If the process is taking 'p' time in doing I/O then,

CPU Utilization = $(1-p)$

If $p = 80\%$

then,

CPU Utili. = 20% only

So, it is a big problem.

If we increase main m/m size to 16 MB then we can put 4 procs.

Say each process take 'p' time to take I/O.

then probability of all process doing I/O → p^4

(η) CPU Utilization = $1 - p^4$

if $p = 80\%$

then $(\eta) = 60\%$

If main memory size → m MB

Process size → n MB

then, $\eta = 1 - (p)^{m/n}$

if $m = 32$ MB $n = 4$ MB

$\eta = 83\%$

If $m = 48$ MB & $n = 4$ MB

$\eta = 93\%$

$\left(\frac{m}{n}\right)$ → total no. of process in main m/m

$\eta = 1 - p^k$

k → degree of multiprogramming

$\lim_{k \to \infty} (1 - p^\infty) = 1 \quad [\because p < 1]$

$= 100\%$

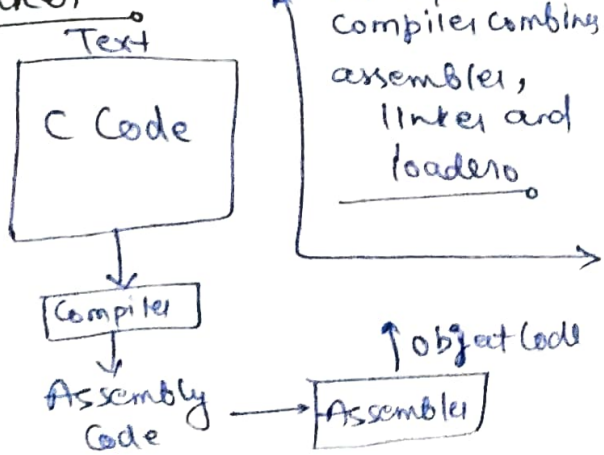
no. of process can get inside main memory at a time called as degree of multiprogramming.

→ we can't get 100% bcoz it will possible if we have $m \to \infty$ and it is not possible. So we take definite size of 'm' and try to maintain all process in the main m/m using m/m management techniques.

→ Memory manager take responsibility for allocation and dellocation of m/m along with security.

→ Program under execution called process. Process resides in RAM but program in ROM & Hard Disk.

Object Code, Relocation and Linker



Object Code is present in hard disk.

Object Code (contains)

- Header
- text segment / Code segment
- Data segment
- Relocation information
- Symbol table
- Debugging info.

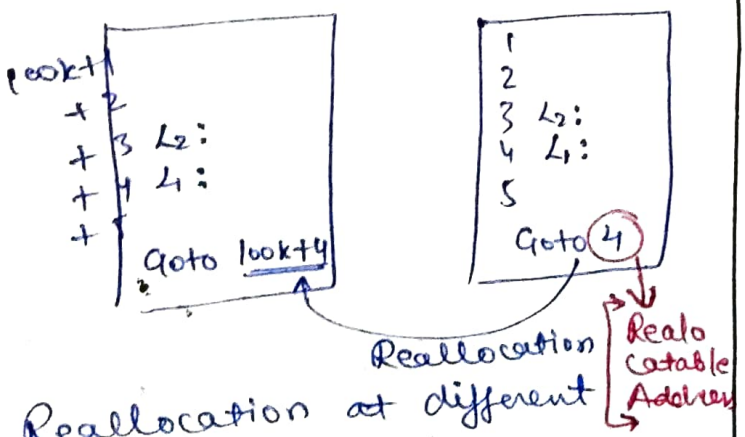
Header performs the role of index to point the starting of all other parts of object code.

→ **Relocation info.**

Assume a program

```
1.
2.
3. L2:
4. L1:
5.
   Goto 4
```

But if we allocate that program in memory it might be possible that starting address in main memory is not from 1. Assume address start from 100



Reallocation at different address.

100+4 → **Absolute Address** (Final address)

We can't provide absolute add. to process at compile time bcoz it might be possible that another process should already running at that address. So providing absolute address is difficult that is why we provide reallocatable address.

Relocation info contains details of parts of program contain reallocatable address.

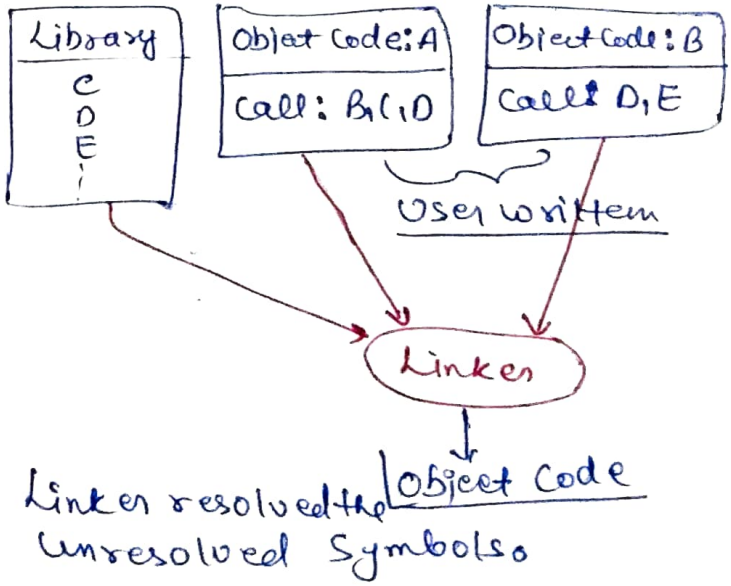
→ Symbol table contains info. of symbol and its address defines in program

Some symbols are unresolved as printf() and scanf() bcoz its definition not given in program but we use it.

→ To resolve this symbolds we use linker.

→ gdb → debugger

Ex. onlinegdb.in

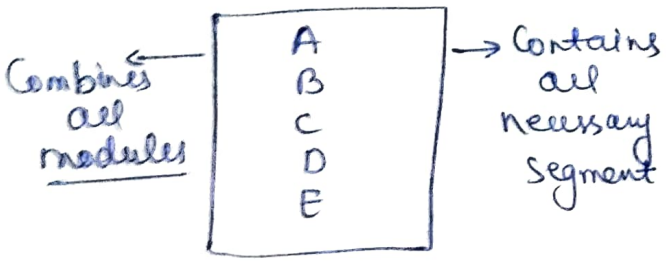


Linker resolved the unresolved Symbols.

Linker tries to find the unresolved in directories of System.

→ Ex: C directory (Local Disk C) Contains all unresolved symbols in windows.

Object Code (Linker) contains.



→ Linker also uses relocatable address.

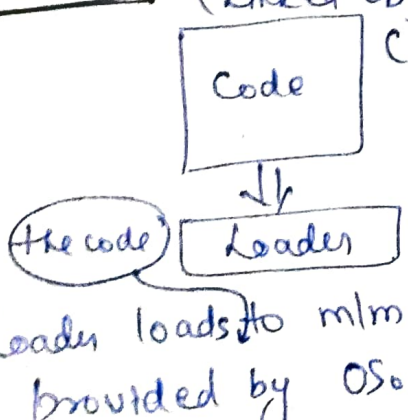
→ Dynamic linking used for shared libraries, but it might lead to page fault if program is not present at that location.

Responsibility of Linker

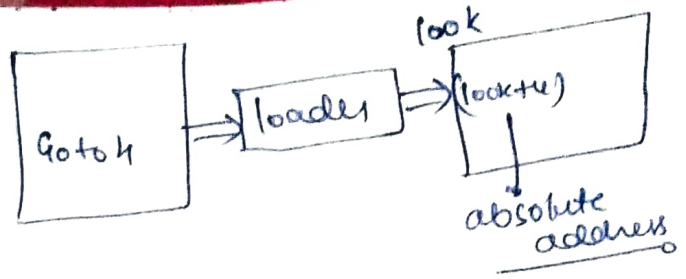
1. Relocation
2. Symbol Resolution of unresolved symbol.

Loader

(Linked Object Code) (Relocatable)



Loader loads to m/m on address provided by OS.



