# File System Interface

Operating Systems
Wenbo Shen

Acknowledgement: some pages are based on the slides from Yajin Zhou@ZJU

# Review

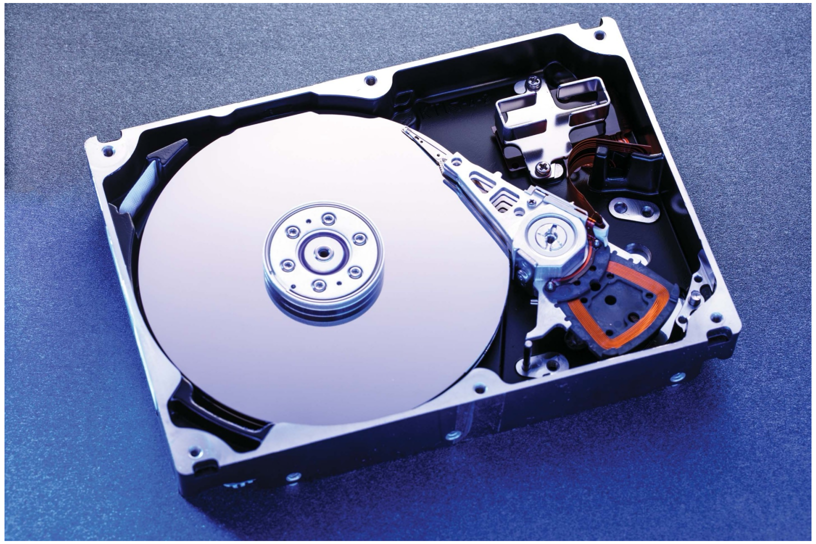
- Mass storage
- Disk structure
- Disk scheduling
  - FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK
- IO hardware
- IO access
  - polling, interrupt
- Device types
- Kernel IO subsystem

# Outline

- File concept
- Access methods
- Directory structure
- Protection

# How to use storage

- Now we have mass storage and IO
    - But how to use?
    - Think as a computer scientist in 1950s
    - Use disk directly



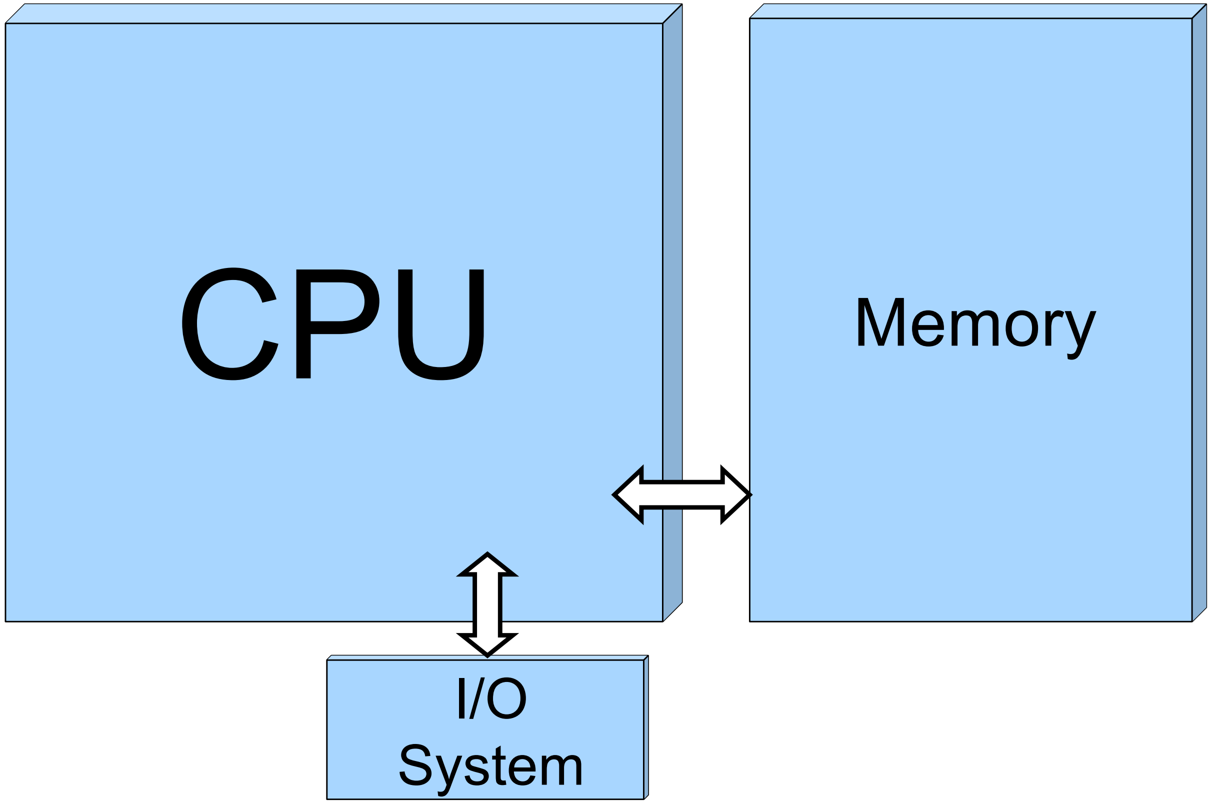
# How to use storage

- Now we have mass storage and IO
  - But how to use?

- **File System**
  - File system vs. Disk
    - File system presents abstraction of disk
    - File → Track/sector
  - To user process
    - File system provides coherent view of a group of files
    - File: a contiguous block of bytes (Unix)
  - File system provides protection

# Abstraction

- CPU is abstracted to _____.
- Memory is abstracted to _____.
- Storage is abstracted to _____.



# Inquiry-based Learning

- How to use file system?
    - How to use file?
    - How to use directory?
- How to implement file system?
    - How to implement file?
    - How to implement directory?

