UNIT-III

memory management strategies

Introduction:

→ strategies divided into several categories

* Fetch strategies
* placement strategies
* Replacement strategies.
*

* Fetch strategies: we are taking the data.

→ it demand or anticipatores data.

→ decides which pieces of data to load next in memory management

✓ Placement strategies:

it decides where in main memory to place. incomming data.

✓ Replacement strategies:

→ it decide which data is to remove from main memory to make more space.

Introduction:

→ memory is a large array of words or bytes each with its own address.

→ the CPU ~~fetches~~ fetches instructions from memory according to the value of the program counter

→ First fetches an instruction from memory. The instruction is then decoded and may cause operands to be fetched from memory. after the instruction has been executed on the operands, results may be stored back in memory.

Address binding:-

→ a program resides on a disk as a binary executable file. To be executed the program must be brought into memory and placed within a process.

→ the process may be moved between disk and memory during its execution.

→ addresses may be represented in different ways during these steps.

~~→ address may be reprented in different ways~~

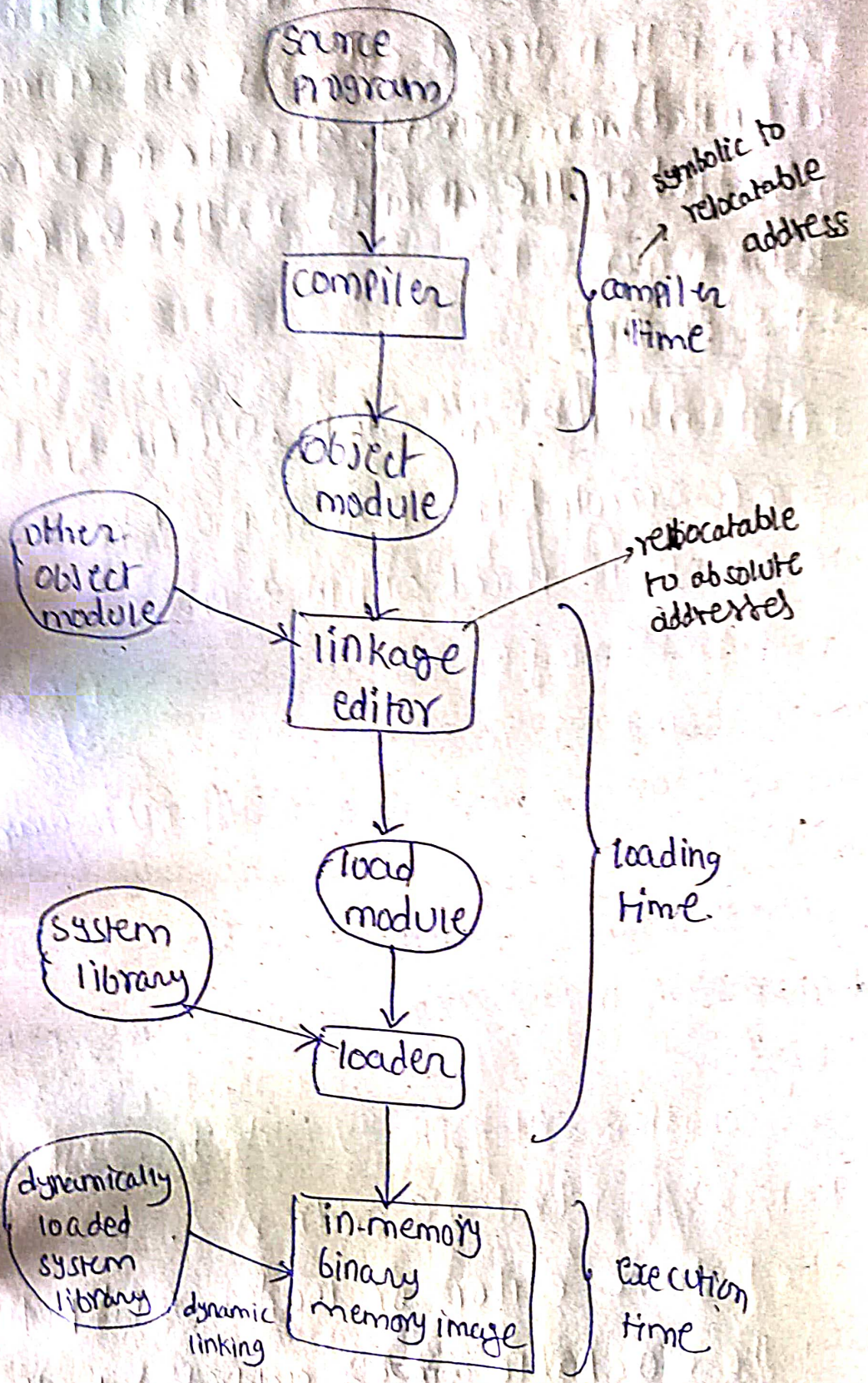
→ addresses in the source program are generally symbolic.

→ a compiler will typically bind these symbolic addresses to relocatable addresses.

→ the linkage editor or loader will in turn bind the relocatable ~~address~~ addresses to absolute addresses

→ each binding is mapping from one address space to another.

* compile time
* load time
* execution time.



Logic- Fig: multistep processing of a user program.

Logical versus physical address space

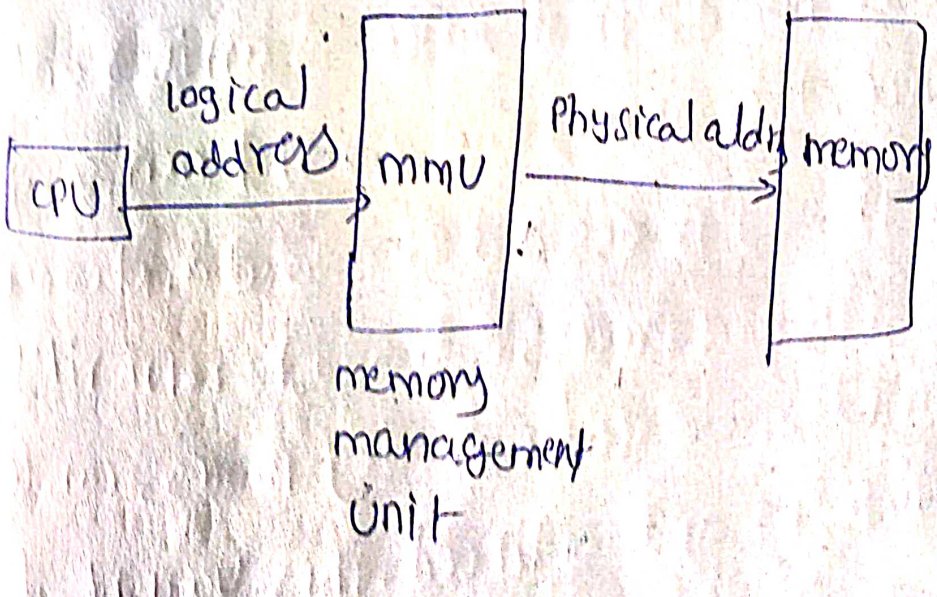
→ The address space is generated by the CPU is called logical address space.

→ The address space is generated by the memory unit is called physical address space.

Logical	Physical
→ generated by the CPU; it also reffered as virtual address	→ address seen by the memory unit
→ the logical address space is the set of all logical address generated by the program.	→ Physical address space is the set of all physical address generated by the program.

⇒ logical & physical address are the "same" in compile time and load time address binding scheme

→ the logical and physicall address are different in execution time



Swapping:

→ moving data from one device to another device temporarily.

→ It is mechanism in which a process can be swapped temporarily out of main memory to secondary memory. Late again swaps back to main memory.