1. Program loading → Loader
2. Relocation. → Linker
3. Symbol Resolution. → Linker

Compile time: At compile time relocation occur as (Symbol names → Relocatable add.)

Link time: At link time (Relocatable code of various module → Relocatable add of Combine module)

Load time: (Relocatable Add. of Combined Module → Absolute Add.)

Run time: Dynamic linking occurs at run time. It requires OS support.

→ Static libraries doesn't require OS support.

Gate as Question on Linker

Q A linker is given object modules for a set of programs that were compiled separately. What info. 'need not' be included in object code.

Ans-4

1. Object Code
2. Relocation code
3. Name and location of all external symbols defined in object module
4. Absolute address of symbols

(98) In a resident OS computer, which of the following software must reside in main memory under all situations,

① Assembler ② Linker
③ Loader ④ Compiler

Ans- Loader because if loads everything (assembler, linker, process, compiler) in memory if needed.
If loader will not not inside everything will be halt.

2001 The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned _____.

① Assembly ② Parsing
③ Relocation ④ Symbol Resolution

→ load address also called absolute address comes under consideration after loading.

2004 Consider a program P - - -

Ans- Link time.

2002 Dynamic linking can cause security concern because _____.

Ans → The path for searching dynamic libraries is not known till run time.

On dynamic linking security issues arises if a process tries to access the resource of another process.

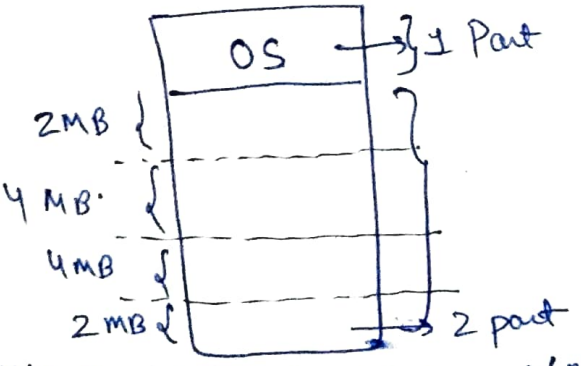
Gate-2003
Which of the following is not an advantage of using shared, dynamically linked libraries as opposed to using statically linked libraries? Ans-2

① Smaller size of executable code.
② lesser overall page fault rate in system.
③ Faster program startup
④ existing program needs not be re-linked to take advantage of newer version of libraries.

Note → If there is any doubt in linker and loader then watch the video of (object code, Realocation and linker) after 20:00 mins.

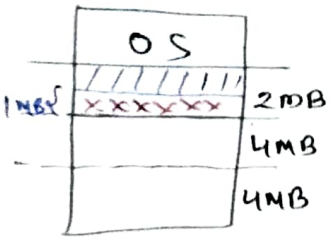
Fixed Partitioning
→ Allows lots of process in main m/m.

Contiguous m/m alloc.



M/m devided into 4 equal/not equal parts, each process is allow to use any part in a contiguous manner also called fixed partitioning.

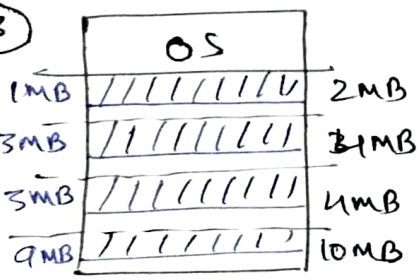
Disadvantage
→ Internal fragmentation



Assume process size is 1 MB and it found first allocatable as 2 MB then process get inside the m/m and 1 MB of space get wasted Called <u>internal fragmentation.</u>

(2) Process with size more than 4 MB can't run in m/m (Limitation on process size)

(3)



Here we have net 4 MB vacant space in all the slots Combinigly but we can't provide space to any process because it needs contiguous space. This problem called as '<u>external fragmentation</u>'.

→ External and internal fragmentation seems similar but they are not.

(4) Degree of multiprogramming is limited or fixed.

<u>Note</u> → Fixed partitioning was used long ago. So its very rare chance of coming in Gate exam. But still we have to prepare it.

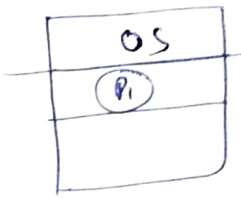
Relocation and Protection in Contiguous m/m allocation

→ If a process load in main m/m then relocation is done by loader. But it is a software approach, and problem with this approach is —

Assume process P_1

```
Goto 10
```

and when loader loads it to main m/m it will get absolute addrs

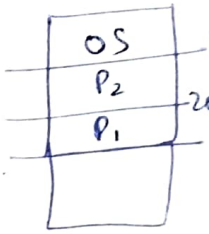


here Goto 10+100k

↓ Absolute address

If P_1 gets swap out from m/m for some purpose and P_2 comes inside after P_1 on its place.

So, if P_1 comes back then scenario will be like



Now, it should be Goto 10+200k but loader doesn't know about its initial value hence it will treat as :

Goto <u>10+100k+200k</u>

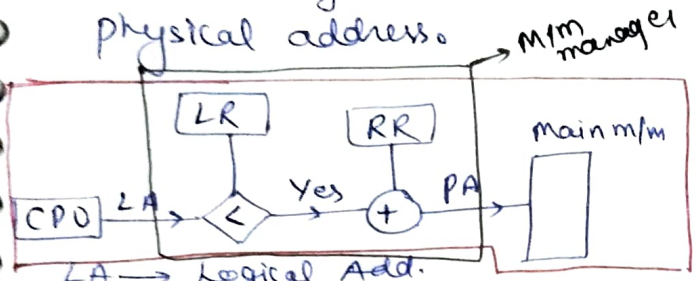
→ So, it is a problem at load time binding.

→ That is why OS prefers run time binding and it introduces the concept of logical & physical addresses.

In case of swap in & out using loader for converting relocatable to absolute add. wasn't a good idea.

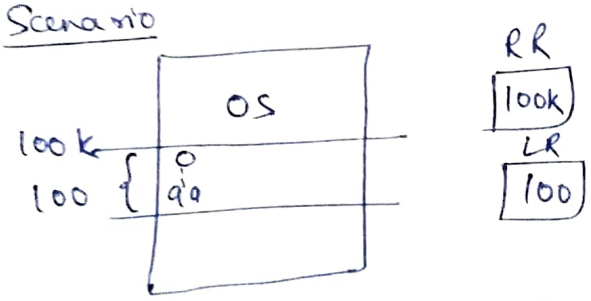
→ Address generated by CPU called as logical started with '0' for each process.

→ Memory manager use to convert logical address to physical address.



LA → Logical Add.
PA → Physical Add.
LR → Limit register
RR → Relocation register.

Scenario



CPU address starts with 0 and should be inside the range of LR. (0-99) 100 Units.

→ CPU sends LA as 20, goes to < verifies that 20<100.

If true then it will + add the value of RR to 20 as 100k+20 and get the value at this physical address from main m/m and return it to CPU

→ If value (LA > LR) then, LR will trap it.

→ LR binds the process in alloted space.

→ If process swaps out and swaps in only value at RR should be changed by OS

→ RR and LR value under the control of OS not in control of process.

→ This is hardware approach of m/m management, in contiguous manner.

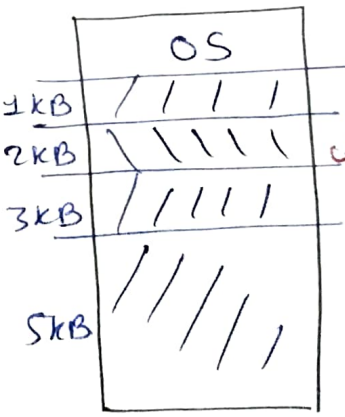
Dynamic Partitioning (Variable Partitioning)

→ Widely used.

→ No two process can put in same partition.

→ One process can't take two partition.

→ Size can be decided as required by process



→ No internal fragmentation

→ If there is internal fragmentation, then there should be external fragment but if there is no internal fragmentation we can't say anything external fragmentation.

→ Assume 1KB and 3KB process get completed and both partition are available now. So, we have total space of 4KB but we can't allot it to 4KB process.

→ So there is external frag. but internal frag. is not there.

→ External frag. mainly occurs due to contiguous allotment only.

