if one of the ~~co~~ circuit present deadlock will not occur.

Mutual exclusion:

→ NO two processes use the same single resource simultaneously

i.e only one process access a resource at a time

→ Mutual should be there for deadlock

Hold and wait:

process are holding atleast one resource and waiting for the occurrence of the additional resource

Circular wait

→ There exist closed chain of process such that each process holds atleast one resource needed by the next process in the chain

(or)

There exist a set of ~~{P0, P1}~~ $\{P_0, P_1, P_2 \cdots P_{n-1}\}$ of waiting process such that P_0 is waiting for resource that is held by P_1, P_1 is waiting for resource / P_{n-1} is waiting for Resource that is held by P_0

No preemption:

→ process can release only voluntary of the resources held by the process

→ no resource can be removed from the process by holding it

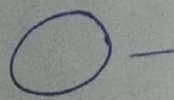
Resource Allocation Graph (RAG)

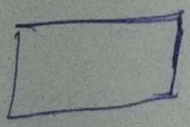
→ RAG denotes the ~~state~~ current state of the system

State: → how many process available
→ how many resources available
→ how resources held by process

$G = (V, E)$

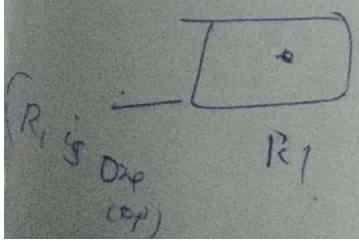
Here, vertices are represents the process & Resources

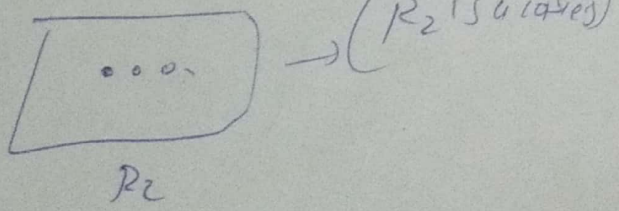




Single instance of Resource

multiple instance of Resource

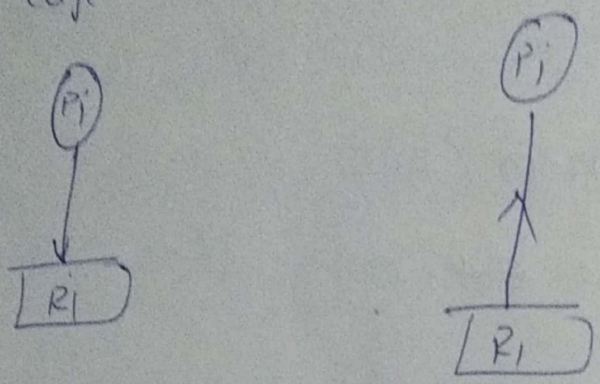




→ Edges

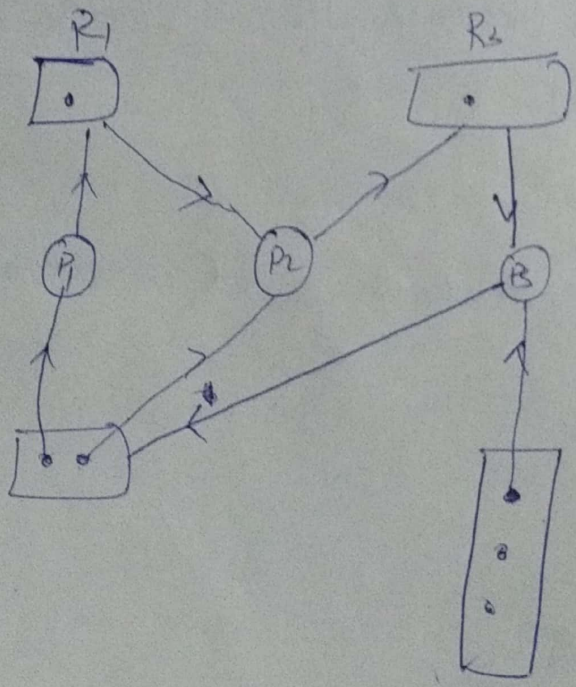
Request edge

Assignment edge



ex

1)

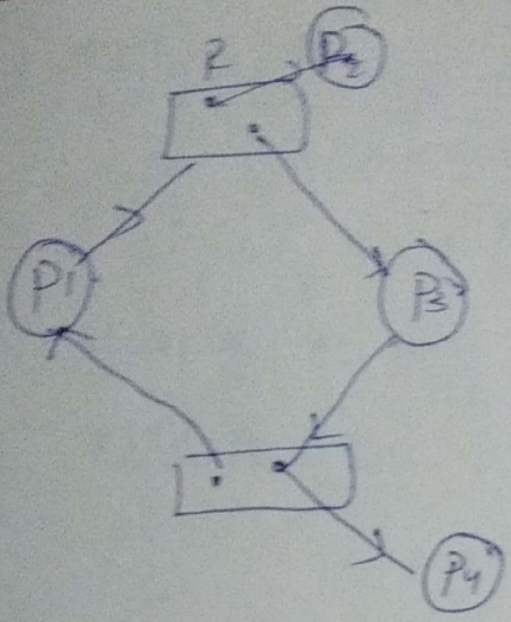


deadlock

P1 – P2 – P3 – P1 → 1) Circular wait
2) hold wait

∴ it is a ~~head~~ deadlock

ex:



no deadlo

P1-P3-P1 → circular wait

→ ~~but~~ P0 P1 and P3 are in hold and wait conditon

~~but~~ P4, P2 are not in hold and wait. so

after some time ~~it~~ P2, P4 will Released

observation

1) if RAG containing No cycles then ~~do~~ "no deadlock"

2) if RAG contains cycles :-

a) if there is only one instance fore resource type then there is a deadlock (Neccessary and sufficient conditions to feel deadlock)

b) if there are several instances for resource type then there is a possibility of deadlock (it is neccessary but not sufficient conditions).

possibility → may (or) may not

Methods for handling of deadlock :-

(1) Deadlock prevention :-

→ Disathsfying the one of neccessary and sufficient conditions of deadlock

Mutalexclusion:
It cannot be ~~death~~ dissathsfied since every resource acces must be the mutuallly exclusive

Hold & wait:
Disathsfying the hold and wait means either hold the resources (or) wait for the resoures ~~but not~~ both at a time

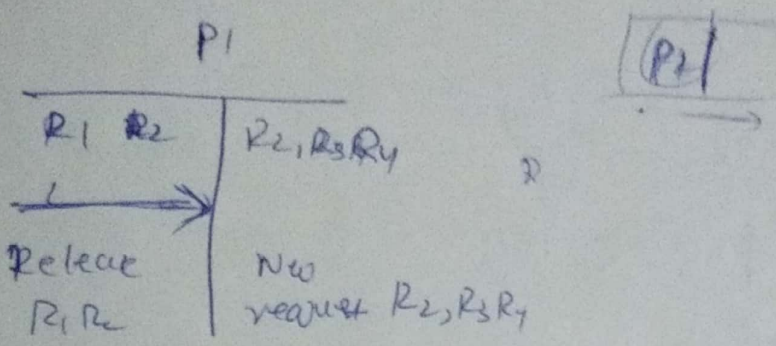
condition to disathsfey hold and wait

1) * Take all reqiured resources before beginning of execution
   * so that while executing ~~it can~~ process cannot request resources

limitation for condition)

* Resources not properly utilized and process get starvated

2) Can make new request for the resource by releasing held resources



limitation

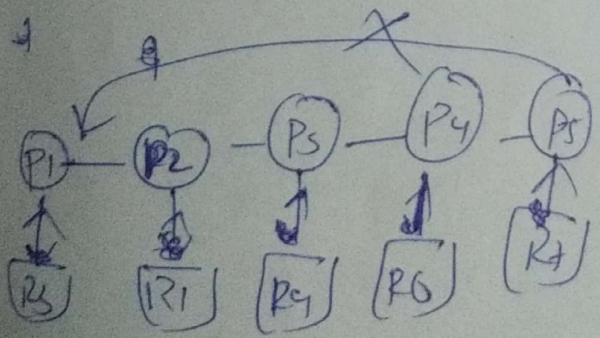
Current process will get starvated if R2, R3 R4 are not available are busy with other process

Circular wait:

→ Disatisfying the circular wait by breaking the cycle

* All resources in the system uniquely identified by the id or number

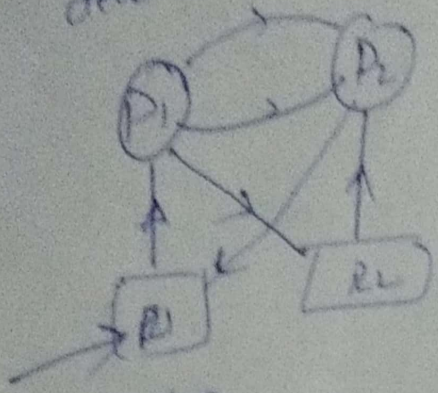
* never allow process to request to lower number resource than the last one requested and allocated



R1 = 10
R2 = 17
R3 = 5
R4 = 20
R5 = 30
R6 = 37
R7 = 40
R8 = 47

→ Don't allow process P5 to Request R3 because it lower number i.d

→ The process must be preempted to avoid deadlock



preemption

when you apply preemption R_1, R_1 will be allocated to P_2 and after some time P_2 will release R_2 and R_1 is turn can be used by P_1

NOTE :

→ "Deadlock prevention" more restrictive than "Deadlock avoidance"

because in deadlock prevention we are restricting

//

II. Deadlock avoidance :-

→ It is a simple to implement

→ It works on the state of the system.

→ It need a prior information about process request for resources ie. processes must declare their maximum demand of resources

System States :-

① safe state [no deadlock]

② unsafe state [leads to deadlock]

↓

It possibility to deadlock

Safe State :-

A state is a safe if the system can allocate resources to each (upto its maximum) in some order and avoid the deadlock

(or)

A state is safe only if there exist a safe sequence.

Safe Sequence :-

A sequence of processes $\langle P_1, P_2, P_3 \cdots P_n \rangle$ is a safe sequence for the current allocation state if for each P_i, the resource that P_i can still request can be

satisfied currently available resources +
The Resources held by all Pj (Remaining Pro

→ "Deadlock avoidance algorithm" uses "Resource allocation Graph" for single instances.

→ Deadlock avoidance uses "Bankers algorithm" for multiple instances

Bankers Algorithm

< P0, P1, P2 ... Pn >

m - Resources

< R0, R1, R2 ... Rm >

① Maximum [1 ... n, 1 ... m]$_{n \times m}$

max [i, j] = k

Pi → k [Rj]

Pi demands k instances of Resource Rj type

process Pi requires k copies of Rj type

Simply process Pi its peak demand of k copies of Resource Rj type

(2) Allocation $[1 \cdots n, 1 \cdots m]_{n \times m}$

Allocation $[i,j] \Leftrightarrow (P_i) \longleftrightarrow a[R_j]$

a copies (a) instances of R_j type is allocated to process P_i

$$a \leq k$$

(3) Need $[1 \cdots n, 1 \cdots m)_{n \times m}$

Need $[i,j]$

$(P_i)' \xrightarrow[\text{Need}]{b} b[R_j]$

→ process P_i remaining need ~~Resource R_j type~~ ~~be to~~ 'b' copies of Resource R_j type

$$b = k - a$$

(4) Request $[1 \cdots n, 1 \cdots m]$ ~~nxm~~ $n \times m$

Request $[i,j]$

$(P_i) \xrightarrow[\text{Request}]{} c[R_j]$

process P_i Request $= c$ copies of Resource R_j type at time t

$$c \leq b$$

Total [1 ... m]

Total[j] = d

There are d copies of Resource type j

Available [1 ... m]

Available [j] = e

There are e copies of Resource Rj type still available at time t.

Eg

Let us consider 5 process n = 5 (P1, P2, P3, P4, P5)

m = 1, R = 29 (Total

m = 1 (Resource type = 1)

R > 29 (29 copies)

available

① 3 - 1 = 2

② 1 + 3 = 4

③ 4 - 3 = 1

④ 1 + 5 = 6

⑤ 6 - 5 = 1

⑥ 1 + 10 = 11

⑦ 11 - 9 = 2

⑧ 2 + 17 = 19

⑨ 19 - 10 = 9

⑩ 9 + 20 = 29

R

pid	max Demand	Allocation	Need
P1	10	5	5 ✓
P2	5	2	3 ✓
P3	17	8	9
P4	3	1	2 ✓
P5	20	10	10
	55	26	

<P4 - P2 - P1 - P3 - P0>

Available = Total - Allocation

need = maxdemand - Allocation

shortcut:
available = available + allocation

Consider 5 process n=5, m=3 <A, B, C> <10, 5, 7>

Pid	maxdemand of each type resour A	B	C	current allocate A	B	C	nee A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1
				7	2	5			

↓

Available

	A	B	C
P1)	3	3	2
−	0	1	1
	3	2	1

+ 0

= 10 − 7 = 3 = A
5 − 2 = 3 = B
7 − 5 = 2 = C

P1 − P3

shortcut
- available
= available + allocate

A	B	C
3	3	2
+ 2	0	0 → Allocation of P1
5	3	2
+ 2	1	1 → Allocation of P3
7		

743
002 — allocation of P4

745
302 — allocation of P2

1047
010 — allocation of P0

1057 = R (in queue)

→ it we able to get the sequence it is not deadlock so it is ~~not~~ safe

Deadlock Detection and Recovery :-

* Allow the system to enter deadlock ~~lock~~ lock if it is enters use the detection algorithm to find processes and Resources involved in deadlock and then use recovery scheme to recover from the deadlock state.