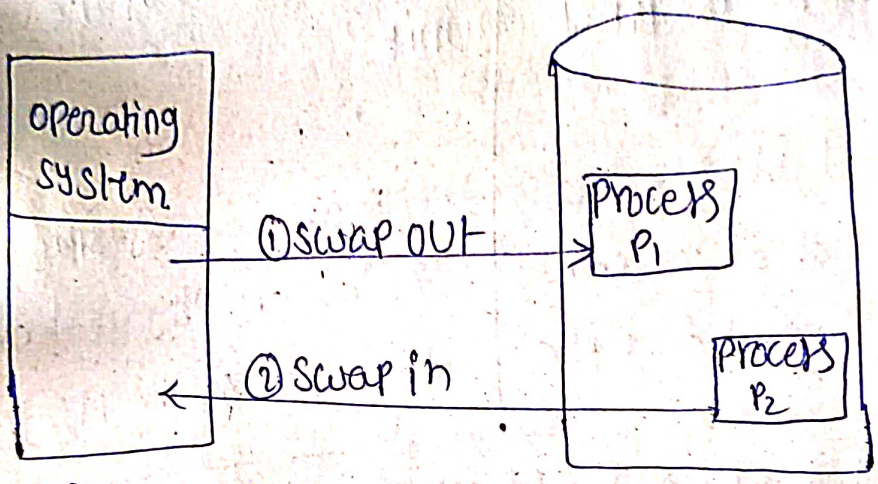
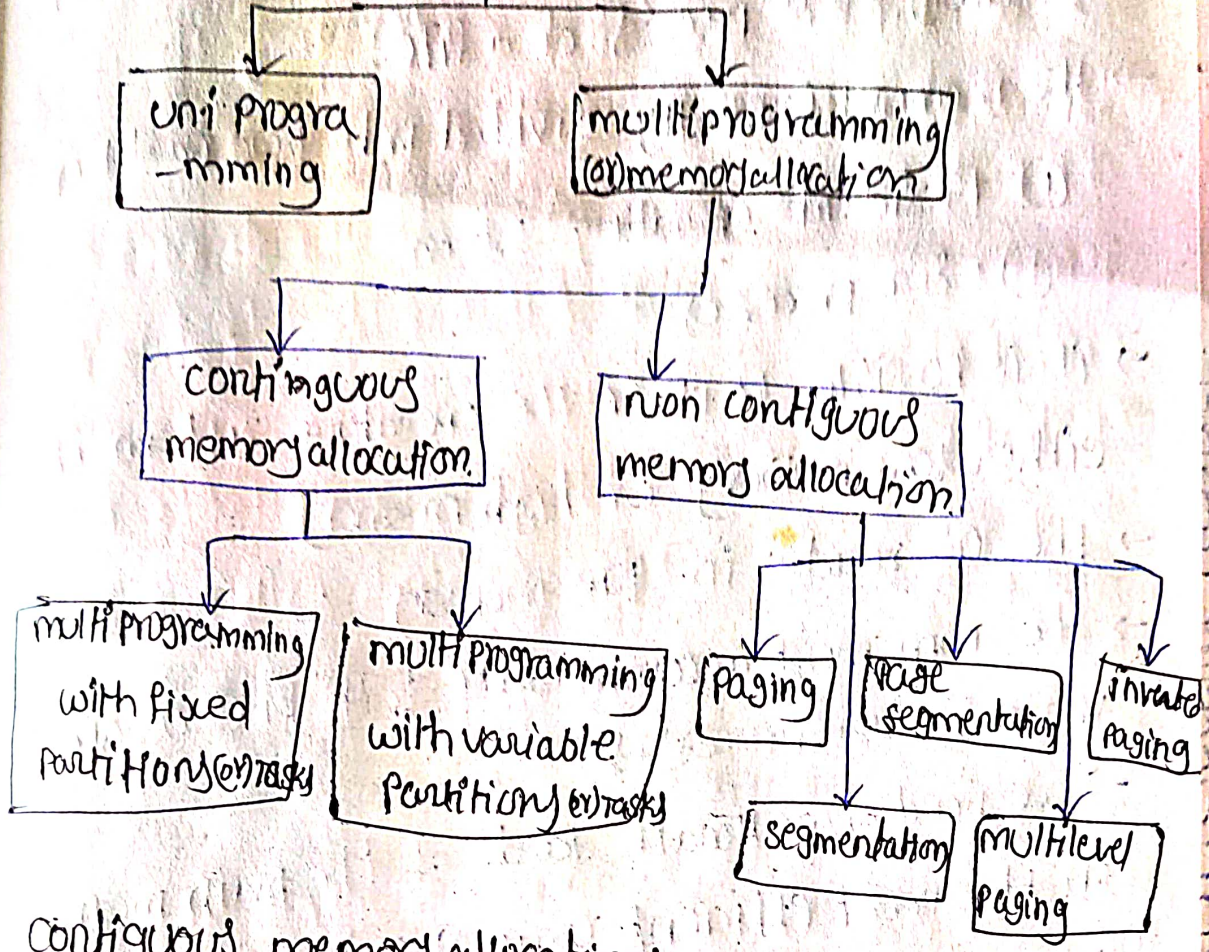


fig: swapping of two processes

Ex: Swapping policy is used for priority based scheduling algorithms. if higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. when higher priority process finishes, the lower priority process can be swapped back in and continued. This variant of swapping is sometimes called swapout (or) roll out nd swapin (or) roll in.

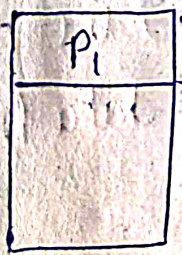
memory allocation ⇒ memory management techniques



Contiguous memory allocation:

The operating system can allocate a ~~single~~ single continguous block of memory for the processes

Ex: the process is P_1 it having 5 mB memory allocated by the OS.



This are two types:

1. multiprogramming with partitions (or) tasks
2. multiprogramming with variable partitions (or) tasks

→ contiguous memory allocation is a memory allocation method that allocate a single contiguous section of memory to a process or a file.

→ the memory is usually divided into two partitions
  ① Resident operating system
  ② User processes

→ it is possible to place the operating system in either low memory or high memory

→ the OS resides in low memory

→ the mm must accommodate both the OS and the various user processes

→ we need to allocate parts of the main memory in the most efficient way possible, this section explain one common method, contiguous memory allocation.

* memory protection

→ if the os is residing in low memory and the user processes are executing in high memory, we need to protect the operating system code and data from changes by the user processes

→ we also need to protect the user process from one another

→ we can provide this protection by using a relocation registers

→ the relocation registers contains the value of the smallest physical address.

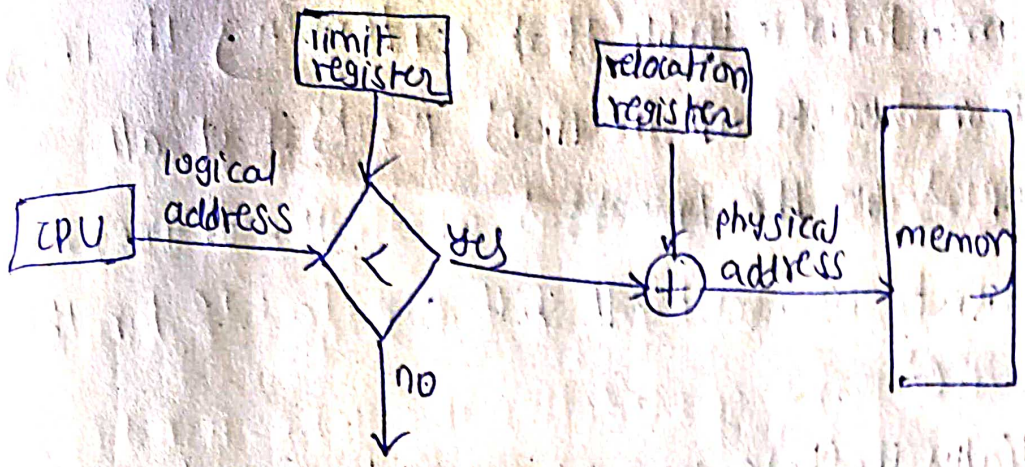


Fig: hardware support for relocation and limit registers

* memory allocation:

whenever a process is request to execute that process must be stored into a variable memory i.e RAM

→ this are two types

① multiprogramming with fixed tasks (MFT).

② multiprogramming with a variable no of tasks (MVT)

① MFT (static)

→ MFT in the fixed size partition, the memory is divided into fixed sized blocks and each block contains exactly one process. But the fixed sized partition will limit the degree of multiprogramming as the no of the partition will decide the no of processes.

MVT (dynamic)

MVT in the variable size partition method, the OS maintain a table that contending the information about all memory parts that all occupies by the processes and all ~~mer~~ mem information about all still available for the processes.

→ initially the hole memory space is available for user processes as a large block called hole.

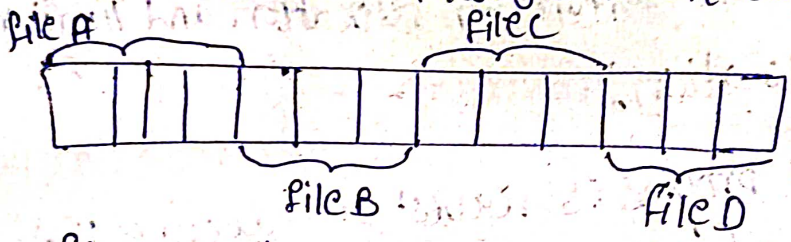


fig: contiguous memory allocation of 4 files

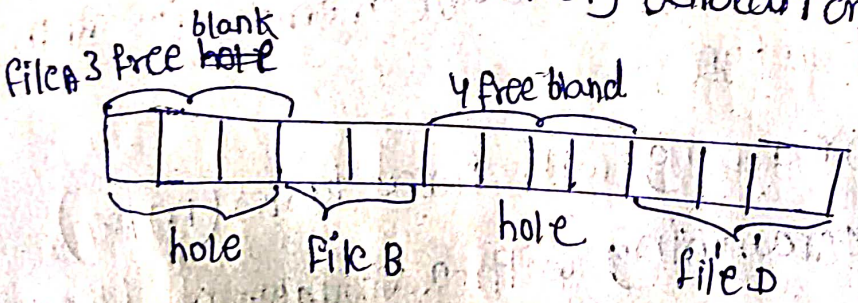


fig: when the file A and C terminates the release the memory creating hole.

Dynamic storage allocation:

→ ~~the~~ 1. First fit
* best fit
* worst fit

This strategies are the ones most commonly used to select a free hole from the set of available holes

* First fit:

→ allocate the "first hole" that is ~~beig~~ big enough.

→ searching can start either at the beginning of the set of holes or at the location where the previous first fit search ended.

* Best fit:

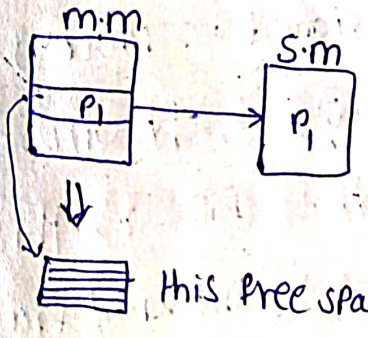
allocate the "smallest hole" that is big enough.

