File-System Interface

Politecnico di Torino

Department of Control and Computer Engineering

System Programming - Sarah Azimi

CAD & Reliability Group
DAUIN– Politecnico di Torino

Objectives

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

File-System Interface

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

- In this chapter:
  - We consider the various aspects of **files** and major **directory structure**.
  - Discussing the semantics of **sharing files** among **multiple processes**, **users**, and computers.
  - Methods to handle **file protection**, necessary when we have multiple users and want to control who may have access files and how files may be accessed.

File Concept

- Computer can store information on various storage media, such as NVM devices, HDDs, …
- The operating system provides a **uniform logical view** of stored information.
  - **File** - abstraction from a physical properties of the storage device to define a logical storage.
    - From the user perspective, **a file is a named collection of related information that is recorded on secondary storage.**
  - Files types
    - Program
    - Data (numeric, alphabetic, alphanumeric, binary)
    - The information in the file is defined by the creator, many types including **text file**, **source file**, **executable file**.

File Attributes

- A **file attributes** vary from one operating system to another but typically consist of these:
  - **Name** – only information kept in human-readable form
  - **Identifier** – unique tag (number) identifies the file within the file system. Non-human-readable name of the file
  - **Type** – needed for systems that support different types
  - **Location** – pointer to file location on the device
  - **Size** – current file size
  - **Protection** – controls who can do reading, writing, executing
  - **Time, date, and user identification** – information about creation, last modification, and last use useful for data for protection, security, and usage monitoring
- Information about files is kept in the directory structure, which is maintained on the disk

File Operations

- File is an abstract data type with operations that can be performed on them:
  - Creating a file:
    - A space in file system must be created.
    - An entry for the new file must be made in a directory.
  - Opening a file:
    - All operations except create and delete, require a file open() first.
    - If successful, the open call returns a file handle that is used as an argument in the other calls.
  - Writing a file:
    - Specifying both the open file handle 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:
    - Specifying both the file handler and where in memory the next block of the file should be put
    - The system needs to keep a **read pointer** to the location in the file where the next read is to take place.
    - Since the process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **current-file-position pointer**.
    - Both the read and write operations use this same pointer, saving space and reducing system complexity.

File Operations

- File is an abstract data type with operations that can be performed on them:
  - Repositioning within a file:
    - The current file position of the open file is repositioned to a given value which does not involve any I/O.
    - This operation is known also as a **file seek**.
  - Deleting a file:
    - Searching the directory for the named file.
    - Releasing all file space, so it can be reused by other files and erase or mark as free the directory entry.
  - Truncating a file:
    - Erasing the contents of a file but keeps its attribute, except for the file length.

Open Files

- Most of the fie operations involve searching the directory for the entry associated with the named file.
- Several pieces of information are associated with an open file:
  - **Open-file table**: containing information about all open files. When a file operation is requested, the file is specified via an index into this table. So, no searching is required.
  - File pointer: pointer to last read/write location, per process that has the file open.
  - **File-open count**: counter of the number of times a file is open – to allow removal of data from the open-file table when the last processes close it
  - Location of the file: Most file operations require the system to read or write data within the file. The information that is needed to locate the file is kept in memory, so the system does not have to read from the directory structure for each operation.
  - Access rights: each process opens a file in an access mode. This information is stored on the pre-process table so the operating system can allow or deny the request.

Open File Locking

- File locks allow one process to lock a file and prevent other processes from gaining access to it.
  - File locks are useful for files that are shared by several processes.
  - **Shared lock** several processes can acquire the lock concurrently.
  - **Exclusive lock** only one process at a time can acquire such a lock.
- File locking mechanisms:
  - **Mandatory** – once a process acquires an exclusive lock, the operating system will prevent any other process from accessing the locked file. The operating system ensures that locks are appropriately acquired and released
  - **Advisory** –It is up to the software developers to ensure that locks are appropriately acquired and released.
  - Normally, windows operating systems adopt mandatory locking and UNIX systems employ advisory locks.

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

File Structure

- Disk systems have a well-defined block size determined by the size of a sector.
- All disk I/O is performed in units of one block,
  - All blocks are the same size.
- It is unlikely that the physical record size will exactly match the length of the desired logical record.
  - Logical records may vary in length.

- Packing a number of logical records into a physical blocks is a common solution to this problem.
  - The logical record size, physical block size and packing technique determine how many logical record are in each physical block.
  - Packing can be done either by the user application program or by the operating system

