Memory Management

As we know that to enhance the overall performance of computer, several processes must be executed simultaneously. So, several process must be kept in memory simultaneously. So, memory should be shared and managed.

Memory is one of the important component of a computer system. In terms of memory, the efficiency of a computer system will depend on size, access time and cost of the memory.
So, size should be larger, access time should be smaller and cost should be smaller.
By maintaining a hard disk in secondary memory, size and cost factor can be managed but to reduce the access time, we need maintain primary memory i.e. RAM or random access memory.

Locality of reference:- A program is a set of instructions. When CPU executes a program, it executes in sequential manner. i.e. if currently CPU is executing instruction number 120 then it is highly probable that next instruction executed by CPU will be 121 until any jump statement is not coming. So, this concept of next instruction getting executed is called locality of reference.
So, even if size of RAM is smaller than hard disk, due to the concept of locality of reference, reality of its smaller size can be minimized.

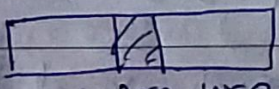
Allocation of main memory: The process of assigning space to a program in RAM is called memory allocation.

Page No.:
Date: __/__/__

memory allocation can be contiguous or non-contiguous. In contiguous allocn, whole program is given space in RAM at a stretch whereas in case of non-contiguous allocn, the program is broken into multiple pieces and each piece is kept in different locn of RAM.

Advantages and disadvantages of both methods can be realited with the example of array (contiguous) and linked list (non-contiguous).

In contiguous allocn, access will be fast but space will be ill-managed. e.g.

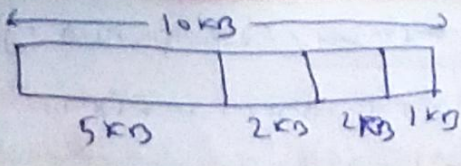


Here, 5KB process cannot be accommodated even if 8KB is vacant. This drawback is called external fragmentation.

In case of non contiguous allocn, like linked list, random access is not possible, however, space will be well managed.

Contiguous memory allocn is of two types:
(i) fixed size partitioning (ii) variable size partitioning

In fixed size partitioning, an entire partition has to be allocated to a process.

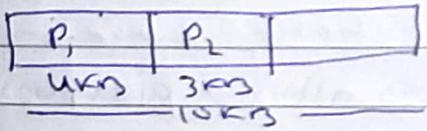


Page No.:
Date: / /

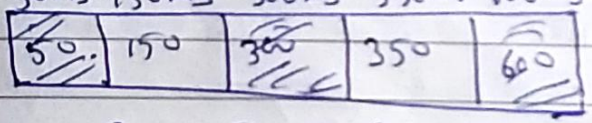
Here, 10KB RAM is divided into 3 types of partitions 5KB, 2KB and 1KB.
Now, if a process is of size 3 KB comes then whole 5KB partition has to be allocated to the process. So, 2 KB will be unused and cannot be allocated to other smaller size process.
This wastage of space is called internal fragmentation.

In case of variable size partitioning, there is no pre-defined partitioning.

$P_1 = 4KB$
$P_2 = 3KB$



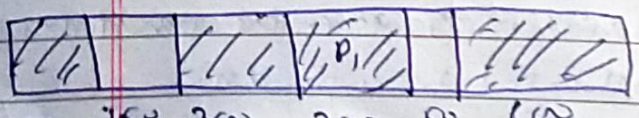
So, no internal fragmentation
Managing the memory in fixed size partitioning is easy.



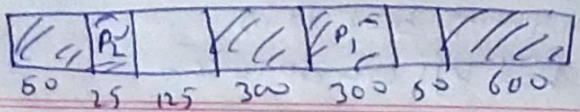
$P_1 = 300$ KB, $P_2 = 25$ KB, $P_3 = 125$ KB
$P_4 = 50$ KB

We are following variable size partitioning
First fit allocation:- Process will be stored in the 1st capable partition.

So, after allocation of P_1, status of memory.

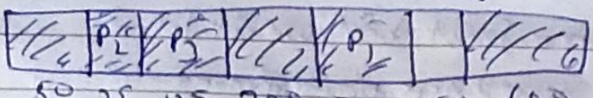


After allocation of P_2 status of memory

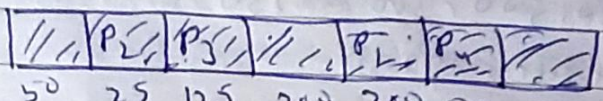


Page No.:
Date: / /

After allocation of P_3 status of memory

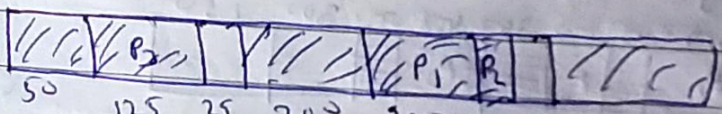


After allocation of P_4 status of memory



Best fit allocation :- Process will be stored in the smallest capable partition.

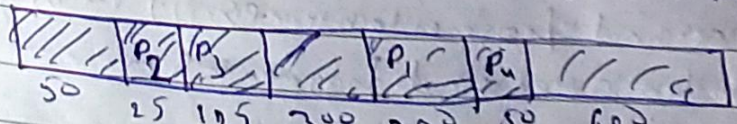
Status of memory after allocation of P_1, P_2, P_3



Here, P_4 cannot be allocated. i.e. Case of external fragmentation is there.

Worst fit allocation :- Process will be stored in the longest capable partition.