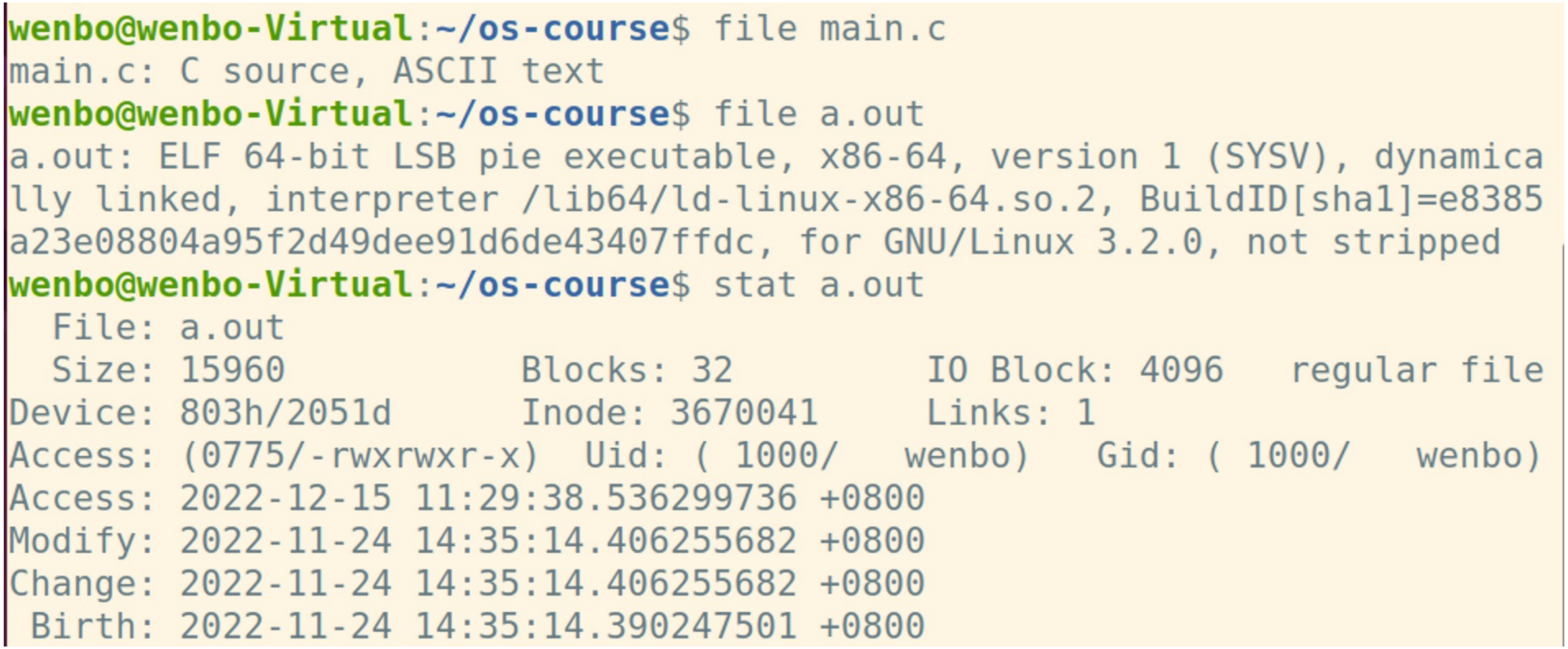
# File Concept

- **File** is a contiguous logical space for storing information
  - database, audio, video, web pages…
- There are different types of file:
  - data: character, binary, and application-specific
  - program
  - special one: proc file system - use file-system interface to retrieve system information

# File Attributes

- **Name** – only information kept in **human-readable form**
- **Identifier** – unique tag (number) identifies file **within file system**
- **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
- Many variations, including **extended file attributes** such as file checksum

# File info on Linux



```
wenbo@wenbo-Virtual:~/os-course$ file main.c
main.c: C source, ASCII text
wenbo@wenbo-Virtual:~/os-course$ file a.out
a.out: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=e8385
a23e08804a95f2d49dee91d6de43407ffdc, for GNU/Linux 3.2.0, not stripped
wenbo@wenbo-Virtual:~/os-course$ stat a.out
  File: a.out
  Size: 15960           Blocks: 32         IO Block: 4096   regular file
Device: 803h/2051d      Inode: 3670041     Links: 1
Access: (0775/-rwxrwxr-x)  Uid: ( 1000/   wenbo)   Gid: ( 1000/   wenbo)
Access: 2022-12-15 11:29:38.536299736 +0800
Modify: 2022-11-24 14:35:14.406255682 +0800
Change: 2022-11-24 14:35:14.406255682 +0800
 Birth: 2022-11-24 14:35:14.390247501 +0800
```

# File Operations

- OS provides file operations to
  - create:
    - space in the file system should be found
    - an entry must be allocated in the directory
  - open: most operations need to file to be opened first
    - return a handler for other operations
  - read/write: need to maintain a **pointer**
  - reposition within file – seek
  - close
  - delete
    - Release file space
    - Hardlink: maintain a counter - delete the file until the last link is deleted
  - truncate: empty a file but maintains its attributes
- Other operations can be implemented using these ones
  - Copying: create and read/write

# Open Files

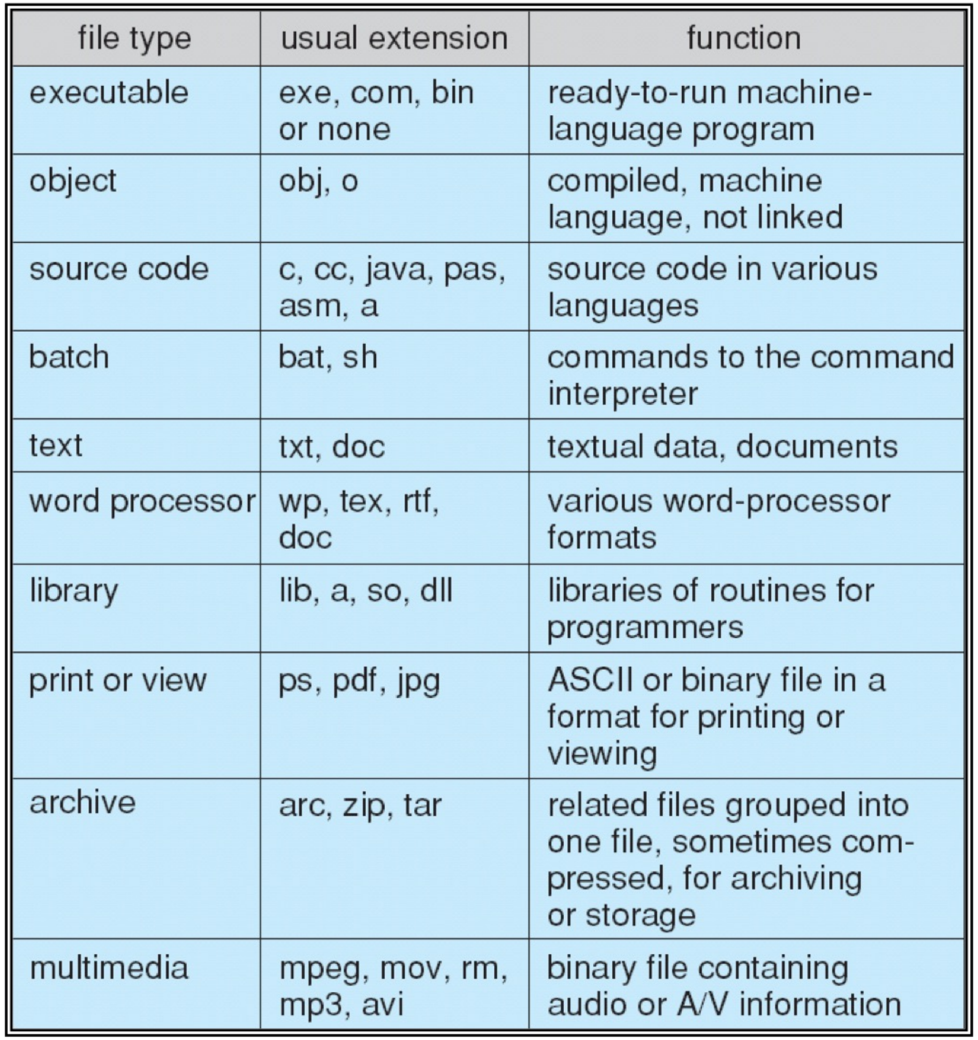
- Several data are needed to manage open files:
  - Open-file table: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

