Short notes OS
CPU Scheduling

Scheduler:- (a.) Long term scheduler:- creating and bringing new processes into system, controls degree of multiprogramming, decides process from "New to suspended Ready", transition of a job from secondary to MM. Should select good comb^{n} of CPU bound & IO bound process. (Not executed frequently).

(b.) Medium term scheduler: suspending & resuming process (swapping).
Doing swap in & swap out operations.
- Swap in - moves process from Suspended (disk) ready to ready (m.m.) state.
- Swap out - moves process from Suspended block (~~disk~~ MM) to blocked (~~MM~~ disk) state.
- Involves in a transition of a job from "MM to SM" or "SM to MM".

c.) Short term scheduler:- select a process from ready state (in memory), which will be executed by the processor is called "Dispatching". → Executed frequently
- Select process from "Ready state and changes to running state."
- DISPATCHER - may help to dispatch the selected process by STS aSAP to CPU.
- Involves in a transition of a process from "MM to CPU".

DISPATCH LATENCY - The time takes to stop one process and start another running is known as the dispatch latency.

Turn Around Time :- $TAT = CT - AT$
$TAT = WT + BT$
{CT - completion Time, BT - Burst Time, WT - waiting Time, AT - Arrival Time}

Completion Time :- $CT = TAT + AT$ {usually finding using gant chart}

Response Time :- $RT = t_{(first\ response)} - At_{(Arrival\ time)}$

Batch Systems	Interactive Systems
* First come first Serve	* Round Robin
* Shortest Job first	* Highest Response Ratio next
* Shortest Remaining Time first	* Priority Scheduling with Multilevel queue
	* Priority scheduling with multilevel queue feedback.

⇒1 FCFS :- Implemented using queue, Non preemptive algorithm,
Advantage : Simple to Implement.
Disadvantage : CONVOY EFFECT: shorter jobs stuck behind longer jobs, it effects lower utilization of CPU and I/O.
i.e. if shorter processes arrived after longer process, then shorter processes responds very late.

⇒2 SJF :- based on smallest burst time processes, Non preemptive,
Criteria - Burst time, if all jobs are having same B.T., SJF works as "FCFS".
Adva - Minimizes average Response time. (max throughput)
Disadvan - Starvation may occur if so many shortest jobs are requesting then long jobs will starve.

⇒3 SRTF :- Preemptive, can preempt job while it executing if any smaller job needs to execute compared to remaining time of current job. If all jobs arrived at same Time, the SRTF works same as SJF algorithm.
Advan - minimizes Average TAT. Throughput - high.
Disadvan - May lead to Starvation if smaller jobs are many.

4
⇒ **Round Robin Scheduling** :- Preemptive, time quantum,
* Small time quantum lead to more Context switches (Overhead)
* very large T.Q makes RR same as FCFS.

Adv – good for shorter job.
Disadv – poor when all jobs have same length.

NOTE :- if n processes are in ready queue & TQ is q, the every process gets CPU service less than $(n-1)q$ time units.

5
⇒ **Higher Response ratio Next (HRRN) scheduling** :-

$$\text{Response ratio} = \frac{\text{TAT}}{\text{Burst time}} = \frac{\text{TAT}}{\text{BT}} = \frac{\text{WT}+\text{BT}}{\text{BT}} = \frac{\text{CT}-\text{AT}}{\text{BT}}$$

- HRRN selects a process which have higher Response ratio.
- Non-preemptive.

Advan – minimizes average TAT. favours both short & long
Disadv – jobs, Compromise b/w FCFS & SJF.

up through with theory book
⇒ **Priority Scheduling** :- higher priority, Preemptive or non preemptive,

1.) Preemptive Priority scheduling :- A process is preempted whenever a higher priority process is available in the ready queue.
2.) Non preemtive Priority scheduling :- The scheduler simply picks the processor with the highest priority to run next.

NOTE :- If all processes have equal priority then it work similar to FCFS.

