INTERPROCESS COMMUNICATION

Processes are of 2 types :-

1. Independent - ~~as~~ not affected by other processes.
2. Cooperating - affected by other processes

↳ need Interprocess Commn

"The best Vitamin to be a Happy"

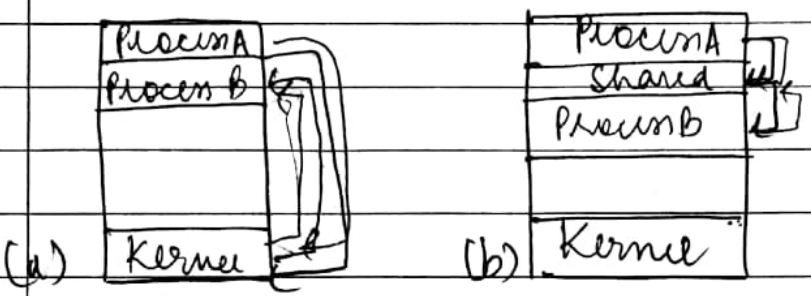
Evergreen

Cooperating process reasons- Date...../...../.....

- Info. sharing
- Computation speedup
- Modularity
- Convenience

Models of IPC

- Shared memory - msg passing



Msg Passing Shared memory

IPC provides 2 op^n

1. Send (msg)
2. Receive (msg)

If P & Q wish to communicate, they need to
- establish a comm^n link b/w them
- exchange msgs via send/receive

Implementation of comm^n ~~but~~ link

- Physical (eg. shared memory, hardware bus)
- Logical (eg. logical prop.)

Properties of comm^n link

- Link established only if processes share a common mailbox.

- Link established only if processes share a comm. mailbox
- A link may be ass with many processes
- Each pair of processes may share several comm^{n} links.
- Link may be Uni'directional or Bi'directional

Date...../...../.....

Operations

→ Creat a new mailbox
→ Send/receive msg through mailbox
→ destroy a mailbox

Busy Waiting – When a process repeatedly checks to see if a cond^{n} is true.

DEADLOCK

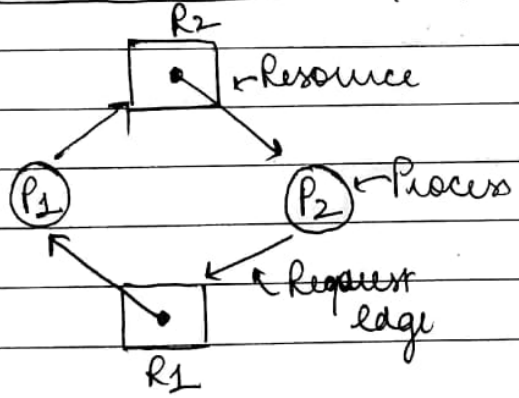
Date...../...../.....

In a multiprogramming system, a no. of process compete for limited no. of resources & if a resource is not available at that instance then process enters into waiting state.
If a process is unable to change its waiting state indefinitely because the resources requested by it are held by another waiting process, then system is said to be in deadlock.

System model

→ Every process will request for the resource
→ If entertained, then process will use the resources
→ Process must release the resource after use

Resource Allocation Graph

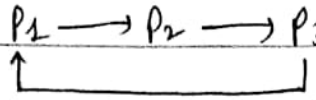


Necessary Condn for Deadlock

1. Mutual Exclusion – At least one resource type in the system which can be used in non sharable mode i.e mutual exclusion (one at a time) eg. Printer

2. Hold and wait – A process is currently holding

at least one resource and requesting Date...../...../.....
add^{n}al resources which are being held by other processes.



~ Jitni ~~process~~ resource uss time pe available hai utni hold kar lena baaki resources ke liye wait karna

3. ~~Non~~ No pre-emption - A resource cannot be preempted from a process by any other process resource can be released only voluntarily by the process holding it.

~ Resources ko chhina (forcefully) nahi jaa sakta. Process apne ichaanusar hi ~~pro~~ resource release kar sakte hain.

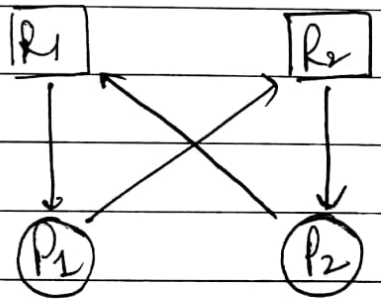
4. Circular wait - Each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

~ Circular fashion me resource waiting honi chahiye.

DEADLOCK PREVENTION (Circular Wait)

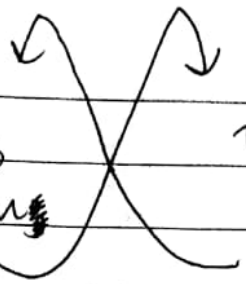
→ Circular wait can be eliminated by first ~~going~~ giving a natural no. of every resource.
→ allows every process to either run only in increasing or decreasing order of the resource no.
→ If a process requires a lesser no. of process (in case of increasing order) then it must release all the resources with no. larger than req'd.

Saare processes ~~ek hi~~ resources ko Date...../...../.....
ya to increasing ya decreasing order mei hi request kar sakte hain.
~~kisi process ko~~
Jaise agar P_1 ko R_2 ~~chahiye~~ ko resources iss order mei chahiye:
$P_1 (R_2, R_1)$ aur $P_2 (R_1, R_2)$.



toh phir deadlock ka case h.

Par agar dono ka resource sqst ek hi order m h jaise $P_1 (R_2, R_1)$ aur $P_2 (R_2, R_1)$
aur R_2, P_2 ke paas pehle hai toh P_1 pehle wait karega R_2 ke liye kyunki wo direct pehle P_1 ko use nahi karega.



P_1	P_2
while (S1 == S2);	while (S1 != S2)
critical section	critical section
S1 = S2;	S2 = not (S1);

Date...../...../.....

~~ME~~ P
~~ME~~ P
~~ME~~ P
~~ME~~ P

- Strictly alternating fashion
  - always mutual exclusion
  - No Progress

Race Condition - When order of execution can change result.

```
P()
{
  R(a)   <- context switch
  a = a+1
  W(a)
}
```

P_1	P_2
10	
	11
11	

Critical Section is that area of program where shared resources are present.

Solving Critical Section Problem

✗ (1) Mutual Exclusion - Ek baari mei ek hi process critical section mei ja sakti hai.

✗ (2) Progress - Round Robin fashion mei process allocate hoti hai critical section mei jaane ke liye. To Progress kehta hai ki bas unhe hi jaana chahiye jinhe zaroorat ho.

3) Bounded wait – After a particular time, process should be allowed to go to critical section

Mutual exclusion principle

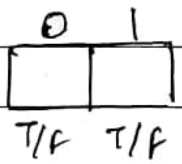
Po ensures mutual exclusion	Po turn=0.	P1
Po	while (1)	while (1)
{	{	{
entry section	while (turn != 0);	while (turn != 1);
C.S	critical section ←	critical section
exit section	turn = 1;	turn = 0;
remainder section	remainder section	remainder section
}	}	}

flag[0] = flag[1] = F

Progress principle

Po	P1
while (1)	while (1)
{	{
flag[0] = T	flag[1] = T
while (flag[1]);	while (flag[0]);
critical section	critical section
flag[0] = F	flag[1] = F
}	}

DUE
2) Progress X



PETERSON PROBLEM

We use both turn as well as flag.

Po	P1
while (1)	while (1)
{	{
flag[0] = T	flag[1] = T
turn = 1	turn = 0
while (turn == 1 && flag[1] == T);	while (turn == 0 && flag[0] == T);
critical section	critical section
flag[0] = F	flag[1] = F

"The best Vitamin to be a Happy"

Scanned by CamScanner

Date...../...../.....

S_1, S_2 : boolean values	
P1	P2
while ($S_1 == S_2$);	while ($S_1 != S_2$);
critical section	critical section
$S_1 = S_2$;	$S_2 = not(S_1)$;

Process X	Process Y
while (T)	while (T)
{	{
Var P = T	Var Q = T
while (Var Q == T);	while (Var P == T);
CS	CS
Var P = F	Var P = T
}	}

Semaphore - A semaphore is an integer variable that apart from initialization, is accessed only through two standard atomic operations.

(i) wait(S)

```
wait(S)
{
  while (S<=0);
    S=S-1;
}
```

decrements the value by 1.

(ii) Signal(S)

```
Signal(S)
{
  S=S+1;
}
```

increments the value by 1.

Process w and x increment value by one. Date ..... / ..... / .....
while process Y, Z increment value by 2. Each semaphore
If S is initialized to two, find max possible value of w

S = 2

	w	x	y	z
	wait(S)	wait(S)	wait(S)	wait(S)
	R(x)	R(x)	R(x)	R(x)
	w = w+1	x = x+1	x = x-2	x = x-2
	W(x)	W(x)	W(x)	W(x)
	signal(S)	signal(S)	signal(S)	signal(S)

a) -2
b) -1
c) 1
d) 2

context switch

Order: w, Y, Z, x, w

Other benefits of semaphores.

- For deciding order of execution

$S_1, S_2 = 0$

$S_1 = -1$	$S_1 = +1$	$S_2 = -1$
wait(S_1)	P_2	wait(S_2)
P_1		P_3
signal(S_2)	signal(S_1)	

If we want execution order to be $P_2 \rightarrow P_1 \rightarrow P_3$ then, we can assign wait & signal as

- For managing resources

P_i (
wait(S)
C.S
signal(S)
R.S
)

S = 5

Processes = $P_1 P_2 P_3 P_4 P_5$

$P_1 P_2 P_3 P_4 P_5$

Process P	Process Q
while(1)	while(1)
P(S) W P(S) *	P(T) Y
print '0'	print '1'
print '0'	print '1'
V(S) X	V(T) Z

Should Print.

'0011001100 11'

	W	X	Y	Z	
a)	P(S)	V(S)	P(T)	V(T)	S,T=1
b)	P(S)	V(T)	P(T)	V(S)	S=1, T=0
c)	P(S)	V(T)	P(T)	V(S)	S,T=1
d)	P(S)	V(S)	P(T)	V(T)	S=1, T=0

Semaphore is a synchronization tool

Producer Consumer Problem

Producer - insert data

Consumer - removes data

Semaphores

S=1 E=n F=0

Producer	Consumer
do{	do{
wait(E); n-1	wait(F); F=n-1
wait(S); S=0	wait(S); S=0
//Add item to Buffer	//Consume items
Signal(S); S=1	Signal(S); S=1
Signal(F);	Signal(E); E=1
}while(true);	}while(True);

READER WRITER PROBLEM

For Writer wrt=1

wait(wrt)

write operation

Signal(wrt)

E=0

F=n

for reader

Date..... /..

for entering

```
wait (mutex)
  readcount++
if (readcount == 1)
    wait (wrt)
signal (mutex)
    read op^n
```

for exit

```
wait (mutex)
  readcount--
if (readcount == 0)
  signal (wrt)
signal (mutex)
```

Now write op^n can be performed.