Status of memory after allocation of P_1, P_2, P_3, P_4

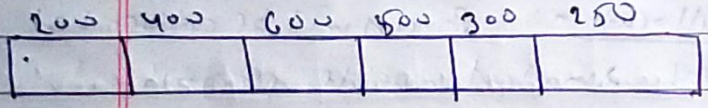


worst fit policy works best in case of variable size partitioning but it is highly probable that left over space can be reused again.

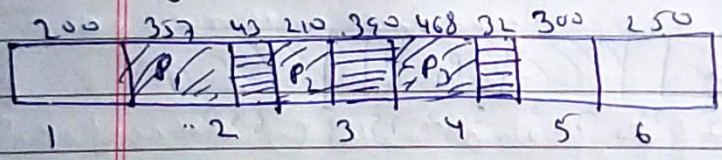
Page No.:
Date: / /

Now we will analyze first, best, worst fit policies on fixed size partitioning

Let there are 6 types of partition of sizes 200, 400, 600, 500, 300 and 250
Let the memory size = 2250
Processes arrival order and sizes are
P_1 357, P_2 210, P_3 468, P_4 491



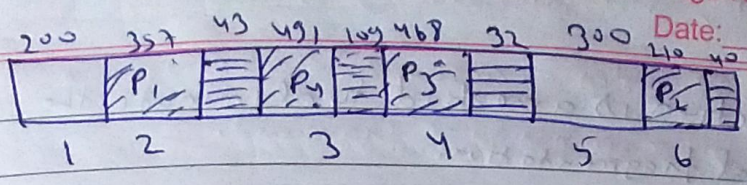
Status of memory after 1st fit policy



Here, partitions 2, 3, 4 have internal fragmentation
Due to external fragmentation P_4 cannot be allocated.
If size of the process is greater than size of available space then it is not called external fragmentation. eg. if P_4 is of size 800 then no external fragmentation.

If size of the process is less than available space then it is called external fragmentation may occur. In that case, size of the process will be the size of external fragmentation.

Status of memory in case of best fit



Here, partitions 2, 3, 4, 6 faces internal fragmentation.

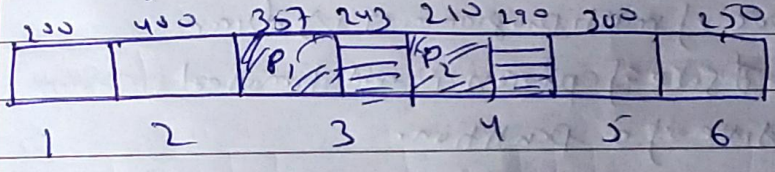
memory wasted in internal fragmentation

= 43 + 109 + 32 + 40

memory wasted in external fragmentation

= 0 but all processes were allocated.

Status of memory in case of worst fit



Partitions 3 and 4 faces internal fragmentation

Space wasted due to internal frag. = 243 + 290

Since, total space available after allocation of P_1 and P_2 is greater than sum of sizes of P_3 and P_4.

So, external fragmentation exists.

So, space wasted due to external fragmentation

= 468 + 491

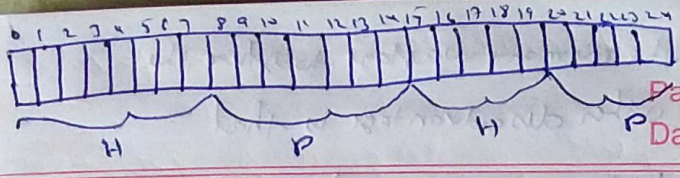
Compaction is a method to eliminate the external fragmentation.
Pulling all occupied part together and pulling all free part together is called compaction or defragmentation.
However, during the compaction, running processes need to be stopped for sometime to shift the place of storage of the process.
So, Due to stopping of processes, the efficiency of the system will go down.

Advantages of Variable size partitioning:
(i) degree of multiprogramming is dynamic
(ii) size of process is not limited by size of a partition.

Disadvantage - (i) allocation & deallocation of memory is complex

Data structures to keep tracks of occupied and free space in case of variable size partitioning.

(i) Bit map: Let allocation unit is 4 byte.
So, whole memory will be divided into chunks of 4 bytes. Now, each allocation unit is represented by a bit. bit will be 0 if the allocation unit is free else 1.



Page No.:
Date: / /

5. ~~Use of bit map~~

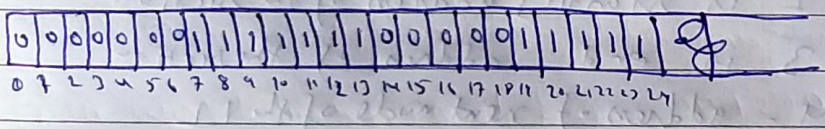
Here, memory is divided into 25 alloca^n units. frm alloca^n unit 0 to 7, there is a hole i.e. no memory space is unallocated.

frm 8 to 14, a process is there.

frm 15 to 19 hole

frm 19 to 24 occupied by process.

now, to represent the above occupied and free space of above memory in bit map we can maintain an array of size equal to no. of alloca^n units in memory. If the value at a particular posi^n in array is zero, it means corresponding alloca^n unit is unoccupied in memory. In this case



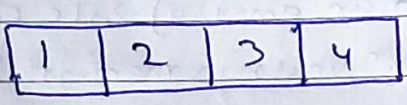
So, if alloca^n unit is of 4 byte i.e. 32 bit then for each 32 bit of memory, 1 bit of bit map is wasted.

If alloca^n unit is of big size then bit map size will decrease but if due to big alloca^n unit memory space will be wasted bcz alloca^n will always start frm beginning of alloca^n unit.

Apart from the memory wastage to store the bit map, other disadvantage is that to find the holes to accomodate a new process, entire bit map has to be traversed.

