Modern Operating Systems

Chapter 4 – Files&Directories

Zhang Yang

Spring 2022

Content of the Lecture

- 4.1 Files
- 4.2 Directories

Files

- File Naming
- File Structure
- File Types
- File Access
- File Attributes
- File Operations
- An Example Program Using File System Calls

Storing Information (1)

- Applications can store it in the process address space.
- Why is it a bad idea?
  - Size is limited to size of virtual address space.
    - May not be sufficient for airline reservations, banking, etc.
  - The data is lost when the application terminates.
    - Even when computer doesn’t crash!
  - Multiple process might want to access the same data.
    - Imagine a telephone directory part of one process.

Storing Information (2)

- Three criteria for long-term information storage:
  - Should be able to store very large amount of information.
  - Information must survive the processes using it.
  - Should provide concurrent access to multiple processes.
- Solution
  - Store information on disks in units called files.
  - Files are persistent, and only owner can explicitly delete it.
  - Files are managed by the OS.

File System Overview (1)

- File Systems: How the OS manages files!

Recall that

- OS: an extended machine or virtual machine
- OS: a resource manager
  - Processor management
  - Memory management
  - I/O management
  - File system management

File System Overview (2)

- User’s view on file systems (4.1, 4.2)
  - What constitutes a file?
  - How files are named and protected?
  - What operations are allowed on them?
  - What the directory tree looks like?
  - …

File and File System

- File
  - A named collection of related information that is recorded on secondary storage.
    - Persistent through power failures and system reboots.
  - OS provides a uniform logical view of information storage via files.
  - OS maps the File to the Physical device.
    - A logical representation of how files are stored on the physical device.
      - Mapping varies from one OS to another.
- File System
  - A method for storing and organizing files and the data they contain to make it easy to find and access them.

Basic Functions of File System

- Present logical (abstract) view of files and directories.
  - Hide complexity of hardware devices.
- Facilitate efficient use of storage devices.
  - Optimize access, e.g., to disk
- Support sharing.
  - Provide protection.

File Naming (1)

- Motivation
  - Files abstract information stored on disk.
  - You do not need to remember block, sector, …
  - We have human readable names.
- Very important. A major function of the file system is to supply uniform naming.
- How does it work?
  - Process creates a file, and gives it a name.
  - Other processes can access the file by that name.

File Naming (2)

- Naming Conventions
  - OS dependent.
  - Textual Names
    - Strings of letters, digits, and special characters.
  - Number of characters
    - All current OS allow strings of 1 to 8 characters as legal file names.
    - Many OS support up to 255 characters.
  - Case Sensitive
    - UNIX family
      - E.g. three distinct files: maria, Maria, MARIA

File Naming (3)

- Naming Conventions
  - Case Insensitive
    - MS-DOS and Windows
      - E.g. the same file: maria, Maria, MARIA
- File Extension
  - The extensions are suffixes attached to the file names.
  - Usually indicate something about a file.
  - Tied to type of file.
  - Used by applications.
  - Size of extension
    - MS-DOS: 1 to 3 characters
    - Unix: up to the user.

File Naming (4)

- File Extension (ctd.)
  - In UNIX, extensions are not enforced by OS.
    - However C compiler might insist on its extensions.
      - These extensions are very useful for C.
  - Windows attaches meaning to extensions.
    - Tries to associate applications to file extensions.

File Naming (5)

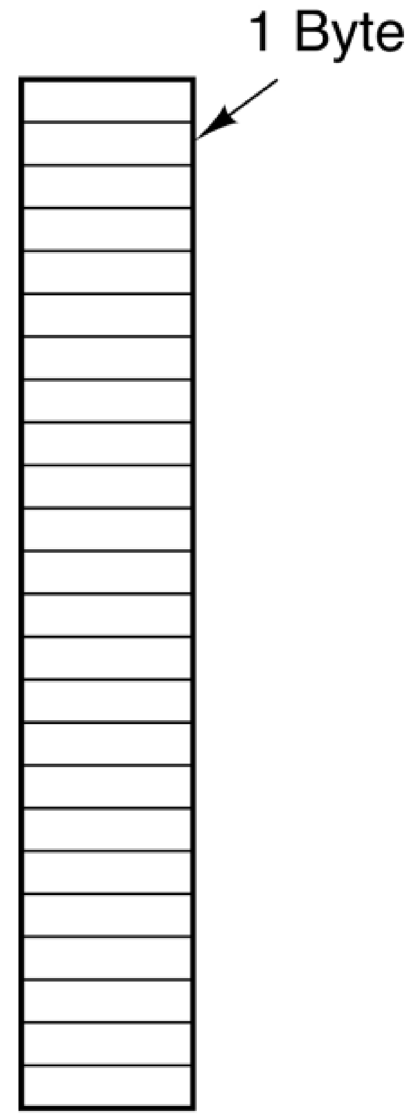
- File Extension (ctd.)
  - Typical File Extensions