* worst fit:

allocate the "largest hole".

Fragmentation:

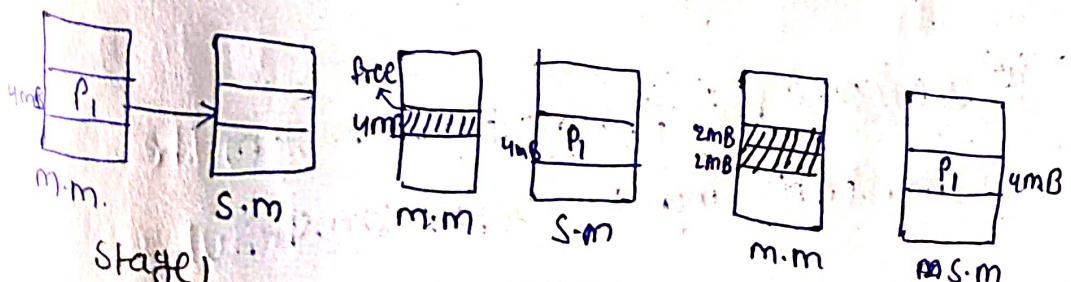
→ a processes are loaded & removed from memory the free memory space is broken into little pieces.



the P_1 process can move from main memory to secondary memory.

This free space broken into pieces.

→ after some times that processes can not be allocated to memory because of small size. and the memory block ~~is~~ remains "unused" this problem is called fragmentation.



→ The fragmentation can be divided into 2.

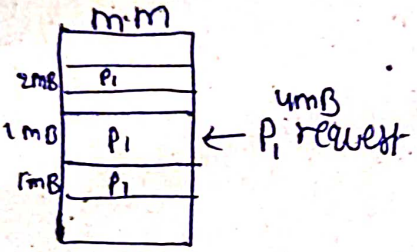
① External fragmentation

② Internal fragmentation

① external fragmentation

total ~~RAM~~ memory space is enough to satisfy a request or reside a process in it but it is not contigeous so it can not be used. then it called external fragmentation

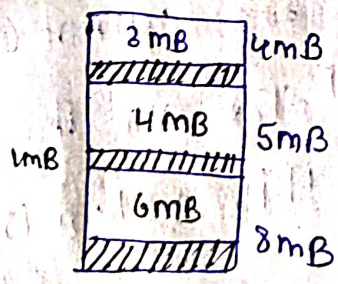
Ex:



① Internal fragmentation

→ The memory block assigned to process is bigger and some portion of memory is left unused, as it cannot be used by another process. this wastage of main memory is called as internal fragmentation.

Ex:



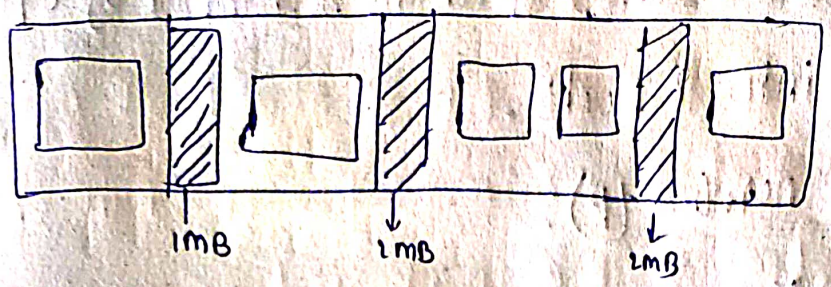
$P_1 = 4mB$

$P_2 = 6mB$

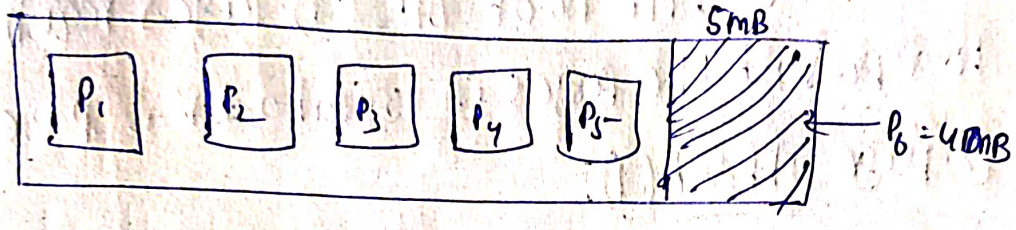
$P_3 = 3mB$

→ it overcome this fragmentation problem the fragmentation memory compact (or) shefful

Fragmented memory before compaction



Fragmented memory after compaction



→ external fragmentation can be reduced by compaction or shuffle free memory together is one large block.

→ internal fragmentation can be reduced by effectively assigning the smallest partitions but large enough for the process.

Paging

→ Paging is a memory management technique in which process address space (logical memory) is broken into blocks of the same size called pages.

→ The size of the process is measured in the no. of pages

→ Similarly, main memory (physical memory) is divided into small fixed sized blocks of memory called frames and the size of a frame is same as that of page.

→ Optimum utilization of the main memory and to avoid external fragmentation

→ The logical address space is generated by CPU is divided into, 2 parts.

① Page number (p):

The page number is used as an index into a page table. The no of bits (or) pages required to represent the pages in logical address space. (or)

② page offset (d):

no of bits required to represent particular page size of logical address (or) page offset.

Page number (p)	page offset (d)

→ The physical address space is generated or divided by the memory. This are two parts.

① Frame number (f)

The no of bits required to represent the frame of Physical address (or) Frame number

② Frame offset (d):

The no of bits required to represent particular frame or frame size of Physical address.

Frame number (f)	Frame offset (d)

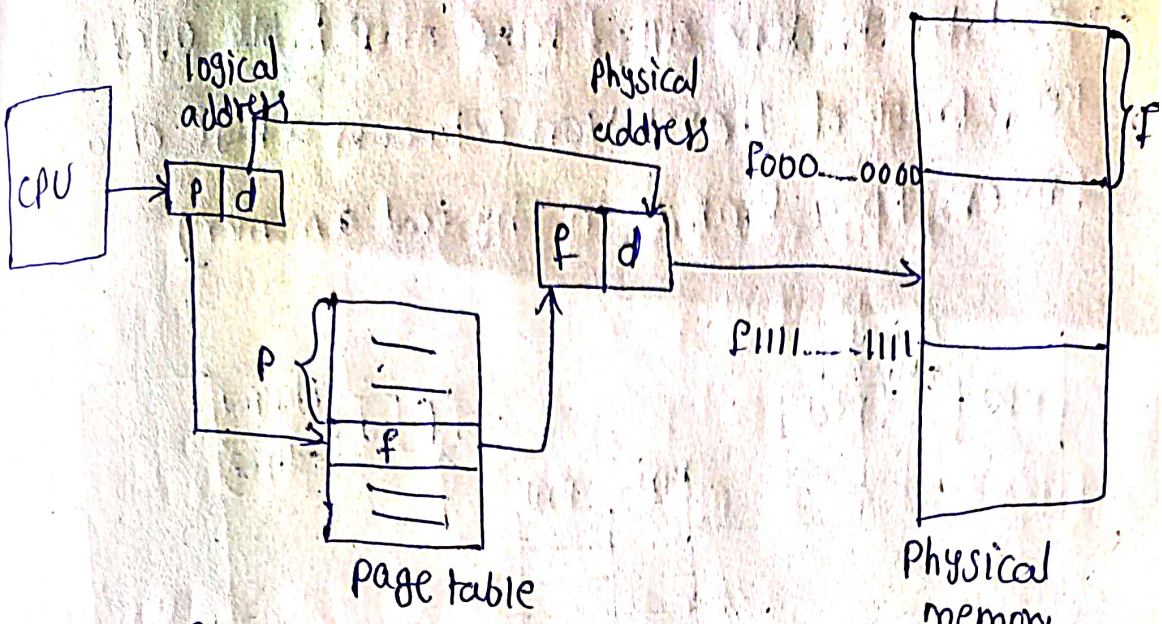
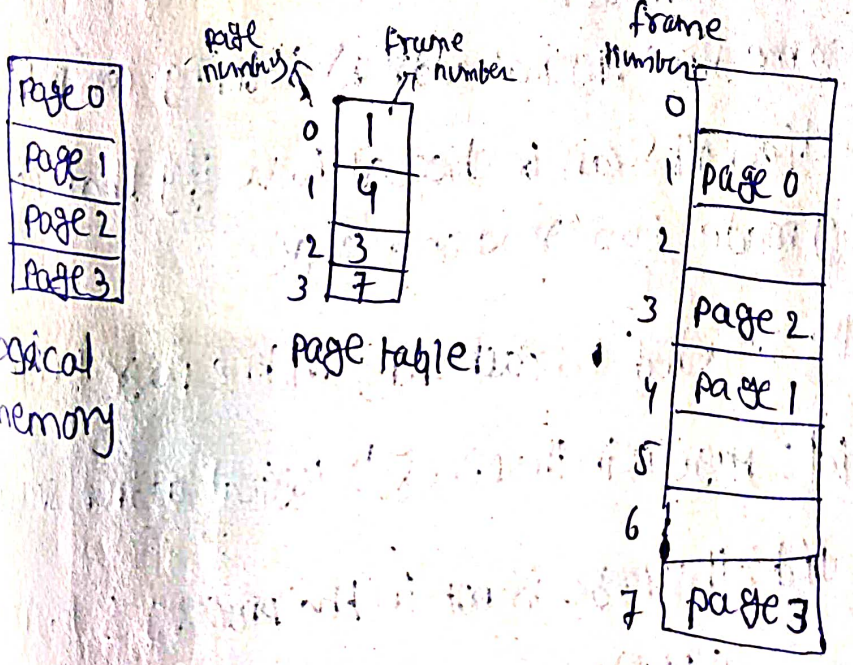


Fig: Paging hardware

→ The page number is used as an index into a page table. The page table contains the base address of each page in physical memory.

