Virtual Memory Management:

→ Virtual Memory is a storage schema that provides user an illusion of having a very big main memory.

→ In this case user can load the bigger size processes than the available main memory.

→ Instead of loading one big process in the main memory the OS loads the different parts of more than one process in the main memory.

→ By that, the degree of multiprogramming will be increased.

→ it is useful for users where physical memory is small

In real time, most processes never need all their pages at once, for following reasons

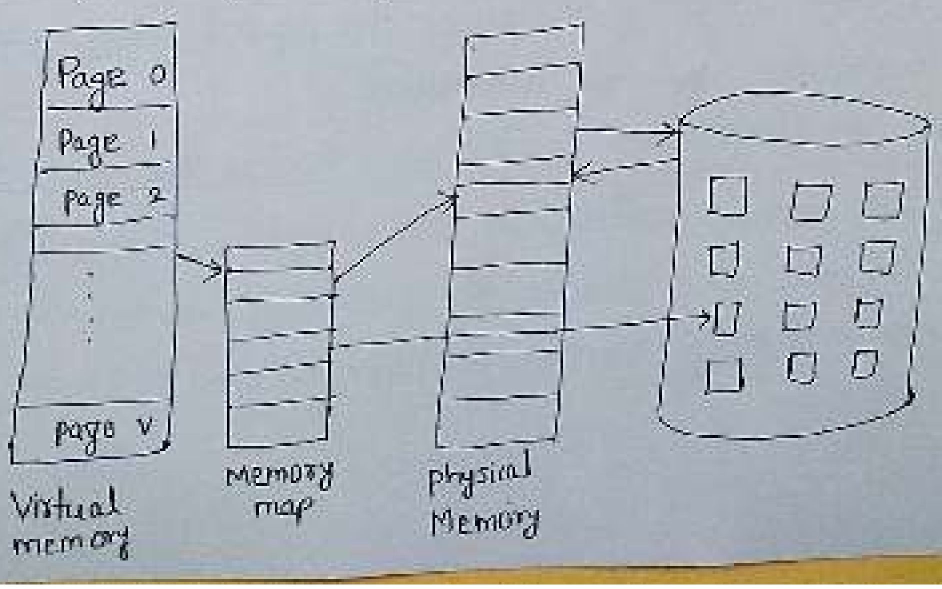
- Error handling code is not needed unless that specific error occurs some of which are quite rare.
- Arrays are often over-sized for worst-case, only a small fraction of the arrays are used in real time.

Advantages:

→ Large programs can be written, as virtual space available is huge compared to physical memory.

→ Less I/O required, leads to faster and easy swapping of processes.

→ More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.



34

Demand paging:

→ Demand paging is a type of swapping done in virtual memory systems. in Demand paging, the data is not copied from the disk to the RAM until they are needed. or being demanded by some program.

→ The data will not be copied when the data is already available on the memory this is called lazy swapper because only the demanded pages of memory are being swapped from the secondary storage.(disk space) to the main memory.

→ In contrast during pure swapping, all the memory for a process is swapped from secondary storage to main memory during the process startup.

Basic concepts or working of Demand paging:

→ The demand paging working is based on a page table implementation The page table maps logical memory to physical memory. the page table uses a bitwise operator to mark if a page is valid or invalid

→ A valid page is one that currently resides in main memory an invalid page can be defined as the one that currently resides in Secondary memory

When a process tries to access a page. the following will happen.

i. attempt to access the page, the page is valid, page processing instruction continues as normal.

ii. if the page is an invalid one, then a page fault trap occurs.

iii. The memory reference is checked to determine if it is a valid reference to a location on secondary memory or not. if not, the process is terminated. otherwise, the required page is paged in.

Scanned by CamScanner

procedure for handling page fault

→ We check an internal table for this process to determine whether the reference was a valid or an invalid memory access

→ if the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

→ we find a free frame, schedule a disk operation to read the desired page into the newly allocated frame.

→ When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

→ we restart the instruction that was interrupted by the trap. The process can now access the page.