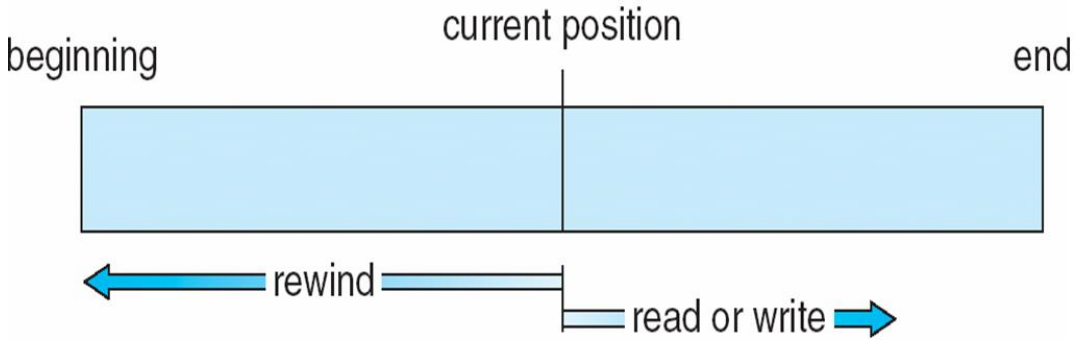
File Structure

- Disk space is always allocated in blocks, therefore, some portion of the last block of each file is generally wasted.
  - If each block were 512 bytes, then a file of 1,949 bytes would be allocated four blocks (2,048 bytes).
  - The last 99 bytes would be wasted.

- The waste incurred to keep everything is units of blocks (instead of bytes) is internal fragmentation.
- All file systems suffer from internal fragmentation
  - The larger the block size, the greater the internal fragmentation.

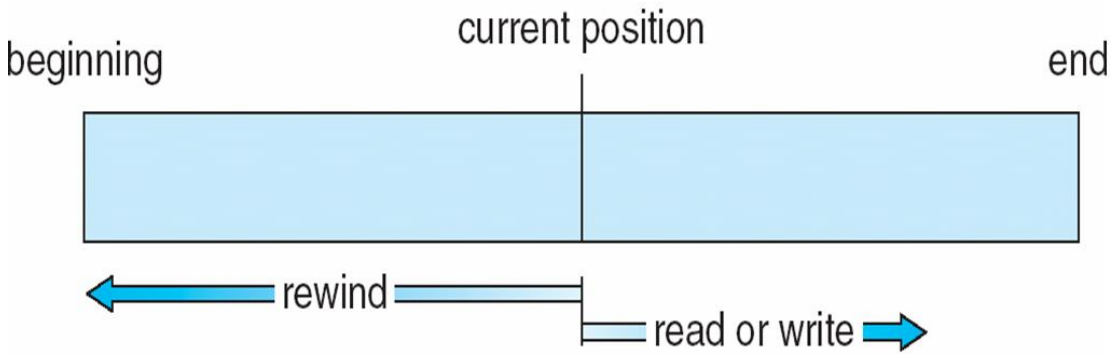
Access Methods

- File store information. When it is used, this information must be accessed and read into computer memory.
- The information in the file can be access in several ways:
  - Sequential Access
  - Direct Access
  - Other Access methods



Sequential Access

- **Sequential Access** is when the information in the file is processes in order, one record after the other.
  - A read operation – *read_next()*- reads the next portion of the file and automatically advances a file pointer which tracks the I/O location.
  - A write operation – *write_next()* – appends to the end of the file and advances to the end of the newly written material.
  - Such a file can be reset to the beginning, a program may be able to skip forward or backward *n* records.



Direct Access

- A file is a made-up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- The direct access method is based on disk model of a file.
  - Disk allows random access to any file block
- For direct access, the file is viewed as a numbered sequence of blocks or records.
  - We may read block 14, then read block 53, and then write block 7.
  - There are no restrictions on the order of reading or writing for a direct-access file.