# Open Files

- Some file systems provide file lock to mediates access to a file
- Two types of lock
  - **Shared lock** - multiple processes can acquire the lock concurrently
  - **Exclusive lock** - one process can acquire such an lock
- Two locking mechanisms
  - **mandatory lock**: access is denied depending on locks held and requested
  - **advisory lock**: processes can find status of locks and decide what to do

# File Types

- as part of the file names - file extension
- magic number of the file - elf



| 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

- A file can have different structures, determined by OS or program
  - **No structure**: a stream of bytes or words
    - Linux dumps
  - **Simple record structure**
    - Lines of records, fixed length or variable length
    - E.G., Database
  - **Complex structures**
    - E.G., Word document, relocatable program file
- Usually user programs are responsible for identifying file structure

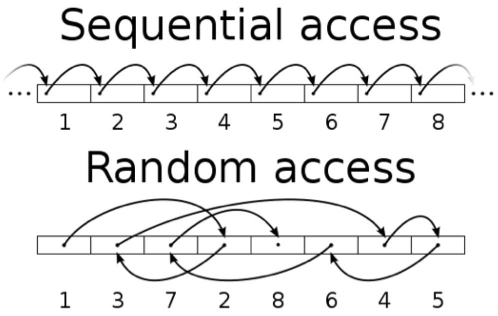
# Outline

- File concept
- Access methods
- Directory structure
- Protection

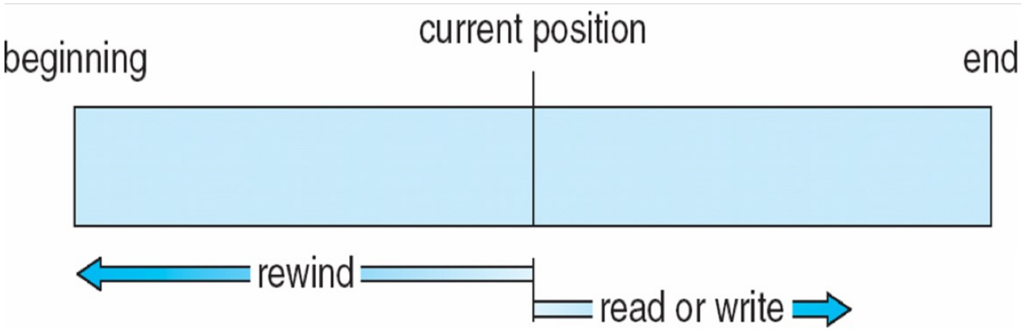
# Access Methods

- Sequential access
  - a group of elements is access **in a predetermined order**
  - for some media types, the only access mode (e.g., **tape**)

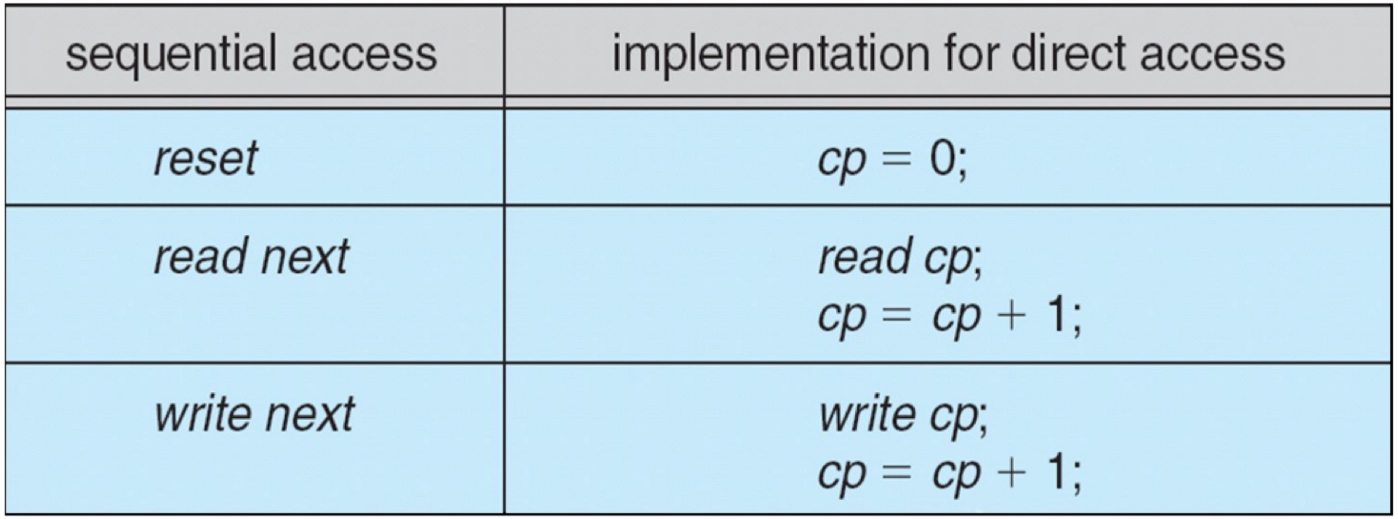
- Direct access
  - access an element at an **arbitrary position** in a sequence in (roughly) **equal time**, independent of sequence size
    - it is possible to emulate random access in a tape, but access time varies
  - sometime called random access



# Sequential-access File



# 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; |

# Other methods

- Based on direct-access method
- An index for the file points to blocks
    - Find a record in the file, first search the index and then use the pointer to access the block
    - We may use multiple layers of index

