

Module – 5

File System Interface: File Concept, Access Methods, Directory and Disk Structure, File-System Mounting, File Sharing, Protection, File-System Structure, File-System Implementation, Directory Implementation, Allocation Methods, Free-Space management, Efficiency and Performance, Recovery.

5.1 File Concept

File Concept Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. So that the computer system will be convenient to use, the operating system provides a uniform logical view of stored information.

- 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. From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files may be free form, such as text files, or may be formatted rigidly. In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user. The concept of a file is thus extremely general.
- The information in a file is defined by its creator.
- Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on.
- A file has a certain defined structure, which depends on its type. 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 is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters, such as `example.c`. Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not. When a file is named, it becomes independent of the process, the user, and even the system that created it. For instance, one user might create the file `example.c`, and another user might edit that file by specifying its name. The file's owner might write the file to a USB disk, send it as an e-mail attachment, or copy it across a network, and it could still be called `example.c` on the destination system.

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.

Some newer file systems also support extended file attributes, including character encoding of the file and security features such as a file checksum. The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes. It may take more than a kilobyte to record this information for each file. In a system with many files, the size of the directory itself may be megabytes. Because directories, like files, must be

nonvolatile, they must be stored on the device and brought into memory piecemeal, 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.
- Let's examine what the operating system must do to perform each of these six basic file operations.
- It should then be easy to see how other similar operations, such as renaming a file, can be implemented.

• **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 12. 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. Because a 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.

• **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

Access methods: Access methods in a file system interface define how data stored in a file can be accessed and manipulated. These methods allow programs to interact with files efficiently and provide different ways to handle data depending on the application requirements. Common access methods include:

1. Sequential Access

- **Definition:** Data is accessed in a linear sequence, starting from the beginning of the file and proceeding to the end.
- **Operations:**
 - **Read-next:** Reads the next block or record of the file.
 - **Write-next:** Appends data to the end of the file.
 - **Rewind:** Resets the file pointer to the beginning.
- **Use Cases:**
 - Text files, logs, and streaming data.
- **Advantages:**
 - Simple and efficient for sequentially stored data.
 - Works efficiently for files meant to be processed in order.
- **Disadvantages:**
 - Inefficient for random access.
 - Requires linear traversal to find specific records.

2. Direct (or Random) Access

- **Definition:** Data can be read or written in any order, accessing arbitrary blocks or records in the file.
- **Operations:**
 - **Seek (or locate):** Moves the file pointer to a specific location.
 - **Read:** Reads data from the current location.
 - **Write:** Writes data to a specific location.
- **Use Cases:**
 - Databases and indexed files.
- **Advantages:**
 - Provides flexibility for accessing specific parts of a file.
 - Efficient for large datasets where only specific parts are needed.
- **Disadvantages:**
 - May require additional overhead to manage indexing.
 - May require indexing or metadata for effective usage.

3. Indexed Access

- **Definition:** Uses an index to quickly locate data. The file maintains an index table that maps keys to specific blocks or records.
- **Operations:**
 - **Search index:** Finds the desired index entry.
 - **Access:** Uses the index to directly locate the data.
- **Use Cases:**

- Large datasets where frequent lookups are required.
- **Advantages:**
 - Fast lookup and retrieval.
 - Enables efficient management of large datasets.
- **Disadvantages:**
 - Additional storage is required for the index.
 - Overhead to maintain the index during file updates.

4. Mapped Access

- **Definition:** Maps a file to memory (via memory-mapped I/O) so that file operations are handled as if manipulating a memory array.
- **Operations:**
 - **Map:** Maps the file content to a virtual memory address.
 - **Access:** Reads or writes directly to memory.
- **Use Cases:**
 - High-performance computing and applications requiring frequent access to large files.
- **Advantages:**
 - High speed and efficiency for certain operations.
 - Reduces the need for intermediate buffers.
- **Disadvantages:**
 - Limited by system memory.
- - May require special permissions or configurations.

