Operating System:

→ It is an interface b/w User and hardware.

throughput: no. of task executed per Unit of time

function:

1) Resource mangement
2) Process management (CPU Scheduling)
3) Storage management (file System)
4) Memory management (RAM)
5) Security

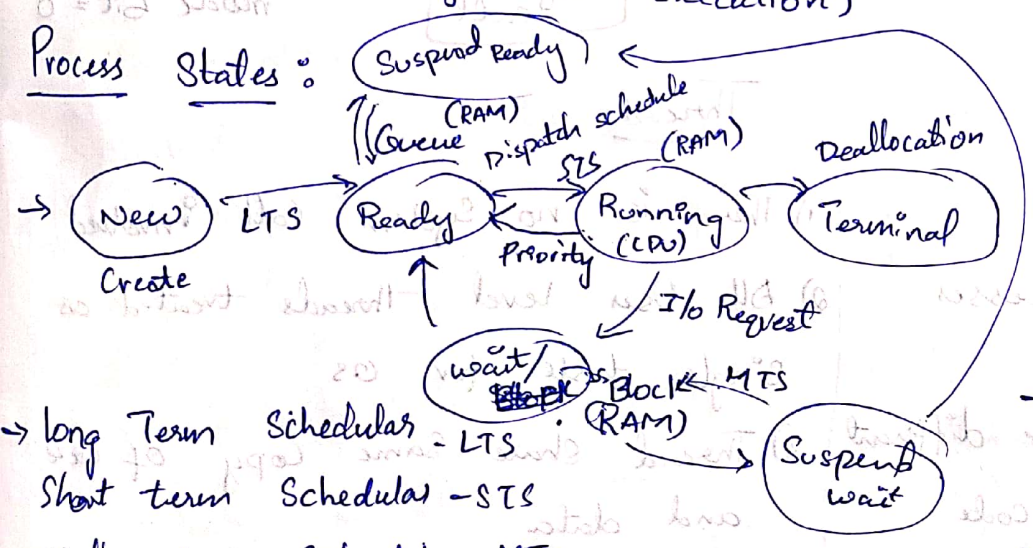
→ Non-preemptive (Non-stop CPU execution)

→ preemptive (priority based execution)

Types of O.S:

→ Batch
→ Multi programmed
→ multi tasking (Time sharing)
→ Realtime O.S (no delay like live videos)
→ Distributed (loosely couple missile env)
→ Clustered
→ Embeddad (specific task) Ex: ac/TV/WM

Process States:



→ long Term Schedular - LTS
Short term Schedular - STS
Medium term Schedular - MTS

Permission:

r - read - 4
w - write - 2
x - execute - 1

u - User } r w x
g - group } r w x
o - other } r w x

→ Chmod Ugo =r note
change mode
→ Chmod 666 note (rw)
→ lseek (to change pos in the file)
→ fork

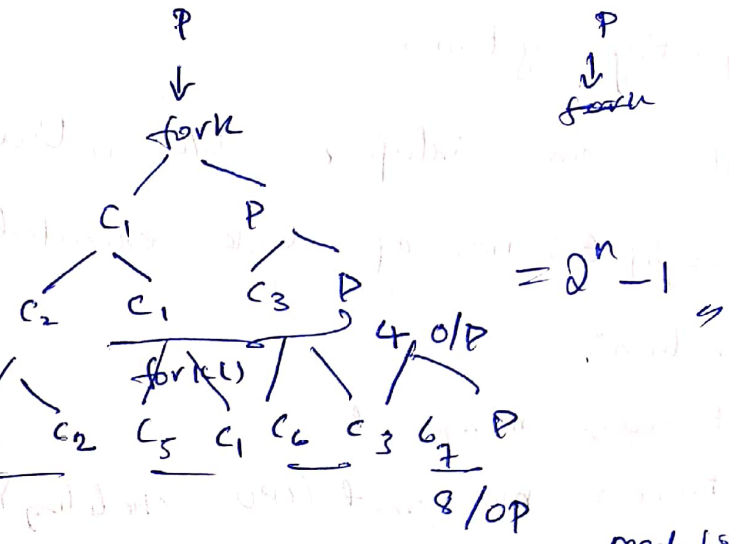
System Call:

To Switch to Kernal mode.

→ file related system call: Open(), Read(), write(), close()
→ Device Related: Read, write, Reposition, ioctl, fcntl
→ Information: get pid, attribute, get system time & date
→ Process Control: load, execute, abort, fork, wait, signal, Allocate etc
→ Communication: pipe(), shmget(), Create/delete Connections.

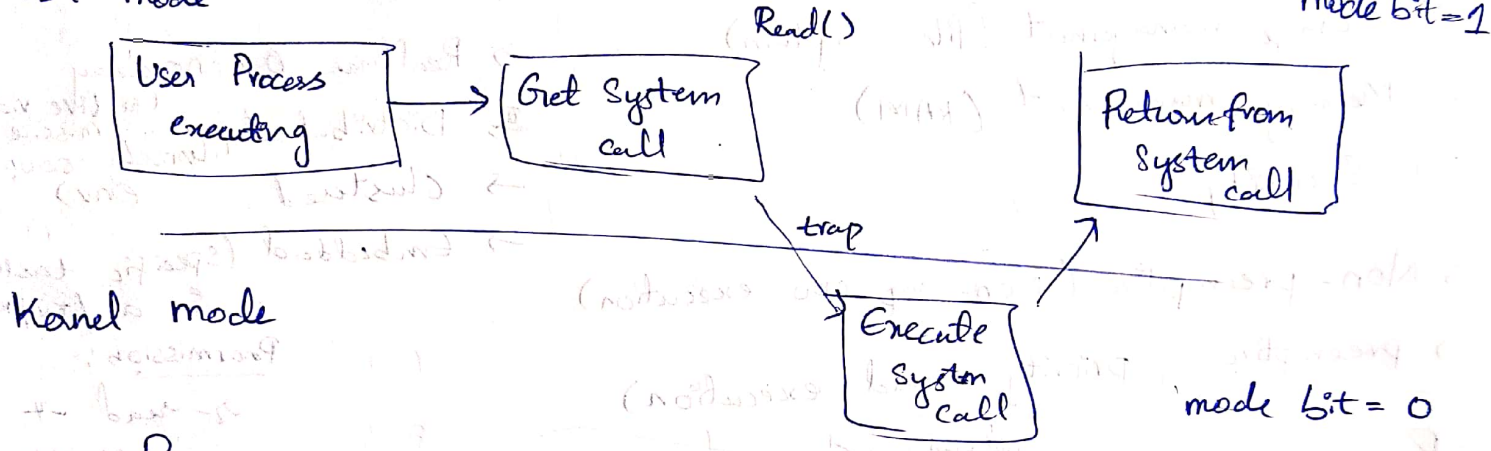
fork() → 0 child, +1 parent

To create a child process



$= 2^n - 1$

User Mode Vs Kernal Mode :



Process	Threads
1) System call involves	1) There is no System Call involved
2) OS treats different processes differently	2) All User level threads treated as single task for OS
3) Different process have different copies of data, files, code	3) Thread share same copy of code and data
4) Content Switching is slower	4) Content Switching is faster
5) Blocking a process will not blocks another	5) Block entire process
6) Independent	6) Independent Inter

User level Thread

1) User level thread are managed by User level library
2) User level thread typical fast.
3) Content Switching is faster.
4) If one User level threads perform blocking operation then entire process get blocked.

Kernel Level Thread

1) Kernel level Thread are managed by OS. (System call)
2) Kernel level thread are slower than User level
3) Content Switching is slower
4) If one kernel level thread blocked, No affect on other.

Scheduling Algorithms

Pre-emptive

- → Shortest Remaining time first (SRTF)
- → Longest Remaining time first (LRTF)
- → Round Robin ("Time Quantum")
- → Priority based (check higher the no. higher the priority)

Non pre-emptive

- → FCFS (first come first serve) (Arrival Time).
- → SJF (shortest job first) (Burst Time)
- → LJF (longest job first)
- → HRRN (Highest Response Ratio next)
- → Multilevel Queue.
- → Priority.

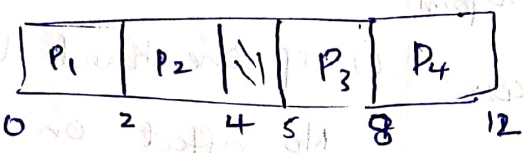
"CPU Scheduling"

→ Arrival time: The time at which process enter the Ready Queue
→ Burst time: Time required by a process to get execute on CPU
→ Completion time: The Time at which process complete it's execution.
→ Turn Around time: {Completion time - A.T}
→ Waiting time: {Turn around time - Burst Time}
→ Response time: [(The time at which a process get CPU first time - (Arrival Time)].