→ deadlock detection with single instances of resources apply "Wait for graph"

→ for multiple instances use the Bankers algorithm.

→ 'Wait for graph' is constructed from the resource allocation graph by applying transitive rule

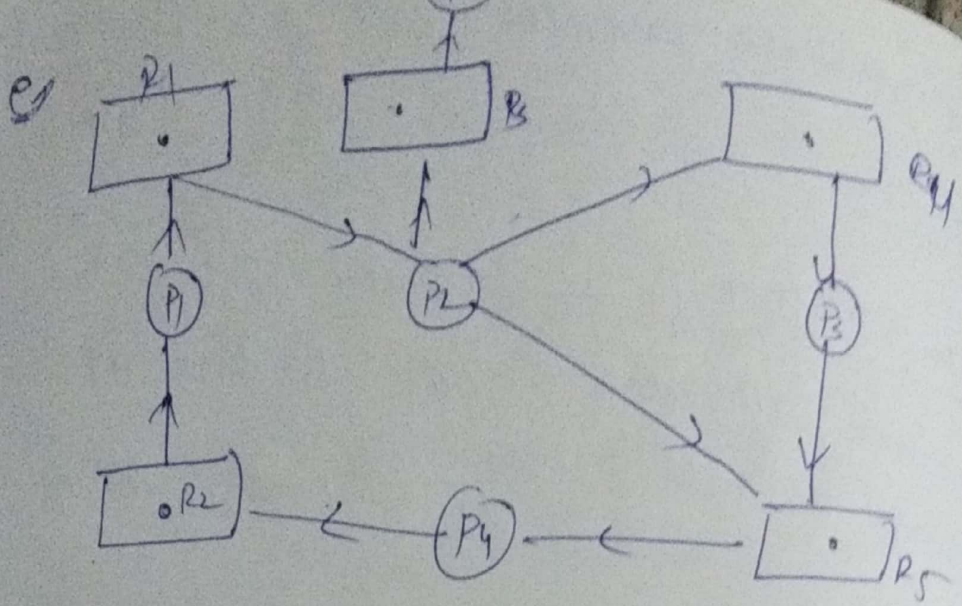
Transitive rule

If $a \to b$ & $b \to c = a \to c$

$a^R b$ & $b^R c = a^R c$

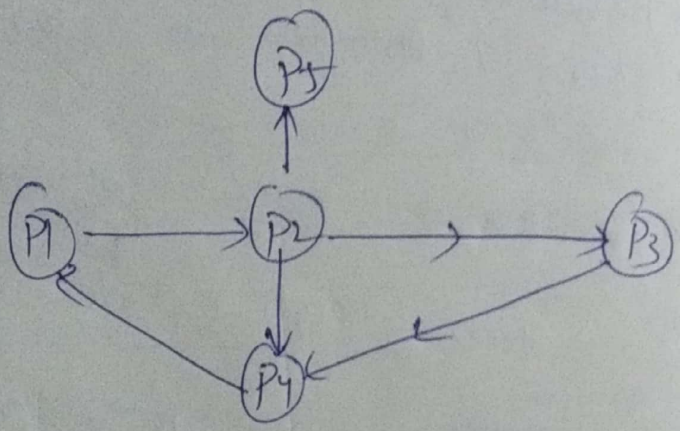
→ Wait for graph is a graph drawing the dependencies between the processes waiting for 1 another & 1 another

→ After drawing wait for graph take largest side and the processes which involved in the longest side those process only will be involving in Deadlock



RAG

Wait for graph



Now take largest size

P1 - P2 - P3 - P4 - P1

Now only P1 P2 P3 P4 are in deadlock and P5 is not in deadlock

for multiple instances

N=5 ; $(P_0, P_1 \cdots P_4)$, m=3 (A, B, C) = (7, 2, 6)

PID	Allocation A B C	Request A B C
P_0	0 1 0	0 0 0
P_1	2 0 0	2 0 2
P_2	3 0 3	0 0 0
P_3	2 1 1	1 0 0
P_4	0 0 2	0 0 2
	7 2 6	

available

<A B C>

0 0 0

3 0 3

3 1 3
+ 2 0 0

5 1 3
+ 2 1 1

7 2 4
+ 0 0 2

7 2 6

① $\langle P_2 - P_0 - P_1 - P_3 - P_4 \rangle$ Safe state

available = available + Allocation

SUPB

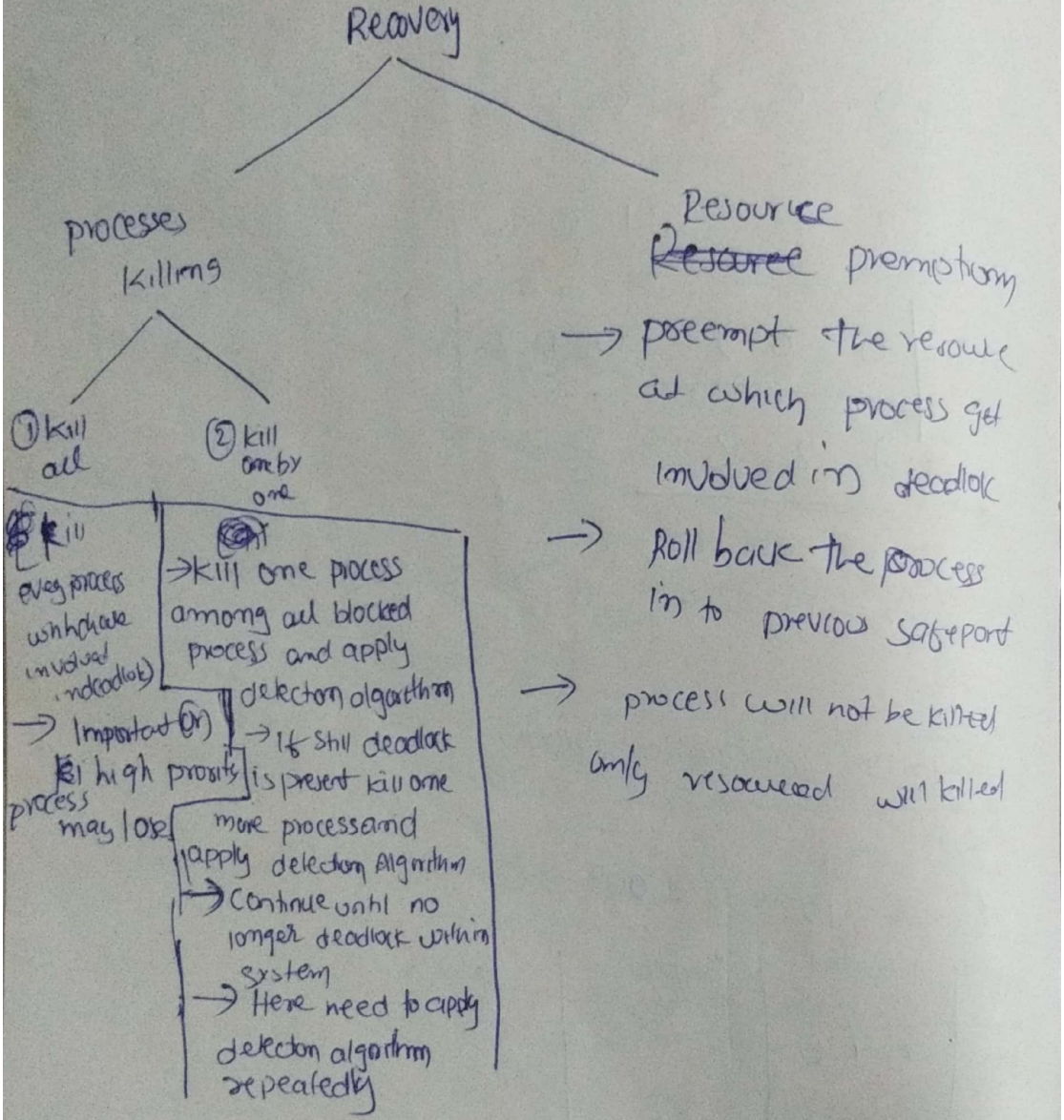
t3: $P_2 \rightarrow \langle 001 \rangle$ → Requested

Pid	Allocation A B C	Request A B C	available
P_0	0 1 0	0 0 0	(000)
P_1	2 0 0	2 0 2	x
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

→ P_1, P_2, P_3, P_4 are in deadlock

P_0 is not in deadlock

Recovery of deadlock:



~~Deadlock~~

Consider System with 3 processes each processes demanding maximum 3 tapes drives. what is the minimum no of resources 'R' that will not entering in to deadlock state.

Sl	Pid	max	Allocation
	P1	3	2 [max-1]
	P2	3	2 ["]
	P3	3	2 ["]

(6+1) → no deadlock

deadlock

→ when allocation vector is not given, allocate the Resources to the process with its [maxdemand −1] and now add 1 extra Resource so that system will not entering in to deadlock.

If you give

Allocation
1
2
2

Then allo the all in deadlock. so give max resource

so ~~max is~~

Allocation
2
2
2

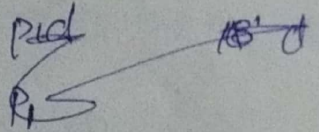
→ (T)

→ The minimum no of resources Required to avoid deadlock in the system = 7

→ The maximum no of resources Required to cause the deadlock will be 6

(8) Consider system with "n" process each process demanding 2 resources. Assume that system having resources. 6 What is maximum number of process can be in a system which is not leading to deadlock.

Sol



Given = R=6

n=?

Pid	max	Allocat	
1	2	2	Let n=2
2	2	2	

Pid	max	Allocation	
1	2	2	Let n=3
2	2	2	
3	2	2	

Let n=4 R=2

pid	max	Allocation
1	2	1
2	2	1
3	2	1
4	2	1

4+(1) no deadlock

Let n=5 R=1

pid	max	Allocation
1	2	1
2	2	1
3	2	1
4	2	1
5	2	1

no-deadlock

5+1

~~for~~ n=6

→ There will be a deadlock

Because R will became '0'

~~Then~~ The max no of process that ~~can make~~ will not lead to deadlock with 6 Resources are 5

→ The minimum no of process that will ~~not~~ lead to deadlock with 6 Resources = 6

Q). Consider 5 process, the claim vector, allocation vector has follows

pid	(demand) Claim	Allocation
1	5	3
2	3	2
3	4	2
4	8	4
5	5	3

What will be minimum resource R so that system enters in to deadlock

don't use for max-1

Memory management techniques.

Pid	claim	Allocation	Need	avalaible
1	5	3	2	1
2	3	2	1	<3>
3	4	2	2	<6>
				<8>
4	8	4	4	<11>
5	5	3	2	<15>

minno of enough

R-15 i to avoid deadlock

→ max no of Resources to go in deadlock = 14

→ Take initial value in avalaible = min value in the need.