→ So, process need not be available in contiguous manner to avoid external fragmentation called as paging.

Compaction → Grouping of vacant and occupied parts together. Also called as defragmentation.

It is a solution before paging.

→ But compaction have its own disadvantage as-

for the compaction it takes some time and for that time all process inside m/m needs to be stopped, means CPU get idle and efficiency get reduced.

→ So, compaction is one of the disadvantage of dynamic partition.

Advantages of Dynamic Partitioning-

① Degree of multiprogramming not fixed. (No. of process)

② Size of process is not limited by size of partition.

③ Flexibility.

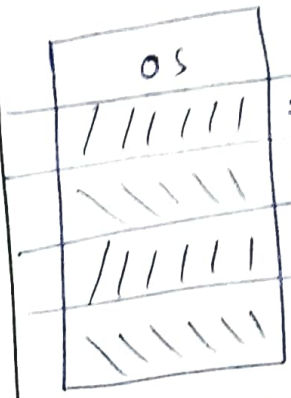
Disadvantages:

① Allocation and deallocation of m/m is very complex.

② Managing holes, (vacant space) (create by swapped out process in m/m (In case of large no of process) is headache.

→ we did it using various data str. and algo.

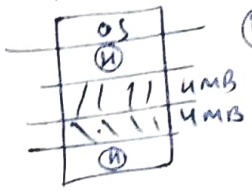
Bit map for dynamic partition



→ Initially there is no fragment.

→ Assume both the 2 MB process swaps out.

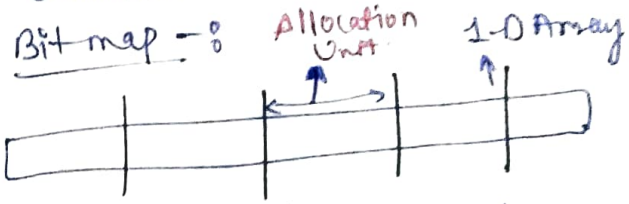
→ After swapping out



H → Holes

So, bit map and linked list are to keep track of holes in dynamic allocation

Bit map :-



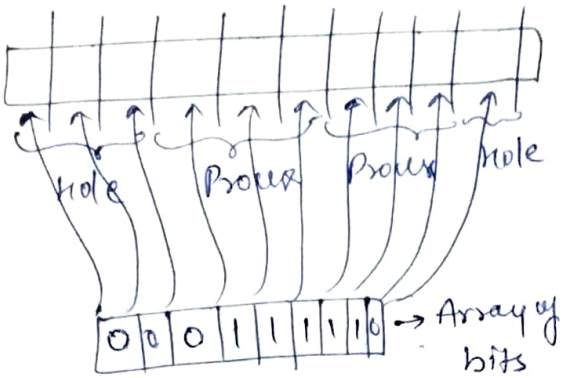
We can decide the size of allocation unit as required.

Assume AU = 4 Bytes.

the complete main m/m get devided into 4 Bytes fragments

1 → 4 Byte each

Assume to be rotate 90° ↻



→ no. of bits = no of allocation units.

→ bits that represent holes contains 0 and occupied contains 1.

→ Assume that Allocation Unit size = 4 Byte = 32 bits and we have a single bit in bit array to represent a Allocation Unit, means for a single AU ~~mean~~ we have 32 units of data and 1 unit for representation.

→ So, total 33 units require for 1 AU, and since 1 bit is extra that of array bit So, 1/33 of space is extra require for each AU.

Assume we are increasing the size of allocation unit then lesser the size of bit array but it creates a problem. If process comes in two different AU's or if a AU contain both process and hole then we can't represent it in bit array.

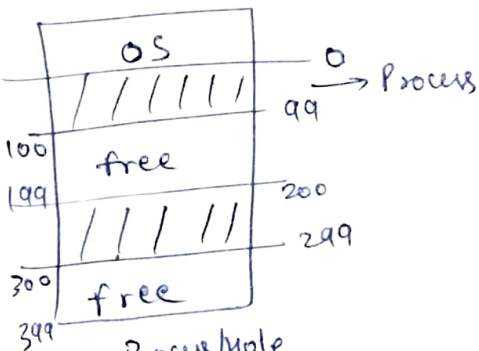
→ find out the hole is also a tidious task bcoz hole is represented by run of 0's and finding sequence of 0 is a slow process.

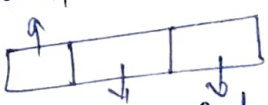
Advantages

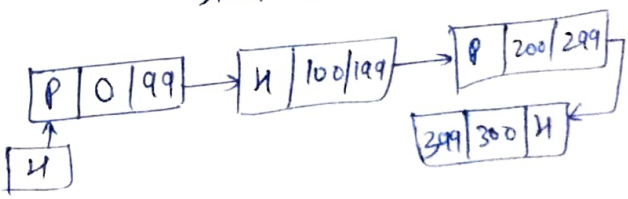
If any process get feed up then it will automatically get combined coith holes (1 converts to 0) and mapped with consecutive 0's.

Linked List

→ Linked list is using more frequent than bit map.







→ Searching doesn't take time.

→ It also takes less m/m.

→ We can insert node at begging in O(1).

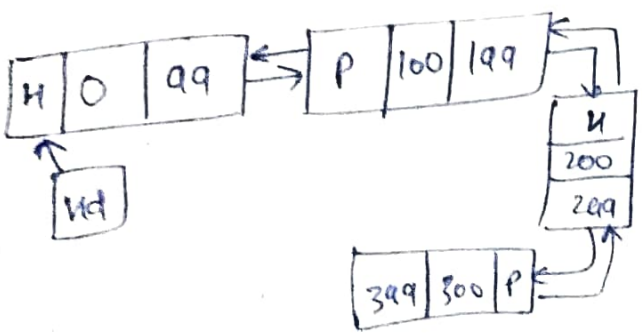
→ We maintained list in increasing order of starting add. so that if process swaps out merging can be easy.

First fit, Best, Fit, Next Fit, Worst Fit

Although all the algo. can use in both dynamic & fixed partitioning but we are considering the case of dynamic partitioning bcoz fixed partitioning is not in use.

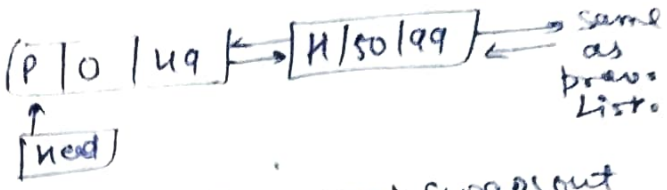
① First Fit [-faster among all]

OS
Hole (0-99)
Process (100-199)
Hole (200-299)
Process (300-399)

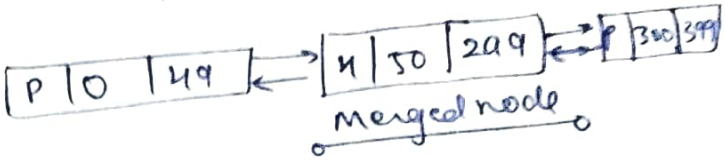


→ Start from beggining and if you get a hole that is enough to hold a process then allocate it to

Assume, P = 50 bytes came in order to execute then in first fit



If Process (100-199) swaps out then



② Next fit (Not Useful)

It is not better than first fit.

→ It is working is similar as first fit but we don't start from beggining we start from the next node to the node we left earlier.

③ Best fit

Find the smallest hole that is big enough to hold the process to reduce the waste of space.

Assume we have 3 holes as 20 MB, 10 MB, 5 MB then best fit allot the 5 MB for 3 MB process

→ Disadvantage

① It is slower than first fit.

② After allocation newly formed hole size is very small, so that we can't allot it to others.

Worst fit → It is just reverse of best fit. We allot the process to largest hole.

→ Searching for largest hole is time taking process.

Quick fit → Maintain a linked list contains most frequently used hole sizes.

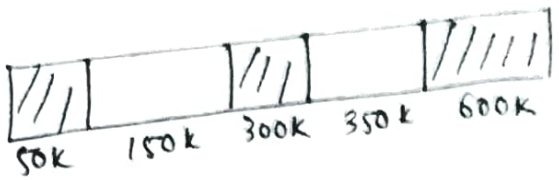
→ It waste lot of space sometimes.

