Operating System

UNIT 2

Part 2 – File System Interface

File Concept

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**.
- Files are mapped by the operating system onto physical devices. These storage devices are usually nonvolatile, so the contents are persistent between system reboots.
- A file is a named collection of related information that is recorded on secondary storage.
- Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free form, such as text files, may be formatted rigidly.
- A **text file** is a sequence of characters organized into lines (and possibly pages). A **source file** is a sequence of functions, each of which is further organized as declarations followed by executable statements.
- An **executable file** is a series of code sections that the loader can bring into memory and execute.

File Attributes

- A file's attributes vary from one operating system to another but typically consist of these:
    - **Name**. The symbolic file name is the only information kept in human readable form.
    - **Identifier**. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
    - **Type**. This information is needed for systems that support different types of files.
    - **Location**. This information is a pointer to a device and to the location of the file on that device.
    - **Size**. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
    - **Protection**. Access-control information determines who can do reading, writing, executing, and so on.
    - **Time, date, and user identification**. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

- The information about all files is kept in the directory structure, which also resides on secondary storage. A directory entry consists of the file's name and its unique identifier. Since directories are like files, must be nonvolatile, they must be stored on the device and brought into memory as needed.

File Operations

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

- **Creating a file**. Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

- **Writing a file**. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a **write pointer** to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

- **Reading a file**. To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a **read pointer** to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.

- **Repositioning within a file**. The directory is searched for the appropriate entry, and the current-file position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek**.

- **Deleting a file**. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

- **Truncating a file**. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

These six basic operations comprise the minimal set of required file operations. Other common operations include appending new information to the end of an existing file and renaming an existing file. These primitive operations can then be combined to perform other file operations.

File Types

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

File types also can be used to indicate the internal structure of the file. The source and object files have structures that match the expectations of the programs that read them.
Certain files must conform to a required structure that is understood by the operating system.

Internal File Structure

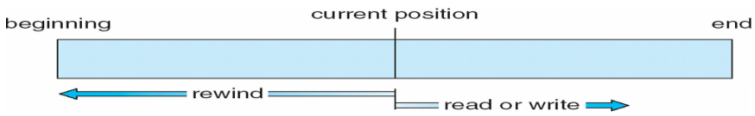
Internally, locating an offset within a file can be complicated for the operating system. Disk systems typically have a well-defined block size determined by the size of a sector.
All disk I/O is performed in units of one block (physical record), and all blocks are the same size.
It is unlikely that the physical record size will exactly match the length of the desired logical record.
Logical records may even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.
The logical record size, physical block size, and packing technique determine how many logical records are in each physical block.
The packing can be done either by the user's application program or by the operating system. In either case, the file may be considered a sequence of blocks. All the basic I/O functions operate in terms of blocks.
The conversion from logical records to physical blocks is a relatively simple software problem.

Access Methods

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

Sequential Access

The simplest access method is **sequential access**. Information in the file is processed in order, one record after the other.

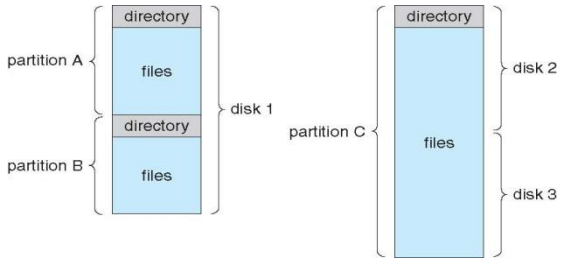


Direct Access

- Another method is **direct access** (or **relative access**). Here, a file is made up of fixed-length **logical** that allow programs to read and write records rapidly in no particular order.
- The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. Direct-access files are of great use for immediate access to large amounts of information.