Performence of Demand paging:

effective access time with page fault is

$$EAT = (1-P) \times ma + P \times \text{page fault time}$$

probability of page fault p: $0 \leq p \leq 1$

Eg: page fault service time of 8 ms, memory access time 200 ns find EAT in nanoseconds.

$$EAT = (1-P) \times ma + P \times \text{page fault time}$$

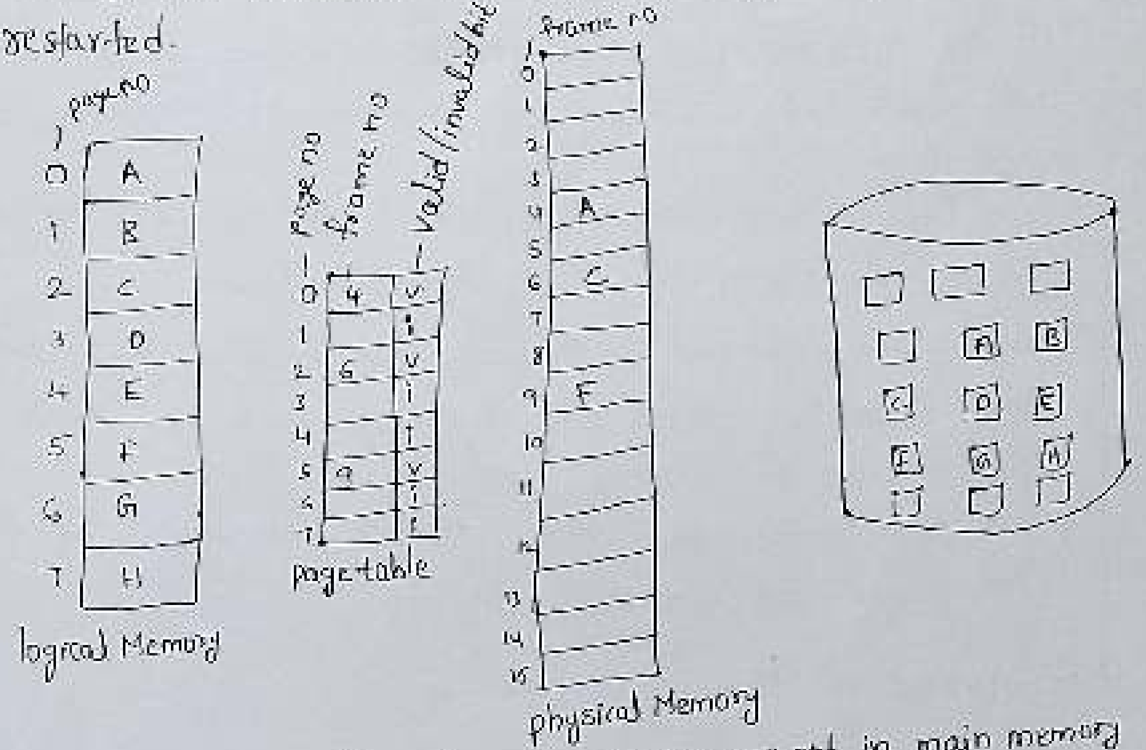
$$= (1-P) \times 200 + P \times 8,000,000 \quad (\because 1ms = 1,000,000 ns)$$