Deadlock Avoidance: BANKER'S ALGORITHM

Data Structures :-

n = no. of processes

m = no. of resource types

1. Available$_m$: if Available[j] = k = there are k instances of resources Rj available

   Available[R_2] = 10 : 10 instances of res. R_2 available.

2. Max$_{n \times m}$: if Max[i, j] = k, process i may request at most k instances of resource j

   ~~Max[R2]~~ Max[P_1, R_2] = 5 P_1 can request max of 5 ~~resources~~ instances of res. R_2.

3. Allocation : Allocation[i, j] = k

   Allocation [P_1, R_2] = 2 P_1 has 2 instances of res R_2

Need$_{n \times m}$ Need$[P_1, r_2] = 3$ Date...../...../.....

P_i needs 3 instances of resource r_2

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

↳ Request Resource Algo

↳ Safety algo.

Request-Resource Algo

Let Process $P_i \rightarrow Request_i$

(i) If $Request_i \leq Need_i$, go to step 2 else error.

(ii) If $Request_i \leq Available$, go to step 3 else wait

(iii) $Available = Available - Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

(iv) Check if the new state is safe or not.

Safety Algorithm

(i) Work = Available, Finish = False

(ii) Find an 'i' such that Finish[i] = False and $Need_i \leq Work$

If no such i, go to step 4.

(iii) Finish[i] = True

$Work = Work + Allocation_i$; Go to step 2.

(iv) If Finish[i] = true for all i, then system is safe.

Q1 = Ex of Banker's Algorithm

Date...../...../.....

5 Processes (P_0, P_1, P_2, P_3, P_4)

3 resource types ⟶ C (7 instances)

↙ A (10 instances) ↘ B (5 instances)

Snapshot at time T0

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Soln Need matrix.

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

i) Work = ~~Available~~ = ~~[3,3,2]~~

~~Finish$_i$ = false~~

i) Calculate the need matrix

need = max − allocation

(ii) Set work = Available = [3 3 2]

Finish$_i$ = false

(iii) Check the safe state.

i=0

P_0 can't be allocated resources at the moment.

[∵ Bo Need of P_0 > Work].

i=1

P_1 finish[1] = true

Work = Work + Allocation = [3 3 2] + [2 0 0] = [5 3 2].

i=2, P_2 ✗ [Need of P_2 > work]

i=3, finish[3] = true Date...../...../.....

work = [5 3 2] + [2 1 1] = [7 4 3].

i=4, finish[4] = true.

work = [7 4 3] + [0 0 2] = [7 4 5].

i=0, finish[0] = true

work = [7 4 5] + [0 1 0] = [7 5 5].

i=2, finish[2] = true

work = [7 5 5] + [3 0 2] = [10 5 7]

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$ Safe Sequence ✓

Q2) If process P_1 requests (1 0 2) check whether this will be safe state or not.

Banker's Algo

5 Processes (P_0 ... P_4)

Resources — A(10), B(5), C(7)

$P_1 \rightarrow R(1\,0\,2)$

Need $P_1 \rightarrow (1\,2\,2)$

Available = (3 3 2) − (1 0 2)

= (2 3 0)

Allocation(P_1) = (2 0 0) + (1 0 2)

= (3 0 2)

Need(P_1) = (1 2 2) − (1 0 2)

= (0 2 0)

Snapshot at time T0:-

	Allocation	Max	Available
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Need Table

	A	B	C
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Safety algo

i) Work = (2 3 0), ~~finish[i] = false~~

Safety algorithm

i) work = Available (2 3 0) finish = false

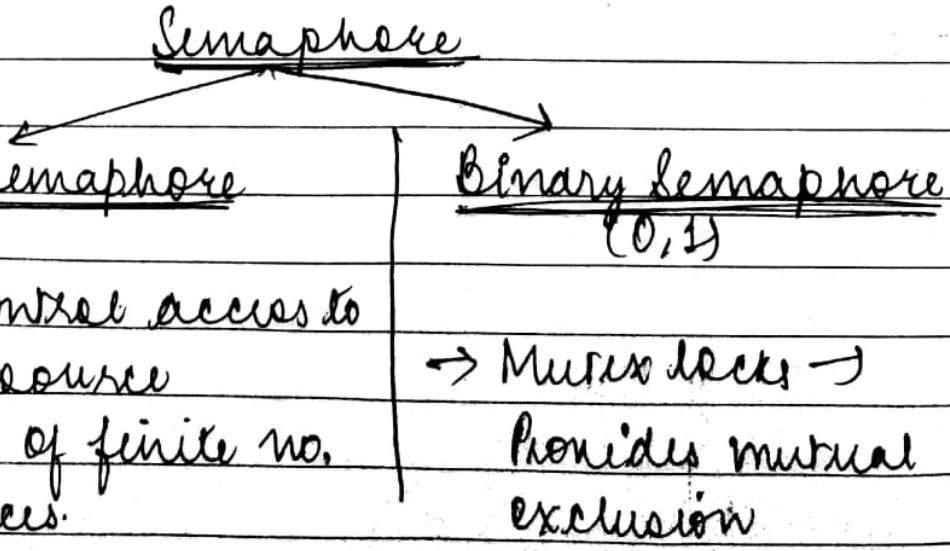
i=0 ✗
i=1 ✓ ~~Ar~~ ~~Work~~ = Available = [2 3 0] + [2 0 0]
= [4 3 0]

i=2 ✗
i=3 ✓ Available = [4 3 0] + [2 1 1] = [6 4 1]
i=4 ✓ Available = [6 4 1] + [0 0 2] = [6 4 3].
i=0 ✗
i=2 ✓ Available = [6 4 3] + [3 0 2] = [9 4 5]
i=0 ✓ Available = [9 4 5] + [0 1 0] = [9 5 5]

Safe Sequence: $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_0$.
It is safe

SEMAPHORES

→ Synchronization tools
→ an integer variable, that apart from initialization is accessed only through two std. at. op^{n}s.



Disadvantages :-

Date...../...../.....

↳ Busy Waiting - When a process is in its C.S, only other ~~process~~ processes that try to enter its C.S. must loop continuously in its entry code.

Memory Management

It is the functionality of an OS which manages primary memory and moves processes back & forth btw main memory and disk during execution.

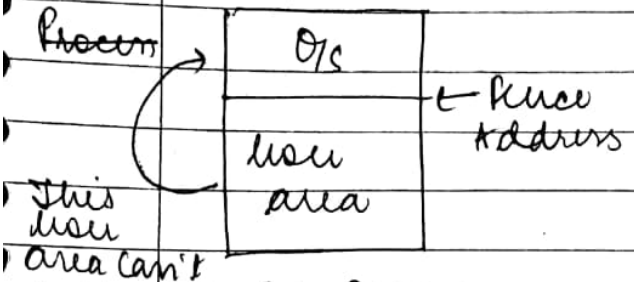
Functions of memory management

1. Keep track of every memory location.
2. Track of whether memory is allocated or not.
3. Track of how much memory is allocated.
4. ~~Takes~~ Takes decision which process will get memory and when.
5. Updates the status of memory location when it is allocated or freed.

Protection

Date...../...../.....

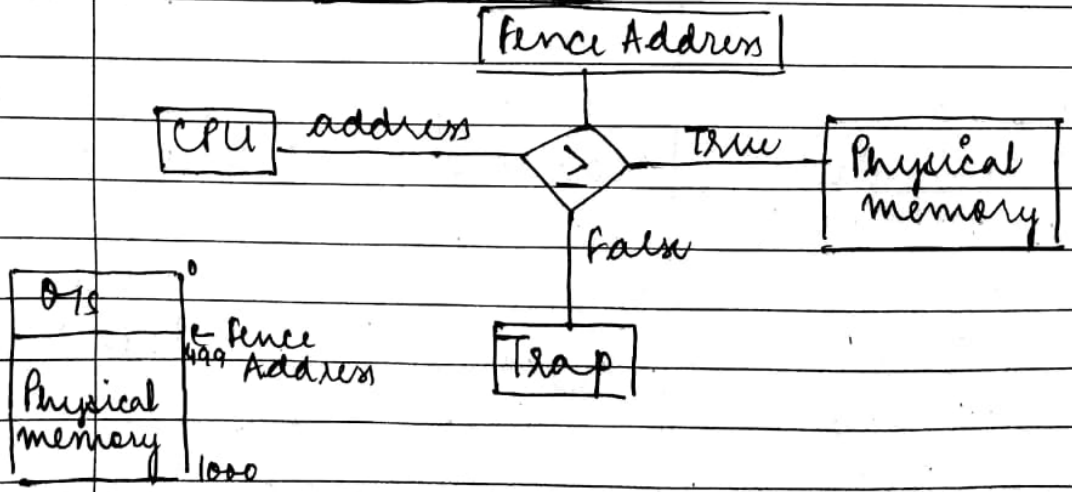
Process can't access O/S and other process memory loc?



Goals of Memory Management

1. Space Utilization
   - Fragmentation → memory loss can occur
   - tries to keep fragmentation as low as possible.
2. Run larger program in smaller memory area
   - can be achieved by Virtual memory.

Fence Address

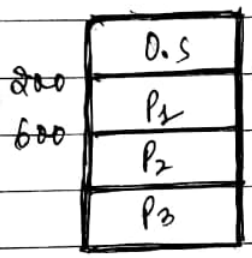


Memory has two parts: (1) O/s (2) User area. Fence Address takes care that user area can't access O/s area.

Date...../...../.....

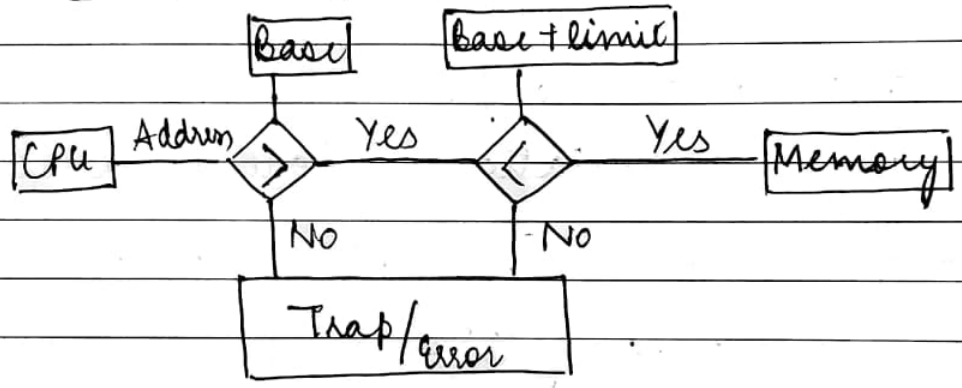
1. Base and Limit Registers

- used so that each process has a separate mem. space
- used to define range of legal add. that a process may access.



