**FILES & DISK MANAGEMENT**

**Basic File Concept**

* All data is stored in **memory** as form of ***file***.
* **Files** can be of many types including **text**, **audio** or ***object files*** etc.

**File Naming**

* File is stored on **disk** & named for identification.
* Notations for file may **vary** across **OSes**.
* File is written in form of ***file\_name.extension***, where (**.**) is period.
* **OS** reads extension to know the **type** **of file** & **type of operations** that can be performed on it.

**Types of Files**

|  |  |  |
| --- | --- | --- |
| **File Type** | **Extension** | **Work** |
| **Executable** | **exe, com, bin** | **Machine instructions input to loader & executed in main memory.** |
| **Object** | **obj, o** | **Compiled machine language.** |
| **Source** | **c, py, java, asm, 177** | **Contains sequence of procedures.** |
| **Batch** | **bat, sh** | **Commands for command interpreter.** |
| **Text** | **txt, doc** | **Textual data documents.** |
| **Word** | **wp, nx, tef** | **Word processor document.** |
| **Archive** | **arc, zip, tar** | **Compressed files.** |

**File Type Categorisation (Summary)**

* Ordinary file
* Directory file
* Special file
  + Character device file
  + Block device file

**File Type Categorisation (Brief)**

Ordinary file:-

* Also known as ***regular file***.
* Data in it are just **stream of characters**.
* It **can’t** contain another **file** or ***directory***.
* Can either be a **text** or ***binary file***.
* **High-level** source code & **binary image files** are too **ordinary files**.
* **Binary file:** Contains ***unprintable characters*** like **newline terminators** unlike text files.
* Can be made & modified through **text editors**.

Directory file:-

* Basically **folders**.
* They **don’t** contain data but just **references** to those data.

**Special Files**

* Also known as ***device files***.
* Files used in **communicating** with devices.
* Special files contain **location**, **type** & **access mode** for a device.
* **Block device file:** Used to access **block device** I/O.
* Device files in **GNU/Linux** are created using ***mknod*** command.

**File Attributes**

* **Name**
* **Identifier:** A unique tag number to identify a file.
* **Type:** File type
* **Location:** Pointer to file location.
* **Size**
* **Protection:** Manages user’s access rights.
* **Time, date & user ID**
* These information are kept in directory structures.

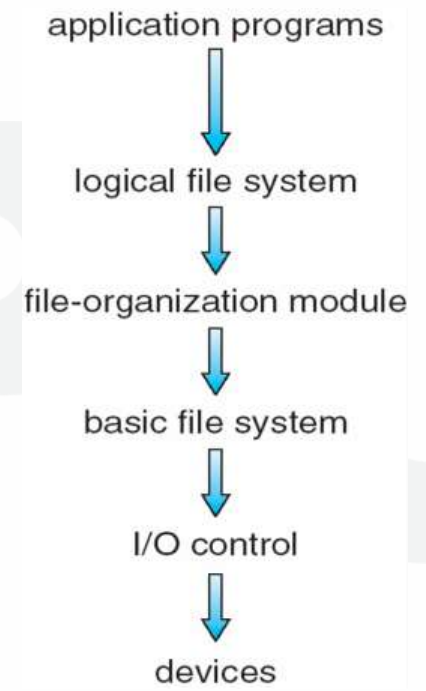
**File Operation**

* ***Create***, ***delete***, ***open***, ***close***, ***read***, ***write***, ***rename***, ***append***, ***truncate*** & ***file*** ***seek*** etc.
* **File seek:** Moving cursor within the file.

**Features of File Structure**

* Unstructured **sequence of bytes** to keep them flexible.
* A custom & **fixed length structure** within a file.
* Keeps tree of records.

**Layered File System**



* Just think it from bottom to top regarding how you **pressing a keyboard key** affects **game**, an **application program**.

**File Access Methods**

* File organisation is done using **I/O control system**.

**Types of Access Methods (Summary)**

* Sequential access
* Index sequential method
* Direct/ random access

**Types of Access Methods (Brief)**

Sequential access:-

* Each **file** is read **one-by-one** in sequence, as well as the **content** in it is read **byte-by-byte**.
* ***Read*** operation **advances** the **file pointer** by one byte.
* ***Write*** operation **appends** the **material** to be added at the **end of file**.
* **Compilers** & **text editors** access files in this order only.
* But one **disadvantage** of this technique is that it’s an **ineffective** **search technique**.

Index sequential method:-

* In this method, **files** are given **index numbers**.
* ***Index file*** contains groups of these **index numbers**.
* When a file is to be found, its **index** is searched in that **index file**.
* Then when it is found, the **file** is accessed with the **pointer** mapped to that **index**.
* To find a particular entry, that **mapped pointer** is **moved** through the **contents of file**.
* But **disadvantages** are that the **index files** may occupy large portion of memory & the **main index** **file** may contain **index** to **smaller index files**.