FCFS (Non-Preemptive) :- (A.T) Criteria

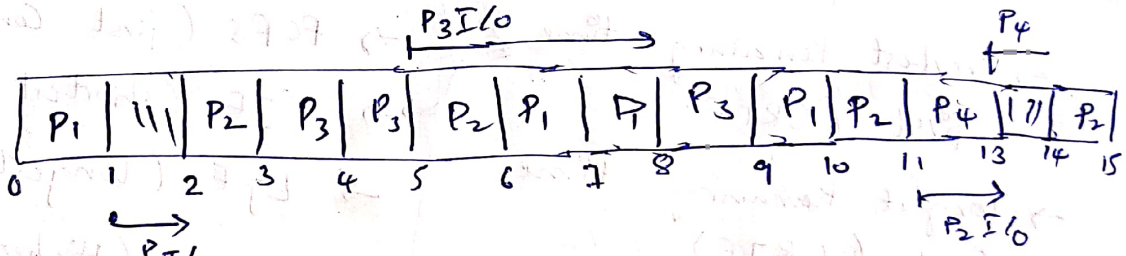
Process No	A.T	B.T	C.T	T.AT.	W.T	R.T
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

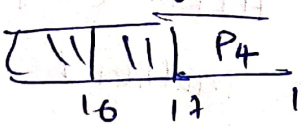
Gantt chart



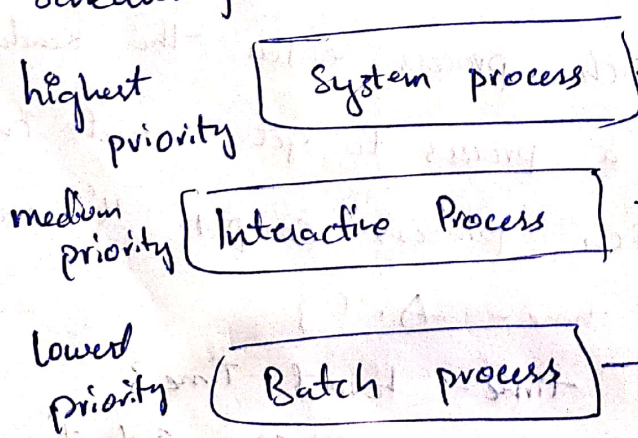
Process	AT	Priority	CPU	I/O	CPU	(C.T) Avg
P_1	0	2	1	5	3	(10)
P_2	2	3	3	3	1	(15)
P_3	3	1	2	3	1	(9)
P_4	3	4	2	4	1	(18)

Gantt chart

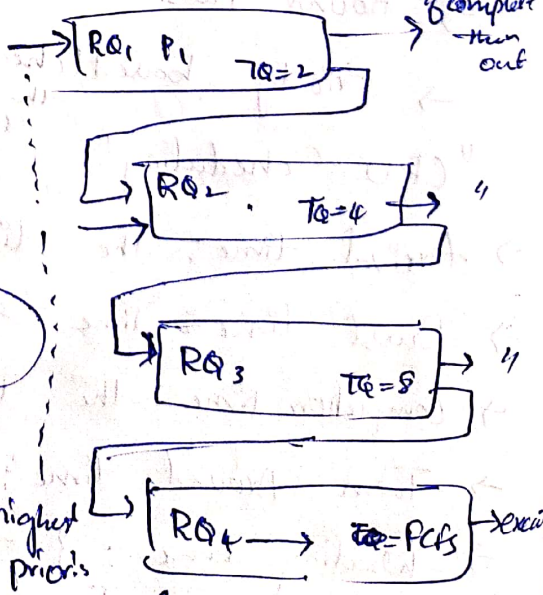




Multilevel Queue Scheduling



Feedback Queue.



Process Synchronization :

Processes

Cooperative process
- Share variable

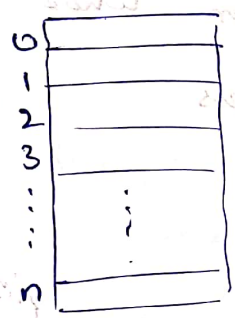
independent process

Produces Consumer Problem :

→ Buffer is share by both process (C & P)

int count = 0;

Buffer [0...n-1]




```
Void producer (void)
{
int itemp;
while (true)
{
Produce_item (itemp);
while (count == n);
Buffer[in] = itemp;
in = (in+1) mod n;
count = count+1;
}
}
```

```
Void Consumer (void)
{
int itemc;
while (true)
{
while (count == 0);
itemc = Buffer (out);
out = (out+1) mod n;
count = count - 1;
Process_item (itemc);
}
}
```

Case i) Best Case producer is producing and consumer is Consuming.

Case ii) producer is producing but preemption is happen so CPU will focus on next process which is Consumer again at the saving variable preemption happen.

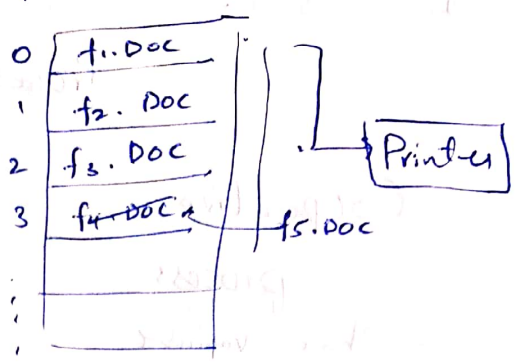
Printer - Spooler problem :

I

1. load R_i, m[in]
2. Store SD[R_i], "F-N"
3. INCR R_i
4. Store m[in], R_i

P_1 I_1, I_2 I_3 | P_2 I_1 I_2 I_3 I_4 | P_1 I_4
(preempt, preempt)

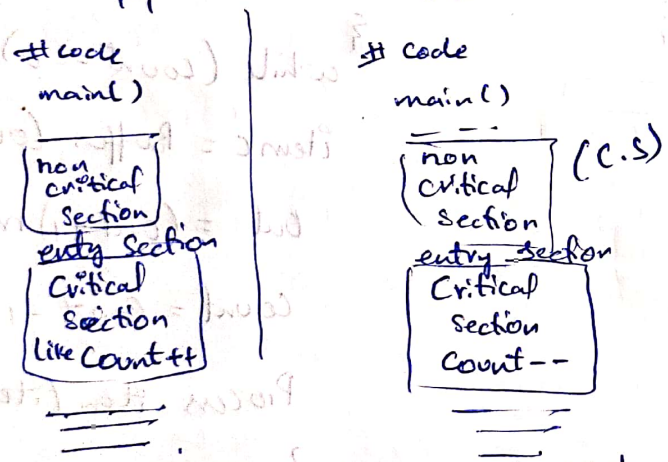
Spooler Directory



P_1 P_2
↓ ↓
f2.doc f2.doc

Critical Section :

It is part of the program where shared resources are accessed by various process



Synchronization mechanisms :

4 Condition

primary 1) Mutual Exclusion (Once entered into C.S no other can enter)
2) Progress (not in CS but stoping to use CS)
secondary 3) Bounded wait (one process only using CS)
4) No assumption related to H/W Speed.

Critical Section Solution Using locks :

```
do {
    acquire lock
        CS
    release lock
}
```

here prompt can create problem (No mutual Exclusion)

1. while (lock == 1)
2. lock = 1
3. Critical Section
4. lock = 0

(1, 2 : Entry code; 4 : Exit code)

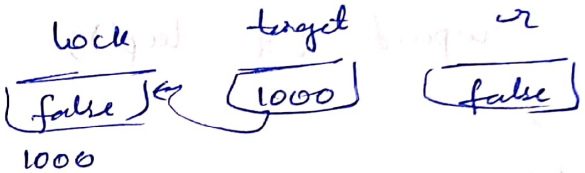
* Execute in User mode
* No mutual exclusion Guarantee
* multi process solution.

Critical Section Solution Using "Test-and-Set" Instruction

```
while (test_and_set (& lock)); // Combine 2 Instruction into One.
                               // to avoid preemption
CS

lock = false;
```

```
boolean test_and_set ( boolean *target)
{
    boolean r = *target;
    *target = TRUE;
    return r;
}
```



Turn Variable (Strict Alteration)

→ 2 process Solution → Mutual Exclusive

→ It run in User mode

Process "P_0"	Process "P_1"
No C.S	Non C.S
Entry code while (turn != 0);	while (turn != 1)
C.S	C.S
Exit Code → turn = 1;	turn = 0;

P_1 P_2 P_3

Entry Section code

C.S

exit Section Code.

Semaphore:

- Counting ($-\infty$ to ∞)
- Binary (0, 1)

" Semaphore is an integer Variable which is used in mutual exclusion manner by Various Concurrent Cooperative processes in Order to achieve Synchronization. "

```
Down (Semaphore S) , wait , P
{
  S.value = S.value - 1;
  if (S value < 0)
  {
    put process (PCB) in
    Suspend list sleep();
  }
  else
    return;
  }
```

```
Up (Semaphore S) , V, signal
{
  S.value = S.value + 1;
  if (S.value ≤ 0)
  {
    Select a process
    from Suspend list
    wake up();
  }
}
```

Binary Semaphore

```
Down (Semaphore S)
{
  if (S.value == 1)
  {
    S.value = 0;
  }
  else
  Block this process
  and place in suspend
  list , sleep();
  }
}
```

```
up (Semaphore S)
{
  if (Suspend list is Empty)
  {
    S.value = 1;
  }
  else
  {
    Select a Process from
    Suspend list and
    wake up(),
  }
}
```

S = ∅ 1
S = ✗ 1

S = ✗ 0

P1 P2
Down Down
CS CS
up up
ready queue

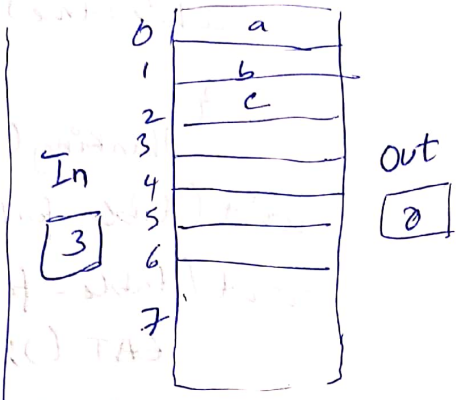
Sol

Counting Semaphore → full = 0 = No. of filled slots
→ empty = N → No. of empty slot

Binary Semaphore S = 1

Procedure - item (item p);

1) down (empty);
2) down (s);
3) Buffer [In] = itemp;
   In = (In+1) mod n;
4) up(s);
5) up (full);



Empty = 5 4
full = 3
S = 1 0

Consumer

down (full);
down (s);
item c = Buffer [out];
out = (out+1) mod n;
up(s)
up(empty);

Reader - Writer problem

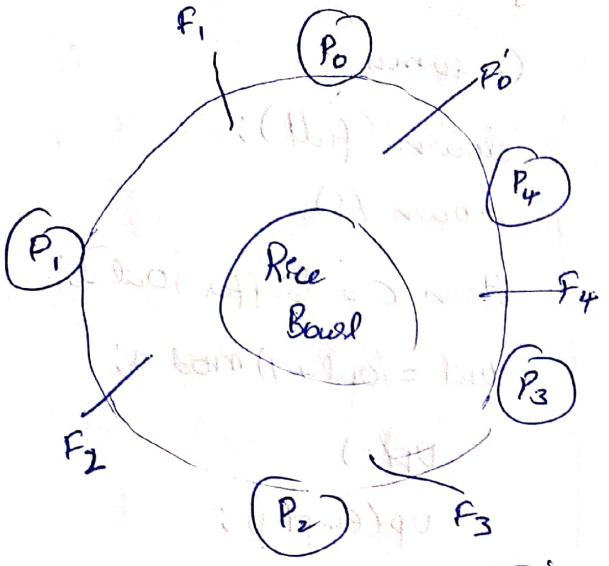
```
int rc=0
Semaphore mutex = 1;
Semaphore db = 1;
Void Reader (void)
{
  while (true)
  {
    down (mutex);
    rc = rc+1;
    if (rc==1) then down (db);
    up (mutex)
    DB (C.S)
    down (mutex)
    rc = rc-1;
    if (rc==0) then up (db);
    up (mutex)
    Process_data
  }
}
Void writer (void)
{
  while (true)
  {
    down (db);
    DB (C.S)
    up (db);
  }
}
```

rc - readcount

Exit code

R - W - Prob
W - R - "
W - W - "
R - R - no problem

Dining Philosophers Problem:



'5' philosophers
'5' fork's

Think
Eat

```
Void philosophers (void)
{
while (true)
{
Thinking();
wait (table_fork(S(i));)   ← left fork
wait (table_fork(S(i+1) %. N);)   ← Right fork
EAT ();
Signal( put fork (i); )
Signal (put fork ((i+1) %. N);)
}
}
```

→ Deadlock situation occur.