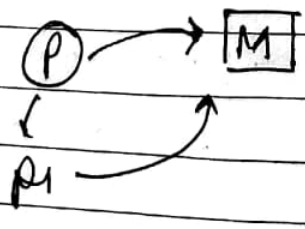
- holds the smallest legal physical memory address
- specifies size of the range.

Hardware Protection



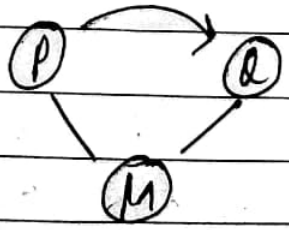
Dynamic Loading

Complete prog. is loaded into memory, but sometimes a certain part of program is loaded only when it is called by prog.



Dynamic Linking

Initially only P gets loaded, and CPU links the dependent prog. (Q) to main executing prog. (P) when it is needed.



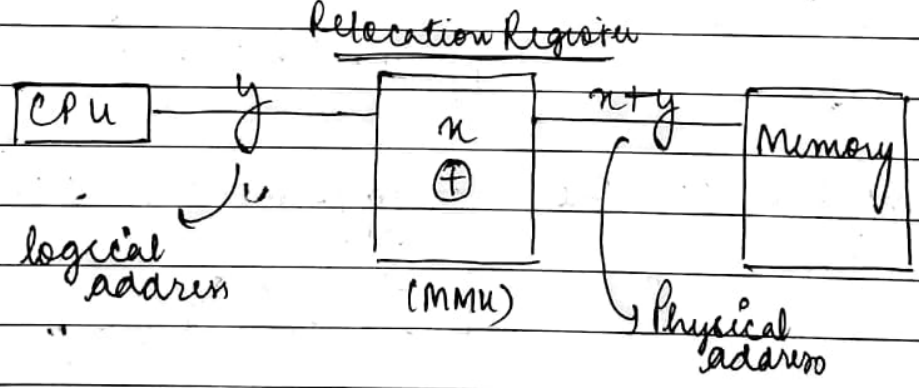
Logical Address Space	Physical Address Space
Address generated by the CPU. (Virtual add.)	Address seen by the memory unit.

In compile time and load time memory address binding (LAS) and (PAS) are same.

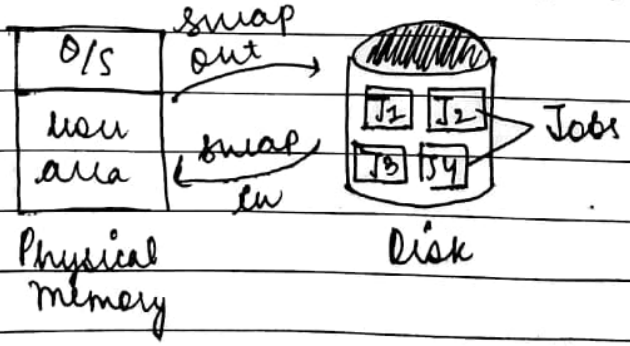
They are different in execution time binding.

Memory Management Unit

- Used to do run time mapping from logical to physical address space.
- Can be done with Relocation register.



Swapping : process of swapping in and out the jobs from disk to physical memory & vice versa



When J1 is completed, then swap out J2 and swap in next scheduled job.

Roll out and Roll in

Date...../...../.....

→ Priority based Scheduling algo.

If a higher priority process (J4 say) arrives, then a lower priority process will be rolled out and memory is allocated (J3) to J4.

Ques. → It does not wait for completion of J3.

Memory Management Techniques

Contiguous	Non Contiguous
- Centralized	- Decentralized
- Prog/Process as a unit is stored. (ek saath hoti hai stored)	- Prog as parts is distributed (ek saath nhi hoti fatke hoti hai stored)
- Implemented with the help of Partitioning	- Implemented with → Paging → Segmentation → Virtual Memory

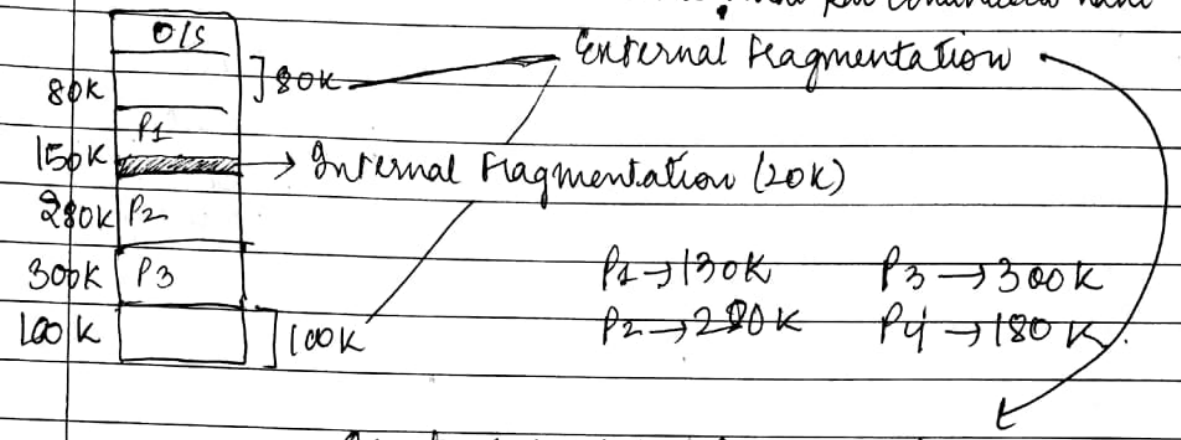
Fragmentation - Memory loss

External and Internal Fragmentation

External Fragmentation - Area when unable to accomodate a process of size even we have sufficient size of memory but it is not contiguous.

MVT - Multiprogramming with a variable no. of Tasks

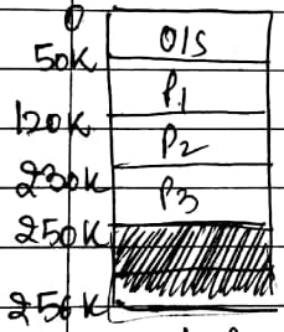
Internal Fragmentation - Arises when a Date...../...../..... process of size smaller than allocated partition is placed in that position, that leaves behind small amount of unused memory.



Can be solved by Compaction
[Placing all free memory partitions together]

✗ No Internal Fragmentation in MVT → External Fragmentation

↳ no pre-decided partitions

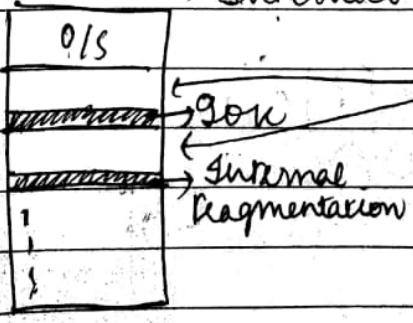


P1 = 70K
P2 = 110K
P3 = 20K

↳ But there can be case of external fragmentation

✓ MFT [Multiprogramming with a fixed no. of Tasks]

↳ Internal Fragmentation



P1 = 70K
P2 = 110K

There can be Internal Fragmentation

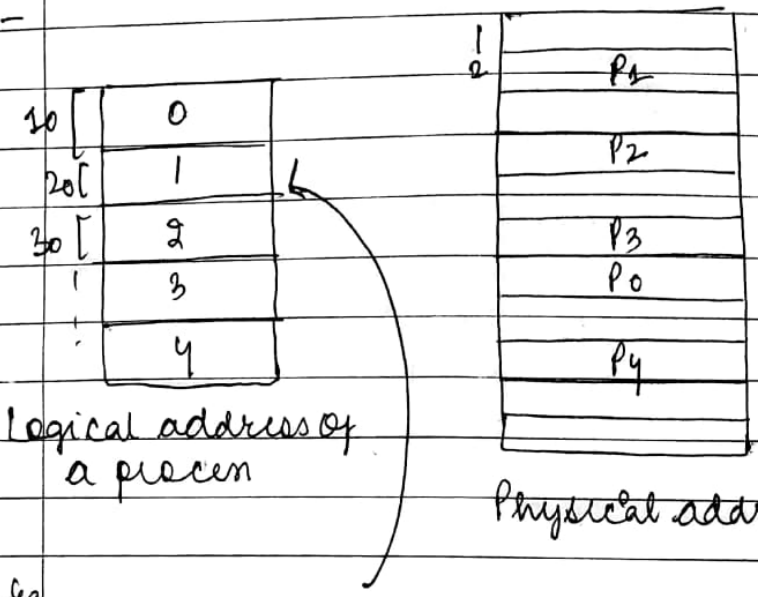
Secondary memory is divided into equal size partitions and these partition are called pages. and main memory - frame

Paging

Size of page/frame is same in which they are divided

- Non Contiguous Memory allocation

-



Logical address of a process

Physical address

Page size = P
L = Logical address
L = (p, d)
p = page no.
d = offset within that page

for eg

for 14, page no. = 1, offset 4
⇒ (1,4) ⟶ (p,d)
↓
(24)

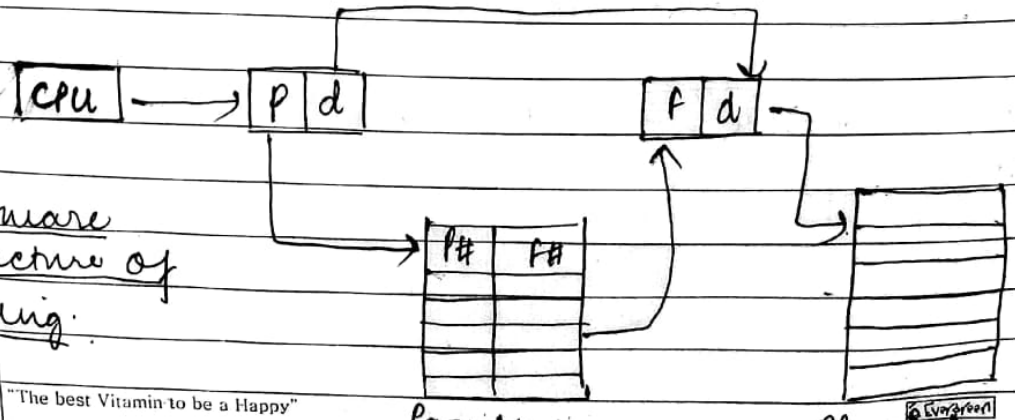
d = displacement

CPU generates logic address having two parts p, d
p = pg no and d = offset

p = L (div) P
d = L (mod) P

P = 10, for 14 → p = 14 div 10 = 1, d = 14 mod 10 = 4] (1,4)