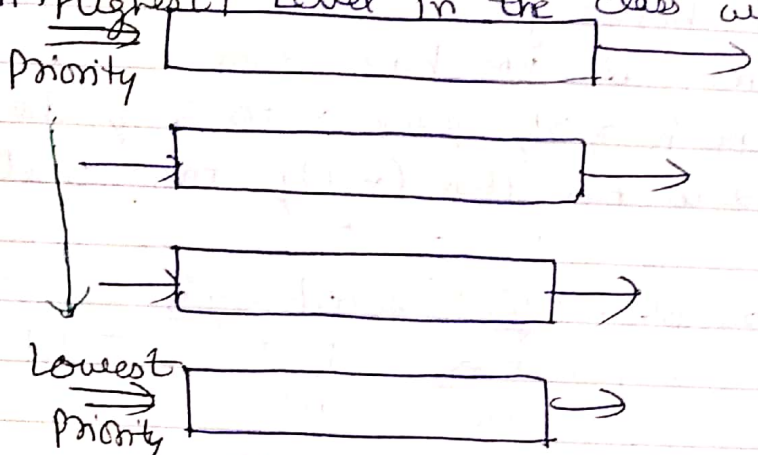
Advan – provides a good mechanism where relative importance of each process may be precisely defined.
Disadvan → STARVATION (If higher priority processes uses a lot of CPU time, lower priority processes may starve and be postponed infinitely. It can be solved by aging.

AGING :- Aging is a technique which gradually increases the priority of processes that wait in the system for long time.
2.) Another problem is deciding factor, which process will gets which priority level assigned to it.

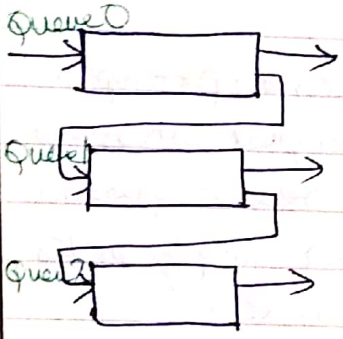
$\tau_{n+1} = \alpha \cdot T_n + (1-\alpha) \cdot \tau_n$; τ_{n+1} → next expected B.T. τ_n → previous expected B.T.
T_n – Actual Burst time (past history). α – relative weight of recent B.T.

7.) MULTILEVEL QUEUE SCHEDULING :- Depending on the priority of a process, in which particular ready queue, the process has to be placed will be decided.

for each Highest Level in the class will have own CPU scheduling algorithms. STARVATION may be occured at lowest levels, but aging can be used to prevent starvation.



8.) Multilevel feedback Queue Scheduling :-



Starvation may occured,

9.) LONGEST JOB FIRST :- Non preemption,
Disadva → Throughput decreases.
NOTE :- If burst time matches, then schedule which have lowest A.T.

10.) LONGEST REMAINING TIME FIRST :- Preemptive, does not suffer starvation. good R.T. High overhead. favour high CPU bound process.
Disad low throughput.

NOTE :- FCFS, RR, LRTF, HRRN will not suffer from starvation.
SJF, SRTF, LJF, Priority, MLQ, MLFQ, will suffer from starvation.

Virtual Memory

Virtual Memory:- separation of user logical memory from physical memory. (A technique that allows programs to be executed even though they are not stored in M.M totally.)

Advantages :- 1.) A process can execute without having all its pages in Physical memory. (2.) high degree of multiprogramming. 3.) Less I/O for loading and unloading for individual user processes. 4.) Higher CPU utilization and throughput. 5.) Allows address spaces to be shared by several processes. 6.) It eliminates external fragmentation and minimizes internal fragmentation by combined segmentation and paging. 7.) It allows for sharing of code and data.

Disadvantages:- Increased hardware costs. Increased overheads for handling paging interrupts. Increased software complexity to prevent thrashing.

Virtual Memory techniques:- Overlays, Paged VM, Segmented VM.

NOTE:- Windows practically follows the concept of demand paging.

swap in / swap out

Demand Paging - Loding the page into memory on demand is DMPag

Page fault - A type of hardware interrupt caused by a reference to a page not residing in memory.

Performance in VM:- let p be page fault rate, S - PF Service time, M - MM access time, EMAT = $P \times S + (1-P)\,M$

NOTE ⇒ Page fault time = Page fault overhead + swap out + swap in + restart overhead.