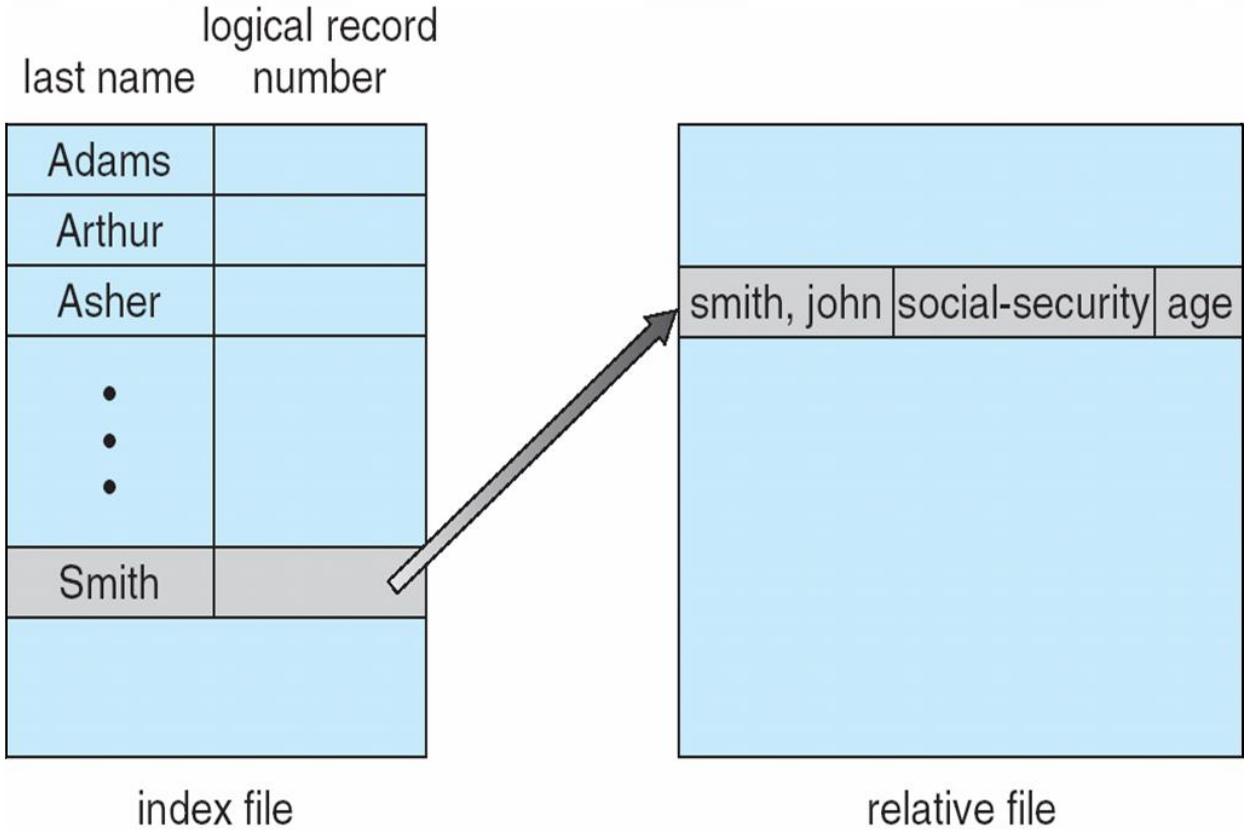
Direct Access

- For the direct-access, the file operations must be modified to include the block number as parameter.
  - read(n) and write(n) where n is the block number (instead of read_next() and write_next())
  - The block number provided by the user to the operating system is **relative block number**.
    - It is an index relative to the beginning of the file
  - The relative block number allows the operating system to decide where the file should be places, **allocation problem**.

Other Access Methods

- Can be built on top of base methods
- General involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
  - 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 (see next slide)

