Unit 1
File Management

Structure of the Unit

1. Unit Outcomes
2. Operating system objectives and functions
3. Files and File systems
4. File Organization and Access
5. File Directory and Sharing
6. Self-Assessment Questions
7. Self-Assessment Activities
8. Multiple Choice Questions
9. Keys to Multiple Choice Questions
10. Summary of the Unit
11. Keywords
12. Recommended Resources for Further Reading

1.1 Unit Outcomes

After the successful completion of this unit, the student will be able to:

1. Define and explain the attributes associated with files.
2. Describe the basic file operations in an operating system.
3. Compare and contrast the two primary access methods for files: sequential access and direct access.
4. Describe the basics of the directory structure.

OS → bridge between hardware & software

1.2 Operating system objectives and functions

An operating system (OS) is a program that manages the execution of application programs and serves as a bridge between applications and computer hardware. It fulfills three main objectives:

1. **Convenience:** The OS enhances the user experience by making the computer more user-friendly and accessible.
2. **Efficiency:** By efficiently allocating and managing computer system resources, the OS ensures optimal utilization of hardware capabilities.
3. **Ability to evolve:** A well-designed OS allows for the effective development, testing, and integration of new system functions without disrupting existing services.

Now, let's explore these three aspects of an OS in detail.

1.2.1 The Role of the Operating System as an Interface between Users and Computers

The interaction between users and computers involves a layered structure, as illustrated in Figure 1.1. Generally, users are not concerned with the intricacies of computer hardware and perceive a computer system in terms of the applications they use. Applications are developed by programmers using programming languages. However, developing an application solely responsible for controlling the hardware would be an overwhelmingly complex task.

To simplify this process, system programs are provided, including utilities and library programs. These programs implement frequently used functions that aid in program development, file management, and input/output (I/O) device control. Application programmers leverage these facilities during development, and running applications utilize utilities to perform specific tasks. The most critical collection of system programs is the OS itself.

The OS shields the programmer from hardware details, providing a convenient interface for utilizing system resources. It acts as an intermediary, facilitating easier access and usage of system facilities and services for both programmers and applications.

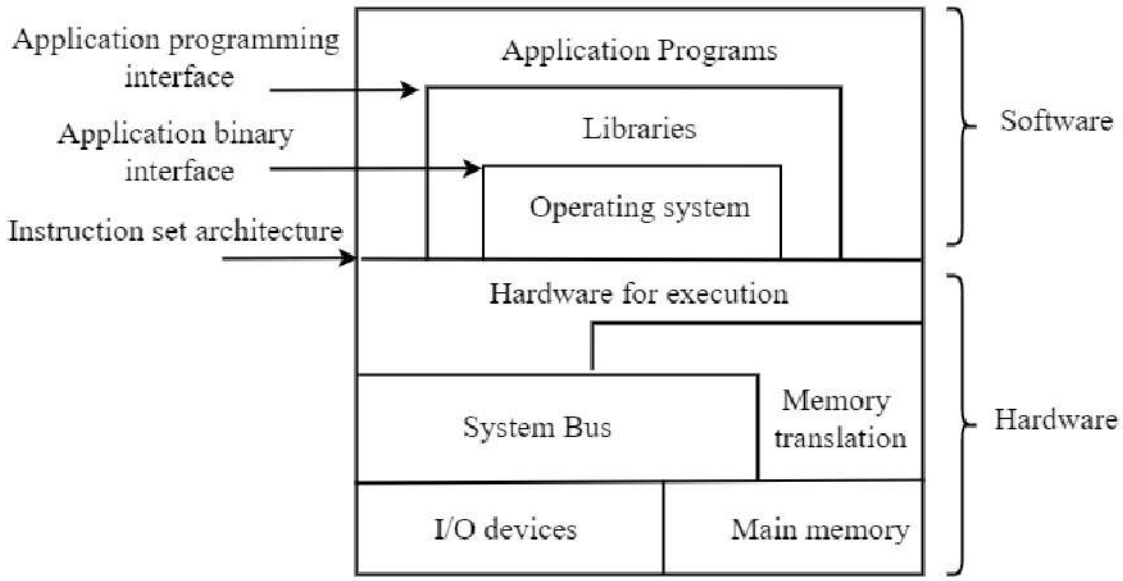

Figure 1.1: Structure of the computer

The OS typically offers services in various areas:

- Program development: When people write computer programs, the operating system gives them helpful tools like editors and debuggers. These tools are like special helpers that make it easier to create programs. They're part of the operating system and are called application program development tools.
- Program execution: When we run a program on a computer, a bunch of things happen. The computer gets the instructions and information ready, makes sure devices like printers are ready, and sets up everything needed. The operating system helps with all these steps to make things

work smoothly for the user.

- Access to I/O devices: Imagine different devices like printers or keyboards for a computer. Each device needs special instructions to work. The operating system makes things easier by giving a common way for programmers to talk to these devices. This way, programmers can use simple "read" and "write" commands to make devices work without worrying about the complicated stuff.
- Controlled access to files: The OS must have a comprehensive understanding of both the I/O device nature (e.g., disk drive, tape drive) and the structure of data stored in files. In multi-user systems, the OS may incorporate protection mechanisms to control file access.
- System access: When we use computers that many people share or that are available to the public, the operating system controls who can use the whole system and its different parts. This control helps to keep our things safe from people who shouldn't be using them and solves problems that happen when many people want to use the same thing at once.
- Error detection and response: Various errors can occur during the operation of a computer system, including hardware failures, software errors, and application-related issues. The OS must respond to these errors with minimal impact on running applications, ranging from terminating the faulty program to retrying operations or reporting errors to the application.
- Accounting: In the world of computers, a carefully created operating system gathers data about how resources are used and keeps an eye on things like how quickly the system responds. This data helps us predict how to make the system better in the future and make it work even faster. When many people are using the system, the collected data can also help in making bills for the services used.

