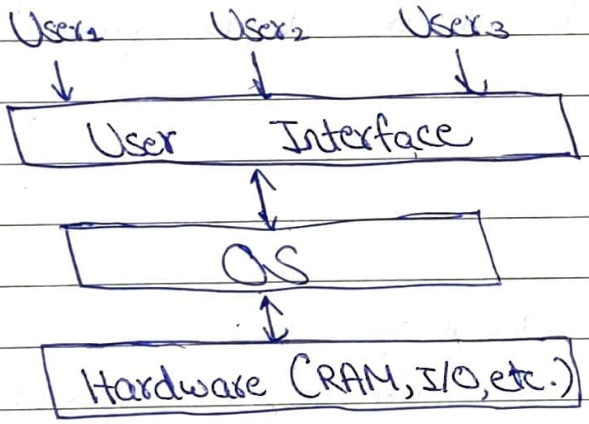
Operating System

I. Basics:

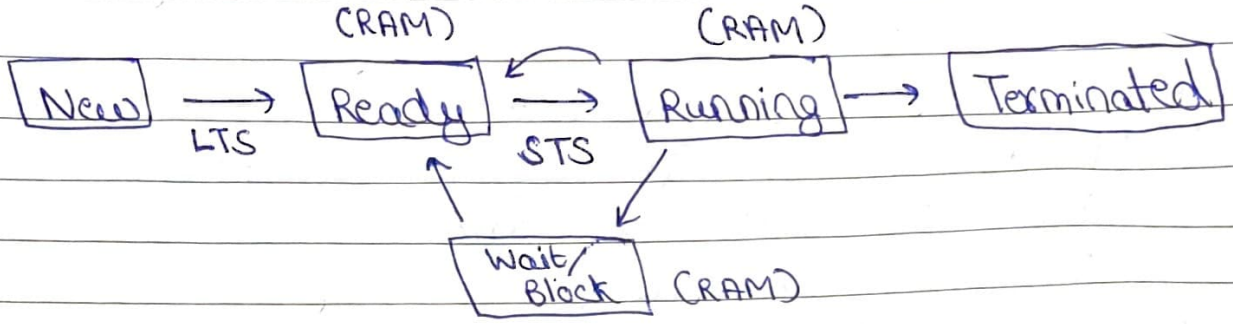
- OS works as an interface b/w user and hardware.
- Need for OS: If OS is not present, user will have to write a program each time they want to use hardware.
- Functionalities of OS:
  1. Convenience
  2. Throughput
  3. Resource management
  4. Process management
  5. Storage management
  6. Security.



- Types of OS:

1. Batch OS: Converting processes into batches and executing each batch one by one. (non pre-emption)
2. Multiprogrammed OS: Executing processes completely unless process goes for I/O (then we switch to another process). (non pre-emption)
3. Multitasking OS: Executing each process for a specific time and then switching b/w processes. (Pre-emptive)

- Process life cycle:



→ New: Creation of process
→ Ready queue: Queue of processes that are ready to be executed
→ Running state: Process gets CPU & starts execution
→ Terminated: Deallocation of the process
→ Wait / Block: When a process wants to perform I/O or block itself, it is passed to this state.

** Special cases:

→ If RAM space is running out or a priority wise high process arrives, then mid term scheduler will pass the process ~~for~~ from wait / ready state to suspend (wait / ready) state.
→ Long term scheduler schedules processes from secondary memory to RAM (Ready state).
→ In case of a high priority process in ready state or exhaustion of time quantum, short term scheduler switches process between running state and ready state.

- System call: A way to access functionalities of kernel mode from user mode.

- Fork system call: Creates a child process which has a different process id.

⇒ Return value :- 0 child ✓
+1 +ve → Parent
-1 -ve → child ✗

- Threads are a path of execution within a process. It helps to achieve parallelism by dividing a process into multiple threads.