Physical address
↳ (f, 1) * P + d

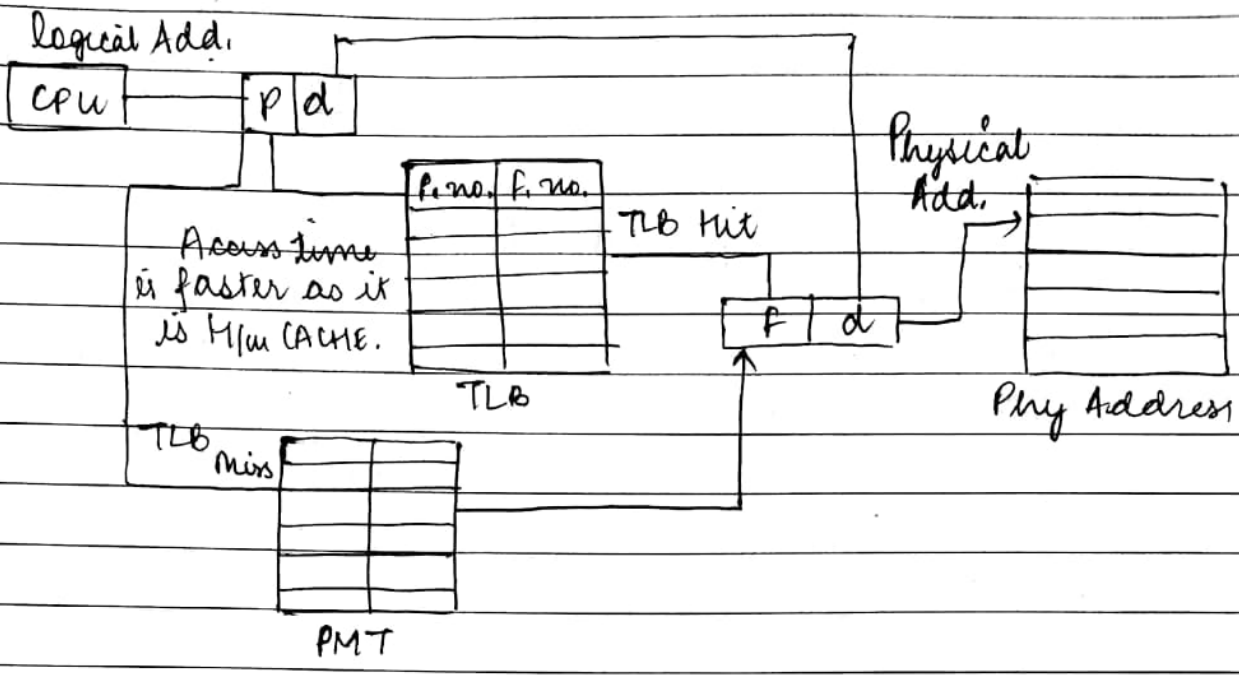


Hardware Architecture of Paging.

"The best Vitamin to be a Happy"

Paging with TLB [Translation Lookaside Buffer]

↳ Contains few Page Table Entries ↳ Hardware Cache mem



Valid-Invalid bit can also be associated with page.

	F. No	V.I. bit
0	2	V
1	3	V
2	4	I
3	1	V
4	5	I
5	6	V
6	8	V

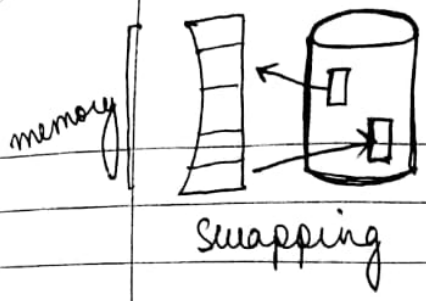
bit → Valid: Process is in the Process logical Address Space

bit → Invalid: Page is not in the Process logical Address Space.

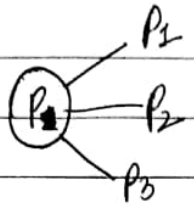
Virtual Memory

- Separation of user logical memory from physical memory.

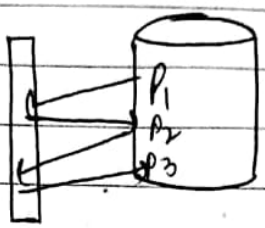
↳ In this, we keep only a part of the process in the memory and other part on the disk. (secondary storage).



Swapping



Date...../...../.....



Advantages:

1. Only part of the prog needs to be in memory for execution.
2. Logical Address Space is much larger than physical address space.
3. Need to allow pages to be swapped in & out.

User's pt. of view

Programmers or developers are relieved of trying to fit a program into limited memory.

System pt. of view

Reduces external fragmentation.

Demand Paging – In this technique, a page is bought into the memory for its execution only when it is demanded.

Advantages of Demand Paging
- Reducing memory req.
- Swap time is also reduced
- ↑ degree of multiprogramming

Disadvantages → [Page fault]

Page Replacement Algorithms

Date...../...../.....

- It is the technique used by OS to decide which memory pages to swap out.
- It is also decided that in memory, how many frames to allocate to each process.
- When page replacement is req., we must select the frames that are to be replaced.

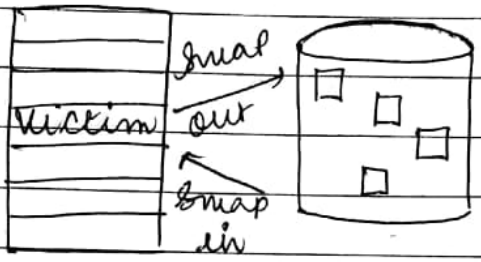
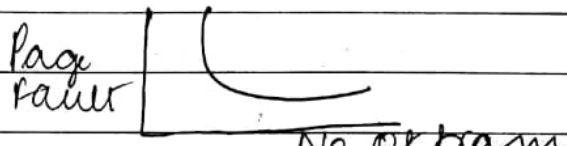


Diagram of Page Replacement



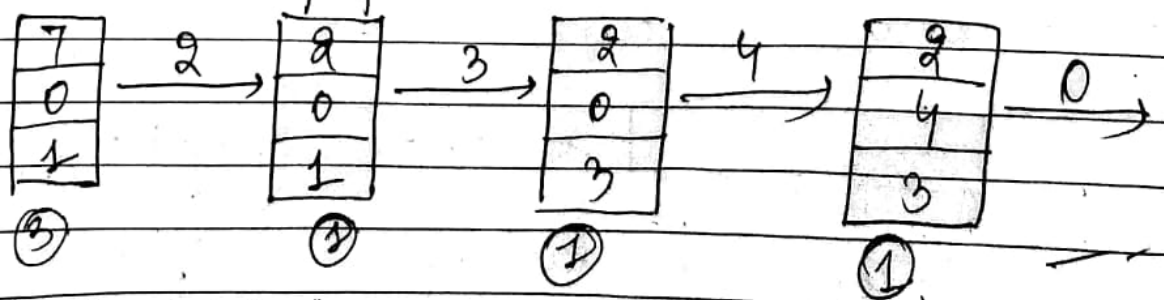
Page Fault - It is a type of interrupt, raised when a running process accesses a memory page, that is mapped into virtual memory but not loaded in main memory.

1. FIFO Page Replacement
2. Optimal Page Replacement -
   ① Replace the page that will not be used for the longest period of time.

Example: calc. the no. of page faults for the foll. reference string

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

Assume no. of frames = 3



"The best Vitamin to be a Happy"

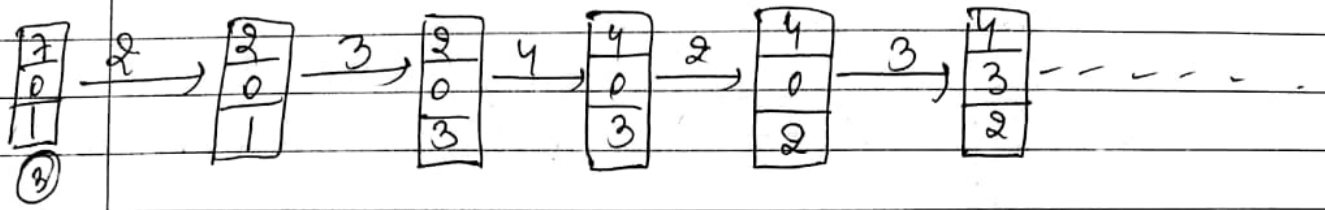
Evergreen

3. Least Recently Used Algorithm Date...../...../.....

(7) Page which has not been used for the longest time in main memory the one which will be selected for replacement.

Example - calc. no of pg faults for foll.

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



* Address binding can take place in 3 diff. ways.
1. Compile time
2. Load time
3. Execution time

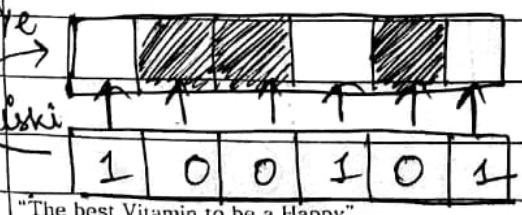
Logical address space ↓ generated by CPU

Physical add space ↓ loc^n that exists in the memory unit.

The set of all logical addresses generated by the CPU for a program is called.
Logical Address Space.

Memory management with bit maps

- Reqd when memory is allocated dynamically



bit value ie 1 or 0

If the bit is zero, memory unit is free and if bit is 1, then memory unit is occupied

"The best Vitamin to be a Happy"

- Memory is divided into allocation units called bits. Date...../...../.....
- Each allocation unit is mapped in a map called bit map.

Main issue: - choosing the size of allocation unit
- Smaller size of allocation unit means a larger bit map.
- Larger the allocation unit, chances are there that the memory unit may not be used completely.
- A memory size of 256 bytes, allocation size of 4 bytes requires 64 searches.