Figure 1.1 also highlights three main interfaces that are part of a computer system:

- Instruction set architecture (ISA): The ISA defines the set of machine language instructions that a computer can execute. Instruction Set Architecture (ISA): The ISA is like the rulebook for a computer. It lists all the special instructions the computer understands and follows. This rulebook acts as a bridge between the computer's physical parts and the software that runs on it. Think of it this way: When we use a computer, we give it instructions in a language it understands. This language is part of the ISA. Some instructions are for regular programs we use, like games and apps (we call this the "user ISA"). Other instructions are like secret tools that only the operating system can use to manage the computer better (this is the "system ISA"). So, the ISA helps both the everyday programs and the computer's manager (the operating system) work together smoothly.
- Application binary interface (ABI): The ABI establishes a standard for binary compatibility across programs. The ABI sets up a common rulebook that helps different programs work well together. It's like a friendly agreement that ensures programs can talk to each other without any confusion. This agreement also explains how programs can talk to the computer's manager (the operating system) and use the special tools and powers that the computer has. In simple words, the ABI makes sure that programs can understand each other and work smoothly, and it shows them the right way to ask the computer for help when needed.

- Application programming interface (API): The API allows programs to access hardware resources and services using the user ISA supplemented with high-level language (HLL) library calls. API lets programs use the computer's tools and special abilities by talking in a way they all understand. It's like a special phone that programs can use to call for help or ask for things. When programs need something from the computer, like using the printer or getting data from the internet, they can ask for help using the API. This API uses a mix of the computer's basic language and some more advanced words from a library that helps them understand each other better. API makes it easy for programs to work on different computers that understand the same language. It's like a translator that helps programs run on different computers without too much trouble.

1.2.2 Operating System as Resource Manager

The Operating System (OS) serves as a manager for a computer's resources, such as input/output (I/O), primary and secondary memory, and processor execution time. However, the way this control is exercised is unique. Typically, a control mechanism is seen as something separate from the object being controlled. For instance, a thermostat controls a residential heating system, but it is distinct from the heat generation and distribution apparatus. In contrast, the OS functions differently as a control mechanism in two ways:

The OS operates like regular computer software, meaning it is a program or a set of programs executed by the processor.

The OS frequently gives up control and relies on the processor to allow it to regain control.

Similar to other computer programs, the OS comprises instructions that are executed by the processor. While executing, the OS determines how processor time should be allocated and which computer resources are available for use. However, for the processor to act on these decisions, it must temporarily stop executing the OS program and switch to running other programs. Consequently, the OS relinquishes control to enable the processor to perform "useful" tasks and then resumes control briefly to prepare the processor for the next set of tasks. The details of these mechanisms will become clearer as you progress through the chapter.

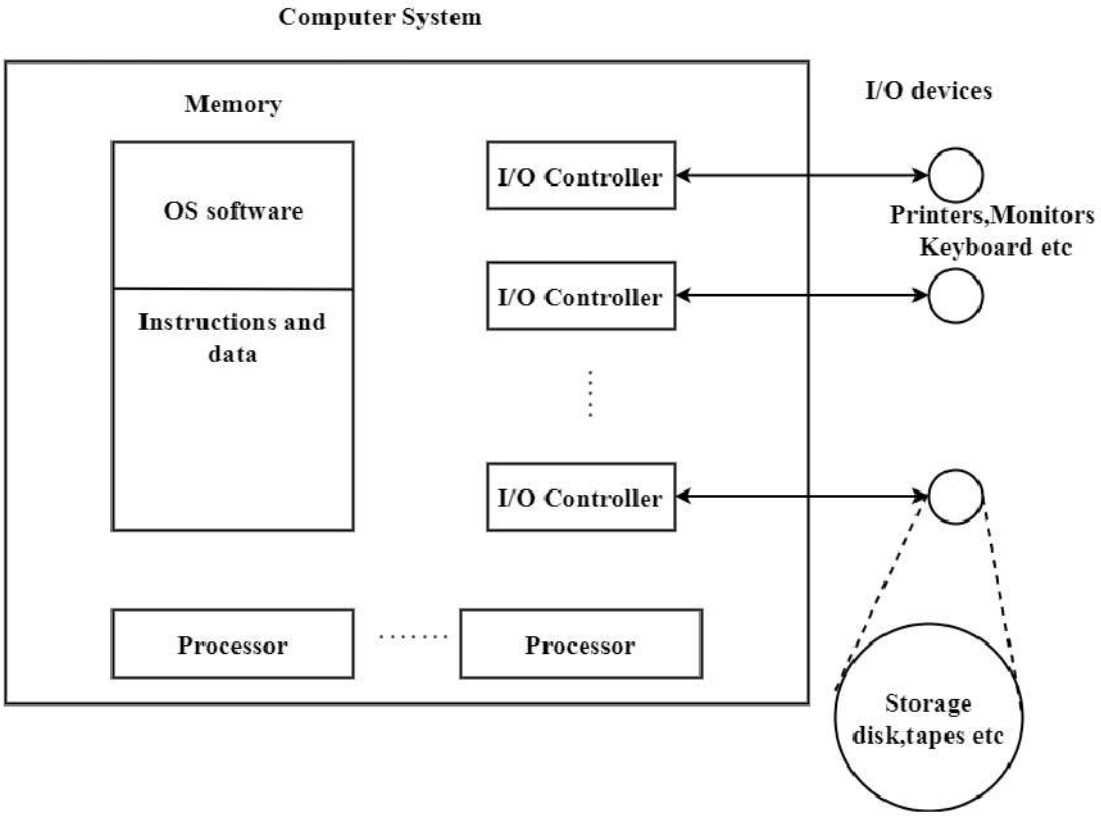

Figure 1.2: OS as Resource Manager

Figure 1.2 illustrates the main resources managed by the OS. A portion of the OS resides in the computer's primary memory. This section includes the kernel or nucleus, which contains the most frequently used functions of the OS, as well as other portions of the OS currently in use. The remaining space in primary memory holds user programs, utility programs, and data. The OS, along with the memory management hardware in the processor, collaboratively controls the allocation of primary memory, as you will learn in the following sections. The OS determines when a program in execution can access and utilize an I/O device, and it manages access to and usage of files. Furthermore, the processor itself is considered a resource, and the OS must decide how much processor time should be dedicated to executing a specific user program.

