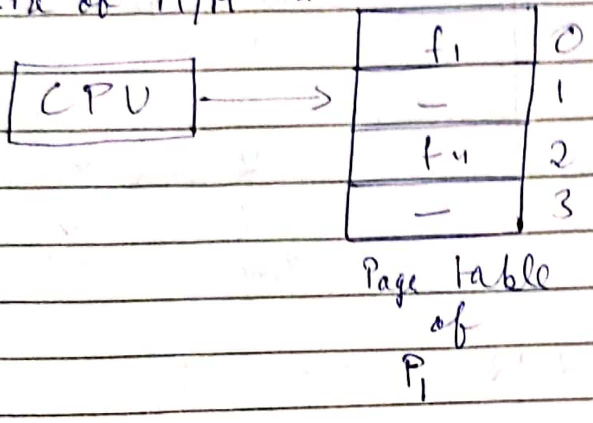
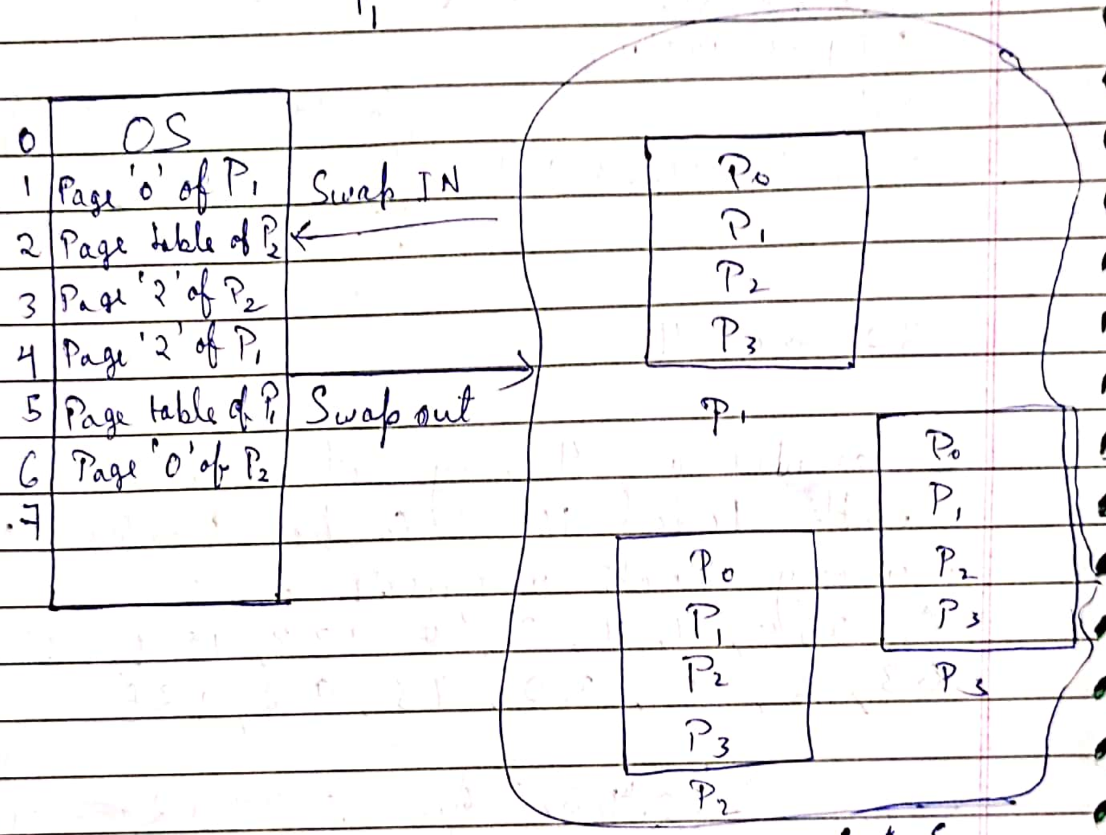
Date / /
Page

Virtual Memory :- It provides a illusion to the programmer that a process ~~CRUX~~ whose size is larger than the size of M/M can also be execute