Change P_4 : S_0 S_4 to remove deadlock.

Soln

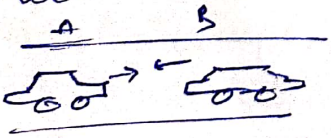
S[i] - five Semaphore

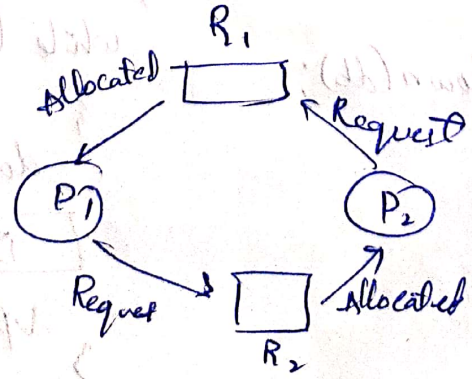
S_0	S_1	S_2	S_3	S_4
↓	↓	↓	↓	↓
1	1	1	0	1

P_0 : S_0 S_1
P_1 : S_1 S_2
P_2 : S_2 S_3
P_3 : S_3 S_4
P_4 : S_4 S_0

Deadlock:

if two or more processes are waiting on happening of some event, which never happens, then we say these processes are involved in deadlock then that state is called deadlock





P_1 P_2
S_1 → S_2
S_2 → S_1

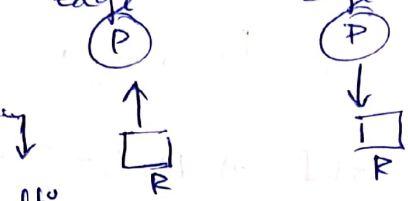
Cond^n

1) Mutual exclusion
2) No preemption
3) Hold & wait
4) Circular wait.

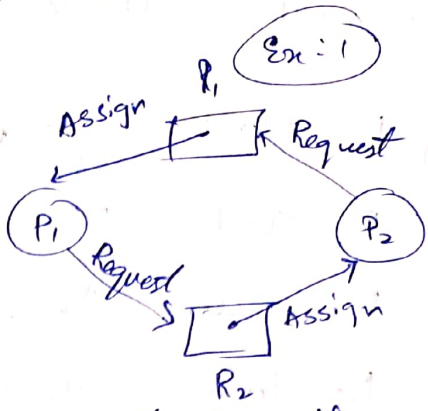
Resource Allocation Graph (RAG) (for dead lock detection).

Vertex
- Process Vertex (P_i)
- Resource Vertex
  - Single Instance — Eg: CPU, monitor
  - Multi Instance — Eg: Registers

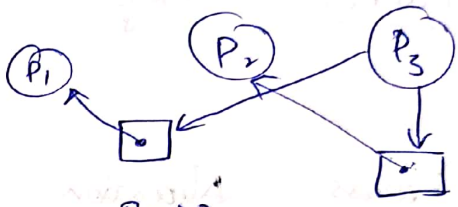
Edge
- Assign Edge
- Request edge



Ex: 1



Circular wait



Ex: 2

Ex: 1

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

Availability (R_1, R_2) (0, 0).

Ex: 2

	Allocate		Request	
	R_1	R_2	R_1	R_2
✗ P_1	1	0	0	0
✗ P_2	0	1	0	0
✗ P_3	0	0	1	1

Waiting
- finite (Starvation)
- infinite (Dead lock).

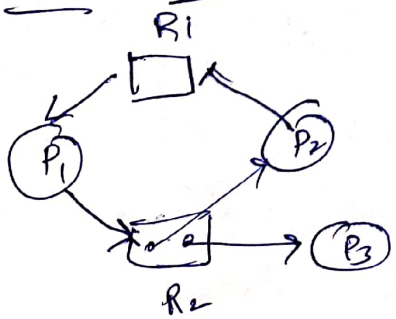
(0, 0) P_1 ✓
(1, 0) P_2 ✓
(1, 1) P_3 ✓

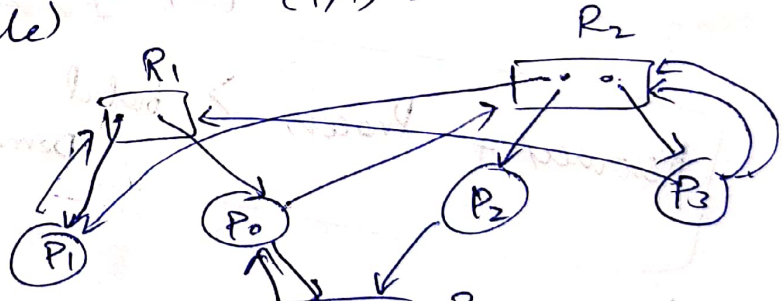
Single Instance Resource

If RAG has Circular wait (cycle)
→ Always Deadlock
→ True.

If RAG has no cycle then no deadlock = true.

Multi Instance RAG (Maybe for circular wait)





	Allocate			Request			Availability
	R_1	R_2	R_3	R_1	R_2	R_3	
✓ P_0	1	0	1	0	1	1	After exe (0, 0, 1)
✓ P_1	1	1	0	1	0	0	P_2 (0, 1, 1)
✓ P_2	0	1	0	0	0	1	P_0 (1, 1, 2)
P_3	0	1	0	1	2	0	P_1 (2, 2, 2)
							P_3 (2, 3, 2)

Deadlock Handling

1) Deadlock ignorance (ostrich method)
2) Deadlock prevention.
3) Deadlock Avoidance (Banker's Algo)
4) Deadlock detection & Recovery.

Banker's Algo: Total A=10, B=5, C=7

Deadlock Avoidance
Deadlock Detection.

Process	Allocation			Max Need			Available			Remaining Need			
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	0	1	0	7	5	3	3	3	2	7	4	3	P_1(4)
P_2	2	0	0	3	2	2	5	3	2	1	2	2	P_2(1)
P_3	3	0	2	9	0	2	7	4	3	6	0	0	P_3(5)
P_4	2	1	1	4	2	2	7	4	5	2	1	1	P_4(2)
P_5	0	0	2	5	3	3	7	5	9	5	3	1	P_5(3)
							10	5	7				

(3, 3, 2)

Safe Sequence

P_2 → P_4 → P_5 → P_1 → P_3

Resources + Process > total Demand

4 process
4 Resources

Memory Management

RAM 8MB, Process 4MB } 2 process can enter RAM

'K' = I/O operation (70)%

CPU = (1 − K) (30)%

2 process K^2 (70%)

CPU = $1 - K^2$ (76%)

4 process 'K' → K^4 (70%)

CPU utilization = $1 - K^4$ (93%)

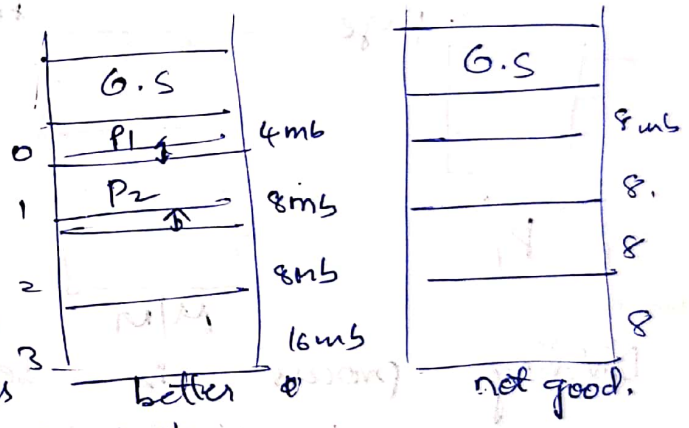
Increase the RAM, CPU Utilization (or) Efficient Paging will Increase.

Memory Management Techniques

- Contiguous
  - fixed partition
  - Various partition
- Non Contiguous
  - ~~Fixed~~
  - Multilevel Paging
  - Inverted Paging
  - Segmentation
  - Segmented Paging

Fixed Partitioning (Static Partition):

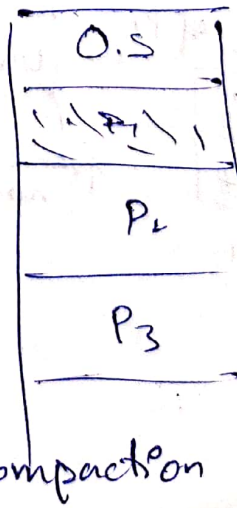
- No. of partition are fixed.
- Size of each partition may (or) may not Same
- Contiguous allocation so Spaning (no process divides b/w process) is not allowed.



P_1 = 2mb
P_2 = 7mb

1) Internal fragmentation (left space in fragment)
2) limit in process size
3) limitation on Degree of multiprogramming.
4) External fragmentation.

