Implementing File Systems

- File systems provide mechanism for on-line storage & access to file contents (data + program).
- File systems reside permanently on secondary storage medium — the disk.
- Disks are convenient because :-
  → It can be rewritten in place — read block from disk, modify it, write it back into same place.
  → It can access directly any block of info it contains either sequentially / randomly and switch from 1 file to another involves only moving read & write pointer and the disk to rotate
  (imagine a gramaphone)
- Disks are broken into equal size blocks to improve I/O efficiency, I/O transfer bet^n itself & memory.

I) Layered File System (File System Structure)

Application progs
↓
Logical file system
↓
file - organization module
↓
basic file system
↓
I/O control
↓
Devices

- In the figure, each level in the design uses features of lower levels to create new features for use by higher levels.

- File systems :- → Provide efficient & convenient access to disk by allowing data to be stored, located & retrieved easily.

→ Problems in file systems :-

* How will the file system look to the user?
* How to define a file & its attributes, operations allowed on it, directory structure to organize files.
* How to create algo & DS to map the logical file system onto physical secondary storage device.

- I/O control level :- → It consists of device drivers and interrupt handlers to transfer info betⁿ main memory & disk system.

→ Device driver is basically a translator whose i/p are high level commands (in english) & o/p consists of low level h/w specific instructions that are used by hardware controller which interfaces the I/O device to rest of the system.

Device drivers therefore write specific bit patterns to special locations in the I/O controller's memory to tell it which device location to act on & do what.

- Basic file system :- → It issues generic commands to appt. device driver to read & write physical blocks on the disk
→ Each physical block is identified by its numeric disk address.
→ It also manages memory buffers & caches that hold various file systems, directory & data blocks.

- File organization module :-
→ It knows about files & their logical blocks as well as physical blocks.
→ ∴ By knowing the type of file allocation used & location of file, this module translates logical block address to physical block address for basic file system to transfer.
→ It also includes a free space manager to track unallocated blocks & provide these to file-organization module when required.

- Logical file system :-
→ It manages metadata info. (metadata means data about data ∴ does not include actual data)
→ It also manages the directory structure to provide the file org. module with the info it ne given a symbolic file name.
→ It maintains file structure via file control blocks (FCB) that contains all info about the file.

Adv of layered structure :- • Duplication of code is minimized. (so memory, I/O control & basic file system can be used by multiple file system)

• Each file system has its own logical file system & file org. module.

Disadv of layered structure :- • more overhead
• how many layers to use?
• What should each layer do?
• How to design new system?

II) File System Implementation :-

A) Overview :-

• Several on disk & in memory structures are used to implement a file system.

• These structures vary depending on OS & file system.

i) On disk file system has various info such as

a) boot control block (per volume) –
→ contain info. needed by system to boot an OS from that volume.
→ If disk does not contain OS, this block is empty
→ ∴ It is first block of volume (called boot block in Unix FS & partition boot sector in NTFS)

b) volume control block (per volume)
- contains volume / partition details like no. of blocks in it, size of blocks, free block count, free block pointers etc
(In unix, it is called superblock
In NTFS, " " " master file table)

c) Directory structure (per FS)
to organise file.

d) A per file FCB to contain details of file. It has a unique identifier no. to allow association with directory entry.

FCB ⇒

file permissions
file dates (create, access, write)
file owner, group
file size
file data blocks / pointer to file data blocks

ii) In-memory info is used for both file system management of performance improvement via caching.
∴ Data is loaded at mount time, updated by file system operations & discarded at dismount.
∴ We have
a) mount table - has info about each mounted volume

b) directory structure cache holds directory info of recently accessed directories.

c) System wide open file table contains a copy of the FCB of each file

d) per process open file table contains a pointer to appropriate entry in System wide open file table for a particular process.

e) Buffers hold file system blocks when they are being read / written

* Creation of a new file :-
  - An applicat^n program calls the logical file system.
  - logical file system knows the format of directory structure : to create a new file, it allocates a new FCB.
  - System then reads the appt. directory into memory, updates it with new file name & FCB, and writes it back to the disk.

* How to use a newly created file for I/O :-
  - First the file must be opened
  - open() passes file name to logical file system.