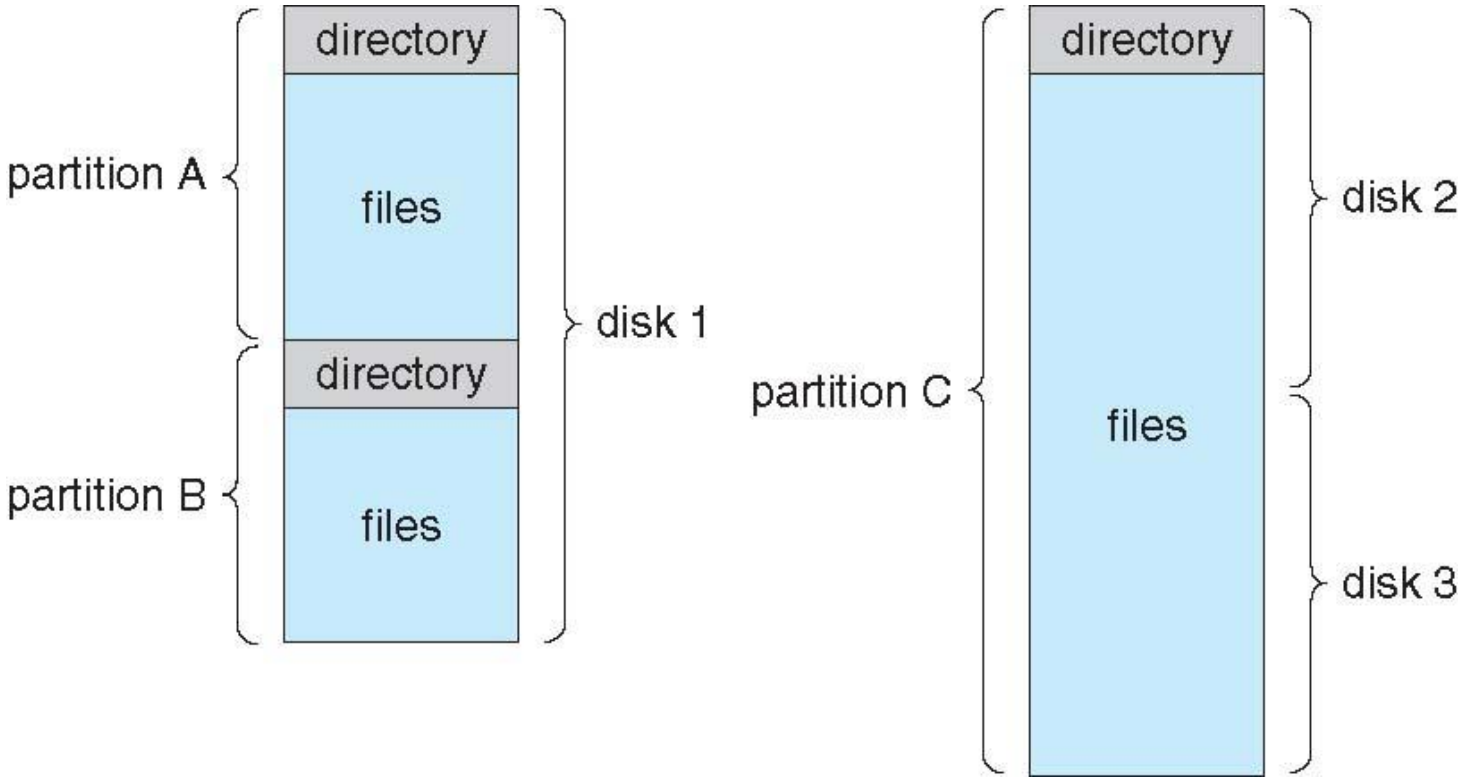
Example of Index and Relative Files



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 also known as minidisks, slices.
- Entity containing file system known as a **volume.**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents.**
- As well as **general-purpose file systems** there are many **special-purpose file systems,** frequently all within the same operating system or computer.

A Typical File-system Organization

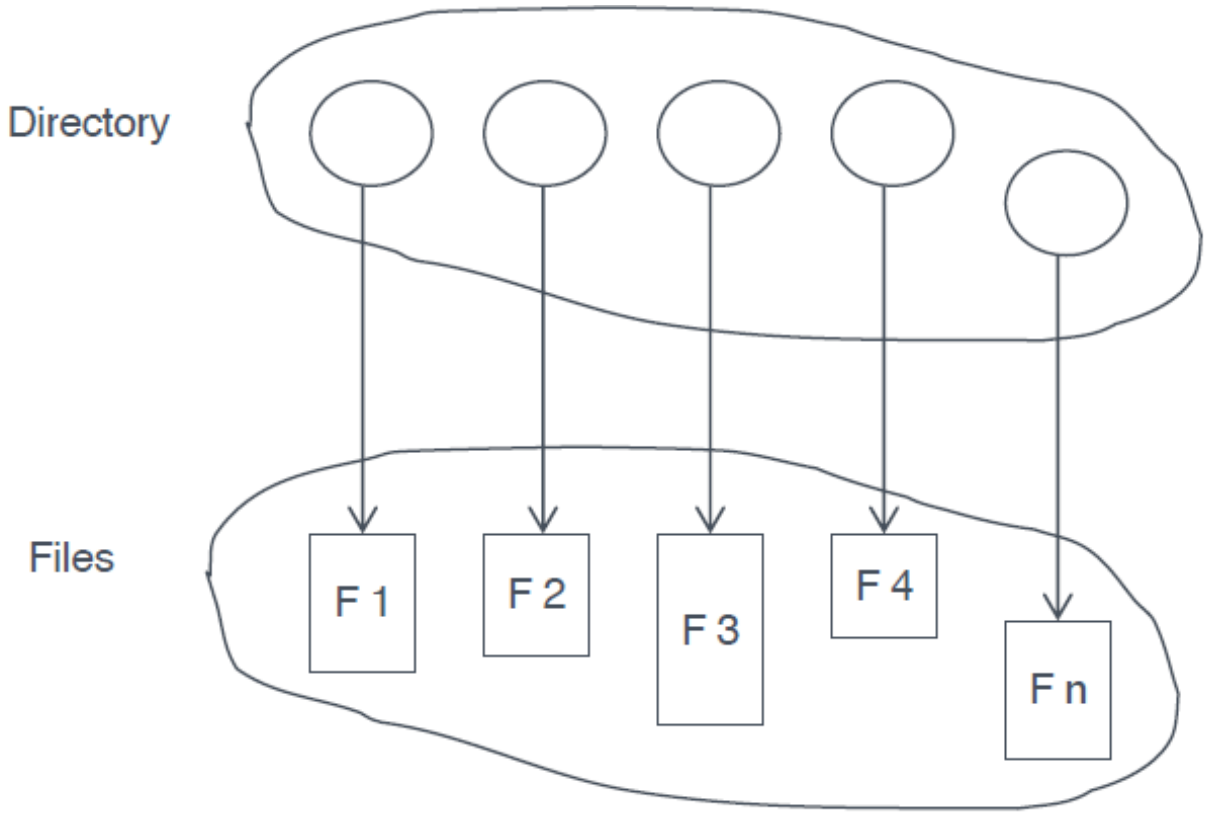


Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Directory Structure

- A collection of nodes containing information about files
  - Both the directory structure and the files reside on disk