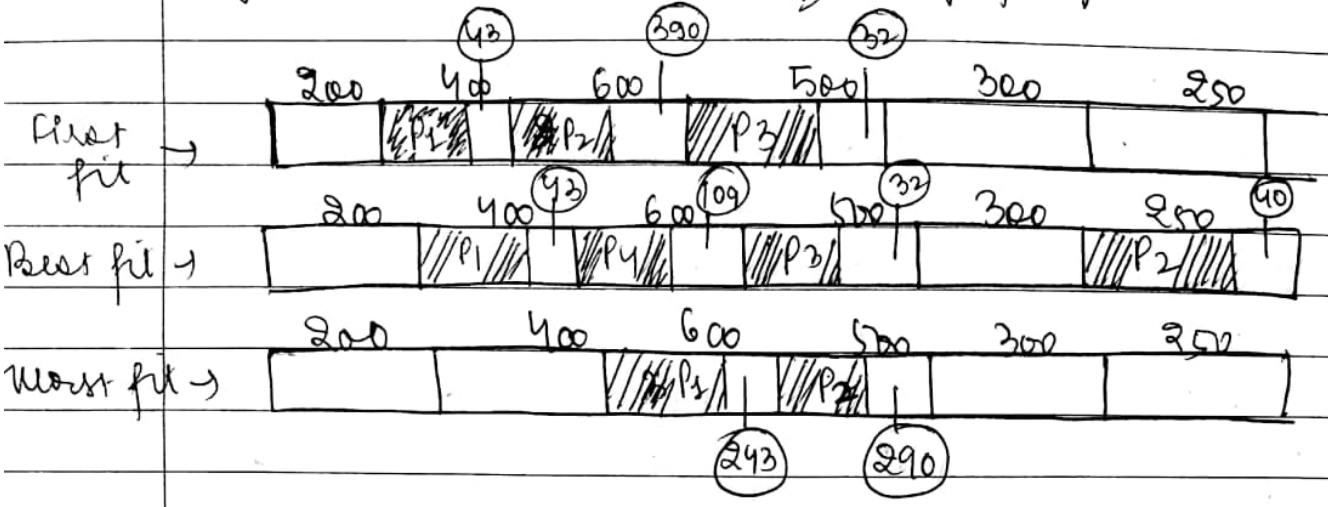
~~Ans~~ Memory management: Linked list

- maintain a linked list of allocated & free memory segments.
  ↓
  process / hole b/w two processes.
- Each entry holds the following data
  - Port: for Process or Hole
  - Starting segment add
  - Size
  - Ptr. to the next
- This info is useful for searching the allotted and free units.
- Also once memory is released, it is easy to merge requires searching.

possible search algorithms are -
- First fit
- Best fit
- Worst fit
- Quick fit

$P_1 = 357$, $P_2 = 210$, $P_3 = 468$, $P_4 = 491$ Date...../...../.....

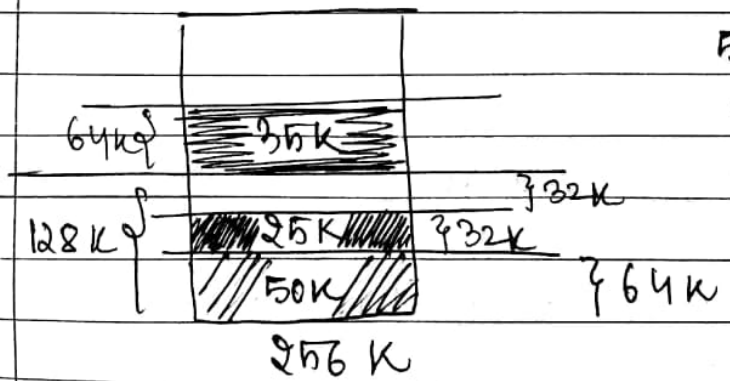
First fit - first available and suitable fragment will be used
Best fit - segment with size closest will be filled.
Worst fit - maximum possible size of fragment to be filled.



Mem Mgmt. in Buddy System

Whenever a request by a process arrives, memory is divided into two half blocks. Nearest and most suitable memory size partition is allocated.

Q In a buddy sys. mem system 50K, 25K and 35K are satisfied with 256K. How many blocks are left?

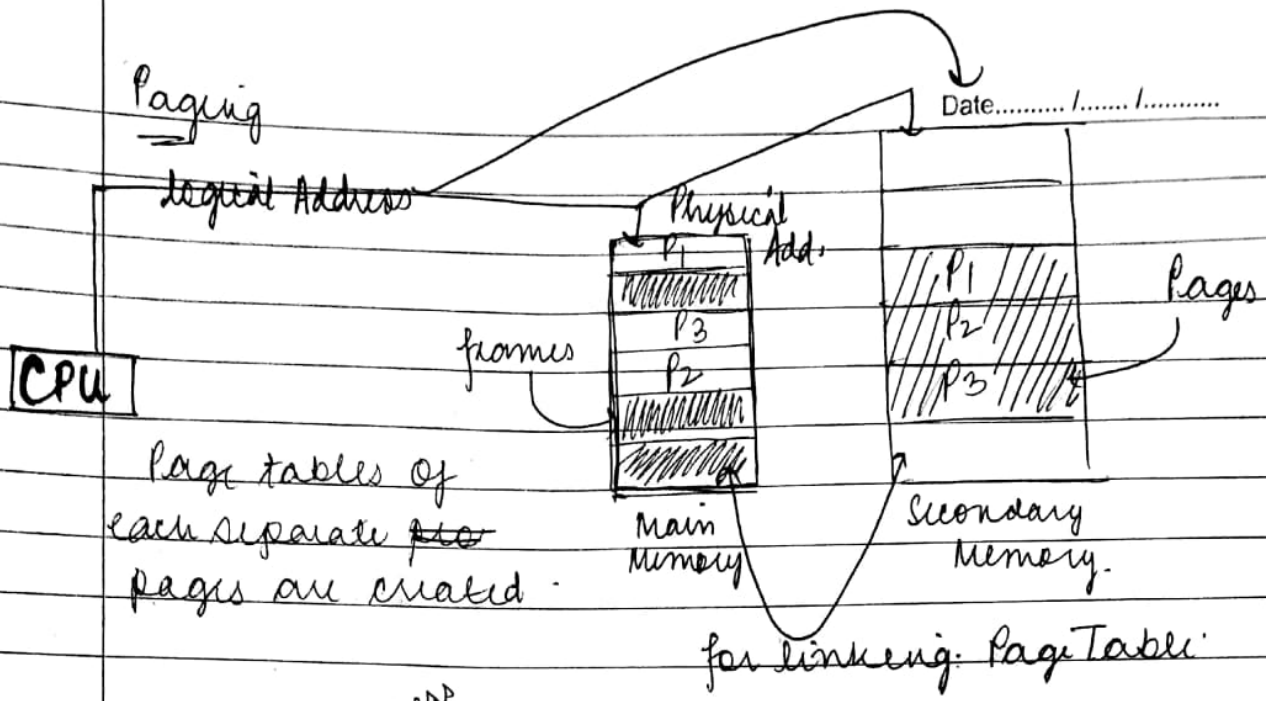


50K, 25K, 35K
↓ ↓ ↓
P_1 P_2 P_3

35K > 32K

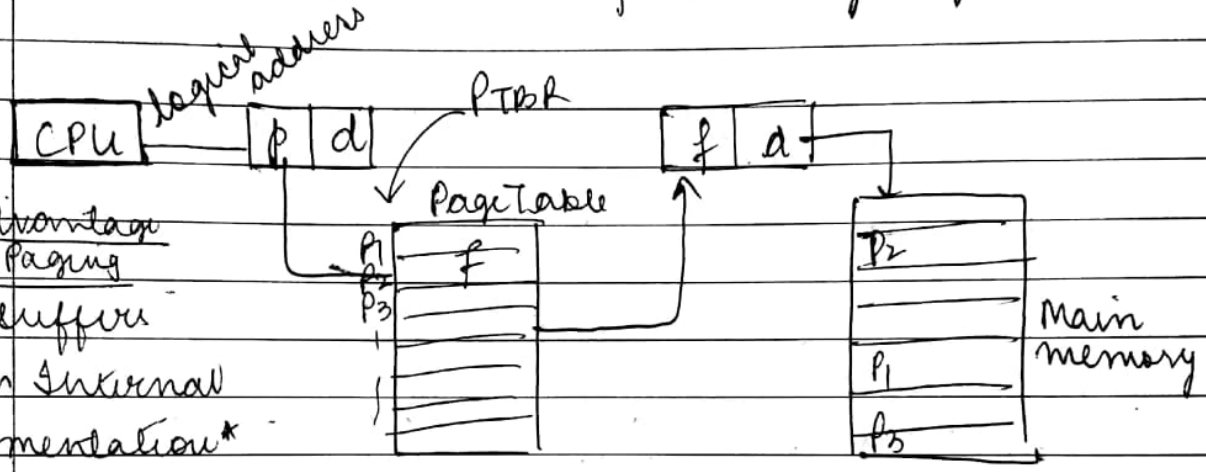
Date...../...../.....

Paging



Page tables of each separate ~~pro~~ pages are created.

for linking: Page Table



Disadvantage of Paging

* It suffers from Internal fragmentation *

Associative Memory

→ memory unit accessed by contents, ~~is called~~ rather than add. also called content Addressable Memory (CAM)

- used in high speed searching app^n's.

Read/Write Op^n in CAM

Write Op^n

→ When a word is written in ass. memory, no add is given.

= memory is capable of finding an unused loc^n to store the word.

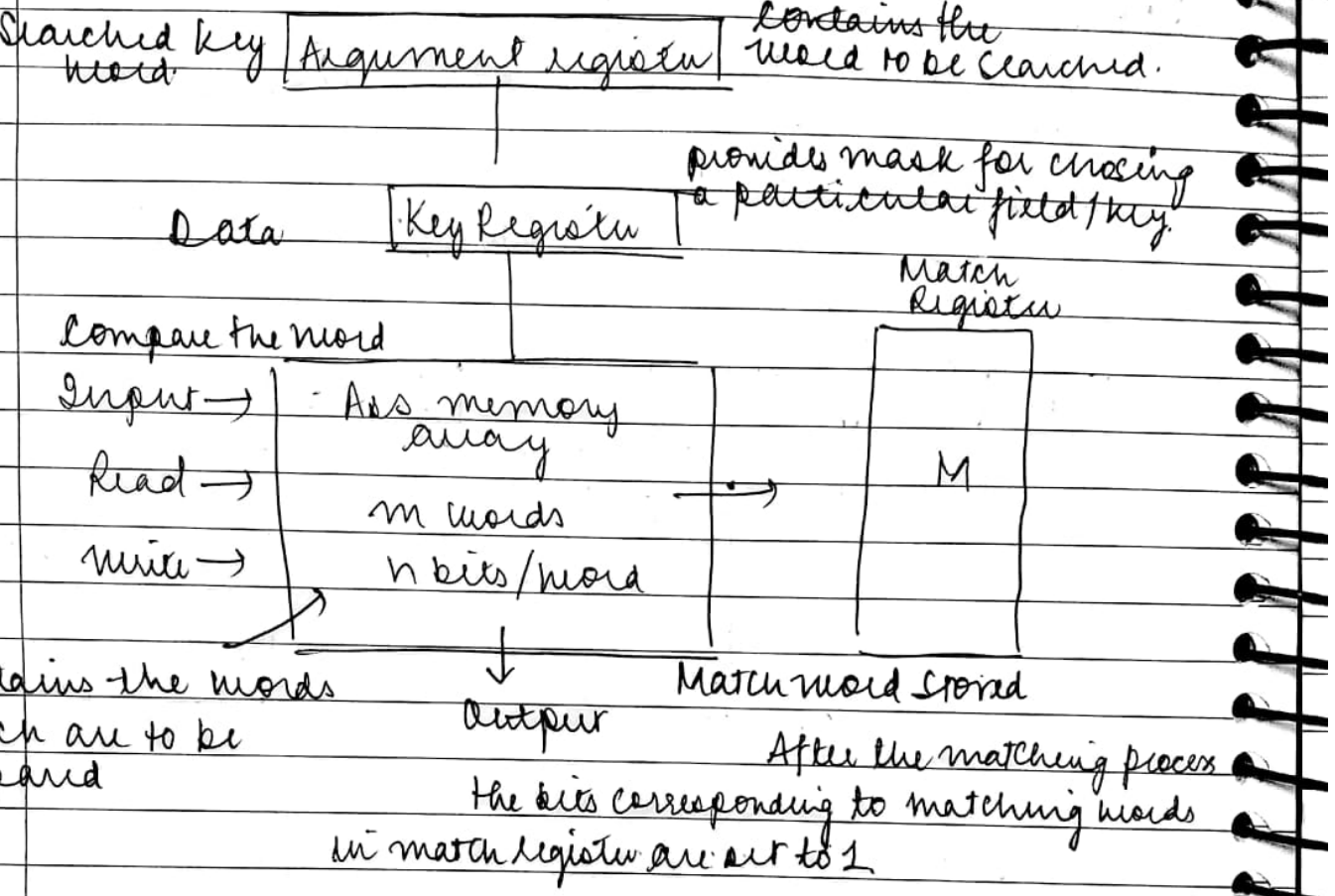
"The best Vitamin to be a Happy"