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Figure 4-1 Typical file extensions.

File Structure (1)

- No structure: Byte Sequence
  - Simplifies file management for the OS.
  - Applications can impose their own structure.
  - Used by UNIX, Windows, most modern Oses.

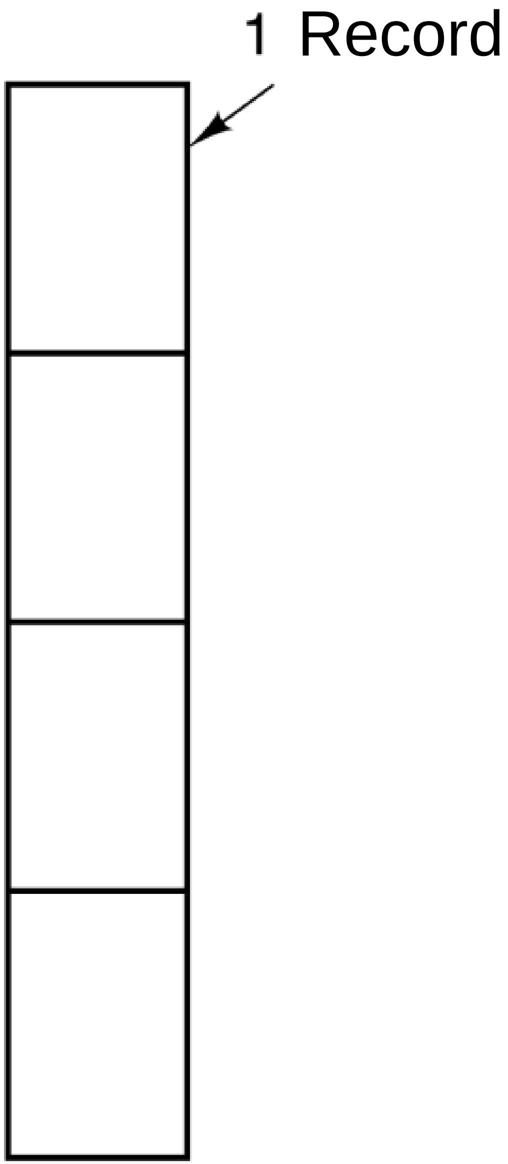


(a) Byte sequence

Figure 4-2 Three kinds of files

File Structure (2)

- Simple record structure: Record Sequence
  - Collection of bytes treated as a unit.
  - Example: employee record
  - Operations at the level of records (read_rec, write_rec)
  - File is a collection of similar records.
  - OS can optimize operations on records.



(b) Record sequence

Figure 4-2 Three kinds of files

File Structure (3)

- Complex Structures: Tree
  - Records of variable length.
  - Each has an associated key.
  - Record retrieval based on key.
  - Used on some data processing systems (mainframes)
    - Mostly incorporated into modern databases.