Deadlock Ignorance.

just ignore if deadlock comes and system will crash

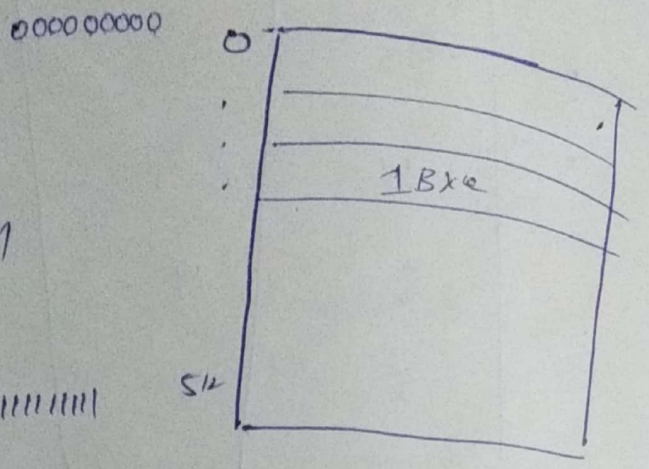
memory management :

* It deals with main memory i.e with RAM

* Ram (or) mm

is one dimensional large size array which contains set of locations,

set of words,
set of Bytes

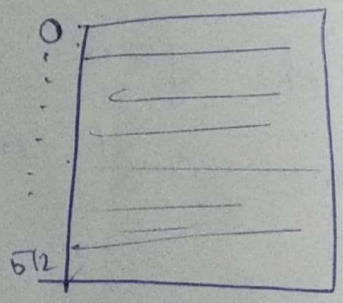


① N = no of words / Bytes

② m = length of the word

③ n = no of ~~required~~ bits required to addres a word / location

from above diagram



W = 512
m = 8bit
n = 9bit

* ~~Ram~~ Relation

① $N = 2^n$

② $n = \log_2 N$

③ memory size = N x m (Byte / bits)

$= 2^n \times m$

N= 128K words

m= 16bits

m=16bits
2B×KS

(1) words | words in memory = 128k words

Bytes = 128k × 2B = 256KB

bits = 256k × 8bits = $2^8 \times 2^{10} \times 2^3$ bits

$= 2^{21}$ bits

$2^3 = 8$

$2^5 = 32$

$2^7 = 128$

$2^9 = 512$

$2^{10} = 1024B = 1KB \cong 10B$

$2^{11} = 2048B = 2KB$

$2^{12} = 4096B = 4KB$

$2^{13} = 8192B = 8KB$

$2^{14} = 16384B = 16KB$

$2^{17} = 2^7 \cdot 2^{10} = 128KB$

$2^{19} = 512KB$

$2^{20} = 2^{10} \cdot 2^{10} = 1MB \cong 10^6 B$

$2^{21} = 2MB$

$2^{23} = 8MB$

$2^{28} = 256MB$

$2^{30} = 1GB \cong 10Bytes$

$2^{35} = 8GB$

$2^{39} = 512GB$

$n = 27$ bits

$m = 32$ bits

① words $(N) = 2^n = 2^{27}$ words

$= 128M$ word

$n = 27$ bits

② Bytes = 128M words

Given length of word = 32 bits

$= 128M \times 4$ bytes

$= 512MB$

$n = 29$

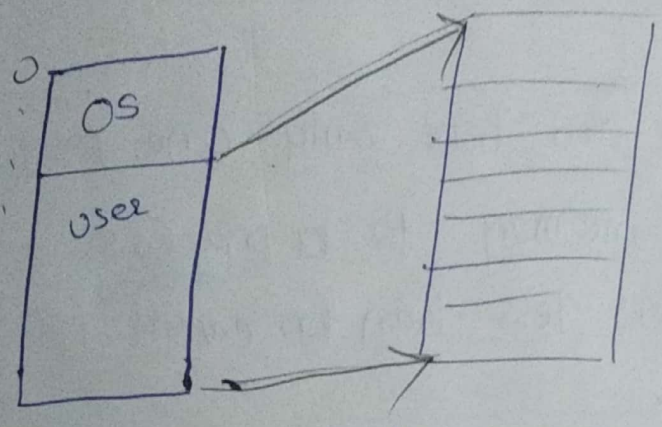
③ bits = $512M \times 8$ bits

$= 2^9 \cdot 2^{20} \cdot 2^3$ bits $= 2^{32} = 45$ bits

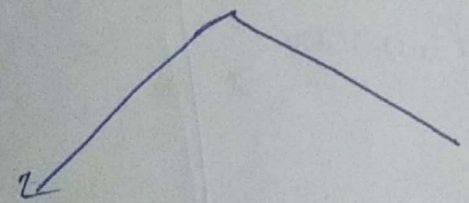
$n = 32$ bits

* Memory management techniques *

* Generally main memory divided in 2 parts
  1) OS Resident
  2) User space



→ User area of the memory further division takes place to accomodate several processes
→ These division is carried out by the operating system dynamically is know memory management



① Contiguos Allocation

Non-contiguos Allocation

[Centralized Allocation]

[entire program is posted in different region of the memory]

partition

① fixed partition (multiprogramming with fixed no of task (mft))

② variable partition (mvt)

→ Simple paging
segmentation
segmentation pages
→ virtual memory

I. Contiguous Allocation

① fixed partition (M.F.T)

(M.F.T = Multiprogramming with fixed no of tasks)

→ MM is divided in to equal (or) unequal sized partitions.

→ Each partition can hold only one program

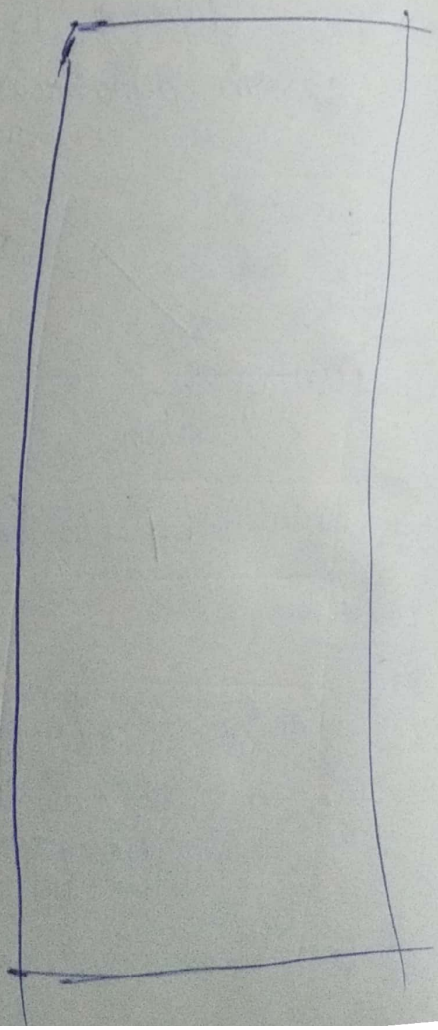
→ Allocate the memory to a process whose size is less than (or) equal to partition size.

Let consider process Request

P1: 120KB, P2: 200KB

P3: 160KB, P4: 220KB

P5: 140KB, P6: 250KB



Allocation policies.

① first fit

② Best fit

③ Next fit

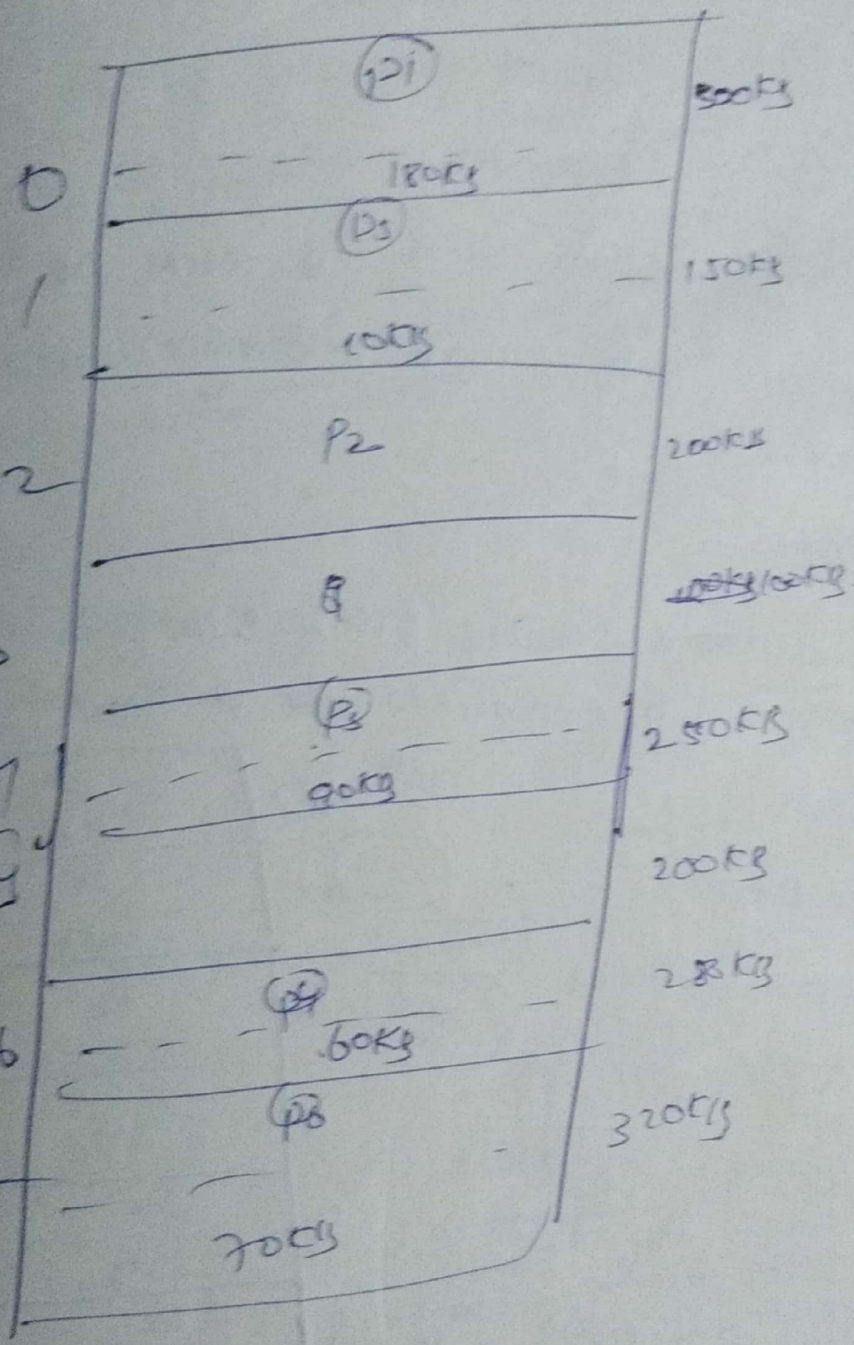
④ worst fit

First fit

→ Allocate the partition with first free hole (or free block that is big enough.

(Or)

→ If scan the memory from beginning allocate the free block that is large enough.



If (internal fragmentation) = 60kB + 30kB

= 410kB

If percentage in memory

$= \frac{410}{} \times 100$

300 + 150
+ 200 + 100
+ 250 + 200
+ 280 + 320

first fit performs better in terms of time

first fit gives worst performance in terms of space

(1) Bestfit :-

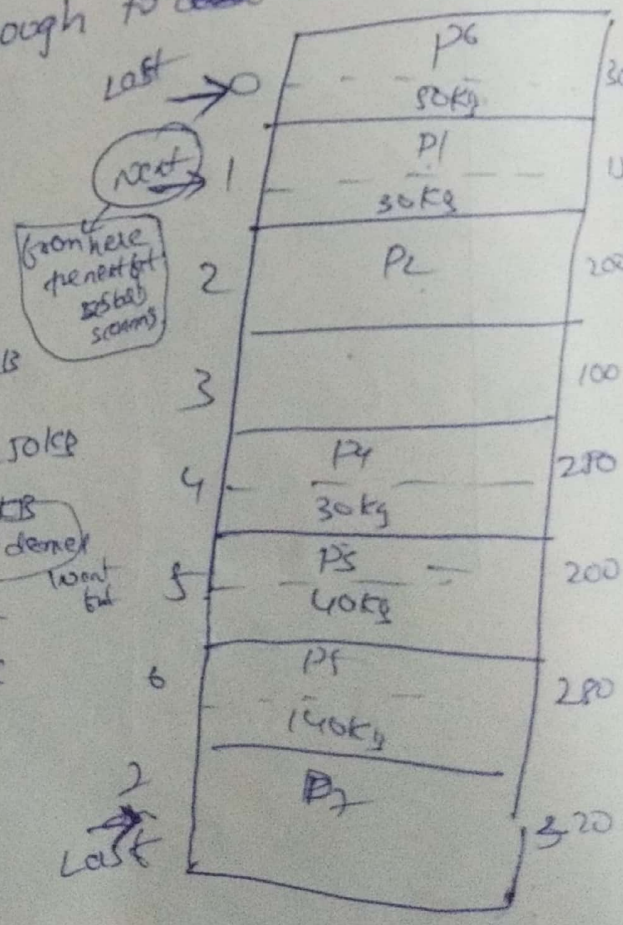
→ It allocates smallest free hole or free block that is big enough to accomdate a process.