Evergreen

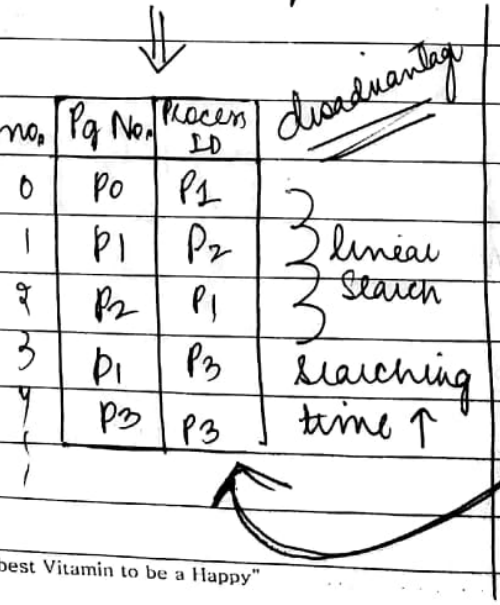
Read operation

Date...../...../.....

→ content is specified
→ memory locates all the words which match and marks them for reading.

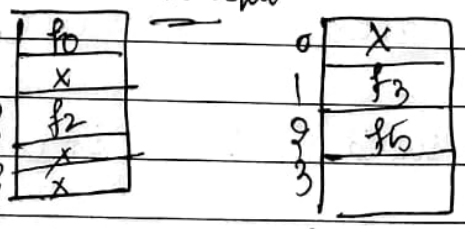


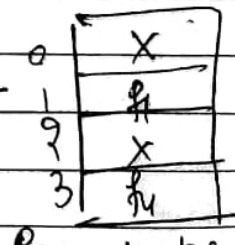
Inverted Page Table



adv: less memory space used.

Normal Paging Technique





"The best Vitamin to be a Happy"

E Evergreen

Segmentation

Non Contiguous Storage

Date...../...../.....

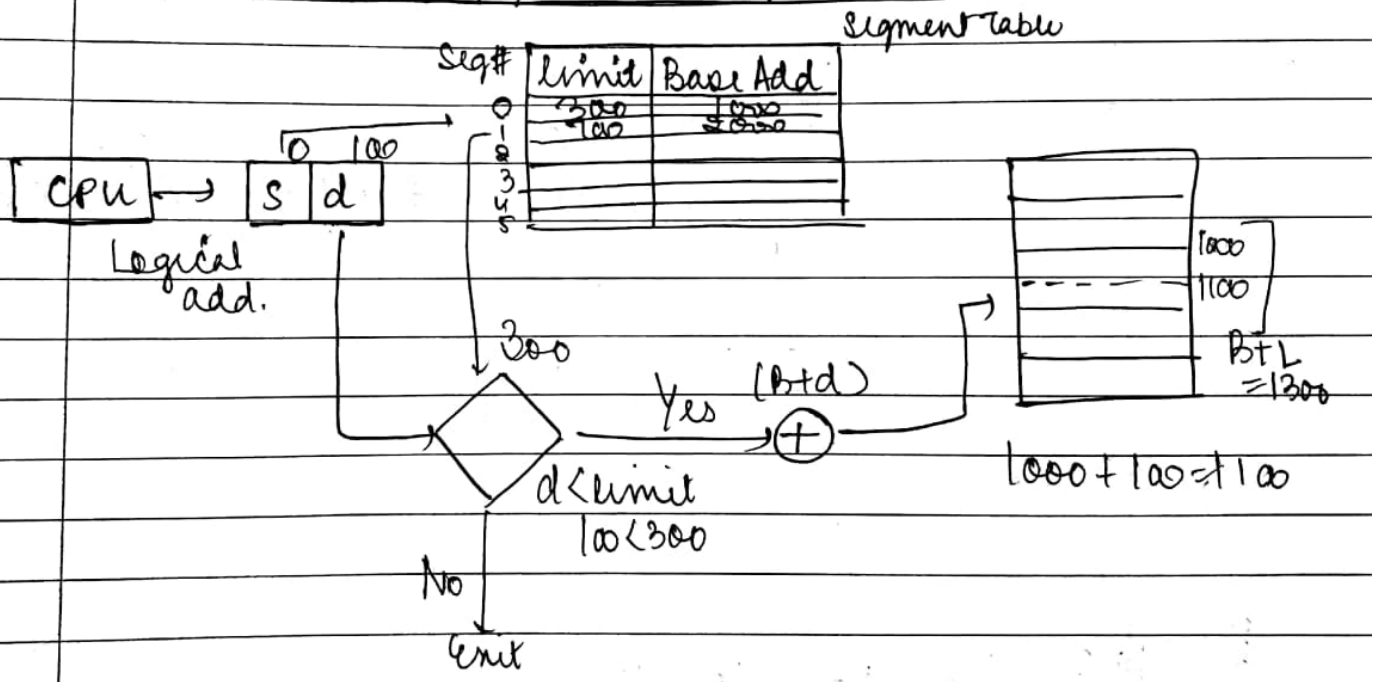
allocation → Paging (fixed size, Physical memory)

→ Segmentation (Variable size, logical mem).

Each segment has a Name and a length which is loaded into physical memory. → Segment no.

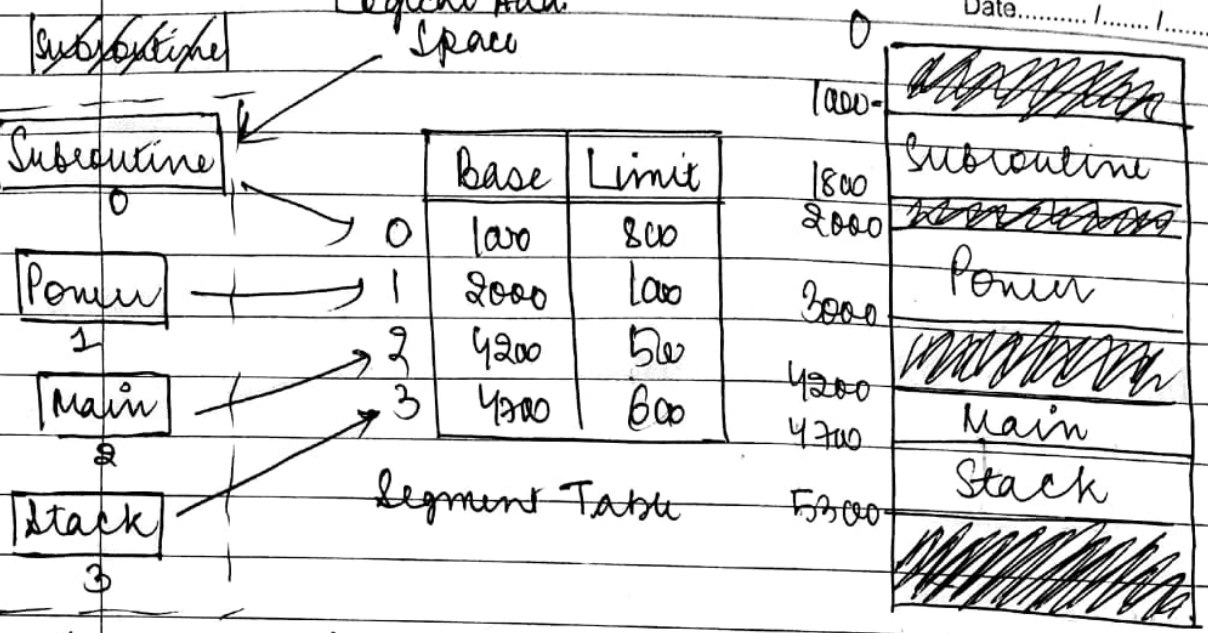
Segment no.	Offset → displacement

Hardware support in Segmentation



~~The value of offset~~ value of offset 'd' must be b/w 0 and segment limit. If not then move out & if it is legal then value of 'd' is added to the segment Base to produce the address in the physical memory.

Example



Consider the following table

Ques

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the phy add for the following logical add?

s	d

Log. add = B + d

(i) 0|430 → s=0, d=430 (430 < 600) ✓ ⇒ ⇒ 219 + 430 = (649)

(ii) 1|10 → s=1, d=10 (10 < 14) ✓ ⇒ 2300 + 10 = (2310)

(iii) 2|550 → s=2, d=550 (550 ≮ 100) d > leng not possible to map.

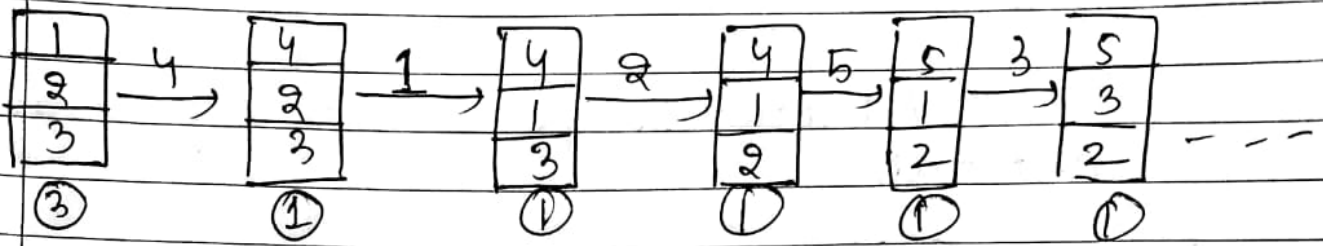
(iv) 3|400 → s=3, d=400 (400 < 580) ✓ ⇒ 1327 + 400 = (1727)

(v) 4|112 → s=4, d=112 (112 ≮ 96) not possible to map.