PAGE REPLACEMENT ALGORITHM:-

1.) FIFO:- Replace the old page with new ones. It can be implemented using either "Queue" or "a time-stamp on pages".

Belady's Anomaly:- By increasing no. of frames, PF should decrease, but it increases.

2.) Optimal:- Replace the page, which will not used for longest duration in future. for a fixed no. of frames, optimal has lowest PF in all of algorithms. but it is not practical. Because it requires future knowledge.

NOTE:- if 2 or more than two pages are not used in future, replace the page which comes first.

3.) LRU algorithm :- Replace the page which was not recently used. It can be implemented using "stack" or "counter".

4.) MRU :- Replace the page which is most recently used.

THRASHING :- Thrashing is a situation which occur when a process is spending more time in paging than executing. upto λ, CPU utilization & throughput are very high, if we further increase the degree of Multiprogramming, the throughput will drastically fall down & the system will spend more time only in page replacement.

Causes :- 1.) High Degree of Multiprogramming 2.) Lack of frames.

recovery :- i.) Instruct the long term scheduler not to bring the processes in the system after point of maxima.

ii.) If the system is already in thrashing then instruct the mid term scheduler to suspend some of the process,

LOCALITY : It is a property of the page reference string (property of programs). A set of pages that are actively used together.

- Temporal locality : Pages that have been used recently are likely to be used again.
- Spatial locality : Pages near to those that have been used are likely to be used next. That is once a location is referenced, a nearby location is often refenced soon.

DISK SCHEDULING

Purpose of Disk scheduling - Select a disk request from the queue of I/O requests and decide when to process this I/O request.

Issues in Disk scheduling - Throughput, fairness (some disk requests may have to wait a long time before being served).

Goal of disk scheduling :-
- High throughput
- fairness.
- Minimed seek time.

Characteristics of disk scheduling :-
scheduling means to impove the average disk service time.
Speed on Service depends on :
- Seek time, most dominating in most disks
- Latency time, or rotational delay
- Data transfer time

Each disk drive has a queue of pending requests. Each requests made up of :-
- whether input or output
- Disk address (disk, cylinder, surface, sector)
- Memory address
- Amount of information to be transferred.

For moving-head disk, disk scheduling algorithms are needed to minimize seek time.

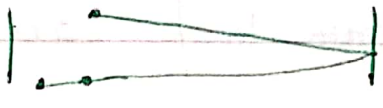
1) FCFS Scheduling :- FCFS service I/O request in the order they arrived.

Disadvan - does not try to optimize seek time. Wild swings occur because the requests do not always come from the same process; they are interleaved with requests from other processes.

2) SSTF :- Service all requests close to the current head position before moving the head far away.

Disadvan - SSTF is a form of SJF; may cause starvation.
- may create a lot of disk seeking activity as the head may seek back & forth with ever wider swings as it services the more distant requests.

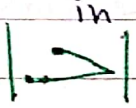
3) SCAN (ELEVATOR) :- starts from one end & go to the other end completely then come back to the previous end until last request.



In C-scan & C-look, Request will be done only in forward direction.

C-scan :- (Circular); It starts from one end, go to the other end completely, then reversed back to the one end completely, then again go to the last requests. It works in forward direction only.

LOOK :- LOOK is improvement over Scan. Move the head as far as the last request in that direction, If no requests left in that current direction, reverse the head movement.



C-look :- same as C-scan but instead of seeking to the start of the disk, we seek to the lowest track with scheduled I/O. Disk travels as far as the last request in each direction, then reverse immediately.

Strategy	Advantages	Disadvantages
"FCFS"	• easy to implement • Sufficient for light loads	• Doesn't provide best Av. service • Doesn't maximize throughput
"SSTF"	• Throughput better than FCFS • Tends to minimize arm movement	• may cause starvation of some requests • Localizes under heavy loads
"SCAN/LOOK"	• Eliminates starvation • Throughput similar to SSTF • Works well with light to moderate loads.	• Needs directional bit • More complex algo to implement • Increased overhead.
"N-Step Scan"	• Easier to implement than SCAN	• The most recent requests wait longer than with SCAN
C-SCAN/ C-LOOK	• Works well with moderate to heavy loads. • No directional bit • Small variance in service time. • C-LOOK doesn't travel to unused tracks	• May not be fair to recent requests for high-numbered tracks. • More complex algo than N-Step Scan, causing more overhead

Concurrency and Deadlock

Concurrency:- A state in which process exist simultaneously with another process then those it is said to be concurrent. Concurrency allows multipk applications to share resources in such a way that applications appear to run at the same time.