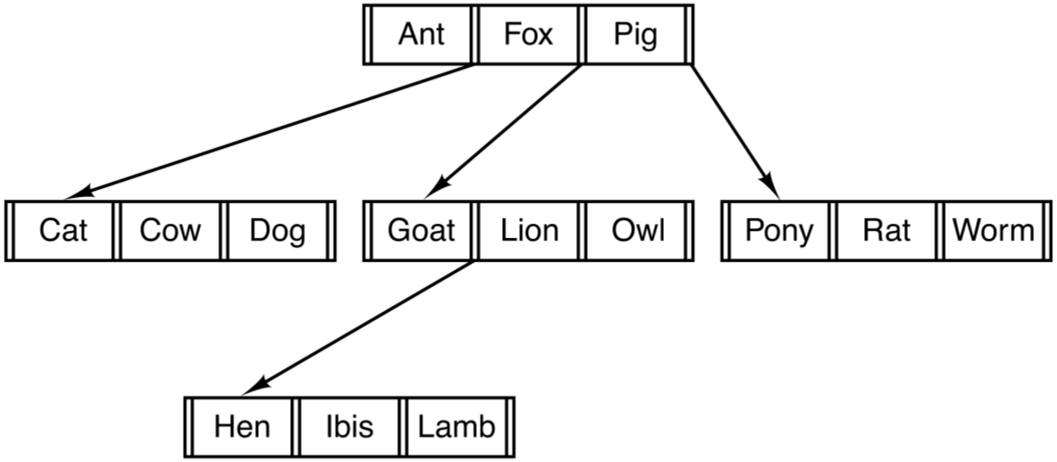


Figure 4-2 Three kinds of file (c) Tree

File Types (1)

- Files may have types.
  - Understood by file systems.
    - Device, directory, symbolic link, etc.
  - Understood by other parts of OS or runtime libraries.
    - Executable, dll, source code, object code, text, etc.
  - Understood by application programs.
    - jpg, mpg, avi, mp3, etc.
- Common File Types
  - Regular File
  - Directory
  - Character Special File (Unix)
    - Used to model serial I/O devices, such as terminals, printers, etc.
  - Block Special File (Unix)
    - Used to model disks.

File Types (2)

- Regular Files
  - Regular files are the ones that contain user information.
  - ASCII Files
    - A text file in which each byte represents one character according to the ASCII code.
    - Consist of lines of text.
    - Lines need not all be of the same length.
    - Advantages
      - Can be displayed, printed, edited with any text editor.
      - Facilitate information exchange.

File Types (3)

- Regular Files (ctd.)
  - Binary Files
    - A binary file is a file whose content must be interpreted by a program or a hardware processor that understands in advance exactly how it is formatted.
    - Have internal structure.
    - Example: An Unix executable file
      - Header
      - Text
      - Data
      - Relocation bits
      - Symbol table

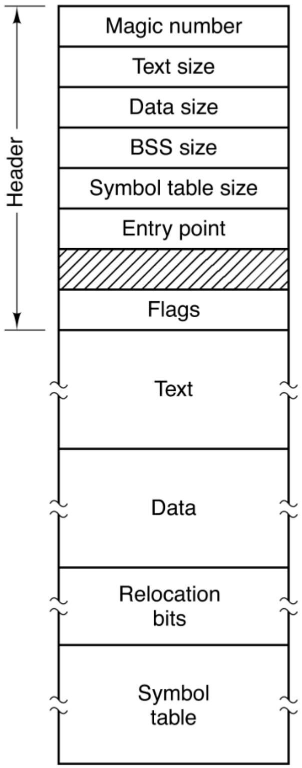


Figure 4-3 (a) An executable file

File Types (4)

- Regular Files (ctd.)
  - Binary Files (ctd.)
    - Example: An Unix archive file
      - Consists of a collection of library procedures compiled but not linked.