Advantage of bit map is that when a process is terminated, the corresponding allocation unit entry in the bitmap is made 0 which is easy. However, due to its disadvantages, bit map is not recommended.

(ii) Linked list: node of the LL will be

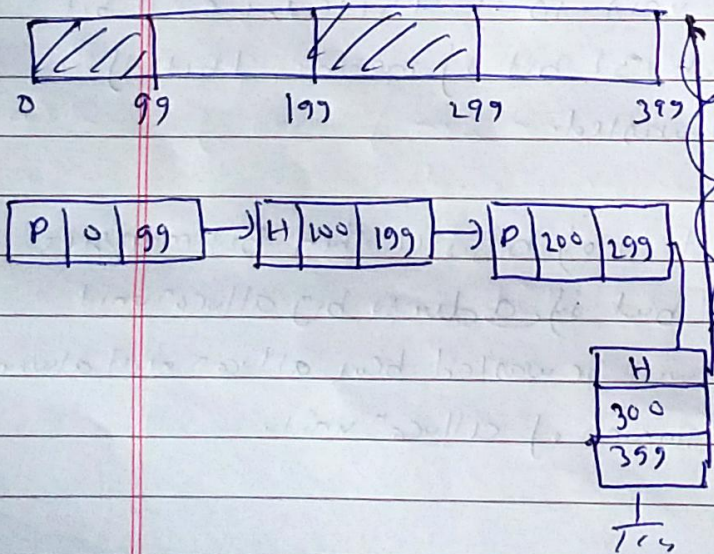


1 will contain either the section is a process or whole.

2 is starting address of the section

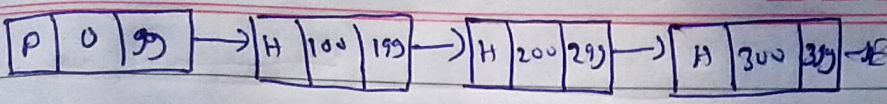
3 is end address of the section

4 - address of next node of the LL

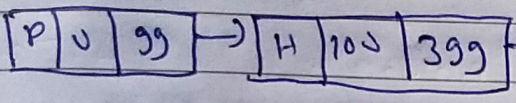


If the chunk b/w 199 to 299 becomes vacant then LL will become:

Page No.:
Date: / /



If in place of single LL, if we use double LL then 2nd, 3rd and 4th nodes can be merged

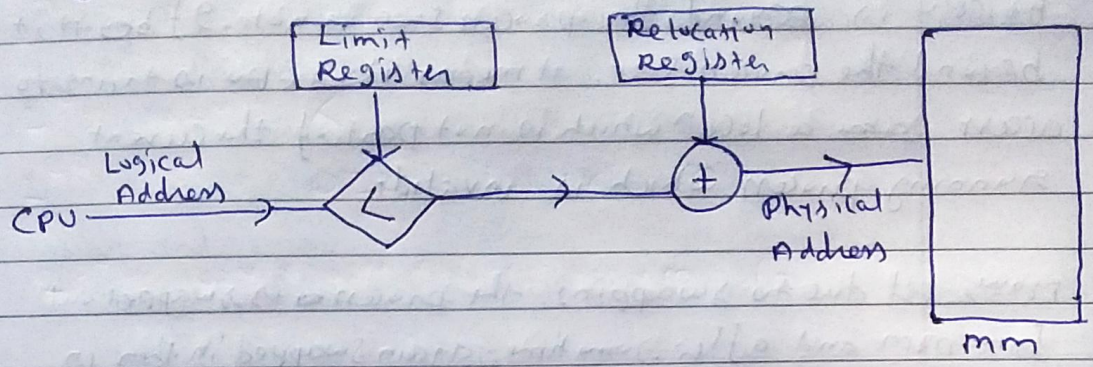


⇒ one of the sol^n of external fragmentation is non-contiguous allocation.

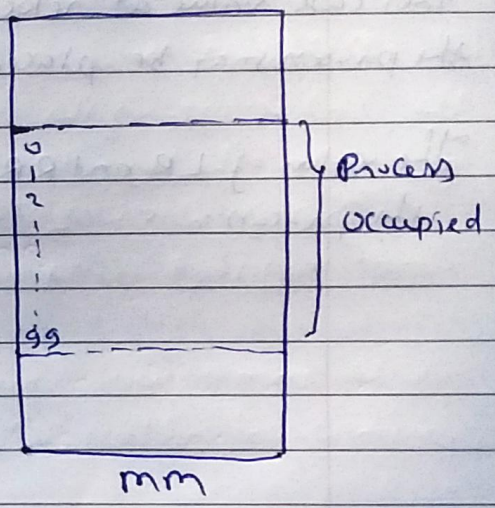
Non-contiguous allocation is of two type:
(i) paging (ii) Segmentation

Page No.
Date

Relocation in case of contiguous memory allocation



Let a process is of 100 byte is in the MM. Let it has occupied the space in MM from 100 byte starting from 100Kth byte of the MM.



In order to fetch any instruction of the process from MM, the CPU will generate logical address, i.e., any no. b/w 0 to 99 in this case. However, in reality Let, the logical address is 5. But, in reality, in MM the instruct^n at 5th byte of the process is actually located at (100K+5)th byte of the MM, which is the physical address of the instruction.
The conversion of logical address to physical address is done by memory management unit (MMU).

In this case, value at limit register will be 100
Relocation register :. 100K.

Page No.
Date

Limit register is to check that logical address generated by CPU is beyond the process site or not. If beyond it is beyond the process site, it means CPU is trying to access a location which is not part of the current running process, which is invalid.

Now, let due to swapping, the process is swapped out from MM and after some time, again swapped in then in this case value at relocation register will change but the process may be placed at some different location in MM.