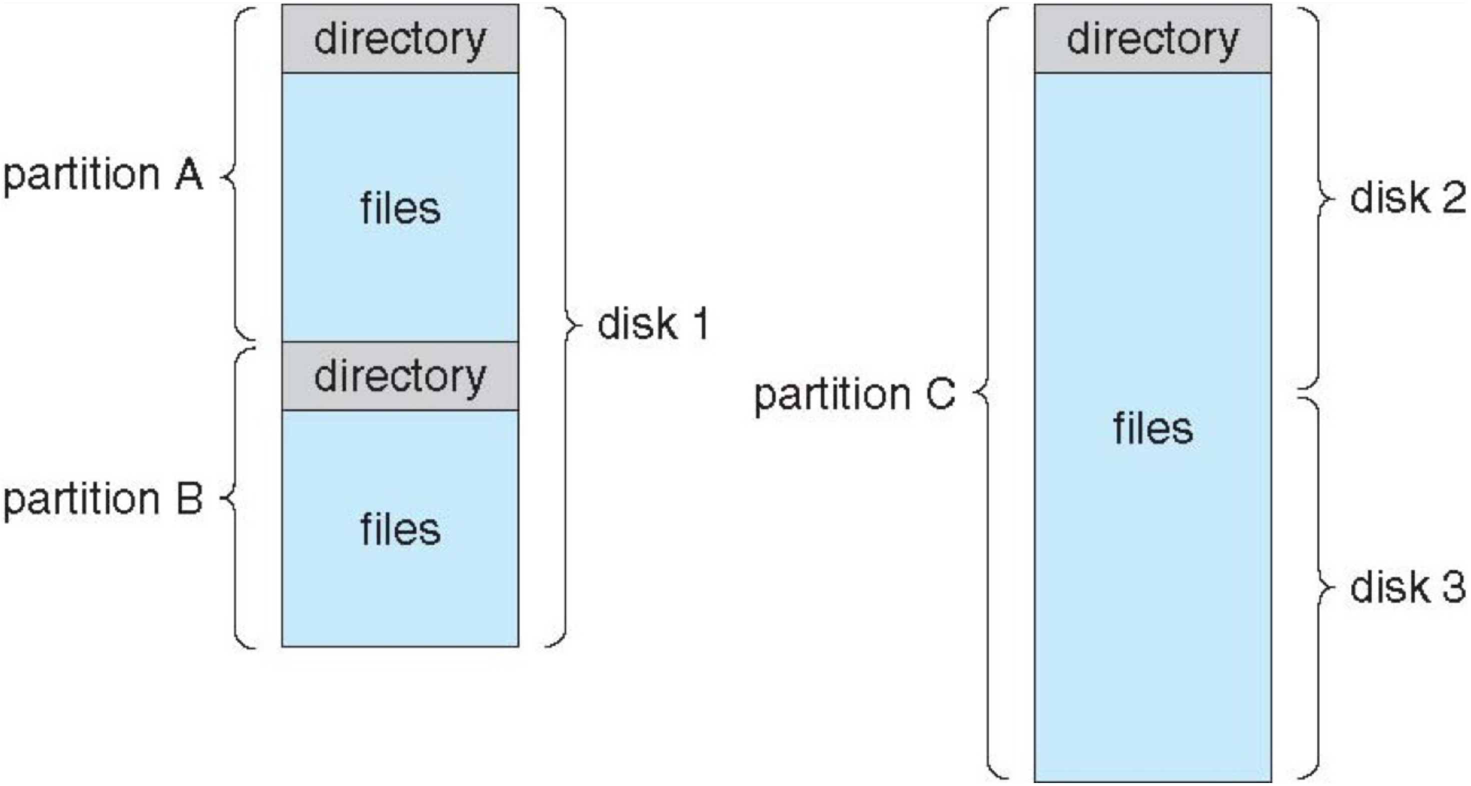
# Outline

- File concept
- Access methods
- Directory structure
- Protection

# Disk and file system

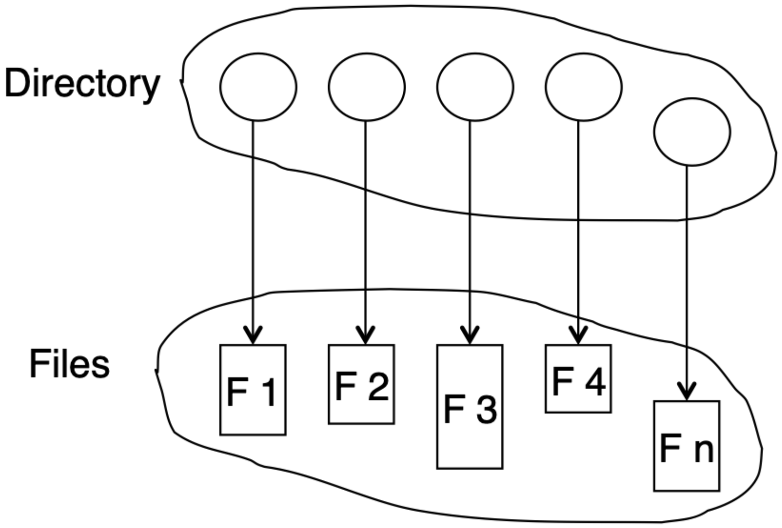
- Disk can be subdivided into **partitions**
  - partitions also known as **minidisks**, **slices**
  - different partitions can have different file systems
    - a partition containing file system is known as a **volume**
    - each volume tracks file system info in the volume's table of contents
    - a file system can be general purpose or special purpose
  - disk or partition can be used **raw** (without a file system)
    - applications such as database prefer raw disks

# A Typical File-system Organization



# Directory Structure

- Directory is a collection of nodes containing information about all files



Both the directory structure and the files reside on disk

# Operations Performed on Directory

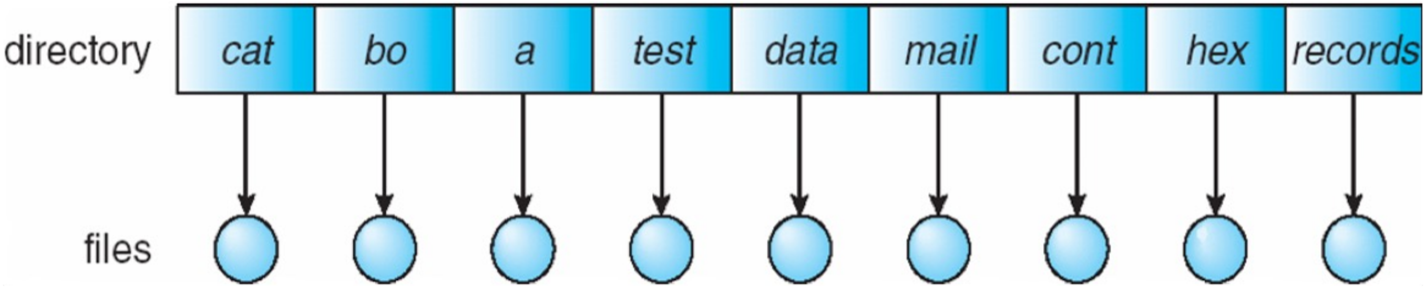
- Create a file: new files need to be created and added to directory
- delete a file: remove a file from directory
- List a directory: list all files in directory
- Search for a file: pattern matching
- Traverse the file system: access every directory and file within a directory
- …

# Directory Organization

- Organize directories to achieve
  - **Efficiency**: to locate a file quickly
  - **Naming**: organize the directory structure to be convenient to users
    - Two users can have same name for different files
    - The same file can have several different names
  - …

