FILE SYSTEM

Files

- are used to provide a uniform view of data storage by the operating system. All the files are mapped onto physical devices that are usually non-volatile, so data is safe in the case of system failure.

Files Attributes

- are pieces of information associated with every file and directory that includes additional data about the file itself or its contents.

Common File Attributes

- **Name:** this denotes the symbolic name of the file. The File name is the only attribute that is readable by humans easily.
- **Identifier:** this denotes the file name for the system. It is usually a number and uniquely identifies a file in the system.
- **Type:** if there are different types of file in the system, then the type attribute denotes the type of file.

Common File Attributes

- **Location:** this point to the device that a particular file is stores on and the location of the file on the device.
- **Size:** this attributes define the size of the file in bytes, words, or blocks. It may also specify the maximum allowed file size.
- **Protection:** the protection attribute contains protection information for the file such as who can read or write on the file.

Operations on Files

- **Creating a file:** to create a file, there should be space in the file system. Then the entry for the new file must be made in the directory. This entry should contain information about the file such as its name, its location etc.
- **Reading a file:** to read a file, the system call should specify the name and location of the file. There should be a read pointer at the location where the read should take place. After the read process is done, the read pointed should be updated.

Operations on Files

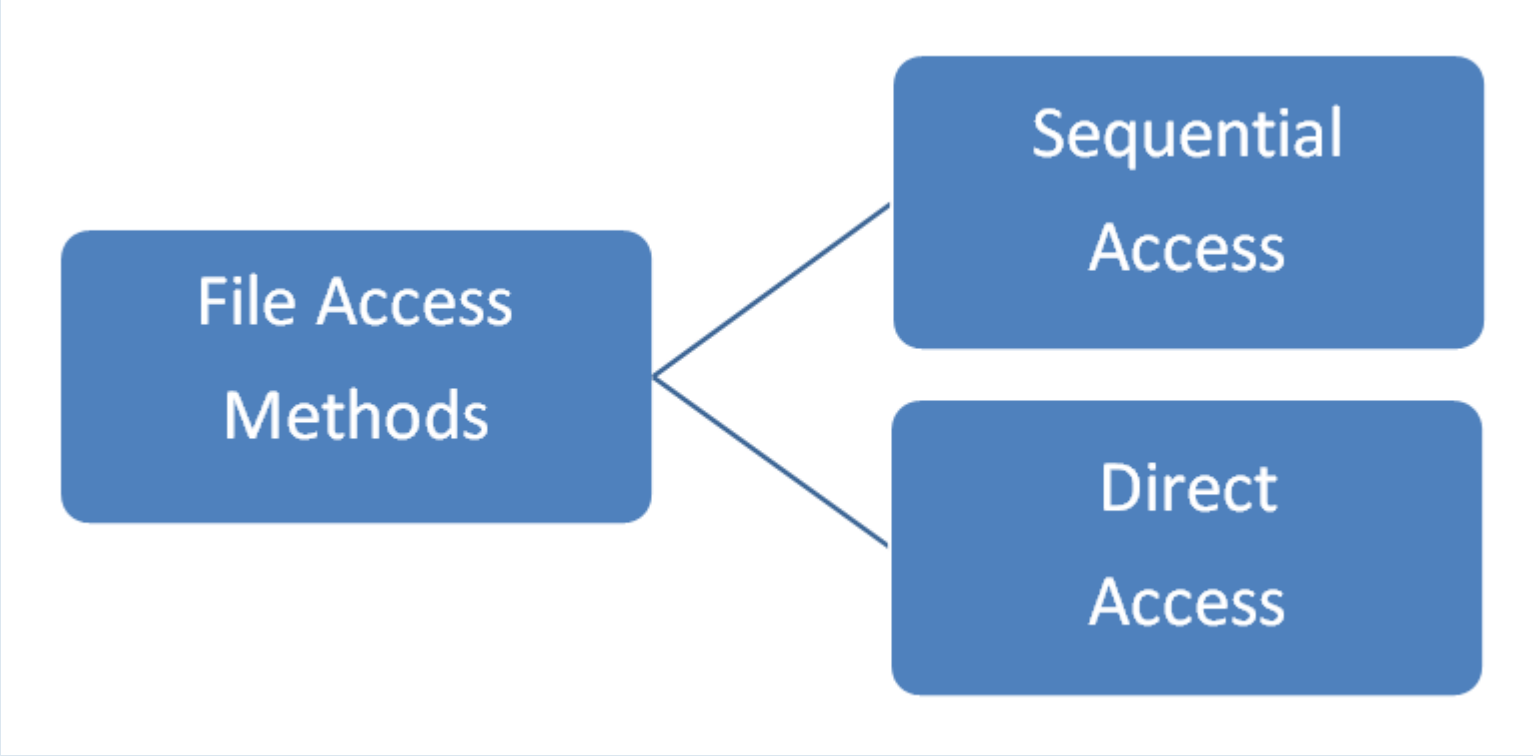
- **Writing a file:** to write into a file, the system call should specify the name of the file and the contents that need to be written. There should be a write pointer at the location where the write should take place. After the write process is done, the write pointer should be updated.
- **Deleting a file:** the file should be found in t he directory to delete it. After that all the file space is deleted so it can be reused by other files.