Process	Thread
① System calls are involved.	① No system calls involved
② OS treats diff. process differently.	② User level threads are treated as single task for OS.
③ Diff. process have diff. copy of data, files, code.	③ Threads share same copy of code and data.
④ Context switching is slow.	④ Context switching is faster
⑤ Blocking a process will not block another	⑤ Blocking a thread blocks entire process
⑥ Independent	⑥ Interdependent.

- User level threads are managed by users, are faster and blocking a thread leads to blocking the process.
Kernel level threads are managed by OS, are slower and blocking a thread does not affect others.

II. Scheduling algorithms:

- Arrival time: Time at which process arrives in ready state.
- Burst time: Time required by process to get executed on CPU.
- Completion time: Time at which process gets completed.
- Turn around time: {Completion time - Arrival Time}
- Waiting time: {Turn around time - Burst time}
- Response time: {(Time at which process gets CPU for first time) - (Arrival time)}

⇒ FCFS (First Come First Serve)
Criteria: Arrival time, Mode: Non pre-emptive

⇒ SJF (Shortest Job First)
Criteria: Burst time, Mode: Non pre-emptive

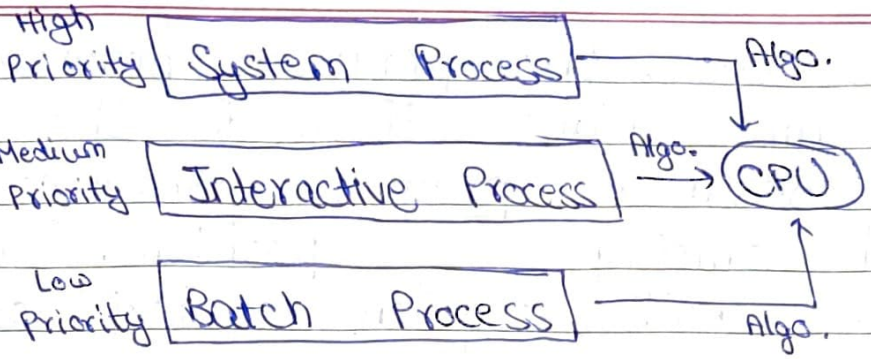
⇒ SRTF (Shortest Remaining Time First)
Criteria: Burst time, Mode: pre-emptive

⇒ RR (Round Robin)
Criteria: Time Quantum, Mode: pre-emptive

⇒ Priority scheduling
Criteria: Priority, Mode: pre-emptive

⇒ Multilevel queue scheduling

- Every queue will have a certain type of processes that will follow their own algorithm.



- Starvation: When low priority process keeps waiting for longer time because of large number of high priority processes.

⇒ Multilevel feedback queue scheduling

- In this scheduling, low priority processes give feedback to the system after each defined period, which helps them to upgrade their level and avoid starvation.

III. Process synchronisation:

- Used for cooperative processes (processes that share something, be it variable, memory, code, resources etc.)

- In absence of synchronisation, race condition may occur (competition of processes leading to wrong results).

⇒ Producer consumer problem:

- Two processes with shared buffer memory.

- In best case, the producer program produces items and places them in the buffer, and consumer program consumes items without causing any problem. (Synchronised processes with regular context switching).

- Let's assume that a shared variable which keeps the count of elements in buffer, gets increased by producer and decreased by consumer. In case of context switching while the variable is still getting updated, inaccurate results will be produced.

⇒ Printer Spooler problem:

- In best case, processes come one by one and spooler program keeps loading the document in shared memory and updating the shared variable. Printer prints the documents one by one.

- If context switching occurs inside spooler program due to which pre-emption occurs in absence of synchronisation, thus leading to loss of data.

- Critical section: It is part of the program where shared resources are accessed by various cooperative programs.

- Conditions for synchronisation:

Primary
1. Mutual exclusion: Only one program executes critical section at a time.
2. Progress: Programs should not prevent other programs to enter critical section if it is empty.

