graph Directory



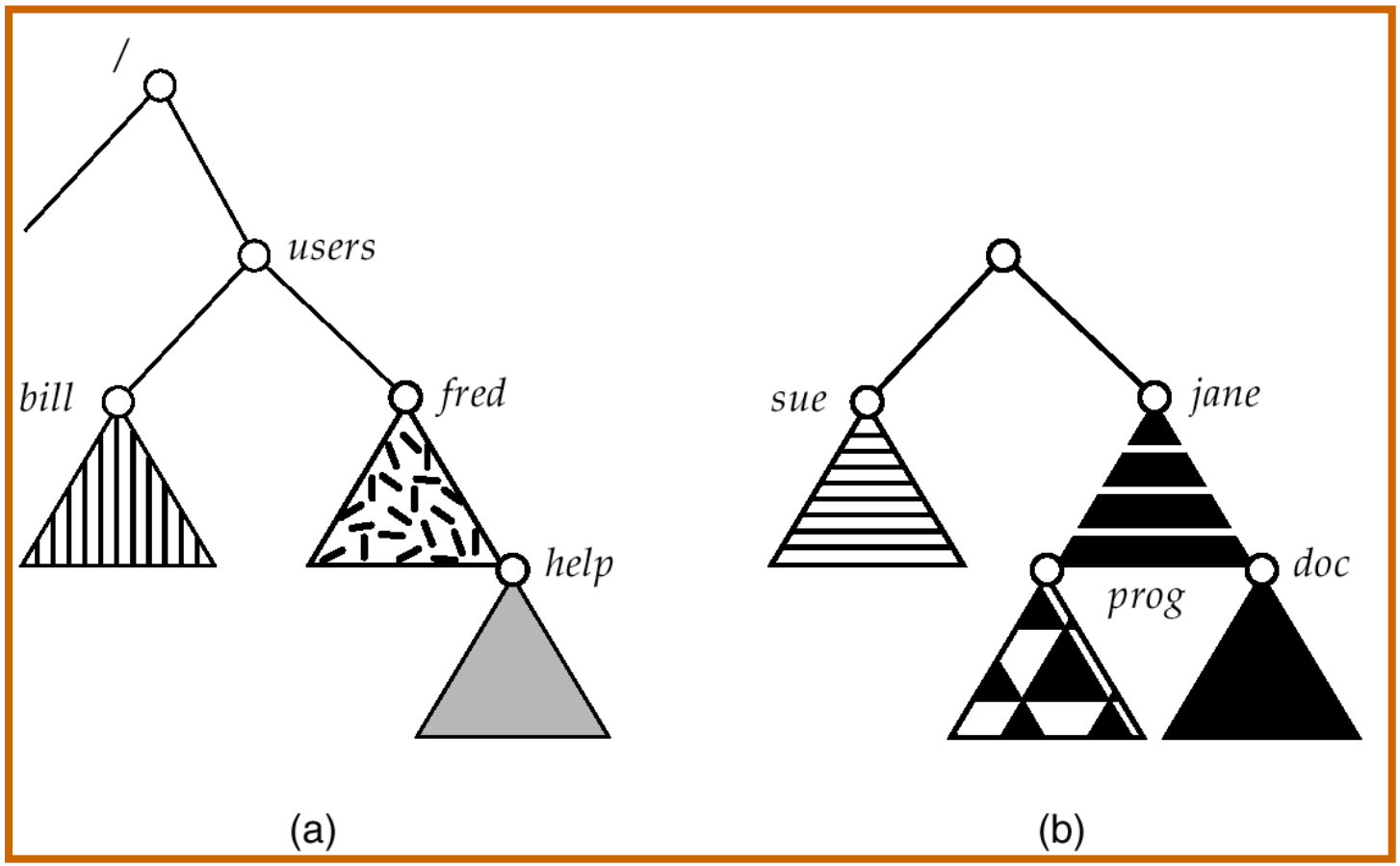
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

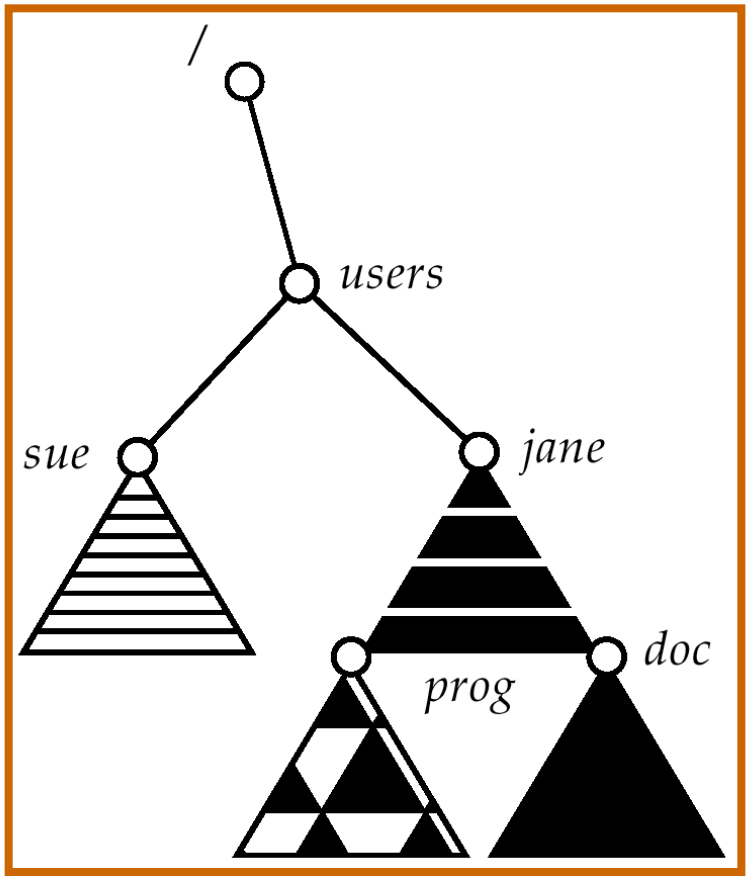
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**

(a) Existing. (b) Unmounted Partition



Mount Point



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

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** 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**
  - Semi automatically via the **world wide web**
- **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 implement unified access to information needed for remote computing

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **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 (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes to an open file by a user are not visible immediately to oher users that have the same file open.
    - Writes only visible to sessions starting after the file is closed

Protection

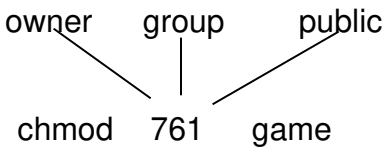
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			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 (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

```
chgrp  G  game
```