There are several motivations for allowing concurrent execution:
- **Physical resource sharing:** Multiuser environment since hardware resources are limited.
- Logical resource sharing: Shared file (same piece of information).
- Computation speedup: Parallel execution.
- Modularity: Divide system functions into seperation processes.

Relation b/w Processes:- The processes executing in the OS is one of the two types
- **Independent Process** - Its state is not shared to any other process.
  - The result of execution depends only on the Input state.
  - The result of the execution will always be same for same input.
  - The termination of the independent process will not terminate any other.
- **Cooperating Process** - Its state is shared along other processes.
  - The result of the execution depends on relative execution sequence and cannot be predicted in advanced (Non deterministic).
  - The result of execution will not always be same for same input.
  - The termination of the cooperating process may affect other process.

- **Advantages of Concurrency:-**
  - Able to run Multipk apps at same time
  - **Better resource utilization** - Resource that are unused by one application can be used for other applications.
  - **Better average response time** - without concurrency, each application has to be run to completion before the next one can be run.
  - **Better performance** - If one app uses only the processor, whik another app uses only the disk drive, the time to run both app concurrently to completion will be shorter than the time to run individual

Drawbacks of Concurrency:-
- Multipk apps need to be protected from one another
- Multipk applications may need to coordinate through additional mechanism
- Switching among applications requires additional performance overheads and complexity in OS (e.g., deciding which application to run next).

• In extreme cases of running too many applications concurrently will lead to severely degraded performance.

Issues of Concurrency :- • Non-atomic : Operations that are not atomic, but interruptible by multiple processes can cause problems.

• Race Conditions :- A race condition occurs if the outcome depends on which of several processes gets to a point first.

• Blocking - Processes can block waiting for resources.

• Starvation - It occurs when a process does not obtain service to progress.

• Deadlock - It occurs when two processes are blocked and hence neither can proceed to execute.

Precedence Graph :-

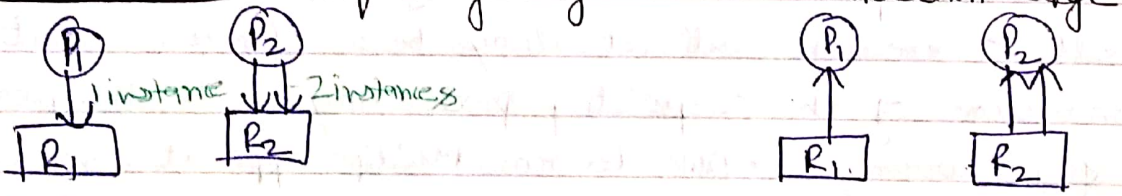
Bernstein's Conditions :- Three conditions must be satisfied for two successive statements S1 and S2 to be executed concurrently and still produce the same results.

(i) $R(S1) \cap W(S2) = \{\}$

(ii) $W(S1) \cap R(S2) = \{\}$

(iii) $W(S1) \cap W(S2) = \{\}$

DEADLOCK - • Requesting Edge • Allocation Edge



• Release the resource - The process releases the resource through system call.

NOTE :- The resource request and resource allocation will be represented by using resource allocation graph. RAG = G (V, E)
↳ Processes, Resources
↳ Requesting, allocation

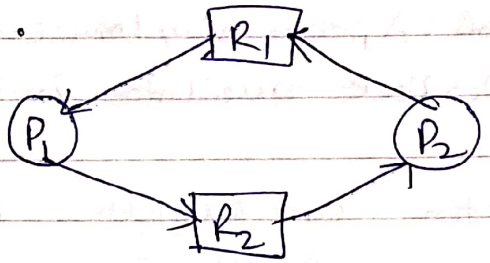
CONDITIONS FOR DEADLOCK :-

• Mutual Exclusion :- Only one process can use a resource at a time.