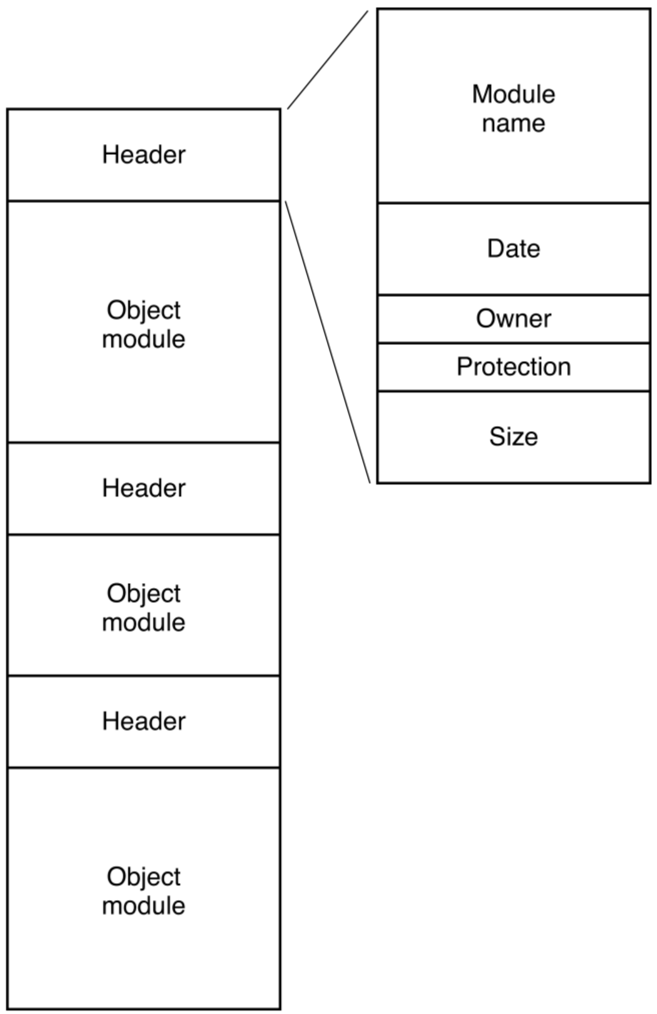
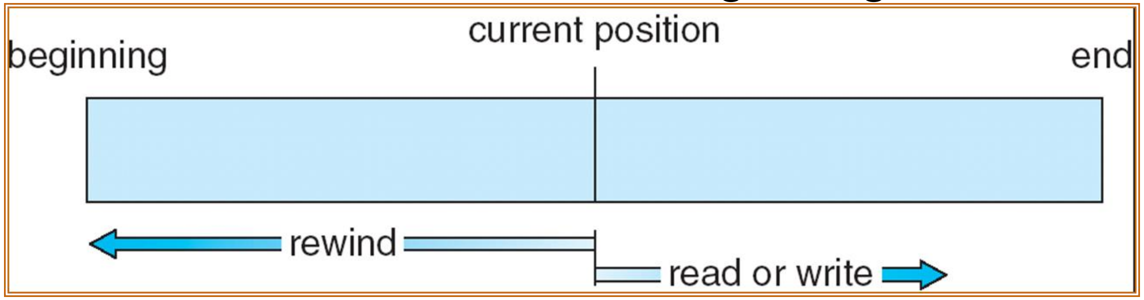


Figure 4-3 (b) An archive

File Access (1)

- Sequential Access
  - Read all bytes/records from the beginning.
  - Cannot jump around, could rewind or back up.
  - Convenient when medium was magnetic tape.
  - Read Operation (read next) – reads next portion of file and advances the file pointer.
  - Write Operation (write next) – appends to end of file (eof) and advances file pointer to the new eof.
  - File Pointer tracks I/O location
    - Allows file to be reset at the beginning.



File Access (2)

- Random Access
  - Bytes/records read in any order.
  - Essential for database systems.
  - Read can be …
    - Move file marker (seek), then read or …(Unix and Windows).
    - Read and then move file marker.
  - Read Operation (read n): "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.
  - Write Operation (write n)
  - Convenient when medium was magnetic disk.
  - Random Access File
    - The file is viewed as a numbered sequence of blocks or records.

File Access (3)

- Question
  - Systems that support sequential files always have an operation to rewind files. Do systems that support random-access files need this, too?

File Attributes (1)

- Dependent on OS
  - *Name* – Symbolic file name is only information kept in human-readable form.
  - *Type* – needed for systems that support different types of files.
  - *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 Attributes (2)

- Possible File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4 Possible File Attributes

File Operations (1)

- Operations for “sequence of bytes” files
  - Create: Announce that file is coming and set attributes and allocate space.
  - Delete: Free disk space, adjust directory structure.
  - Open: Allow the system to fetch the attributes and list of disk addresses into main memory.
  - Close: Release internal table space and writing the file’s last block.
  - Read: Data read from the file and put into memory for user access.
  - Write: Data are written to the file usually at the current position.

File Operations (2)

- Operations for “sequence of bytes” files (ctd.)
  - Append: Adds data to the end of file.
  - Seek: Random access data from the file, repositioning the file pointer for reading.
  - Rename: Change the name of the file.
  - Get & Set Attributes: Get attributes of file or set attributes of a file (e.g., get and set read only attribute)
  - A few more on directories: talk about this later

File Related System Calls

- fd = open (name, mode)
- byte_count = read (fd, buffer, buffer_size)
- byte_count = write (fd, buffer, num_bytes)
- close (fd)

An Example Program Using File System Calls (1)

- copyfile Program
  - E.g. copyfile abc xyz (copy the file abc to xyz)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                     /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);          /* ANSI prototype */

#define BUF_SIZE 4096                      /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                   /* protection bits for output file */

int main(int argc, char *argv[])
{
     int in_fd, out_fd, rd_count, wt_count;
     char buffer[BUF_SIZE];

     if (argc != 3) exit(1);               /* syntax error if argc is not 3 */
```

argc: how many strings were presented

argv: an array of pointers to the arguments

An Example Program Using File System Calls (2)

- copyfile Program (ctd.)