Direct access method:-

* Also known as ***random access method***.
* Files are accessed after **flag** is set to **read/write** before reading or writing.
* And file is treated as a **block number**, meaning starting from a particular block of memory.
* Can **directly** access the record, basically how **hard disk** allows users to do so.
* There is **no restriction** in order of reading or writing.
* **Relative block number:** Block number provided by the user to **OS**, which starts from **0** to **1**.
* But **disadvantage** is that it **doesn’t** utilizes I/O devices properly & consumes a lot of CPU time.

**Directory**

* ***File system*** maintains **directories/ folders**.
* (**.**) means ***current directory*** & (**..**) means ***parent directory***.
* While (**/**) is ***root*** in **GNU/Linux**.

**Operations on Directory**

* ***File searching***, ***creation***, ***deletion***, ***renaming*** & ***listing of directories***.

**Structures of Directory (Summary)**

* Single-level directory
* Two-level directory
* Tree structured directory
* Acyclic graph directory
* General graph directory

**Structures of Directory**

Single-level directory:-

* **All files** are stored in **one** directory.
* **Files** in **same** directory must have **unique name** (file name includes the **extension**).
* File names in **MS-DOS** can be **at max 11 characters long**, while in **UNIX** systems it can be of **255 characters long**.

Two-level directory:-

* It solves the **"unique name only"** problem.
* Each user has their **own directory** called ***user file directory*** (**UFD**).
* File is searched as per **which user** has **logged in** the system.
* **Master file directory:** Contains **entries** of user accounts which are **mapped** to their respective **UFD**.
* Though files in **one UFD** must be **unique**.
* Users **can’t** share their files to **other users** within the system.
* However, this structure **isn’t** ideal when maintaining **large number of files**.

Tree structure directory:-

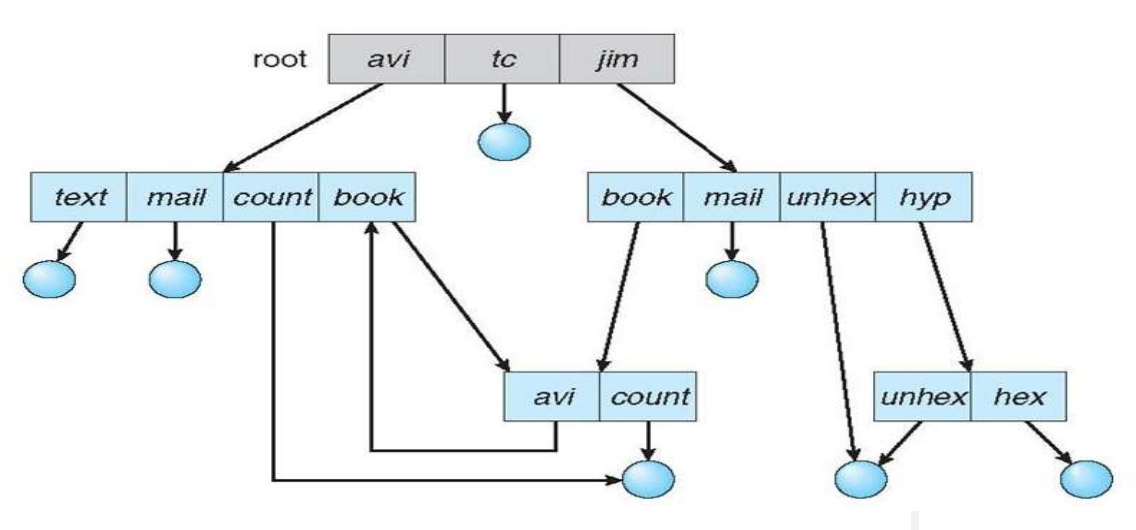
* Users can create their **own** directories & subdirectories.
* **Absolute path:** Path from **root** **directory** to a specific file.
* **Relative path:** Path from **current working directory** to a specific file.
* If user tries to delete directory or sub-directory which **isn’t** empty, then it **won’t** be removed.
* Files & directories **can’t** be shared.

Acyclic graph structure:-

* It is similar to **tree structure directory** but files & directories can be **shared** here.
* Also, a file or sub-directory maybe **linked** to **multiple directories**.
* But one disadvantage is that during **deletion**, files are linked to multiple directories.

**General Graph Directory**

* Unlike **acyclic graph structure**, it is **cyclic**.
* Means **same** file or directory linked in **two** different sub-directories are linked to another file or sub-directory further.



* Take the sub-directory ***book*** for example.