• No Preemption :- No process is allowed to preempt another process forcibly in order to gain a release of a resource from it.

• Hold and wait - A process acquires a resource and waiting on some other resource simultaneously.

• Circular wait :- The processes are circularly waiting on each other for the resources.



$R_1 \rightarrow P_1 \rightarrow R_2 \rightarrow P_2$

NOTE :- If all these four conditions are existing simultaneously in the system, then definitely there exists a Deadlock.

DEADLOCK PREVENTION :- Invalidate any of the four necessary conditions and the deadlock will not occur.

• Prevent/deny Mutual exclusion - Not possible to dissatisfy the mutual exclusion always because of sharable and non sharable resources. (Ex - Printer).

• Prevent/ deny No Preemption - The process P_1 is requesting for resource R_1.

If the Resource R1 is free then it will be allocated to the process 'P_1'.

The Resource R_1 is not free and it is allocated to some other Process 'P_2'.

If the Process 'P_2' is in the execution, then Process 'P_1' has to wait

Problems:-
Some resources cannot be preempted (Printer).
May require the job to restart.

The Process P_2 is not in the execution and it is waiting on some other Resource 'R_2'.

↳ Preemption. Preempt the resource R_1 from process P_2 and allocate it to the Process 'P_1'.

• Prevent/Deny Circular Wait - order resources and allow process to request for them only in increasing order. If a process needs several instances of the same resource, it should issue a single request for all of them. Problems :- Adding a new resource that upsets ordering requires all code ever written to be modified. Resource numbering affects efficiency. A process may have to request a resource well before it needs it, just because of the requirement that it must request resources in ascending order. • Prevent Hold and wait - (i) Allocate all the Resources required by the process before start of execution. (ii) The process should release all existing resources, before making the new Request.

Problems:- Low device utilization or low resource utilization.
Starvation possible. A process may have to wait indefinity.

DEADLOCK AVOIDANCE:- Deadlock avoidance is implemented by using Banker's Algorithm.

Max Need			Current Allocation			Curr. Available			Remaining need			
A	B	C	A	B	C	A	B	C	A	B	C	
7	5	3	0	1	0	3	3	2	7	4	3	$\rightarrow P_0$
3	2	2	2	0	0	5	3	2	1	2	2	$\rightarrow P_1$
9	0	2	3	0	2	7	4	3	6	0	0	$\rightarrow P_2$
2	2	2	2	1	1	7	5	3	0	1	1	$\rightarrow P_3$
4	3	3	0	0	2	10	5	5	4	3	3	$\rightarrow P_4$
total Available			7	2	5							
10	5	7										

Remaining need = Max need - current Allocation
Current available = total Available - total Allocation

Safe sequence - P_1, P_3, P_0, P_2, P_4.

NOTE:- Deadlock avoidance is less restrictive than deadlock prevention.

DEADLOCK DETECTION:- If all the resources are of single instance type, then the cycle in the resource allocation graph is necessary and sufficient condition for occuring of deadlock.

⇒ If all the resource are of non single instance type then cycle in the Resource allocation graph is just a necessary condition but not sufficient condition for occuring of a deadlock.

DEADLOCK RECOVERY:- (i) Process Termination (Killing the process)
(ii) Resource Preemption
(iii) Ostrich algorithm:- Ignore deadlock.

FILE SYSTEM

FILE:- Collection of logically related entities (record).

Attributes – • filename • filetype (may be) • location • size • creation Date • last modified date • permissions • owner • passwords • timestamp.

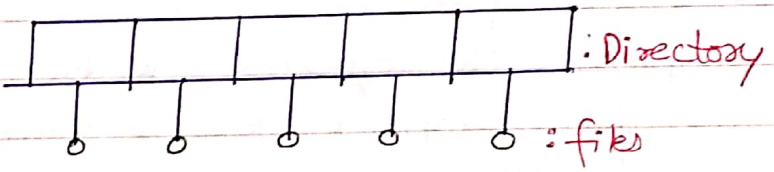
TYPES – : doc • Txt • Png jpg • mpy • xlsx • dll • exe • 3gp

OPERATIONS – • create • write/append • read • seek • delete • truncate • open, close • hide • save/saveas

ACCESS METHODS – • Sequential file Access • Random file Access.