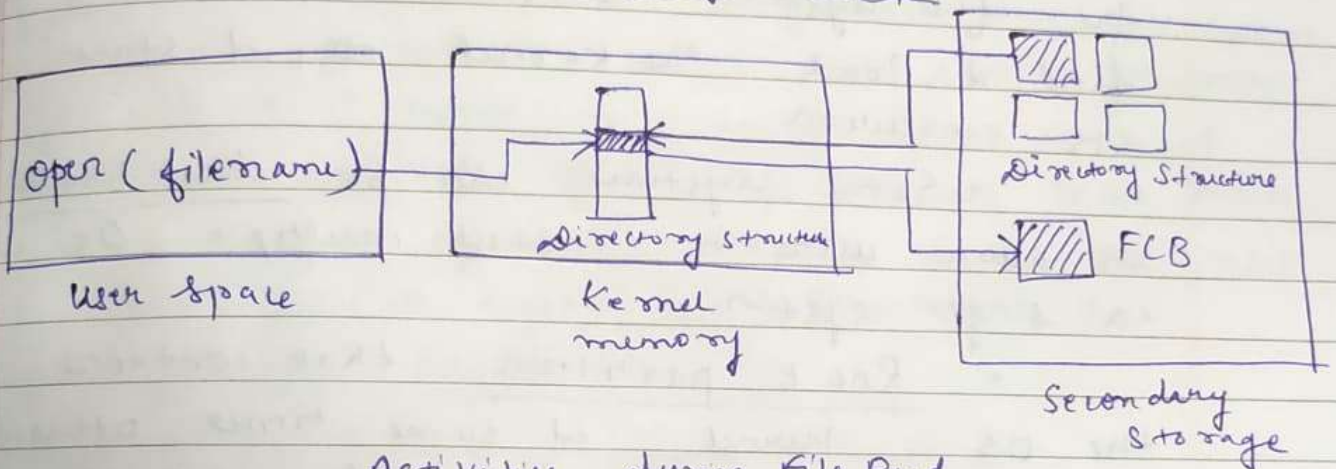
- It next first searches the system wide open file table to see if file is already in use by another process
- If it is, a per process open file table entry is created pointing to the existing system wide open file table.
- If not already open, directory is searched for given file name (use cache)
- Once file is found, FCB is copied into system wide open file table in memory. It helps to track the no. of processes that have the file open.
- Next entry is made in per process open file table with pointer to system wide file
- It may also include a pointer to current location in file for read() and write() & info about access mode of file
- Finally the open() returns a pointer to appr. entry in per process file system table.
- All further operations is via this pointer & file name is of no use anymore
- The entry is also cached to save time next time.

This is basically file descriptor or file handler.

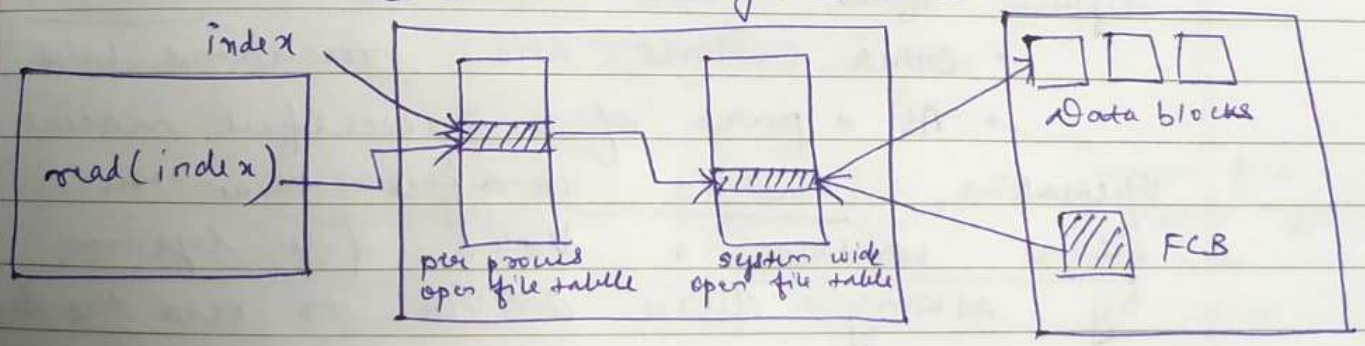
* close() removes an entry from the per process table & system wide entry's open count is decremented. Finally when it is 0, system wide entry is also removed.