Process Request

P1 : 120kB, P2 : 200kB
P3 : 160kB, P4 : 220kB
P5 : 140kB, P6 : 250kB

P7 : 120kB, P8 : 180kB ↑ denied

If 50kB + 30kB + 40kB + 140kB
= 290kB



→ It gives better performance in terms of space

→ It is takes more time to process

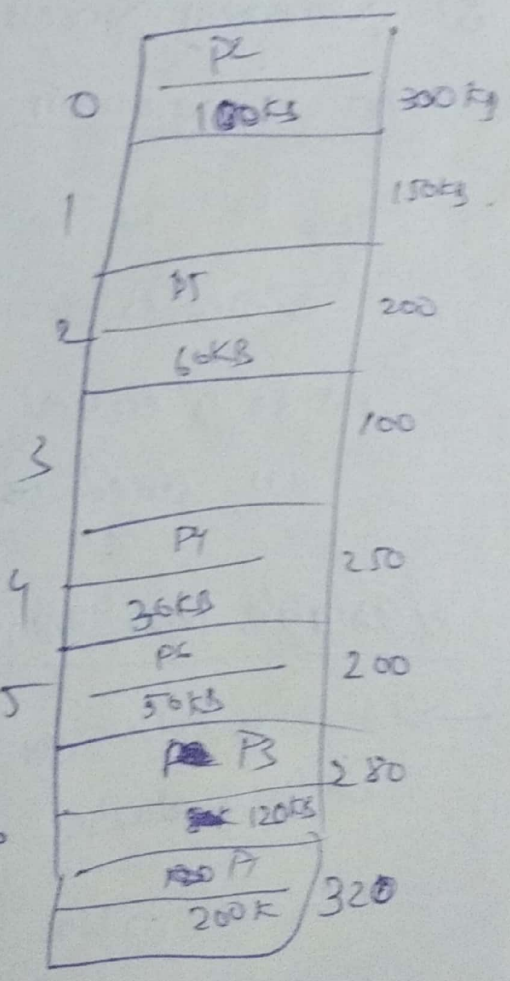
Next fit :-

It is same as a first fit except it scans the memory from next to last placement was made

worst fit:-

It allocate largest position that is big enough to accomodate a process

P1: 120KB, P2: 200KB
P3: 160KB, P4: 220KB
P5: 140KB, P6: 150KB



If= 100+60+30
+50+120+200
= 560KB

worst fit would have more IF

performance of fixed partition

1) Best fit performs better interms of space
2) Internal fragmentation is present
3) No External fragmentation
4) Degree of multiprogramming is restricted by no of partitions

> Degree of multiprogramming = No of fixed partions

5) Maximum process size is proportional to maximum partion size

i.e The maximum process can be stored is, = whose program size is equal to largest partition in fixed partition.

Variable partioning :

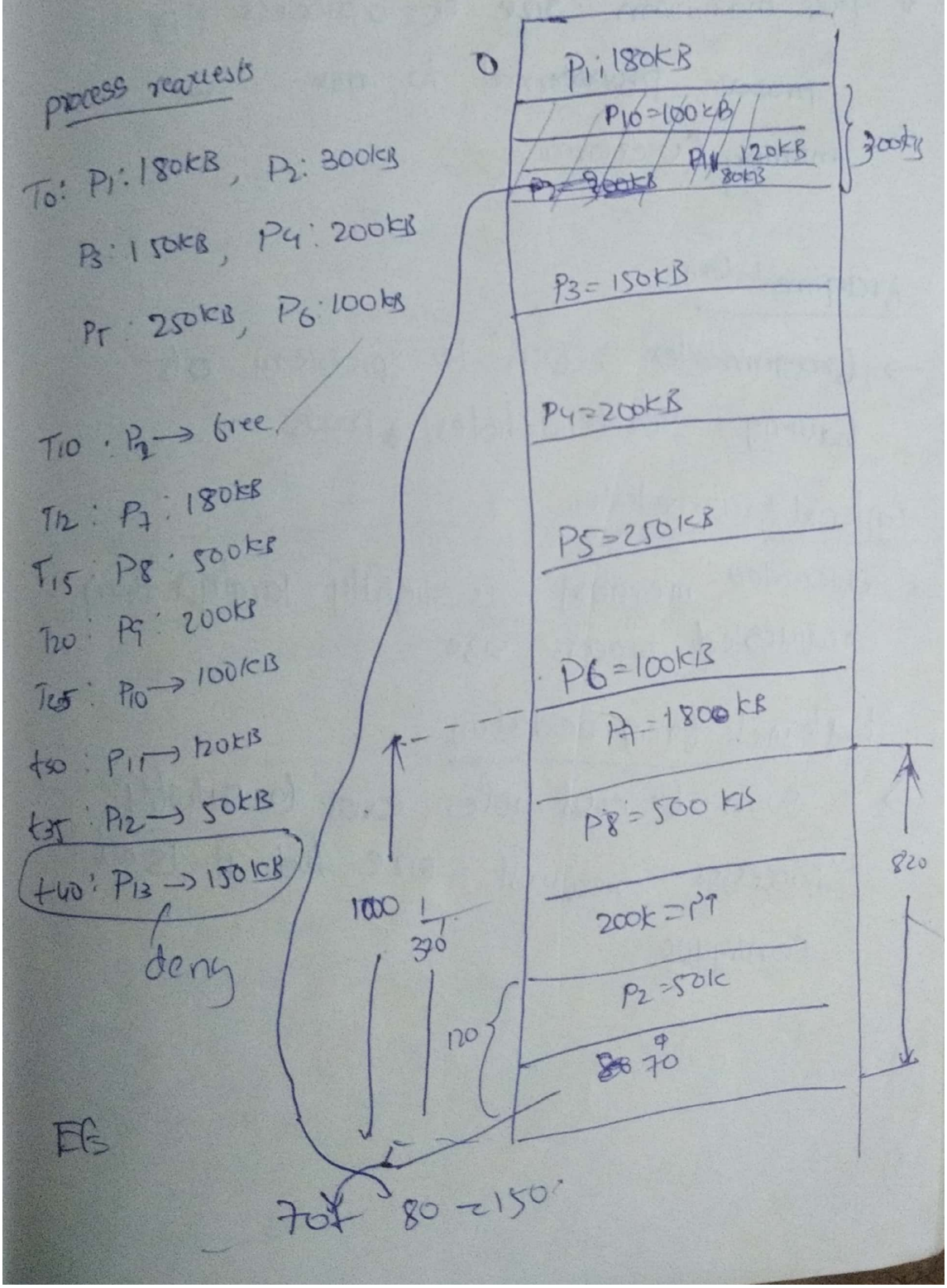
→ It is also refered as multiprogramming with variable hoof tasks

→ It is also reffered as dynamic contiguous allocation method

→ Here partitions are variable size and variable members

→ Allocate the memory exactly as much process is requires

→ Here ~~first fit~~ worst fit allocation will perform better



Performance of variable partition

1) No internal fragmentation

2) External fragmentation is (✓)

3) Degree is of multiprogramming
   ↳ Flexible

4) The maximum size of a process is proportional to user memory (User area)

fragmentation

① → fragmentation refers to problem of having scattered holes (• pools)

internal fragmentation:

② → allocated memory is slightly larger than requested process size

External fragmentation

→ Sum of all holes exist to satisfy processes request size but it is not contiguos

solution of external fragmentation

1. Compaction (joining the free spaces (or) by forcible compression)
2. NON contiguous allocation

~~Compaction~~ **Compaction :**

→ combining of all free holes in to single big hole at the end part of memory

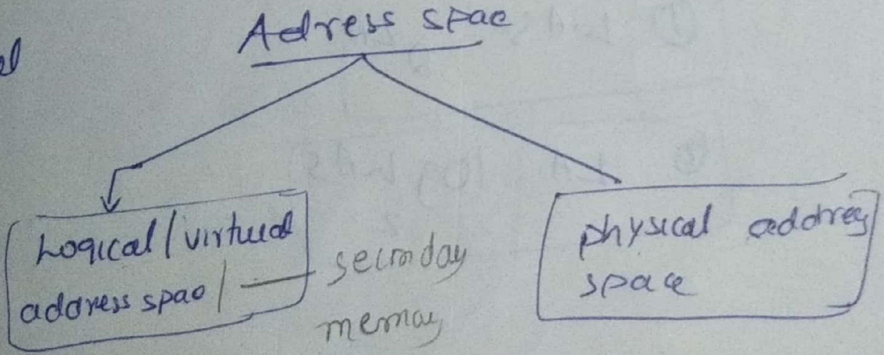
→ Inorder to implement compaction Hardware support is highly needed i.e reallocatable → changing addresses dynamically.

NON-Contiguous Allocation :

Address space :

→ set of Locations/words/ set of addresses

~~Logical~~



Logical address space : collecti of / set of Logic addresses

→ program size is nothing but logical addresses

Physical address space

(or absolute address) set of physical addresses / set of all addresses in memory (MM)

Logical Address :

Logical address which is generated by cpu.

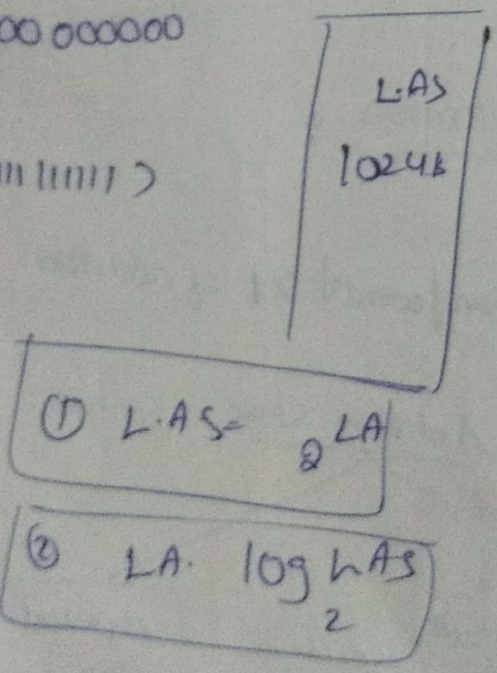
→ It is a reference to a memory location, independent of current assignment of data to a memory location.

→ must be translated into physical address

LA = 10 bits

<0000 000000

<111 111111>



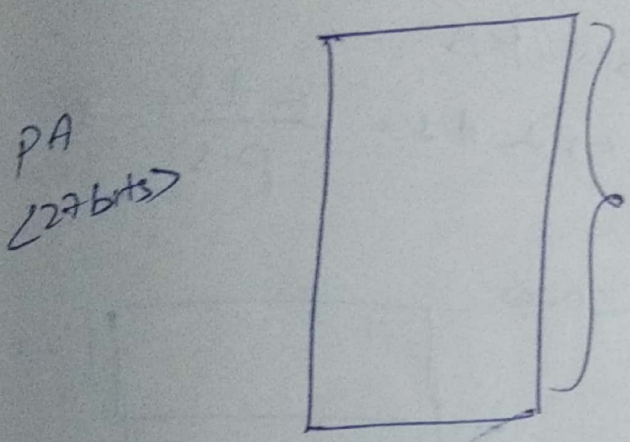
① L.A.S = 2^{LA}

② LA. $\log_2$ LAS

PA:

→ physical address is ~~a ...~~ which is seen by the memory unit. It is also reffered absolute address (or) actual locations in the memory

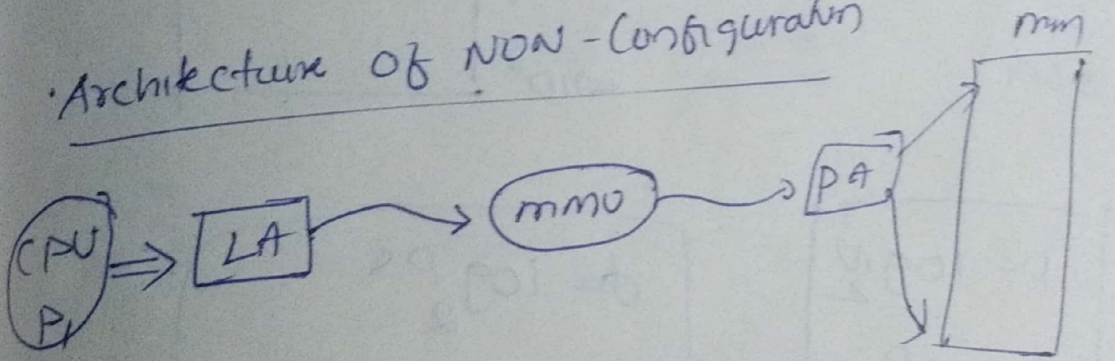
MM = PAS



① PAS = 2^{PA}

② PA = $\log_2$ PAS

· Architecture of NON-Configuration



■ Simple paging is implemented by

① organisation of LAS

② " of PAS

③ " of mm

④ Address Translation (LA to PA)