Secondary
3. Bounded wait: Programs should not keep accessing critical section infinitely while other programs starve.
4. No assumption related to H/w, speed

- LOCK variable is used to solve critical section problem. But, if preemption occurs just before value of variable is updated, then we cannot ensure mutual exclusion with this method.

- Test and set instruction attempts to achieve mutual exclusion by combining the testing and setting of value of the variable. Thus, we can avoid the problem that occured with LOCK variable.

- Turn variable (strict alteration) defines the turn for processes to enter critical section. Mutual exclusion and bounded wait will be achieved but progress cannot be ensured since a process prevents other processes from using critical section even if it is empty.

classmate
Date
Page

- Semaphore is an integer variable which is used in a mutual exclusive manner by various concurrent co-operative processes in order to achieve synchronization.

Semaphore
↙ ↘
Counting (-∞ to +∞) Binary (0,1)

Operations: → P(), Down, Wait
→ V(), Up, Signal

⇒ Performing P() operation reduces value of semaphore.
⇒ Performing V() operation increases value of semaphore.

- Producer consumer problem and reader writer problem can be solved with the help of semaphores.

⇒ Dining philosophers problem:

- This problem consists of N philosophers (they perform either thinking or eating) and N forks. In order for a philosopher to eat, the left and right fork of the position should be available. (seated in circle).

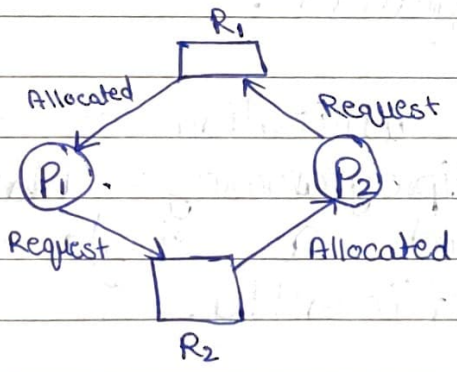
- The problem can have a solution using binary semaphore as whenever adjacent forks cannot be ~~required~~ acquired, ~~and~~ philosophers get blocked and can resume after it becomes available.

IV. Deadlock:

- If two or more processes, are waiting for some event to happen, which never happens, then the processes are said to be involved in deadlock.

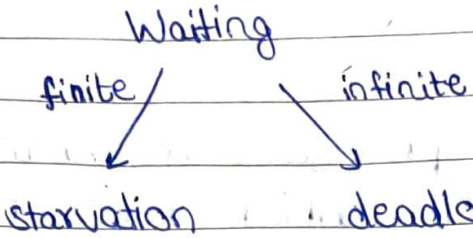
- Dining philosopher's problem suffers from deadlock as if every philosopher acquired left fork and got pre-empted, then none of them will get to eat.
  Solution to this problem is changing the order of picking up the fork for one philosopher.
  Example: If there are 100 philosophers, and 99 of them pick left fork first and right then, the remaining philosopher should pick right fork first and then left fork.

- Deadlock:



- Necessary conditions for deadlock:

1. Mutual exclusion
2. No pre-emption
3. Hold and wait
4. Circular wait



- Methods to handle deadlock:
  1. Deadlock ignorance (Ostrich method)
  2. Deadlock prevention
  3. Deadlock avoidance (Banker's algo)
  4. Deadlock detection and recovery

- Banker's algo. is a deadlock avoidance algo in which the system configuration (processes, resources, etc.) are given and it checks if deadlock will occur or not. Otherwise, it provides a safe sequence of processes that will not occur deadlock.

V. Memory management:

- Goal: Efficient utilization of memory.
  Degree of multiprogramming: Maxm. number of processes that a single-processor can accomodate efficiently.

⇒ Techniques: (Contiguous)

1) Fixed Partitioning (Internal Fragmentation):

- No. of partitions are fixed.
- Size of each partition may or may not be ~~fixed~~ same.

- Contiguous allocation so spanning is not allowed.