Operations on Files

- **Repositioning in a file:** this is also known as file seek. To reposition a file, the current file value is set to the appropriate entry. This does not require any actual I/O operations.
- **Truncating a file:** this deletes the data from the file without destroying all its attributes. Only the file length is reset to zero and the file contents are erased. The rest of the attributes remain the same.

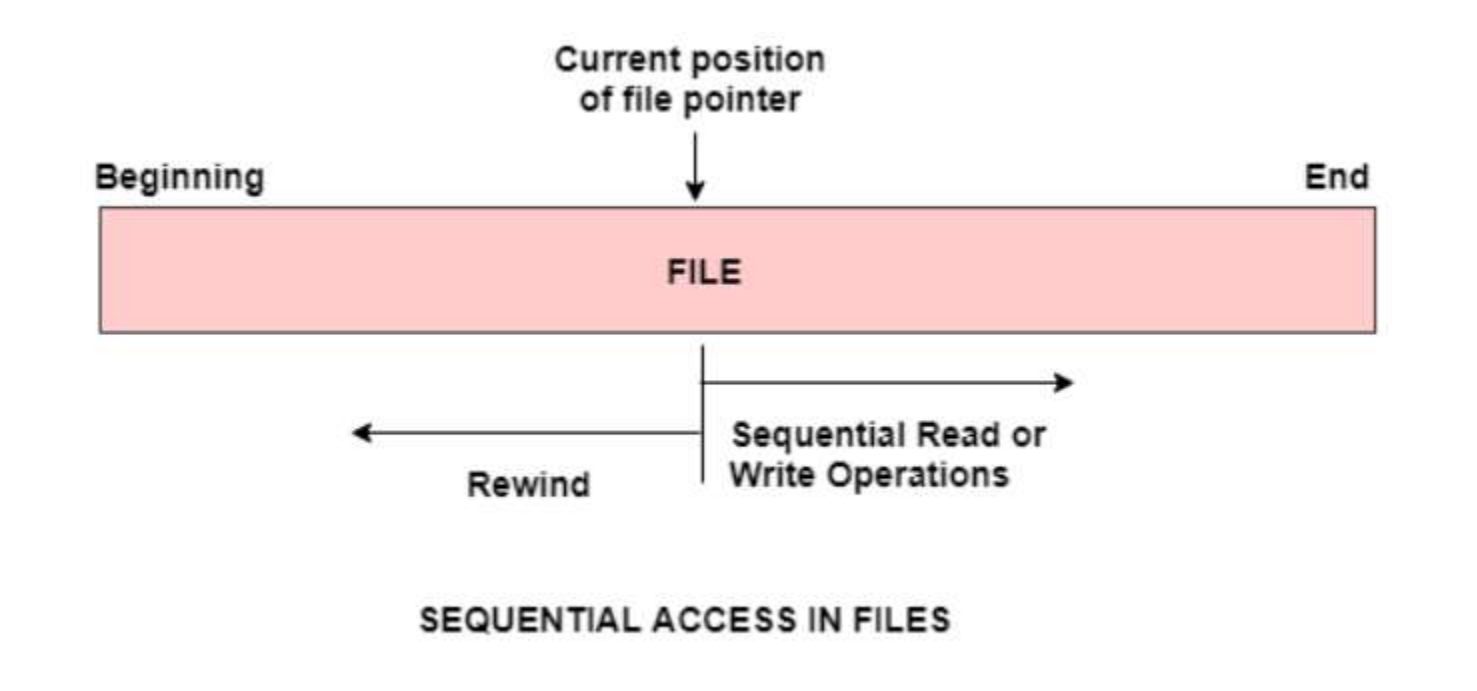
Files Access Methods

- The most common among them are using **sequential access or direct access.**



Sequential Access

- The information in a file is processed in order using sequential access. The files records are accessed on after another. Most of the file systems such as editors, compilers etc. use sequential access. It is based on the tape model of a file and so can be used with sequential access devices as well as random access devices.