1.2.3 The Evolution of an Operating System and its Flexibility to Adapt

An operating system (OS) undergoes significant changes and improvements over time due to various factors. Let's explore some of the reasons behind the evolution of a major OS:

- Hardware advancements and new types of hardware: As technology progresses, hardware upgrades and the introduction of new hardware types influence OS development. For instance, early versions of UNIX and the Macintosh OS did not incorporate a paging mechanism because

the processors they ran on lacked paging hardware. However, subsequent versions of these operating systems were modified to take advantage of paging capabilities. Additionally, the use of graphics terminals and page-mode terminals, which differ from traditional line-at-a-time scroll mode terminals, impact OS design. For example, graphics terminals enable users to view multiple applications simultaneously through "windows" on the screen. This necessitates more advanced support within the OS.

- Introduction of new services: The OS expands its capabilities to meet user demands or fulfill the requirements of system managers. If existing tools fail to maintain optimal performance for users, the OS may incorporate new measurement and control tools to address this issue.
- Bug fixes: Every OS has its flaws, which are discovered over time. Consequently, fixes are implemented to rectify these issues. However, it's important to note that these fixes may inadvertently introduce new problems.

The need for regular updates places specific demands on the design of an OS. It should be constructed in a modular manner, meaning it consists of distinct modules with well-defined interfaces between them. Furthermore, comprehensive documentation is crucial. For large programs like modern OSs, a simple modularization approach, which involves dividing the program into modules, is insufficient. Additional measures beyond partitioning the program are necessary to ensure effective modularization.

1.3 Files and File systems

The storage and retrieval of data and programs is facilitated by the file system, which comprises two main components: a set of files that contain relevant data, and a directory structure that manages and offers details on all the files present in the system.

A file is a logical storage unit that is created by the operating system to allow users to organize and store their information in a structured and convenient way. A file is a collection of related information that is stored as a single entity on a secondary storage device. The contents of a file can be anything from text, images, videos, programs, or any other kind of digital data.

Files are an essential part of an operating system because they allow users to store and organize their data in a structured and convenient way. Without files, users would have to rely on physical media such as paper or tape to store their data, which would be much less efficient.

The operating system provides a set of services that allow users to create, read, write, copy, move, delete, and modify files. These services are provided through a file system, which is responsible for managing the physical storage of files on the secondary storage device.

Files are identified by a unique name that is assigned to them by the user or the operating system. The file name is accompanied by an extension that indicates the type of file. For example, a file with the extension ".txt" is a text file, while a file with the extension ".jpg" is an image file.

Files have attributes that determine their properties and access permissions. These attributes include the file size, date and time of creation, owner, and permissions that determine who can access and modify the file.

1.3.1 File Attributes

A file is identified by its name, for example, trial.c. Once a file is named, it becomes independent of the user, the process, and the system that created it. This means that any user can create a file, and another user can copy, edit or send it to someone else via disk or mail.
Every file has several attributes that vary from one operating system to another. However, there are some common attributes that all files share. These attributes are listed below:

• **Name**: Every file has a unique name that is in human-readable form. This makes it easy for users to identify and locate files.
• **Identifier**: This is a unique tag that identifies the file within the file system. It is usually a number and is in non-human-readable form.
• **Type**: This attribute is required for systems to support different types of files. For example, text files, image files, and executable files all have different file types.
• **Location**: This attribute points to a device and the location where the file is stored on that device. It helps users to locate and access files easily.
• **Size**: This attribute shows the current size of the file in bytes. It may also contain the maximum allowed size of the file. This helps users to manage their storage space.
• **Protection**: This attribute contains access-control information that determines who can perform reading, writing, executing, and other actions on the file. It is essential for maintaining file security and privacy.
• **Time, date, and user identification**: This attribute shows the date and time the file was created, modified, and last accessed, as well as the user who performed these actions. It is important for monitoring usage and ensuring file integrity.