The value of LR and RR is set by MMU and not by the process.

Page No.
Date

* Non-Contiguous Allocation

In case of contiguous allocⁿ, external fragmentⁿ is a major drawback.

In case of variable size partitioning, keeping track of occupied and unoccupied spaces in mm is troublesome.

As a solⁿ, concept of paging is proposed.

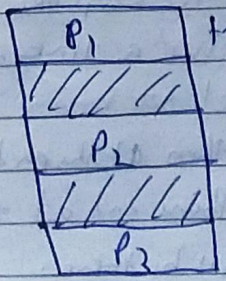
It is a solⁿ for non-contiguous allocⁿ with fixed partition.

In paging, a process is divided into multiple fixed size pages. Similarly, mm is also divided into ~~same~~ multiple same size page. In case of mm, these pages are called frames.

Let, page size = 1KB
process size = 1MB
mm size = 1GB

So, we can say that Process site is of 1K fully Pages.
mm site is of 1M pages.

Let a process ~~ba~~site is of 3 pages. In case of non-contiguous allocⁿ,
Here, the pages P_1, P_2, P_3 of the processes are stored in 3 different places in mm.
So, efficient use of spaces in mm.



Page No.
Date

Since, before coming to mm, a process space is already divided into pages. ~~Of course~~ In other word, the hard disk ~~is actually divid~~ space is actually divided into pages.

In case of paging, now the logical address generated by CPU will be in terms of page no. and offset

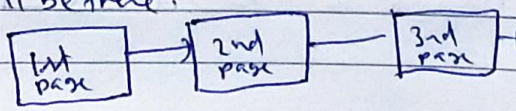
Let, CPU want to access the 4th instruction of the 2nd page of the process then it will generate the logical address as

page no. - 2
offset - 4

Now, mmu will convert this logical address (LA) to physical address (PA), i.e., actual address in mm.

Since, page size and of process and frame size of mm are same. So, offset no. will be same in both LA and PA.
Now, mmu will have to find frame no. corresponding to the page no. 2 of the process.

One of the method to find the corresponding frame no. can be to keep the address of next page in the previous page, like a linked list.
e.g. Let we know the address of 1st page of process in mm then at the end of 1st page, address of 2nd page will be there and at the end of 2nd page, address of 3rd page will be there.



Page No.
Date

But, this method will result into slow access of the required page.

The second method which is followed is to maintain a data structure, called as page table, corresponding to each process. This page table will contain frame no. in mm corresponding to each page of the process. Let, there are 3 pages of a process. viz. P_1, P_2, P_3.
Let P_1 is stored at 1001th frame of the mm.
P_2 " " 59 " " "
P_3 " " 5036 " " "

So, page table corresponding to this process will be

1001	P_1
59	P_2
5036	P_3

Since, each process will have their personal page table and these page tables are also stored in mm.

The addresses of beginning to these page tables will be store in page table base register (PTBR) which is located in CPU.
So, when CPU generates a logical address in terms of page no. and offset, the mmu will first find the address of beginning of the page table corresponding to the process running in CPU from PTBR. Now, mmu will go to the page table located in mm. Now, from the page table, the mmu will extract the corresponding frame no. and then again mmu will access the mm to access the exact. After extracting the frame no., mmu

will generate physical address (PA) which is frame no. and offset.
Here, offset will be same as that of LA but page size and frame size are same.

So, if LA is page no. - 2
offset - 4

then PA will be frame no. 59
offset - 4

After generating the PA, the MMU will go to MM to acces the generate frame no.

In this whole procss, we can see that to access a memory instruction, two times memory access is needed. First time for to search in the page table and 2nd time for actual access of the instruction.

If a page size is 1 KB and Process size is 3.5 KB
So, here fourth page will face internal fragmentation.

Page No.
Date

$2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $25 = 32$, $2^6 = 64$

$2^7 = 128$, $2^8 = 256$, $2^9 = 512$,

$2^{10} = 1024 = K$

$2^{20} = 1024 \times 1024 = 1024K = M$

$2^{30} = 1024 \times 1024 \times 1024 = 1024M = G$

$2^{40} = 1024 \times 1024 \times 1024 \times 1024 = 1024G = 1T$

$128GB = 2^7 \times 2^{30} = 2^{37}$

$2^{28} = 2^8 \times 2^{20} = 256MB$

Page No.
Date

fm 2 bit we can generate 4 combinaⁿs
00 01 10 11

Similarly fm 3 bits, we can generate 8 combinaⁿs
000 001 010 011 100 101 110 111

∴ fm n bits, we can generate 2^n combinaⁿs

If mm is of size 8 byte then to give an unique address to each byte, we need 3 bits.

In other words, if it is given that 3 bits are used for addressing the memory location and each locaⁿ is of 1 byte long size the total size of the memory is $2^3 = 8$ byte.

If 14 bits are used for addressing and each memory locaⁿ is of size 1 byte then size of the mm
$= 2^{14}$ byte $= 2^{10} \times 2^4$ byte $= 16$ KB

If no. of bits used is 22 and each memory locaⁿ is of size 2 byte then size of mm
$= 2^{22} \times 2$ B $= 2^{20} \times 2^3$ B $= 8$ MB

Page No.
Date

=) If size of mm = 64 KB and size of memory location is of 1B then no. of bits required for addressing

$= \frac{64KB}{1B} = 64K = 2^6 \times 2^{10}$

$= 2^{16} = 16$ bits

=) If size of mm = 32KB and size of memory location in of 1B then no. of bits required for addressing.