Directory and Disk Structure

- A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control.
- Partitioning is useful for limiting the sizes of individual file systems, putting multiple file-system types on the same device, or leaving part of the device available for other uses.
- A file system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a **volume**. The volume may be a subset of a device, a whole device, or multiple devices.
- Each volume can be thought of as a virtual disk. Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.
- Each volume that contains a file system must also contain information about the files in the system.
- This information is kept in entries in a **device directory** or **volume table of contents**. The device directory (more commonly known simply as the **directory**) records information—such as name, location, size, and type—for all files on that volume.



Storage Structure

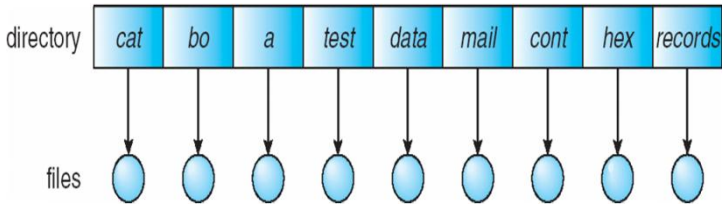
- A general-purpose computer system has multiple storage devices, and those devices can be sliced up into volumes that hold file systems.
- Computer systems may have zero or more file systems, and the file systems may be of varying types. Consider the types of file systems
    - **tmpfs**—a "temporary" file system that is created in volatile main memory and has its contents erased if the system reboots or crashes.
    - **objfs**—a "virtual" file system (essentially an interface to the kernel that looks like a file system) that gives debuggers access to kernel symbols.
    - **ctfs**—a virtual file system that maintains "contract" information to manage which processes start when the system boots and must continue to run during operation.
    - **lofs**—a "loop back" file system that allows one file system to be accessed in place of another one.
    - **procfs**—a virtual file system that presents information on all processes as a file system.
    - **ufs, zfs**—general-purpose file systems.
- The file systems of computers, then, can be extensive. Even within a file system, it is useful to segregate files into groups and manage and act on those groups. This organization involves the use of directories.

Directory Overview

- The directory can be viewed as a symbol table that translates file names into their directory entries. If we take such a view, we see that the directory itself can be organized in many ways.
- The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory.
- When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:
    - **Search for a file**: We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.
    - **Create a file**: New files need to be created and added to the directory.
    - **Delete a file**: When a file is no longer needed, we want to be able to remove it from the directory.
    - **List a directory**: We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
    - **Rename a file**: Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
    - **Traverse the file system**: We may wish to access every directory and every file within a directory structure.

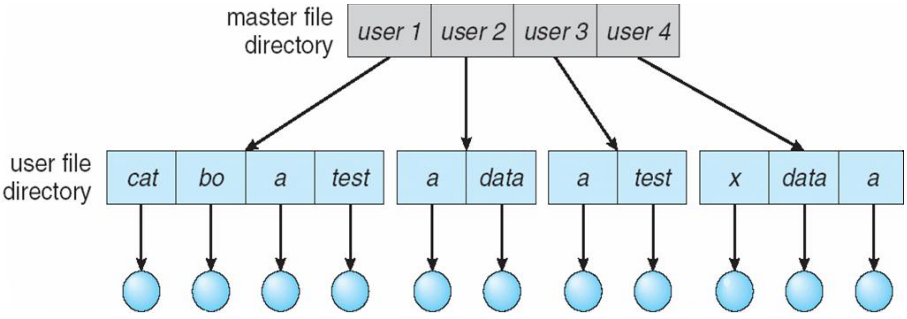
Single-Level Directory

- The simplest directory structure is the single-level directory. All files are contained in the same directory.
- A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user.
- Since all files are in the same directory, they must have unique names. Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.
- Keeping track of so many files is a daunting task.