SEQUENTIAL ACCESS IN FILES

File systems of Windows

Microsoft Windows employs two major file systems: **NTFS**, the primary format most modern versions of this OS use by default, and **FAT**, which was inherited from old DOS and has **exFAT** as its later extension. **ReFS** was also introduced by Microsoft as a new generation format for server computers starting from Windows Server 2012. **HPFS** developed by Microsoft together with IBM can be found only on extremely old machines running Windows NT up to 3.5.

FAT

FAT (File Allocation Table) is one of the simplest FS types, which has been around since the 1980s. It consists of the *FS descriptor sector* (boot sector or superblock), *the block allocation table* (referred to as the File Allocation Table) and *plain storage space* for storing data. Files in FAT are stored in directories. Each directory is an array of *32-byte records*, each defining a file or its extended attributes (e.g. a long name). A record attributes the first block of a file. Any next block can be found through the block allocation table by using it as a linked list.

The numbers in *FAT12*, *FAT16*, *FAT32* stand for the number of bits used to address an FS block. This means that **FAT12** can use up to 4096 different block references, while **FAT16** and **FAT32** can use up to 65536 and 4294967296 accordingly. The actual maximum count of blocks is even less and depends on the implementation of *the FS driver*.

FAT12 and *FAT16* used to be applied to old **floppy disks** and do not find extensive employment nowadays. *FAT32* is still widely used for **memory cards** and **USB sticks**. The format is supported by smartphones, digital cameras and other portable devices.

FAT32 can be used on Windows-compatible external storages or disk partitions with the *size under 32 GB* when they are formatted with the built-in tool of this OS, or up to 2 TB when other means are employed to format the storage. The file system also doesn't allow creating files the size of which exceeds *4 GB*. To address this issue, **exFAT** was introduced, which doesn't have any realistic limitations concerning the size and is frequently utilized on modern external hard drives and SSDs.

NTFS

NTFS (New Technology File System) was introduced in 1993 with Windows NT and is currently the most common file system for end user computers based on Windows. Most operating systems of the Windows Server line use this format as well.

This FS type is quite reliable thanks to journaling and supports many features, including *access control*, *encryption,* etc. Each file in NTFS is stored as a descriptor in the *Master File Table* and its data content. The **Master file table** contains entries with all information about them: size, allocation, name, etc. The first 16 entries of the table are retained for the BitMap, which keeps record of all free and used clusters, the Log used for journaling records and the BadClus containing information about bad clusters. The first and the last sectors of the file system contain its *settings* (the boot record or **the superblock**). This format uses *48* and *64 bit* values to reference files, thus being able to support data storages with extremely high capacity.

ReFS

ReFS (Resilient File System) is the latest development of Microsoft introduced with Windows 8 and now available for Windows 10. Its architecture absolutely differs from other Windows formats and is mainly organized in a form of the **B+-tree**. *ReFS* has high tolerance to failures due to new features included into it. The most noteworthy one among them is **Copy-on-Write** (CoW): no metadata is modified without being copied; data is not written over the existing data – it is placed to another area on the disk. After any modifications, a new copy of metadata is saved to a free area on the storage, and then the system creates a link from older metadata to the newer copy. Thus, a significant quantity of older backups are stored in different places, providing easy data recovery unless this storage space is overwritten.

HPFS

HPFS (High Performance File System) was created by Microsoft in cooperation with IBM and introduced with OS/2 1.20 in 1989 as a file system for servers that could provide much better performance when compared to FAT. In contrast to FAT, which simply allocates any first free cluster on the disk for the file fragment, HPFS seeks to arrange the file in contiguous blocks, or at least ensure that its fragments (referred to as **extents**) are placed maximally close to each other. At the beginning of HPFS, there are three control blocks occupying 18 sectors: the **boot block**, the **super block** and the **spare block**. The remaining storage space is divided into parts of contiguous sectors referred to as **bands** taking 8 MB each. A band has its own **sector allocation bitmap** showing which sectors in it are occupied (1 – taken, 0 – free). Each file and directory has its own **F-Node** located close to it on the disk – this structure contains the information about the location of a file and its extended attributes. A special **directory band** located in the center of the disk is used for storing directories, while the directory structure itself is a balanced tree with alphabetical entries.