→ This page base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



Logical memory

Page table

Fig: Paging model of logical and physical memory.

Hardware support

→ If available that particular process will execute

→ a data structure called page map table used to keep track of the relation between a page of a process to a frame in physical memory

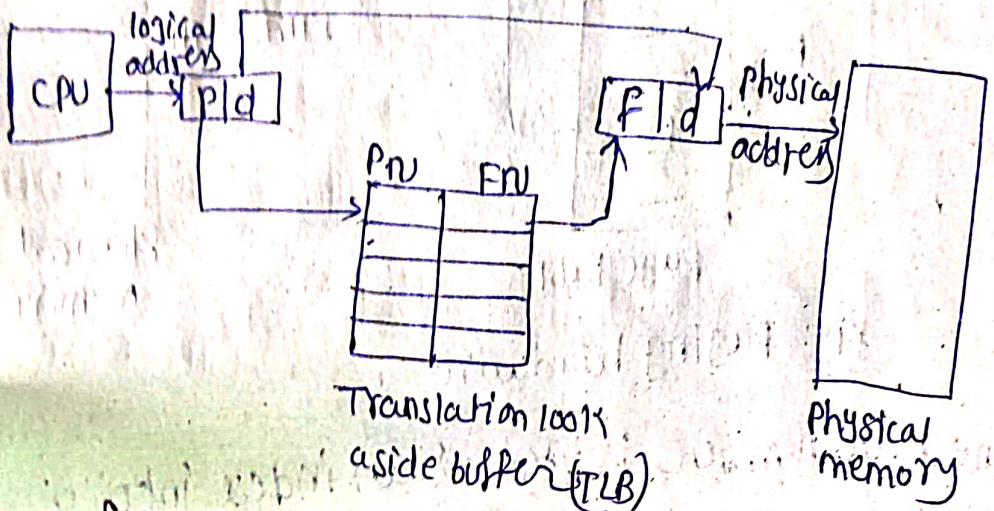


Fig: Paging hardware with TLB

Protection:

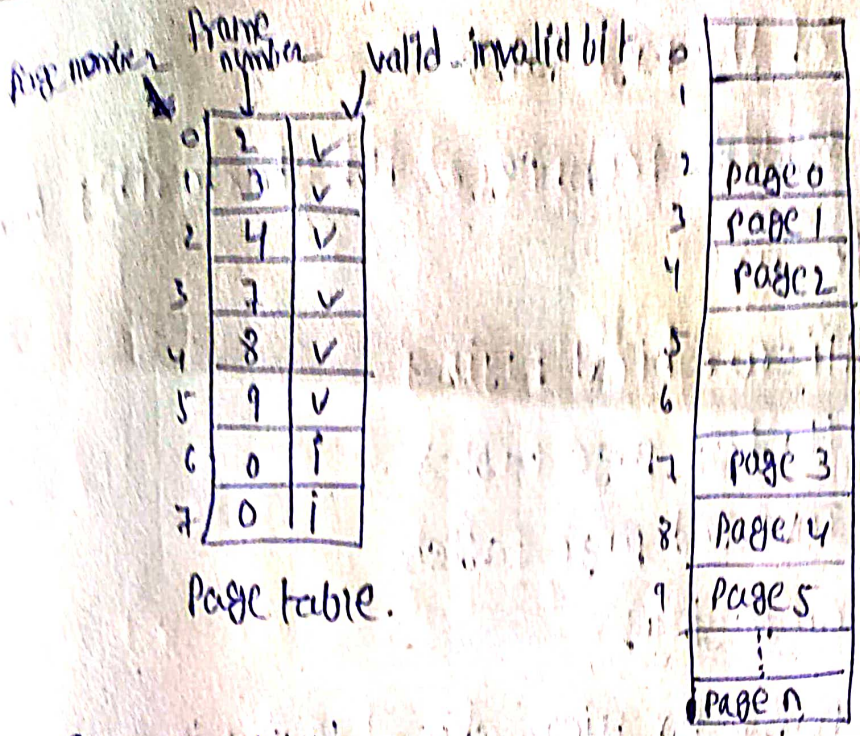
a page to be read-write or read only: every reference to memory goes through the page table to find the correct frame number.

→ at the same time physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read only page.

→ Generally attached to each entry in the page table

① valid: Page is in the process's logical address space

② invalid: the page is not in the process's logical address space.



Page table.

Fig: valid(v) or invalid(i) bit in a page table.

Advantages & disadvantages:

* Paging reduces external fragmentation but still suffer from internal fragmentation
* it simple to implement
* swapping becomes very easy because equal size of pages and frames
* it is not suitable for small RAM's.
* Paging is simple to implement an efficient memory management technique.
* page table requires extra memory space, so may not be good for a system have small RAM.

Structure of the Page Table

The most common techniques for structuring the page table are

* Hierarchical Paging
* Hashed page tables
* Inverted page tables.

* Hierarchical Paging:

→ another name for Hierarchical paging is multiple level paging.

→ The page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.

→ in this type of paging the logical address space is broken up into multiple pagetables.

→ 16 bit page table is used.

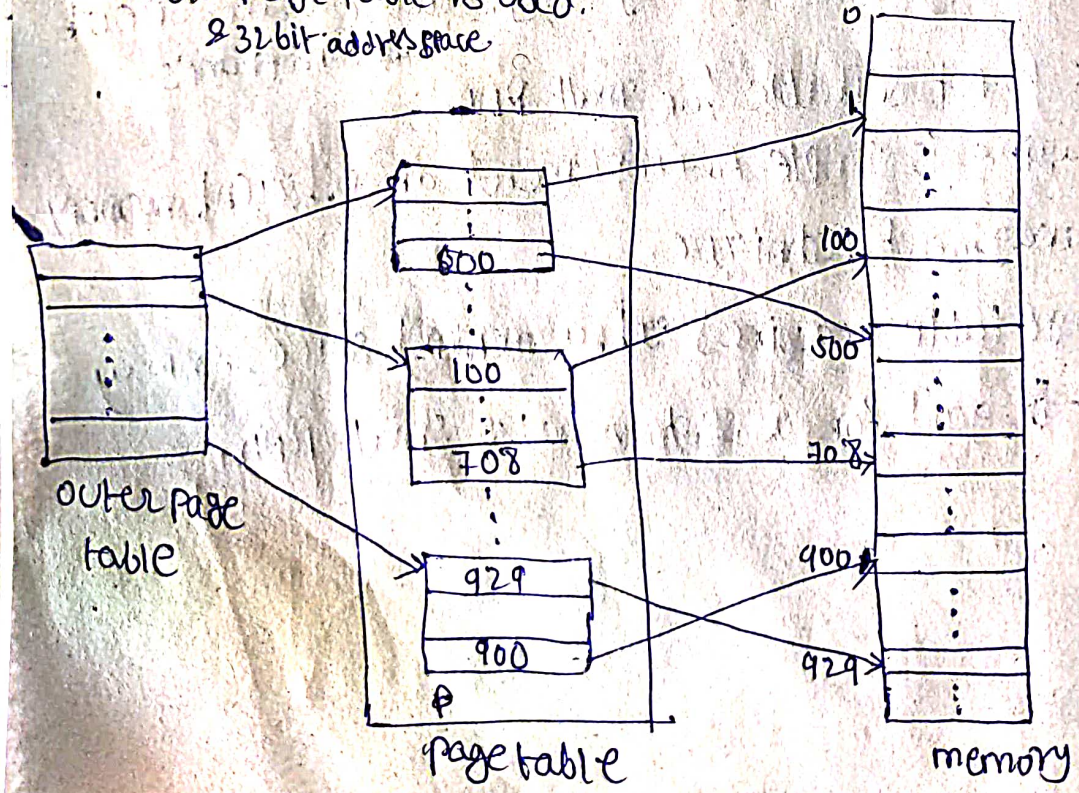


Fig: Two level Page table scheme

Hashed page tables:-

This approach is used to handle address spaces that are larger than 32 bits

each element mainly consists of

* the virtual page number
* the value of the mapped page frame
* a pointer to the next element in the linked list.

→ In this virtual page, the no of hashed into a page table. the page table mainly contains a chain of elements hashing to the same elements.

