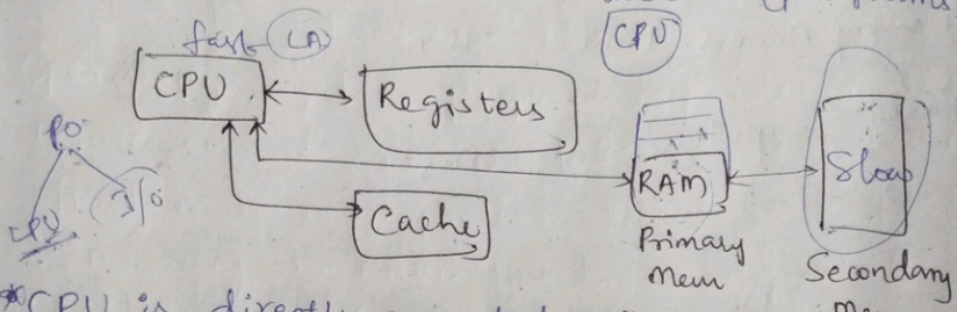
Memory Management - Method of Managing.

Memory resources - RAM, Registers, ROM, Cache, Hard Disk.

Efficient Utilization of Memory is our goal.

CPU executes the set of instructions (programs)



*CPU is directly connected with RAM, Registers & Cache but not with Secondary Memory because, Secondary Memory is very slow. CPU is faster

* If we increase the size of RAM/Reg/Cache, the cost of Systems ↑.

* All the programs (process) to be execute are present in Secondary Memory. CPU should execute all these, So, for this, the processes present in Secondary Memory should "bring" on to the RAM So that CPU can execute them, bcoz it is directly Connected to RAM/--.

- The size of the Secondary Memory is very large we can store as many process as we require.
- Y only RAM? Y not we use Reg/Cache to place processes brought from Secondary Memory, bcoz Reg/Cache's size are very small that in KB's. although they are very fast.
- Degree of Multiprogramming is amount of processes swapped from Secondary Memory to RAM.

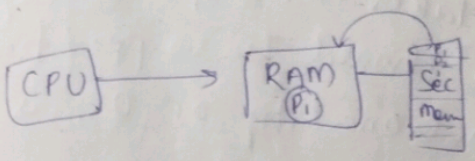
Multiprogramming is multiple processes are swapped on to RAM at a time, So, that the

CPU can processes (executes) the programs one by one

- As the degree of MProgramming is high, the utilization of CPU es also high.

Process to be executed on → CPU, I/O devices

if a program has a need of Input/oput devices like printer or scanner to be scanned, then the CPU frees it to use them. then the CPU is idle. This is incase of 1 process in RAM. everytime



- As we place more no.of process in RAM, So that the System efficiency will increases.

Example: RAM 4MB, Proc 4MB. $= \frac{4}{4} = 1$

1 process can be placed in RAM.

Let 'K' amount of times a process use I/O devices.

CPU = 1 − K.

Let K = 70% = I/O devices

CPU = 30%.

If i increase size of RAM. to 8 MB, then 2 process are accomadated in RAM. So,

I/O device utilization → 'K' units for 1 process.

then for 2 processes → 'K^2'.

CPU = $1 - K^2$

Let K = 70%.

$1 - (0.7)^2 \cong 76\%$.

- As the size of RAM is increased, then the utilization of CPU also increases. & efficiency of System ↑.

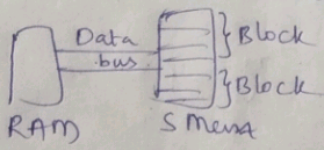
RAM 16 MB Proc 4 MB.

4 process in RAM.

'K' — 70% → I/O devices

CPU — $1-K^4 = 1-(0.7)^4 \approx 93\%$

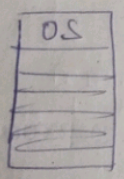
- Operating System plays a ~~Key~~ role in Memory management. i.e. OS takes the responsibility that more no. of procss should be swapped from Smemory to RAM. functionality of OS is allocation and deallocation & security (interference of instructions) is look after by OS.



- Dividing procss in Smemory into Blocks is seen in Paging & Segmentation.