Dynamic Partitioning / Variable partitioning:



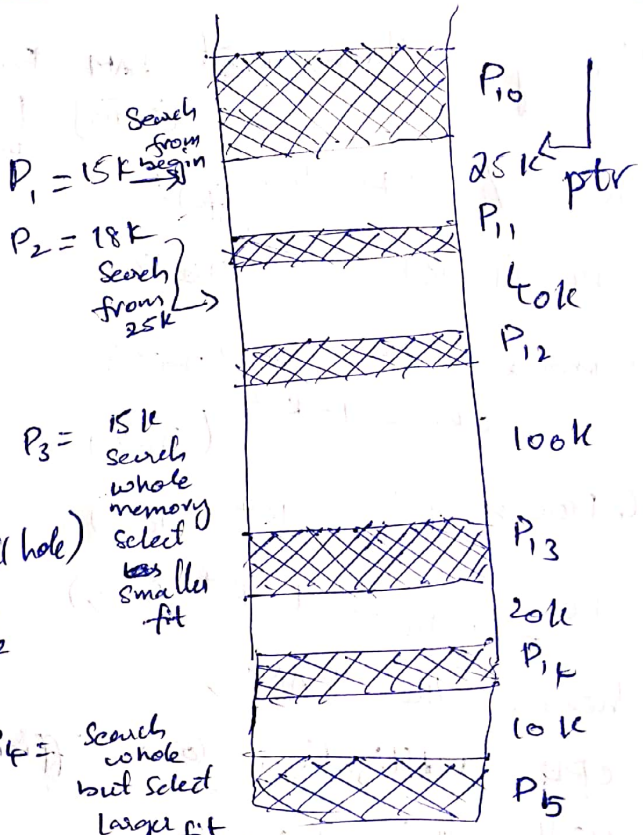
1) No Internal fragmentation
2) no limitation on no. of process.
3) No limitation on process size
4) External fragmentation Suffer. (Compaction used).

First fit: Allocate the first hole that is big enough

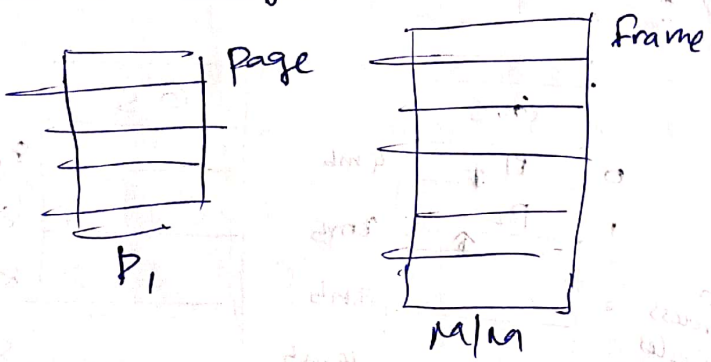
Next-fit: Same as first fit but start search always from last allocated hole.

Best fit: Allocate the smallest hole that is big enough (create small hole)

Worst fit: Allocate the largest hole



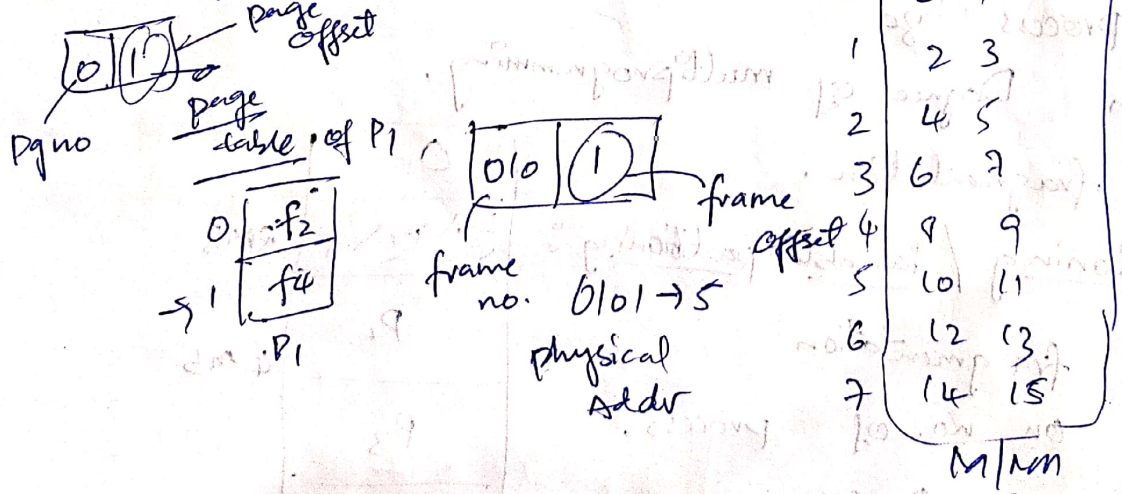
Non - Contiguous Mem. Aloc



Page Size = frame Size

Dividing process in several ~~process~~ pages / part

CPU → ~~physical~~ logical addr → physical addr → frame no.



Logical Address and Physical Address Space :

$2^{20} = 1M$ $2^{10} = 1K$

$2^{30} = 1G$

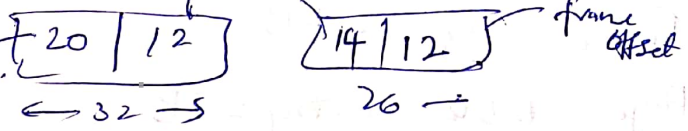
$2^{40} = 1T$

LAS = 4GB $\Rightarrow$ LA = $2^2 \times 2^{30}$ =

PAS = 64B

Page Size = 4kB $\Rightarrow$ $2^2 \times 2^{10} = 2^{12}$

page offset/size



No. of pages = 2^{20}

No. of frames = 2^{14}

no. of entries in PT = no. of pages in a process

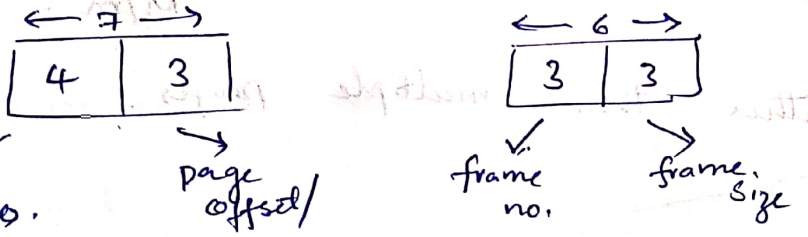
no. of entries in page table = 2^{20}

Size of page table = $2^{20} \times 14$ bits.

Consider a System which has

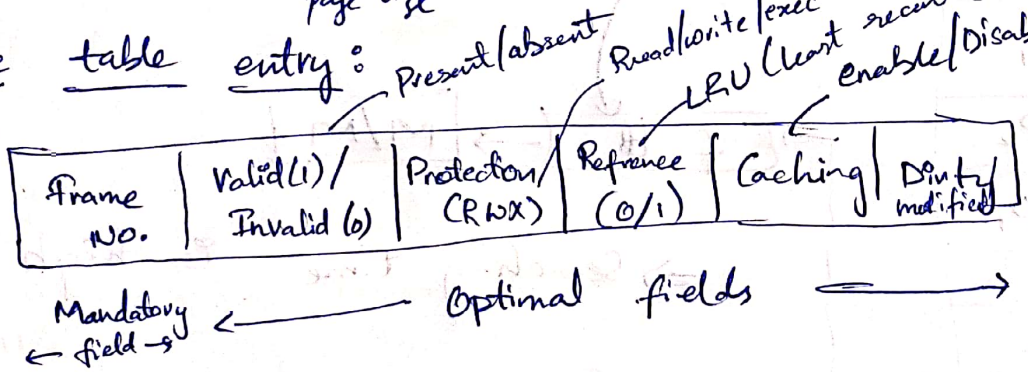
LA = 7bits, PA = 6bits, Page Size = 8 words/byte

then Calculate no. of pages & no. of frames.



no. of entries in P.T of a Process = no. of page in the process

Page table entry :



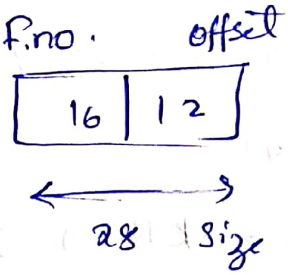
Two-level Page Table Scheme:

→ Physical address Space = 256 MB → 2^{28} B

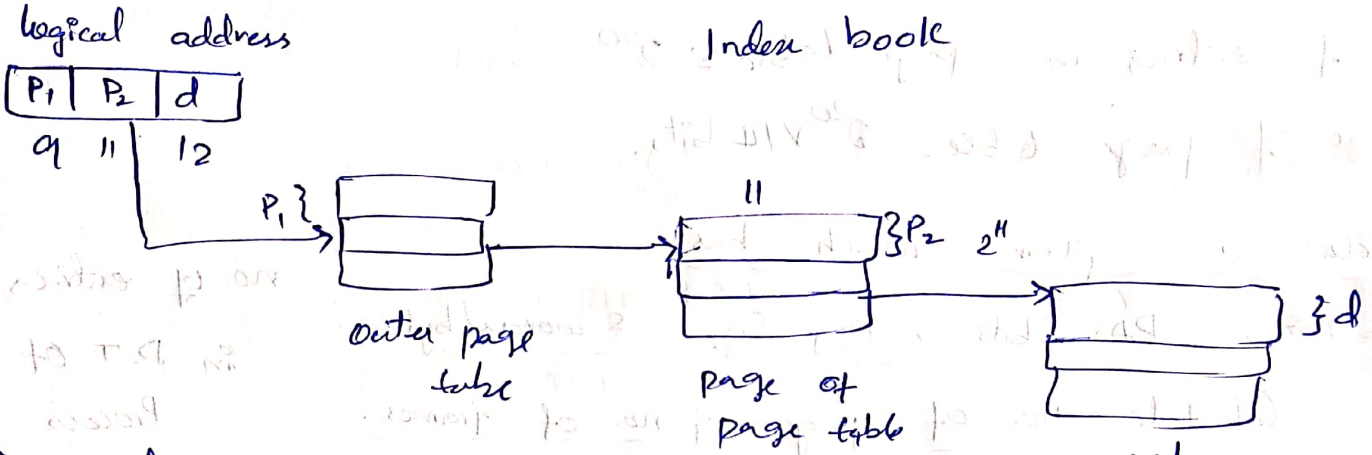
→ logical address Space = 4 GB

→ Frame Size = 4 KB

→ Page table Entry = 2 B

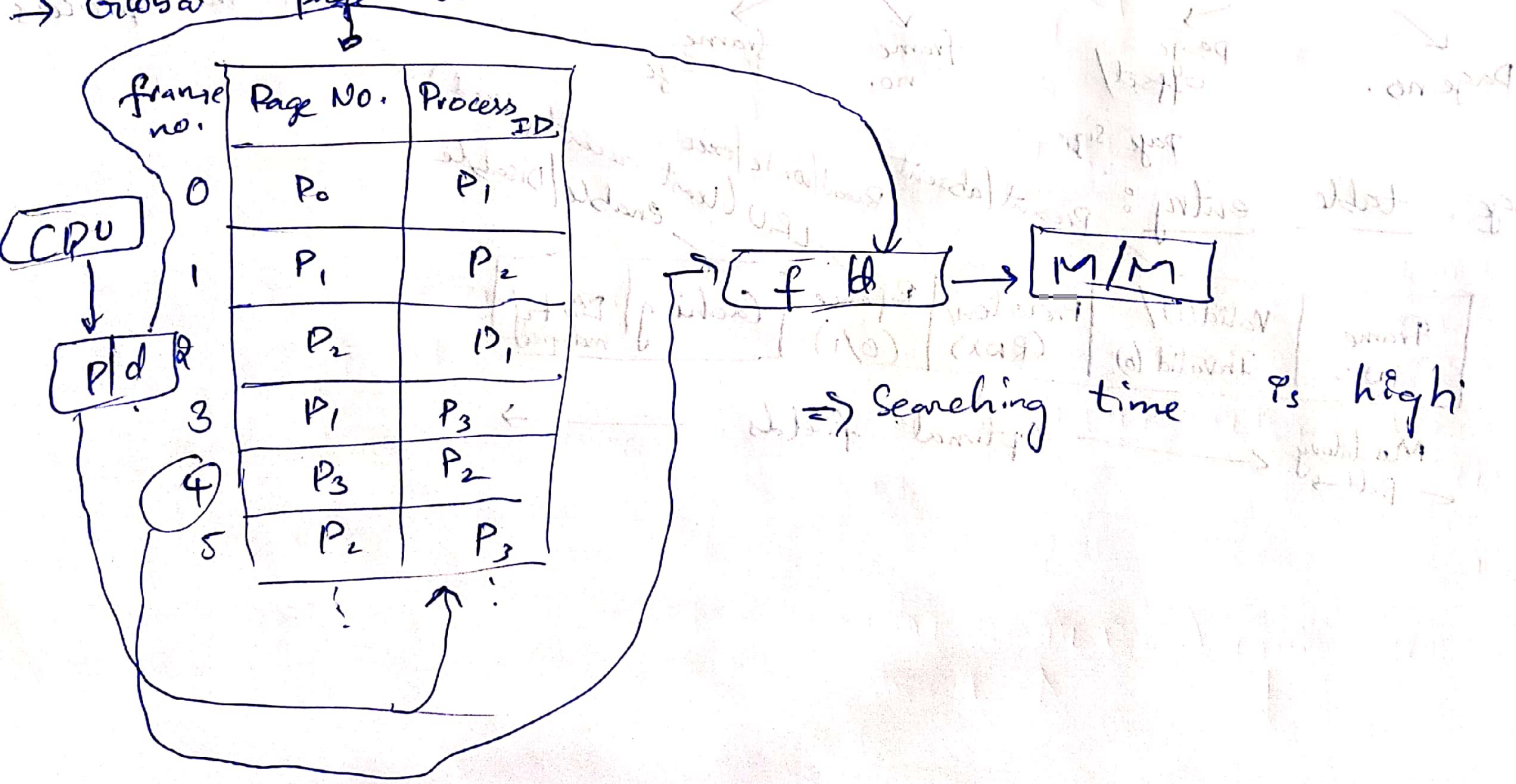


Address - Translation Scheme:



Inverted paging:-

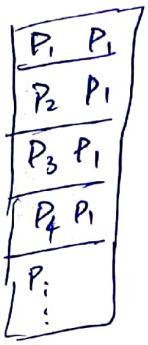
→ Global page table rather than multiple pages.



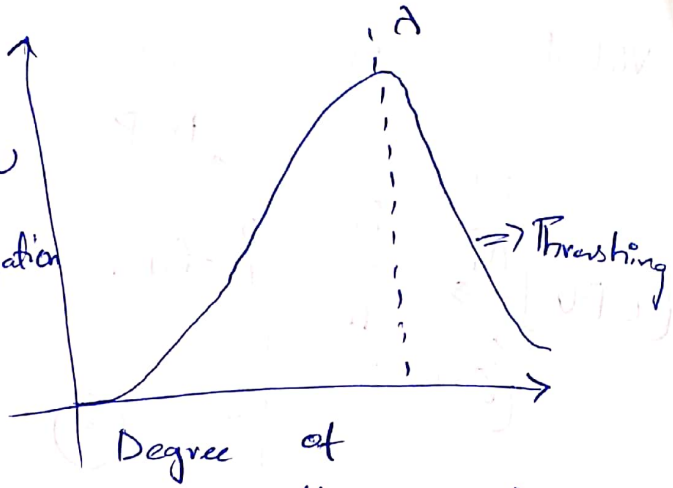
⇒ Searching time is high.

Thrashing :

→ To avoid page fault



Calling P_1 page 2 will get page fault.

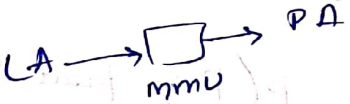


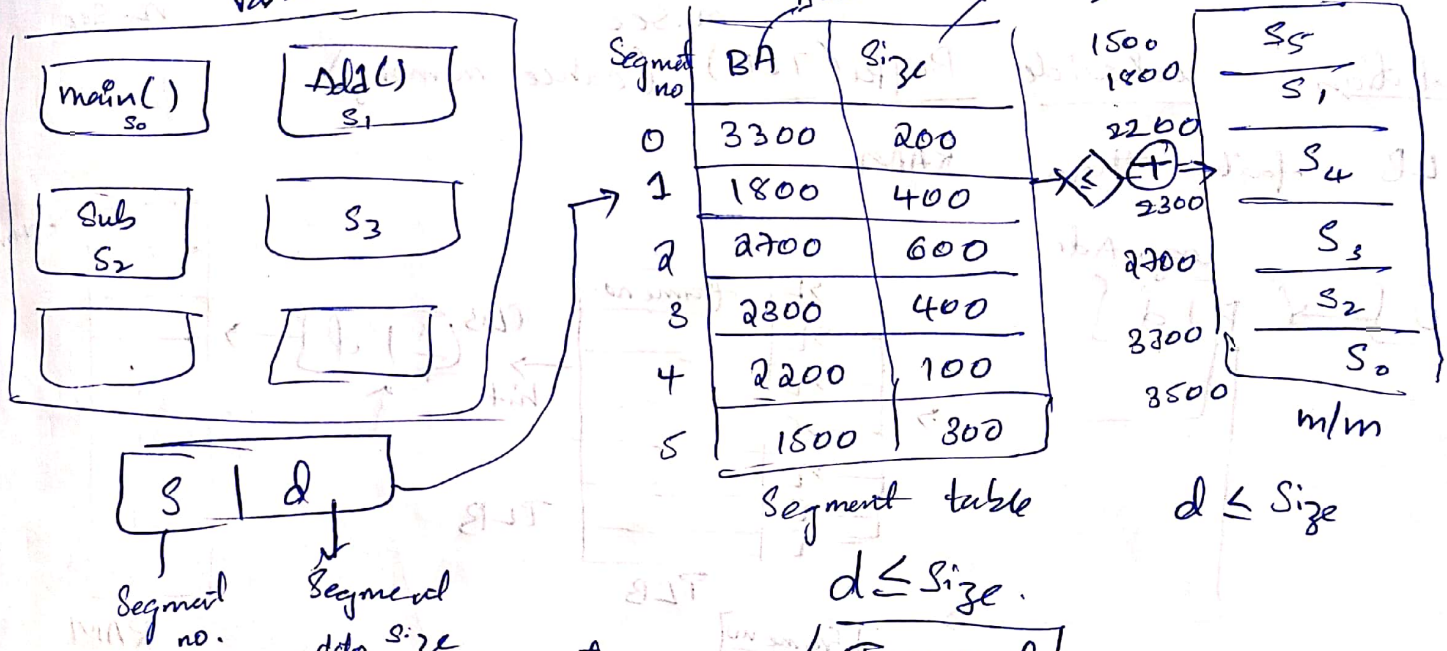
(To get more process in the m/m).

⇒ M/M size increased

2) long term Scheduler.