→ So, best algo. is still first fit.

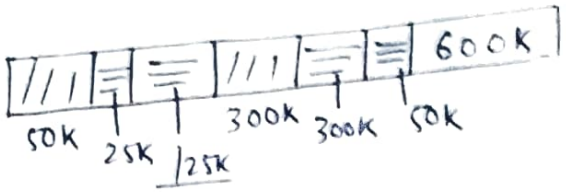
→ If best fit linked list is get sorted in increasing order of hole size, only for holes, then it will works same as first fit.

→ In C Lang., malloc function use first fit algo. and it's faster one.

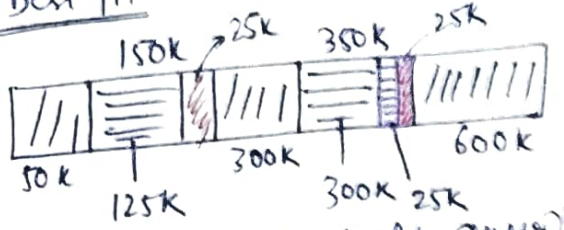
Q Process request
300K, 25K, 125K, 50K



First fit



Best fit



→ So, 50K can't be kept anywhere.

This request can be satisfied in first fit not in Best fit due to external fragmentation.

→ It might be possible that we get different ans. if change the order of request.

Variable Partitioning

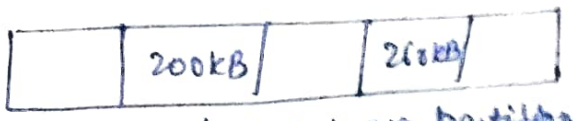
Gate 1996

Given space = 1000 kB

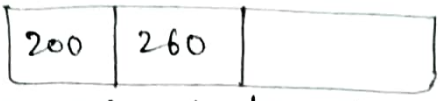
Two partition of size → 200 & 260 kB

So remaining space
= 100 − 460
= 540 kB

1000 kB



We want to get max partitions so we arrange in this order.

① 

→ only 1 partition

② 

→ Two partition

③ 

→ max. 3 partition possible.

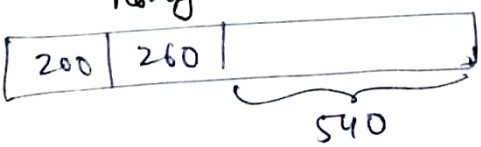
To get 'smallest portion request that can be denied', devide into 3 partition of equal size.

→ $\frac{540}{3} = 180$

any request greater than 180 is denied, but 181 is smallest.

→

If they ask for largest request to denied then arrange in way to get min. no. of partition as -

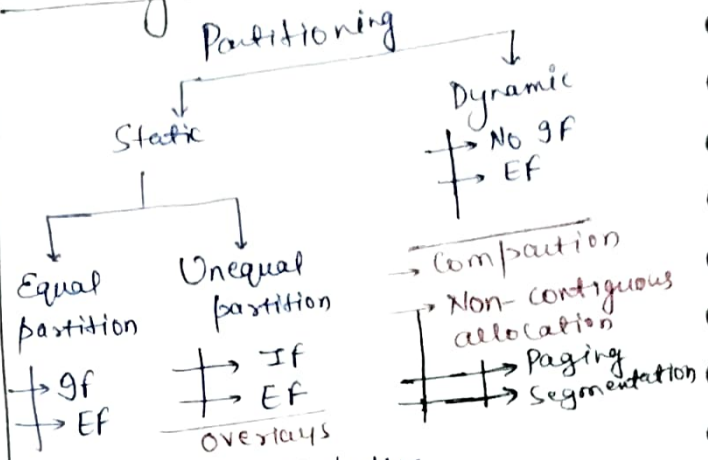


→ Ans- 541 should be denied.

Gate 1998 → Good question do it by self.

Q. When will 20k job get completed?

Summary on Partitioning



→ But Unequal is better than equal partitioning

If → Internal fragmentation

Ef → External fragmentation

→ In unequal size we are limited with process size we can't take process with size more than max. partition. for the solution we have 'overlays'.

whenever process is running it doesn't use complete program at a time (just use some part of it). So if there is no sufficient partition then upload only some part of program in m/m and perform swap in and swap out as the parts get executed called overlays.

→ Compaction is time taking and costly.

→

Overlays and virtual m/m are similar concept but overlays done by user and virtual m/m handle by OS.

→ Virtual m/m means to use RAM effectively we keep all programs in Hard Disk and take a part of it to Ram. So that more process can run simultaneously.

Overlays

Overlays driver is code written by user bcoz its responsibility of user.

Q Consider a 2 pass assembler
pass 1: 70kB pass 2: 80kB
Symbol Table: 30kB (ST)
Common Routine: 20kB (CR)
Total m/m ⟶ 200kB

But at any time only one pass will be in use and both the passes always need ST and CR. If overlays driver is 10kB, then what is the min. partition size required?

Soln

Pass 1: 70 + 30 + 20 + 10 = 130kB

Pass 2: 80 + 30 + 20 + 10 = 140kB

So, we require min. 140 kB partition bcoz both passes are coming alternatively.

This way we can manage in 140 kB only instead of 200kB using overlays concept.

Gate(98) (Overlay tree)

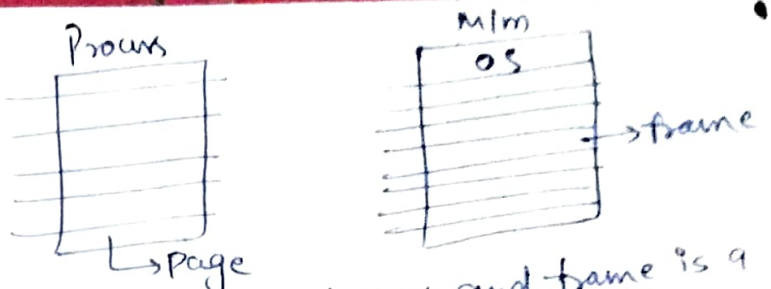
Ans- 14 KB

Traverse from Root to leaf and find m/m size in all four cases and select max. size and become min. partition required.

Need for Paging:

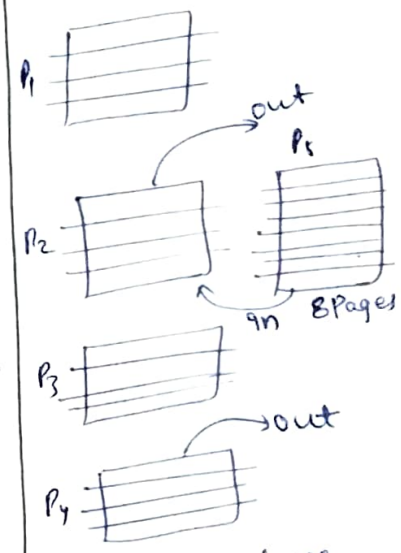
→ Solution of external fragmentation

→ we devide the process and main m/m both in small parts of same size.

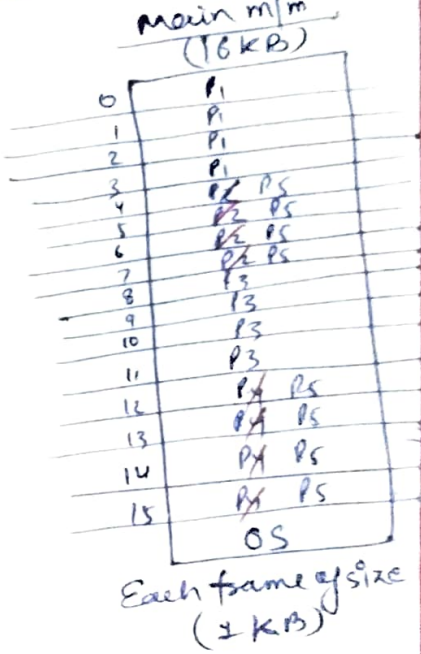


Page is a part of process and frame is a part of memory.