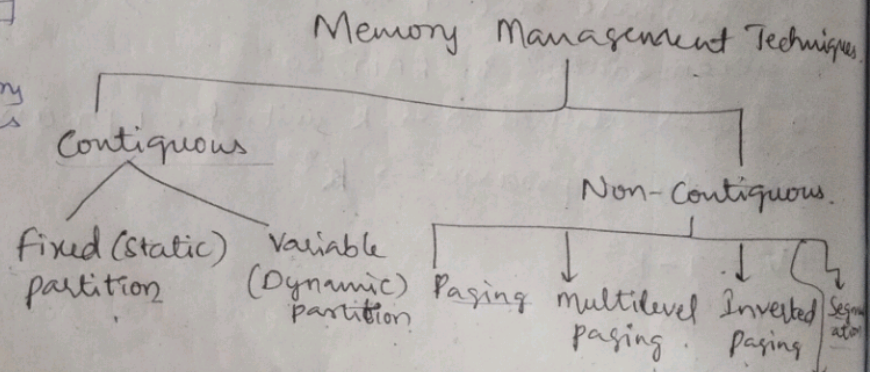
- Memory Management Techniques:

OS uses various techniques to manage Primary Memory (RAM).



Proc

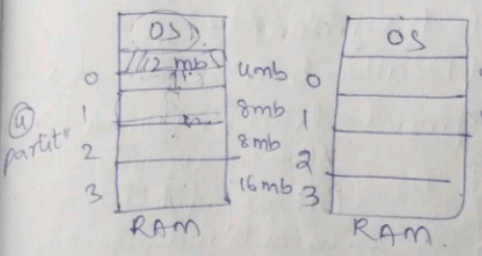
Initially some amount of Block is occupied by OS & remaining by procss



- Whenever we start our System, firstofall, the Op.System is mount in RAM.

Fixed Partitioning:

- No. of partitions are fixed during the configuration of RAM only.
- Size of each partition may or may not be same.
- Contiguous allocation, so Spanning is not allowed.
  - ↳ 1 whole process can be placed in 1 partition.

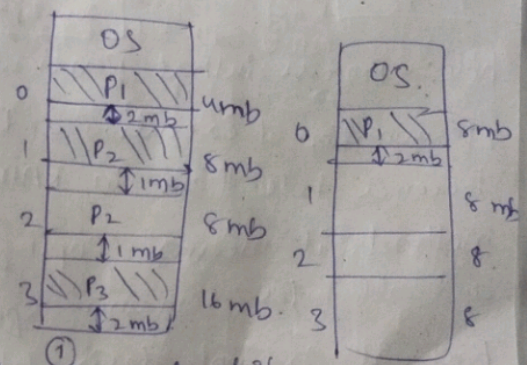


① partitions

if a process is 12 mb.
8 mb in one partition
4 mb in other partition
is not possible.

Problems of Using Contiguous allocation (Disadvantages)

1) Internal fragmentation
2) Limit in process size
3) Limitation on Degree of Multiprogramming
4) External fragmentation

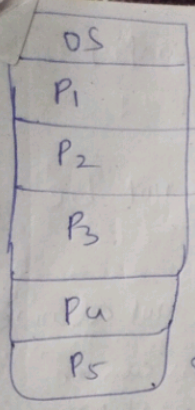


- Internal fragmentation is occured while a process of size < the size of partition in RAM. Another process can't be accomodated in the remaining/leftover space in that particular partition of RAM.
- External fragmentation is occured while, we have space in RAM, but can't accomodate more process than no. of partitions having in RAM.

Ex:- in ①, we have 6mb left over. upcoming process of size 5 mb, can't be accomodated, bcoz of contiguous allocation.

Dynamic partitioning: (Variable)

Whenever the process comming into RAM, then only we are allocating the space in RAM during runtime according to size of processes.



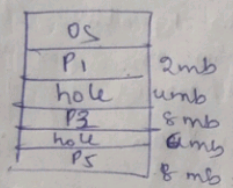
Advantages of Dynamic Partit:

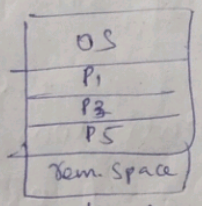
1. No internal fragmentation
2. No. Limitation on no. of process
3) No. Limitation of process size

Disadvantage:-

1). External fragmentation.
2) Allocation & Deallocation is complex

Solution for External fragmentation is Compaction





Allocation & Deallocation

Bit map & LList are used to manage where is hole & where is occupied

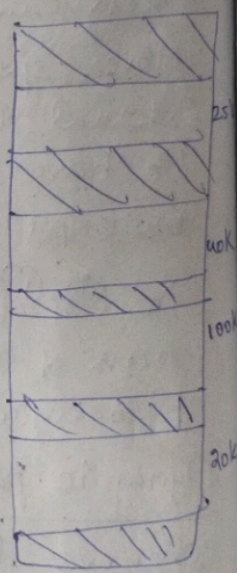
hole:- Process in that particular partit" either completed /termination of execution or Blocked whatever, the it becomes hole their in RAM. Then by using Compaction method, it means coping all existing Process to one location and all holes (remaining space) to one location

But, it is not desirable. bcoz copying from one address to other address takes much time.

mm. Strategies:-

First fit:- it is convient to use, simple & fast.
Allocate the first hole that is big enough.