Page Replacement

1. FIFO. 1,2,3,4,1,2,5,1,2,3,4,5. Date...../...../.....

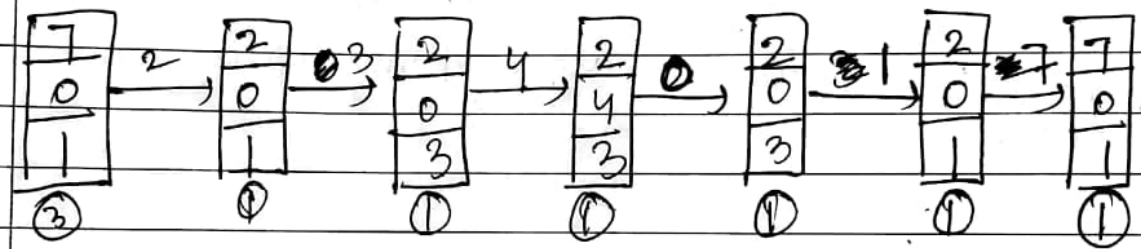
3 frames



2. Optimal Page Replacement (Best)

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

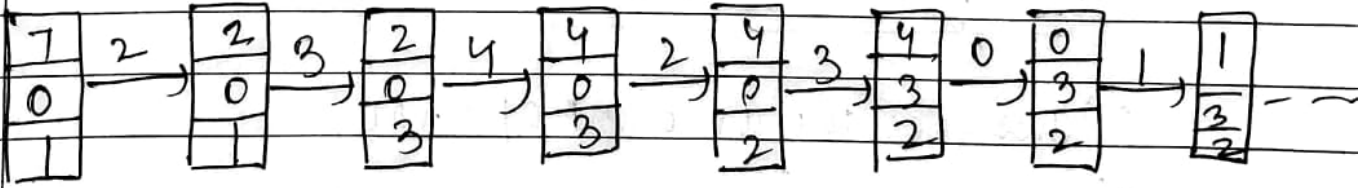
no. of frames = 3



~~Shuru~~ Uss no ke aage se chalna aur ~~tuno~~ teno mei se jo sabse last me hoga use replace karna.

3. Last Recently Used (Past)

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



Uss no ke pichi chalenge, jo ~~ta~~ teno mei se sabse last mei hoga usse replace karenge.

~~THRASHING~~

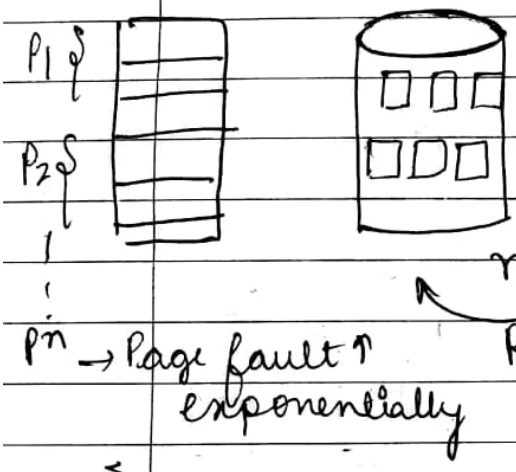
THRASHING

Date...../...../.....

situation if system spends more time in paging instead of their execution

Or

Thrashing = A process is busy swapping pages in and out.



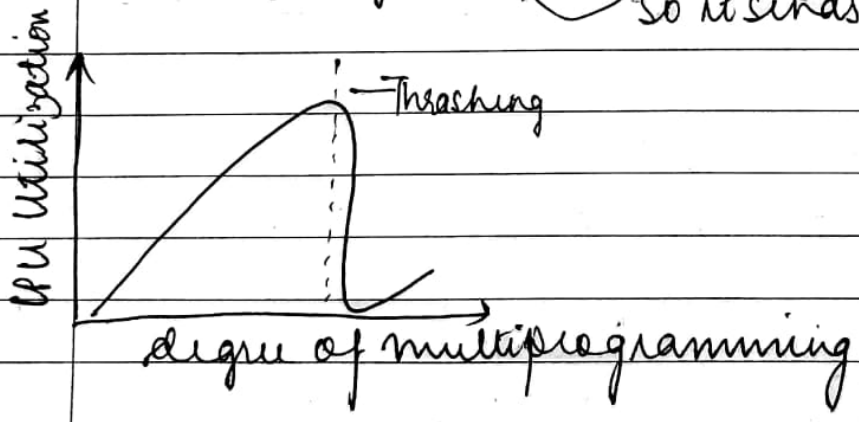
Process does not have enough pages for execution then page fault will occur.

↓ low CPU utilization

↓ OS needs to think the degree of multiprogramming should ↑.

So it sends another process

Pn → Page fault ↑ exponentially



Techniques to avoid Thrashing

→ Increase the amount of RAM.

→ Decrease no. of programs being run on comp.

MODULE 5 - FILE SYSTEMS AND DISK SCHEDULING

★ FILE CONCEPT

- Named collⁿ of related info that is recorded on secondary storage
- Represents programs and data
- It has a certain defined structure
- Files are mapped by the OS onto physical devices.
- It is a sequence of bits, bytes, lines or records.

★ File attributes

- Name
- Identifier
- Type
- Location
- Size
- Protection
- Time, Date and user identification

★ File Opⁿs

- Creating a file
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file

★ Several pieces of info. are associated with an open file

→ File pointer
→ File open count
→ Disk locⁿ of the file
→ Access Rights

"The best Vitamin to be a Happy"

Evergreen

Access Methods

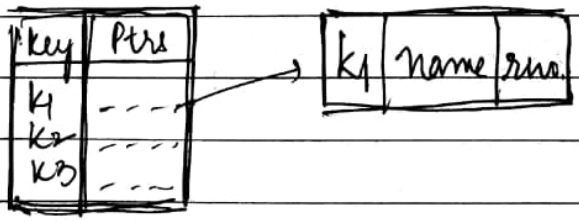
Date...../...../.....

→ Sequential Access
→ Direct Access
→ Other Access Methods

1. Sequential Access: One record processed after another. Supports following operations:—
   ① Read next ② Write next
   ③ Rewind ④ Skip n records

2. Direct Access: allows random access. User can jump to any record and access that record. Following operations are supported.
   (i) read n : Reading record 'n'.
   (ii) write n : Writing record 'n'.
   (iii) jump to record n → jump to 10
   (iv) Query current record → used to return back to this record later.

3. Indexed Access: Index is created which contains a key field & pointers to the various blocks.



FILE ALLOCATION METHOD

Allocating space to files so that disk space is utilized in an efficient manner.

Date...../...../.....

Factors to consider :-

1. Processing Speed → Sequential
   → Random

2. After allocation, that file should have ability to use multisector and multi track transfer.

3. Disk space utilization
   ↳ max files can be stored / min wastage

4. Main memory requirement — least

Allocation Method

1. Contiguous Allocation

→ Linear Ordering
→ Locⁿ of file defined by – disk add of first block and its length.
→ Both sequential and direct access supported

Directory

File	Start	length
A	0	2
B	3	3

Disadvantages

↳ finding space for New file
↳ Internal fragmentation

Continuous Allocation Appⁿ: Dynamic Storage Allocation

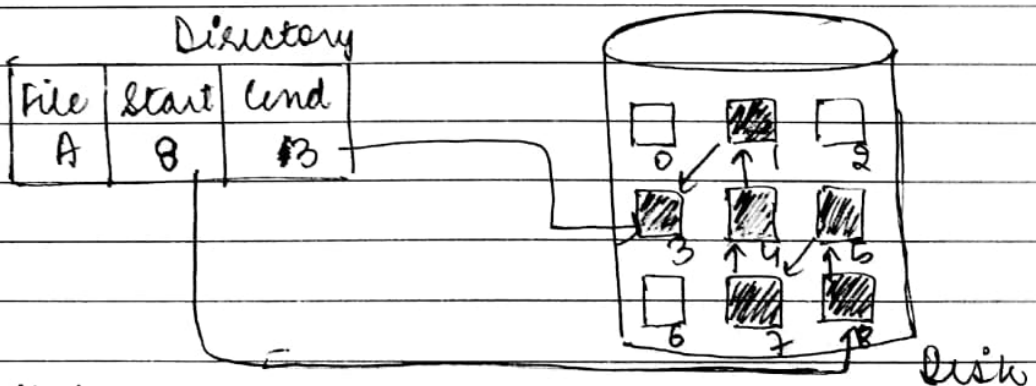
1) First fit: 290K → 400K (110K wasted).
300K

2) Best fit: 300K
400K
290K → 298K

EXTERNAL FRAGMENTATION.

3) Worst fit: 300K
290K → 400K (110K wasted)
298K

2. Linked Allocation : Solves all problems of contiguous allocⁿ. Each file is a linked list of disk blocks.
→ No external fragmentation
Disadvantage → can be used only for sequential-access file
No direct access

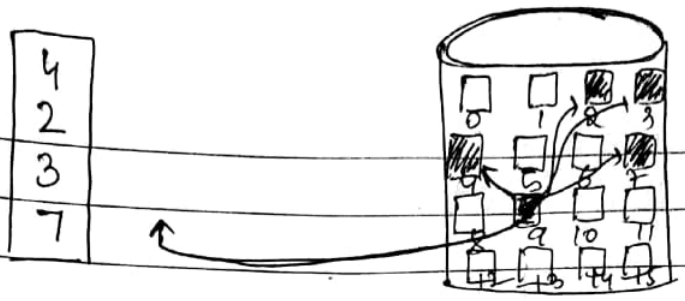


If it wants to acc

If it wants to access file 4, then it has to first visit all the nodes behind it in the linked list.

3. Indexed Allocation - All ptrs. are brought together into one locⁿ called Index Block. Each file has its own index block.

File	Index block
A	9



Date...../...../.....

File System Implementation

→ resides permanently on secondary storage
→ it must be able to hold a large amount of data permanently.

Issues associated:-
- Disk space allocⁿ
- Free space recovery
- Tracking the locⁿ of data
- Performance

FILE DIRECTORIES

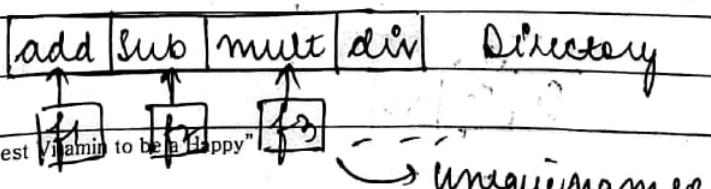
A physical disk can be broken up into multiple partitions or mini disks.