→ Page size and m/m frame size should be same



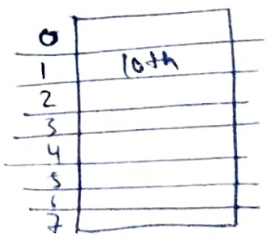
Each process have 4 page (P_1-P_4)



Each frame of size (1 kB)

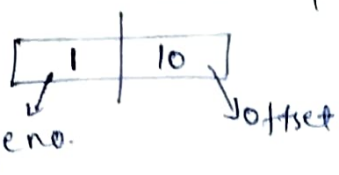
Assume that P_2 and P_4 go out for I/O and in that time P_5 that have 8 pages get inside the m/m. So P_5 will take place on the slots make free by P_2 and P_4.

→ But it creates issues as CPU will generate logical address b/w (0-7) and how will it get mapped on main memory, so, we need a m/m management unit.

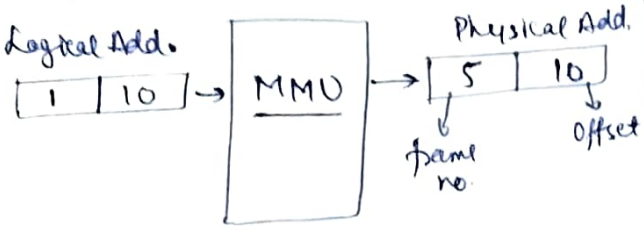


Assume CPU wants 10th word of 1st Page of Process than it will send logical address as—

Logical address (Two parts)



Offset contains value of 'word or byte' we require.



→ CPU generates relocatable add.

Ex (Paging)

→ 64B m/m

$2^6 = 64$ (6 bits require in address)

→ frame = 4B

no. of frames = $\frac{64}{4}$ = 16 frames

0	0	1	2	3	X
1	4	5	6	7	X
2	8	9	10	11	P_1
3	12	13	14	15	P_1
4	16	17	18	19	P_1
5	20	21	22	23	P_1
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

64B (MM)

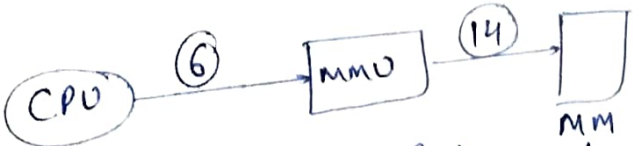
→ Process size = 16B.

→ Page size = frame size = 4B

→ No. of pages = $\frac{16}{4}$ = 4 pg.

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

16B Process



→ All the integers are Byte no. because each part have size 4B.

Page table

→ Every process have page tables separately.

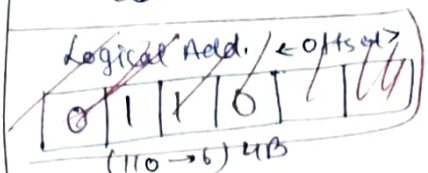
0	f_2
1	f_3
2	f_4
3	f_5

No of entries in page table = no. of pages

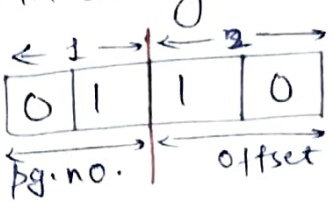
→ It contains info. about frames that contains particular page.

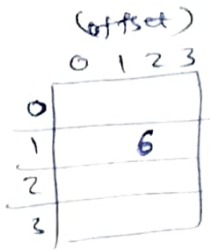
→ No. of bits in logical add. depends upon size of process and in physical add. depends on size of main m/m.

→ We are considering byte addressable.

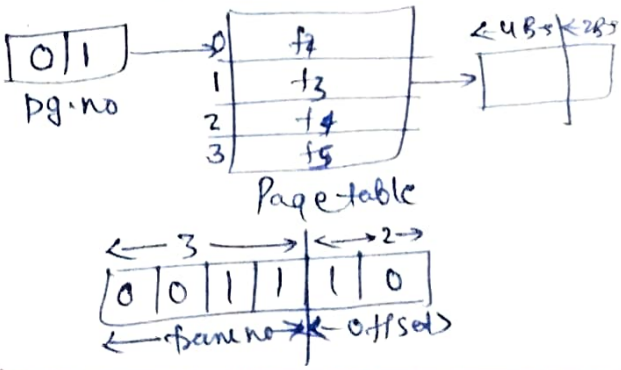


→ 6 in binary 110





We put 6 (0110) in logical add and then separate 2 bit from it for offset because each page size is 4B and two represent 4 we need two bits only (00, 01, 10, 11).



but $(1110)_2 \rightarrow (14)_{16}$ so this way we map the addresses.

→ we can't use load time binding bcoz loader provides absolute address and if process performs I/O and comes back then again new absolute add. req. and it is a tidious task.

→ So we go for run time binding. In run time binding if we use software approach then each time OS will get called, so we do hardware approach for simplicity, bcoz OS is already busy in doing lot of tasks.

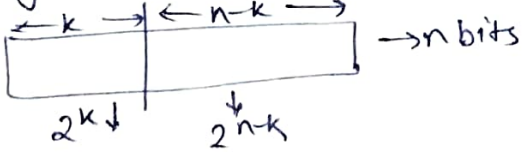
Basics of Binary Add.

2^{10} → Kilo 2^{20} → Mega 2^{30} – Giga

2^{40} – Tera

→ 2^{28} Byte = 256 MB

Using 'n' bits we can address 2^n units

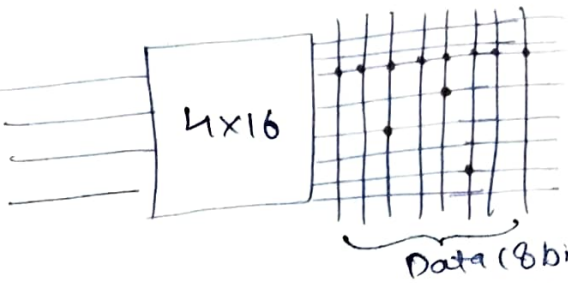


(frame no.) (frame size)

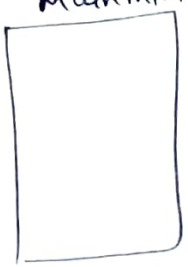
→ Everything is going in power of 2.

Physical Address space and logical address space (PAS / LAS)

Word → Smallest addressable unit in a computer.



→ So word size is 1 byte (8 bits).



Assum main m/m have 128 words then no. of bits required in address

$= \log_2 128$

$= \underline{7 \text{ bits}}$

2mb

PAS = size of main memory.

Given, PAS = 128 kB

$= 2^{7+10}$ B $= 2^{17}$ B

word size = 4 B = 2^2 B

Size of PAS in words = $\frac{2^{17}}{2^2} = \underline{2^{15} \text{ words}}$

→ 15 bits requires to add. each words.

→ Physical address required 15 bit.

→ If PAS = M words then PA requires $(\log_2 M)$ bits.

LAS = size of process.

LAS = 256 MB = 2^{28} B

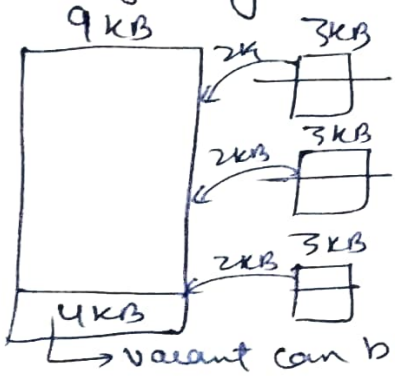
word size = 4 B

LAS in words = $\underline{2^{26} \text{ words}}$

→ 26 bits required to address a logical address.

→ PAS = 128 KB, LAS = 256 MB
it means process can't enters completely in main m/m. so we have concept of virtual m/m, so, we add only a part of process in main m/m.

→ But virtual memory concept can also use, if process can completely enters, in a way to increase the degree of multiprogramming.



→ vacant can be use for more process.

If (LAS > PAS) then,

→ LAS → called as virtual address space here.

LA → virtual address

→ But we can use the name in case of multiprogramming too.

→ Assume,
Page size = frame size = P words
→ then, to address each word in a page require $(\log_2 P)$ bits called as page offset.