Segmentation :



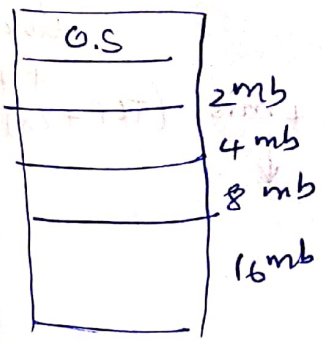


→ User related data get Segment

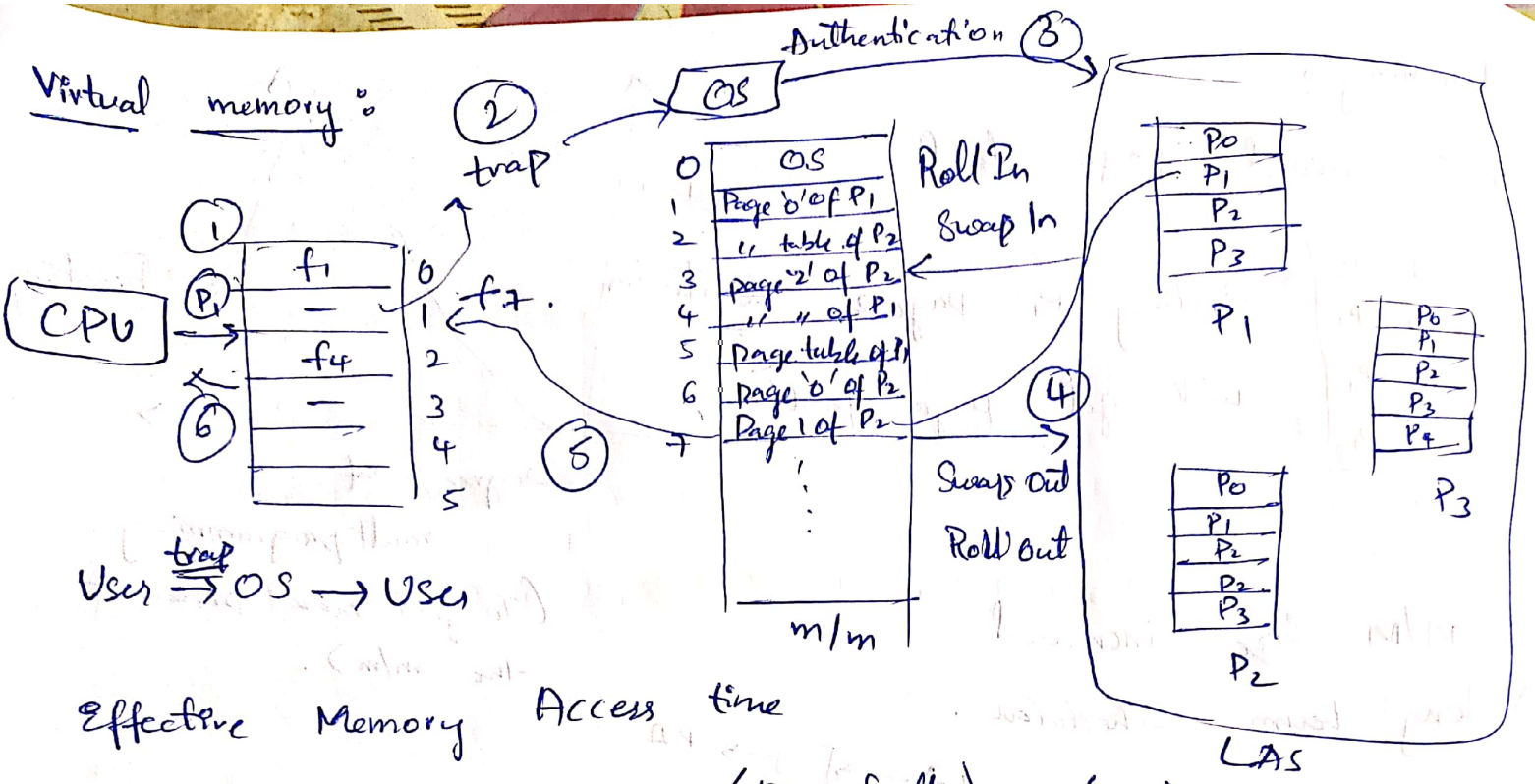
→ POV of User interest.

Overlay : Large size process can accomodate in the m/m.

⇒ when m/m is limited.



Virtual memory :



User $\xrightarrow{trap}$ OS → User

Effective Memory Access time

'p' → page fault

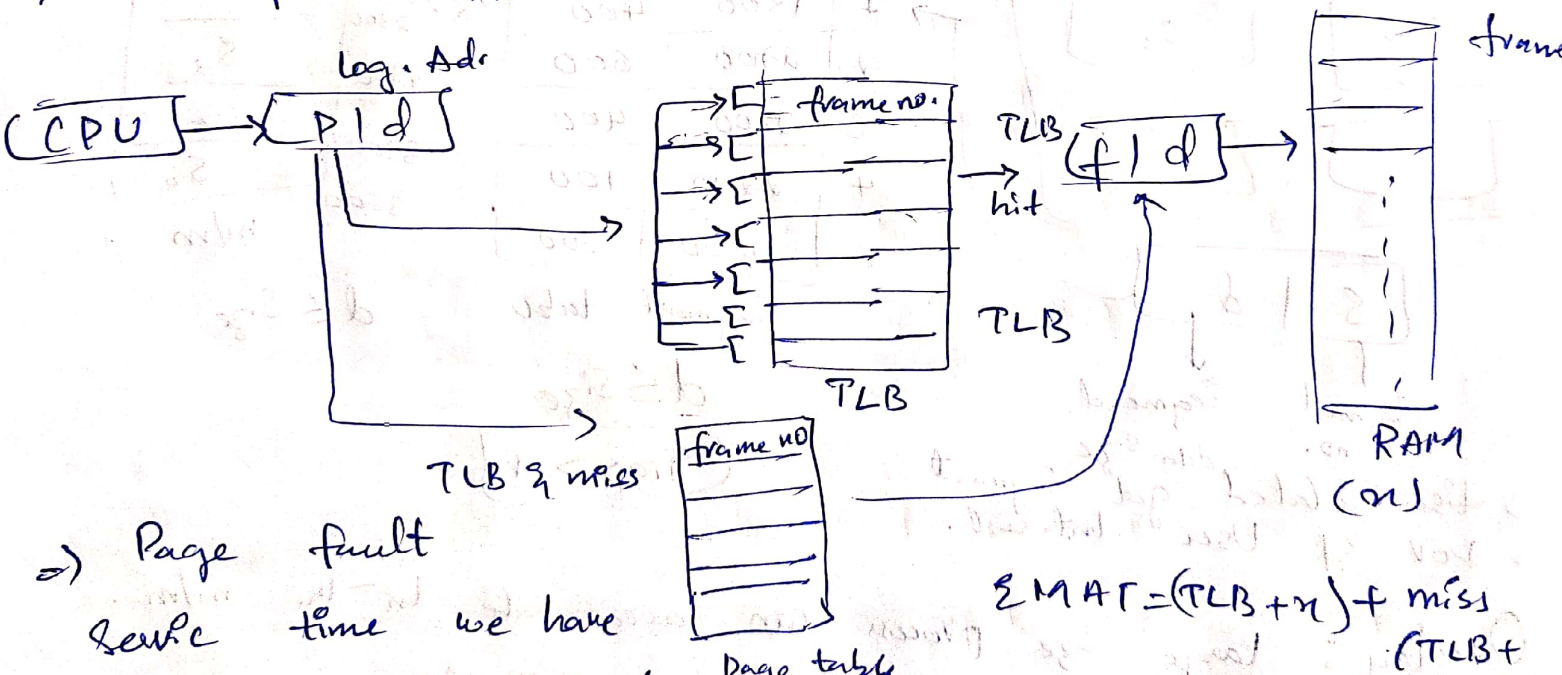
$EMAT = p(\text{page fault service time}) + (1-p)(\text{m/m Access time})$

page fault service time — m.sec

m/m Access time — n. sec

Translation lookaside Buffer (TLB) (cache memory)

→ TLB faster than RAM



⇒ Page fault service time we have get from the secondary memory.

$EMAT = (TLB + x) + miss$ (TLB + Page table (x))

$EMAT = \overset{\text{hit}}{(TLB + x)} + miss\ (TLB + 2x)$ (+ x)

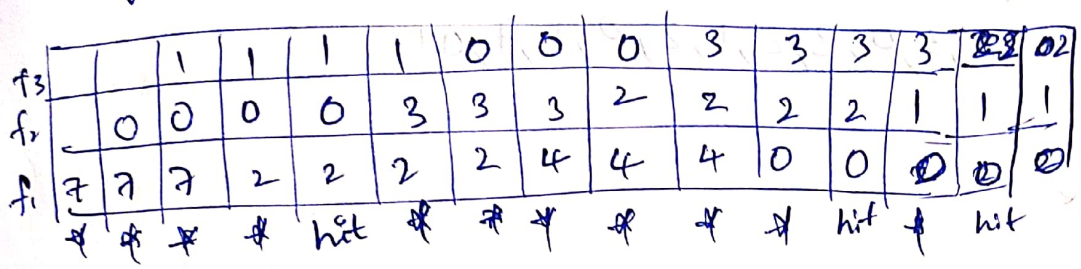
Page Replacement algo

1) FIFO
2) Optimal Page Replacement
3) least Recently Used (LRU).

Belady's anomaly
If you increase no. of frame then no. of page fault increase.

FIFO

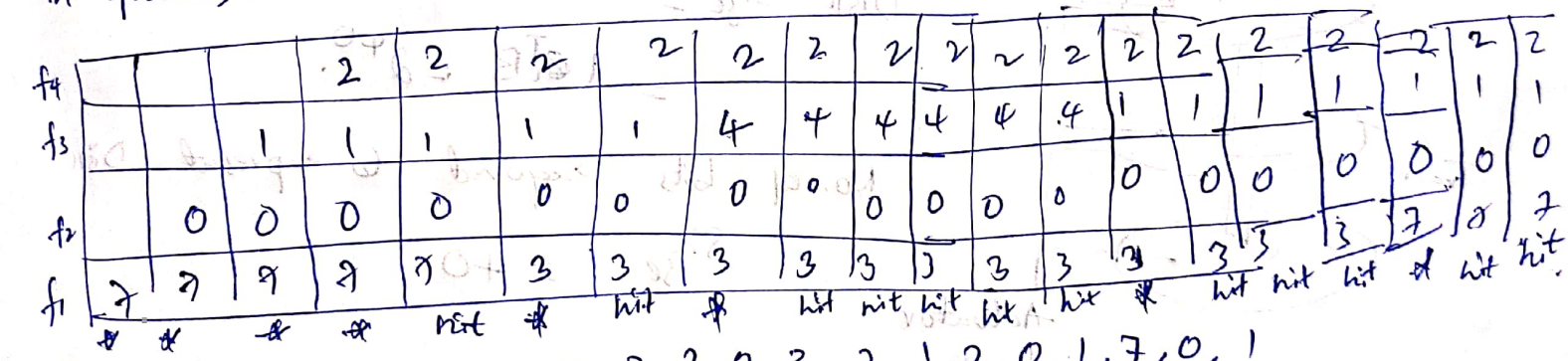
Reference string 7,0,1,2,0,3,0,4,2,3,0,3,1,2,0



Page hit = 3
page fault
hit/Ratio = 3/15 × 100.
miss.