~~Partition & mounting :-~~

Activities during File open



Activities during File Read



B) Partitions & mounting :-

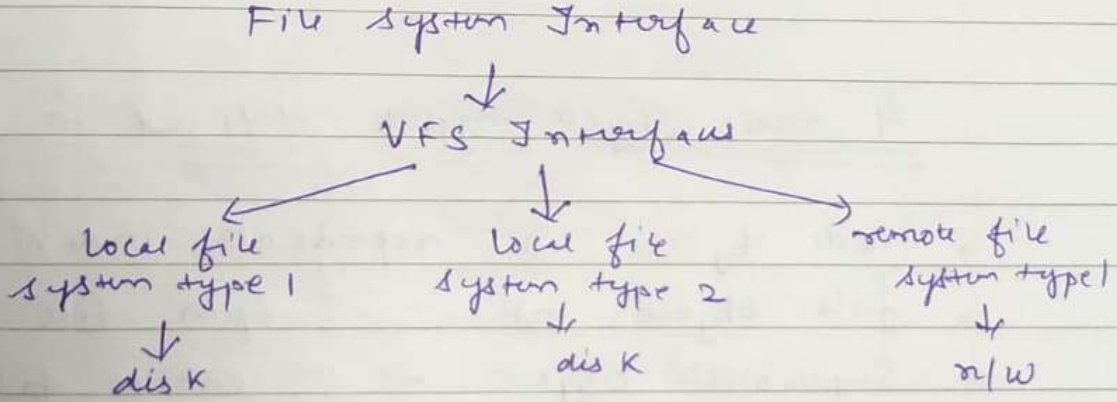
- Disk can be partitioned into multiple slices
- Each partition can be
  - raw : containing no file system
  - cooked : containing a file system
- Raw disk is used when no file system is appt.
- Boot info. is stored in a separate partition. It has its own format

as at boot time the system does not have any file system code loaded & ∴ cannot interpret the file system format.

- Boot Info is a sequential series of blocks loaded as an image into memory
- Boot loader knowns enough about the file system structure & is able to find & load the kernel & start its execution
- Some systems can be dual booted to allow users to install multiple OS on a single system.
- Root partition that contains the OS kernel & some time other system files are mounted at boot time.
- Other volumes are mounted later.
- As a part of successful mount operation, the OS verifies that the device contains a valid file system by asking device driver to read the device directory & verify its format.
- If invalid format is found, consistency is checked & corrected
- Finally OS notes in its in-memory mount table that a file system is mounted along with the type of the file system

c) Virtual File System

- To implement multiple types of file systems, directory and file routines for each type are written.
- However most OS use object oriented techniques to simplify, organize & modularize implementation.
- This enables very dissimilar file system types to be implemented within the same structure.
- ∴ Users can access files contained within multiple file systems on local disk or on file systems available across the n/w.
- File system implementation has 3 major layers



- The first layer of file system interface based on open(), read(), write(), close() system calls and on file descriptors.

- Second layer of Virtual File System (VFS) has 2 functions

a) It separates file system generic operations from their implementation by defining a clean VFS interface. It enables ∴ transparent access to different types of file systems mounted locally.

b) It provides a mechanism for uniquely representing a file throughout a n/w. VFS is based on a file representation structure called vnode that contains a numerical designator for a n/w wide unique file.

∴ VFS distinguishes local file from remote ones & local files are further categorized as per their types.

- The layer implementing the file system type / remote file system protocol is the third layer.

4 main Object types defined in Linux VFS

- inode object → represents individual file
- file object → " open file
- Superblock object → " entire file system
- dentry object → " individual directory entry

List of operations for file objects

- int open( ) → open a file
- int close( ) → close an already open file
- ssize_t read( ) → Read from a file
- ssize_t write ( ) → Write to a file
- int mmap ( ) → memory map a file

∴ VFS s/w layer can perform an operation on one of the objects

III) Allocation Methods :-

Agenda is how to allocate space to these files so that disk space is utilized effectively & files are accessed quickly.

a) Contiguous Allocation :-

- Each file occupy a set of contiguous blocks on the disk.
- Disk address :: define a linear ordering on the disk.

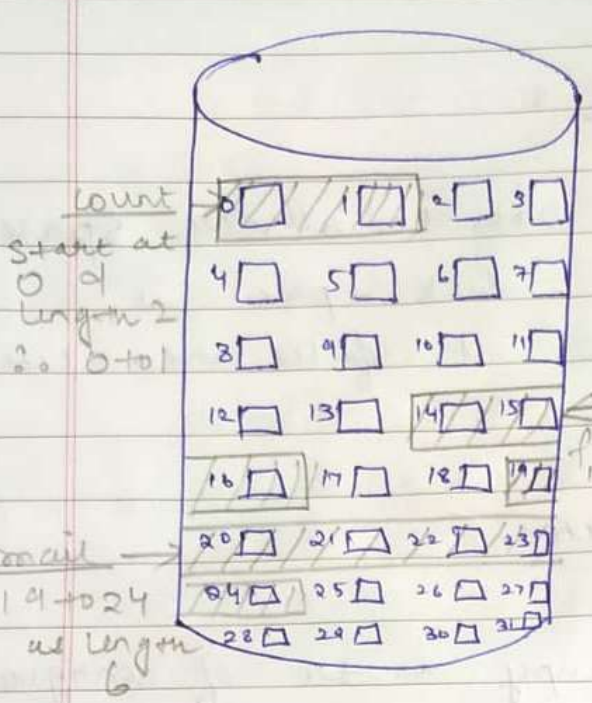
:: if only 1 job is accessing the disk then accessing block b+1 after block b involves no head movement.