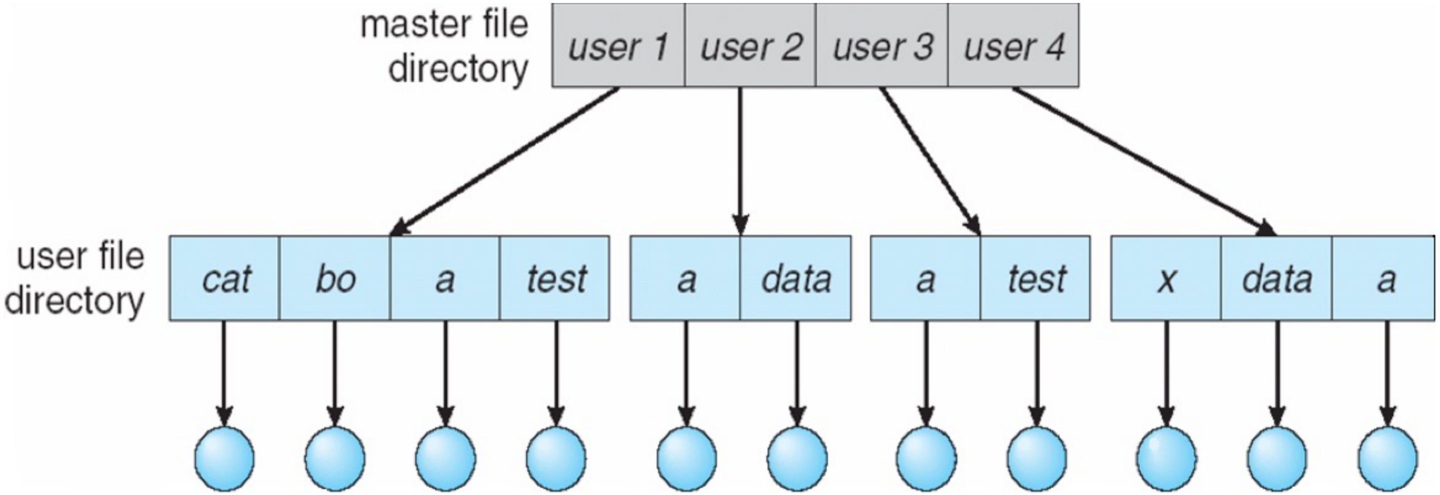
# Single-Level Directory

- A single directory for all users
  - Naming problems and grouping problems
    - Two users want to have same file names



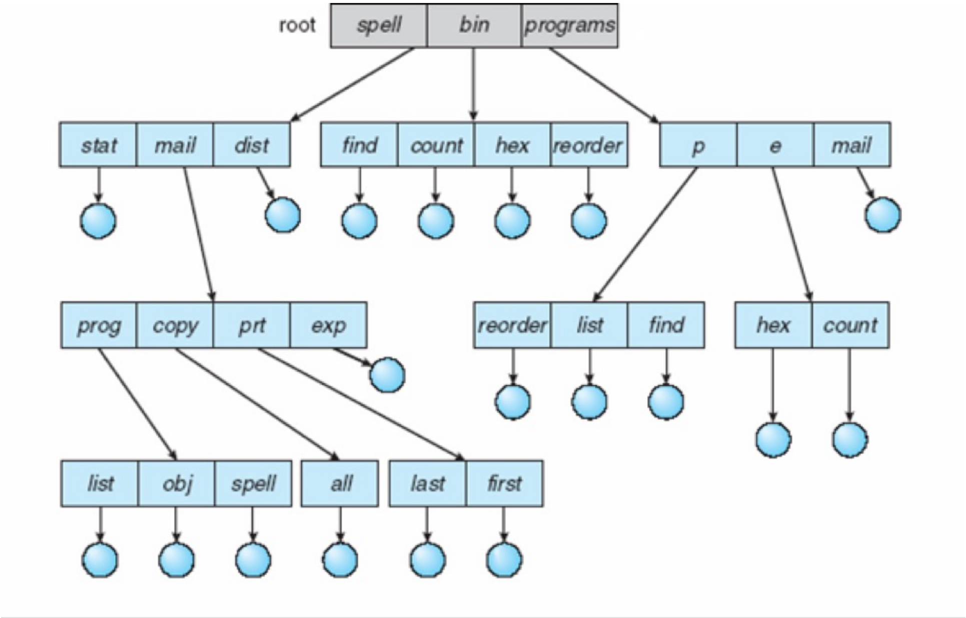
# Two-Level Directory

- Separate directory for each user
  - Different user can have the same name for different files
    - Each user has his own user file directory (UFD), it is in the master file directory (MFD)
  - Efficient to search
  - How to share files between different users?
    - Path concept



# Tree-Structured Directories

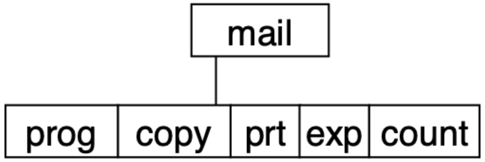
- Files organized into trees
  - efficient in searching, can group files, convenient naming



# Tree-Structured Directories

- File can be accessed using **absolute** or **relative** path name
  - absolute path name: /home/alice/..
  - relative path is relative to the **current directory** (*pwd*)
- Creating a new file: *touch <file-name>*
- Delete a file: *rm <file-name>*
- Creating a new subdirectory
  - Example: if in current directory /mail

    *mkdir count*



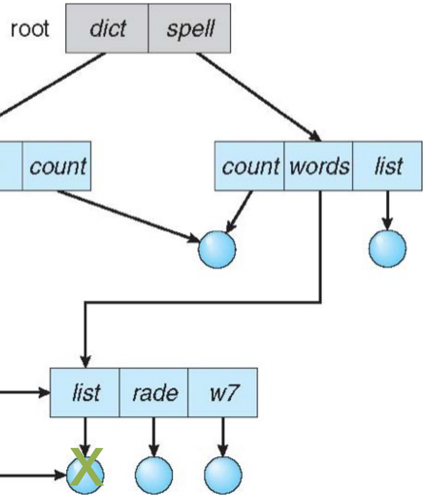
    - e.g., if current directory is /mail, a *mkdir* command will create /mail/count
  - How to share a file/directory? -> it’s not allowed

# Tree-Structured Directories

- Delete directory
    - If directory is empty, then it’s easy to handle
    - If not
        - Option I: directory cannot be deleted, unless it’s empty
        - Option II: delete all the files, directories and sub-directories
        - sudo rm -rf /