```
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);      /* open the source file */
if (in_fd < 0) exit(2);               /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
if (out_fd < 0) exit(3);              /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
     rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;             /* if end of file or error, exit loop */
     wt_count = write(out_fd, buffer, rd_count); /* write data */
     if (wt_count <= 0) exit(4);      /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0)                    /* no error on last read */
     exit(0);
else
     exit(5);                         /* error on last read */
}
```

Exercise (1)

- Some operating systems provide a system call *rename* to give a file a new name. Is there any difference at all between using this call to rename a file and just copying the file to a new file with the new name, followed by deleting the old one?

Exercise (2)

- Solution
  - Yes.
  - The *rename* call does not change the creation time or the time of last modification, but creating a new file causes it to get the current time as both the creation time and the time of last modification.
  - Also, if the disk is full, the copy might fail.

Directory

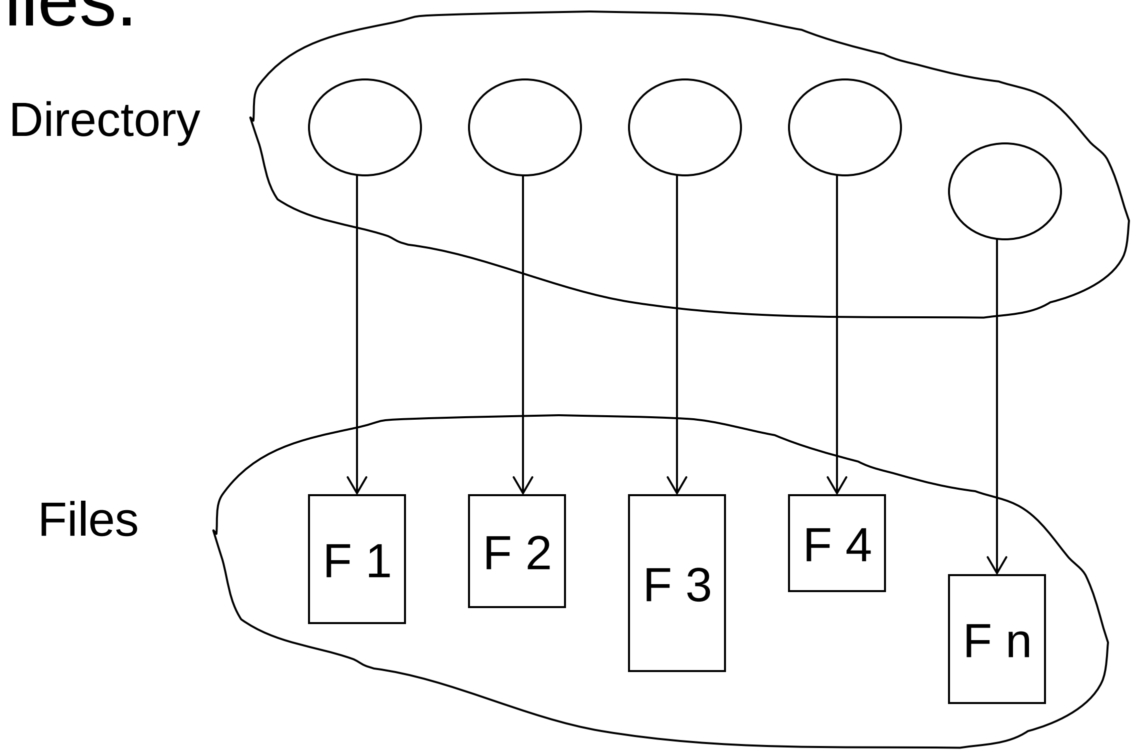
- Directory Structures
  - Single-level Directory System
  - Two-level Directory System
  - Hierarchical Directory System