Next fit: updated version of first fit.
Same as first fit but, don't check for hole from starting everytime, it maint-ains a pointer, by this, search from last allocated hole.

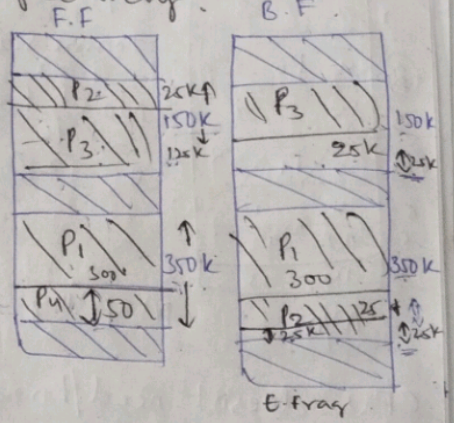


Best fit: Allocate in such a way that the internal fragmentation is very low & slow.

Worst fit: Allocate in such a way, internal fragmentation is high.

Nov 15, 2023, 16:02

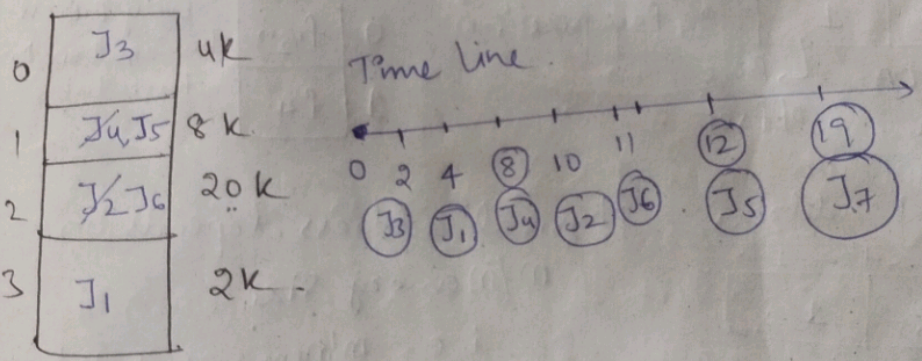
Gate Question: Requests from processes are
P_1 P_2 P_3 P_4
300KB, 25KB, 125KB, 50KB respectively.
The above req. Satisfy with
1) Best fit but not first fit.
2) First fit but not best fit
3) Both 4) None.



Use Best Fit.

Req. no.	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Req. size	2k	14K	3	6	6	10	7	20
Usage time	4	10	2	8	4	1	8	6

Calculate the time at which "J_7" will be Completed.
a) 17 b) 19 c) 20 d) 37

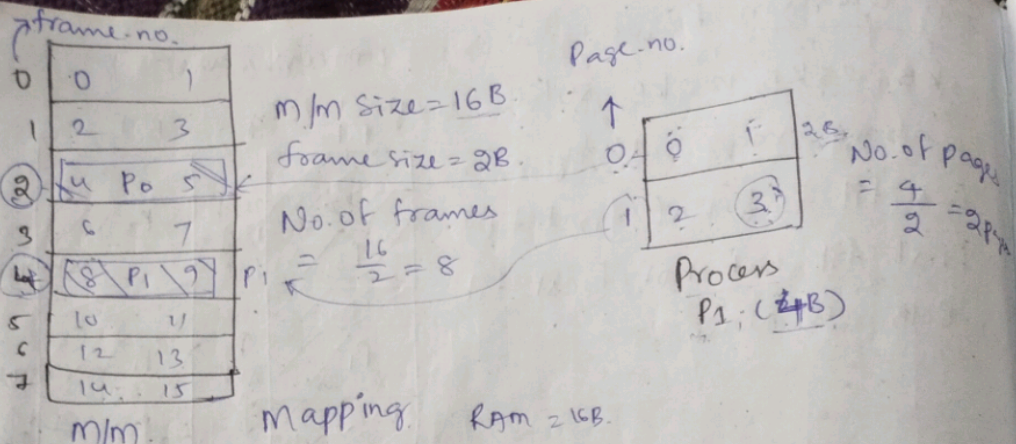


Non-Contiguous Mem Allocation:

- External fragmentation is removed by this.