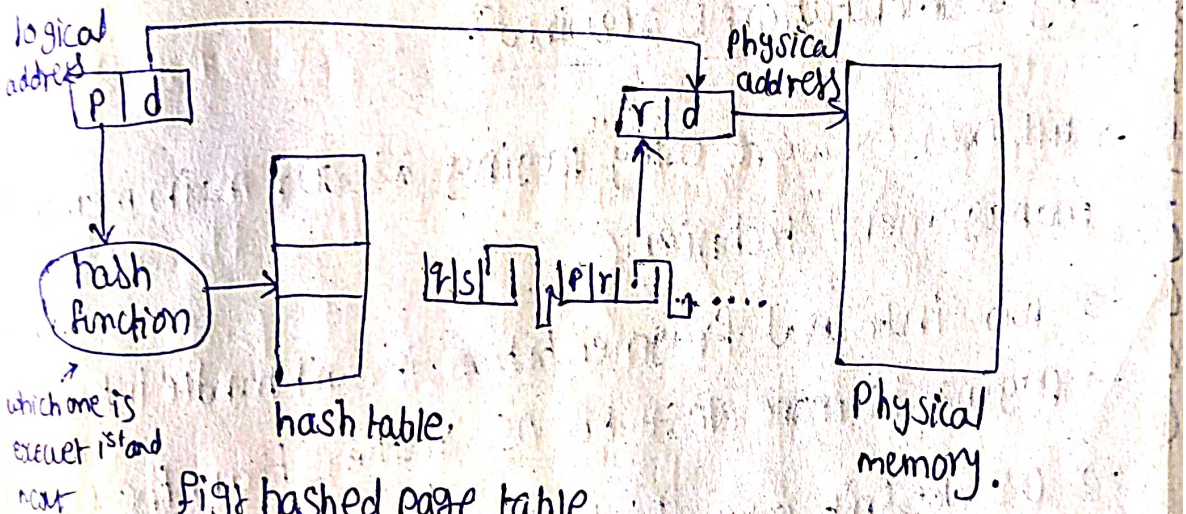


Fig: hashed page table

* Inverted page table:-

The inverted page table basically "combines a page table and a frame table into a single data structure"

→ this technique decreases the memory to store each page table; but it also increases the time to search the table.

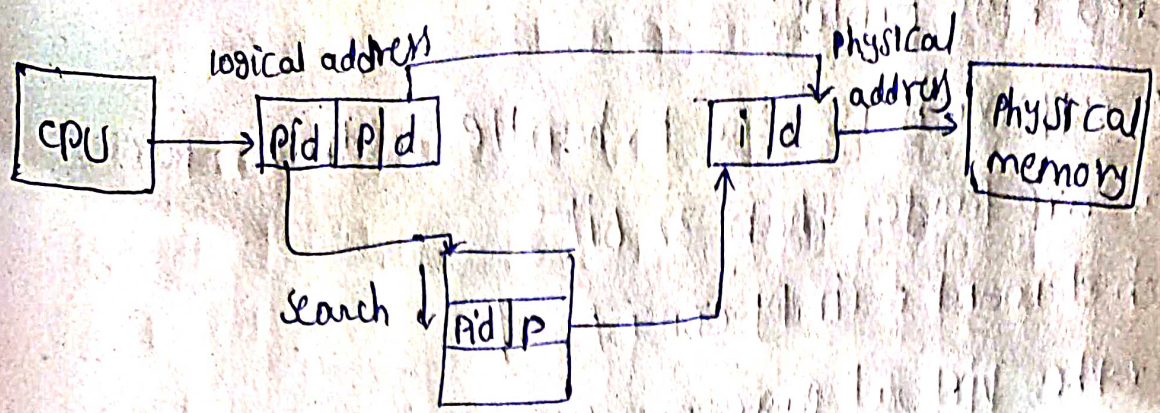


fig: inverted page table

advantage:

decreased the memory to store each page table.

disadvantage:

inverted page table create to take more time so increases the time combine the page and frame table

Why segmentation is required?

→ Till now, we were using "paging as our main memory management" technique.

→ No internal fragmentation: it can't handle internal fragments

→ Paging is more close to OS rather than the user

→ OS doesn't care about the user's view of the process

→ it is better to have segmentation which divide's the process into the segments.

segmentation

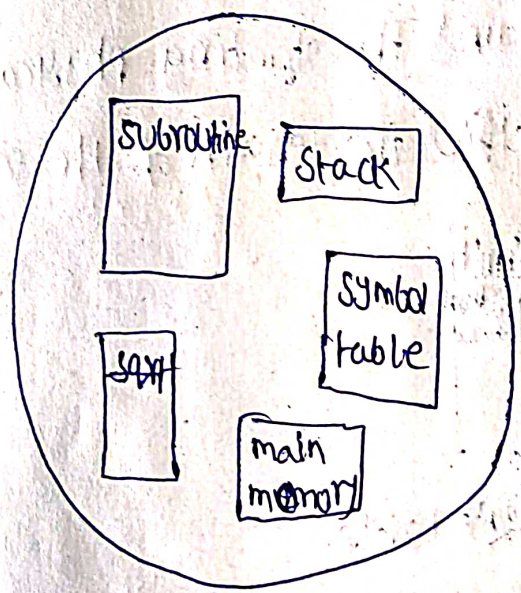
→ In operating systems, segmentation is a memory management technique in which, the memory is divided into the variable size parts. each part is known as "segment"

→ the details about each segment are stored in a table called as segment table.

→ segment table contains mainly two information about segment:

① Base:- it contains starting physical address. it is the base address of the segment

② limit:- it is the length of the segment.



Logical address

fig: User's view of a program.

→ A program is a collection of segments

→ a segment is a ~~collect~~ logical unit such as main program, procedure, function, local variable, global variable, commonblocks, stacks, symbol table,

→ Segmentation is a memory management scheme that supports this user view of memory.

→ a logical address space is a collection of segments.

→ each segment specifies the 2 quantities

ⓐ segment name (or) number

ⓑ segment offset.

→ logical address space consists of a two tuples.

<segment-name, offset>

→ Sigment table maps two dimensional physical address.

① base :- it contains the starting physical address

② limit: it specifies the length of the segment

base	limit
address of seg	length of segment

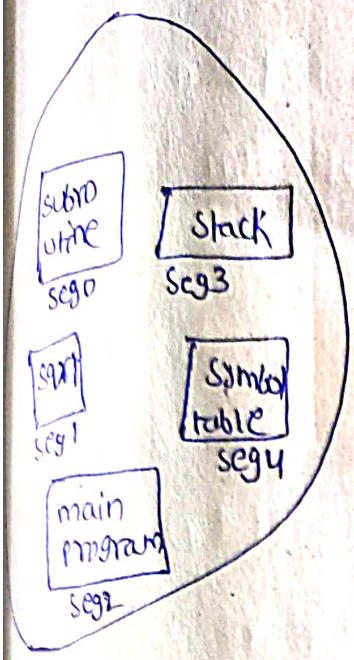
fig segment table

→ segment table base register (STBR). it points to the segment tables location in memory

→ segment table length register (STLR). it indicates no. of segments used by program

STLR

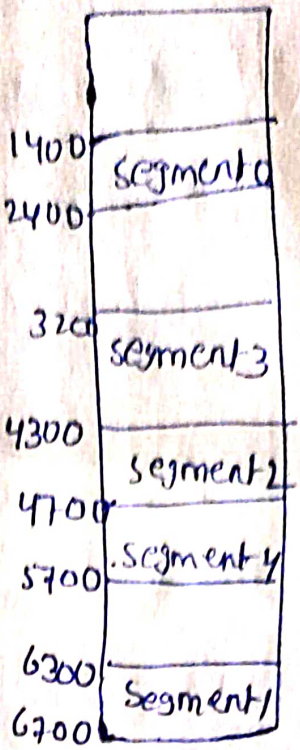
Example:-



logical address space

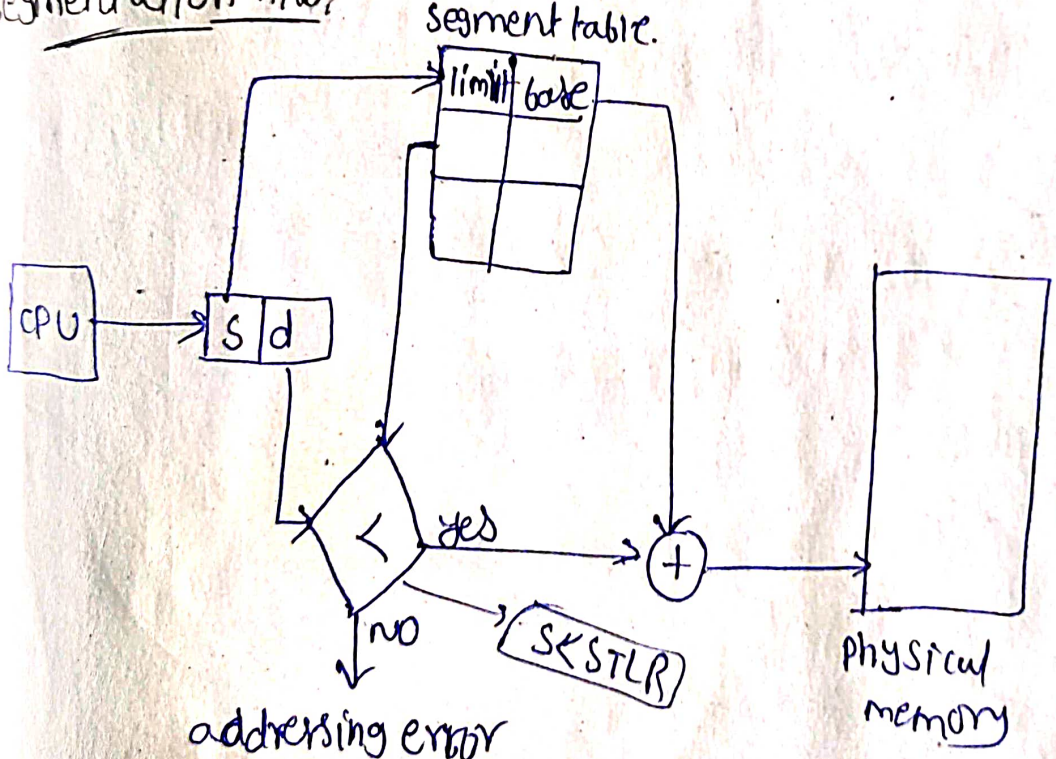
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segmentation table



Physical memory

Segmentation H/W:



Unit-III
Part-2

Virtual memory management

Virtual means:-

Virtual is some as "real" but "not real". (or) "imaginary" some part of the secondary storage memory is acting as main memory.