- Problems:
  1. Internal fragmentation
  2. Limit in process size (Process with more size than the maximum slot size cannot be accomodated).
  3. Degree of multiprogramming is limited. (No. of partitions are fixed).
  4. External fragmentation

2) Variable Partitioning

- Space allocation will be done as per the need of processes. Thus, wastage of memory will not be there. No limitation on process size and degree of multiprogramming.

- Problems:
  1. External fragmentation.
  2. Allocation / deallocation is complex.

★ For : contiguous memory allocation:

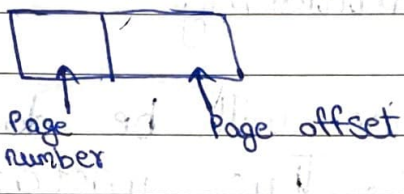
- First fit: Allocate first hole that is big enough
- Next fit: Same as first fit but start search always from last allocated hole.
- Best fit: Allocate the smallest hole that is big enough.
- Worst fit: Allocate the largest hole.

⇒ Techniques: (Non-Contiguous)

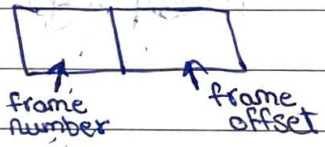
1) Paging

- Dividing processes into fixed size pages. Main memory is divided into frames and frame size is equal to page size.
- Every process has its own page table. Whenever any page is requested, its corresponding frame number is obtained and then it is accessed. (Page tables are mounted on main memory)

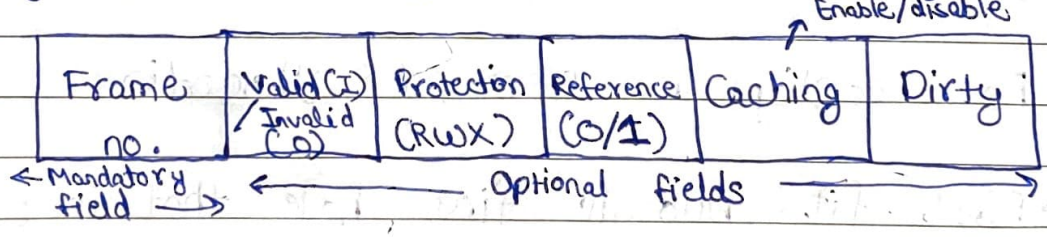
Logical address:



Physical address:



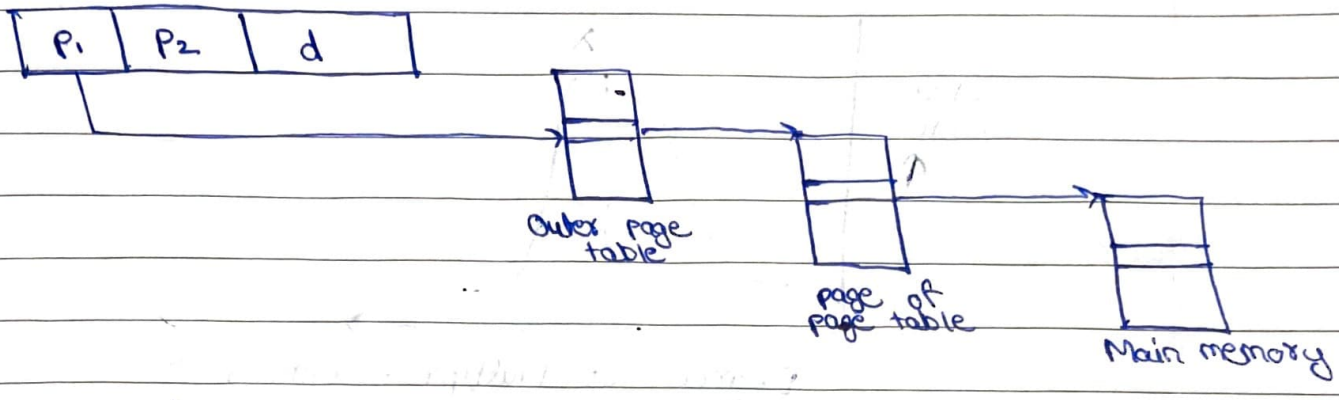
- Page table entry:



2) 2-Level Paging

- When the size of page table is more than size of main memory, then we need to divide page table into further pages.