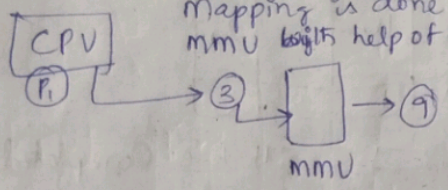
1. **Paging**: Before swapping the processes from S mem to RAM/main memory, each processes are divided into pages and RAM into frame.

> Page size = Frame size



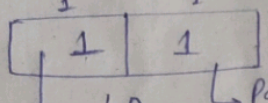
Mapping RAM = 16B.

CPU doesn't need/know about Page Table or anything of this. It just generates the address of that particular required byte of that Running Process.



Mapping is done by MMU with help of P.Table. CPU generates logical address & that logical address converted to physical address.

Logical address



Logical address depends on Page size i.e (2B)

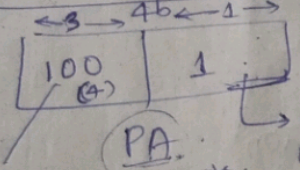
Page Table of P_1.

0	f_{rm}
1	f4

Every process having their own Page table.

Physical address :- Physical address depends on m/memory size.

i.e. m/mem size is 16 byte so, 4 bit



from LA, get pg.no. i.e. 1 & from P.Table by using pg.no get fr.no. i.e 4. Convert into binary & put in PA of fr.no. & now, from LA get Pg.offset & place it in fr.offset Then the whole four bits represents some number i.e. 9.(1001). Then go to that 9 position in m/m & get data to CPU.

Nov 15, 2023, 16:03

Logical & Physical address:

Memory is byte addressable

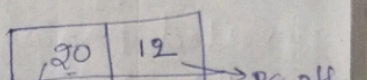
$2^{10} = 1K$
$2^{20} = 1M$
$2^{30} = 1G$
$2^{40} = 1T$

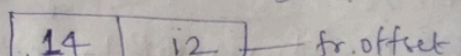
Ex:-

Logical address Space = 4 GB = 5 Memory.
PAS = 64 MB = $2^6 \times 2^{20} = 2^{26}$ = RAM
Page Size = 4 KB = $2^2 \times 2^{10} = 2^{12}$

- No. of pages = 2^{20}
- No. of frames = 2^{14}
- No. of entries in P.Table = 2^{20}
- Size of P.Table = $2^{20} \times 14$

$LA = 2^2 \times 2^{30} = 2^{32}$





No. of entries in PT = No. of Pages in a Process.

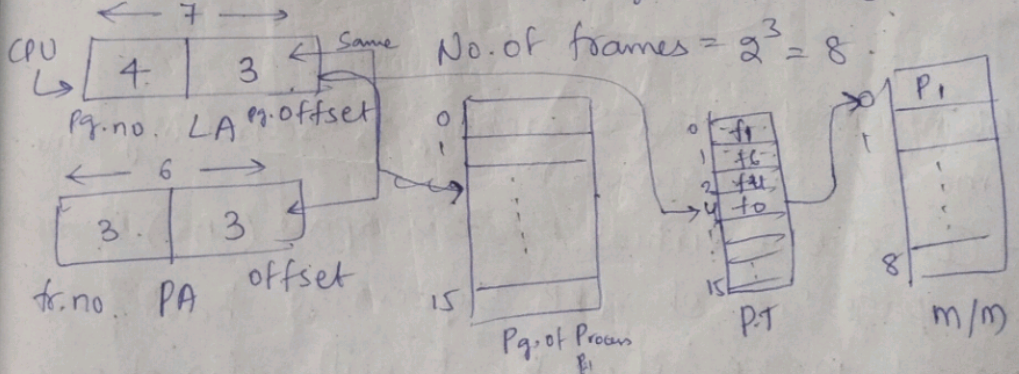
PA = 26 bits uses to represent
we require 14 bits to represent frame no's.

Example:- related to Paging.