5. Network Access (Special Case)

- **Definition:** Accessing files over a network, often integrated into distributed file systems.
- **Operations:**
 - **Remote read/write:** Accesses files stored on a remote server.
- **Use Cases:**
 - Cloud storage and network-based file sharing.
- **Advantages:**
 - Accessibility from multiple locations.
 - Supports scalable distributed architectures.
- **Disadvantages:**
 - Dependent on network reliability and latency.
 - Latency can affect performance.

By supporting these access methods, file systems can cater to a wide range of application requirements, from simple text processing to complex database management.

Directory and Disk Structure:

Directory & disk structure deals with how to store files. A disk can be partitioned into quarters, and each quarter can hold a separate file system.

- Partitioning is useful for limiting the sizes of individual file systems, putting multiple file-system types on the same device.
- Any entity containing a file system is generally known as a volume.
- 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.

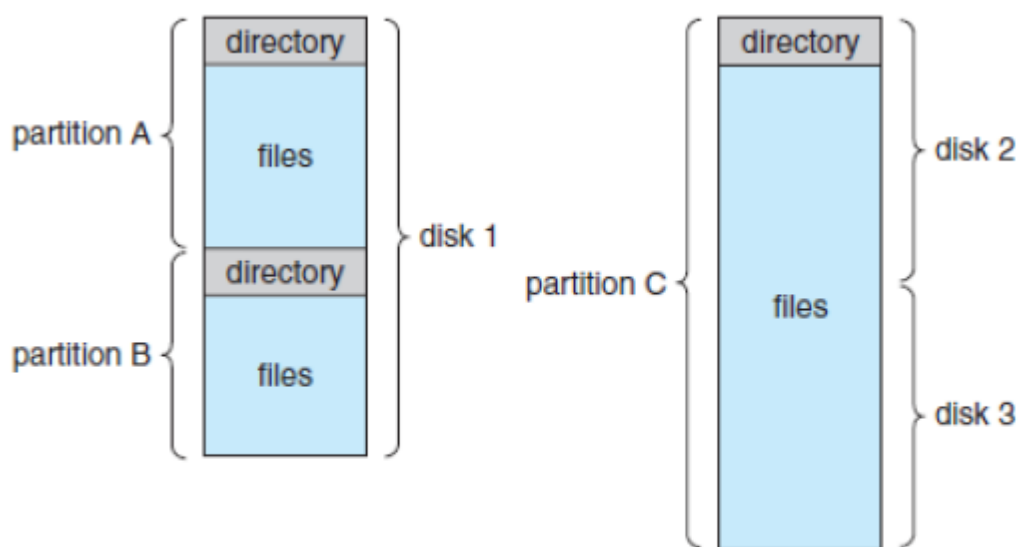


Fig: A typical file-system organization

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.
- For example, a typical Solaris system may have dozens of file systems of a dozen different types.
- tmpfs—a “temporary” file system that is created in volatile main memory and has its contents erased if the system reboots or crashes

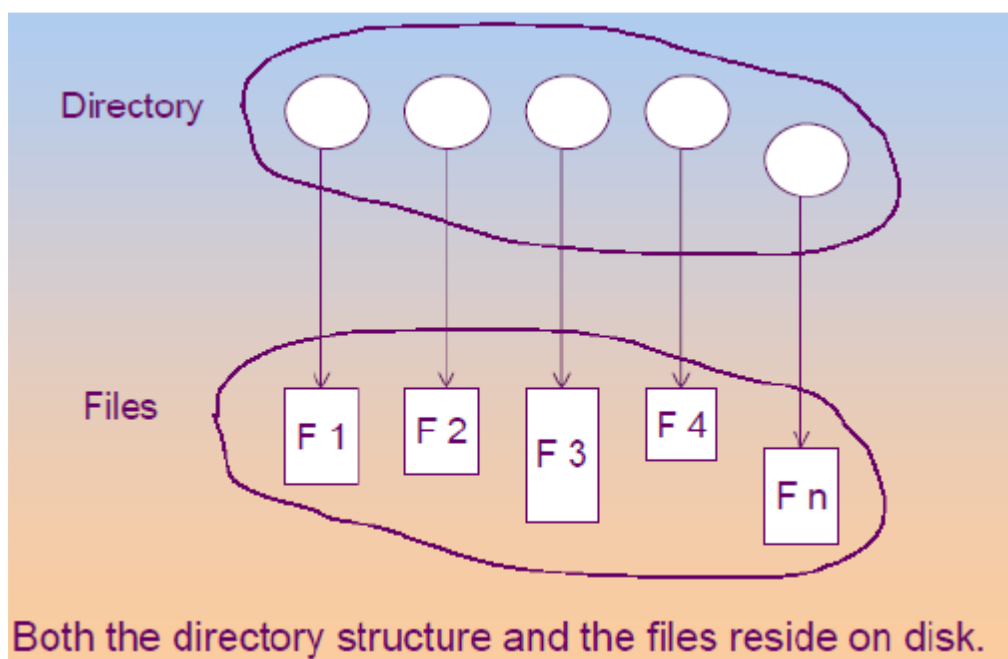
- lofs—a “loop back” file system that allows one file system to be accessed in place of another one
- procfs—a virtual file system that presents information on all processes as a file system
- ufs, zfs—general-purpose file systems.