Ex: mirror it will display your image it is not a real it is a virtual i.e it just gives illusion to user.

memory means:-

Every task is executed with in the main memory.

Introduction:-

Virtual memory is a technique in which it is executed may not be completly in main memory.

→ Virtual memory capability is double of the physical memory.

Ex: 16 GB ⟺ 4GB

users use V.M real memory of computer

a programmer can write a program which requires more than the capacity of main memory such a program is executed by virtual memory technique.

→ more programs can running concurrently by V.M.

→ less I/o needed to load or swap process.

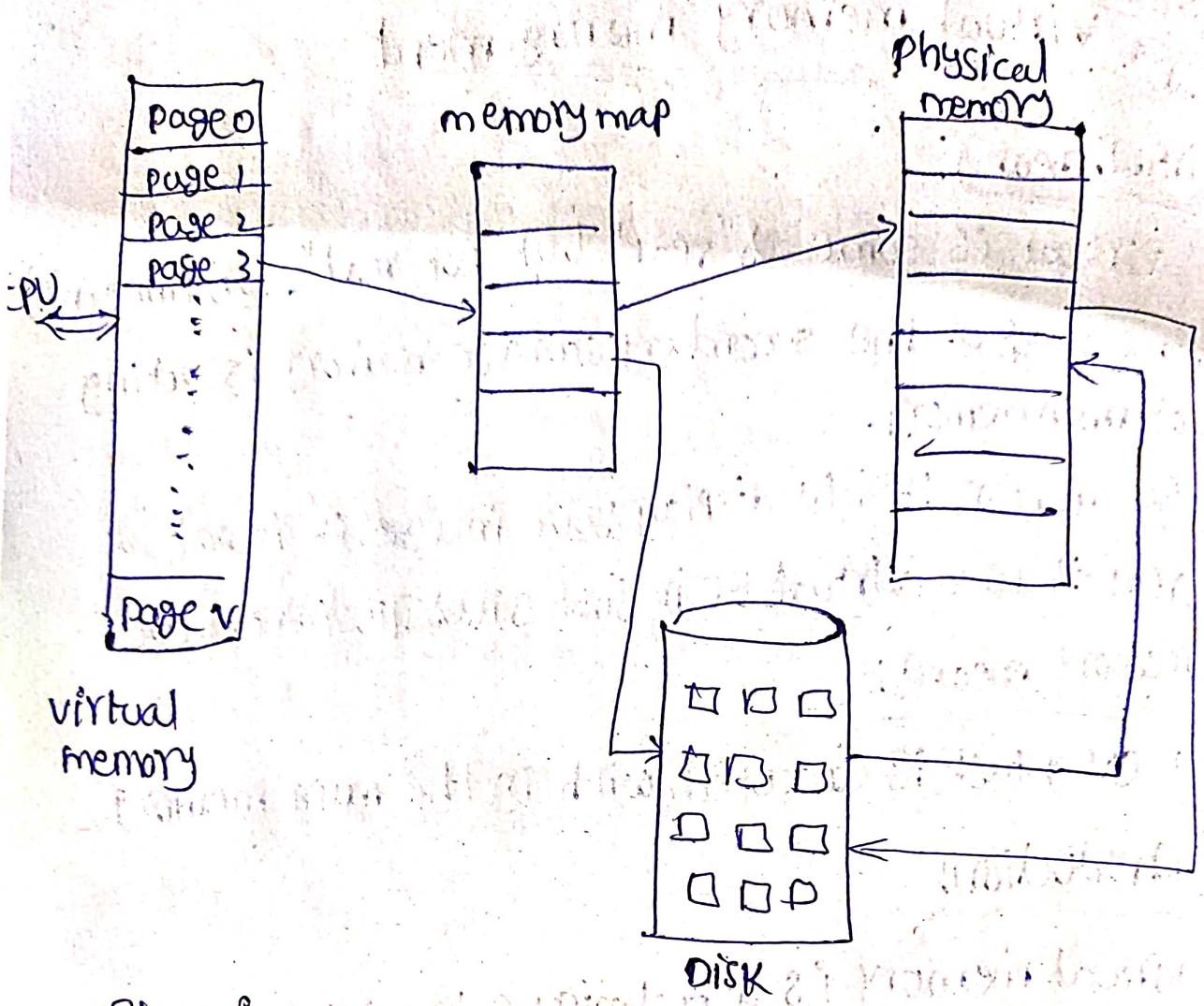


Fig: Virtual memory that is larger than Physical memory.

Demand Paging:

→ virtual memory is commonly implemented by demand paging

→ demand paging is a method of virtual memory management

→ In a system that uses demand paging the operating system copies a disk page into physical memory.

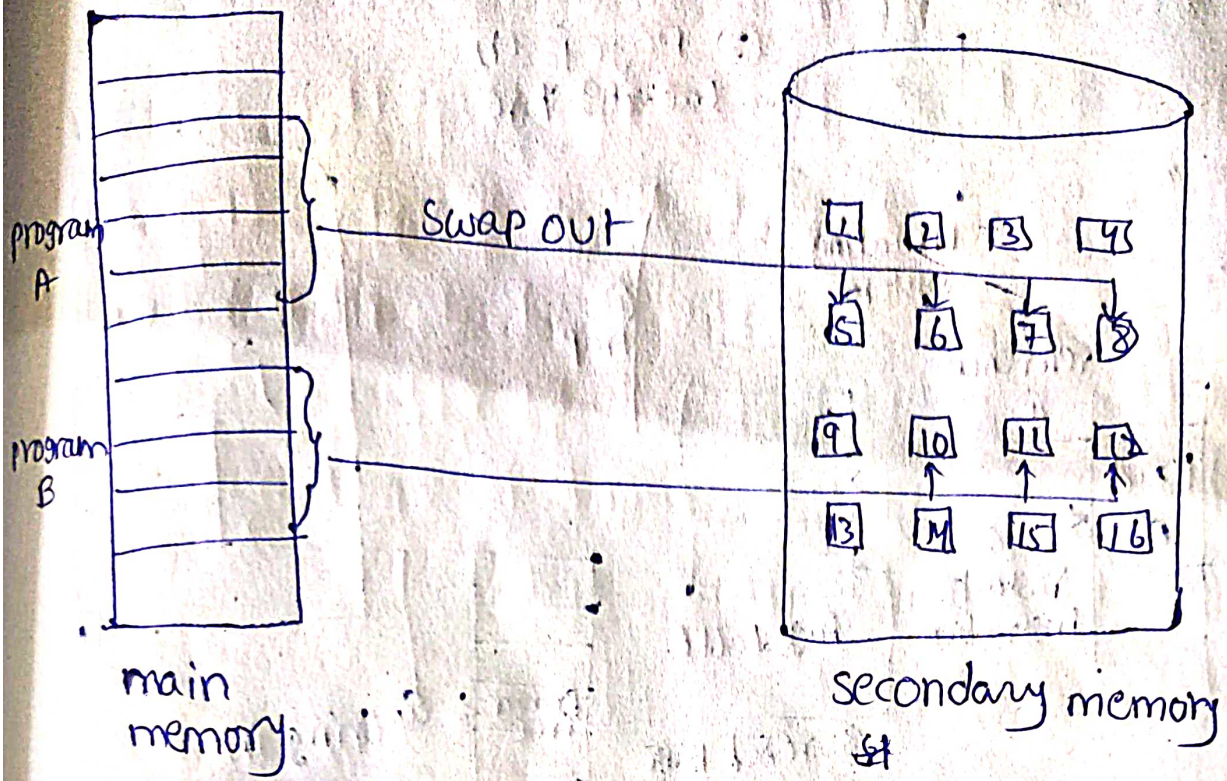


Fig: Transfer of a paged memory to contiguous disk space.

→ if the referred page is not present in the main memory then there will be a miss and the concept is called "page miss (or) page fault".

→ the CPU has to access the missed page from the secondary memory.

→ if the no. of page fault is very high then the effective access time of the system will become very high.

→ if page fault occure then our CPU can run effectively.