If we need to move from last sector of 1 cylinder to first sector of next, the head only moves from 1 track to other.

No. of disk seeks required for accessing contiguously allocated file is minimal when a seek is finally met
↳ Seek Time.

- Contiguous allocation of a file is defined by disk address and length of first block.

- For sequential access, the file system remembers the disk address of last block referenced & reads next block.



Directory

filename	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Disadvantages of Contiguous Allocation

- Difficult to find free space for new file

It also may follow first fit, best fit
Suffer from external fragmentation & use compaction.
(you already know these)
Compaction is costly & effects performance

- Difficult to exactly determine how much space will be needed for a file
Even if it is known in advance, preallocation is inefficient.

* If your file is slowly growing over a period of time, it must be allocated enough space in the beginning as per its final size. A lot of this space is unused for a long time ∴ high internal fragmentation.

To solve these problems :-

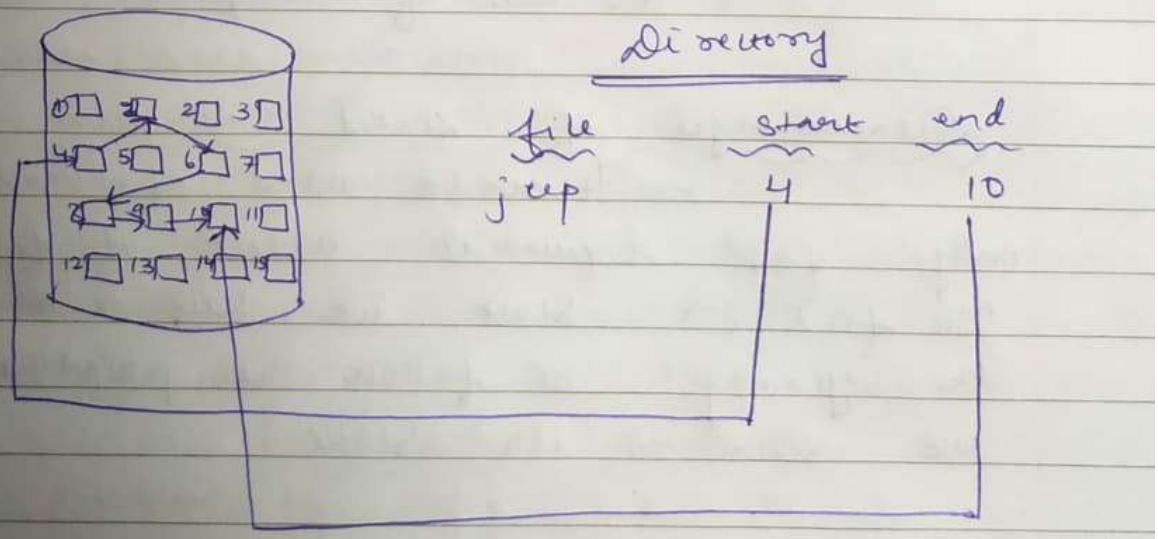
A continuous chunk of space is allocated initially.

If that proves not to be large enough then another chunk of contiguous space is added of is known as an extent.

b) Linked Allocation :-

- Here each file is a linked list of disk ~~blocks~~ blocks which may be scattered anywhere on the disk.
- Directory has a pointer to first & last block of the file.
- In between each block has a pointer to next block.
- The pointers are not made available to the user.

Eg. jaa



∴ Store at 4 → 1 → 6 → 8 → 9 → 10 →

• To create a new file we simply create a new entry into directory.
Initially the point is null to signify empty file.
The size field is :. 0.
• A write to the file causes the free space management system to find a free block of this new block is written to of is linked to the end of the file.
• To read a file, read blocks by following the pointer.

Advantages of Linked Allocation :-
• no external fragmentation
• any free block on the free space list can be used to satisfy a request
• Size of file need not be declared while creation
• File can grow as long as free space is available.
• No need of compaction

Disadvantages of Linked Allocation :-
• It can be used efficiently only for sequential access of files
To find ith block, we have to start from the beginning of follow the pointers till we reach the ith block.
• It is inefficient for direct access capability for linked allocation file.