Directory Overview Directory operations to be supported include:

- Search for a file.
- Create a file - add to the directory.
- Delete a file - erase from the directory.
- List a directory - possibly ordered in different ways.
- Rename a file - may change sorting order.
- Traverse the file system.

Directory Structure:

A collection of nodes containing information about all files.

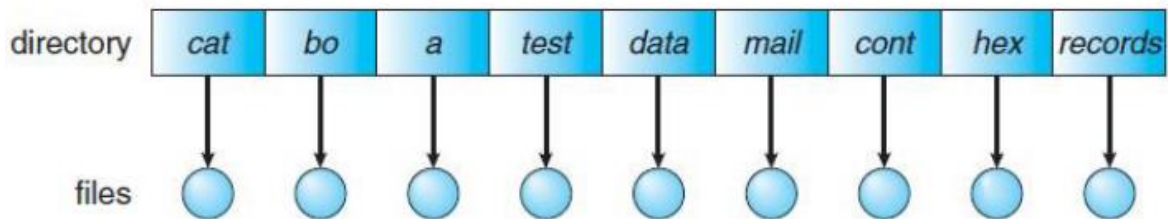


There are five directory structures. They are

1. Single-level directory
2. Two-level directory
3. Tree-Structured directory
4. Acyclic Graph directory
5. General Graph directory

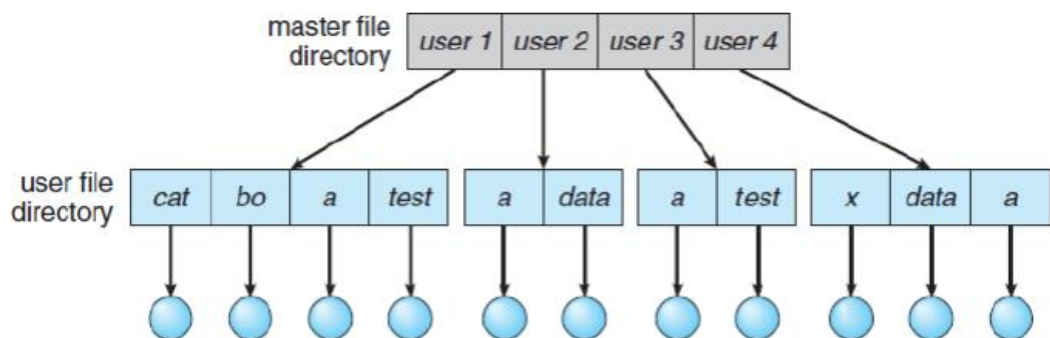
1. Single – Level Directory :

- The simplest directory structure is the single- level directory.
- All files are contained in the same directory.
- Disadvantage: 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.



2. Two – Level Directory

- Separate directory for each user.
 - In the two level directory structures, each user has her own user file directory (UFD).
 - 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.
 - The two – level directory structure solves the name-collision problem
- Disadvantage:
- Users cannot create their own sub-directories.