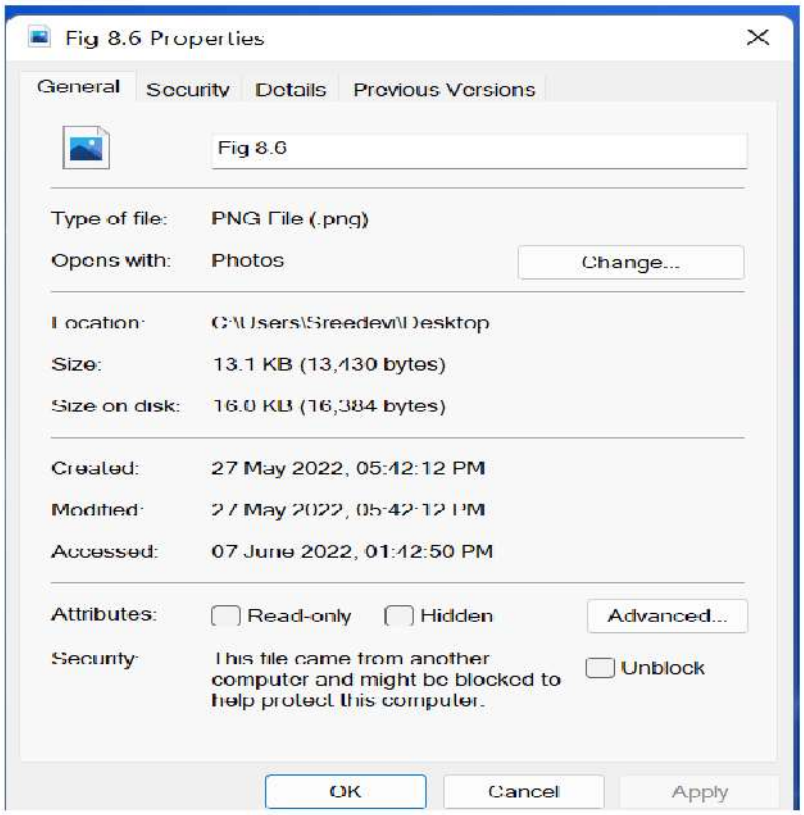

Figure 1.3: Example of File Attributes

Figure 1.3 provides an example of file attributes. In this example, the file name is Fig 8.6, the file type is a PNG file, and it opens with the Photos application. The figure also shows the file's location on the system, its size, and the date and time it was last created, modified, and accessed.

Thus, file attributes are essential for managing files and ensuring their security and privacy. Understanding file attributes is important for users, system administrators, and developers alike, as it helps them to effectively manage and manipulate files in the file system.

1.3.2 File Operations

The concept of file operations is essential in understanding how computers store and manipulate data. A file is a collection of data stored in a computer system that is identified by its name and is independent of the user or process that created it. To perform operations on a file, the operating system provides system calls that define the basic operations that can be performed on all types of files.

The six basic operations that can be performed on a file are create, write, read, reposition, delete, and truncate.

- The ***create*** operation involves finding space in the file system for the new file and entering the file in the directory.
- The ***write*** operation involves making a system call that specifies the name of the file and the information to be written onto the file. The operating system maintains a write pointer to keep track of where the next write operation will occur.

- The ***read*** operation involves making a system call that specifies the name of the file and where in memory the data of the file should be put. The operating system maintains a read pointer to keep track of where the next read operation will occur.
- The ***reposition*** operation, also known as a file seek, involves searching the directory for the file and repositioning the current file-position pointer to a given new value.
- The ***delete*** operation involves searching the directory for the named file, releasing all file space used by the file, and deleting the directory entry.
- The ***truncate*** operation allows the user to erase the contents of a file while maintaining its attributes. The length of the file is set to zero, and its file space is released.

Apart from the basic operations, other operations can be performed on a file, such as appending new information, renaming an existing file, copying a file, saving a file, sending a file, and editing an existing file. These operations are useful for managing files and organizing data in the computer system. Understanding file operations is crucial for developing software applications and managing data storage efficiently.

To perform these operations, the operating system (OS) needs to search the directory for the named file, which can be time-consuming. To address this issue, the OS keeps a small table called the open-file table that contains information about all open files. This table is checked whenever a file operation is requested, and when a file is closed, its entry in the open-file table is removed.

1.3.3 File types

To enable an OS to operate a file in a desired way, it needs to identify the type of that file. A file name typically consists of two parts: the name and the extension. The file extension usually indicates the type of the file. Table 1.1 provides a list of different file types.

Table 1.1: Different file types

File type	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

1.4 File Organization and Access

An operating system (OS) manages the computer hardware and provides services to the applications that run on it. One of the fundamental tasks of an operating system is to manage files and their associated data. In modern operating systems, files are used to store and organize information.

When a program needs to access information stored in a file, it must first load that file into memory. This process is known as file I/O (input/output). There are several ways to access the information stored in a file, but two of the most commonly used methods are sequential access and direct access.

1.4.1 Sequential Access:

Sequential access is a simple method of accessing files that involves reading or writing data linearly, from the beginning to the end of a file. This means that the operating system reads or writes data from the first record in the file and continues reading or writing in sequence until it reaches the end of the file. This method is commonly used for files that have a fixed or known size, such as text files or log files.

The advantage of sequential access is that it is easy to implement and efficient for reading or writing

large amounts of data in a specific order. However, the disadvantage is that it can be slow if you need to access data at a specific location that is not near the beginning of the file. For example, if you need to read the last line of a large text file, the operating system would have to read through the entire file to get to the end.

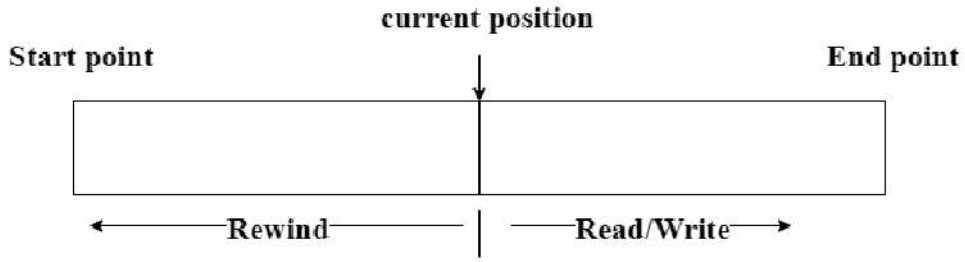

Figure 1.4: Sequential Access of File

The simplest method to access a file is the sequential access method, where records in the file are accessed in order, one after the other. This method is commonly used by editors and compilers.
The main operations on a file are reading and writing. The 'read next' operation reads the next record and automatically advances the file pointer, while the 'write next' operation appends the next record to the end of the file and advances the file pointer to the new end of the file. A sequential file can be reset to the beginning of the file, skip forward, or skip backward. This method can be used on both sequential-access devices and random-access devices. Figure 1.4 shows the sequential access method along with rewind and read/write operations.