$$= 200 - 200P + P \times 8,000,000$$

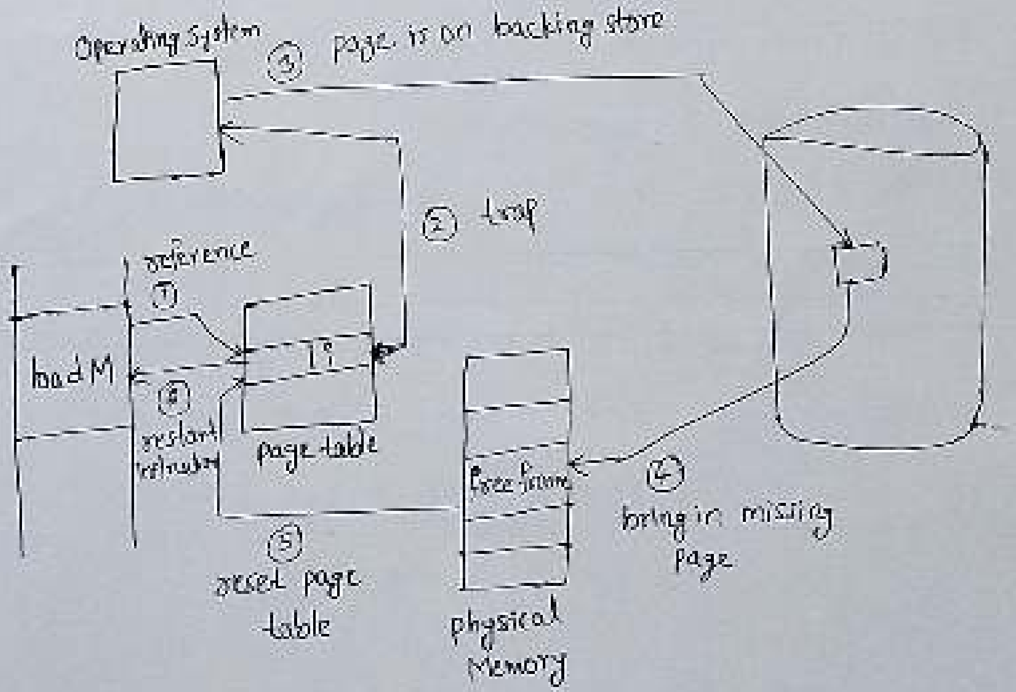
$$= 200 + 7,999,800P$$

iv) Now the disk operation to read the desired page into main memory is scheduled.

v) finally, the instruction that was interrupted by the OS trap is restarted.



Page table when some pages are not in main memory



Steps in Handling a page fault

Page Replacement :

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with an the required page.

Page replacement is required when, all the frames of main memory are already occupied. Thus, a page has to be replaced to create a hole or room for the required page.

Page Replacement Algorithms:

Page Replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

1. FIFO Page Replacement Algorithm
2. LIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm
4. Optimal page Replacement Algorithm
5. Random page Replacement Algorithm
6. LFU
7. MFU

→ A good page replacement algorithm is one that minimizes the number of page faults.

Page replacement takes the following approach

→ if no frame is free, find the location of the desired page on the disk.

→ find a free frame: if there is a free frame, use it; if there is no free frame, use a page replacement algorithm to select a victim frame. write the victim frame to the disk. change the page and frame tables accordingly.

→ Read the desired page into the newly freed frame, change the page and frame tables.

→ Restart the user process.

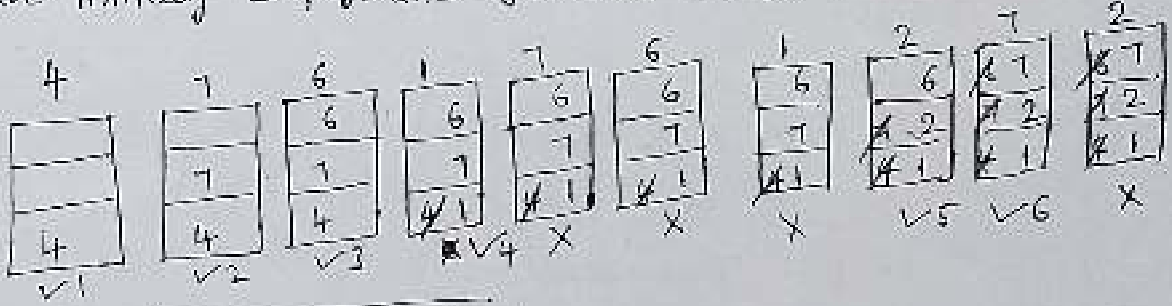
FIFO Page Replacement Algorithm:

⇒ It works based on principle of first in first out

⇒ it replaces the oldest page that has been present in the main memory for the longest time.

⇒ it is implemented by keeping track of all the pages in a queue. we replace the page at the head of the queue when a page is brought into memory, we insert it at the tail of the queue.

Eg1: reference string = 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, find total no. of page faults, hit ratio, miss ratio. Assume that all the frames are initially empty and system uses 3 page frames for storing process