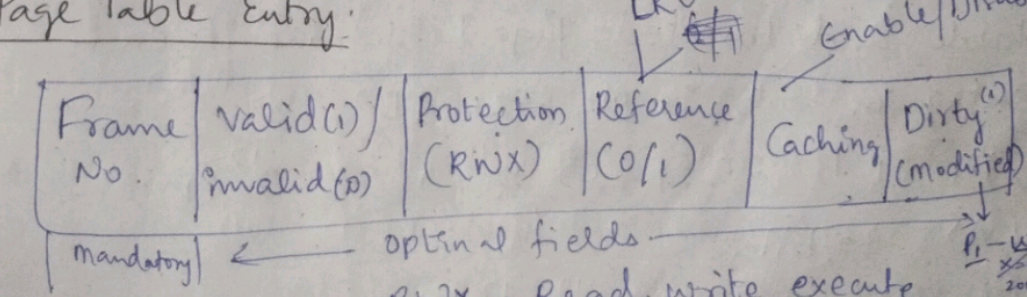
Consider a system which has LA = 7 bits, PA = 6 bits
Page size = 8 words/byte, then calculate no. of pages & no. of frames

1 word = 1 byte.

No. of pages = $2^4 = 16$
No. of frames = $2^3 = 8$



Page Table Entry:



RWX - Read, write, execute permissions on data of Page.

Caching - mostly/repeatedly used thing is kept cache for CPU. ease

Nov 15, 2023, 16:03

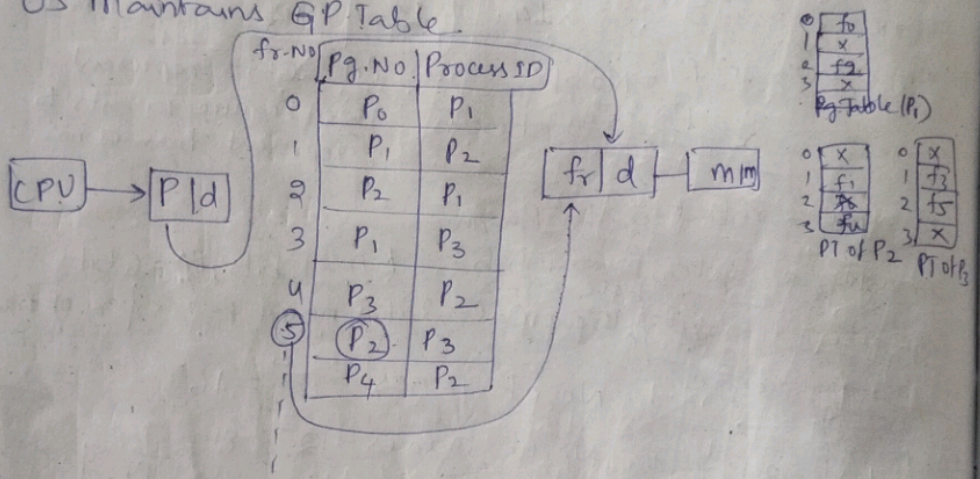
Page Table will be in main Memory.

~~2-level paging~~: Inverted Paging.

we know that m/m is divided into frames of equal size. Page Table of Every process is mounted in Main Memory. If I have more no. of process, then most of frames of m/m are occupied the Page Tables. So, in order to avoid the wastage of space, inverted paging is introduced.

In this, instead of creating Page Tables for every process, create only one Page Table. i.e Global P.T.

OS maintains GP Table.



Problem with Inverted Searching is searching time is more (Linear search).