$= 32K = 2^5 \times 2^{10} = 15$

=) $256MB = 256M = 2^8 \times 2^{20} = 28$ bits

=) $16GB = 2^4 \times 2^{30} = 34$ bits

=) Let a sector in a city has 16 houses and 4 streets. Each street has 4 houses.
So, addressing of houses can be done either by giving any no. between 0 to 15 to each house. (i.e. 4 bit needed)
In this method, searching a house will be difficult.

So, in the 2nd method, if each street is given a no. and a house is given a no. in that street.
So, 1st 2 bit can be used to give street no. and last 2 bits can be used to give house no. in a particular street. So, address of house no. 2 of 3rd street will be 10 01

10 → street no.
01 → house no.

In this method, searching a house will be easy.

Page No.
Date

Similar method can be used to generate logical and physical addresses.

Let, hard disk on secondary memory (SM) is of size 16 MB and it is byte addressable.

So, $2^4 \times 2^{20} = 24$ bits will be needed to give address in addressing of SM.

So, logical address (LA) generated by CPU will be of 24 bits.

But this i.e. to access an instruction from RAM, CPU will generate 24 bit LA (which will be later converted into PA) but in this method searching will take time. So, we can divide this 24 bits in page no. and instruction within a page.

Let page size = 1KB

So, 1K instructions in a page. So, 10 bits are needed to address each instruction in a page.

$$\text{No. of pages in SM} = \frac{16\text{MB}}{1\text{KB}} = \frac{2^4 \times 2^{20}}{2^{10}} = 2^{14}$$

So, 14 bits needs to give numbering to each page.

So, now 24 bit in 24 bits of LA, 1st 14 bits will be used to number a page and last 10 bits will be used to number an instruction in a page.

now, let RAM or MM is of size 32 KB

So, no. of frames $= \frac{32\text{KB}}{1\text{KB}} = 32$

$\therefore$ 5 bits will be used to number each frame of MM.

10 bits will be used to number each instruction of a page.

So, physical address will be of 15 bits where 1st 5 bits will be used to number pages and last 10 bits will be used to number instructions.

Let, LA = 33 bits long.

PA = 24 bit long

Page size = 2KB

So, 2^{33} bytes $= 2^3 \times 2^{30} = 8GB$

Page size = 2KB = 2^{11} B

i.e. 11 bits needs to number instructions in a page.

So, 33 - 11 = 22 bits needs to number pages.

PA = 24 bit long

$= 2^{24} = 2^4 \times 2^{20} = 16MB$

∴ bits needed to number the frames = 24 - 11

= 13

In case of accessing an instruction from MM, MMU needs 1st to access the MM to find the frame no. in the page table of the running process. Second time to actual access of the instruction. Accessing two times the MM is actually slows down the operation.

CPU generates LA which contains page no. and offset. Since, execution of instructions of a process is sequential, it is highly probable that page no. in the LA generated by CPU for current instruction will be same as that of previous instruction. So, each time accessing MM for searching the same page no. in the page table is not needed can be avoided if a register hardware is maintained in the CPU which contains the page no. & and its corresponding frame no. (same as page table) for the previous instruction. This hardware is called translational lookaside buffer (TLB)

⇒ MM access time = 400 ns

TLB = 50 ns

Hit ratio of TLB = 90%

So, average memory access time = ?
(AMAT)

In case of no TLB, AMAT = 2 × 400 = 800 ns

In case of TLB = 0.9 [50 + 400] + 0.1 [50 + 2 × 400]

= 0.9 × 450 + 0.1 × 850

= 490 ns

Date

So, EMAT = h [TLB + mm]
+
(1-h) [TLB + 2 * mm]

TLB can hold the information of pages of one process at a time. In case of Context switch, all TLB data will be erased and page information of new process will start filling in the TLB. However, it will take some time to fill all the places of TLB with the page information of new process. So, in case of frequent context switch, TLB miss will increase, So, EMAT will increase. As a solution, multiple TLBs can be maintained so that in case of Context switch, Page information of old process will not be erased. In that case, when old process will again come to the 'running', no need to wait for TLB to fill, So, TLB miss will be reduced. However, multiple TLBs will increase the Cost of the system.

Context switch also occur when kernel goes to user mode to kernel mode. Here, again TLB miss will increase. In general, OS goes to kernel mode for less time and again control is given to user mode. To reduce the TLB miss, small portion of the TLB can be reserved for OS so that TLB erase is not need in case of switching from user to kernel mode.

=) mm = 64 MB
LA = 32 bits
Page size = 4 KB
find total space wasted in maintaining page tables?
maximum

Ans:- total no. of frames in mm = $\frac{\text{mm size}}{\text{page size}}$

$= \frac{64\text{ MB}}{4\text{ KB}}$

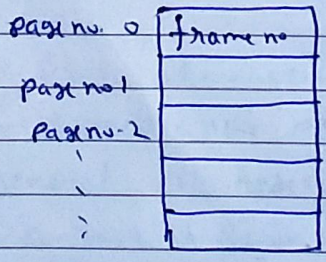
$= \frac{2^6 \times 2^{20}}{2^2 \times 2^{10}} = 2^{14}$

∴ 14 bits required to address frame no.

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

∴ 12 bits required to address offsets of a page.

A page table will look like



So, an entry in page table will need 14 bits
≅ 2 byte

total no. of pages in hard disk = $\frac{2^{LA}}{\text{page size}}$

$= \frac{2^{32}}{2^2 \times 2^{10}} = 2^{20}$

i.e. 20 bits required to address all pages.

Page No. _____
Date

∴ total no. of entries in all page tables = 2^{20}

total space wasted due to all page tables

= total no. of entries in all page tables × size of an entry in a page table

= 2^{20} × 14 bits = 2^{20} × 2 bytes

= 2 MB

⇒ MM = 256 MB
Page size = 4 KB
LA = 40 bits
Process size = 4 MB
Find total space wasted to maintain the page tables of this process?