**File Sharing & Protection**

* Owner or creator must be able to provide **accessibility** to who it wants to.
* Types of access – **Read**, **write** & **execute**.
* Some computers provide **backup** & **hardware safety** to avoid physical damage.
* **Access control:** Control on which users can access what in which way.
* Controlled accesses – **Rename**, **repositioning** & **truncate**.

**Access List**

* **Access list:** Provides access to files based on their identity.
* Only allowed users can access the file.
* ***ls -l*** command provides **long details** about the file.

**Groups**

* It’s a **protection system**.
* Group classification – **Owner**, **group** & **universe/others**.
* **Owner:** User who created the file.
* **Group:** Users who are **accessing** & **sharing** the file.
* **Universe/others:** Other users in the system.
* Possible operations – **Read-r**, **write-w** & **execute-x**.
* Value for **r** is **4**, **w** is **2** & **x** is **1**.
* A **group** is attached to a file so that **each** member can **access** that.

**File Allocation Method**

* Files are stored in **free blocks** in disk.
* A block size is **512 bytes**.

**File Allocation Method (Summary)**

* Contiguous allocation method
* Linked allocation method
* Indexed allocation method

**File Allocation Method (Brief)**

Contiguous allocation method:-

* First block location to **store file** is allocated **randomly** from disk.
* Then rest of the blocks (if required) are **contiguous** to it.
* ***Contiguous directories*** contain **filename**, **start block** & **length**.
* It supports both **sequential** & **direct access**.
* However, its prone to **fragmentation** for **not** being continuously ordered in disk.
* Hence, **compaction** techniques are used.

Linked allocation method:-

* Also known as ***chain allocation***.
* Just like **linked lists**, blocks are allocated **randomly** & **linked** with each other.
* ***Linked directories*** contain **filename**, **start block** & **end block**.
* So, it’s uses **sequential access** only.
* Disadvantage is that **pointer** occupies large part of **memory** & is hard to maintain.

Indexed allocation:-

* For **all** part of files, various **pointers** are present.
* These **pointers** are kept in a **single block**.
* Each file has their own **index block**.
* ***Index directories*** contain **filename** & **index block**.
* It supports only **direct access**.

**Memory & Free Space Management**

Methods used:-

* **Bit map:** **Free blocks** are represented with **0** & **busy ones** are **1**.
* **Linked list:** All **free blocks** are **linked** with each other.

**File System Management Methods (Summary)**

* On-disk structure
* In-memory structure

**File System Management Methods (Brief)**

On-disk structure:-

* Disk contains **information** about all files, **directories**, **empty blocks** & **occupied blocks** etc.
* Let’s discuss some of these structures there.
* **Boot block control:** A block containing information about **how to boot an OS** (only if disk contains the OS).
* **Volume control block:** Contains details about **blocks** & the way they are **partitioned** across the memory.
* **File control block (FCB):** Store information related to a file in it.

In-memory structure:-

* This structure **enhances performance** of file system via **caching**.
* **Directory structure cache:** Holds information about recently accessed directory.

**Directory Implementation**

Methods used:-

* Linear list
* Hash map

Linear list:-

* Maintains a **linear list** of filenames in a directory.
* New files are added at the **end** of the list.
* Uses **cache** to store recently used files.
* If list is **sorted**, then **binary search** can be applied to save time.

Hash table:-

* Same as **linear list** but **hash table** is used too.
* Returns a **pointer** to filename when its created.
* It helps decrease the **search time** for the user when searching for a file through its name.
* When the filename is searched, the pointer to its name is **immediately returned**.
* Also, there are mechanisms to **avoid collision** between files with **same name** but in different directories.
* This because files with **same name** may point to **same locations** as the location in this method is decided as per **filename**.
* To **avoid collision**, when **hash values** point to same location, then a **linked list** is created at that location instead.

**File System Recovery (FSR)**

* A program called ***consistency checker*** checks **consistency** of files during **reboot**.
* It compares data in **directory structure** to data in **blocks of disk**.
* Not only it finds inconsistencies but also **fixes it**.

Signs of consistency:-

* All files, directories & sub-directories etc are arranged the **same way** without user interference.
* In case the **system crashes**, a process either **completes** or is **rolled back** to initial stage, displaying **atomicity**.

**Methods of FSR (Summary)**

* Backup & restore
* Log-structured file system

**Methods of FSR (Brief)**

Backup & restore:-

* Used in case the **magnetic disk** fails.
* Files are backed up & stored in another storage device.
* Files copied are tagged with **date** & **time**, indicating when **backup** was performed the last time.

Log-structured file system:-

* It gives **better performance**.
* Writes changes made in file to **cache**.

**Virtual File System (VFS)**

* A program which acts as a bridge between **kernel** & **foreign file system**.
* This allows the client application to **access local & servers** etc, which are different types of file systems.