File systems of Linux

Open-source Linux aims at implementing, testing and using different types of file systems. The most popular formats for Linux include:

Ext

Ext2, Ext3, Ext4 are simply different versions of the *"native"* Linux Ext file system. This type falls under active developments and improvements. **Ext3** is just an extension of **Ext2** that uses transactional file writing operations with a **journal**. **Ext4** is a further development of Ext3, extended with the support of optimized file allocation information (extents) and extended file attributes. This FS is frequently used as a *"root"* one for most Linux installations.

ReiserFS

ReiserFS - an alternative Linux file system optimized *for storing a huge number of small files*. It has good search capabilities and enables compact allocation of files by storing their **tails** or simply very small items along with metadata in order to avoid using large FS blocks for this purpose. However, this format is no longer actively developed and supported.

XFS

XFS - a robust journaling file system that was initially created by Silicon Graphics and used by the company's IRIX servers. In 2001, it made its way to the Linux kernel and is now supported by most Linux distributions, some of which, like Red Hat Enterprise Linux, even use it by default. This FS type is optimized for *storing very big files* and volumes on a single host.

JFS

JFS - a file system developed by IBM for the company's powerful computing systems. JFS1 usually stands for **JFS**, **JFS2** is the second release. Currently, this project is *open-source* and implemented in most modern Linux versions.

Btrfs

Btrfs - a file system based on the copy-on-write principle (COW) that was designed by Oracle and has been supported by the mainline Linux kernel since 2009. Btrfs embraces the features of a **logical volume manager,** being able to span multiple devices, and offers much higher fault tolerance, better scalability, easier administration, etc. together with a number of advanced possibilities.

F2FS

F2FS – a Linux file system designed by Samsung Electronics that is adapted to the specifics of storage *devices based on the NAND flash memory* that are widely used in modern smartphones and other computing systems. This type works on the basis of the log-structured FS approach (LFS) and takes into account such peculiarities of flash storage as constant access time and a limited number of data rewriting cycles. Instead of creating one large chunk for writing, F2FS assembles the blocks into separate chunks (up to 6) that are written concurrently.

The concept of "**hard links**" used in this kind of operating systems makes most Linux FS types similar in that the file name is not regarded as a file attribute and rather defined as an alias for a file in a certain directory. A file object can be *linked from many locations*, even multiply from the same directory under different names. This can lead to serious and even insurmountable difficulties in recovery of file names after file deletion or logical damage.

File System Creation Considerations

- **Device name of the file system**

File system names must be unique within a GPFS cluster. However, two different clusters can have two distinct file systems with the same name.

- **NFS V4 deny-write open lock**

You can specify whether a deny-write open lock blocks writes, which is expected and required by NFS V4, Samba, and Windows.

- **Disks for your file system**

Disks must be defined as NSDs before they can be added to a GPFS file system.

- **Deciding how the file system is mounted**
- **Block size**

The size of data blocks in a file system can be specified at file system creation by using the **-B** option on the **mmcrfs** command or allowed to default to 256 KB. This value *cannot* be changed without re-creating the file system.

File System Creation Considerations

- **atime values**

atime is a standard file attribute that represents the time when the file was last accessed.

- **mtime values**

mtime is a standard file attribute that represents the time when the file was last modified.

- **Block allocation map**

GPFS has two different methods of allocating space in a file system. The **-j** parameter specifies the block allocation map type to use when creating a file system. The block allocation map type cannot be changed once the file system is created.

- **File system authorization**

The type of authorization for the file system is specified on the **-k** option on the **mmcrfs** command or changed at a later time by using the **-k** option on the **mmchfs** command.

File System Creation Considerations

- **Strict replication**

Strict replication means that data or metadata replication is performed at all times, according to the replication parameters specified for the file system. If GPFS cannot perform the file system's replication, an error is returned.

- **Internal log file**

- **File system replication parameters**

The metadata (inodes, directories, and indirect blocks) and data replication parameters are set at the file system level and apply to all files. They are initially set for the file system when issuing the **mmcrfs** command.