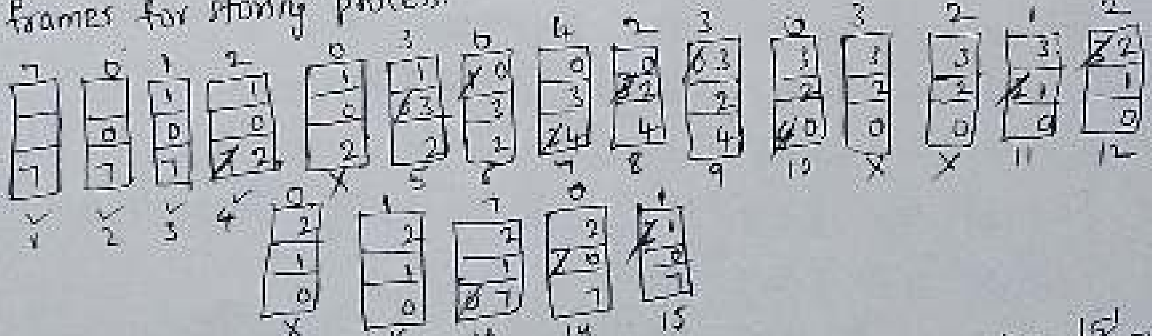
Queue: 7 2 1 ~~6~~ ~~7~~ ~~4~~

Total number of page faults = 6

hit ratio = 1 - miss ratio
= 1 - 0.6 [page fault or page misses = $\frac{\text{total no. of page misses}}{\text{total no. of references}}$ = $\frac{6}{10}$ = 0.6]
= 0.4

∴ hit ratio = 0.4, miss ratio = 0.6

Eg2: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 = reference string, 3 page frames for storing process.



miss ratio = $\frac{15}{20}$ = 0.75

no. of page faults = 15

hit ratio = 0.8

→ disadvantage of FIFO Page replacement algorithm is, it suffer from belady's Anomaly.

Belady's Anomaly:

Belady's Anomaly is the phenomenon of increasing the number of page faults on increasing the number of frames in main memory.

→ an algorithm suffers from belady's anomaly if and only if does not follow stack property.

→ Algorithm that follow stack property are called as stack based Algorithms. these algorithms do not suffer from belady's anomaly.

Note: 1. FIFO, Random page Replacement and second chance algorithms suffer from belady's Anomaly.

2. LRU, optimal page Replacement Algorithms follows the stack based algorithms, hence they do not suffer from belady's Anomaly.

Optimal page Replacement Algorithm:

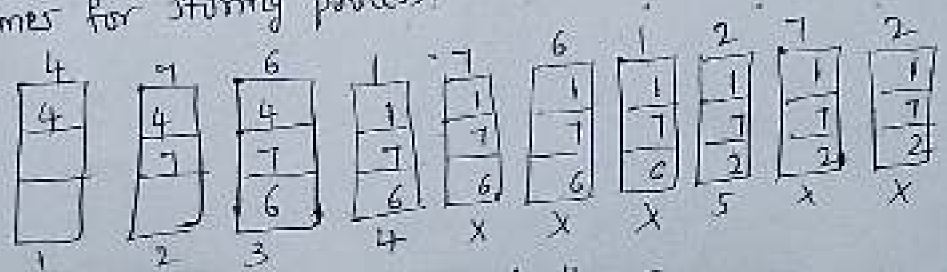
→ it replaces the page that will not be refered by the cpu in future for the longest time.

→ it is practically impossible to implement this algorithm

→ This is because the pages that will not be used in future for the longest time cannot be predicted.

→ However, it is the best known algorithm and gives the least number of page faults.

Eg:- Reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, it uses 3 page frames for storing process.



page faults = 5

Eg2: Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Reference	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	2	2	2	2	2	2	2	2	2	2	7	7	7
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
	1	2	3	4	X	5	X	6	X	X	7	X	X	8	X	X	X	9	X	X

no. of page faults = 9

hit ratio = 1 - 0.55 = 0.45

miss ratio = $\frac{11}{20}$ = 0.55

LRU Page Replacement:

→ it works based on principle of Least Recently used

→ it replaces the page that has not been refered by the CPU for the longest time.

Eg: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

Reference	4	7	6	1	7	6	1	2	7	2
Frame 1	4	4	4	1	1	1	1	6	1	1
Frame 2		7	7	7	7	7	7	2	2	2
Frame 3			6	6	6	6	6	6	7	7
	1	2	3	4	X	X	X	5	6	X

no. of page faults = 6

hit ratio = 0.4   miss ratio = 0.6

Eg2: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Reference	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	1	1	1	1	1	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
	1	2	3	4	X	5	X	6	7	8	9	X	X	10	X	11	X	12	X	X

no. of page faults = 12

miss ratio = $\frac{12}{20}$ = 0.51

hit ratio = 0.49

Counting-based page Replacement:

1. Least frequently used (LFU) page replacement Algorithm
2. Most frequently used (MFU) page replacement Algorithm

→ LFU requires that the page with the smallest count be placed
→ MFU is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
→ implementation of LFU & MFU is expensive

Allocation of frames:

→ Frame allocation algorithms are used if you have multiple processes it helps decide how many frames to allocate to each process.