Operation Performed on Directory

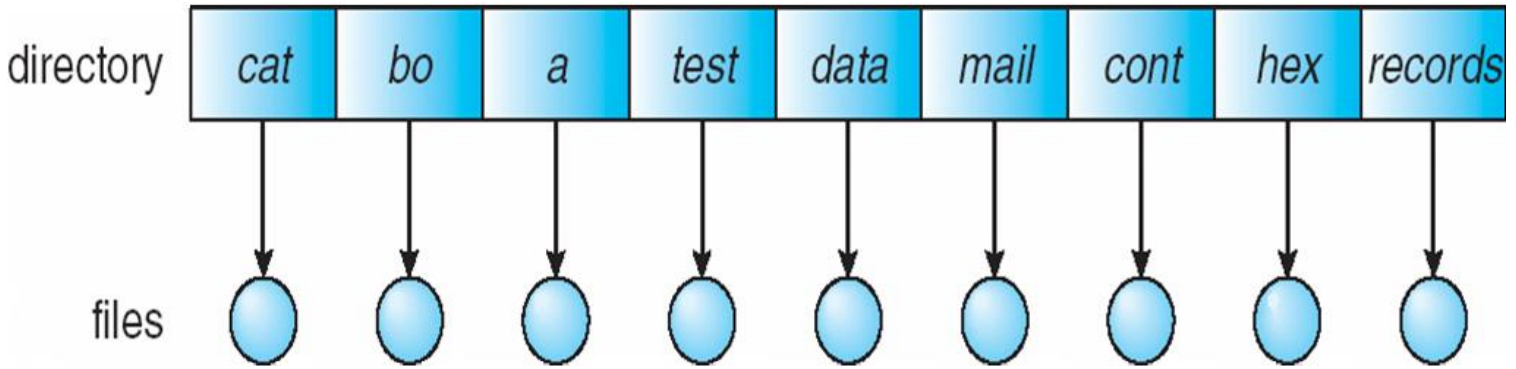
- The directory can be organized in many ways performing the following operations:
  - Search for a file
    - Searching a directory to find the entry for a particular file
  - Create a file
  - List a directory
    - Listing the files in the directory
  - Rename a file
  - Traverse the file system
    - Saving the contents and structure of the entire file system at regular intervals
    - Copying all files to magnetic tape, other secondary storage or …

Directory Organization

- The directory is organized 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, …)

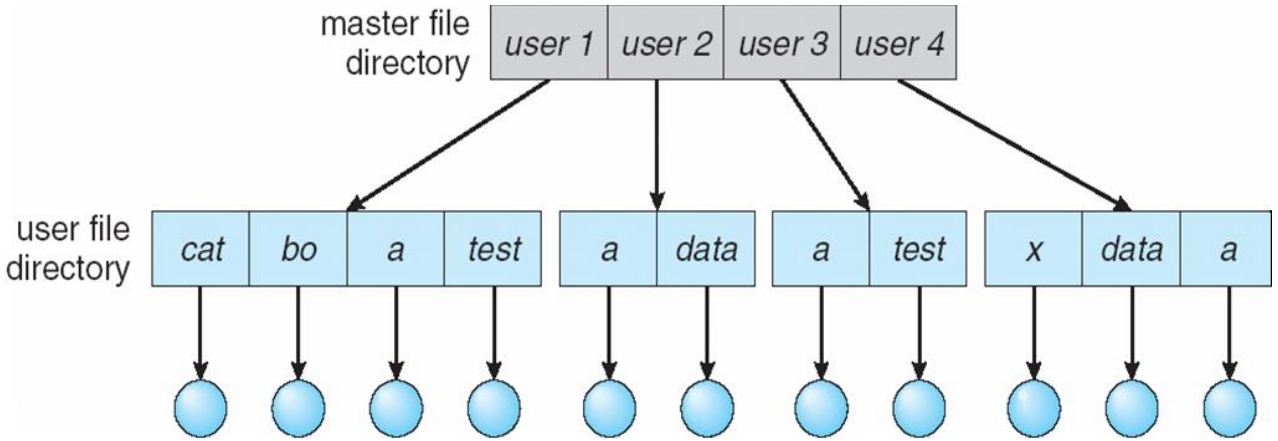
Single – Level Directory

- The simplest directory structure is the single-level directory.
- All files are contained in the same directory.
  - Easy to support and understand
  - All the files are in the same directory, they must have unique names.
  - If two users call their file the same, this rule is violated.
  - Single user or a single level directory should remember the names of all files as the number of files increases.
  - Keeping track of all files is a daunting task.