Optimal Page Replacement

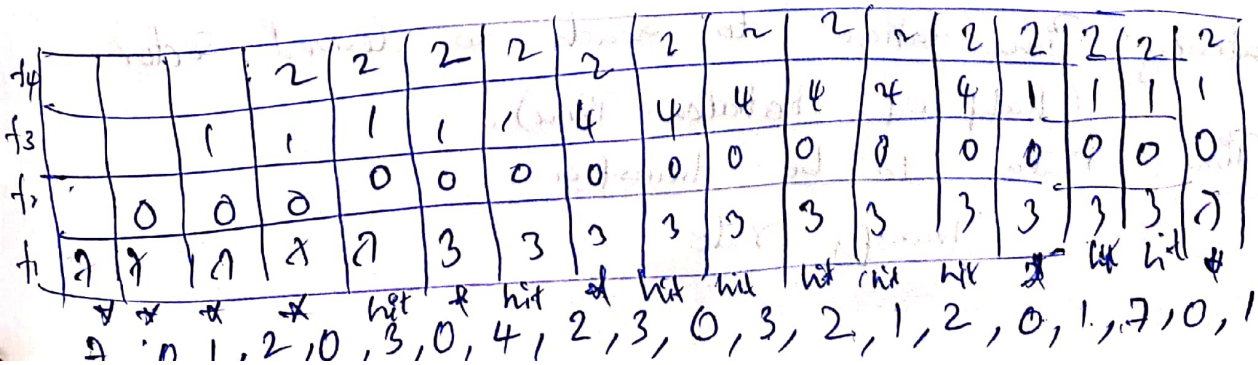
(Replace the page which is not used in longest Dimension of time in future).



String: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

page hit = 12 page fault = 8.

Least Recently Used (Replace the least recently used page in past)



7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1.

Most Recently Used

Replace the most recently used page in past.

2	2	2	2	2	2	3	0	3	2	2	2	0
1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	3	0	4	4	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7
*	hit	*	*	*	hit					hit	hit	*

0 1 4 7

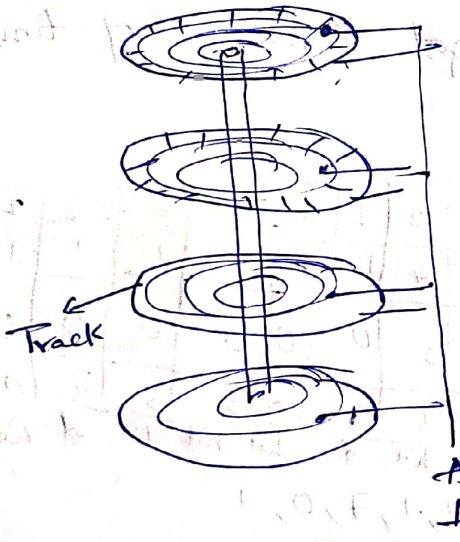
hit hit hit hit

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Disk Architecture

→ Uni direction
→ platters.

Platters → Surface → Track → Sectors → Data



$8 \times 2 \times 256 \times 512 \times 512\,KB$

$= 2^3 \times 2^1 \times 2^8 \times 2^9 \times 2^9 \times 2^{10} = 2^{40}$

Disk Size = P × S × T × S × D

= 1 TB = 2^{40}.

No. of bits required to represent Disk Size = 40

Disk Access Time:

1) Seek Time: Time taken by R/W head to read 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: $\frac{\text{Data to be transfer}}{\text{Transfer rate}}$

Transfer rate: { No. of heads × Capacity of One track × No. of Rotations in One Sec }

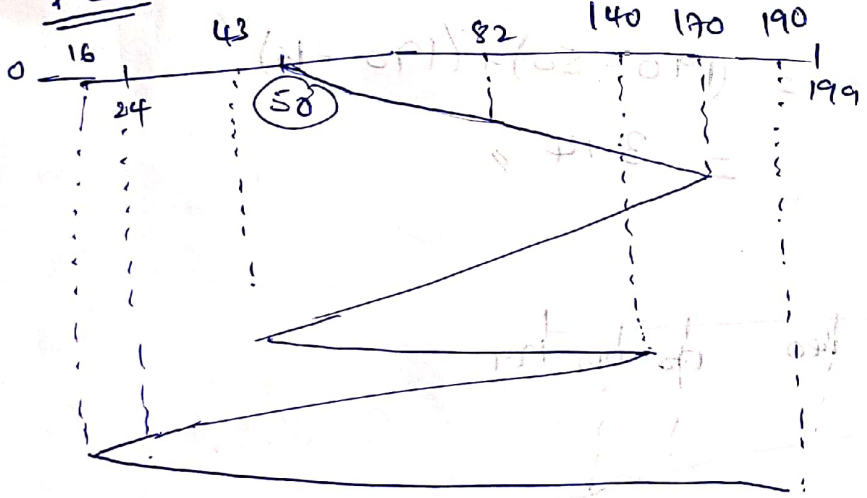
Disk Scheduling Algo:

Goal: To minimize the Seek Time.

- FCFS
- SSTF (Shortest seek time first)
- SCAN
- look
- CSLAN (circular SCAN)
- CLOOK (circular look)

Q A disk contains 200 track (0-199) Request queue contains track no. 82, 170, 43, 140, 24, 16, 190 respectively. Current position of R/w head = 50. Calculate total no. of tracks movements by head.

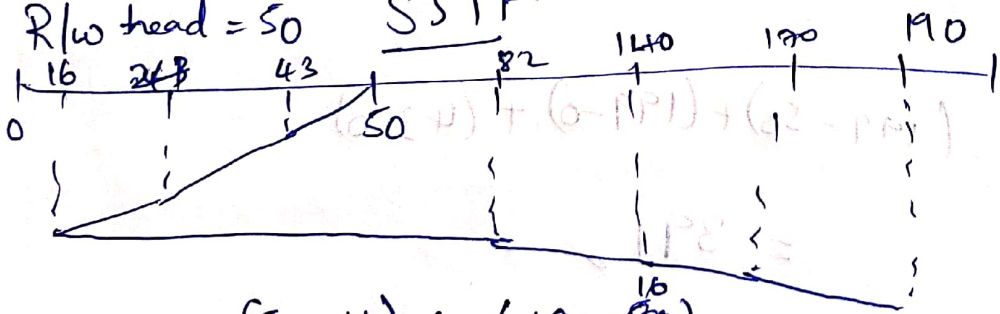
FCFS



(170-50) + (170-43) + (140-43) + (140-16) + (190-16)

= 642 ✓

R/w head = 50 SSTF



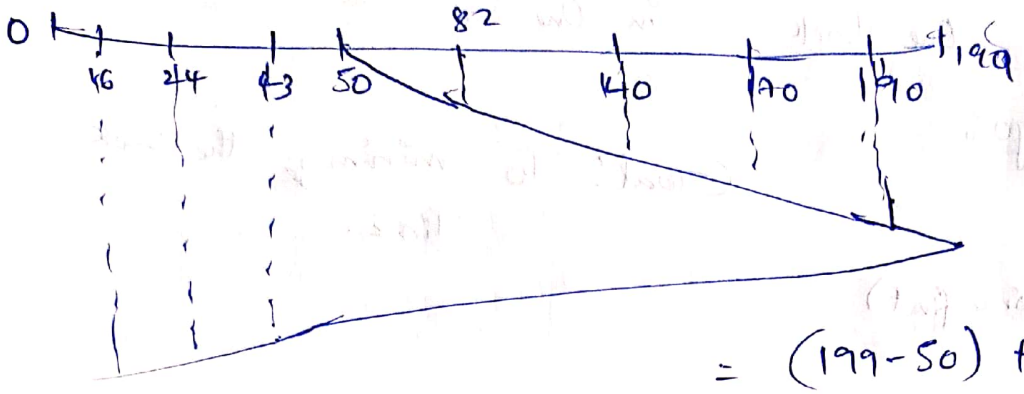
→ Starvation.
→ Overhead.

= (50-16) + (190-16)

= 34 + 140 = 174 ✓

= 208

SCAN algo

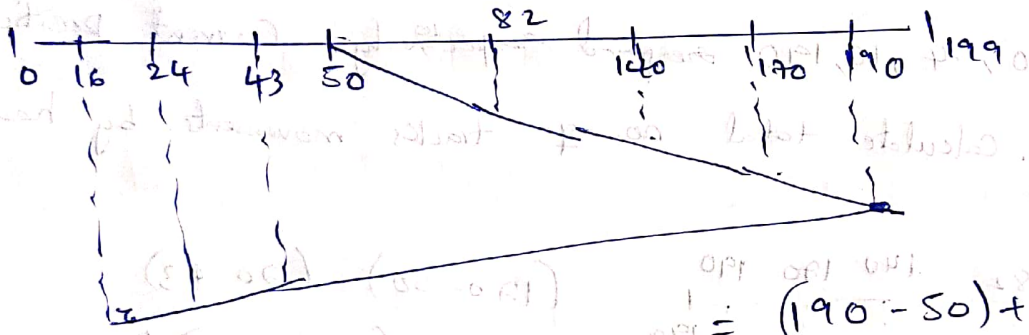


$= (199-50) + (199-16)$

$= 332.$

look algo

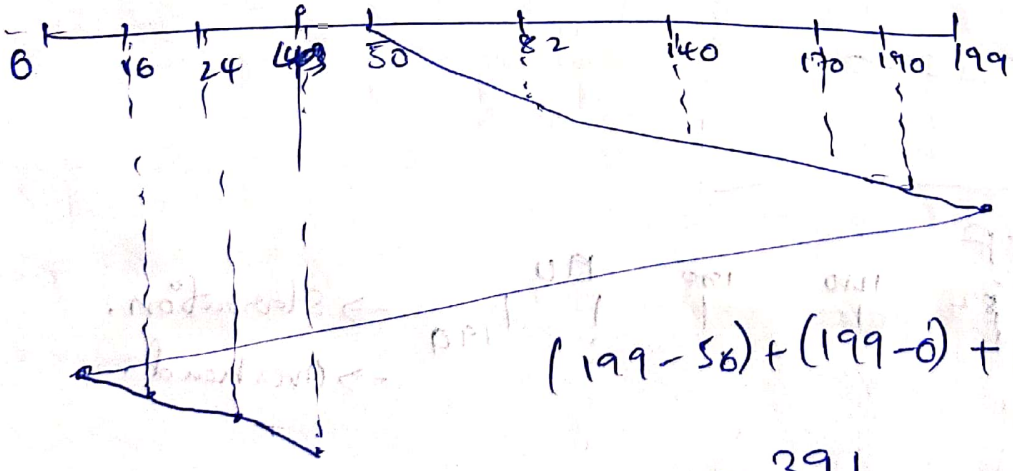
→ Direction is towards larger value.