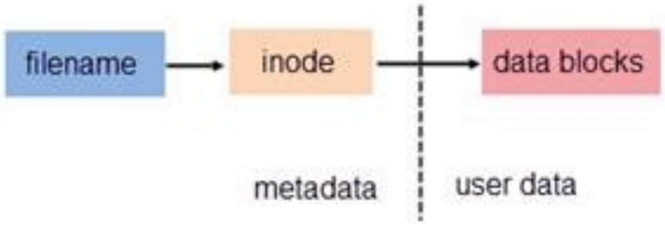
# Acyclic-Graph Directories

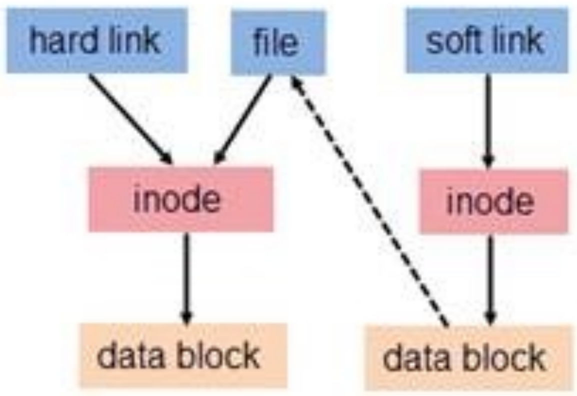
- Organize directories into acyclic-graphs
  - allow links to a directory entry/files for **aliasing** (no longer a tree)
- Dangling pointer problem:
  - e.g., if delete **file** /dict/all, /dict/w/list and /spell/words/list are dangling pointers
  - Solution: **back pointers/reference counter**
    - Back pointers record all the pointers to the entity, a variable size record
    - Or count # of links to it and only (physically) delete it when counter is zero



# Acyclic-Graph Directories

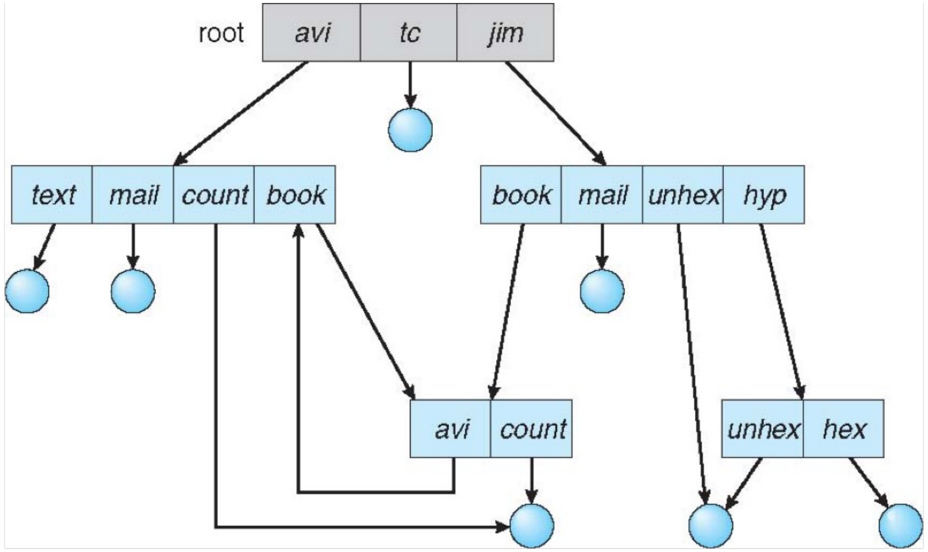
- Share files
  - Hardlink
    - Reference count
  - Softlink





# General Graph Directory

- Allowing arbitrary links may generate cycles in the directory structure
- Solution
  - allow cycles, but use **garbage collection** to reclaim disk spaces
  - every time a new link is added use a **cycle detection** algorithm