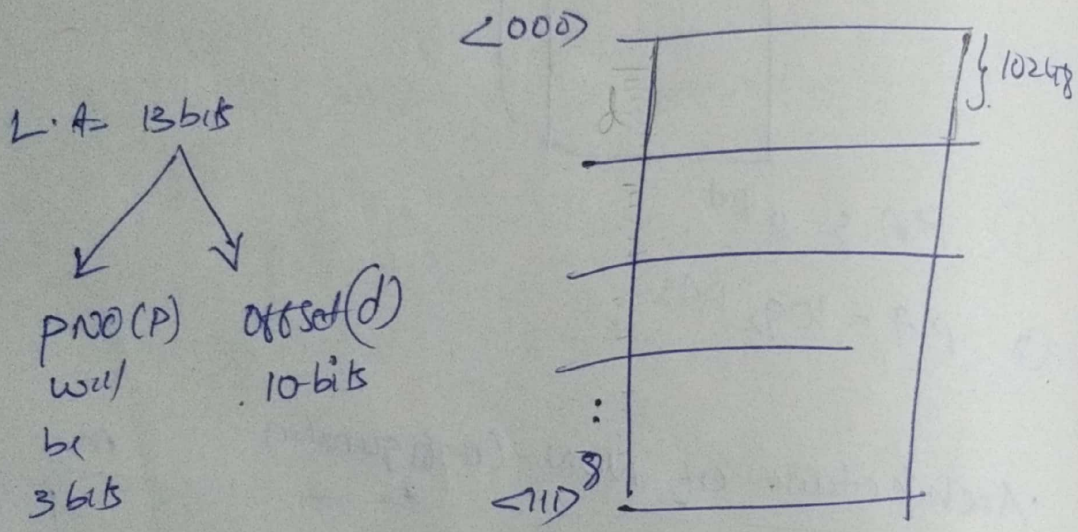
① Organisation of L.A.S

→ Divide the L.A.S / Virtual Address space/ program address space into equal sized units are called pages

Let L.A.S = 8KB, L.A = 13 bits

Let page size = 1KB

# of pages (N) in L.A.S = $\frac{L.A.S}{P.S} = \frac{8KB}{1KB} = 8$



L.A = 13 bits

PNO (P) will be 3 bits

offset (d) 10-bits

$P = \log_2 N$

$N = 2^P$

P = No bits to address page

N = No of pages

$d = \log_2 P.S$

~~P.S = 2 Bytes~~

$P.S = 2^d$ Bytes

P.S = Page size

d = No of bits to address a word / Byte within a page

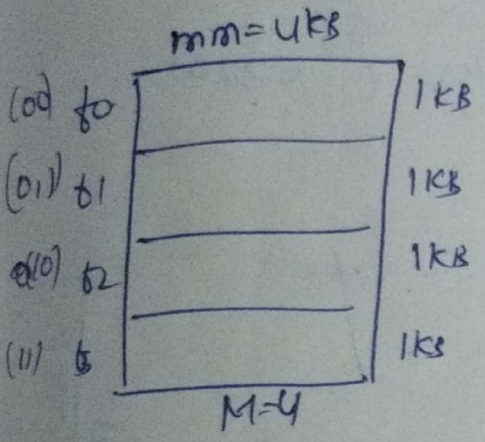
② Organization of main memory (mm)

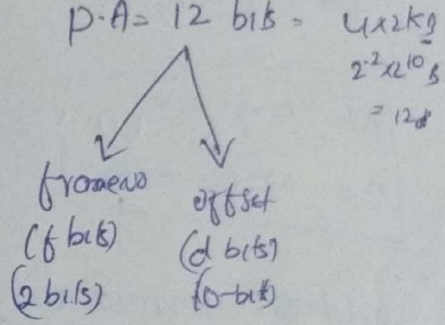
→ Complete main memory is divided into equal size page frames / frames, where these size being equal to frame size

→ Let P.A.S = 4KB, & Let frame size = 1KB

→ No of frames (M) in physical address space

$$= \frac{P.A.S}{\text{page size}} = \frac{4KB}{1KB} = 4$$





→ frame no (f bits) depend $f = \log_2 M$

→ $M = 2^f$

→ offset (d bits) depends on $= \log_2 PS$

③ organization of memory management unit

→ organizing the page table is called memory management unit'

→ Page table is organized into series of entries where no of entries in the Page table is equal to no of entries in virtual address space'

→ page table entry contains essentially frame number of physical address space

→ every process has its own page table

→ page tables are kept in main memory (it is pure overhead) (kept in frames)
↳ burden

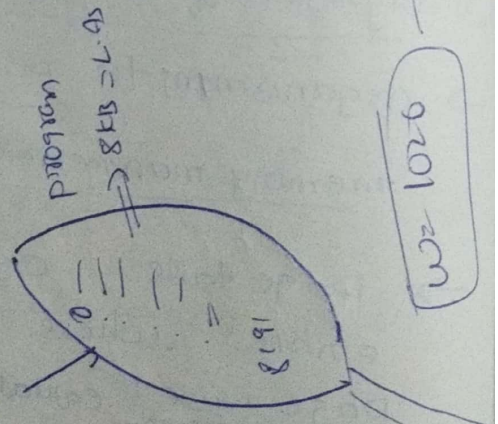
→ page table size is measured in to series of entries multiplies by entry size
(col nodes entry)

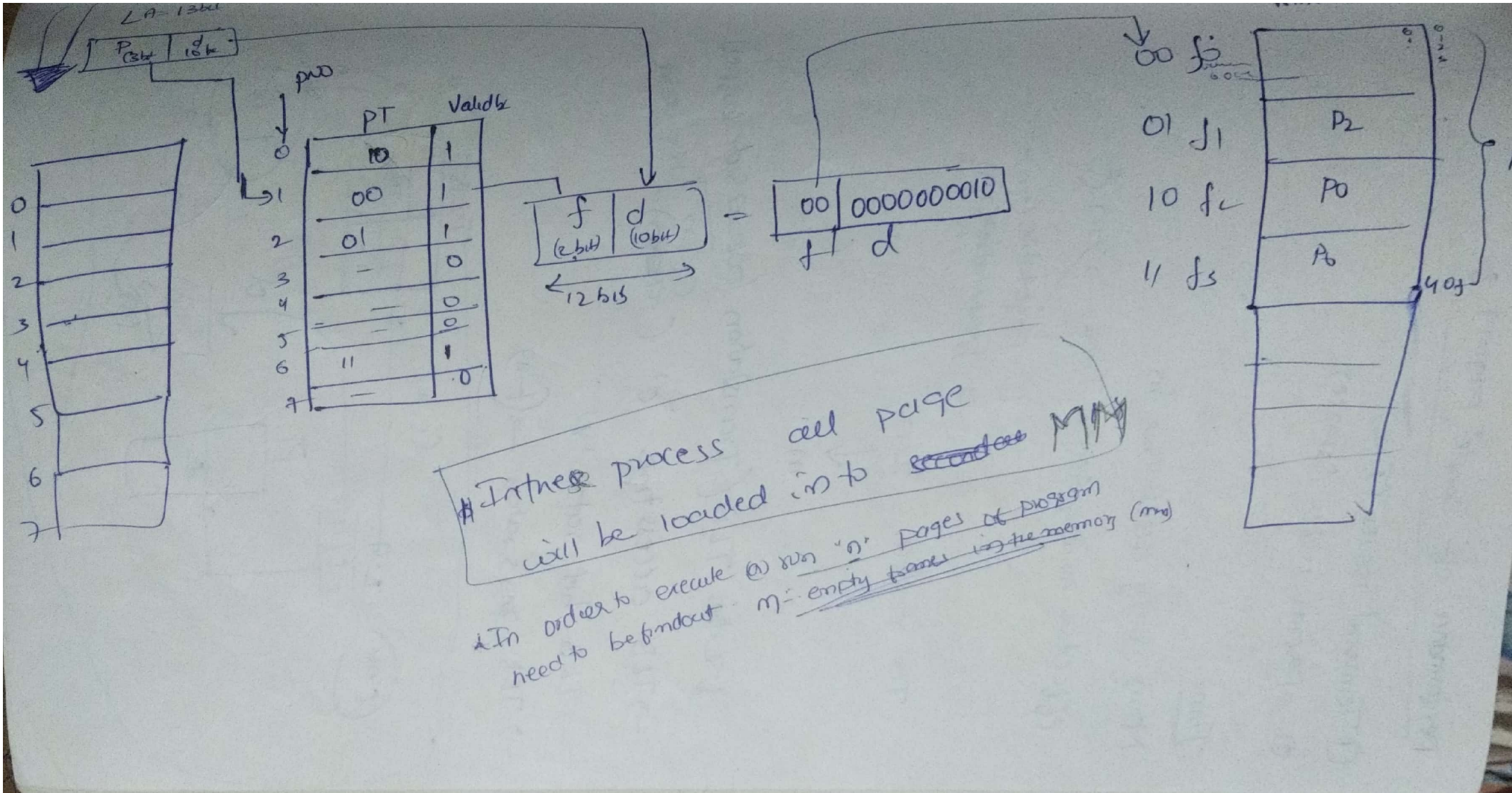
PT size = N * e Bytes [e >= 1B]

N = ~~Page size~~ No of Pages
e = No of entries

(4) Address translation:

→ Converting from logical address to physical address





Performance of simple paging :

1. Temporal issues (Time)
2. Spacial issues (space)

Time

Main memory access time 'm'

effective memory Access Time → 2m :

1. page table (m)
2. frame table (m)

To reduce 2m

↓

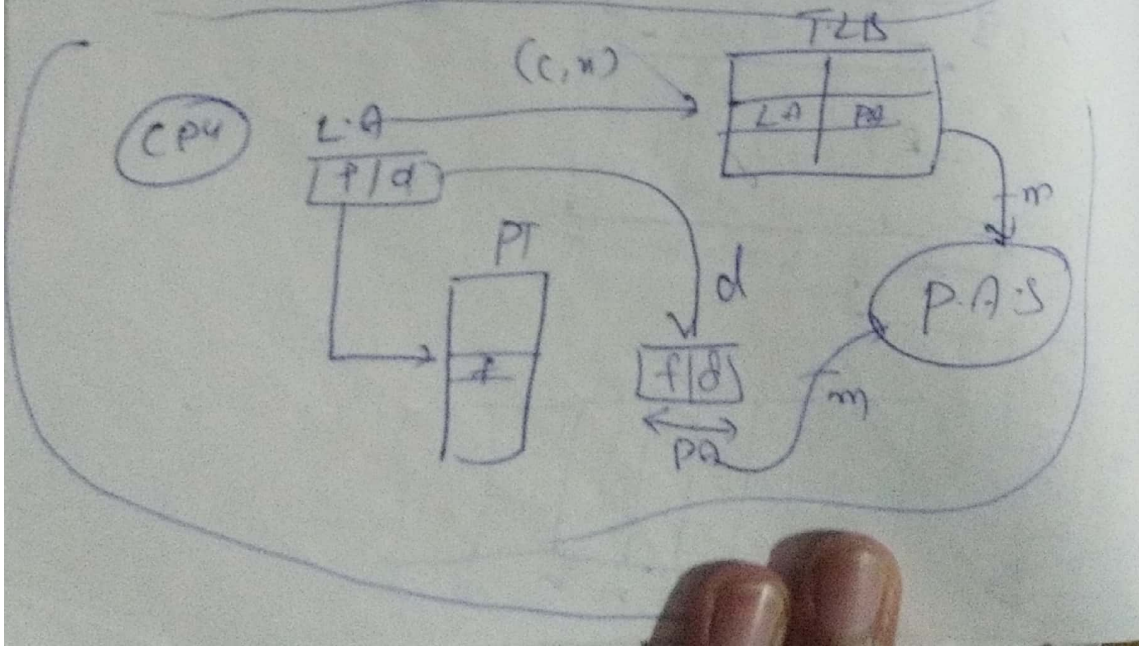
m

Use the TLB (Translation look aside Buffer)

→ TLB access time 'c' (c<<m) very less

→ TLB hit ratio 'x'

→ TLB miss ration → (1-x)



Effective access time with TLB with simple paging:

$$x(c+m) + (1-x)(c+2m)$$

→ for access of TLB

→ 1m for page table, 1m for perform

→ miss ratio

Ex

let $m = 100ns$

$c = 10ns$

$x = 90\%$

Emem access $= 90\%(10+100) + 10\%(c+2m)$

$= 99ns + 21ns = \boxed{120ns}$

→ If simple paging without TLB used then the effective access time would be $\boxed{200ns}$

(5) Consider logical address 22 bit, physical address 18 bit, page size 2 KB, page table entry(e) = 2 bytes, what is the logical address space, physical address space, capital (N), M, f (frame number), p (page number), d and page table size?