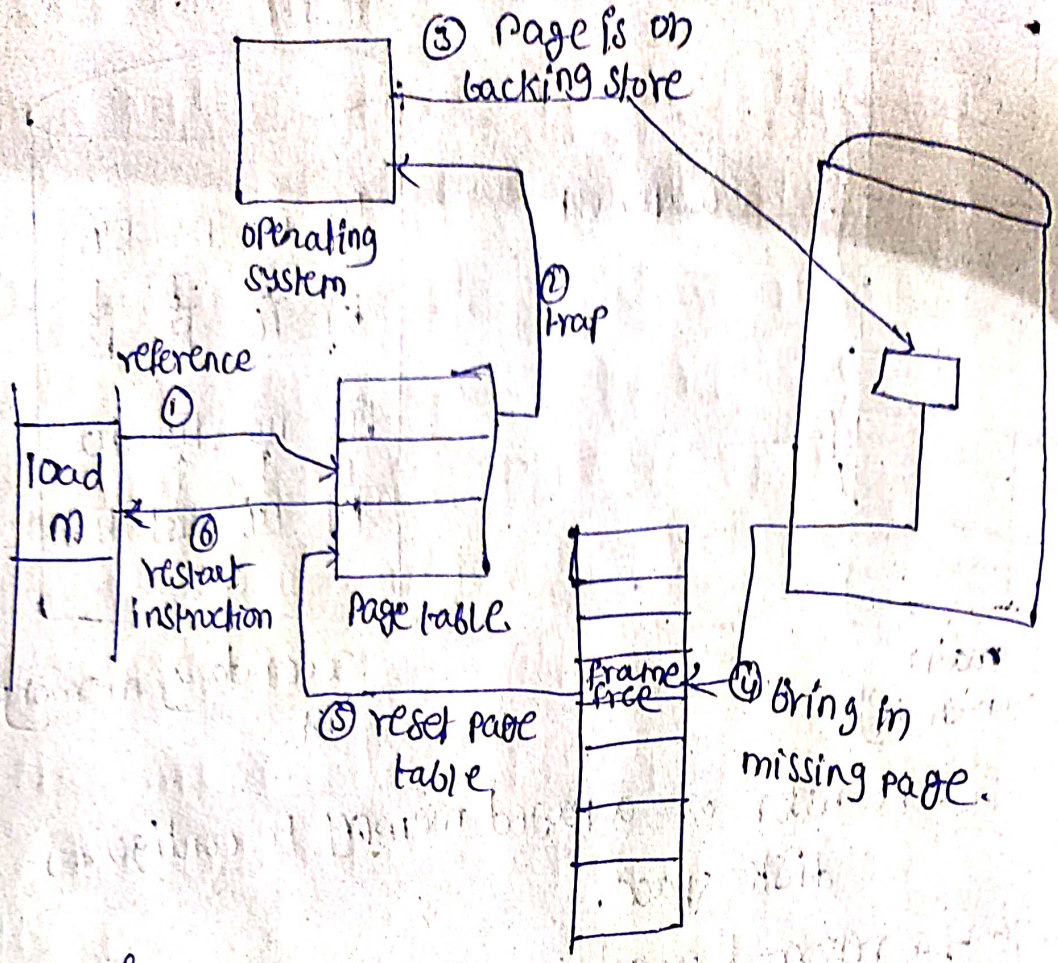


fig: steps in handling a page fault.

copy on write

→ copy on write (or) simply "cow" is a resource management technique.

→ one of its main use is in the implementation of the fork system call in which it shares the virtual memory (page) of the OS.

→ the idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages

will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and ~~these~~ thus not affecting the other process

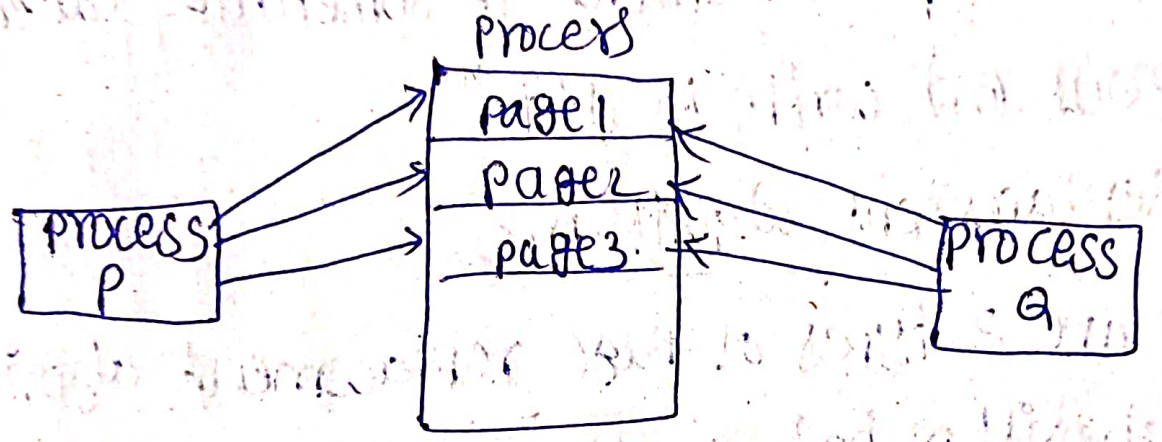


Fig: Before process P modifies page 3

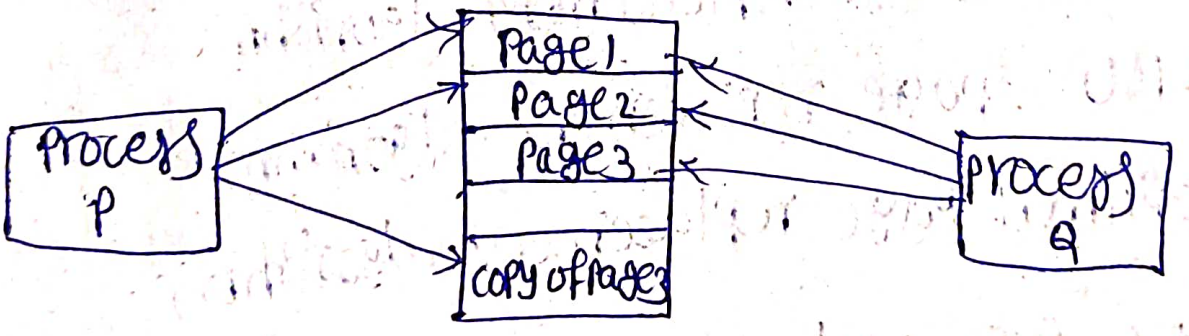


Fig: after process P modifies page 3.

Page Replacement:

→ Page replacement is done when the requested page is not found in the main memory (page fault)

→ These are page replacement algorithms decide which memory page is to ~~re~~ replaced

→ The process of replacement is sometimes called "swap out (or) write to disk."

Page Replacement algorithm:

→ These are 3 types of page replacement algorithm each algorithm has a efficient method to replace the pages.

① FIFO page replacement algorithm.
② LRU page replacement algorithm
③ Optimal page replacement algorithm

① FIFO page replacement algorithm:

→ FIFO (First come first out) the simplest page replacement algorithm

→ when a page want to replaced, the oldest page is chosen. we can create a FIFO queue to hold all page in memory.

Ex: To illustrate the problems that are possible with a FIFO page replacement algorithm.

We consider the 20 reference string and 3 frames in the memory.

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

initially the 3 frames are empty.

page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
frame 2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
frame 3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
	m	m	m	m	H	m	m	m	m	m	m	H	H	m	m	H	H	m	m	m

m = miss → page fault → we are take the page number is not match

H = Hits → the page number is match

Hit ratio = $\frac{\text{total no of hits}}{\text{total no of pages}} = \frac{5}{20} = 0.25$

miss ratio = $\frac{\text{total no of miss}}{\text{total no of pages}} = \frac{15}{20} = 0.75$

→ In FIFO, if the item is not there in memory then we call it has "miss", and that item will be taken from secondary memory.

→ if the item is in main memory then we call it has "hit"

② LRU page replacement algorithm:

→ Least Recently used (LRU) replacement associates with each page the time of that pages last use.

→ when a page must be replaced, LRU choosen that page that has not been used for the longest period of time.

Ex: given reference strings (pages) are 20 as follow and no. of frames are 3.

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

Sol

page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
F2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
F3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
	m	m	m	m	H	m	H	m	m	m	m	H	H	m	H	m	H	m	H	H

Hits = 8, miss = 12.

$$\text{Hit ratio} = \frac{\text{total no. of Hits}}{\text{total no. of Pages}} = \frac{8}{20} = 0.4$$

$$\text{miss ratio} = \frac{\text{total no. of miss}}{\text{total no. of Pages}} = \frac{12}{20} = 0.6$$

optimal page replacement algorithm

most frequently used (MFU), most Recent used (MRU)

ex: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
m	m	m	m	H	m	H	m	H	H	m	H	H	m	H	H	m	H	m	H	H

more 7 is using after a long time so, 7 will be placed by '2'. where as 0 and 1 will be using in next upcoming

Optimal Page replacement gives best solution among FIFO and LRU

Hit = 11, Hit Ratio = $\frac{\text{Hit}}{\text{total pages}} = \frac{11}{20} = 0.55$

miss = 9, miss Ratio = $\frac{\text{miss}}{\text{total pages}} = \frac{9}{20} = 0.45$

we can see more no. of Hits is optimal page replacement

Frame allocation:-

There are two important tasks in virtual memory management: a page replacement strategy and frame allocation strategy.

↓

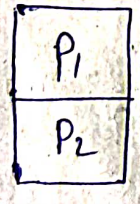
There will be no wastage of main memory and CPU.

→ Frame allocation can be divided into 5 types.

1. equal allocation
2. Proportional allocation
3. Priority allocation
4. Global allocation
5. Local allocation.

