What is a File System?

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

1) **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.

2) **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.

3) **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.

4) **HFS (Hierarchical File System):** A file system used by macOS.

5) **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

tmpfs (Temporary File System):

tmpfs is a file system that uses a portion of the system's memory (RAM) to store files and directories. It is commonly used for temporary file storage and provides high-speed access.

XFS (X File System):

XFS is a high-performance file system designed for scalability and parallelism. It is often used in enterprise environments and supports large files and volumes.

ZFS (Zettabyte File System):

ZFS is a feature-rich file system that includes advanced data integrity and management features. It is known for its high performance, especially in environments with large amounts of data.

F2FS (Flash-Friendly File System):

F2FS is designed specifically for NAND-based flash storage, such as SSDs and eMMC. It aims to provide high performance and wear-leveling for flash memory.

Advantages of using a file system in OS

Organized Data Storage: A file system provides a structured way to store and organize data on storage devices, making it easy to locate and access files.

Efficient Data Retrieval: It enables quick and efficient retrieval of files by providing a hierarchical directory structure and indexing mechanisms.

Data Security and Permissions: File systems offer security features like access control lists and file permissions, ensuring that only authorized users can access or modify specific files.

Data Integrity and Reliability: They include features like journaling and checksums to maintain data integrity and protect against data corruption or loss due to unexpected events.

Space Management: File systems handle allocation and deallocation of storage space, preventing issues like fragmentation and ensuring optimal use of available space.

Disadvantages of using a file system in OS

Limited File Naming Conventions: Some file systems impose restrictions on file names, such as maximum length or prohibited characters, which can be limiting for users.

Potential for Fragmentation: Over time, file systems can become fragmented, meaning that files are stored in non-contiguous blocks on the storage device, potentially leading to reduced performance.

Security Vulnerabilities: While file systems provide security features, they can still be susceptible to vulnerabilities and attacks if not properly configured or updated.

Overhead for Small Files: File systems may allocate a minimum block size for each file, which can result in wasted space for small files, leading to less efficient use of storage.

Complex Maintenance: Managing a file system, especially on large storage systems, can be complex and may require regular maintenance tasks such as defragmentation, disk cleanup, and periodic backups.

File Structure in OS

A file is a logical unit of information. They are produced by processes. The operating system manages files. While creating a file, a name is assigned to it. After this process has terminated, the file exists and remains accessible to other processes. The name of a file has two parts, separated by a dot.

File Attributes in OS

File attributes are configuration and information related to files. These attributes grant/deny requests of a user/process for access, modifying, relocating, or deleting it. Some common examples of file attributes are:

- **Read-only:** Allows a file to be only read.
- **Read-Write:** Allows a file to be read and written to.
- **Hidden:** File is made invisible during non-privileged regular operations.
- **Execute:** Allows a file to be executed like a program. Other than these there are several attributes. Many of them are platform-specific.

File Access Mechanisms in OS

Direct

This method represents a file's disk model. Just like a disk, direct access mechanism allows random access to any file block.

A file is divided into a number of blocks. These blocks are of the same size. The file is viewed as an ordered sequence of these blocks.

Thus the OS can do a read-write operation upon any random block. Following are the three operations under direct mechanism:

1. Read x: read contents of block x
2. Write x: write to block x
3. Goto x: jump to block x

Sequential Access

This is a simple way to access the information in a file. Contents of a file are accessed sequentially (one record after another).

This method is used by editors and compilers.

Tape drives use a sequential method, processing a memory block at a time.

They were a form of the storage medium in computers used in earlier times.
Following are the 3 operations in sequential access:

1. Read next: read the next portion of the file.
2. Write next: add a node to the end of the file and move the pointer to it.
3. Reset: move the pointer to the starting of the file.

Indexed Access Method

This method is a variant of the direct access method. It maintains an index that contains the addresses of the file blocks. To access a record, the OS will first search its address in the index which will then point to the actual address of that block of the file that has the required record.

File Types

1. **Media:** img, mp3, mp4, jpg, png, flac, etc.
2. **Programs:** c, cpp, java, xml, html, css, js, ts, py, sql, etc.
4. **Operating System Level:** bin, sh, bat, dl, etc.
5. **Document:** xl, doc, docx, pdf, ppt, etc.
6. **Miscellaneous:** Generic text file(.txt), canvas files, proprietary files, etc.

File Types in an OS

Regular Files

Regular files consist of information related to the user. The files are usually either ASCII or binary. ASCII files contain lines of text. The major benefit of an ASCII file is that it can be displayed or printed as it is, and it can be edited using a text editor.
Binary files on printing may give some random junk content.

Directories

A directory in the filesystem is a structure that contains references to other files and possibly other directories. Files could be arranged by storing related files in the same directory. Directories are supported by both Windows as well as UNIX-based operating systems.

Character Special Files

A character special file provides access to an I/O device. Examples of character special files include a terminal file, a system console file, a NULL file, a file descriptor file, etc.

/dev/tty: Represents the controlling terminal for the process.
/dev/null: A bit-bucket that discards any data written to it and returns an end-of-file (EOF) when read.
/dev/zero: Yields an infinite stream of null bytes when read.
/dev/random: and /dev/urandom: Provide access to the kernel's random number generator.

Block Special Files

Block special files enable buffered access to hardware devices They also provide some abstraction from their specifics. Unlike character special files, block special files always allow the programmer to read and write a block of any size or alignment. Block special files are supported by UNIX-based operating systems.

Examples of Block Special Files:

/dev/sda: Represents the entire first storage device (block device).
/dev/sda1: Represents the first partition on the first storage device.
/dev/nvme0n1: Represents a NVMe storage device.
/dev/md0: Represents a software RAID device.