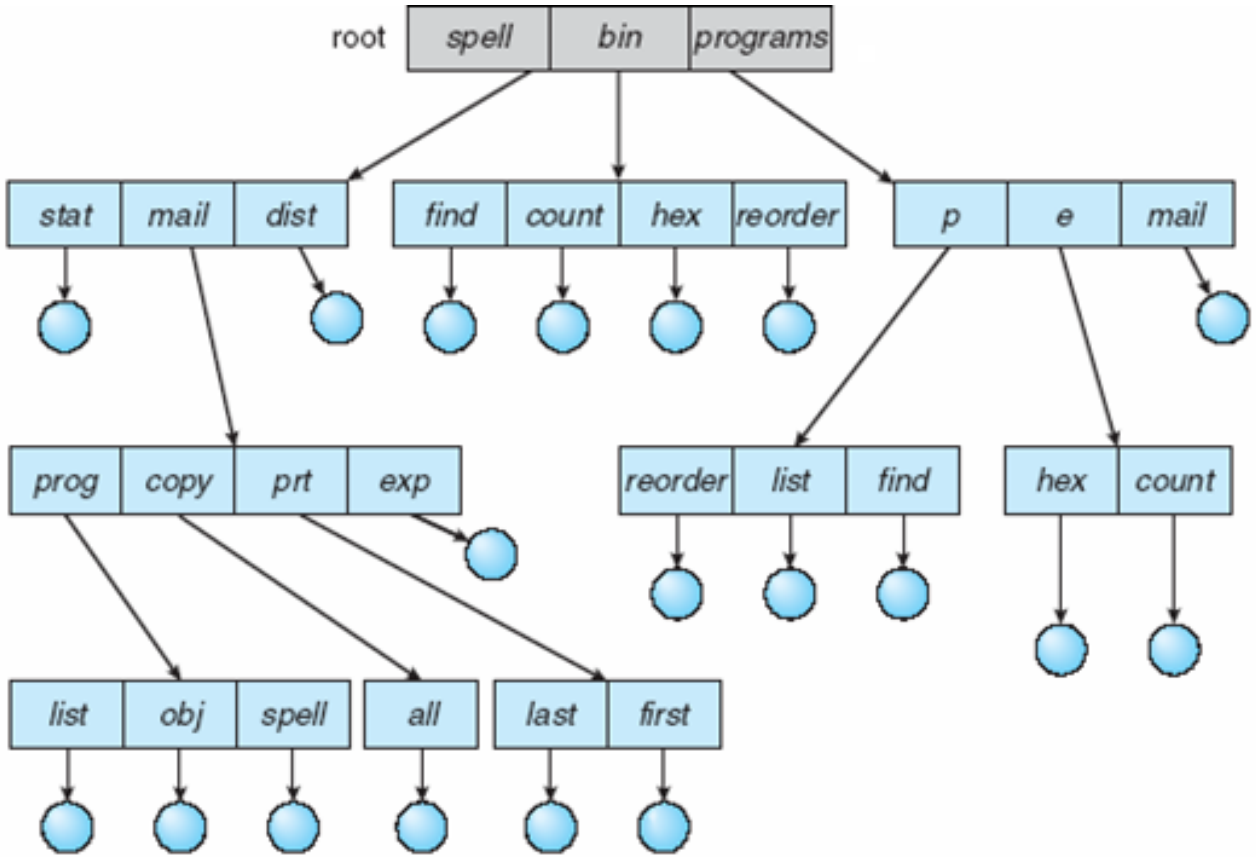
Two – Level Directory

- A solution to the confusion of single-level directory is to create a separate directory for each user.
- In two-level directory structure, each user has his own **User File Directory** (UFD)
  - The UFDs have similar structures but each lists only the files of a single user.
- When a user logs in, the systems **Master File Directory** (MFD) is searched.
  - The MFD is indexed by username or account number, and each entry points to the UDF for that user.
- When a user refers it a particular file, only his own UFD is searched.
- Users can have files with the same names.