(1) equal allocation:-

The main memory size = 40



P_1 = 10 frames

P_2 = 30 frames

→ Here we have taken only 2 process, so memory is divided into 2 parts i.e

P_1 can take 20 frames

P_2 can take 20 frames

So we use can allocate to 10 frames in P_1 where as 10 will be wasted and in P_2, it can allocate only

o frames remaing 10 frames cannot allocate in memory.

, If we do equal allocation then we have a disadvant age. so we move on to next allocation.

② Propositional allocation:

the propositional allocation we have to take the frames based upon the process size ~~the process it will size of the proce~~ Ex) $P_1 = 10$, $P_2 = 30$

Proportional allocation $(P_1) = \frac{P_1}{P_1+P_2} * (\text{memory allocation size})$

$$P_1 = \frac{10}{(10+30)} \times 40 \Rightarrow \frac{10}{40} * 40 = 10$$

Proportional allocation $(P_2) = \frac{P_2}{(P_1+P_2)} * (\text{memory allocation size})$

$$P_2 = \frac{30}{(10+30)} * 40 = \frac{30}{40} * 40 = 30$$

③ Priority allocation:

The frame allocation is depending on the priority. which process will be executed first and which process will be executed next.

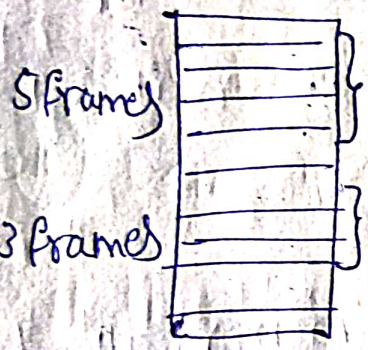
④ Global ~~replacement~~ allocation:

This is mainly useful for performing so we know what is replacement. i.e

whenever we want to use a page, mm checks, if the page is available in main memory, then it will

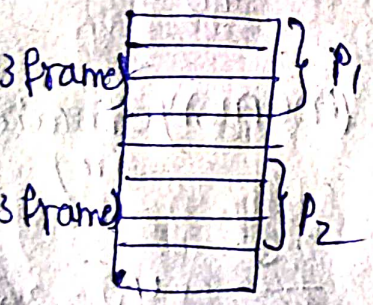
execute, and if the page is not occured in mm then we say page fault has occured, then it has to get that particular page from secondary memory to main memory.

for goblal allocation:



in this, we can allocate a page from secondary memory to main memory in any of the process.

for local allocation:



In this we has to allocate a page which is necessary for P_1 in that particular space itself.

Trashing:-

if a process spends more time on paging rather then executing. is called as an trashing.

cause of trashing:-

→ if a process does not have "enough" pages, the page fault rate is very high. this leads to: o low CPU utilization.

→ Trashing is a process is busy swapping pages in and out.

→ os thinks that it needs to increase the degree of multiprogramming.

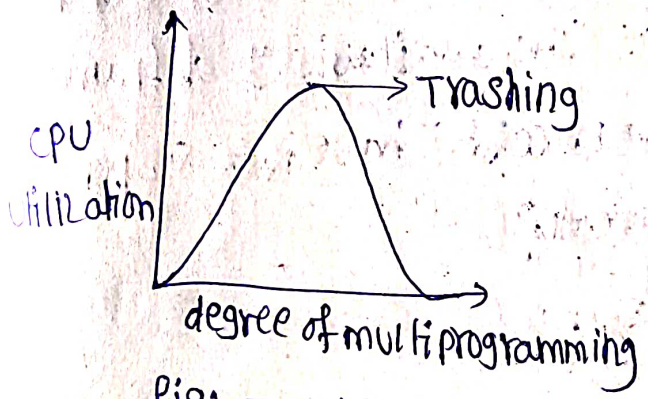


Fig:- Trashing

→ if trashing occurs cpu performance goes down.

→ techniques used to handle the trashing:-

① working set model

② page fault frequency.

① working set model:-

The accuracy of the working set is depending on the value of parameters.

→ if 'D' is the total "demand for frames", now, if 'm' is the "no of frames available in the memory",

these are 2 possibilities.

(i) D > m i.e total demand for frames exceeds the no. of frames, then "trashing will occur" as some processes would not get enough frames

(ii) D ≤ m, then there would be "no trashing".

→ if there are enough extra frames, then some more processes can be loaded in the memory.

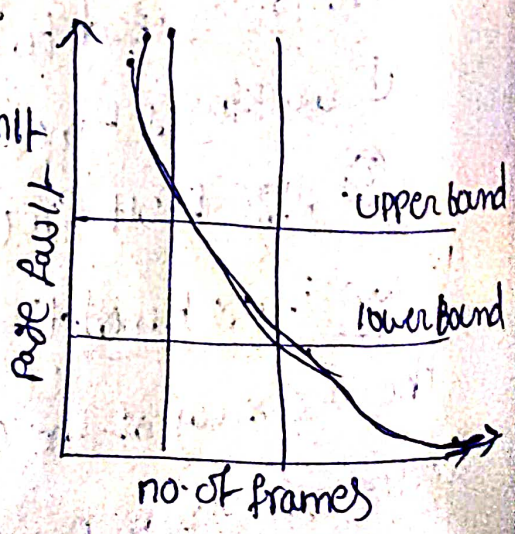
→ the other hand if the summation of working set sizes exceeds the availability of frames, then some of the processes have to be suspended (swapped out of memory)

② page fault frequency :-

more direct approach to handle trashing is the one that uses page fault frequency concept.

→ if the page fault rate is too high it indicates that upper unit the process has too few frames allocated to it.

→ if the page fault is below the lower unit, frames can be removed from the process.



memory mapped files:-

→ ~~according~~ accessing a file on disk will requires the standard system calls:

① open() ② read() ③ write()

→ every time the file is accessed requires a system call and disk access.

memory mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to page in memory.

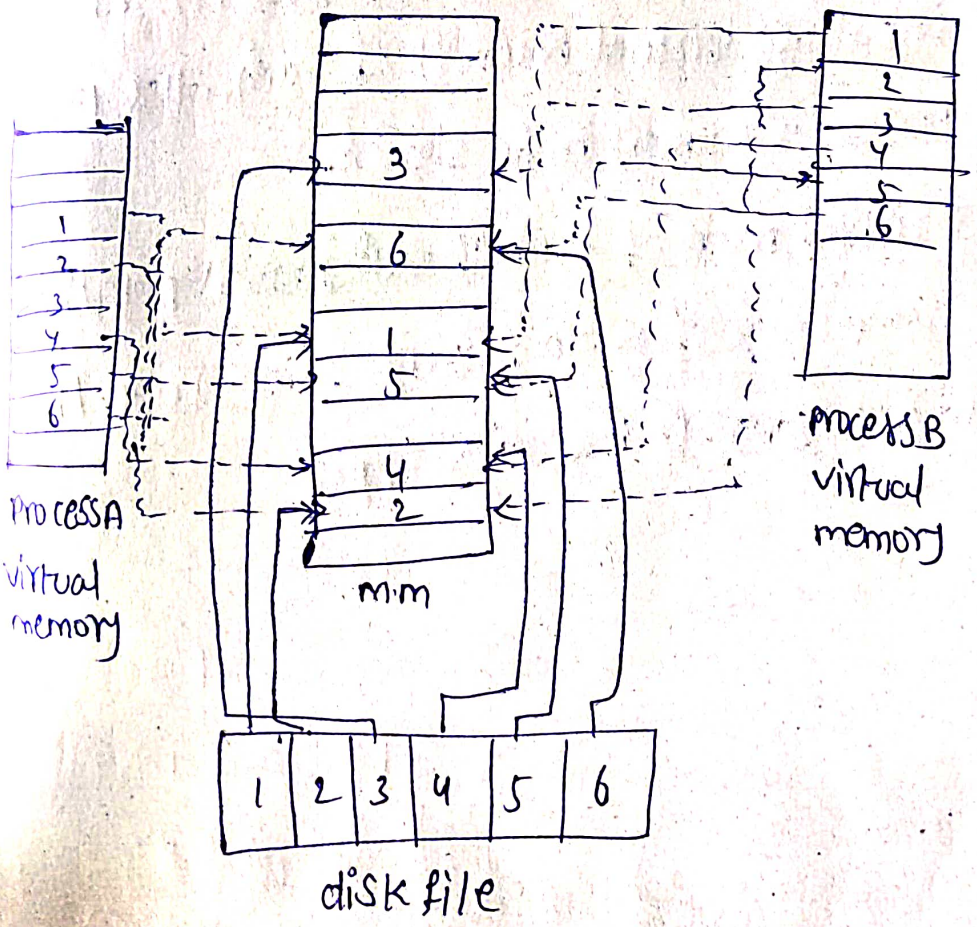


Fig:- memory mapped files

→ a file is initially read using demand paging, resulting in a page fault. i.e

Ex:- we are having page 7 it is not matched with the memory at that time page fault is occure

→ Page sized portion of the file or pages is read the file system into a physical page (or) memory pages.

→ The OS periodically checks of the page in the memory mapping the file have been modified, then update the physical file on disk (or) memory file on disk

→ When the file is closed, all the memory mapped data are written back to disk and removed from the virtual memory of the process.

→ also allows several processes to map the same file into the virtual memory of each, allowing the pages in memory to be shared.

→ The virtual map of each shared process points to the same page of main memory --- the page that holds a copy of the disk block.