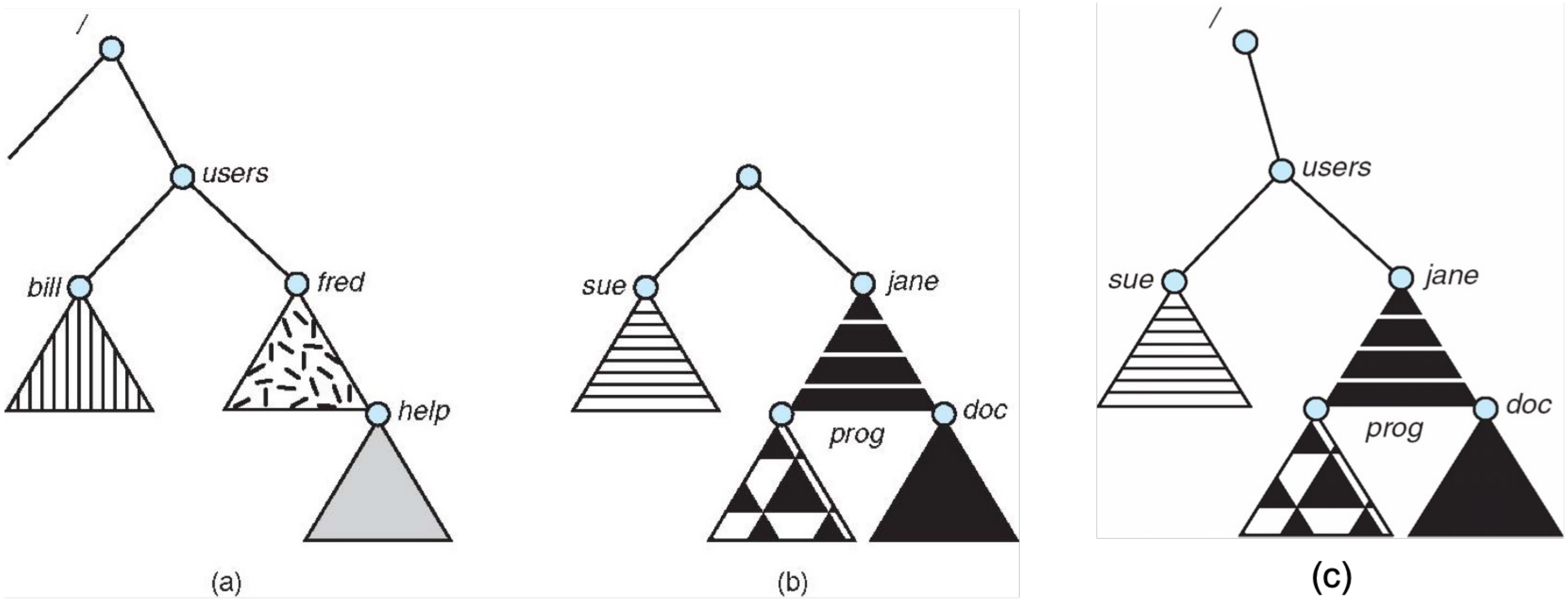


# File System Mounting

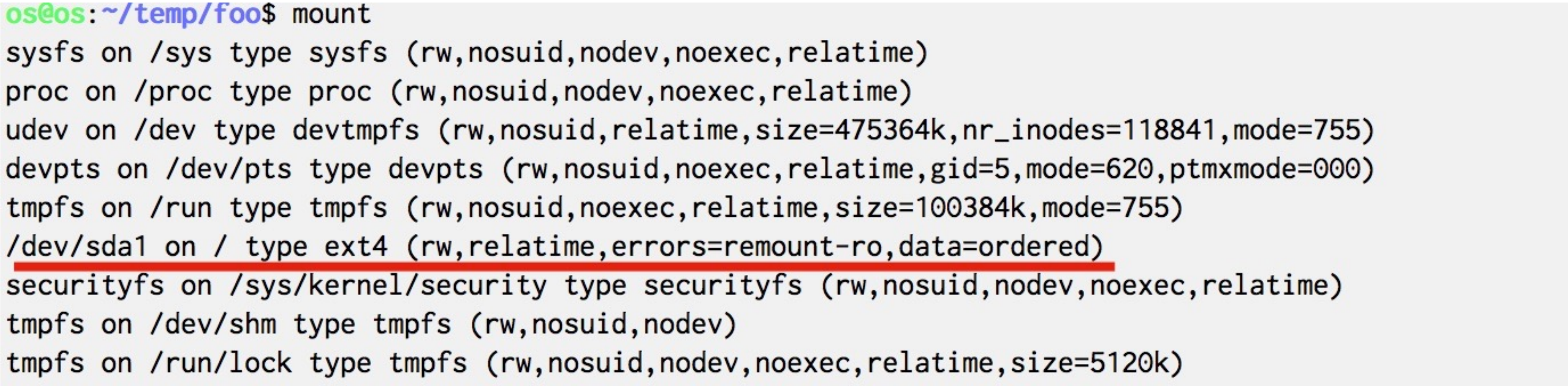
- A file system must be **mounted** before it can be accessed
  - mounting links a file system to the system, usually forms a **single name space**
  - the location of the file system being mounted is call the **mount point**
  - a mounted file system makes the old directory at the mount point **invisible**

# File System Mounting

- **a**: existing file system
- **b**: an unmounted partition
- **c**: the partition mounted at **/users**



# Mounting a file system



```
os@os:~/temp/foo$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=475364k,nr_inodes=118841,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=100384k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
```

# File Sharing

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

# Remote File Sharing

- **Use 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 FS from servers
  - a server can serve multiple clients
  - client and user-on-client identification is complicated
    - server cannot assume the client is trusted
    - standard OS file calls are translated into remote calls
  - **NFS** is standard UNIX file sharing protocol, **CIFS** is standard for Windows

# Outline

- File concept
- Access methods
- Directory structure
- Protection

# Protection

- File owner/creator should be able to control
  - what can be done
  - by whom
- Types of access
  - read, write, append
  - execute
  - delete
  - list

# ACL

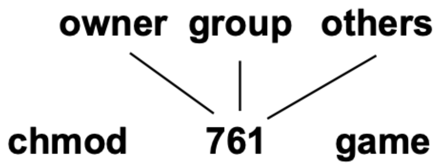
- Assign each file and directory with an access control list (ACL)
- Advantages: fine-grained control
- Disadvantages
  - How to construct the list
  - How to store the list in directory

# Unix Access Control

- Three modes of access: **read**, **write**, **execute** (encoded in three bits)
- Three classes of users: **owner**, **group**, and **others**

| | | RWX |
|---|---|---|
| a) owner access: | 7 | 1 1 1 |
| b) group access: | 6 | 1 1 0 |
| c) others access: | 1 | 0 0 1 |

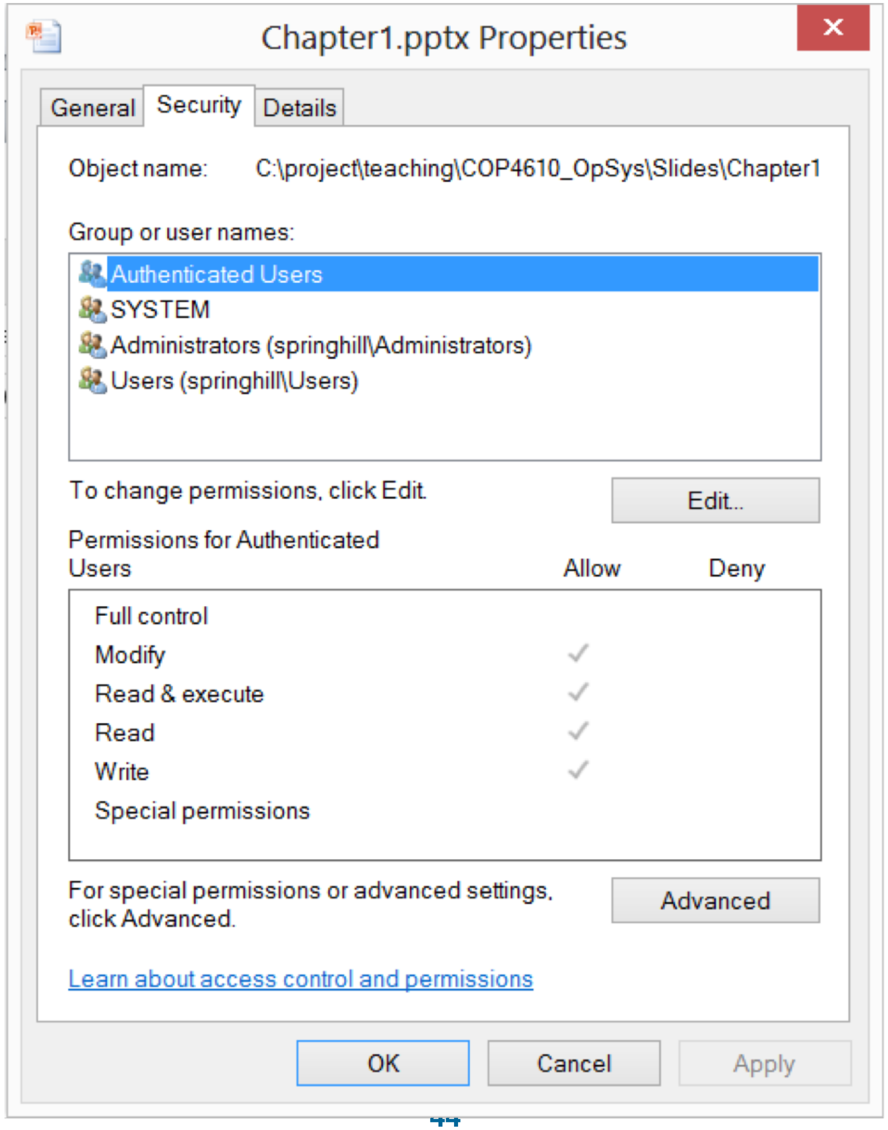
- To grant access to users, create a group and change its access mode
  - in Linux, use **chmod** and **chgrp**



owner group others

chmod 761 game

# Windows 8 File Access-Control



Chapter1.pptx Properties

Object name: C:\project\teaching\COP4610_OpSys\Slides\Chapter1

Group or user names:

- Authenticated Users
- SYSTEM
- Administrators (springhill\Administrators)
- Users (springhill\Users)

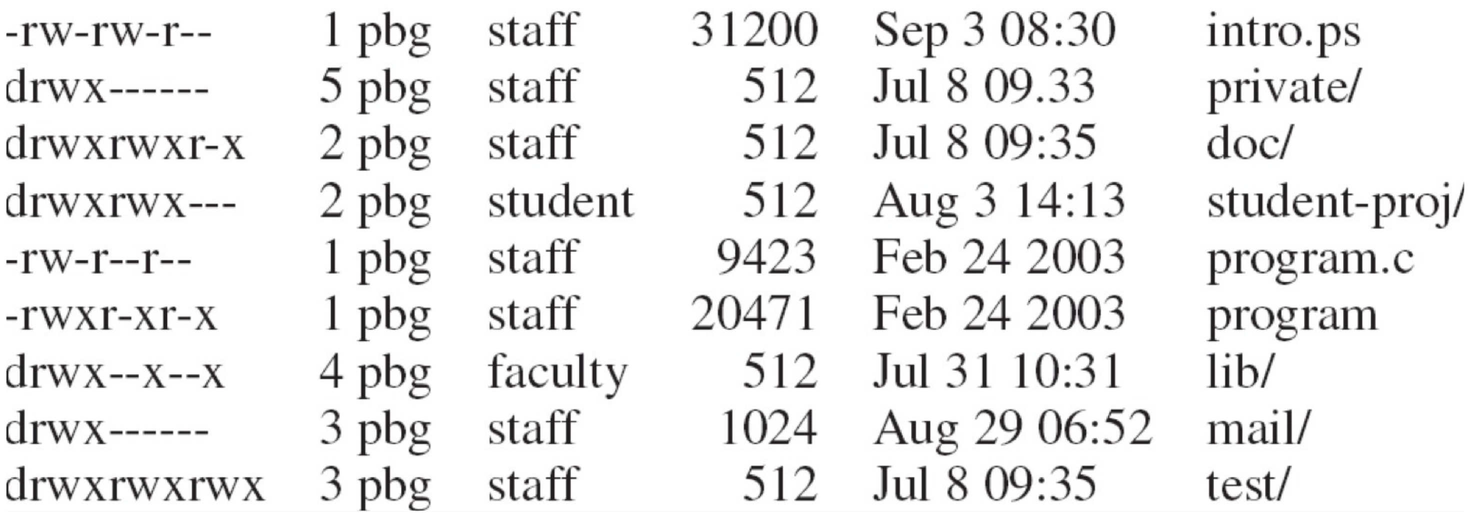
To change permissions, click Edit.

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

For special permissions or advanced settings, click Advanced.

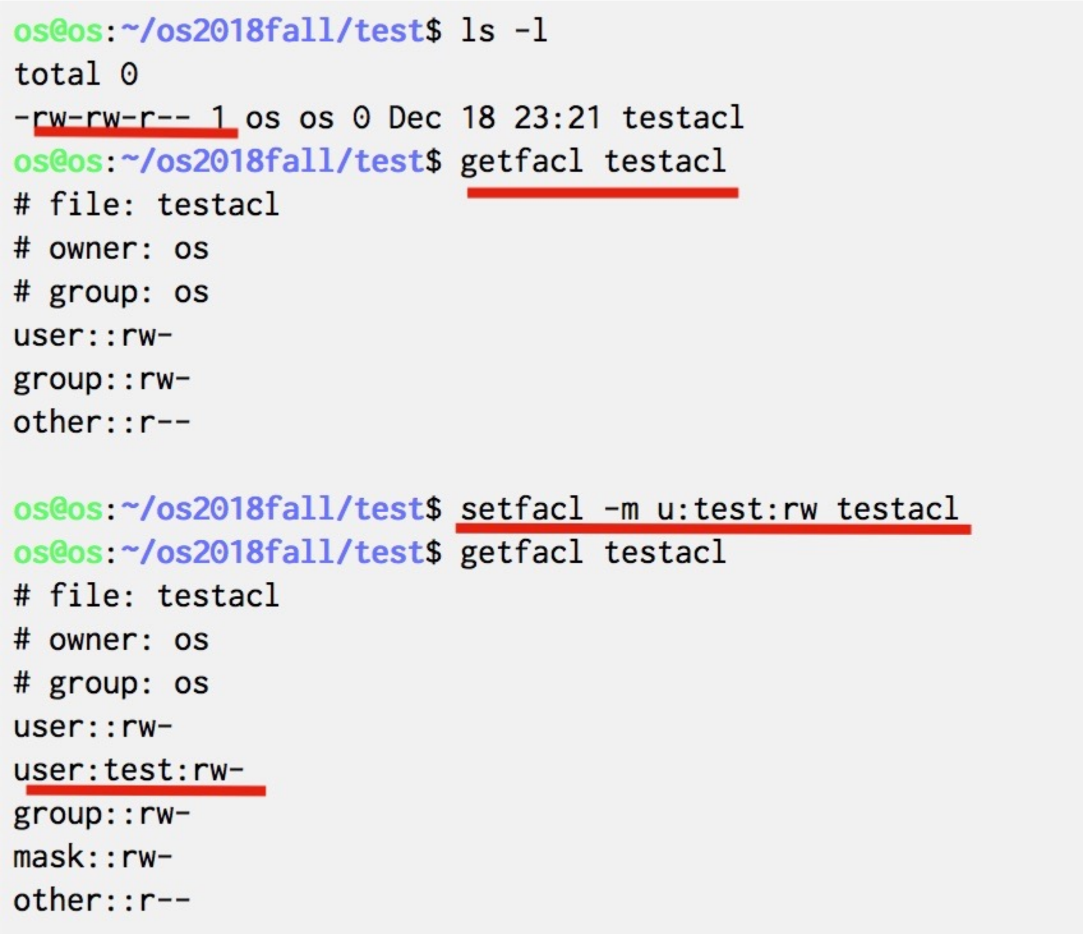
Learn about access control and permissions

# 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/
```

# ACL in practice



```
os@os:~/os2018fall/test$ ls -l
total 0
-rw-rw-r-- 1 os os 0 Dec 18 23:21 testacl
os@os:~/os2018fall/test$ getfacl testacl
# file: testacl
# owner: os
# group: os
user::rw-
group::rw-
other::r--

os@os:~/os2018fall/test$ setfacl -m u:test:rw testacl
os@os:~/os2018fall/test$ getfacl testacl
# file: testacl
# owner: os
# group: os
user::rw-
user:test:rw-
group::rw-
mask::rw-
other::r--
```

# Takeaway

- File system
- File operations
  - Create, open, read/write, close
- File type
- File structure
- File access
- Directory structure
  - Single level, two-level, tree, acyclic-graph, general graph
- Protection
  - ACL