DIRECTORY STRUCTURE –

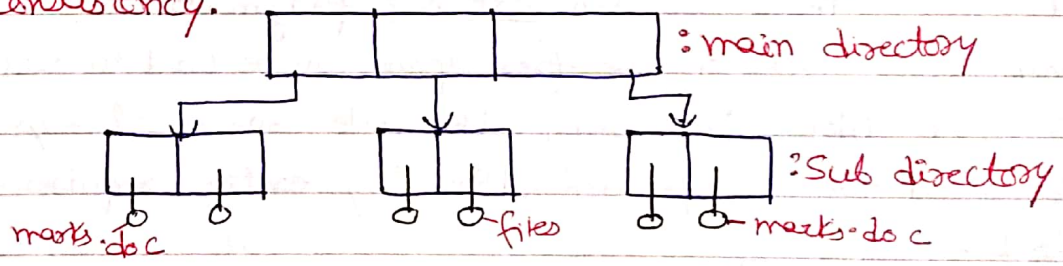
1) Single level Directory –



→ Two files cannot have same name.
→ Implementation is easy. → Searching time for file will be more.
→ could be necessary for multiple users / programs on a disk.

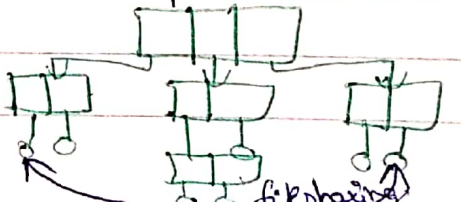
2) Multi Level Directory (Tree Level directory) –

→ The implementation of the directory structure is difficult or complicate
→ Better classification of the files as per the criteria.
→ Searching Time for the files will be less.
→ If the same file exist in the 2 different directories, if one file is updated then the other file has to be updated accordingly, otherwise there will be inconsistency.



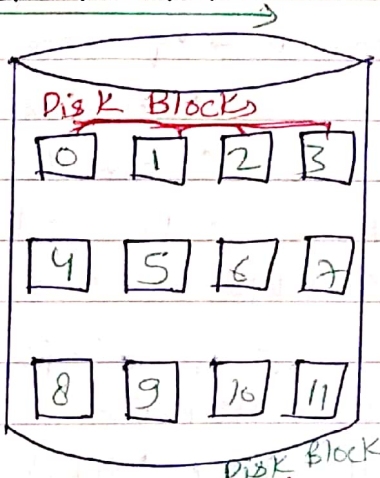
3) Acyclic Graph Directory structure :-

→ Implementation of directory structure is difficult.
→ The better classification of the file as per the criteria.
→ Searching time for the files will be less.
→ If the same file exist in the two different directories, then if one file is updated then the other file will be updated automatically by using file sharing concept.



FILE ALLOCATION METHODS / DISK SPACE ALLOCATION METHODS :-

1) Contiguous Allocation :-



→ Each file is allocated a set of contiguous disk blocks.

→ Every file is associated with the 2 parameters :-
1) Starting DBA 2) Size (wrt no. of Disk Blocks)

→ It is suffering from external fragmentation.

→ Internal fragmentation may exist in the last disk block of the file.

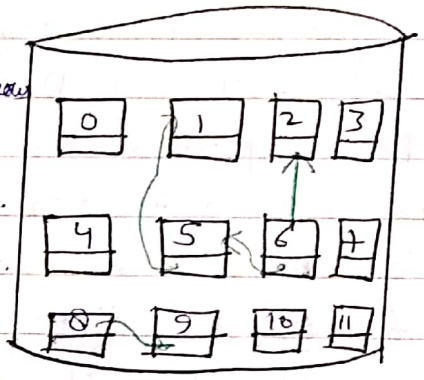
→ It supports both sequential & Random file access.

Starting DBA	+ offset ⇒	Random Access

Disk Block Addr

File	Starting DBA	Size
abc.doc	2	4
xyz.doc	9	5

2) Linked Allocation :- (Non-contiguous)



→ Each file is allocated with a set of non-contiguous blocks.

→ Every file is associated with the two parameters.
1) Starting block Address 2) Ending block Address.