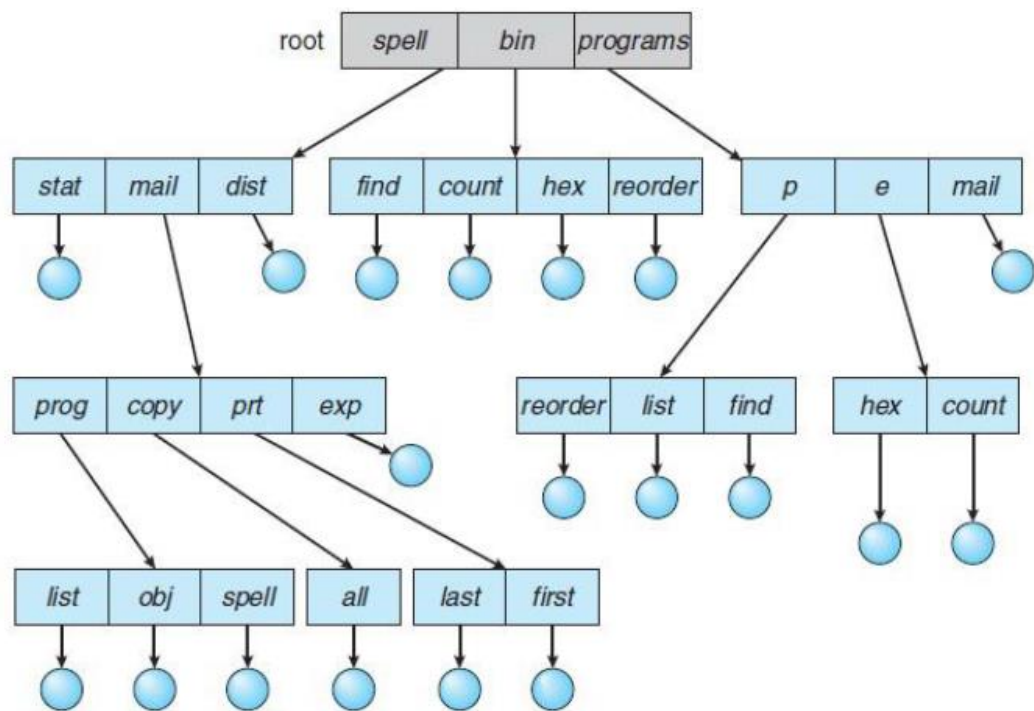


Path Name

- If a user can access another user's files, the concept of path name is needed.
- In two-level directory, this tree structure has MFD as root of path through UFD to user file name at leaf.
- Path name :: username + filename
- Standard syntax -- /user/file.ext.
-

3. Tree – Structured Directory :

- A tree is the most common directory structure.
- The tree has a root directory. Every file in the system has a unique path name.
- A path name is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- 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.
- Path names can be of two types: absolute path names or relative path names.
- 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.



4. Acyclic Graph Directory: An acyclic graph is a graph with no cycles.

- To implement shared files and subdirectories this directory structure is used.
- An acyclic – graph directory structure is more flexible than is a simple tree structure, but it is also more complex.

1. Links – A new type of directory entry is created. It is effectively a pointer to another file or subdirectory – Links are implemented as an absolute or relative path name. – The deletion of a link does not need to affect the original file; only the link is removed.

- Upon traversal of file system, do not want to traverse shared structures more than once on deletion.