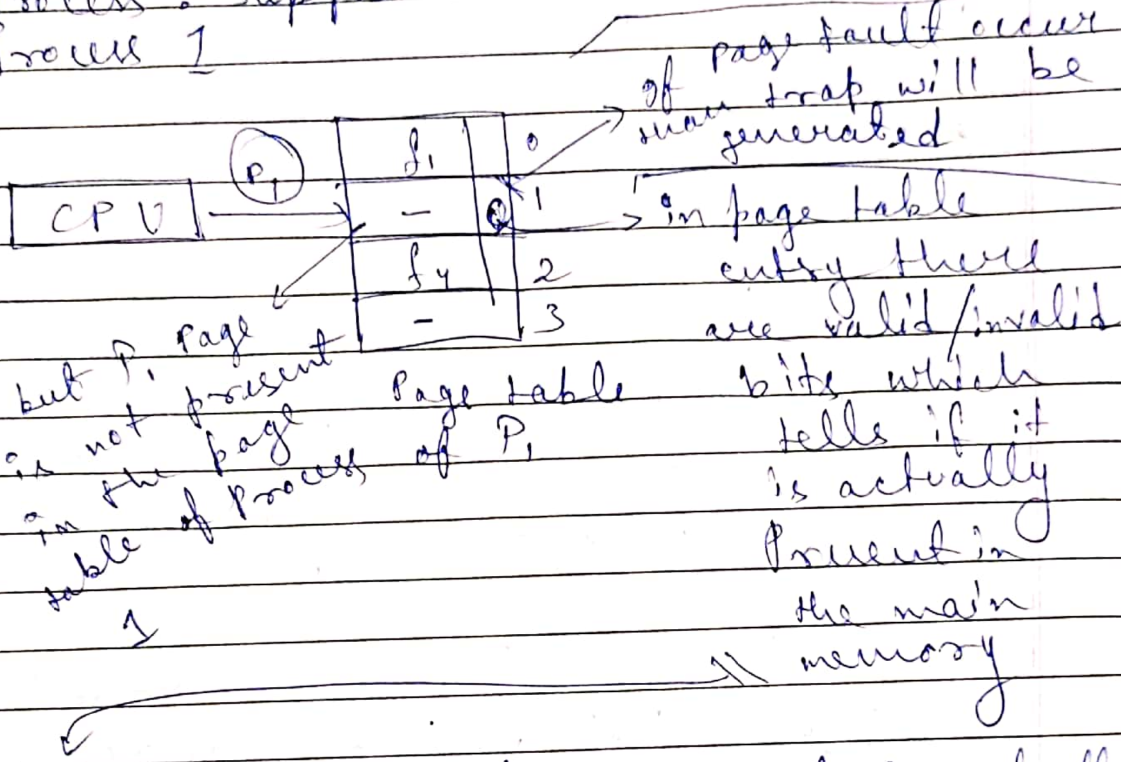


M/M is finite

LAS
Logical address space
hard disk
there are multiple processes which we have divided into Pages

Date / /
Page

Virtual memory :- CPU is executing P1 process. Suppose it wants the P_1 of Process 1


CPU
P_1
f_1
–
f_4
–
0
1
2
3
Page table
if page fault occur then trap will be generated
in page table entry there are valid/invalid bits which tells if it is actually present in the main memory
but P_1 page is not present in the page table of process 1

If the page that we want is actually absent in the M/M than it is called Page fault
~~if it is present than it~~
☆☆ when trap is generated, ~~than~~ control from user is passed to OS
It is called Context switching
↓
now ~~fist~~ first
→ OS will check authentication
→ OS will check in LAS where the page that CPU is searching for is present and from there it will bring it to an empty space of M/M & the address i.e the frame number we will update that in the page table & then we will give the control back to user

Date / /
Page

· Effective Mem. access time

$$EMAT = P\begin{pmatrix}\text{Page fault}\\ \text{service time}\end{pmatrix} + (1-P)\begin{pmatrix}\text{main}\\ \text{memory}\\ \text{access}\\ \text{time}\end{pmatrix}$$

'P' → Probability of Page fault occur

Main memory is very fast
Hard disk is very slow

$$EMAT = P\begin{pmatrix}\text{Page fault}\\ \text{service time}\end{pmatrix} + (1-P)\begin{pmatrix}\text{main mem.}\\ \text{access time}\end{pmatrix}$$

m sec   n sec

$+\begin{pmatrix}\text{main memory}\\ \text{access time}\end{pmatrix}$ n sec

∵ n sec << m sec
so we ignore it here

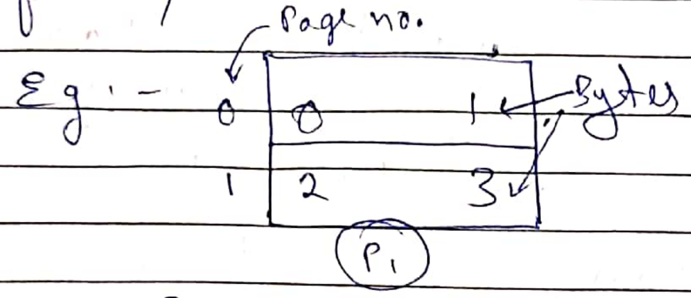
If $P\Uparrow$ performance $\Downarrow$

Date / /
Page

☆ There is no limitation of size and ~~proces~~ number of processes

Paging :- we have a process we divide the process in equal size pages & ~~than~~ ~~we~~ put ~~those~~ ~~in~~ them in frames of M/M