Note: page table entry (e) = 2 byte

frame no.	absent/present bit	protection bit

A) Logical Address

$= 2^{18}$ B $\quad 2^{22}$ B

$= 2^{8} \cdot 2^{10}$ B $= 4$ MB

256 KB

B) physical Adress space

$= 2^{18}$ B

$= 256$ KB

c) N(No of pages) = $\frac{\text{Logical adress space}}{\text{page size}}$

$= \frac{2^{8} \cdot 2^{10}\text{ KB}}{2\text{ KB}} \quad \frac{4\text{ MB}}{2\text{ KB}}$

$= \frac{2^{8} \cdot 2^{10}\text{ KB}}{2\text{ KB}} = \frac{2^{2} \times 2^{10} \cdot 2^{10}\text{ B}}{2\text{ KB}}$

$= \frac{2^{22}\text{ B}}{2^{11}\text{ B}}$

$= 2^{11}$

$= 2$K page

d) M(No of frames) = $\frac{\text{PAS}}{\text{frame size}} = \frac{2^{8} \cdot 256\text{ KB}}{2\text{ KB}}$

$= 128$

Here frame size = page size

A) Logical Address space

$= 2^{18}$ B $\quad 2^{22}$ B

$= 256$ B $= 4$ MB

256KB

B) physical Adress space

$= 2^{18}$ B

$= 256$ KB

c) N(No of pages) = No of Logical address space / page size

① $= \frac{4\text{MB}}{2\text{KB}}$

$= \frac{2^2 \times 2^{10} \cdot 2^{10}\text{B}}{2\text{KB}}$

② $= \frac{2^{22}\text{B}}{2^{11}\text{B}}$

$= 2^{11}$

$= 2\text{K}$ page

d) M(No of frames) = $\frac{\text{PAS}}{\text{frame size}} = \frac{256\text{KB}}{2\text{KB}}$

$= 128$

Here frame size = page size

6) $f($ [redacted] $) = \log_2 M = \log_2 2^7 = 7$

↓ number of bits required to address (or) $128 = 2^{7} \rightarrow 7$

7 bits are required to address

7) p bits $= \log_2 N = 11$ bits

8) d bits $= \log_2 PS = \log_2 2KB = \log_2 2^{11} = 11$ bits

9) page table size = ~~no~~ no of entries ~~into~~ × no of ~~pages~~

$= 2K \times 2B$ (given)

$= 4KB$

$\frac{512}{2^{10}} = 2^{2^2}$

10) Consider system with 512 frames, logical address 22 bits, page size 4KB, find out logical address space PAS.

1 frame size 4KB

LAS = 2^22 4MB

PAS = 2MB

m = 512

N = 4MB / 4KB = 1K

$\log_2 1K = 10$

P = 10

M = 2MB / 4KB = $\frac{2^{21}}{2}$

LAS $= 2^{22} = 4MB$

PAS:

$M = 2^f$ [f is no of bits]

[m is no of frames]

$512 = 2^f$

$2^9 = 2^f$

$f = 9$

PAS $= 2^9 = 512B$

~~No. of pages~~

(3) P.A [f | d]

$f = \log_2 M = \log_2 512 = 9$ bits

M = total no. of frames

$d = \log_2 P.S = \log_2 4KB = 12$ bits

(or) given 512 frames

frame size = page size

= 512 × 4KB
= $2^9 \times 2^2 \times 2^{10}$ B
= 2^{21} B
= 21 bits ⇒ P.A

(4) P.A = f + d
= (9+12) bits
= 21 bits

(4) $P.A.S = 2^{21} = 2MB$

(5) $N = \frac{4MB}{4KB} = 1K$

(6) $P = \log_2 N = \log_2 K = 10$ bits

(7) page Table size = N × e Bytes
= 1K × 2B
= 2KB

e ~ ~~frame~~ frame no + other info
(Valid bit, protection bit,
Dirty, ref bit)

e = f + other
e = 9 bits ≅ 2B

(2) spacial issues
(space)

let L.A = 32bit, L.A.S = 4GiB

page size = 4KB

# of pages (N) = $\frac{4GiB}{4KB}$ = 1M entries

let e = 2B

page table size = N x e Bytes

= 1M x 2B

= 2MB

When you
* Increase the page size that will reduce pagetable size
* But internal fragmentation will be increases
* When you decrease the pagesize it will reduce internal fragmentation but increase pagetable

increase

NOTE = internal fragmentation will be only in the last page

4

program = 1026B × 2

small

large

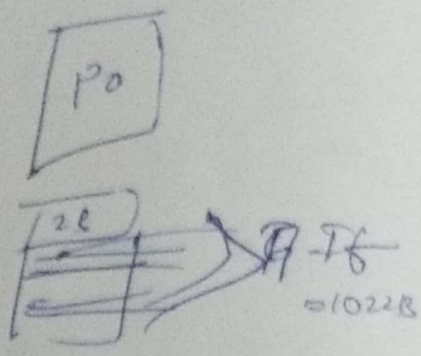
page size = 2B

page size = 1KB

① N = 512

② PTS = 512 × 2B
= 1026B

① N = 2



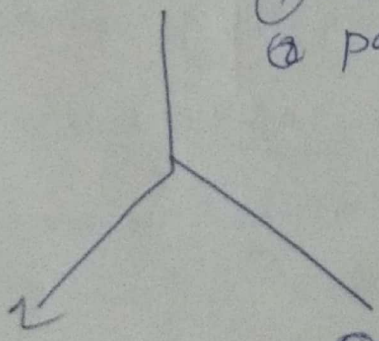
② PTS = N * e
= 2 × 2B = 4B

I·f=0

Solutions to minimizing

① If

② page table size



① finding optimal (page size)

sol. $P = \sqrt{2 \times S \times e}$

S = logical space

e = #e in page table

② multilevel paging

SEGMENTATION

→ Logical address space is divided into Logical 'chunks' Representing function, procedure, method, routine and data structure are known as segments.

→ Each segment may be variable size.

→ Memory References are translated into physical Address space through segment table.

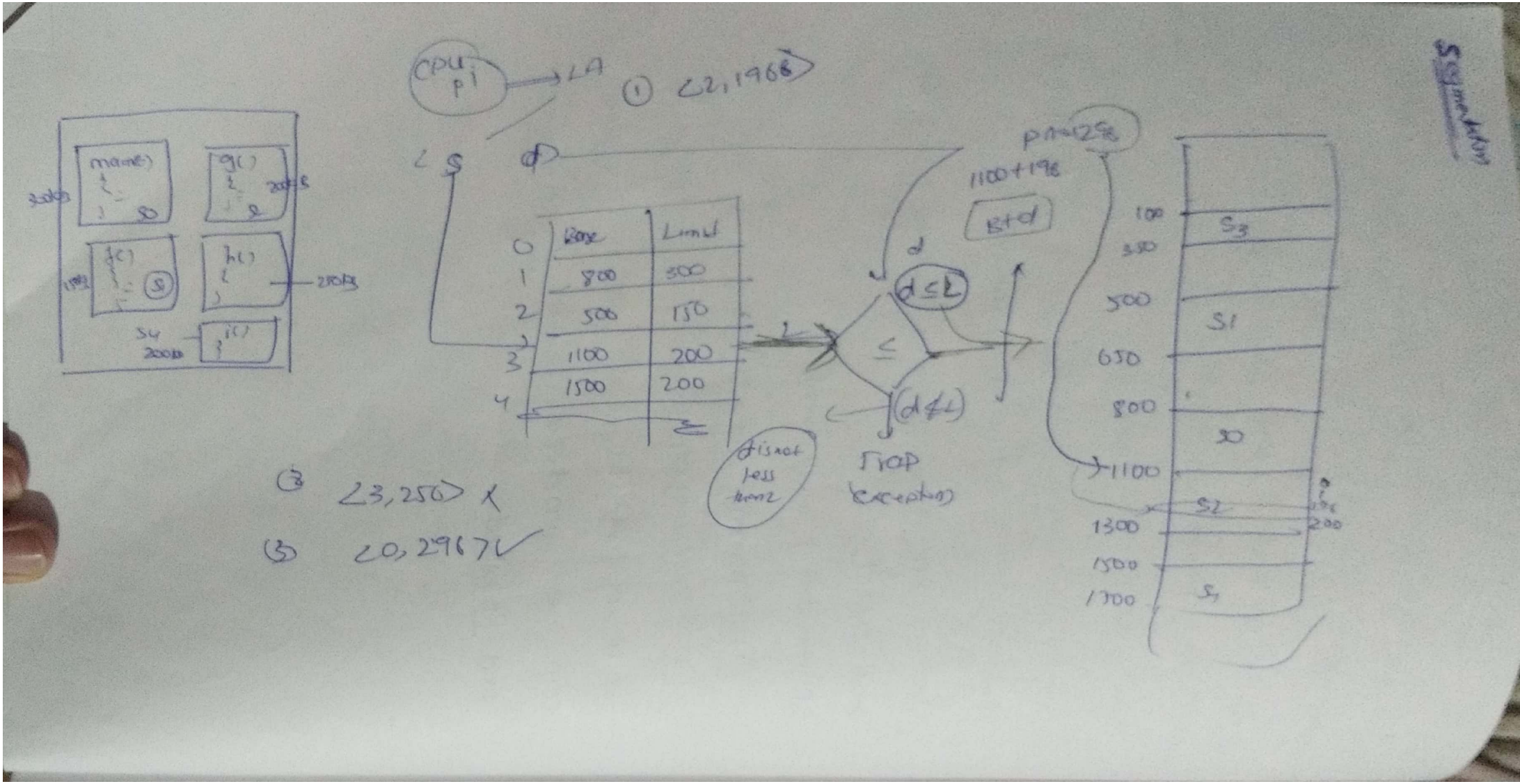
→ Memory Reference contains segment number and offset

→ Segmentation will handle

(1) Growing data structure

(2) Segmentation supports Reusability

→ Segments of Logical address space can be independently Re-compiled and Re-linked



DOTF
segments size are function size because all functions divided into to the segments

performance of segmentation :-

① Temporal issues :-

m → m

Effective memory access Time → 2m

① to get segment Table
② into offset

To reduce 2m → m.

* TLB can be used to reduce memory access time 2m → m

② Special Issues

① NO · Internal fragmentation
② External fragmentation

Virtual memory :-

* Virtual memory gives an illusion to the programmer that a huge amount of memory is available greater than available physical memory.

→ Virtual memory concept is implemented by "Demand paging"

→ mapping of logical address space to secondary storage device like disk and loading pages of process in to memory on demand basis is known as demand "paging".

→ In Demand paging all pages of process need not to be load in to memory at a time (or) during at Runtime. The Disk Load those pages which are required to compute immediately

→ Demand pagins needs les I/o operation

→ It allows more processes (or users to run (Degree of multiprogramming increases)

→ It improves cpu utilization and throughput

→ Demand pagins uses Lazy Swapper (Lazy pager)

→ Demand paging can be
① pure demand paging
② pre fetched ,

pure demand:

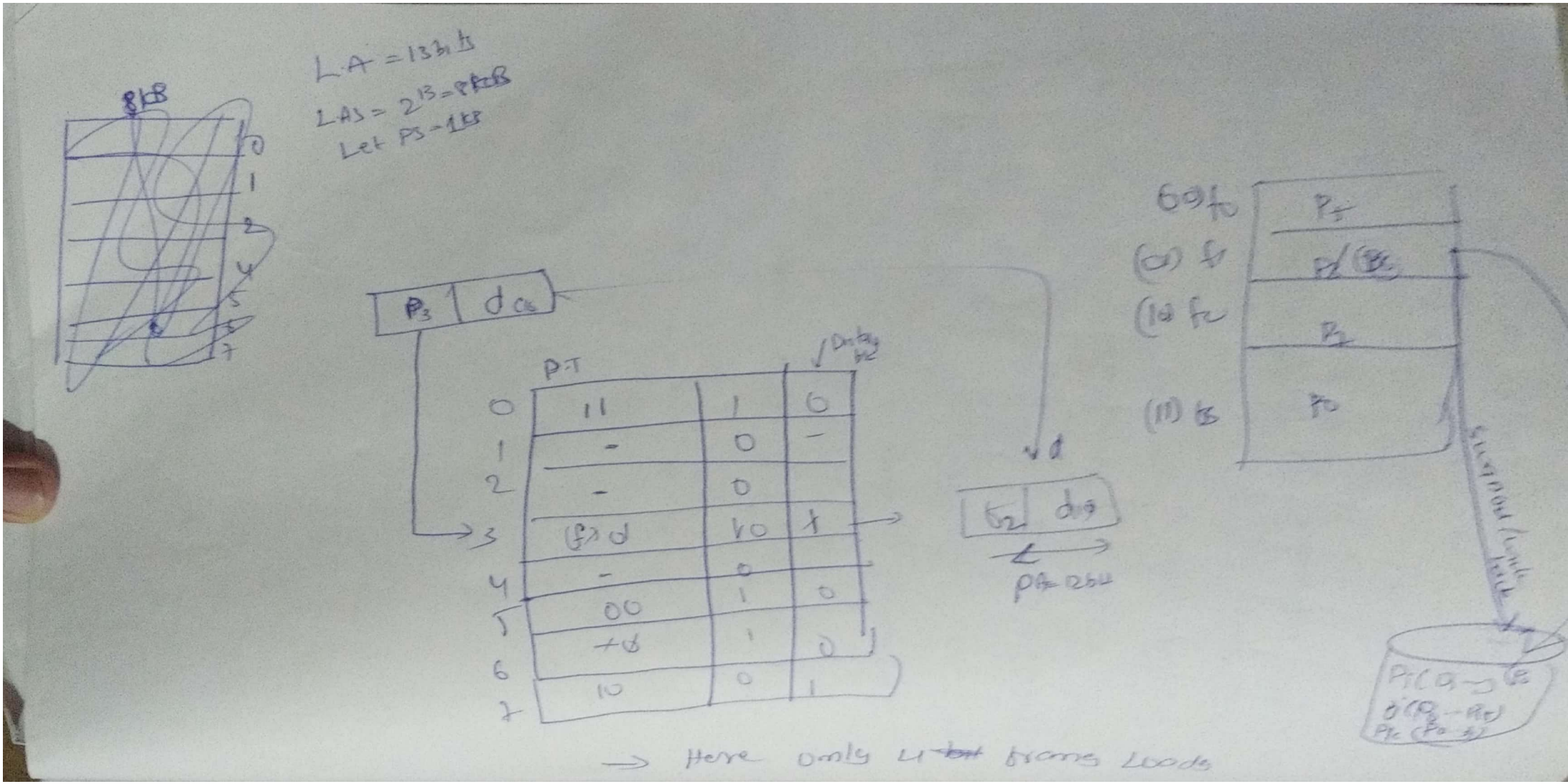
→ No pages of a process are loaded into memory i.e start the program execution with empty frames