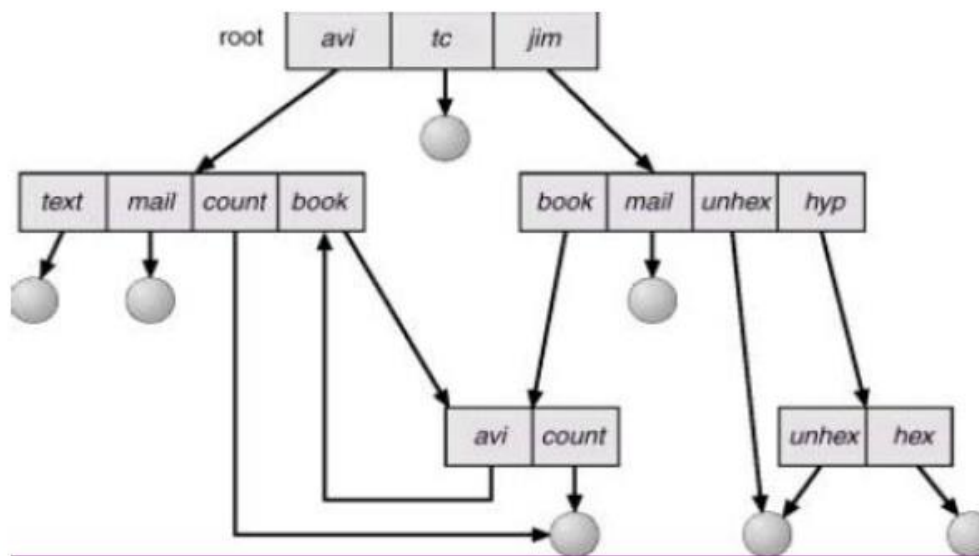
- The deletion of a link does not need to affect the original file; only the link is removed.
- Another approach to deletion is to preserve the file until all references to it are deleted. keep a count of the number of references. When count=0, file is deleted.
- Maintain a file reference list containing one entry for each reference to the file.

On deletion just delete the entry from file reference list.

File is deleted when this list becomes empty. With a shared file only one actual file exists, so any changes made by one person are immediately visible to the other.

5. General Graph Directory

- When links are added to an existing tree-structured directory, a general graph structure can be created.



- A general graph can have cycles and cycles cause problems when searching or traversing file system.
- How do we guarantee no cycles?
 - ◆ Allow only links to files not subdirectories.
 - ◆ Use Garbage collection. {computationally expensive}
 - ◆ Every time a new link is added, use a cycle detection algorithm to determine whether a cycle now exists.

Disadvantage: Computationally expensive An alternative approach is to bypass links during directory traversal.

File System Mounting in the File System Interface

File system mounting is the process of making a file system accessible to the operating system and its users by attaching it to a specific point in the directory structure. Mounting is crucial for integrating different file systems and storage devices into a single cohesive hierarchy.

Key Concepts of Mounting

1. Mount Point:

- A directory in the existing file system where the new file system will be attached.
- Example: In Linux, mounting a USB drive to /mnt/usb allows access to the USB contents through that directory.

2. Mount Operation:

- During mounting, the operating system reads the file system metadata from the storage device and integrates it into the directory tree.

3. Unmounting:

- The process of detaching a mounted file system.
- Necessary to ensure all cached data is written to the disk and avoid data corruption.

Steps in Mounting a File System

1. Identify the File System:

- The device containing the file system must be specified (e.g., /dev/sda1).
- The file system type may need to be declared (e.g., ext4, NTFS).

2. Choose a Mount Point:

- A directory where the file system will be accessible (e.g., /mnt/drive).

3. Perform the Mount Operation:

- Mount the file system to the mount point using system utilities.

1.

- Example in Linux:

```
bash
Copy code
sudo mount /dev/sda1 /mnt/drive
```

2. Access the File System:

- The files and directories in the mounted file system become accessible through the mount point.

3. Unmount the File System (when done):

- Example:

```
bash
Copy code
sudo umount /mnt/drive
```

Types of Mounting

1. Manual Mounting:

- The user explicitly mounts the file system when needed.
- Example:

```
bash
Copy code
sudo mount /dev/sdb1 /mnt/usb
```

2. Automatic Mounting:

- The operating system automatically mounts the file system during boot or when the device is detected.
- Configured using system files (e.g., /etc/fstab in Linux).

3. Lazy Mounting:

- Delays the unmount operation until the file system is no longer in use.
- Example:

```
bash
Copy code
sudo umount -l /mnt/usb
```

Mount Options

Mounting can be customized with various options to suit specific needs:

- **Read-Only:** Mount the file system in read-only mode.

```
bash
Copy code
sudo mount -o ro /dev/sda1 /mnt/drive
```

- **User Access:** Allow non-root users to access the mounted file system.

```
bash
Copy code
sudo mount -o user /dev/sdb1 /mnt/usb
```

- **File System Type:** Specify the type of file system.

```
bash
Copy code
sudo mount -t ntfs /dev/sda1 /mnt/ntfsdrive
```