- Path Names
- Directory Operations

Directory (1)

- Directory is a file containing correspondence between filenames and file locations.
- Directory entries contain infomation about a file.
  - i.e. its attributes
- Directory entries are created when the files they describe are created and removed when files are deleted.

Directory (2)

- 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.

Directory Contents

- File name: symbolic name
- File type: indicates format of file
- Location device and location
- Size
- Protection
- Creation, access, and modification date
- Owner identification

Organize the Directory (Logically) to Obtain

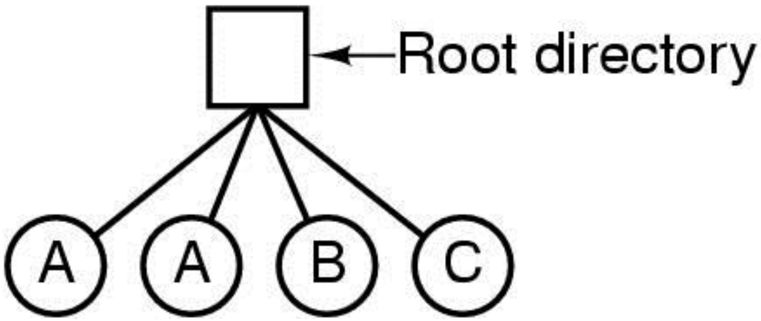
- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files.
  - The same file can have several different names.
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Directory Structures

- Single-Level Directory System
- Two-Level Directory System
- Hierarchical Directory System

Single-Level Directory Systems

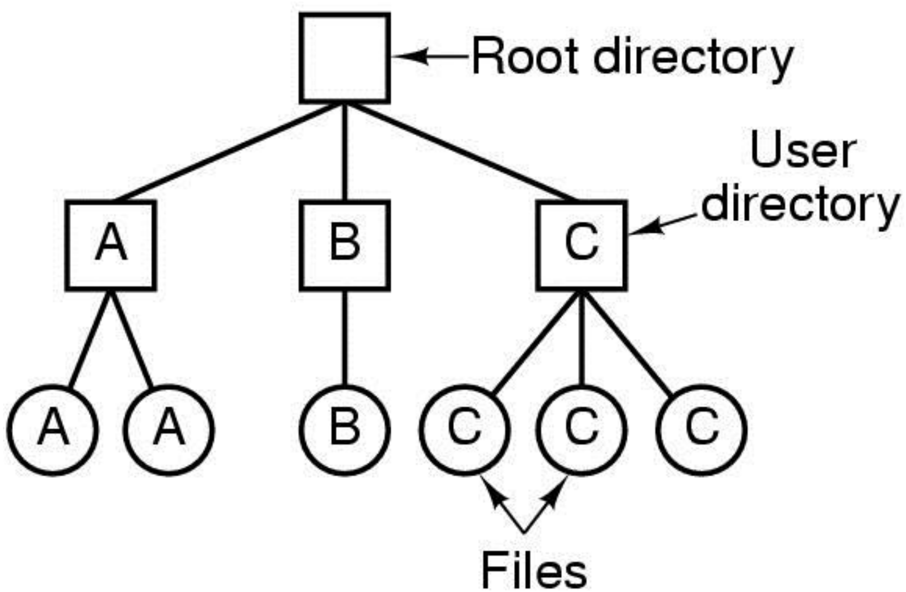
- A single directory for all users.
- Example: A single level directory system
  - Contains 4 files.
  - Owned by A, B, and C.



- Advantage
  - Easy to support and understand.
- Problems
  - Naming Problem (Name Collision Problem)
    - All files must have unique names.
  - Grouping Problem
    - Difficult to remember file names in a large file system.