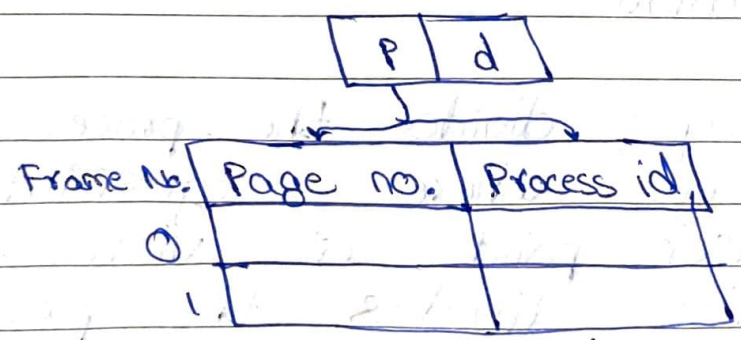
Logical address



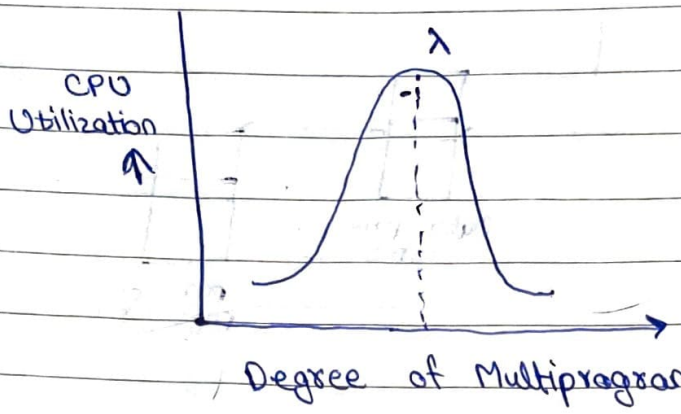
3) Inverted Paging

- Instead of having each process's page table into main memory (leading to wastage of frames), a global page table is maintained in the main memory.
- Query will consist of both page number and process id, which will lead to a disadvantage that always a linear search will be performed.