Ex Page size = 4KB
word size = 4B
No. of words page size = 2^{10} words
→ Page offset = 10 bits

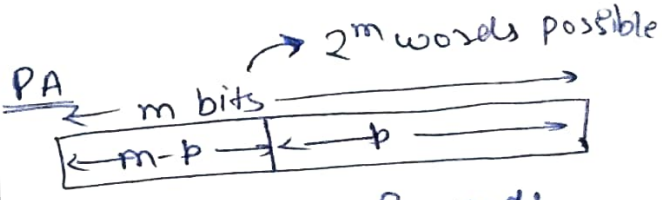
Page Table

PAS = main m/m size = M words
LAS = process size = N words
Page size = P words
$(PA)_{bits} = \lceil \log_2 M \rceil$ = m bits
$(LA)_{bits} = \lceil \log_2 N \rceil$ = n bits
$(\text{Page size})_{bits}$ = offset = $\lceil \log_2 P \rceil$
= p bits

Ex → Say,
LAS = 128 MB PAS = 1 MB
Page size = 4 KB
If don't talk or given about word assume word = 1B.

LAS = 128 M words = 2^{27} words
PAS = 2^{20} words
Page size = 2^{12} words

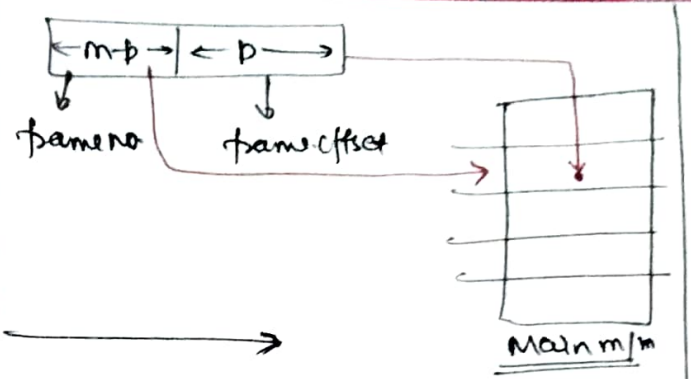
→ LA = 27 bits
→ PA = 20 bits
$(\text{Page size})_{bits}$ = 12 bits = page offset



frame size = P words
Total 2^m frames words in m/m
frame = 2^p word
PAS = 2^m W

No. of frame in PAS = $\frac{2^m}{2^p}$
= 2^{m-p}

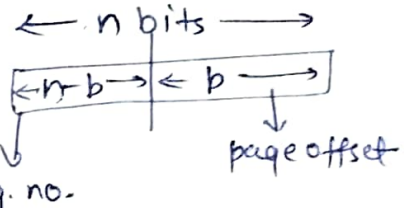
→ 2^{m-p} frames with 2^p words in each frame



LAS = 2^n words

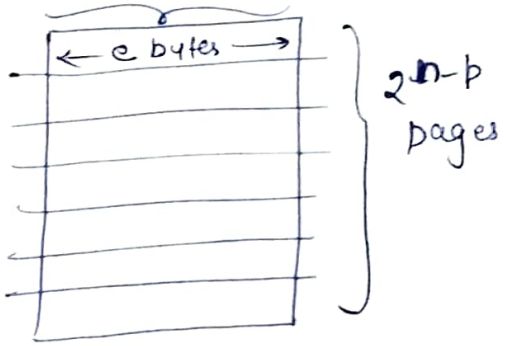
Page size = 2^b words

No. of pages = $\frac{2^n}{2^b} = 2^{n-b}$



Both page offset and frame offset should be same.

Page table



No. of entries in page table = no. of pages in process

Entry size = e bytes

Page table size = $(2^{n-b} * e)$ B

If don't mention about 'e' in question then, Calculate it,

'e' → frame size because entry contains frame no.

⇒ $e = (m-b)$

Page table size = $(2^{n-b})(m-b)$.

Practically '$e > (m-b)$' because entry doesn't contain info. of frame no. only it also contain various other info. too. but if in question 'e' doesn't given then we will definitely go with $e = m-b$ only.

Ex → LA = 27 bits (15 | 12)

PA = 20 bits (8 | 12)

P. offset = 12 bits

Page Table size = $2^{15} \times 8 = 2^{18}$ bits

$= 2^{15}$ Bytes = 32 kB

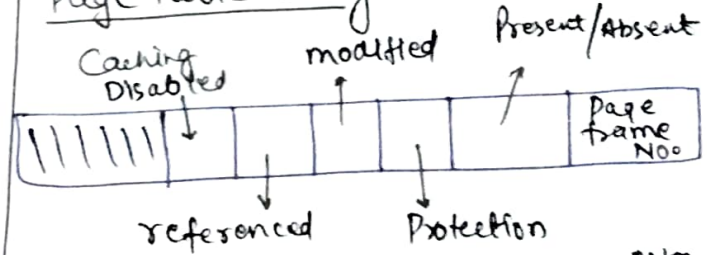
Page size = 4 KB (Given in Ques.)

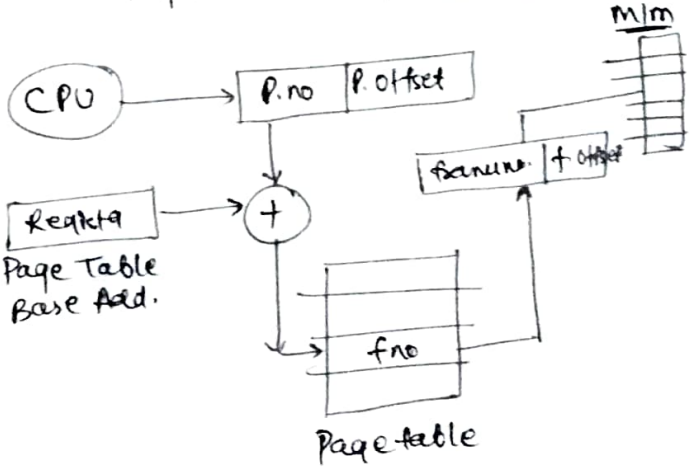
LAS = No. of pages * page size.

PAS = No. of frames * frame size.

Note → Do as questions as possible on this topic.

Page table entry





Caching disabled → Process should not be maintained in Cache for the freshness purpose.

Referenced bit → Page is refered or not.

→ Modified bit says page is modified or not.
If page is modified and you want to keep some another info. in page then that modified info. should be kept in hard disk.
This bit also called dirty bit.

→ Protection bit shows the info. of protection like Read only, write only, R/W, execute, etc.

→ P/A is important in case of virtual m/m concept.
In case of demand paging if page is not present then it will cause a page fault.

→ In this case load the page from hard disk and give it to CPU.

9 mb
Gate 2004 → On Page table entry.

LA = 32 bit ⇒ VAS = 2^{32} B
PA = 30 bits
Page size = 4KB = frame size = 2^{12} B
e = 32 bit

→ Byte addressable.
⇒ VAS = 2^{32} B.
PAS = 2^{30} B.
If given words addressable then 2^{32} w and 2^{30} w.

Page table entry

| Frame no | Others |

Frame no → (m - p)

PA = 30 bit
Page offset = 12 bits
frame no = 30 - 12 = 18 bits

Others = 32 - 18
= 14 bits Ans.

~~No. of pages~~
(Page no.) bits = 32 - 12 = 20
No. of pages = 2^{20}
Table size = $2^{20} \times 32$ bits

Note → Always remember about Byte and word addressable.

Need of multilevel paging

V. Imp
LA = 22 bits
LAS = 2^{22} B (Byte Addressable)
Page size = 4KB = 2^{12} B
Page offset = 12 bits.
(Page no.) = 22 - 12
= 10 bits
No. of pages = 2^{10} = 1k
(Page table entry) size = 4B
Page table size = 1k × 4B
= 4KB.

Working set → No. of pages of process inside main memory bcoz we are inserting a part of process in m/m.