Ans :- no. of frames in MM = $\frac{256\text{ MB}}{4\text{ KB}}$

$= \frac{2^8 \times 2^{20}}{2^2 \times 2^{10}} = 2^{16}$

∴ 16 bits are needed to address each all frames.

no. of ~~Process~~ Pages in Process = $\frac{4\text{ MB}}{4\text{ KB}}$

$= \frac{2^2 \times 2^{20}}{2^2 \times 2^{10}} = 2^{10}$

∴ total wasted due to the given Process = no. of pages in the Process × bits required for each page table entry

= 2^{10} × 16 bits = 2 KB

⇒ TLB access time = t

main memory access time = m

TLB miss rate = P

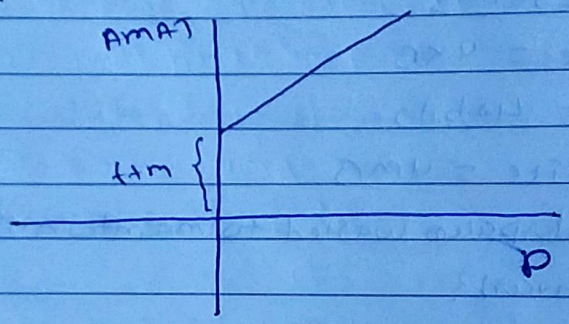
AMAT = P(t+m+m) + (1-P)(t+m)

= t+m+Pm

~~Amat - Pm = t+m~~

Pm - AMAT = -(t+m)

$$\frac{P}{-\left(\frac{t+m}{m}\right)} + \frac{AMAT}{t+m} = 1$$



So, if P increases, AMAT increases.

minimum value of AMAT = t+m

Page table entries

As we know that an entry in a page table contains a corresponding frame no. in mm. However, apart from frame no., some extra fields also exists in a particular entry of the page table.

(a) Caching disabled bit :- Sometime, mm contains fresh information than cache memory. e.g. if user is providing input in run time, so, this input will go to mm, i.e. mm has fresh page in comparison to cache memory. So, in this case, cache memory should be by-passed and directly mm should be accessed. So, in this case, caching for this particular page should be disabled.

(b) Referenced bit :- whether this page is refered in last clock cycle or not. It is used in case of demand paging (next chapter).

(c) modified bit or dirty bit :- whether the page has been modified or not. If a page has been modified then the changes should also reflected in hard disk.

(d) Present / absent bit :- whether the page looking in mm is present there or not. If not present then it is the case of page fault. In that case, the page will be brought from virtual memory to mm called as demand paging. During the der time of loading the page from VM to mm, the process will be in wait state.

(e) Protection :- page should be used in read only or r/w, or r/execute etc.

(f) Page frame no. :-

for simplicity, we used only frame no. as only field in page table.

The fields in page table however may vary from OS to OS.

⇒ Logical address space = 4MB
page size = 4KB

then what should be page table entry size so that page table will fit in one page

Ans:- Logical address space = 4MB
$= 2^2 \times 2^{20}$ B
$= 2^{22}$ B

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

∴ no. of pages $= 2^{(22-12)} = 2^{10}$
$= 1K$

Let the page table entry size is x

∴ $1K * x \leq$ page size

$x \leq \frac{4KB}{1K}$

$x \leq 4$ Byte

Page No.
Date

=) if Logical address space = 256 MB
Page table entry size = 4B

what should be page size so that page table should fit in one page or frame.

Ans: Let page size = P

$$\text{Pages} = \frac{256\text{MB}}{P} = \frac{2^{28}}{P}$$

$$\therefore \text{Page table size} = \frac{2^{28}}{P} * 4$$

Page size ?, Page table size

$$P ?, \frac{2^{28}}{P} * 4$$

$$P^2 ?, 2^{16}$$

Page No.
Date

Multilevel paging

Let Logical address = 32 bits
Page size = 4KB
Physical address = 44 bits.

no. of pages in process on hard disk = $\frac{2^{32}}{4KB}$

$= \frac{2^{32}}{2^2 \times 2^{10}} = 2^{20}$

Here, out of 32 bits of LA, 12 bits are dedicated to address each byte of a page and 20 bits are dedicated to address each page.

Let Physical address = 44 bits

∴ no. of frames in RAM = $\frac{2^{44}}{4KB}$

$= 2^{32}$

∴ out of 44 bits, 12 bits for addressing bytes in a page (12 bit offset) and 32 bits for addressing each frame in mm.

No. of entries in the page table is equal to no. of pages in the process on hard disk.
$= 2^{20}$

Page No.
Date

If we consider only frame no. as an entry in the page table then size of an entry in a page table
= 32 bit $\log_2$ (no. of frames)
$= \log_2 2^{32} = 32$ bits $=$ 4 bytes.

Since, PT (page table) is also stored in the MM i.e. in pages. So, no. of pages required to store the PT will depend on PT size.

PT size = no. of entries in PT * entry size
= no. of pages in the process * 4 byte
$= 2^{20} \times 4$ byte $= 2^{20} \times 2^2$ byte
$= 2^{22}$ byte

∴ no. of pages needed to store the PT
$= \frac{\text{PT size}}{\text{page size}} = \frac{2^{22}}{4\text{KB}} = \frac{2^{22}}{2^2 \times 2^{10}} = 2^{10}$

So, 1K pages are needed to store the PT.