Logical address



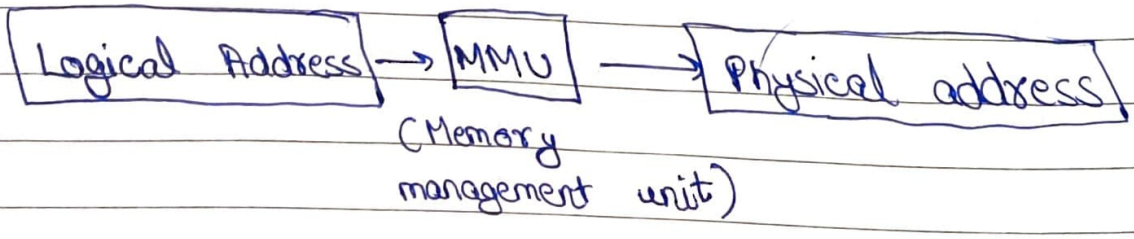
✡ Thrashing



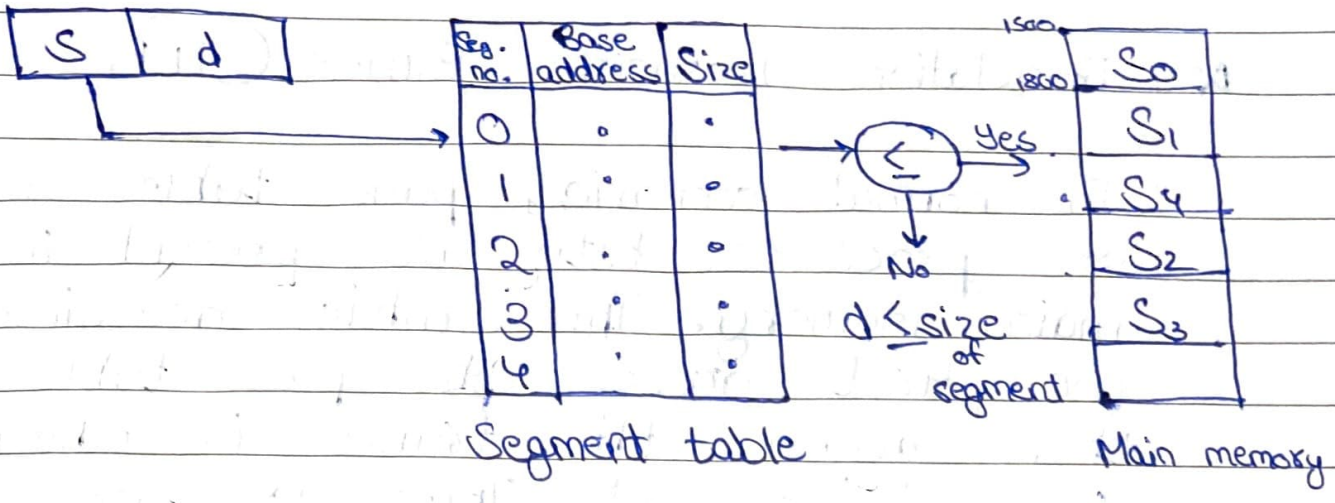
- Thrashing is a situation in which after a certain point, if we increase the ~~de~~ degree of multiprogramming (try to increase the no. of processes in main memory), then CPU ~~ut~~ utilization decreases due to increase in page faults.
- Solutions to thrashing:
  1. Increase size of main memory
  2. Long term scheduler.

4) Segmentation

- Paging simply divides the process into equal size pages whereas segmentation works from user point of view.
- Segmentation divides the process into different segments according to user requirements. Thus, segments can be of different size.



Logical address



★ Overlay: (for fixed partitioning)

- It is generally used in embedded systems. When the size of process is larger than main memory, then user divides the process into independent segments which can be brought into main memory and after it is processed, other segments can be brought for further execution.

⇒ Virtual memory

- Processes are divided into pages so that even when the physical main memory is limited with us, pages can be loaded into main memory and swapped in or out with secondary memory according to needs.

- Whenever a page requested is not present in main memory, page fault occurs and that page fault is serviced which consumes

a large time.

✡ Translation Lookaside Buffer (TLB)

- In normal scenario, page table and pages of processes, both are present in the main memory. Thus, while accessing a page, we first access the page table and then access the frame in which that page is present. Thus, the access time will be $2x$, where x is the time to access main memory.

- To make access time a little faster, some of the page table entries are stored in TLB which is cache memory. Cache access is faster than main memory and thus if we have a TLB hit for a certain page, time gets reduced.

Estimated Memory Access Time : (Hit) $(TLB + x)$ ✡ Miss $(TLB) + x + x)$

- Whole page table cannot be loaded on cache since the size of cache is limited and cost of increasing that size is high.

⇒ Page Replacement Algorithms

① FIFO : The frame that is filled first will be made vacant first.

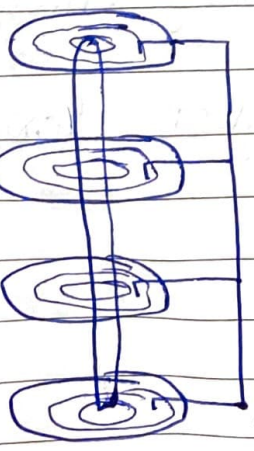
- Belady's anomaly:
Ideally, if we increase the number of frames, page faults should decrease. But it isn't the case since in certain situations, page faults increase with increase in frames. FIFO suffers from Belady anomaly.