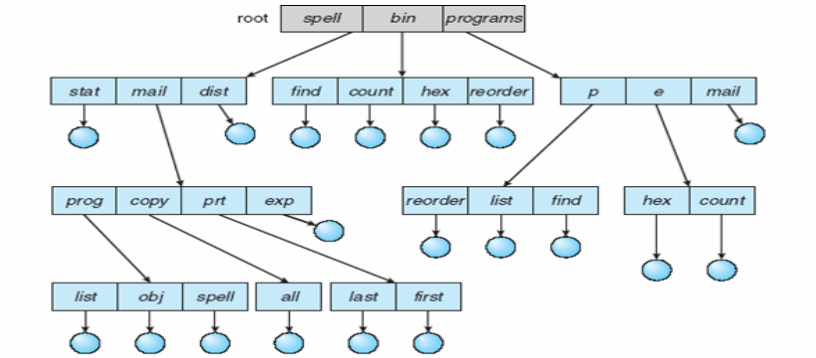
Two-Level Directory

- In the 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 job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
- To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists.
- To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.



Tree-Structured Directories

- The natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly.
- A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories. In normal use, each process has a current directory. The **current directory** should contain most of the files that are of current interest to the process.
- When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.
- Path names can be of two types: absolute and relative. An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path. A **relative path name** defines a path from the current directory.



File-System Mounting

- A file system must be mounted before it can be available to processes on the system. More specifically, the directory structure may be built out of multiple volumes, which must be mounted to make them available within the file-system name space.
- The mount procedure is straightforward. The operating system is given the name of the device and the **mount point**—the location within the file structure where the file system is to be attached.
- The operating system verifies that the device contains a valid file system. It does so by asking the device driver to read the device directory and verifying that the directory has the expected format.
- The operating system notes in its directory structure that a file system is mounted at the specified mount point. This scheme enables the operating system to traverse its directory structure, switching among file systems, and even file systems of varying types, as appropriate.

File Sharing

File sharing is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal.

Multiple Users

- When an operating system accommodates multiple users, the issues of file sharing, file naming, and file protection become preeminent.
- Given a directory structure that allows files to be shared by users, the system must mediate the file sharing.
- The system can either allow a user to access the files of other users by default or require that a user specifically grant access to the files.
- To implement sharing and protection, the system must maintain more file and directory attributes than are needed on a single-user system.
- Although many approaches have been taken to meet this requirement, most systems have evolved to use the concepts of file (or directory) **owner** (or **user**) and **group**.
- The owner is the user who can change attributes and grant access and who has the most control over the file. The group attribute defines a subset of users who can share access to the file.

Remote File Systems

- Networking allows the sharing of resources spread across a campus or even around the world. One obvious resource to share is data in the form of files.
- The first implemented method involves manually transferring files between machines via programs like ftp.
- The second major method uses a **distributed file systems (DFS) in which remote directories are** visible from a local machine.
- The third method, the **World Wide Web**, is a reversion to the first. A browser is needed to gain access to the remote files, and separate operations are used to transfer files.

The Client–Server Model

- Remote file systems allow a computer to mount one or more file systems from one or more remote machines.
- In this case, the machine containing the files is the **server**, and the machine seeking access to the files is the **client**. The client–server relationship is common with networked machines.
- The server declares that a resource is available to clients and specifies exactly which resource (in this case, which files) and exactly which clients.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client–server facility.

Protection

When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).

Types of Access

- The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection.
- Thus, we could provide complete protection by prohibiting access.
- Protection mechanisms provide controlled access by limiting the types of file access that can be made.
- Access is permitted or denied depending on several factors, one of which is the type of access requested.
- Several different types of operations may be controlled:
  - **Read**: Read from the file.
  - **Write:** Write or rewrite the file.
  - **Execute**: Load the file into memory and execute it.
  - **Append**: Write new information at the end of the file.
  - **Delete**: Delete the file and free its space for possible reuse.
  - **List**: List the name and attributes of the file.
- Other operations, such as renaming, copying, and editing the file, may also be controlled.

Access Control

- The most general scheme to implement identity dependent access is to associate with each file and directory an **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 system 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 access to the file.
- To condense the length of the access-control list, many systems recognize 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.
  - **Universe**: All other users in the system constitute the universe.