So, we again need a PT to identify the page frame which is containing the portion of the actual PT which will give the exact frame no. of the required byte.
Since, no. of pages needed to store the PT is 2^{10}, so, we need a new PT where in each entry, the frame no. corresponding to these pages will be given.

Since, Page size = 4KB

So, in the new PT, 2^{10} entries will be there.

Page No.
Date

each entry is of size 4 byte (same as original PT)

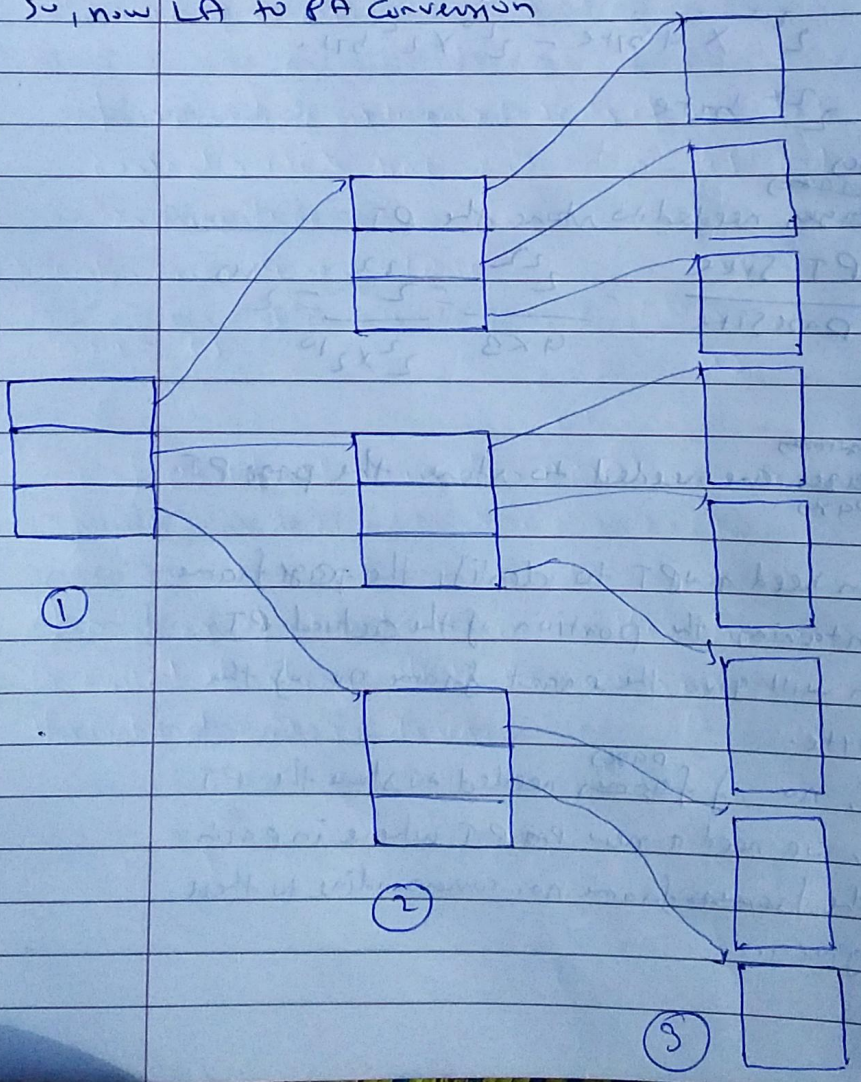
∴ Space needed to store new PT = $2^{10} \times 4$

= 4 KB

∴ the new PT can be stored in 1 page itself.

If space needed to store the new PT is more than the page size then we will need again one more level of paging i.e. one more PT.

So, now LA to PA conversion



So, page at level (1) will give the page no. at level (2) should be searched. At level (2) actual PT is there. Now, page at level (2) will give page no. at level (3) to reach to the required byte.

Since, no. of entries at page in level (1) is 2^{10}.
So, 10 bits are needed to addhess each entry of the page at level (1).

Similarly, no. of entries at page in level (2) is 2^{10}.
So, 10 bits are needed to addhess each entry of the page at level (2).

no. of entries in a page at level (3) is 2^{12}, where pages contain the actual instructions needed to be fetch by the CPU.
$\therefore$ 12 bits needs. So, 32 bit of LA will be converted into (10 + 10 + 12).

Here, role of 44 bits comes to decide the size of an entry in a page table, i.e. size of an entry in a page at level (1) and (2).

Addhess of the page at level (1) will be stored in PTBR register.

Page No.
Date

Page fault

In the PT, one entry is that whether the required page is in the mm or not. The bit is present / absent bit.

If the required page is not in mm then it is called page fault.

As soon as page fault occurs, a context switch will take place bcoz now OS kernel will take the control to bring the required page from hard disk to mm. Now, when the required page is brought to the mm by OS, the control will be again given to the user process.

If memory access time is in terms of ns then context switch time is in - - ns and time in bringing the required page - - - ms

again CS - - - ns

So, if page fault occurs then AMAT is order of ms.

If high no. of page faults occurs then this situation is called thrashing. In this case, System throughput will go down.