1.4.2 Direct Access:

Direct access, also known as random access or indexed access, is a method of accessing files that allows the operating system to directly access data at a specific location in the file. This means that the operating system does not need to read through the entire file to find the data it needs.
To enable direct access, each record in the file is assigned a unique identifier, such as a record number or key. This identifier allows the operating system to quickly locate the record without having to search through the entire file. This method is commonly used for files that are accessed frequently or contain large amounts of data, such as databases.
The direct access method is based on a disk model of a file. On a disk, blocks can be randomly accessed, and a file is considered a numbered sequence of blocks or records. For example, we can read block 20, then 65, and then write block 10, with no restriction on the order of reading or writing in a direct-access file. These files are useful for accessing large amounts of information, such as databases for airline reservations or movie reservations.
The read and write operations include the block number as a parameter, such as 'read n' and 'write n', where n is the block number. The block number mentioned is a relative block number, where the first relative block is 0, the next is 1, and so on.
The advantage of direct access is that it is faster than sequential access for accessing specific records.

However, the disadvantage is that it can be more complex to implement and requires additional overhead to maintain the index or record numbers.

In summary, sequential access is good for reading or writing data in a specific order, while direct access is good for quickly accessing specific records in a large file. Direct access is also known as random access or indexed access.

1.5 File Directory and Sharing

Access methods are important for understanding how to store files on secondary storage devices such as optical disks and hard drives. After learning about access methods and file system partitioning, it's important to understand how files are stored and organized within a file system. A key component of this is the directory structure. A computer system can store an enormous number of files, and the directory structure provides a way to manage and organize these files.

With the vast amount of data that a computer can store, it's important to have a system in place for managing and organizing these files. One way to do this is through the use of a file system (FS), which can be implemented either for the entire device or for individual partitions, as illustrated in Figure 1.5.

When a device is divided into partitions, each partition is often referred to as a "mini-disk" and can have its file system. Additionally, multiple devices can be combined to create a single file system. There are several benefits to partitioning a device, including the ability to limit the size of individual file systems, support for multiple file system types on the same device, and the ability to reserve part of the device for other uses.

For each file, the directory maintains the file's name, location, size, and type.

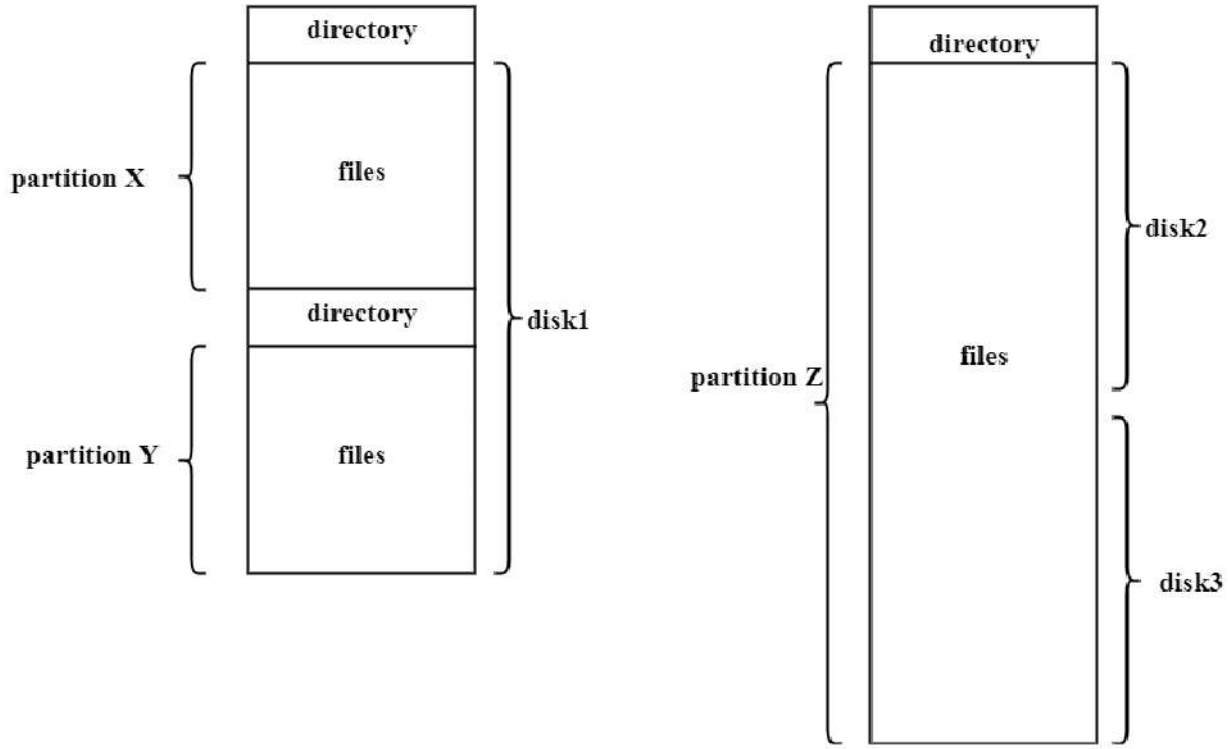

Figure 1.5 File System Organization

The directory itself can be organized in many different ways. For example, it may be organized alphabetically or by file type. Regardless of how it is organized, the directory must support various operations for managing files. These operations include:

Searching for a file: The directory can be searched to find a file whose name matches a particular pattern.

Creating a file: New files can be created and added to the directory.

Deleting a file: Unwanted files can be removed from the directory.

Listing a directory: The directory can be listed to display all files and their contents.

Renaming a file: File names can be changed if needed.

Traversing the file system: It's possible to access every directory and every file within a directory structure.

Understanding how the directory structure works is essential for managing files in an operating system. By using the directory to perform operations such as searching for files or creating new ones, users can efficiently manage their files and keep their systems organized.

Partitioning a device can be particularly useful in situations where different file system requirements exist. For example, one partition might be optimized for large multimedia files, while another might be designed for small text documents. By separating these types of files into different partitions, you can optimize the storage space for each type of file and ensure that it is stored in the most efficient manner possible. Additionally, partitioning can help to prevent data loss in the event of a system failure. If one partition becomes corrupted or damaged, the files stored on the other partitions will remain unaffected. Similarly, if a virus infects one partition, the other partitions may remain unaffected, minimizing the overall impact on the system.