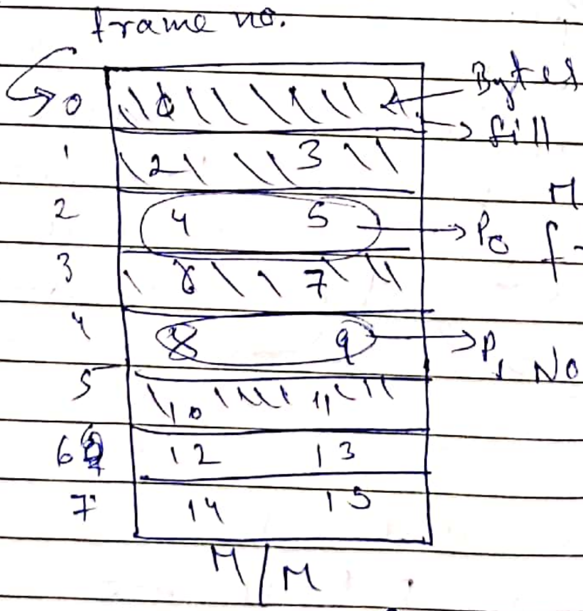
Eg :-



Process size = 4B
Page size = 2B

No. of Pages / Process = $\frac{4B}{2B}$ = $\boxed{2}$

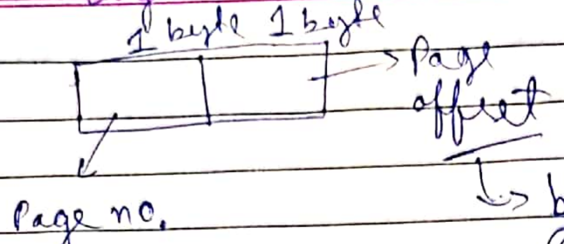
frame size & page size should always be same



M/M size = 16B
frame size = 2B
No. of frames = $\frac{16B}{2B}$
= 8 frames

CPU will ~~generat~~ generate logical address

Logical address

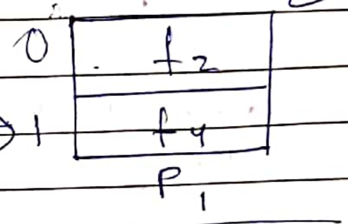


↳ because page size is 2 & to represent 2 number we need 1 ~~byte~~ bit

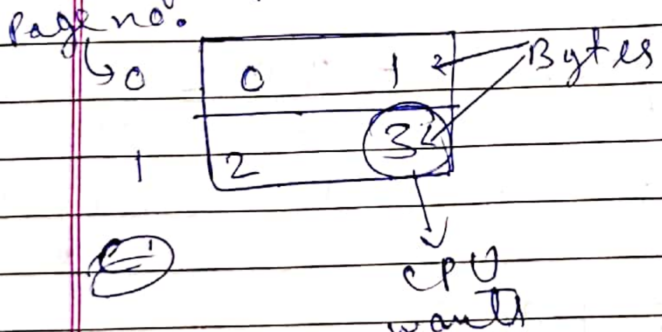
There are 2 pages & so represent 2 pages we need 1 ~~byte~~ bit

we have a page table for every page

Page table



If CPU wants the page no. 3 of process 1

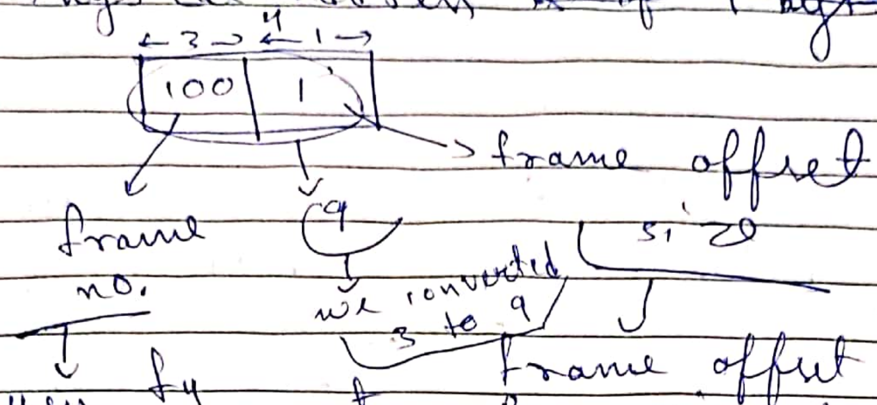


& this 3 is in frame 4, 9 so number we need to 9 so we convert this 3

now ~~we~~ lets say the P_0 of Process 1 is in frame 2 & ~~Process~~ P_1 of Process 1 is in frame 4

* logical address that the CPU will generate is 11 i.e 3 in binary is 11 &
↳ page no. ↳ Page size