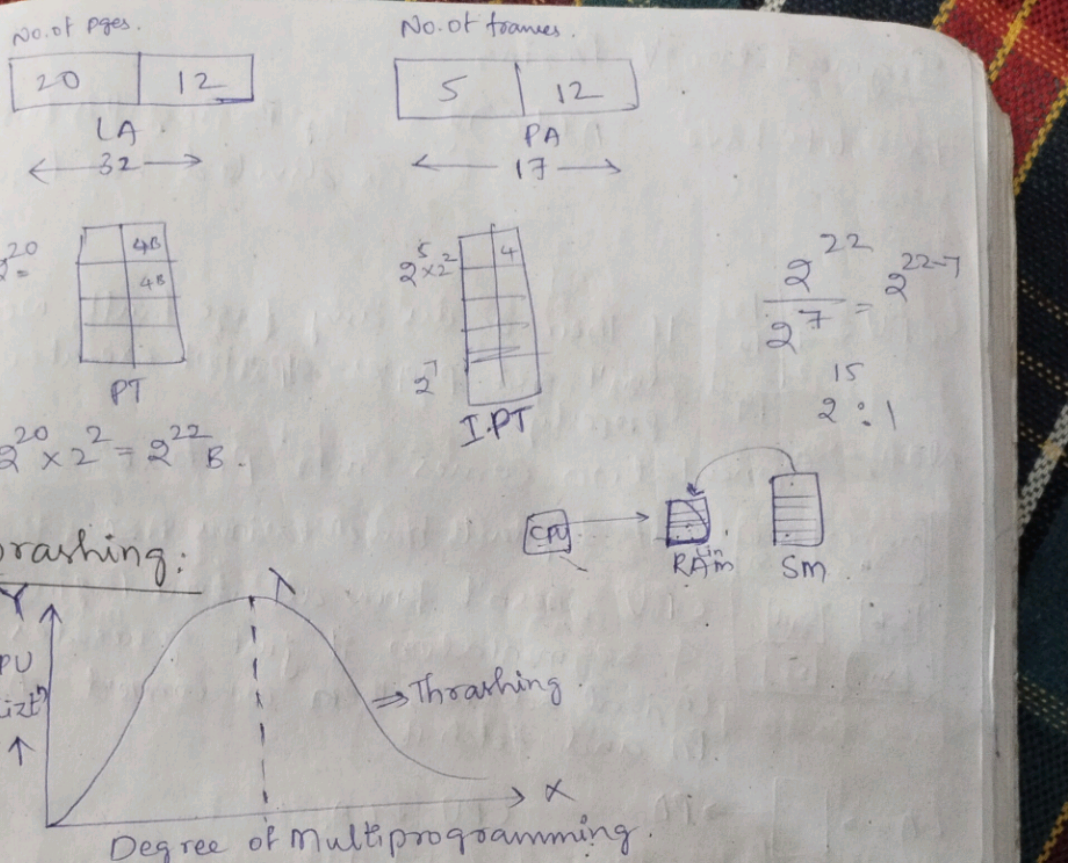
Q:- Consider a Virtual Address Space (LAS) of 32 bit and page size of 4KB System is having a RAM of 128KB. Then what will ratio of page table and Inverted Page Table size if each entry in both is of size 4B?

a) $2^{15}:1$ b) $2^{20}:1$ c) $2^{10}:1$ d) $2^{12}:1$

LAS = 32 bits

Pg size = 4KB = $2^2 \times 2^{10} = 2^{12}$ B

RAM (PAS) = 128KB = $2^7 \times 2^{10} = 2^{17}$ B.



Thrashing:

- As the Degree of Multiprogramming is very high, utilization of CPU is also high. It means, more no of processes are kept in Ready state for running. This is until a certain limit of Multiprogramming becoz RAM is limited. we can't place all the processes on RAM.

Thrashing is due to less Page hit & More Page fault, then it takes much time to fetch data from ~~Secondary~~ Memory.

Page fault is page not found in Page Table.

System's time is wasted in Serving Page fault that, bringing Pages from Harddisk to RAM/main.

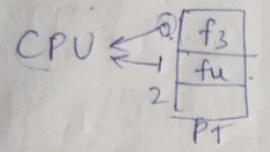
Then CPU performances falls down.

Thrashing is removed by

1) Increase size of Main Memory.
2) Long term Schedular := bringing more no. of Process into Ready State to RAM
Controlling DoM