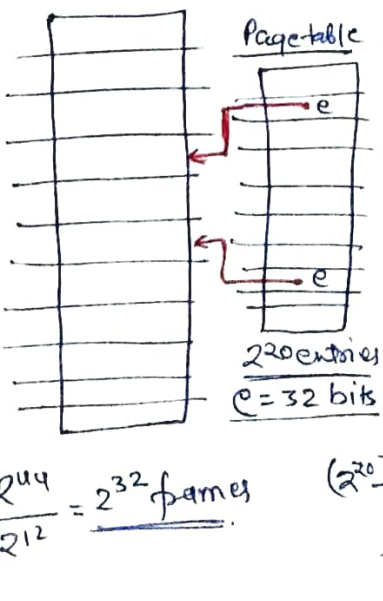
Principle of locality → It says that at a point of time a process can insertanly few pages called working Sets, according to requirement.

→ Page table resides in main m/m. main m/m devides in frames of some size then if page table size is greather than frame size then we will devide the page table according to frame size and then kept it to

→ Assume page size or frame size is 256 kB but page table size is 1024kB then table will devide in 4 frames.

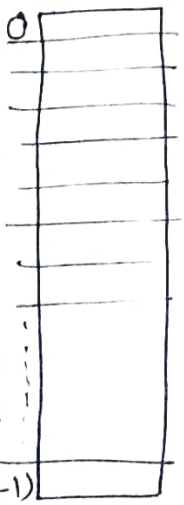
Two-level paging examples

Page size = 4KB
PAS = 2^{44} B



$\frac{2^{44}}{2^{12}} = 2^{32}$ frames

LAS = 2^{32} bits
LA = 32 bits



$\frac{2^{32}}{2^{12}} = 2^{20}$ pg.

→ We have page table to map pages and frames.

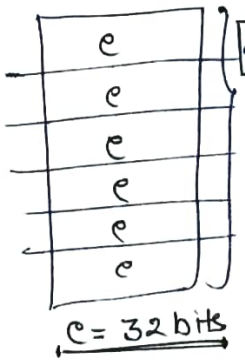
Page table size = $(2^{20} \times 4)$ B
= 4MB

frame size = 4kB

→ So, page table can't fit in one frame, so we will devide it.

→ No. of pages in page table after division = $\frac{4MB}{4kB} = 1k$

So, to keep track of record that which page of page table present in which frame of main m/m.



2^{10} (1k) This new page table will have entries equals to no of pages in previous or devided page table.

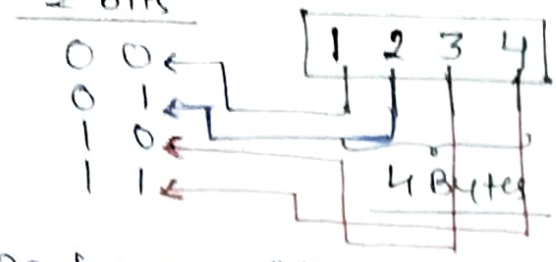
Page table size = $(2^{10} \times 4B)$
= 4kB.

→ So, now page table size equals to frame size hence we have no need to further devide the 2nd page table.

→ So, here we have 2 page tables thats why it is known as 2-level paging.

→ In LAS, page size is 2^{12} B so, to check or identify 1 B we needed 12 bits.

Ex Assum page has 2^2 Bytes data then we require 2 bits to identify a single Byte in LAS as,



We have a different 2 bits combination for each byte in page.

→ Similarly, for 2^{12} Bytes we need 12 bits.

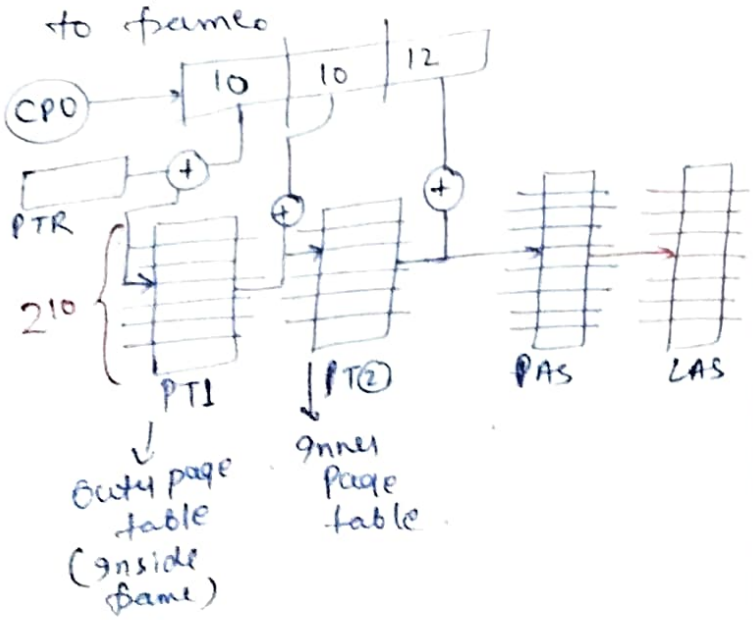
Similarly assume each entry of page table is 4 Bytes then how many entries will be present in one page.

Page size = 4KB and entry = 4B

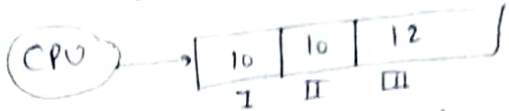
⇒ no. of entries = $\frac{4KB}{4B} = 1K = 2^{10}$

So, to address any bit in page table we need '10 bits'.

→ Each CPU have a Page table register that have the address of starting of page table, and page table is resides inside the frame. so basically register points to frame.



→ Load a process without paging limits the degree of multi-programming.



I contains 10 bits info bcoz, in page table 1 (outer table) contains 2^{10} entries so to point out an entry we need 10 bit number.

This 10bit no. get add with base add. of PT1 to gives the exact page no, we look at the frame no in that entry and then add it with base add. of PT② to get the final frame no.

→ To get final frame in PT2 we have II part 10 bit because frame size is fix for both the tables.

→ and finally as we get final frame in main m/m we add Offset to it, so that we can get that particular byte that we needed.

Examples on multilevel paging

→ Assume that everything is byte addressable

Cont...

Note → Video is also downloaded first do all the questions by self and then watch video if required.

	Virtual Address	Page Size	Page Table Entry size	PT 1	PT 2	PT_3
①	48 b	16 KB	4B	2^{36} B	2^{24} B	2^{12} B (4KB)
②	64 b	1 MB	4B	2^{48} B	2^{28} B	2^{16} B
③	72 b	1 GB	4B	2^{44} B	2^{16} B	—
④	72 b	256 MB	4B	2^{46}	2^{20}	—
⑤	72 b	16 MB	4B	2^{50}	2^{28}	2^{6}

Address Split

① | 10 | 12 | 12 | 14 |
PT3 PT2 PT1 offset

② | 8 | 18 | 18 | 20 |

③ | 14 | 28 | 30 |

④ | 18 | 26 | 28 |

⑤ | 4 | 22 | 22 | 24 |

Soln ① VA = 48 b VAS = 248 B

Page size = 2^{14} B No of pages = $\frac{2^{48}}{2^{14}} = 2^{34}$

(PT1) size = $2^{34} \times 2^2 = 2^{36}$ B

(PT2) = $\frac{2^{36}}{2^{14}} = 2^{22}$ pages

(PT2) size = $2^{22} \times 2^2 = 2^{24}$ B

$(PT_3) = \frac{2^{24}}{2^{14}} = 2^{10}$ pages $= 2^{10} \times 2^2$

$= 2^{12}$ B (4KB)

4KB can fit in 16 KB

No. of entries in page table = $\frac{16\text{kB}}{4\text{B}} = 2^{12}$ entries

= 12 bits

multi-level paging which table (outer or inner) get level 1 or level n → To see.

② How to fill or the order of filling the logical address block in multi-level paging

Both are important must do

ze, it means table is
address split assume that
hem,

→ 12 bits

Take care of pagetable size and page size.

paging

Assume that except the outer level page table, all the pages of inner level Page table are completely full.