Physical address ~~of 4 byte~~



then f4 i.e 4 we can represent it in 100

in PA frame offset size & Page size in LA will always be same so 1 will be in frame offset here

Physical address is directly related with M/M ~~to~~ main memory. Physical address ~~is~~ represent where the actual byte is present. ~~& P.A~~

To represent P.A we need how many Bit that depends on the size of M/M. here size of main memory is given 16 ∴ To represent $16 \to 2^{4}$ we need 4 bits

* Total number of frames in M/M is $8 \to 2^{3} \to$ we need 3 bits

* CPU will give us a byte no. we need to convert the byte to L.A & then convert it to P.A & take the Byte from the main memory

Segmentation :- we divide a process into segments and than we keep those in the main memory.

Difference betn Segmentation & paging

Paging :- without knowing whatever is there in the ~~program~~ process, it will divide the process into ~~equal pages~~ and will put it in the M/M

fixed size pages

☆ Pages are always of same size.

Segmentation :- Segmentation won't divide the Program into equal ~~Pag~~ segments rather it will ~~A into~~ divide it into different segment for eg:- we have a code, segmentation will divide it into main, add e.t.c

☆ Segments can be of various size.

☆ we put the segments in the main memory after dividing the process

CPU will generate a logical address & with LA we will access which segment does CPU wants & till where it wants

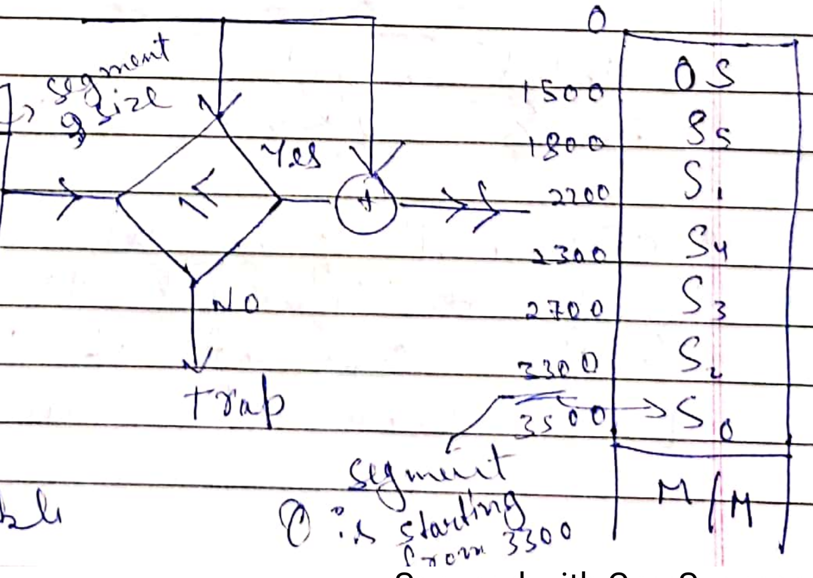
Base address from which the segment will start

	BA	Size
0	3300	200
1	1800	400
2	2700	600
3	2300	400
4	2200	100
5	1500	300

segment size

segment no.

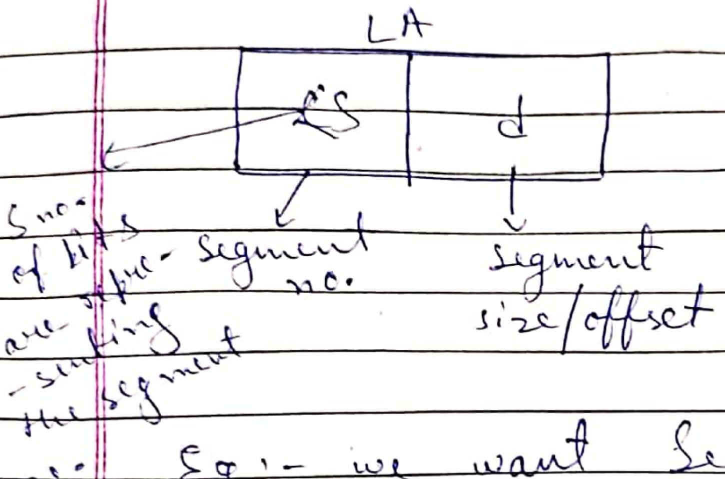
Segment table



MMU (memory management unit)

$LA \xrightarrow{MMU} PA$

CPU →



d ≤ size of the segment

d means how much does the CPU wants to access from the segment

Eg:- we want Segment 1. ~~we will~~ In the Segment table it's base address is 1800 & size is 400

If d > size than ~~it will be~~ error because segment 1 is from 1800 - 2200

for eg:- If s no. is S_3 & d is 200
2300 + 200 = 2500
So it will read the data from 2300 - 2500 & will give that to CPU.