Two-level Directory Systems (1)

- Separate directory for each user.
- File names only need to be unique within a given user's directory.
- A separate directory is generally needed for system (executable) files.
- A user name and a file name define a path name.
- Example



Letters indicate *owners* of the directories and files

Two-level Directory Systems (2)

- Advantages
  - Resolves name-collision problem
    - Can have the same file name for different user.
  - Efficient searching.
  - File sharing and protection.
- Disadvantage
  - Grouping problem not resolved.

Hierarchical Directory Systems (1)

- Generalization of two-level directory (with arbitrary height).
- Leaf nodes are files.
- Interior nodes are directories.
- Each user has a current directory (working directory).
  - Can change current directory via cd command or system call.
- Path names can be absolute or relative.

Hierarchical Directory Systems (2)

- Example

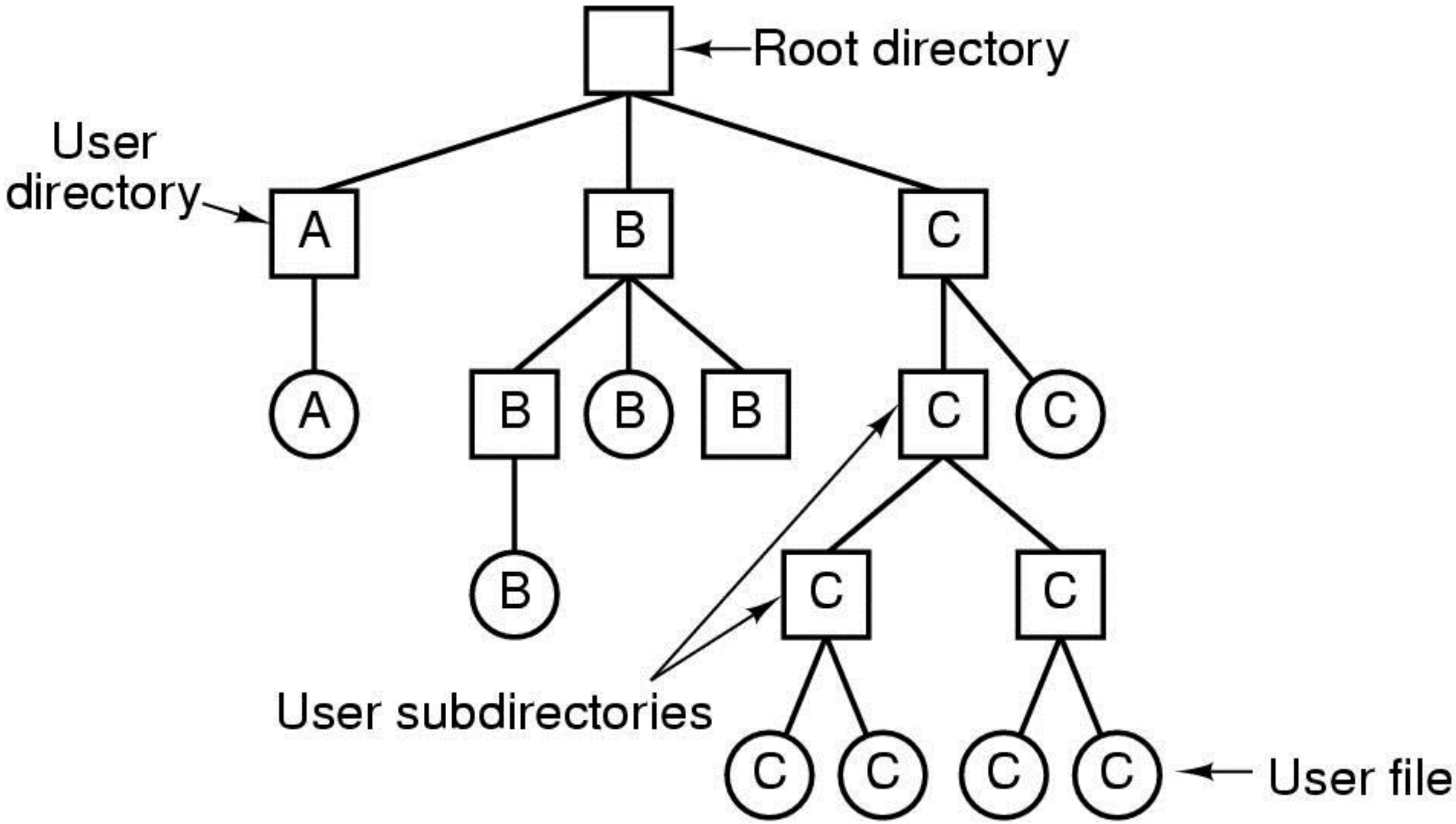


Figure 4-7 A hierarchical directory system

Hierarchical Directory Systems (3)

- Advantages
  - Resolves name-collision problem.
    - Can have the same file name for different user.
  - Efficient searching.
  - File sharing and protection.
  - Grouping capability.

Path Names (1)

- MULTICS

  >usr>harry>mailbox
- Windows

  \usr\harry\mailbox
- Unix

  /usr/harry/mailbox
- Absolute Path Name