- It needs too much space for pointers.

Solution :- collect blocks into multiples of called them clusters. Now allocate cluster rather than blocks. Then amount of space wasted on pointers is less.

However it leads to internal fragmentation.

- Linked allocation is less reliable as blocks are scattered all over the disk & problem occurs if pointer is lost / damaged.

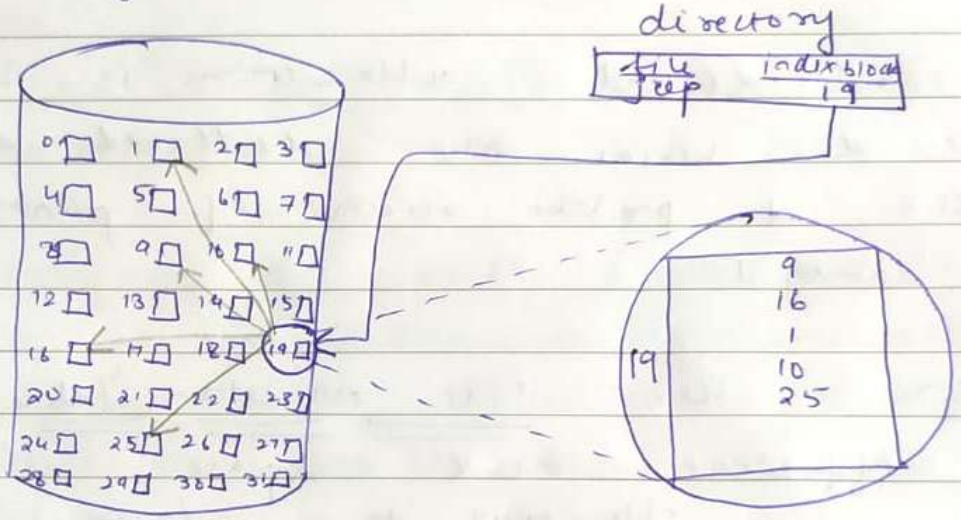
Solution :- Use File Allocation Table (FAT) for efficient disk space allocation. However this increases the seek time.

c) Indexed Allocation

- Linked allocation solves external fragmentation & size allocation problem of contiguous allocation
- In absence of FAT, linked allocation can't support efficient direct access as pointers to the blocks are scattered with the blocks themselves all over the disk but they need to be retrieved in order.

Indexed allocation solves this problem by bringing all the pointers together into one location → index block

- Each file has its own index block which is an array of disk block address
- i^{th} entry in index block points to i^{th} block of the file.
- Directory contains the address of the index block.
- To find & read the i^{th} block, we use the pointer to the i^{th} index block entry.



Advantages of indexed Allocation

- Supports direct access
- no external fragmentation

Disadvantage of indexed Allocation

- Still suffers from wasted space (infact more than linked allocation)
- Difficulty to decide how large the index block should be.

(You may read linked scheme, multilevel index & combined scheme from the book)

V) Free Space Management

- To keep a track of free disk space, the system maintains a free space list. which records all free disk blocks
- To create a file, we search the free space list for the required amount of space & allocate that space to the new file.
- This space is then removed from the free space list
- When file is deleted, its disk space is added to the free space list.

a) Bit Vector :-

- Free space ~~list~~ list is implemented as a bit map / bit vector.
- Each block is represented by 1 bit
  - → if block is free → bit = 1
  - → if block is allocated → bit = 0.

Advantage :- Simple & efficient to find the first free block.

Block No = no. of bits per word × no. of 0 valued words + Offset of first 1 bit.

b) Linked List :-

Link together all free disk spaces keeping a pointer to first free block in a special location on the disk & caching it in memory.

First free block then contains a pointer to the next free disk block & so on.

Disadvantage :- Inefficient & to traverse the list, we must read each block which is time taking.

c) Grouping :-

It stores address of n free blocks in first free block.

The first (n-1) of these blocks are actually free.

The last block has address of another n free blocks & so on.

Advantage :- Address of large no of free blocks can now be found quickly.

d) Counting :-

Several continuous blocks may be allocated / freed simultaneously when contiguous allocation algo / clustering is used.

∴ We can keep the address

of the first free block & the no. n of free contiguous blocks that follow this first block.

∴ Each entry in free space list has a disk address & count.

∴ Each entry needs more space however the list is short.

e) Space maps :—

(You may read from the book)

Its basically a record / log of free & allocated block in timely order.