If p is page fault rate
m is memory accen time
t is service time on page fault time

then ~~EMAT = p*t + (1-p)m~~
EMAT = p(t+m) + (1-p)m
≈ pt + (1-p)m

Page No.
Date

Multi-level example

LA	Page size	Page table entry
48b	16KB	4B

no. of ~~pages~~ =

Page size = 16KB = $2^4 \times 2^{10}$

no. of pages = $\frac{2^{48}}{2^{14}} = 2^{34}$

∴ size of page table) = $2^{34} \times 2^2$

$= 2^{36}$ B

So, no. of pages required to store page table)

$= \frac{2^{36}}{2^{14}} = 2^{22}$

So, size of page table 2 = $2^{22} \times 4$B

$= 2^{22} \times 2^2$

$= 2^{24}$ B

no. of pages required to store page table 2

$= \frac{2^{24}}{2^{14}} = 2^{10}$

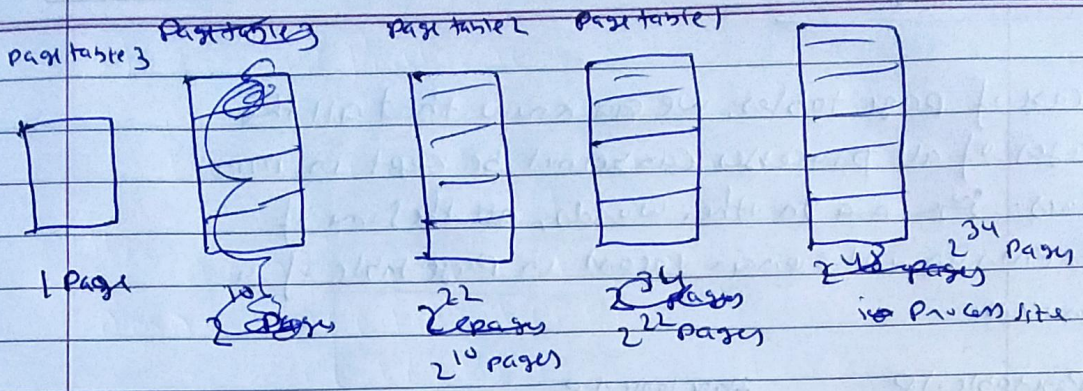
So, size of page table 3 = $2^{10} \times 4$B

$= 2^{12}$

∴ no. of pages required to store page table 3

$= \frac{2^{12}}{2^{14}} = 1$

Page No.
Date



In case of actual instruction, no. of entries in a page = 16 KB
$= 2^4 \times 2^{10}$ B
$= 2^{14}$ B
= 14 bits — (1)

In case of storing Page tables in pages, no. of entries in a page = $\frac{16\text{ KB}}{4\text{ byte}} = \frac{2^4 \times 2^{10}}{2^2} = 2^{12}$ = 12 bit

No. of Pages in the Page table 3 is 2^{10}. But in a page, no. of possible entries are 2^{12}. But since we only 2^{10} entries, so, we need 10 bits to address all pages of Page table 2 in Page table 3.

No. of Pages in Page table 1 are 2^{22}. So, all each page of Page table 2 will have 2^{12} entries. So, 12 bits will be needed to address each page of Page table 1 in Page table 2.

Similarly, 2^{34} pages are there in Page table 1. So, 12 bits will be needed to address each page of Page table 1 in Page table 2.

Finally, 14 bits are needed to address each byte in case of actual instruction as in (1). So, 48 bit of LA will be distributed as follow:

So,

10	12	12	14

Page No.
Date

Inverted Page table

In case of page tables, we can know that all the pages of all processes can be not be kept in mm always. In other words, some of the entries will remain vacant in page table of a process.

Page table of P_1

0	x
1	f_1
2	x
3	f_2
4	x
5	f_3

Page table of P_2

0	
1	
2	f_4
3	f_5
4	f_6
5	

So, Process P_1, page no. 1, 3, 5 are present in mm
P_2 - 2, 3, 4 - - - - mm

So, here we can see that lots of space is being wasted in each page table.

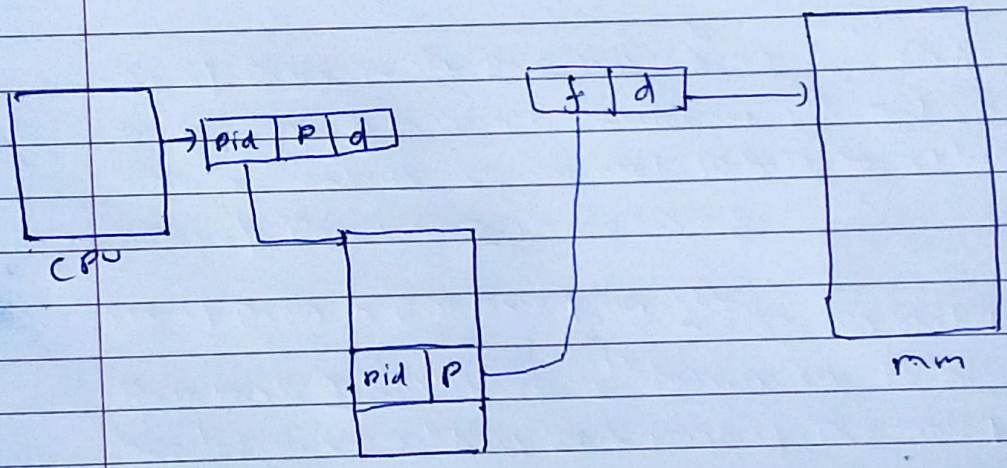
So, in place of giving allotting 1 page table to each process, let us have only one page table for all the processes. No. of entries in this page table will be no. of frames in the mm.

Date

Inverted page table

Frame no.	Page no., Pid
0	
1	1 P_1
2	3 P_1
3	5 P_1
4	2 P_2
5	3 P_2
6	4 P_2
7	
8	
9	

So, in inverted page table, for each frame no. there will be an entry in inverted page table which will contain Page no. and Process id.



Although, this scheme decreases the amount of memory needed to store each page table, it increases the amount of time needed to search the table when a page reference occurs. Because the inverted page table is sorted by frame no., so, a linear search will be needed to search a (Pid, P) in the table.