→ First reference is pagefault for 1st reference pagefault

pre-fetched demand

* Some pages of a process loaded into memory Before Runtime

*



The difference here is that the only required page will be kept in the MM where in simple paging all the pages should be loaded into mm.

Performance of Virtual memory

let mm → 'm'

p → page fault rate

(1-p) Hit rate

S → EFST page fault service

EMAT : $(1-p) \times m + p \times S$

Page Replacement Algorithms :-

→ It occurs when there is no free frame in mm to bring requested page

→ The page Replacement algorithm minimizes the no of page faults

Reference String :

Set of successively uniquely pages that are refered in the given list of logical address space

Ref. String { 122, 204, 039, 056, 274, 356, 457, 124, 094, 572 }

page size = 100 words

122/100 → Page 1, Offset 22

204/100 → Page 2, Offset 4

{ 1, 2, 0, 5, 2, 3, 4, 1, 0, 5 }

∪ { 1, 2, 0, 4, 5, }

Algorithm:

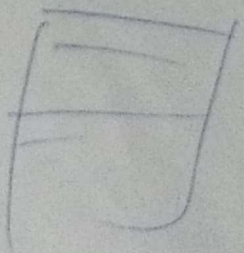
FIFO

Criteria:

It chooses the page replacement based on the arrival time of page

Consider 3 free frames in my

(3)



Ex: Strings = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}

|Uniq| = {20}

Unique = {0,1,2,3,4,7}

= 6 – total size
pages

NOTE :- if n pages for n frames
n pagefault will be occure

- if n pages for n > frames (n-1 max)
more than n pagefaults will occur

eg → 6 pages (above) if 6 frames then
6 pagefaults will occur
→ But allocated only 3 frames so the
pagefaults will more than = 6

4 frames

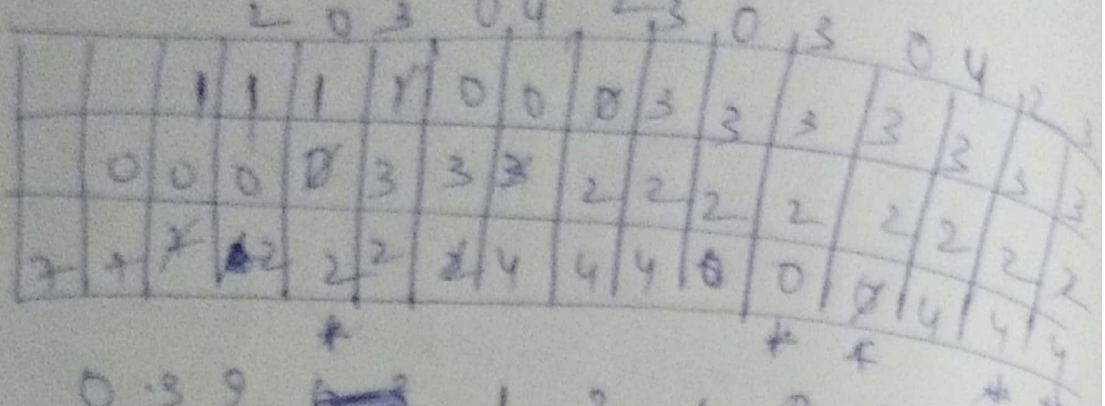
	0	3	0	4	2	3	0	2
			2	2	2	2	2	2
		1	1	1	1	1	0	0
	0	0	0	0	4	4	4	4
7	7	7	7	3	3	3	3	3
*		*	*	*	*		*	

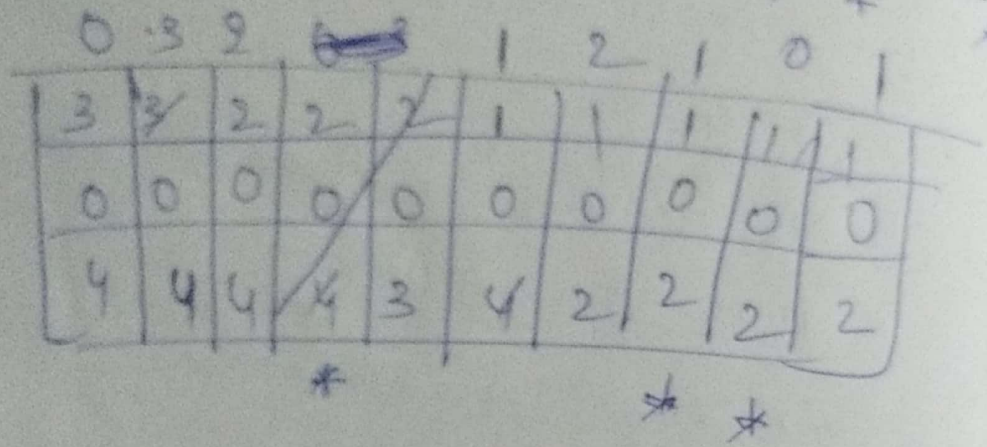
1	2	0	1	7	0	1
1	1	1	1	1	1	1
0	0	0	0	0	0	0
4	4	4	4	7	7	7
3	2	2	2	2	2	2
	*	*		*	*	

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

for strong

2 1 2 0 1 7 0 1 3 0 3





fifo and fifo based algorith suffers from the problem of Belady's algorithms
{1,2,3,4, 1,2,5, 1,2,3,4,5}

Belady's anomaly

→ with increase in no of frames to a process the page fault rate logically should decrease but for few reference string with an increase in no of frames to a process. The page fault rate is also increases is called Belady's anomaly

optimal replacement

* It chooses the page for replacement which is not required longest period of time in the future

* { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 }
  • • • • # • # • # # • # # # # # # #

① 3f

0	~~7~~	~~2~~	7
1	~~0~~	~~4~~	0
2	~~1~~	~~3~~	1

1) no of pf = 9
(2) no of hits = 11
(3) Hit Ratio = $\frac{11}{20} \times 100$
(4) status: { 7, 0, 1 }

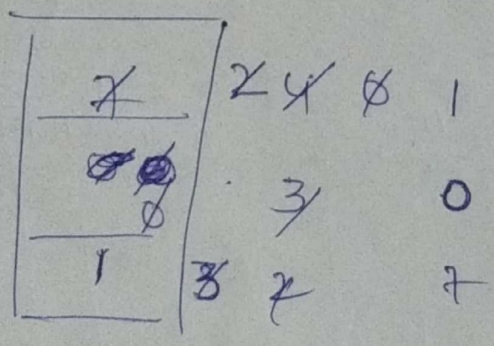
* If is not suitable to implement practical cases since predicting the future reference string is difficult

* It is a bench mark tool which is used to compare all Algorithms with this optimal

LRU (Least Recently Used)

→ It chooses the page for for replacement based on usage

{ 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 }



① No of hits = 8

② No of faults = 12

Disk scheduling Algorithms :-

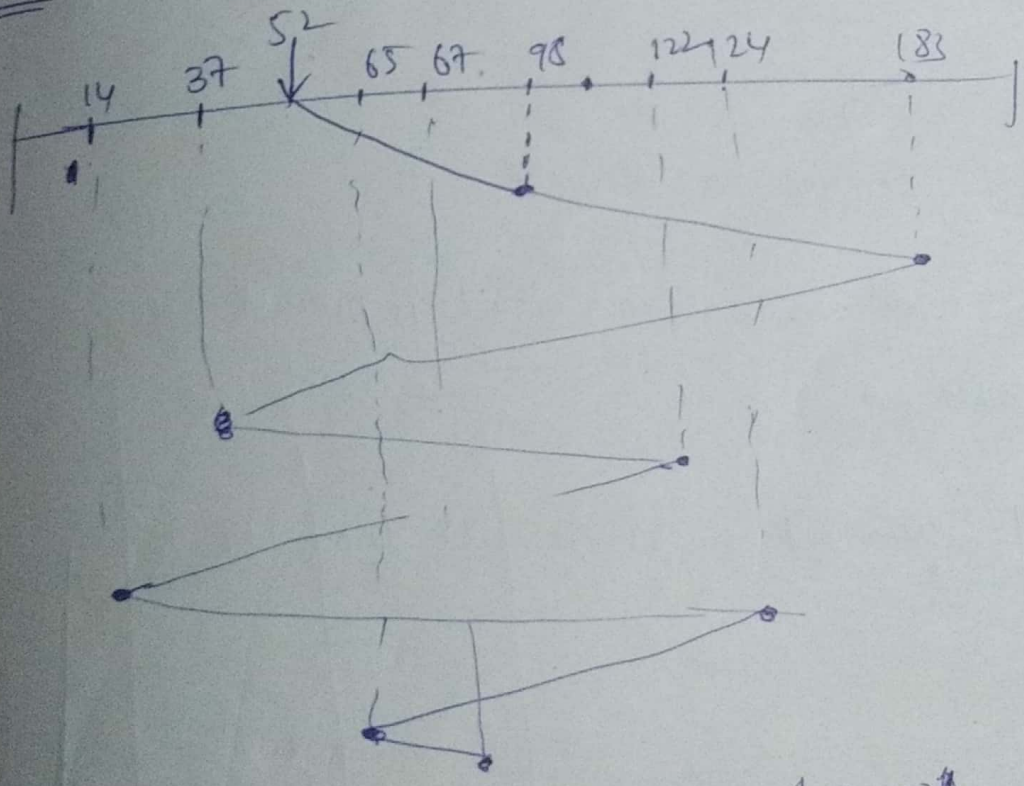
⇒ when more than one request comes to the disk we need to apply some algorithms to minimize no of seeks

⇒ Let us consider Disk has 200 cylinder which are numbered from (0 to 199). And Disk request are

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

Current header position at cylinder no 52

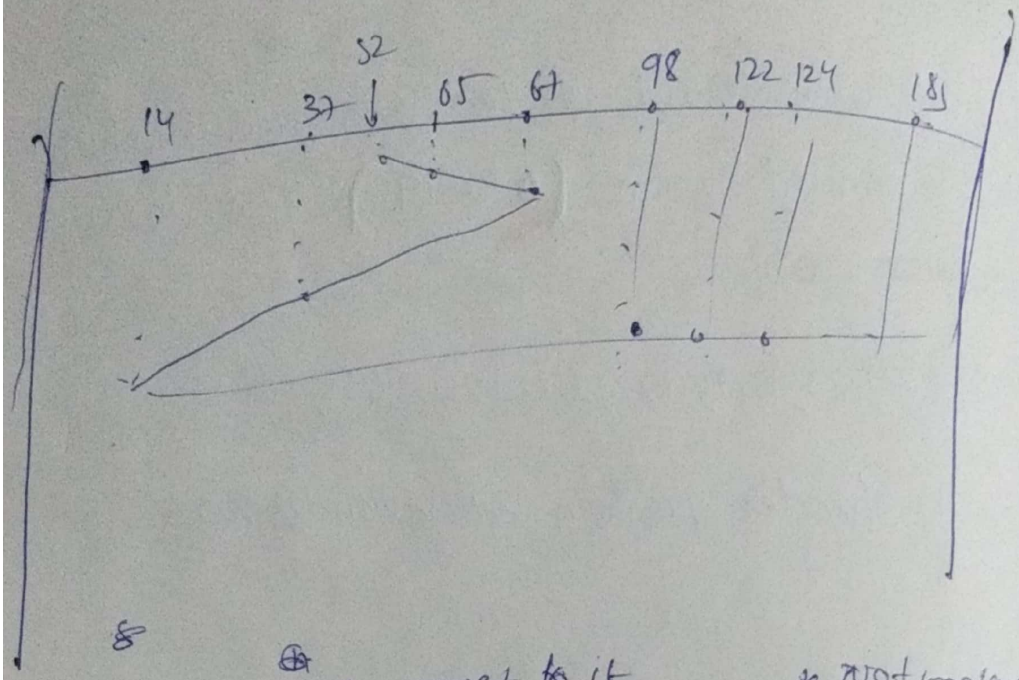
FCFS



Seeks = |52-183| + |183-37| + |37-122|
+ |122-14| + |14-124|
+ |124-65| + |65-67|
= 131 + 146 +