- **Number of nodes mounting the file system**

The estimated number of nodes that will mount the file system may be specified at file system creation by using the **-n** option on the **mmcrfs** command or allowed to default to 32.

File System Creation Considerations

- **Windows drive letter**

In a Windows environment, you must associate a drive letter with a file system before it can be mounted. The drive letter can be specified and changed with the **-t** option of the **mmcrfs** and **mmchfs** commands. GPFS does not assign a default drive letter when one is not specified.

- **Mountpoint directory**

Every GPFS file system has a default mount point associated with it. This mount point can be specified and changed with the **-T** option of the **mmcrfs** and **mmchfs** commands.

- **Assign mount command options**

Options may be passed to the file system **mount** command using the **-o** option on the **mmchfs** command.

- **Enabling quotas**

The GPFS quota system can help you control file system usage.

File System Creation Considerations

- **Enabling DMAPI**

Whether or not the file system can be monitored and managed by the GPFS Data Management API (DMAPI) may be specified at file system creation by using the **-z** option on the **mmcrfs** command or changed at a later time by using the **-z** option on the **mmchfs** command.

- **Verifying disk usage**

The **-v** option controls whether the **mmcrfs** command checks whether the specified disks can safely be added to the file system.

- **Changing the file system format to the latest level**

You can change the file system format to the latest format supported by the currently-installed level of GPFS by issuing the **mmchfs** command with the **-V full** option or the **-V compat** option.

- **Enabling file system features**

By default, new file systems are created with all currently available features enabled.

File System Creation Considerations

- **Specifying whether the df command will report numbers based on quotas for the fileset**

You can specify (when quotas are enforced for a fileset) whether the **df** command will report numbers based on the quotas for the fileset and not for the total file system.

- **Specifying the maximum number of files that can be created**

The maximum number of files that can be created can be specified by using the **--inode-limit** option on the **mmcrfs** command and the **mmchfs** command.

- **Controlling the order in which file systems are mounted**

You can control the order in which the individual file systems are mounted at daemon startup or when using the **mmmount** command with one of the **all** keywords specified for the file system.

- **A sample file system creation**

Mounting in File System

Mounting refers to making a group of files in a file system structure accessible to user or group of users. It is done by attaching a root directory from one file system to that of another. This ensures that the other directory or device appears as a directory or subdirectory of that system.

Mounting may be local or remote. In local mounting it connects disk drivers as one machine such that they behave as single logical system, while remote mount uses NFS(Network file system) to connect to directories on other machine so that they can be used as if they are the part of the users file system. Opposite of mounting is unmounting [commands for the same will be discussed later] and it is removal of access of those files/folders. It was the overview of what a mounting is.

Managing File System

A file system is a complete directory structure, including a root directory and any subdirectories and files beneath it.

File systems are confined to a single logical volume. Some of the most important system management tasks are concerning file systems, specifically:

- Allocating space for file systems on logical volumes
- Creating file systems
- Making file system space available to system users
- Monitoring file system space usage
- Backing up file systems to guard against data loss in the event of system failures
- Making a snapshot to capture a consistent block-level image of a file system at a given point in time
- Maintaining file systems in a consistent state.

Following is a list of system management commands that help manage file systems:

Item	Description
backup	Performs a full or incremental backup of a file system
chfs -a splitcopy	Creates an online backup of a mounted JFS file system
dd	Copies data directly from one device to another for making file system backups
df	Reports the amount of space used and free on a file system
fsck	Checks file systems and repairs inconsistencies
mkfs	Makes a file system of a specified size on a specified logical volume
mount	Attaches a file system to the system-wide naming structure so that files and directories in that file system can be accessed
restore	Restores files from a backup
snapshot	Creates a snapshot of a JFS2 file system
umount	Removes a file system from the system-wide naming structure, making the files and directories in the file system inaccessible.

- **Displaying available space on a file system (df command)**

Use the **df** command to display information about total space and available space on a file system.
The **FileSystem** parameter specifies the name of the device on which the file system resides, the directory on which the file system is mounted, or the relative path name of a file system.

- **File system commands**

There are a number of commands designed to operate on file systems, regardless of type.

- **Comparing file systems on different machines**

When file systems that exist on different machines should be identical but you suspect one is damaged, you can compare the file systems.