② Optimal page replacement : Replace the page which is not used in longest dimension of time in future.

③ Least Recently Used (LRU) : Replace the least recently used page in past.

④ Most Recently Used (MRU) : Replace the most recently used page in past.

VI. Hard disk architecture:



Platters → Surface → Track → Sectors → Data

⇒ Disk Access Time:

1) Seek Time: Time taken by R/W head to reach desired track.
2) Rotation Time: Time taken for one full rotation (360°)
3) Rotational Latency: Time taken to reach to desired sector. (half of Rotation time)
4) Transfer Time: $\frac{\text{Data to be transferred}}{\text{Transfer rate}}$

Formula for transfer rate: (Data rate)

No. of heads (surface) × Capacity of one track × No. of rotations in one second

★ Disk access time = Seek Time + Rotation Time + Transfer Time + Controller time + Queue time

⇒ Disk Scheduling algorithms:

- Goal: To minimize the seek time.

① FCFS: Complete the requests according to the sequence in which they enter request queue.

- Request queue contains request for tracks.
- Advantage: No starvation
- Disadvantage: Performance

(2) Shortest Seek Time First (SSTF): Complete the request that has the shortest seek time from the current position of R/W head.

- Advantages: Response time, trying to minimize total seek time.
- Disadvantages: Starvation, will have to find nearest track at each step

(3) SCAN: Move in one direction till the last track on the surface, and then complete the tracks in opposite direction till last request.

(4) LOOK: Move in one direction till last request possible, then change direction and again complete tracks till last track request.

(5) C-SCAN: Move in one direction till the last track on the surface, shift to the end of opposite direction and complete the remaining requests.

(6) C-LOOK: Move in one direction till last request possible, shift to last request in opposite direction and complete the remaining requests.

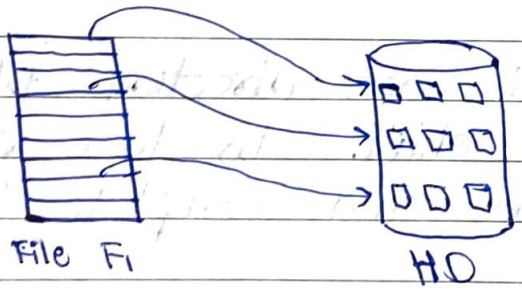
VII. File System:

- It is a software that manages storing and fetching of data.

- Operations on files:
  1. Creating
  2. Reading
  3. Writing
  4. Deleting
  5. Truncating
  6. Repositioning

- File Attributes:
  1. Name
  2. Extension (type)
  3. Identifier
  4. Location
  5. Size
  6. Modified date, created date
  7. Protection / Permission
  8. Encryption, compression

⇒ Allocation methods



- Contiguous Allocation
- Non Contiguous Allocation
  → Linked list allocation
  → Indexed allocation

- Goals:
  1. Efficient Disk utilization
  2. Access faster

- Contiguous Allocation:
  Store the blocks of a file in contiguous locations

→ Advantages:
  1. Easy to implement
  2. Excellent read performance

→ Disadvantages:
  1. Disk will become fragmented
  2. Difficult to grow file.

- Linked List Allocation: (Non Contiguous)
  The blocks will point to next block location of the file just like linked list.

→ Advantages:
  1. No external fragmentation
  2. File size can increase.

→ Disadvantages:
  1. Large seek time
  2. Random access / Difficult direct access
  3. Overhead of pointers.

- Indexed allocation : (Non contiguous)
Every file will have a index block which will contain information about it's blocks location.

→ Advantages:
1. Support direct access
2. No external fragmentation

→ Disadvantages:
1. Pointer overhead
2. Multilevel index

★ Unix Inode structure:
When the size of index block becomes very large, it is not possible to store the locations in a single block. So, Inode is used.