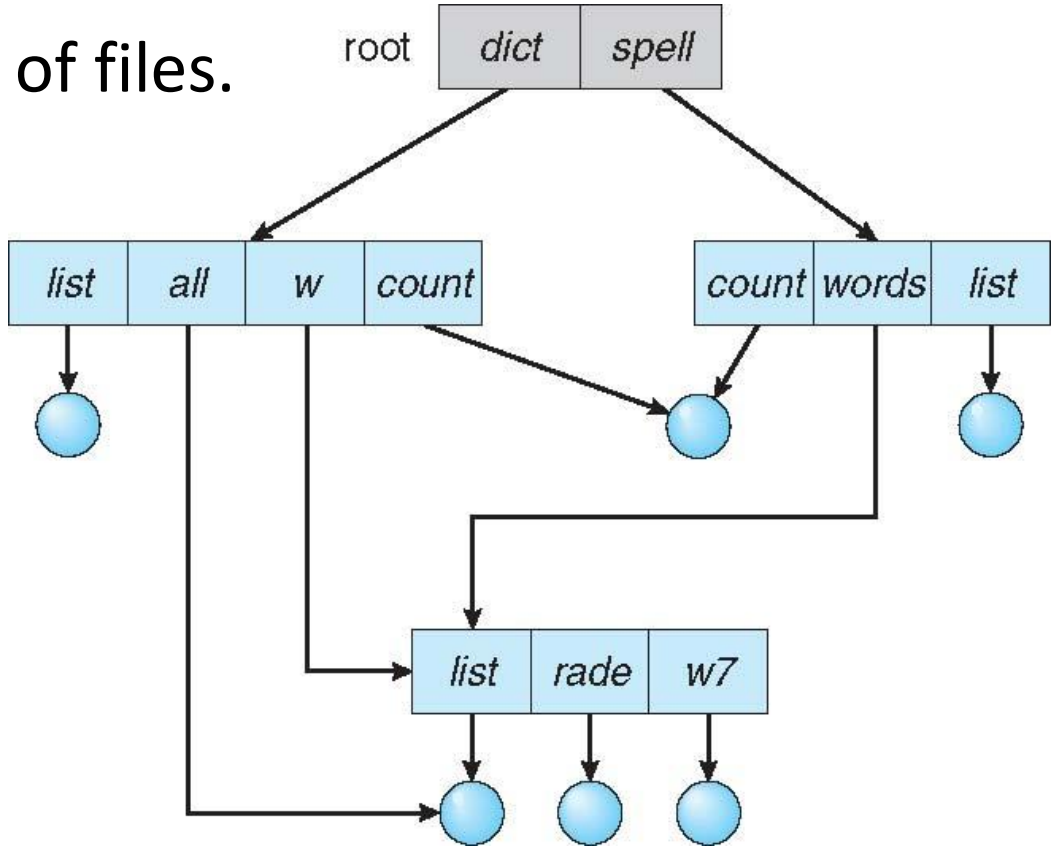
Tree-Structured Directories

- Extending the two-level tree to a tree of arbitrary height.
  - Allows user to create their own subdirectories and to organize their file accordingly.
  - The tree has a root directory, and every file in the system has a unique path name.



Acyclic-Graph Directories

- Have shared subdirectories and files.
- An acyclic graph – that is a graph with no cycles- allows directories to shared subdirectors and file.
- The shared file is not the same as two copies of files.



Acyclic-Graph Directories

- Shared files can be implemented in several ways:
  - Creating a new directory entry, called a **link** which is effectively a pointer to another file or subdirectory.
  - When a reference file is made, we search the directory. If the directory entry is marked a link, then the name of the real file is included in the link information.
  - We **resolve** the link by using the path name to locate the real file.

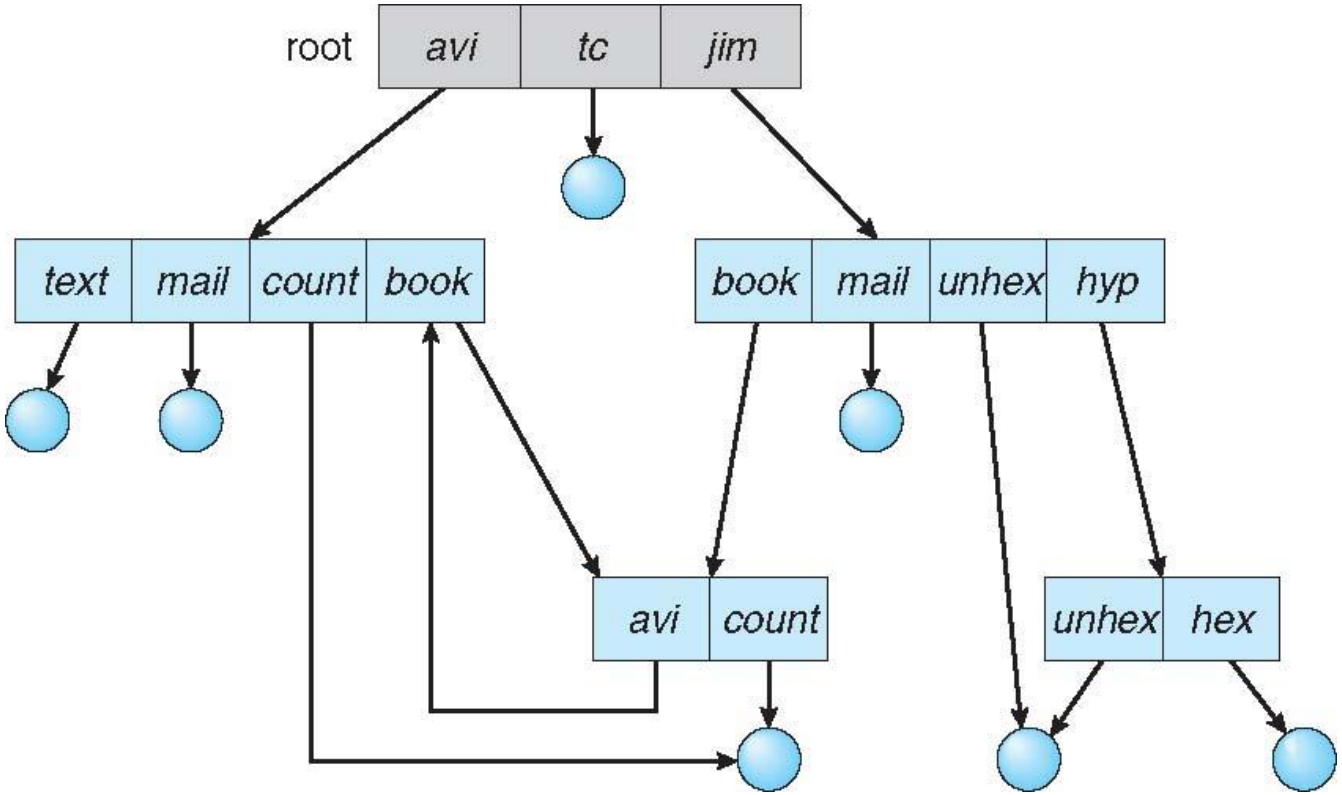
Acyclic-Graph Directories

- Shared files can be implemented in several ways:
  - Another common approach is duplicate all information about them in both sharing directories. Thus, both entries are equal.