$= (190-50) + (190-16)$

$= 314.$

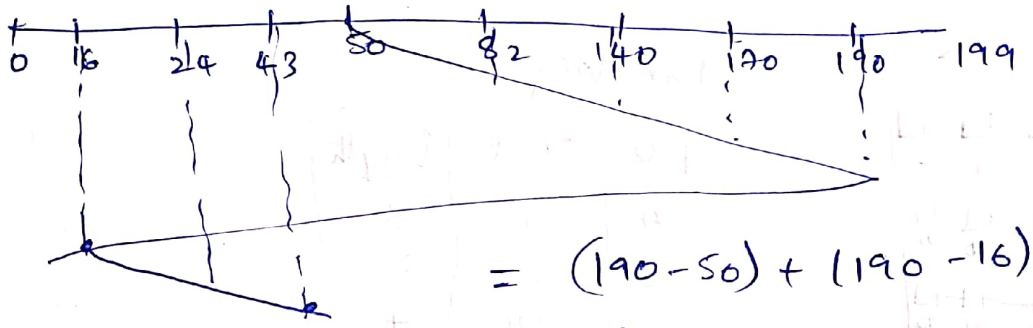
C-SCAN



$(199-56) + (199-0) + (43-0)$

$= 391.$

C-look algo :



$= (190-50) + (190-16) + (43-16)$

$= 341.$

File System In O.S :-

↓

Software (How data will stored (or) fetched).

Operation On Files :

1) Creating
2) Reading
3) Writing
4) Deleting
5) Truncating
6) Repositioning

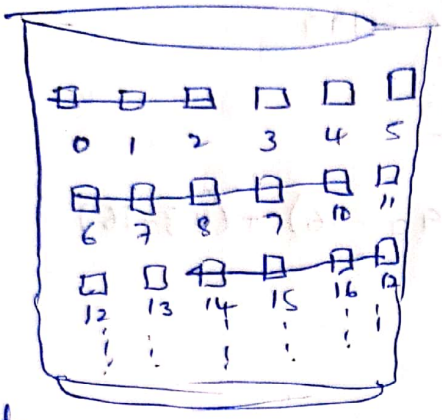
File Attributes :

1) Name
2) Extension (Type)
3) Identifies
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.

Contiguous Allocation :



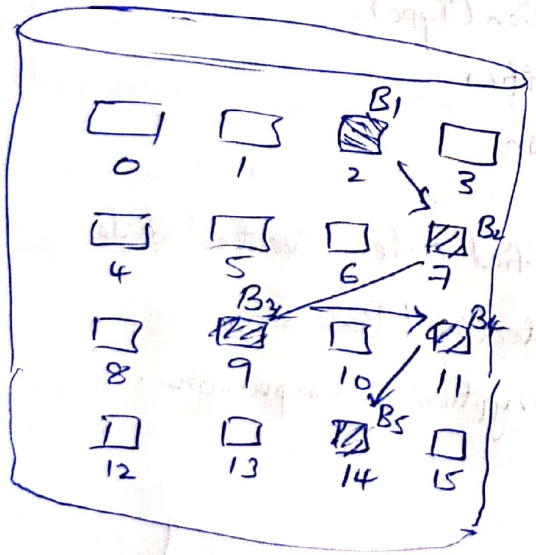
Directory

file	start	length
A	0	3
B	6	5
C	14	4

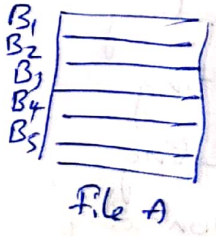
Adjvantags
→ Easy to Implement
→ Excellent Read performance

Disadj
1) Disk will become fragmented
2) Difficult to grow file.

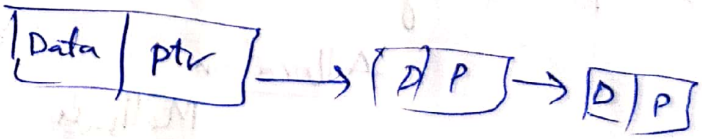
Linked list Allocation



file	Start
f1	2



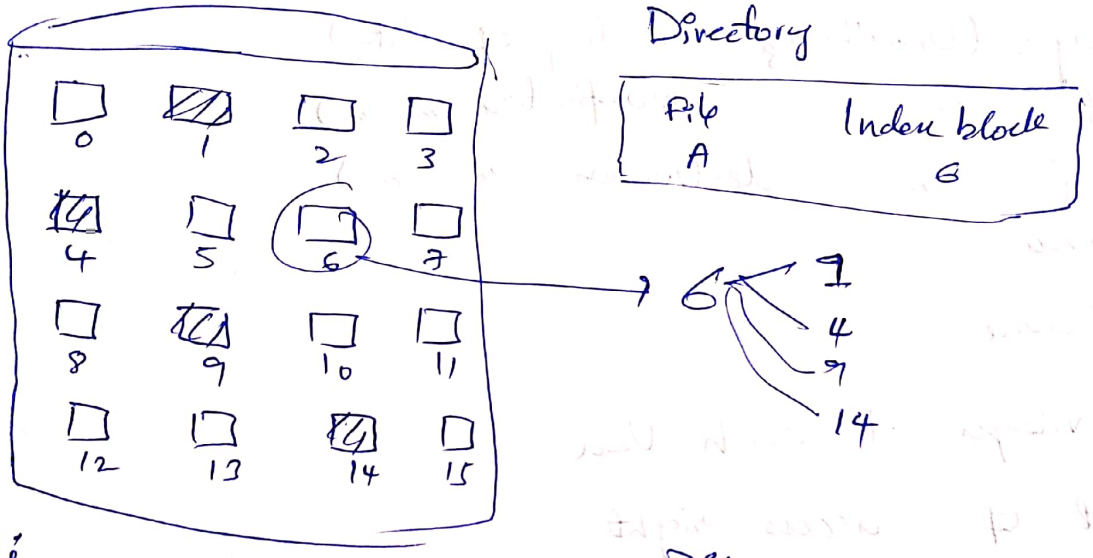
file A



Adj
→ No external fragmentation
→ File size can Increase.

Dis
1) large seek time
2) Random Access / Direct Acces Difficult
3) Overhead of pointers.

Indexed Allocation:



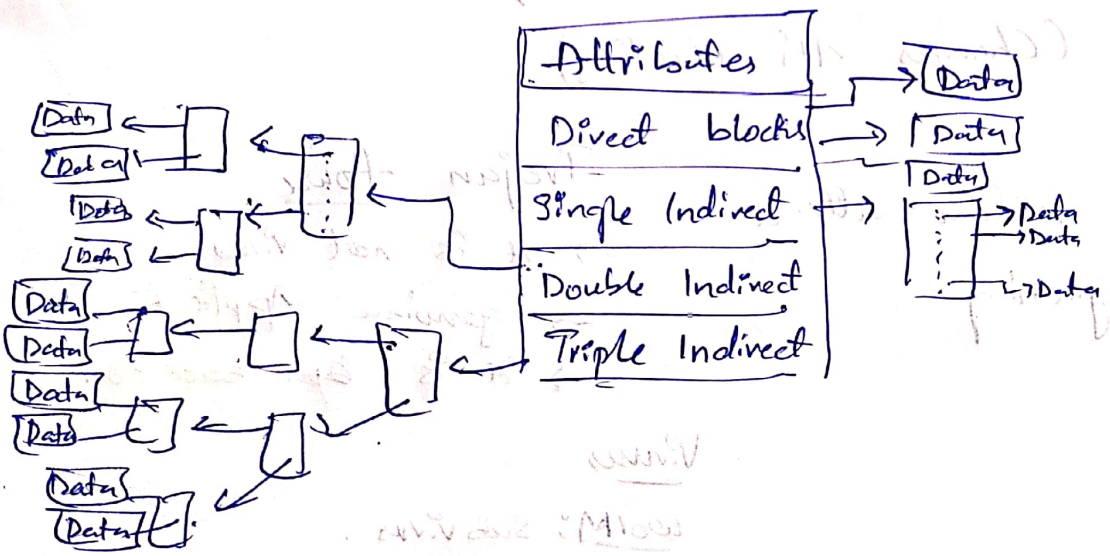
Adj

1) Support direct Access
2) No External fragmentation

Dis

1) pointer Overhead
2) Multilevel Index.

Unix INode Structure:



Protection and Security:

C I A
Confidentiality : (Unauthorized reading of data)
Integrity : (" modification " ")
availability : (" destruction " ")
Theft of service
Denial of Service

→ To set privileges to each User

Domain = set of access right
Access matrix
View protection as a matrix
threat is potential Security violation
Attack is attempt to breach Security.

Masquerading (Changes its identity)
Replay attack
Man in the middle attack
Session hijacking

4 Level
→ Physical
→ Human
→ O.S
→ Network.

Trojan Horse
→ It is not virus
→ genuine Application
→ It is open back door

Viruses
worm: Sub virus.
firewall:

Imp
→ Process state or PCB
→ Mutex Vs Binary Semaphore
→ process & thread
→ Monolithic Vs microkernel
→ System call
→ Demand paging
→ Virtual mem.
→ Chmod command
→ paging vs segmentation
→ Internal & external fragmentation.
→ preemptive vs Non-
→ Deadlock
→ Multiprogramming vs multitasking
→ Spatial Locality vs temporal Locality
→ Virus, worm, Trojan, malware, spyware