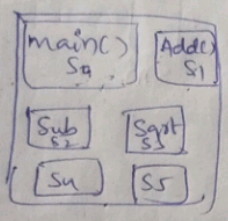
Segmentation Vs Paging

Ex:- Lets have Add() { ... } By Paging this funct is divided into Pages



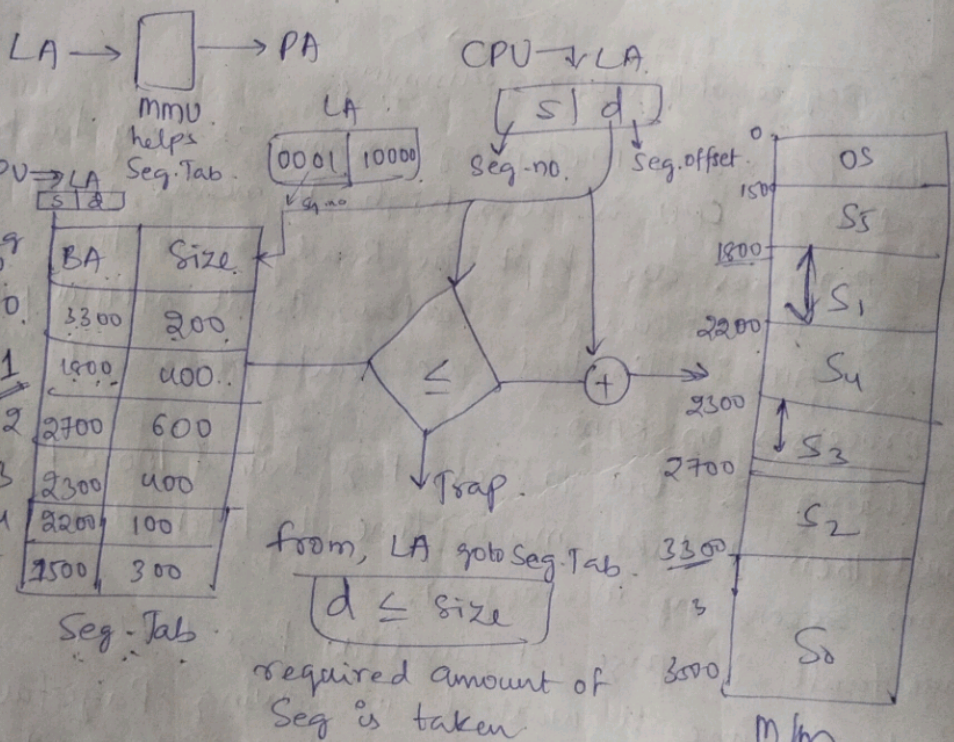
If Heir occurs any page fault, we don't get proper o/p, or not executes Properly

Now, Segmentation comes into picture



dif. segments have various size.

CPU doesn't know all about Paging & Segmentation. it just generates Logical address, then we convert LA to PA and fetch it.



Overlay: By using this method, we accomodate Process of P.size > M/M size in mm.

This method is used in Embedded Systems.

The process is partitioned and brought into the RAM, & after completion of execution, just

remove it from m/m and place the next partition thus, one by one is executed.
partitions should be independent.

Assembler ⟹ Convert Source code to Object code

Ex:- Consider a two pass assembler
Pass 1: 80 KB. Pass 2: 90 KB.
Symbol table: 30 KB. Common routine: 20 KB.
At a time only one pass is in use.
wt is min partition size required if overlay driver is 10 KB size.

Sol:
Pass 1: 80 + 30 + 20 + 10 = 140 KB
P2: 90 + 30 + 20 + 10 = 150 KB.

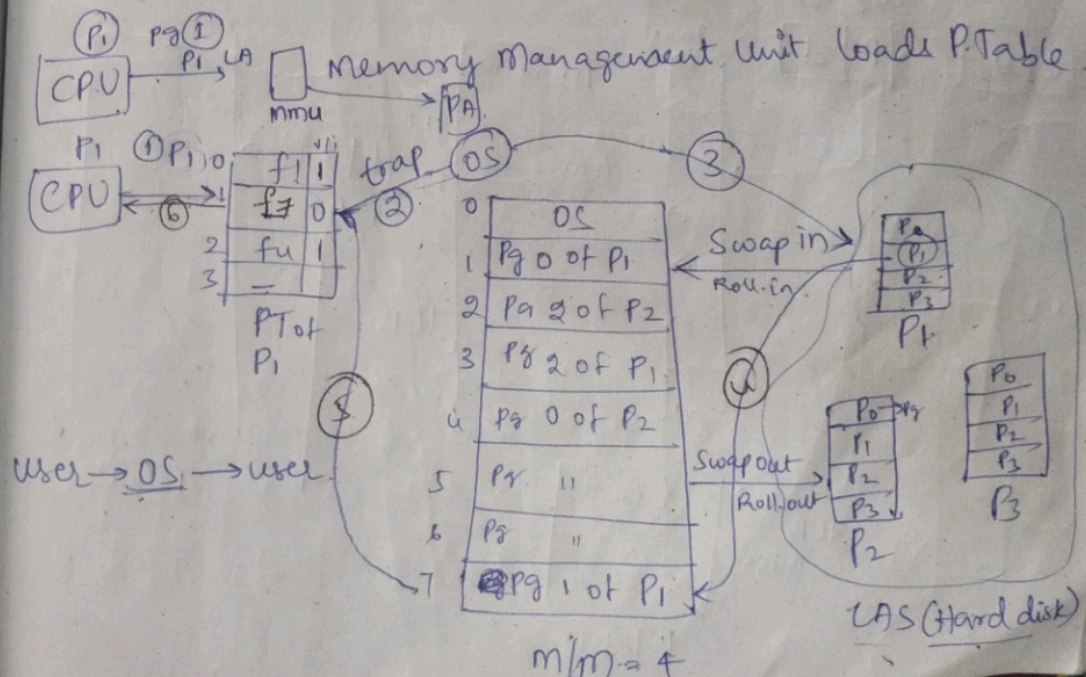
m/m size should be equal's to the size of pass having higher KB/size

Virtual Memory: P.Size > RAM

LAS - Size of Hard disk

Process size > RAM

All the Systems or PC's we are using are works based on Virtual Memory.



trap → interupt

when interupt occurs, then system shifts from User control (mode) to OS (Kernel mode) and then to User control.