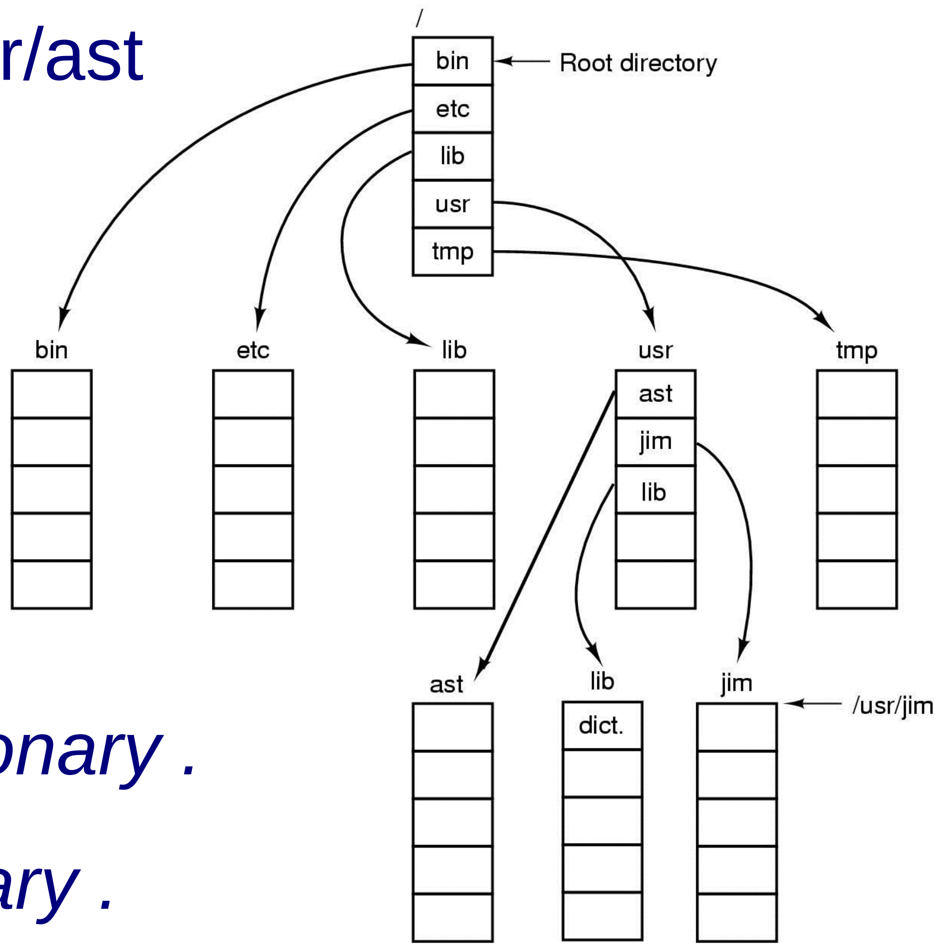
  /usr/harry/mailbox
  - Unique
  - Start from root.
- Relative Path Name
  - working directory (or current directory)

Each process has its own working directory

Path Names (2)

. is the "current directory"
.. is the parent

working dir /usr/ast



Example

cp /usr/lib/dictionary .

cp ../lib/dictionary .

Figure 4-8 A UNIX Directory Tree

Directory Operations

- Directory Operations including:
  - Create a new directory.
  - Delete a directory.
  - Open a directory for reading.
  - Close a directory to free up internal table space.
  - Readdir - return next entry in the directory.
    - Returns the entry in a standard format, regardless of the internal representation.
  - Rename a directory.
  - Link
    - Create a link from the existing file to a path name. (ie. Make a "hard link".)
  - Unlink
    - Remove a "hard link".

Unix Directory Related Syscalls

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

- s = error code
- dir = directory stream
- dirent = directory entry

Linux Directory Structure (1)

- Linux file system is based on a tree hirarchy.
- The Linux filesystem structure has evolved from Unix file structure.
- The file structure has been somewhat convoluted over the years by different flavors of Unix.
- There are few common directory names that are used for common functions.

Linux Directory Structure (2)

- Linux System Directory