Operating System
Virtual Memory

Prepared By:

Dr. Sanjeev Gangwar

Assistant Professor,

Department of Computer Applications,

VBS Purvanchal University, Jaunpur

Virtual Memory

Virtual memory is a technique that allows the execution of processes that may not be completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. This technique frees programmers form the concerns the of memory-storage limitations. Virtual memory also allows processes to easily share files and address space, and it provides an efficient mechanism for process creation.

It is a virtual resource of a computer. It is an illusion that a computer system processes more memory that is actually having. This illusion makes a process independent of the size of main memory. It also permits a large number of processes to share a computer system without constraining each other.

Virtual memory allows execution of partially loaded processes. As a consequence, virtual address spaces of active processes in a virtual-memory system can exceed the capacity of the physical memory. This is accomplished by maintaining an image of the entire virtual-address space of a process on secondary storage, and by bringing its sections into main memory when needed. The Operating System decides which sections to bring in, when to bring them in, and where to place them. Thus, virtual-memory systems provide for automatic migration of portions of address spaces between secondary and primary storage. Virtual memory provides the illusion of a much larger memory than may actually be available, so programmers are relieved of the burden of trying to fit a program into limited memory.

The ability to execute a program that is only partially in memory would confer many benefits:

- A program would no longer be constrained by the amount of physical memory that is available. User would be able to write programs for an extremely large virtual-address space simplifying the programming task.
- Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput, but with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

Thus, running a program that is not entirely in memory would benefit both the system and the user.

Virtual memory is to be provided for programmers when only a smaller physical memory is available. Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Figure 1).