- Difference between this approach and link:
  - Link is different from the original directory entry, thus the two are not identical.
  - Duplicate directory entries, make the original and copy indistinguishable.

General Graph Directory

- 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



Protection

- When information is stored in a computer system, it should be kept safe from improper access - **protection**.
- The need to protect file is a direct result of the ability to access files.
- Protection mechanism, provide controlled access by limiting the types of file access that can be made.
- Several types of operations may be controlled:
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**
  - **Attribute Change**

Protection

- Protection mechanisms:
- The most common one is to make access dependent on the identity of the user.
  - Different user may need different types of access to a file or a directory.
  - Implementing identify dependent access by associating with each file and directory as **Access-Control List (ACL),** specifying user names and the types of access allowed for each user.
    - When a user requests access to a particular file, the operating systems checks the access list associated with that file.
    - If that user is listed for the requested access, the access is allowed.
    - Otherwise, a protection violation occurs, and the user job is denied.

Protection

- Advantage:
  - Enabling complex access methologies.
- Disadvantages:
  - Constructing such a list may be a tedious and unrewarding tasks, especially if the list of users in the system is not known.
  - The directory entry must be a variable size.

- Solution: using reduced version of the access list.

Protection

- Many system organize three classifications of users in connection with each file:
  - **Owner**: the user who created the file is the owner.
  - **Group**: A set of users who are sharing the file and need similar access is a group, or work group.
  - **Other**: All other users in the system.

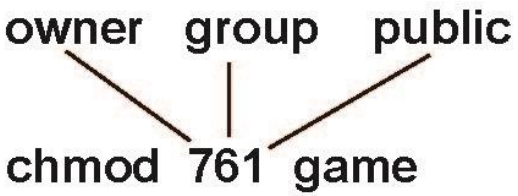
- The most common recent approach is to combine access-control lists with the more general owner, group and universe access control scheme.

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

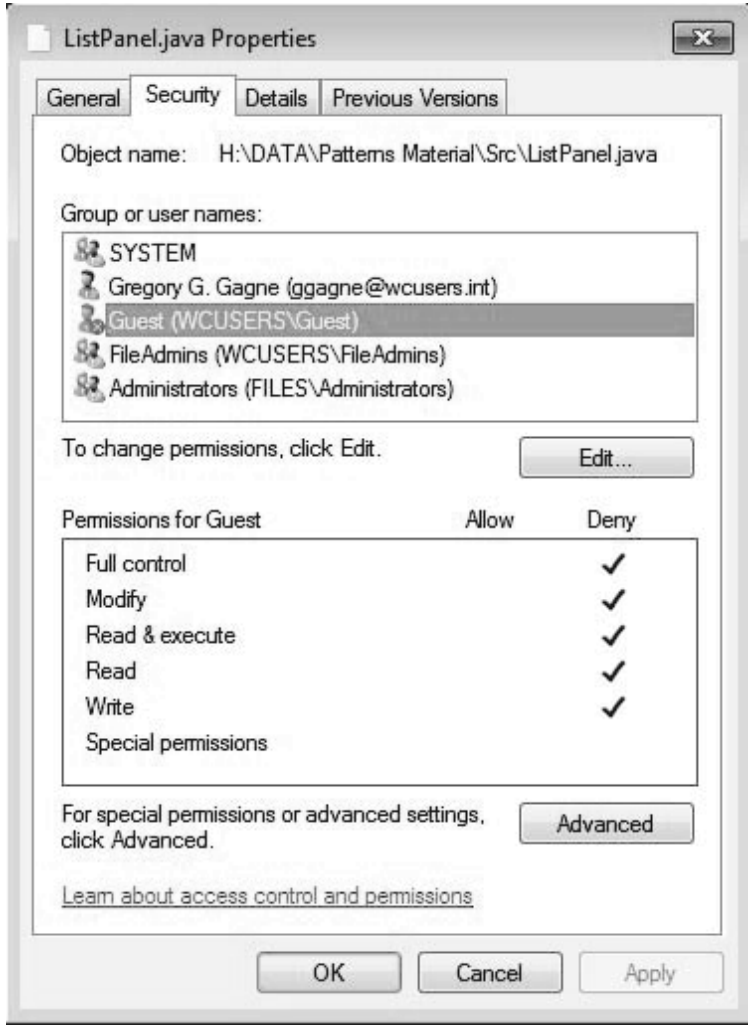
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

```
chgrp       G       game
```

Windows 7 Access-Control List Management



Thanks

Politecnico di Torino

Department of Control and Computer Engineering

Questions?

sarah.azimi@polito.it