Special Mount Scenarios

1. Network File Systems (NFS):

- Mount a file system located on a remote server.
- Example:

```
bash
Copy code
sudo mount -t nfs server:/remote/path /mnt/nfs
```

2. Loopback Mounting:

- Mount a file system image file (e.g., ISO).
- Example:

```
bash
Copy code
sudo mount -o loop disk_image.iso /mnt/iso
```

3. Virtual File Systems:

- File systems like proc and sysfs that provide access to system information.
- Example:

```
bash
Copy code
sudo mount -t proc proc /proc
```

Mount Table and Configuration

1. Mount Table:

- The list of currently mounted file systems.
- Linux Command: mount or cat /proc/mounts

2. File System Table (/etc/fstab):

- Configuration file that specifies file systems to be mounted automatically at boot.

Advantages of File System Mounting

- Integrates multiple storage devices and file systems into a single directory structure.
- Provides flexibility for accessing external and networked storage.
- Enables the use of different file system types on the same system.

Challenges in Mounting

- **Permission Issues:** Only privileged users may mount file systems by default.
- **Data Corruption:** Improper unmounting can lead to data loss or corruption.

- **Compatibility:** Not all file systems are supported natively on all operating systems.

Conclusion

File system mounting is an essential process in operating systems, enabling seamless access to storage devices and file systems. By understanding mounting procedures and options, users can efficiently manage storage resources and ensure system reliability.

File sharing and Protection:

File sharing is the practice of distributing or providing access to digital files between two or more users or devices. While it is a convenient way to share information and collaborate on projects, it also comes with risks such as malware and viruses, data breaches, legal consequences, and identity theft. Protecting files during sharing is essential to ensure confidentiality, integrity, and availability. Encryption, password protection, secure file transfer protocols, and regularly updating antivirus and anti-malware software are all important measures that can be taken to safeguard files. This article will explore the different types of file sharing, risks associated with file sharing, protection measures, and best practices for secure file sharing.

Definition of file sharing

File sharing refers to the process of sharing or distributing electronic files such as documents, music, videos, images, and software between two or more users or computers.

Importance of file sharing

File sharing plays a vital role in facilitating collaboration and communication among individuals and organizations. It allows people to share files quickly and easily across different locations, reducing the need for physical meetings and enabling remote work. File sharing also helps individuals and organizations save time and money, as it eliminates the need for physical transportation of files.

Risks and challenges of file sharing

File sharing can pose several risks and challenges, including the spread of malware and viruses, data breaches and leaks, legal consequences, and identity theft. Unauthorized access to sensitive files can also result in loss of intellectual property, financial losses, and reputational damage.

The need for file protection

With the increase in cyber threats and the sensitive nature of the files being shared, it is essential to implement adequate file protection measures to secure the files from unauthorized access, theft, and cyberattacks. Effective file protection measures can help prevent data breaches and other cyber incidents, safeguard intellectual property, and maintain business continuity.

Types of File Sharing

File sharing refers to the practice of distributing or providing access to digital files, such as documents, images, audio, and video files, between two or more users or devices. There are several types of file sharing methods available, and each method has its own unique advantages and disadvantages.

- **Peer-to-Peer (P2P) File Sharing** – Peer-to-peer file sharing allows users to share files with each other without the need for a centralized server. Instead, users connect to each other directly and exchange files through a network of peers. P2P file sharing is commonly used for sharing large files such as movies, music, and software.
- **Cloud-Based File Sharing** – Cloud-based file sharing involves the storage of files in a remote server, which can be accessed from any device with an internet connection. Users can upload and download files from cloud-based file sharing services such as Google Drive, Dropbox, and OneDrive. Cloud-based file sharing allows users to easily share files with others, collaborate on documents, and access files from anywhere.
- **Direct File Transfer** – Direct file transfer involves the transfer of files between two devices through a direct connection such as Bluetooth or Wi-Fi Direct. Direct file transfer is commonly used for sharing files between mobile devices or laptops.
- **Removable Media File Sharing** – Removable media file sharing involves the use of physical storage devices such as USB drives or external hard drives. Users can

copy files onto the device and share them with others by physically passing the device to them.