A. Directory | files

B. Directory | files

Opⁿ on directory-
1. Search for a file
2. Create new file
3. Delete a file
4. List a directory
5. Rename a file
6. Traverse file system

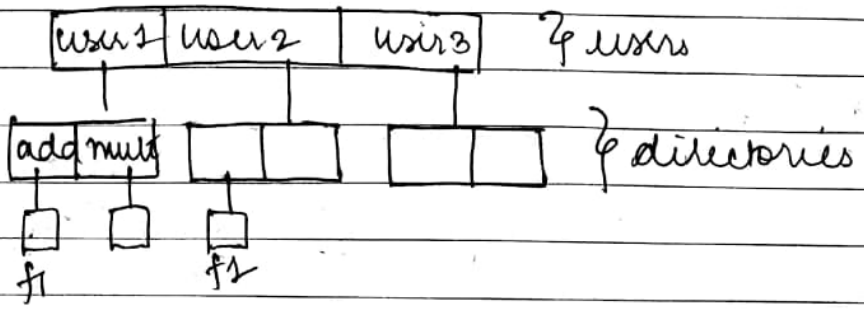
① Single level directory
- all files in same directory
- each file must have unique name.



Disadvantages - all files should have unique names

2) One level Directory - creates a directory for each user

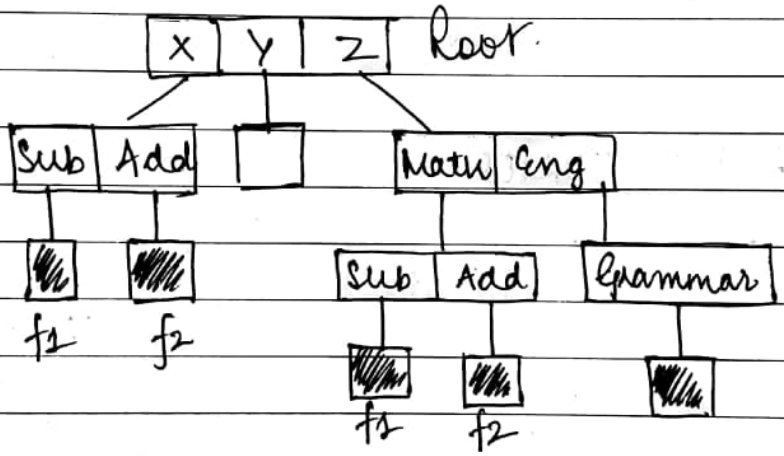
Date...../...../.....



Inside the directory unique names should be there

3) Tree-Structural Directories

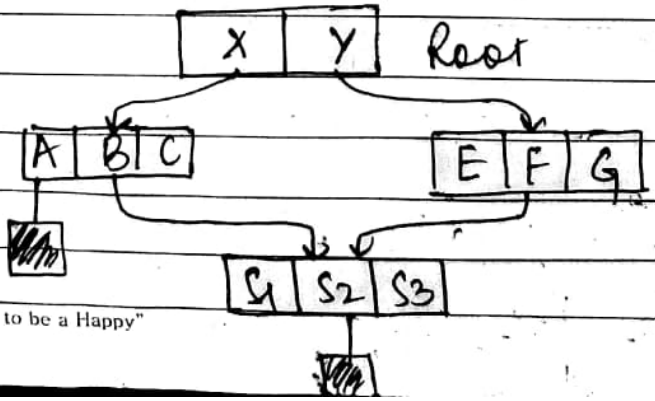
Allows users to create their own sub directories and organize their files accordingly.



4) Acyclic Graph Directories

Useful when the same files need to be accessed in more than one place in the directory structure.

files are shared by more than one user.



Disk Scheduling Algorithm

Date...../...../.....

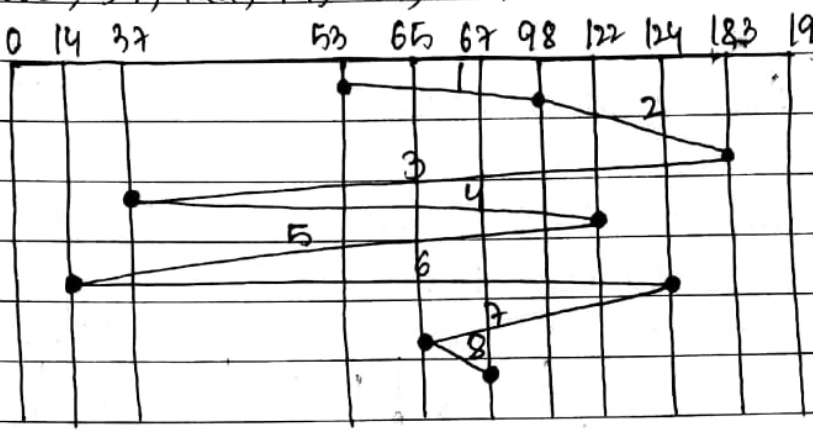
1. First come First Serve Algorithm

(0-199)

Calculate total head movements for the foll. I/o reqts:

98, 183, 37, 122, 14, 124, 65, 67

Head Starts at 53



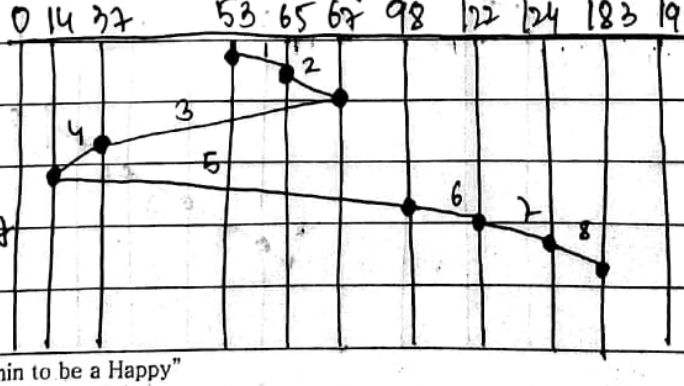
Total head movements = head movements for (1+2+...+8)

1. 98-53 = 45
2. 183-98 = 85
3. 183-37 = 146
4. 122-37 = 85
5. 122-14 = 108
6. 124-14 = 110
7. 124-65 = 59
8. 67-65 = 2

T.H.M = (640)

2. Shortest Seek Time First (SSTF) Algorithm

- Selects the requests with the min^m seek time from the current head position.
- Better than FCFS (less no. of head movements)
- May cause starvation of some requests.



When on 53,

~~67~~ 65-53 < 53-37

Total no. of head movements = (236)

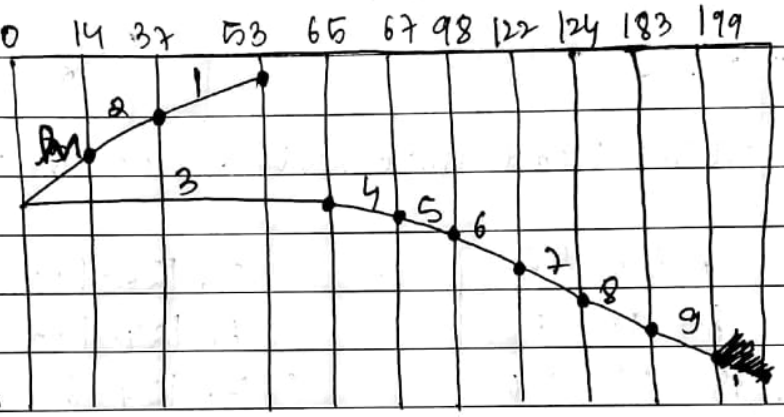
Date...../...../.....

3. SCAN Algorithm

- also known as Elevator Algorithm
- Disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, direction of head movement is reversed and servicing continues.

example- 98, 183, 37, 122, 14, 124, 65, 67

Current Head Posⁿ = 53

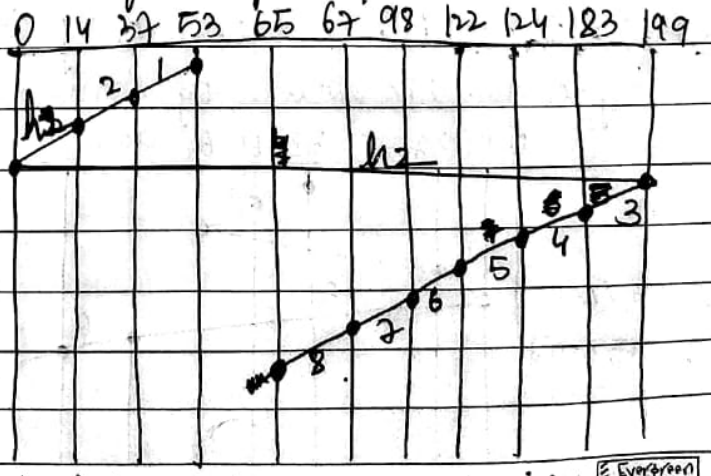


Considering movement towards left.

Total head movements = (236) cylinders

4. C SCAN Algorithm

When the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.



Example: 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53.

Movement towards left.

TNM = (382)

Provides more uniform wait time than SCAN Algorithm

Variation of SCAN & C-SCAN Algorithms

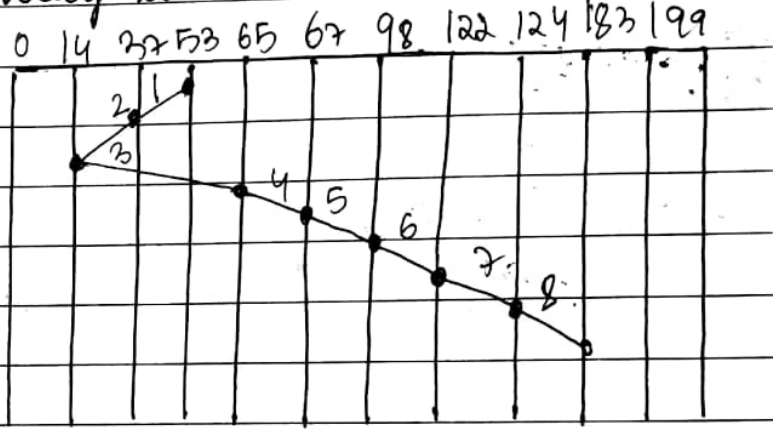
Date...../...../.....

5. LOOK Scheduling

- Same as SCAN, but the arm goes only as far as the final request in each dirⁿ and then reverses dirⁿ without going all the way to end.

Ex: 98, 183, 37, 122, 14, 124, 65, 67

Current head posⁿ = 53



6. C-LOOK Scheduling

- Same as CSCAN but...