Effective Memory Access Time :-

P - probability of occuring Page fault.

(1-P) = " " " page hit.

EMAT = P(PFST) + (1-P)(Memory Access time)

PFST ⇓ milliseconds + mm (ma).

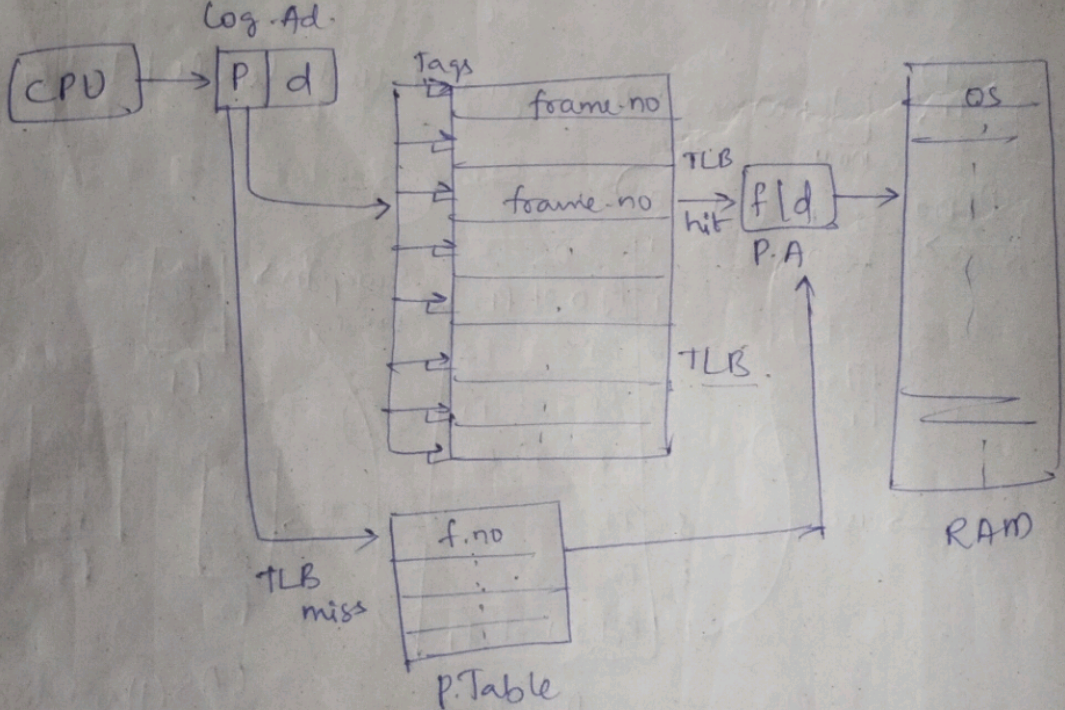
Memory Access time ⇓ nano seconds (ma).

Page fault ↑, Thrashing ↑.

Translation Lookaside Buffer (TLB)

We use faster Memory i.e. Cache it is also known 'Buffer'.

TLB purpose is to reduce the time consumption when there are more no. of processes or Page Table entries which are divided into blocks.



Time required to access RAM or m/m is 'x'
Time required to access Page Table is also 'x' units becoz, PTable also present in RAM.

CPU → PT (x) → frame → RAM (x units)

∴ it takes '2x' units of time

EMAT = Hit (TLB + x) + Miss (TLB + x + x)

TLB is fast as compared to RAM.

Question:- A Paging Scheme using TLB. TLB Access time 10 ns & main memory access time takes 50 ns. Wt is effective memory access time (in ns) if TLB hit ratio is 90% and there is no page fault.

Sol:-

EMAT = Hit (TLB + mm) + Miss (mm + TLB + PT(mm))

= 90% (10ns + 50ns) + 10% (50ns + 50ns + 10ns)

= 90% (60ns) + 10% (110ns)

= 0.9 (60ns) + 0.1 (110ns)

= 54 + 11 = 65 ns.