1.5.1 Single Level Directory

In a single-level directory structure, all files are stored in a single directory without any subdirectories or subfolders. This means that all files are stored in the same directory, and there is no hierarchy or organization of files based on their type or purpose. In this type of directory structure, file names must be unique, since all files are stored in the same directory. This can become difficult to manage as the number of files increases, and it may become difficult to locate a particular file. Figure 1.6 shows the single-level directory.

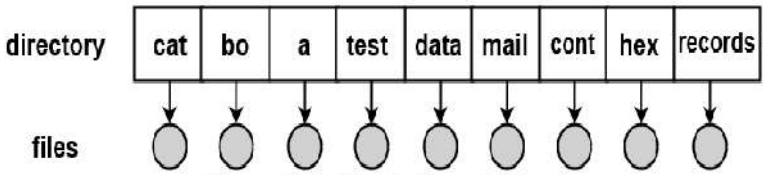

Figure 1.6: Single-Level Directory

One advantage of a single-level directory structure is that it is simple and easy to implement, especially for small systems with a limited number of files. It is also easy to manage since there are no subdirectories to navigate through. However, there are several disadvantages to this type of directory structure. As mentioned earlier, as the number of files increases, it can become difficult to locate a

specific file. It also becomes challenging to organize and manage files based on their type or purpose. Furthermore, since all files are stored in a single directory, it may become difficult to manage the permissions for each file. For example, if one file needs to be restricted to a specific group of users, it can be challenging to implement this restriction without affecting all other files in the directory. Overall, a single-level directory structure is a basic approach to file organization and management, and it may be suitable for small systems with limited files. However, for larger systems with many files, a hierarchical directory structure with subdirectories is recommended for more effective organization and management of files.

1.5.2 Two Level Directory

In a two-level directory structure, the file system is organized into two levels of directories - the root directory and the subdirectories. The root directory is the top-level directory, and all subdirectories and files are stored within it. In a two-level directory structure with User File Directories (UFDs), each user has their directory at the root level. The file names within each UFD are unique, meaning that no two files in the same UFD can have the same name. When creating a new file, the operating system checks the UFD of the user to ensure that no other file exists with the same name.

To delete a file, the operating system deletes it from the user's UFD, but not from another user's UFD. This ensures that each user's files are kept separate and protected from other users. To access another user's file, a user must be granted permission to do so. To name a file uniquely in this type of directory structure, the user's name and the file name must be specified. A two-level directory structure with UFDs can be visualized as a tree of height 2, with the root directory at the top level and each user's UFD at the second level. Figure 1.7 shows an example of this directory structure, with four users (user1, user2, user3, and user4) and their corresponding UFDs.

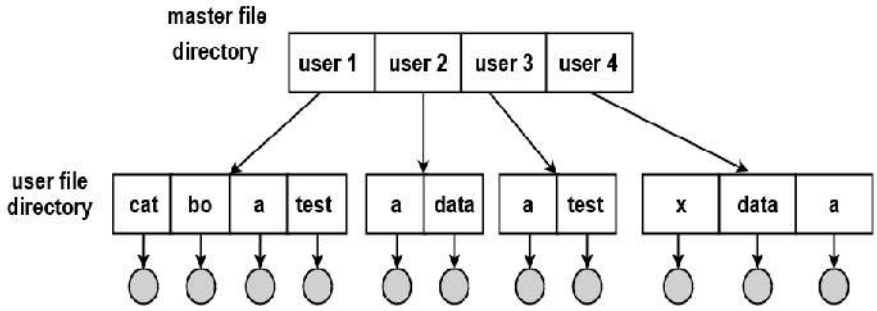

Figure 1.7: Two-Level Directory

One advantage of this directory structure is that it provides better organization and management of files for each user. Each user can manage their files without interfering with other users' files. However, managing permissions and access to files across multiple UFDs can become complex as the number of users increases. Overall, a two-level directory structure with UFDs is a suitable approach for managing files in a multi-user environment. It provides a level of isolation between users and allows for unique file names, making it an effective method for organizing and managing files in a system with multiple users.

1.5.3 Tree-structured directory

A tree-structured directory is a hierarchical directory structure in which a root directory is at the top of the hierarchy, and subdirectories can be added below it. Each directory can contain files or additional subdirectories. Users can create their subdirectories and organize their files as they wish. Figure 1.8 shows the tree-structured directory. In a tree-structured directory, each directory can have subdirectories, and each subdirectory can have further subdirectories, forming a tree-like structure. The tree starts from a root directory, which contains all the other directories and files.

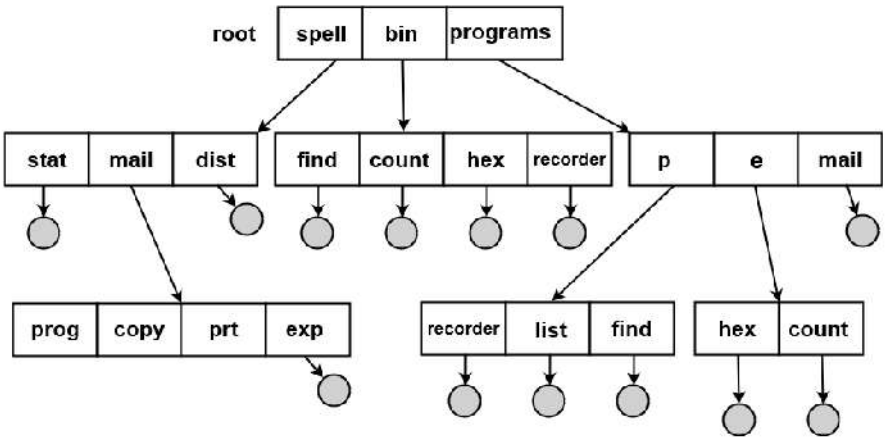

Figure 1.8: Tree-Structured Directory

Every file and directory within a tree-structured directory has a unique path name that specifies its location within the directory tree. A path name consists of the name of the root directory followed by a series of subdirectory names separated by a delimiter, such as a forward slash (/).