There are various constraints to the strategies for the allocation of frames.

→ We cannot allocate more than the total number of available frames.
→ At least a minimum number of frames should be allocated to each process. this constraint is supported by two reasons. the first reason is, the number of allocated frames are less then it increase page fault ratio, decreasing the performence of the execution of the process.
→ Second reason is, there should be enough frames to hold all the different pages that any single instruction can reference.

Frame allocation algorithms:

two algorithms are commonly used to allocate frames to a process.

- Equal Allocation
- proportional allocation

Equal allocation:

→ In a system with x frames and y processes, each process gets equal number of frames. i.e, x/y

Eg if the system has 48 frames and 9 processes, each process will get 5 frames. i.e, $\overset{5}{48/9}$, 3 frames which are not allocated to any process can be used as a free-frame buffer pool.

→ disadvantage is allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames

→

Proportional allocation:

→ frames are allocated to each process according to the process size

→ for a process P_i of size s_i, the number of allocated frames is

$$a_i = (s_i / s) * m$$

m - number of frames in the system

s - sum of the sizes of all the processes

Eg:- A system with 62 frames, if there is a process of 10KB and another process of 127 KB then 1st process will be allocated

$$\left(\frac{10}{137}\right) * 62 = 4 \text{ frames} \qquad [\because s = 10+127 = 137]$$

$$2^{nd} \text{ process } \left(\frac{127}{137}\right) * 62 = 57 \text{ frames}$$

→ advantage is all the processes share the available frames according to their needs, rather than equally.

Global Vs Local Allocation:

→ The number of frames allocated to a process can also dynamically change depending on whether we have used global replacement or local replacement for replacing pages in case of page fault

Local replacement:

→ when a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only.

→ advantage: The pages in memory for a particular process and the page fault ratio is affected by the paging behavior of only that process

→ Disadvantage: A low priority process may hinder a high priority process by not making available to the high priority process its frames.

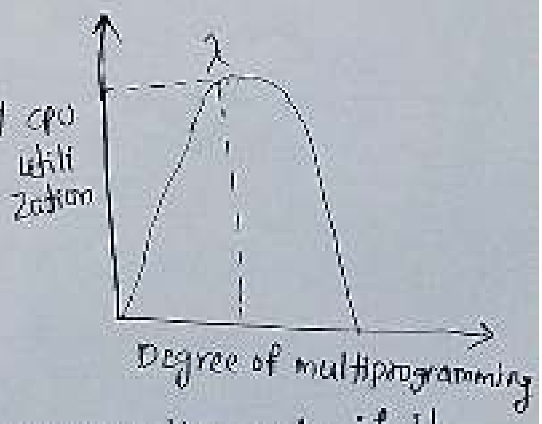
Global replacement:

→ when a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process i.e, one process can take a frame from another.

→ advantage: increase throughput

→ Disadvantage: The page fault ratio of a process can not be controlled by the process itself.

Thrashing;



→ At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory.

→ unused pages are not swapped in and out of memory.

→ when the os brings one page in, it must throw another out. if it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing.

14

→ The system spends most of its time swapping pages rather than executing instructions.

In above diagram initial degree of multi programming upto some extent of point, the CPU utilization is very high and the system resources of are utilized 100%. But if we further increase the degree of multi programming the CPU utilization will drastically fall down and the system will spent more time only in the page replacement and the time taken to complete the execution of the process will increase. this situation in the system is called as thrashing.

Causes of Thrashing:

1. High degree of multiprogramming

2. Lacks of frames.

If a process has less number of frames then less pages of that process will be able to reside in memory this leads to thrashing so sufficient frames are allocated to each process to prevent thrashing.

Recovery of Thrashing:

→ Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold

→ If the system is already in thrashing then instruct the mid term scheduler to suspend some of the processes so that we can recover the system from thrashing.

Virtual Memory in windows:

Windows XP implements virtual memory using demand paging with clustering. clustering handles page faults by bringing in not only the faulting page but also several pages following the faulting page. It works working-set maximum if sufficient memory is available. If not it works working set minimum.