Page size	PTE	Outer page table size	levels of paging	VAS (To find)
4KB	4B	4KB	1	4MB
4KB	4B	4KB	2	4GB

4KB	4B	4KB	3	4 TB
4KB	4B	256B	1	256 KB
4KB	4B	256B	2	256 MB
4KB	4B	256B	3	256 GB

Sol^n

① $PS = 2^2 \times 2^{10} = 2^{12}B$

Page offset = 12 bits

Page table size $= 2^2 \times 2^{10} B = 2^{12} B$

No of entries $= \frac{2^{12}}{4} = 2^{10}$ bits

10	12
← VAS →	

→ 4MB, 2^{22} B

②

PT entry (4KB)

No of pages = 1K

1K pages in inner page table of size 4B

Process

4KB

1K, 1K, 1K, 1K

1K pages of 4KB

total pages $2^{10} \times 2^{10} = 2^{20}$

Size of process $= 2^{20} \times 4 \times 2^{10}$

$= 4GB$

③

4KB

1K, 1K, 1K, 1K

total no of pages $= 2^{30}$ pages

Size of process $= 2^{30} \times 4K$

$= 4 TB$

④ $\frac{2^8}{2^2} = 2^6$ pages

$2^8 \times 4$ KB

28 KB

⑤

Completely full

25B, 64, 1K

total = 64K

~~(4KB + 1KB × 4B)~~

~~4KB (1KB 6~~

$64K \times 4KB = 256MB$

⑤ $\frac{64K \times K^2 \times 4KB}{64 \times 4 \times 2^{30}}$

256 GB

Q 9 MP

Fill in the blanks in such a way that PT will fit in one page in single level paging.

VAS	Page size	Page table entry (e)
4MB	4KB	$\leq 4B$
4GB	128KB	$\leq 4B$
128TB	32MB	$\leq 8B$
256MB	$\geq$ 32KB	4B
512KB	$\geq$ 1KB	2B
16GB	$\geq$ 256KB	4B

① PT × PT entry = Page size

VAS = 4MB

PS = 4KB

No. of pages = 1K

10	12

Page table size = Pages * e

But, page size $\leq$ PS [to fit in 1 page]

$e \leq \frac{PS}{Page} = \frac{1KB}{1K} = 1B$

(2) No. of pages $= \frac{4 \times 2^{30}}{2^7 \times 2^{10}} = \frac{2^{32}}{2^{17}}$

$= 2^{15}$

$e \leq \frac{128 KB}{2^{17}} = 2^2$

(3) Similar as above (2)

Ans $\leq$ 8B

(4), (5), (6) are of same type so we do (4) here and (5) and (6) are similar ones.

(4) 1 page table should be fit in 1 frame.

VAS = 256 MB = 2^{28} B

PS = p bytes (assume)

No. of pages = $\frac{VAS}{PS} = \frac{2^{28}}{p}$

PTS = $\frac{2^{28}}{p} * 4B$

$= \frac{2^{30}}{p} B$

To fit a page table in a page or frame

page size (p) $\geq \frac{2^{30}}{p}$

$\Rightarrow p^2 \geq 2^{30}$

$\Rightarrow p \geq 2^{15} = 32KB$

Similarly

(5) $\geq 1KB$

(6) $\geq 256KB$

Gate 01 and 99

Page table size = 14×2^{20} bits

To convert it into MB divide by 8

$\frac{14 \times 2^{20}}{8} = 2MB$

Imp

Note → It is important to take care of units otherwise you will get error.

Gate 06, 95, 97 (1995)

for virtual m/m concept

LAS must be greater than PAS.

LAS > PAS and LAS < Secondary m/m.

but we can use it in some other extent too.

1997 → Dirty bit also helps avoid unnecessary writes on paging devices.

2006

(1) Multi user related to multi programming.

(2) Cache have no relation with VM.

(3) H/w support required for paging even in case of (no virtual m/m).

(4) CPU scheduling and virtual m/m concept have no relation among them

Gate-2008 (Multi-level paging)

PA = 36 bits, VA = 32 bits, PS = 4 KB

VAS

Ans will be either (20, 20, 20) or (24, 24, 24) bcoz frame size is same in each page table

PA

24	12

36

PS = 4KB

Offset = 36 - 12 = 24

So, Ans is (24, 24, 24).

Note → If nothing is given assume as byte addressable.

→ In the above question, there is given lot of data but we can solve it in just few steps without using all the data.

Note → Sometimes (not always) Question contain unrequired data.

Dirty → write-back policy

R/W → Page Protection

Reference → Page-replacement policy

Valid → Page Initialization.

If page is not refered from long then we can replace it.

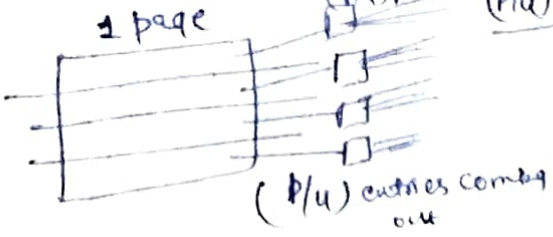
Gate 2013 (Read the question and do it by self first)

VA = 46 bits PA = 32 bits

3-level paging. PTE(e) = 32 bits = 4B

→ Page Size = p (say)

No. of entries = $\left(\frac{p}{4}\right)$



So total $\left(\frac{p^2}{16}\right)$ entries are here.

And in 3rd each $\left(\frac{p^2}{16}\right)$ entry provide $(p/4)$ entry.

So, in 3-level we have $(p/4)^3$ entries = $\frac{p^3}{64}$

$VAS = (p/4)^3 * p$

$2^{46} = \frac{p^4}{64}$

$p^4 = 2^{46} \times 2^6$

$= 2^{52}$

$\boxed{p = 2^{13}}$

We assume that all the inner tables are completely filled.

Finding optimal page size

Assume

VAS = 4 GB

Page size = 4KB

Pages = $\frac{4GB}{4KB} = 1M$

If, Page size = 4MB then

Pages = 1k

If page size ↑ then pages ↓ and page table size ↓.

Consider two cases,

(1) PS = 1KB (Page size)

VAS = 10KB

No. of pages = 10

means no overhead bit

In case (2)

(1) PS = 1 KB , VAS = 10 KB + 1 B

Here, 10 pages get completely filled and for remaining 1 B we have to dedicate another page and this is overhead.

So, on an average a VAS contains both kind of process as discuss in case (1) and case (2) and waste (1/2) of the page.

To avoid this we generally take following approach as—

Assume,

PS = p bytes

PTE = e bytes

avg. VAS = s bytes.

Overheads, $= (p/2) + (s/p) * e$

$(p/2)$ ← waste bcoz of 1 B m/m alloted to last page of process.

$(s/p) * e$ → Bcoz of page table size.

What should be the min. page size to avoid or minimize overhead.

$$\Rightarrow \frac{d\,Ov}{dp} = \frac{d}{dp}\left(\frac{p}{2} + \frac{s}{p} * e\right)$$

$$\Rightarrow \frac{1}{2} - \frac{s}{p^2} e = 0$$

gmp ⇒ $p = \sqrt{2se}$

Q. Give, on avg VAS = s and PTE = e and PS = p find p?

avg.(s)	e	p (Best page size)
4 KB	8 B	$\sqrt{2^{12} \times 2^3 \times 2} = 2^8$
16 MB	2 B	$\sqrt{2 \times 2^{24} \times 2} = 2^{13}$

→ Generally last page of the process gives overhead.

Generally process have 3 sections as—

(1) Code section,

(2) Data Section

(3) Stack section

and they might get increase in no. So say process have 'n' sections.

then process will get devide into 'n' sections and then each section get divide into pages, in that case, page overhead will be $\left(\frac{np}{2}\right)$ instead of $(p/2)$, Hence

$$\Rightarrow \frac{n}{2} - \frac{s}{p^2} e = 0$$

$$\Rightarrow \frac{s}{p^2} e = \frac{n}{2} \Rightarrow p^2 = \frac{2se}{n}$$

gmp ⇒ $p = \sqrt{\frac{2se}{n}}$ → for 'n' sections

Virtual Memory

→ If process size > m/m size then this concept helps OS.

→ It's just maintain by the OS.

→ We can (↑) the degree of multiprogramming too, hence CPU Utilization also increases.

→ OS always reside in main m/m along with loader.

→ Part of process present in main m/m or it can be non contiguous allocation. So there is no external fragmentation

→ Cont—