To delete a directory, the following conditions should be taken into account:

- If a directory is empty, it can be deleted easily.
- If the directory is not empty, different operating systems handle it differently.
  - MS-DOS will not delete a directory unless it is empty. First, all the files and subdirectories should be deleted.
  - UNIX deletes all the files and subdirectories within the directory before deleting the directory itself.

The tree-structured directory is a very common type of directory structure used by various operating systems, including Windows, Linux, and macOS. It provides an organized way of managing files and directories, making it easier for users to find and access their files. One advantage of a tree-structured directory is that it allows for a high degree of organization and flexibility in managing files. Users can create, rename, move, and delete directories and files as needed, and the structure of the tree remains intact.

1.5.4 Acyclic Graph Directories

An Acyclic Graph Directory is a data structure used in file systems to represent the organization and structure of files and directories in an operating system. It is called "acyclic" because it does not allow

for cycles or loops in the directory structure, ensuring that there are no circular references or infinite loops that can cause issues in file system operations.

In this Directory structure, files and directories are represented as nodes, and the relationships between them are represented as directed edges or pointers. Each node in the Directory can have multiple child nodes, representing directories and files that are contained within it. However, unlike traditional directory structures, this Directory allows for multiple parent nodes or links, which means that a file or directory can be linked or referenced from multiple locations in the directory structure. This allows for the creation of symbolic links, hard links, or other types of links in the file system.

It provides several benefits, including improved flexibility, scalability, and performance. For example, it allows for efficient sharing of files and directories among different users, applications, or processes, without duplicating the data. It also allows for the creation of complex directory structures with multiple levels of indirection, which can be useful in managing large and distributed file systems

In an acyclic graph, each node represents a file or directory, and each edge represents a relationship between the files or directories. In the context of file systems, an acyclic graph directory structure allows for the sharing of directories and files between multiple users. This means that a directory can exist in multiple places at once in the file system, and changes made to the directory by one user are immediately visible to all other users who have access to the same directory.

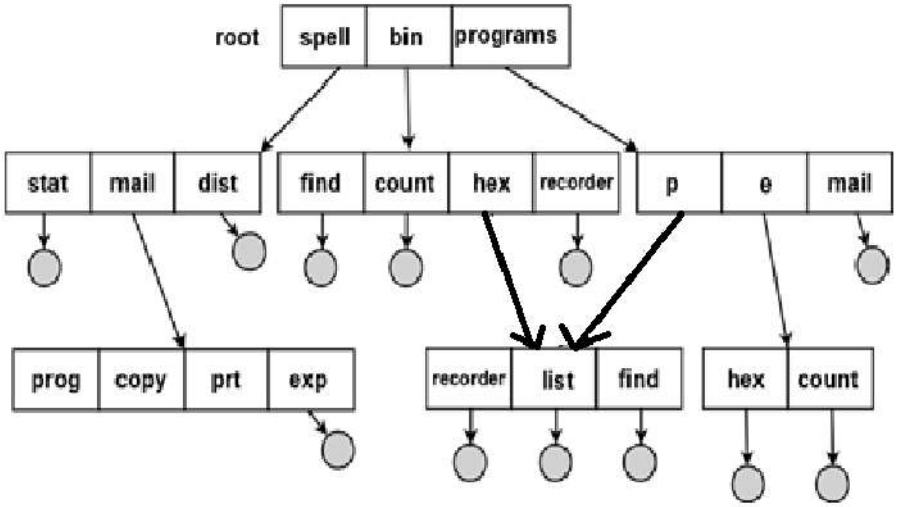

Figure 1.9: Acyclic Graph Directory

Figure 1.9 shows an acyclic graph directory in which the *list* is shared by *hex* and *p*. For example, in a shared directory used by a team of developers, all the project files can be saved in one location, and each team member can have a copy of the directory containing the project files. This allows for easy collaboration between team members, as any changes made to the files are immediately visible to all other team members who have access to the same directory.

The acyclic graph directory structure also allows for efficient storage and retrieval of files and directories, as it avoids the need for multiple copies of the same file or directory to be stored in different locations on the file system. Instead, each file or directory is represented by a single node in the graph, and the relationships between the nodes are used to determine the file's location and its relationships to other files and directories in the system.

1.6 Self-Assessment Questions

Q1. Explain the importance of the operating system in managing system resources efficiently.[4 marks, L2]
Q2. What are the three main objectives of an operating system[3 marks, L1]
Q3. List the file attributes and explain. [6 marks, L1]
Q4. Explain the operations that can be performed on files. [6 marks, L2]
Q5. Discuss the different methods of accessing files. [6 marks, L2]
Q6. Describe the directory service and different types of directories. [10 marks, L2]
Q7. What are some benefits of partitioning a device for file system management? [4 marks, L2]
Q8. What are the various operations that can be performed on a directory for managing files? [4 marks, L2]
Q9. What are the advantages and disadvantages of a single-level directory structure? [4 marks, L2]
Q10. How does a two-level directory structure with User File Directories (UFDs) work, explain with a diagram. [6 marks, L2]
Q11. Why is it important to name files uniquely in a two-level directory structure with UFDs? [2 marks, L2]
Q12. What are some challenges that may arise when managing files in a large directory structure? [4 marks, L2]

1.7 Self-Assessment Activities

A1. Develop a conceptual diagram illustrating the interaction between users, applications, and the operating system.
A2. You are a software developer working on a large project that involves multiple files. One of the files has been accidentally deleted, and you need to retrieve it from a backup. What attribute of the file would you look for in the backup system to ensure you are retrieving the correct file?
A3. You have created a new file on your computer, but it is not showing up in the directory where you expected it to be. Which file attribute might be causing this issue, and what file operation can you perform to resolve the issue?
A4. You are a data analyst working on a large dataset that is stored on a tape drive. You need to analyze the data in a specific order, from beginning to end. Which access method would be most suitable for this scenario, and why?
A5. You are a system administrator responsible for managing a file server for a large company. The file server has multiple directories, each containing files related to specific departments in the company. Which directory structure would be most suitable for this scenario, and why?