→ Each disk block has a pointer to next disk block in the file as well as some file data.

Advantages :- There is no external fragmentation.

→ Any free block on the free space can be used to satisfy a request for free block. Increasing the file size is always possible.

→ No need for compaction. Directory entry requires only starting block.

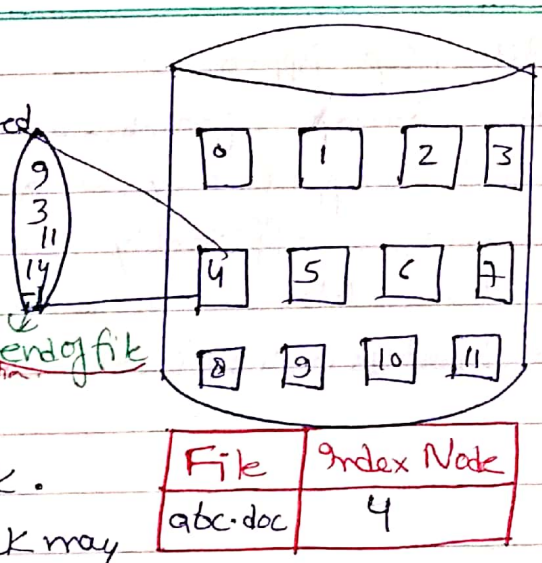
Disadvantages :- It supports only sequential access of the file.

→ Need to traverse each block. Internal fragmentation may exist.

→ overhead of maintaining the pointer in every disk block.

→ If the pointer of any disk block is lost, then the file will be truncated.

3.) INDEXED ALLOCATION :-



File	Index Node
abc.doc	4

- Each file is associated with its own indexed node.
- Indexed node contains all the disk block addresses of the file.
- **Advantages :-** There is no External fragmentation.
- Indexed allocation supports direct access.
- A bad data block cost only that block.

Disadvantages :- for a very large file, one block may not be sufficient to store all the disk block addresses.

- for a very small file, it is waste of using entire one disk block.
- Internal fragmentation may exists in the last disk block of the file.
- Size of a file is limited by the no. of pointers a data block can hold.

UMX-I-NODE Implementation :- (An extension of Indexed allocation)

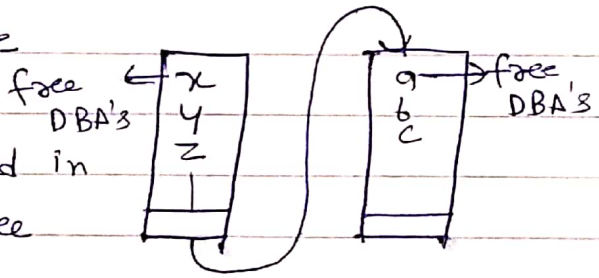
- No. of disk block address possible to store in 1 disk block

$$= \boxed{\frac{\text{DB size}}{\text{DBA}}} = \frac{\text{Size of disk block}}{\text{disk block address length.}}$$

- Total size of file system $= \left[\text{No of Direct DBA} + \frac{\text{DBA size}}{\text{DBA}} + \left(\frac{\text{DBA size}}{\text{DBA}} \right)^2 + \cdots \right] \times \text{DB size}$

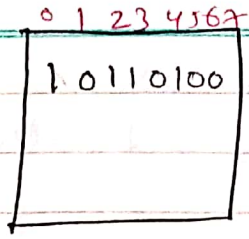
(# single Indirect, # double Indir)

Free list approach :-

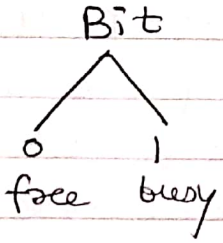


- All disk blocks that are free maintained in a reserved portion of the disk called free space list.
- Each block is assigned as sequential number.
- List of all numbers corresponding to free blocks are maintained in a free space list.
- There is a possibility to store a part of the table in MM by using "Stack or Queue".

Bit Map Approach :-



Bitmap



→ In a bitmap approach, every disk block will mapped only with 1 binary bit.

Advantage :- Efficiently can find the free blocks or n consecutive free blocks on the disk.

Disadvantage :- Inefficient unless the