Each type of file sharing method comes with its own set of risks and challenges. Peer-to-peer file sharing can expose users to malware and viruses, while cloud-based file sharing can lead to data breaches if security measures are not implemented properly. Direct file transfer and removable media file sharing can also lead to data breaches if devices are lost or stolen.

To protect against these risks, users should take precautions such as using encryption, password protection, secure file transfer protocols, and regularly updating antivirus and antimalware software. It is also essential to educate users on safe file sharing practices and limit access to files only to authorized individuals or groups. By taking these steps, users can ensure that their files remain secure and protected during file sharing.

Risks of File Sharing

File sharing is a convenient and efficient way to share information and collaborate on projects. However, it comes with several risks and challenges that can compromise the confidentiality, integrity, and availability of files. In this section, we will explore some of the most significant risks of file sharing.

- **Malware and Viruses** – One of the most significant risks of file sharing is the spread of malware and viruses. Files obtained from untrusted sources, such as peer-to-peer (P2P) networks, can contain malware that can infect the user's device and compromise the security of their files. Malware and viruses can cause damage to the user's device, steal personal information, or even use their device for illegal activities without their knowledge.
- **Data Breaches and Leaks** – Another significant risk of file sharing is the possibility of data breaches and leaks. Cloud-based file sharing services and P2P networks are particularly vulnerable to data breaches if security measures are not implemented properly. Data breaches can result in the loss of sensitive information, such as personal data or intellectual property, which can have severe consequences for both individuals and organizations.
- **Legal Consequences** – File sharing copyrighted material without permission can lead to legal consequences. Sharing copyrighted music, movies, or software can result in copyright infringement lawsuits and hefty fines.

- **Identity Theft** – File sharing can also expose users to identity theft. Personal information, such as login credentials or social security numbers, can be inadvertently shared through file sharing if security measures are not implemented properly. Cybercriminals can use this information to commit identity theft, which can have severe consequences for the victim.

To protect against these risks, users should take precautions such as using trusted sources for file sharing, limiting access to files, educating users on safe file sharing practices, and regularly updating antivirus and anti-malware software. By taking these steps, users can reduce the risk of malware and viruses, data breaches and leaks, legal consequences, and identity theft during file sharing.

File Sharing Protection Measures

- **Encryption** – Encryption is the process of converting data into a coded language that can only be accessed by authorized users with a decryption key. This can help protect files from unauthorized access and ensure that data remains confidential even if it is intercepted during file sharing.
- **Password protection** – Password protection involves securing files with a password that must be entered before the file can be accessed. This can help prevent unauthorized access to files and ensure that only authorized users can view or modify the files.
- **Secure file transfer protocols** – Secure file transfer protocols, such as SFTP (Secure File Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure), provide a secure way to transfer files over the internet. These protocols use encryption and other security measures to protect files from interception and unauthorized access during transfer.
- **Firewall protection** – Firewall protection involves using a firewall to monitor and control network traffic to prevent unauthorized access to the user's device or network. Firewalls can also be configured to block specific file sharing protocols or limit access to certain users or devices, providing an additional layer of protection for shared files.

Important Questions of Module 5:

1. Illustrate file sharing mechanisms and their challenges in multi-user environments.
2. Describe the protection mechanisms used in file systems to ensure data security.
3. Define file and describe its attributes.
4. Explain the role of free-space management in file systems and its impact in efficiency.
5. Explain how directories are implemented and organized in modern file systems.
6. Discuss the operations that can be performed on files.
7. Explain the steps involved in implementing a file system.
8. Discuss the challenges faced in free-space management.
9. Describe how mounting affects the interaction between users and the file system.
10. Describe various access methods used in file systems and their applications.
11. Explain the importance of file-system recovery mechanisms.
12. Discuss the benefits and limitations of hierarchical file-system structure.
13. Explain the concept of a file and its attributes with an example.
14. Describe how files are organized and accessed in a file system.
15. Discuss the factors that affect file-system performance.
16. Explain the structure and organization of directories in modern file system.
17. Describe the concept of file-system mounting and its role in operating system.
18. Explain file access permissions and their significance.