1.8 Multiple-Choice Questions

Q1. What is a file? [1 mark, L1]
A. A collection of related information that is stored as a single entity on a secondary storage device
B. A type of program that runs on an operating system

C. A set of instructions that tell the computer what to do
D. A hardware device used to store data

Q2. What does the ISA (Instruction Set Architecture) define in a computer system? [1 mark, L1]
A. The system call interface to the operating system
B. The collection of instructions in machine language that a computer can understand and execute.
C. The hardware resources and services available in a system
D. The binary portability across programs

Q3. Which part of the operating system contains the most frequently used functions? [1 mark, L1]
A. Kernel
B. Application programs
C. Utility programs
D. User programs

Q4. What is the purpose of a file extension? [1 mark, L1]
A. To indicate the type of file
B. To identify the location of the file
C. To determine the file's size
D. To indicate the file's permissions

Q5. What is the advantage of direct access over sequential access? [1 mark, L1]
A. It is more efficient for reading or writing large amounts of data
B. It is easy to implement
C. It allows the operating system to directly access data at a specific location
D. It is good for files with a fixed or known size

Q6. What is the purpose of the directory structure? [1 mark, L1]
A. To manage and organize files
B. To create files
C. To store files
D. To partition the device

Q7. What are the benefits of partitioning a device? [1 mark, L1]
A. Limiting the size of individual file systems
B. Enabling a device to work with various types of file systems simultaneously.
C. Setting aside a portion of the device for different purposes.
D. All of the above

Q8. What is the advantage of a single-level directory structure? [1 mark, L1]
A. Simple and easy to implement
B. Easy to manage since there are no subdirectories to navigate through
C. Both a and b
D. None of the above

Q9. What is a single-level directory structure? [1 mark, L1]
A. All files are stored in a single directory without any subdirectories or subfolders
B. The file system is organized into two levels of directories - the root directory and subdirectories
C. The directory structure that maintains the file's name, location, size, and type
D. A type of partition

Q10. What file attribute is used to uniquely identify a file in a file system? [1 mark, L1]

A. File size.
B. File name
C. File type
D. File permission

Q11. What is a two-level directory structure? [1 mark, L1]

A. All files are stored in a single directory without any subdirectories or subfolders
B. The file system is organized into two levels of directories - the root directory and subdirectories
C. The directory structure that maintains the file's name, location, size, and type
D. A type of partition

Q12. What is a UFD in a two-level directory structure? [1 mark, L1]

A. A type of file system
B. A type of partition
C. A user's directory at the root level
D. A type of file

1.9 Keys to Multiple-Choice Questions

Q1. A collection of related information that is stored as a single entity on a secondary storage device. (A)
Q2. The collection of instructions in machine language that a computer can understand and execute. (B)
Q3. Kernel(A)
Q4. To indicate the type of file (A)
Q5. It allows the operating system to directly access data at a specific location (C)
Q6. To manage and organize files. (A)
Q7. All of the above. (D)
Q8. Both a and b. (C)
Q9. All files are stored in a single directory without any subdirectories or subfolders (A)
Q10. File name(B)
Q11. The file system is organized into two levels of directories- the root directory and subdirectories (B)
Q12. A user's directory at the root level. (C)

1.10 Summary of the Unit

The unit discusses file and storage systems. OS plays a vital role in managing system resources, providing a user-friendly interface, and enabling efficient utilization of hardware capabilities for both programmers and applications. The operating system defines and implements an abstract data type called a "file", which consists of a sequence of logical records. Think of files as containers for information – they can hold just a tiny piece of data or a whole bunch of stuff. The OS sets the rules for what a file is and how it works. Each file is made up of parts called logical records. These records can be as small as a single

character or more complex, like a sentence or more. Now, the size of these records might not match the space they take up on the computer. It's like fitting different-sized puzzle pieces into a box. The OS helps solve this puzzle, ensuring everything fits neatly and is organized properly. It's like putting together a jigsaw puzzle to make sure all the pieces are in the right place. Directories are like organizers for files. They help keep things neat and easy to find. Imagine if everyone threw their stuff into the same drawer – it would be chaos! So, the OS introduces a two-level directory system. It's like having separate drawers for different people. Each person gets their special spot, and inside that spot, you can find details about their files – like their names, where they are, how big they are, and more.

But we can get even more organized! Imagine a tree with branches – that's a tree-structured directory. The main branches are like big folders, and the smaller branches are like subfolders. It's a great way to keep everything in order. Then there's an acyclic-graph structure that lets you share subfolders and files between different branches. It's like connecting different parts of the tree, but it can make searching and clean up, a bit more complicated. For the ultimate flexibility, there's a general graph structure. It's like a web connecting files and folders in all sorts of ways. This gives you lots of freedom to share and link things, but sometimes it can get a bit messy. So, the OS is like a skilled manager, taking care of files, keeping them organized in drawers and branches, and making sure everything runs smoothly. It's a vital part of how your computer works its magic!

1.11 Keywords

Resource manager, File system, File system partitioning, Access methods, Directory structure, File attributes, File operations

1.12 Recommended Learning Resources

[1] Silberschatz, A., Galvin, P., & Gagne, G. (2005). Operating System Concepts, 7th ed., Hoboken.

[2] William stalling. Operating Systems: Internal and design principles, 7th edition PHI

[3] D.M. Dhamdhere. Operating Systems: A concept-based Approach, 2nd Edition, TMH