SSTF (shortest seek Time first)

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



&

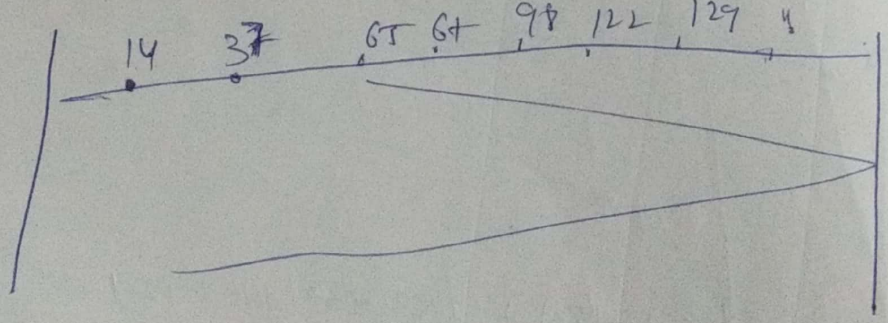
It serves nearest to it * Not implementable

= 237

SCAN (Reffered as elevator

→ Useful when heavy load is present

→ ~~gives~~ for



→ Example: how to know whether to go left (or) Right side

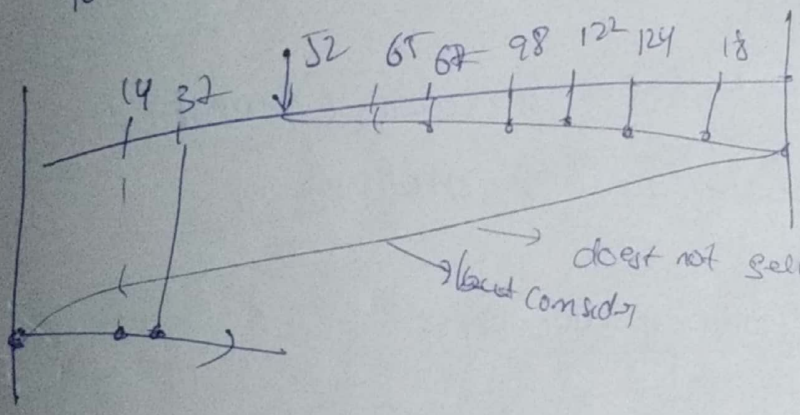
Look
does not go to complete end. ~~is~~ remaining will work as SCAN

Look is more than better than SCAN

→ It ~~also~~ also useful for heavy loaded systems

C-SCAN (Circular SCAN)

doesnot service while returning but touches the ends



|52-199| + |199-0| + |0-37|

14 199 183 16

= 383

C-Look:

It will not go to end

file Allocation methods / Disk space Allocation:

Disk size = 64MB

Disk block size = 1KB

Disk block address = 16-bits

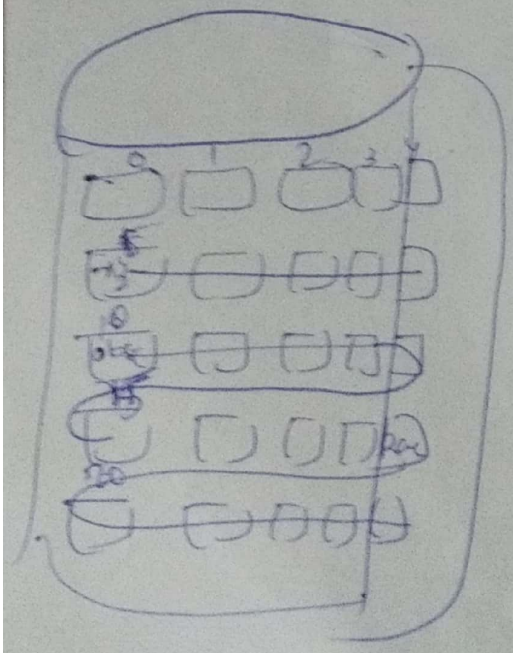
No of Disk blocks = $\frac{64MB}{1KB}$ = 64k

Types of Allocation methods

1. Contiguous Allocation method
2. Non-contiguous Allocation, (Linked Allocation)
3. Indexed Allocation method.

Contiguous memory allocation:

→ contiguous free blocks are assigned for requested file size



filename	start DBA	no of blocks
abcc	10	7
xyzdo	5	3
Ramesh	19	6
Ravi	x	c

~~If we can a~~ → size can b

performance

→ Internal fragmentation is present

→ External fragmentation is present

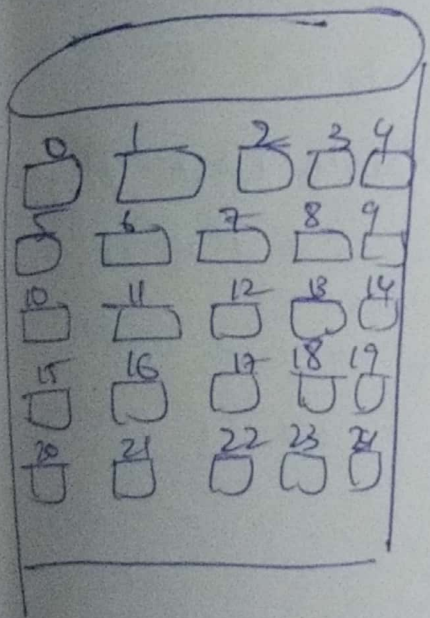
→ increasing file size is may(or) may not flexible(or) not flexible

→ Type of access

→ sequential access

→ Random access (base + number of blocks to be accessed)

Linked allocation:



size	filename	start DBA	end DBA
1	abc.c	10	7
2	xyz.doc	21	19
3	pqu	0	14

1 - {10, 16, 13, 9, 3, 7}

2. {21, 17, 23, 19}

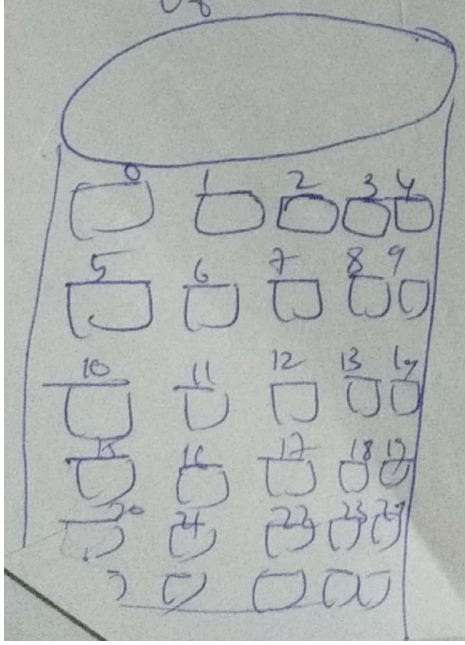
3. {0, 6, 11, 8, 14

Performance

1) Internal fragmented (✓)
2) external fragmetal (x)
3) Increasing the file size flexible
4) Type of access sequential
5) if datablock consist 'n' elements. (n-1) elements used to to store data and 1 is pointer date block
6) memory likage: if link breaks

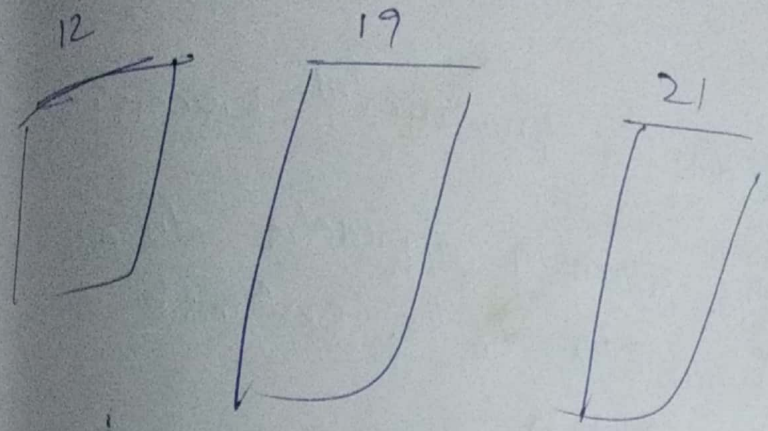
Indexed Allocation:

→ If file contains 'n' data block Then (n+1) datablocks need to be used. i.e for each file there most be one index block which contains list of addresses of data locks



SNO	Fname	Index Block
1	abc.c	12
2	xyz.abc	19
3	Kiran	20

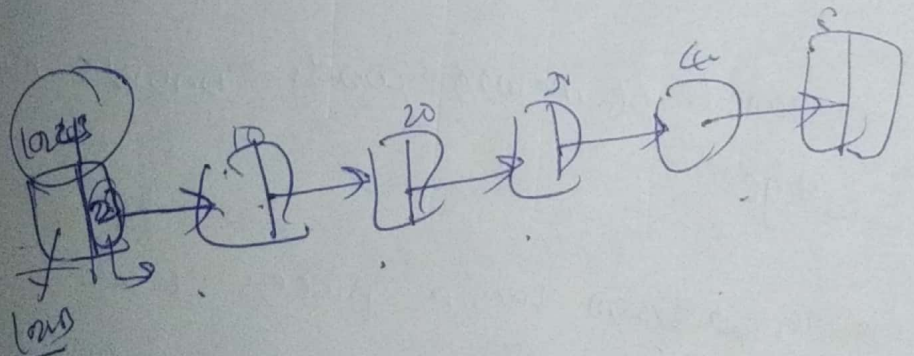
size of 6 x42 - 6KB fat 6+1



performance

* If → ✓
* Elf → X
* Increasing file size ⇒ can be, more
* type of access

⇒ sequential and Random



→ The term page traffic describes the movemet of pages in and out of memory

→ 'blocked' is not a fundamental process state

→ Which of the following approaches do not require knwoledge of the systems ?

A. deadlock detection
B. deadlock prevention
C. deadlock avoidance
D. None

→ Blocking factor of a file is

the number of logical records in one physical records

→ fragmentation :

fragments of memory words unused in a page

→ Consider a system has P process. Each process need a maximum of m resources and total of 'r' resources available. what condition must hold to ~~take~~ make the system deadlock free

$r >= p(m-1)+1$

→ In a system with n CPU what is the maximum number of process that can be in ready, run and blocked state __n__

→ A computer has 6 tape drive with n processes competing for them. Each process may need two drives. For which maximum value of n, is the system deadlock free = 5

→ In which of following, ready to execute process must be present in primary

A - multiprocessing
B - multiprogramming
C - multitasking
D - all

→ Thrashing implies excessive page I/O

3) Consider the following state

S1: The OS is designed to maximise the resource utilisation

S2: The control program manages the system programs

Ans: S1 is true and S2 is false

→ Starvation can be avoided by using first come first serve instead of SJF

→ Variable partition memory management technique with compaction results in reduction of fragments

→ 1. deadlock state is unsafe
2. unsafe state must lead to deadlock situation
3. deadlock state is a subset of unsafe state

→ Loader - performs the necessary relocation of the address and physically loads the executable code in memory

→ A system has 3 process sharing 4 resources. If each process needs a maximum of 2 units then deadlock never occur

→ n processes share m resource of the same type. The maximum need of each process doesn't exceed m and sum of all their maximum needs is always less than m. In this setup deadlock never occur

→ The main function of shared memory is Interprocess communication

→ what happens if a program address space doesnot end on a page boundary

Internal fragmentation (loss of memory)

→ A semaphore. count of negative n means (s=-n) that the queue contains n waiting process

---

Important points

→ Response time are more predictble in preemptive systems than in non preemptive systems

→ Realtime systems generally use preemptive cpu scheduling

* The LRU algorithm

1. pages out pages that have been used recently
2. pages out pages that have not seen recently used
3. pages at pages that have been atleat used recently

→ The function performed by paging software are management of the physical address

---

→ A Linker creates a Load module