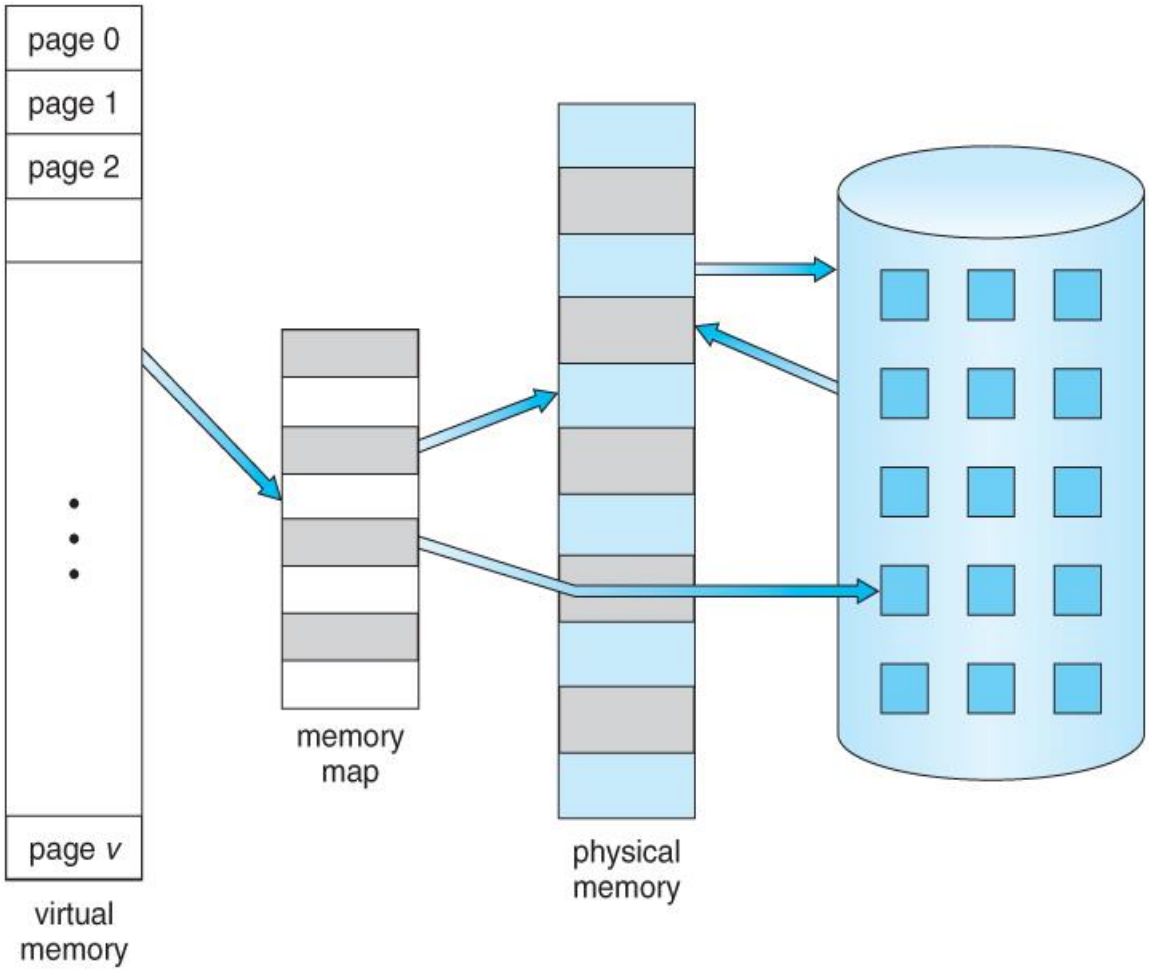

Fig 1: Virtual Memory

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory. The segment replacement algorithms are very complex in use because of variable sizes of segments while page replacement algorithms are easy to use and implement.

Demand Paging: A demand paging is similar to a paging system with swapping (Figure 2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process; the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

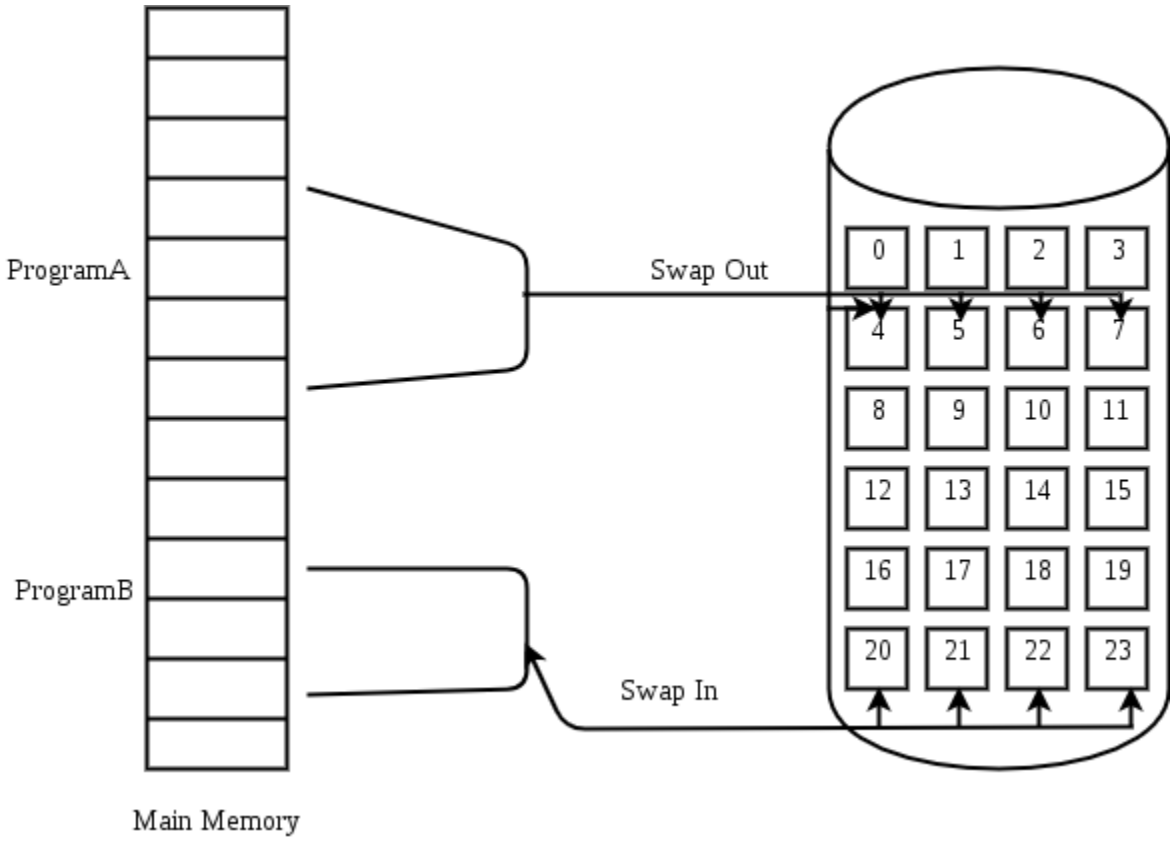

Fig 2: Transfer of a paged memory to continuous disk space

With this scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. The valid-invalid scheme can be used for this purpose. When this bit is set to "valid", this value indicates that the associated page is both legal and in memory. If the bit is set to "invalid", this value indicates that the page either is not valid or is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simply marked invalid, or contains the address of the page on disk. This situation is depicted in Figure 3.

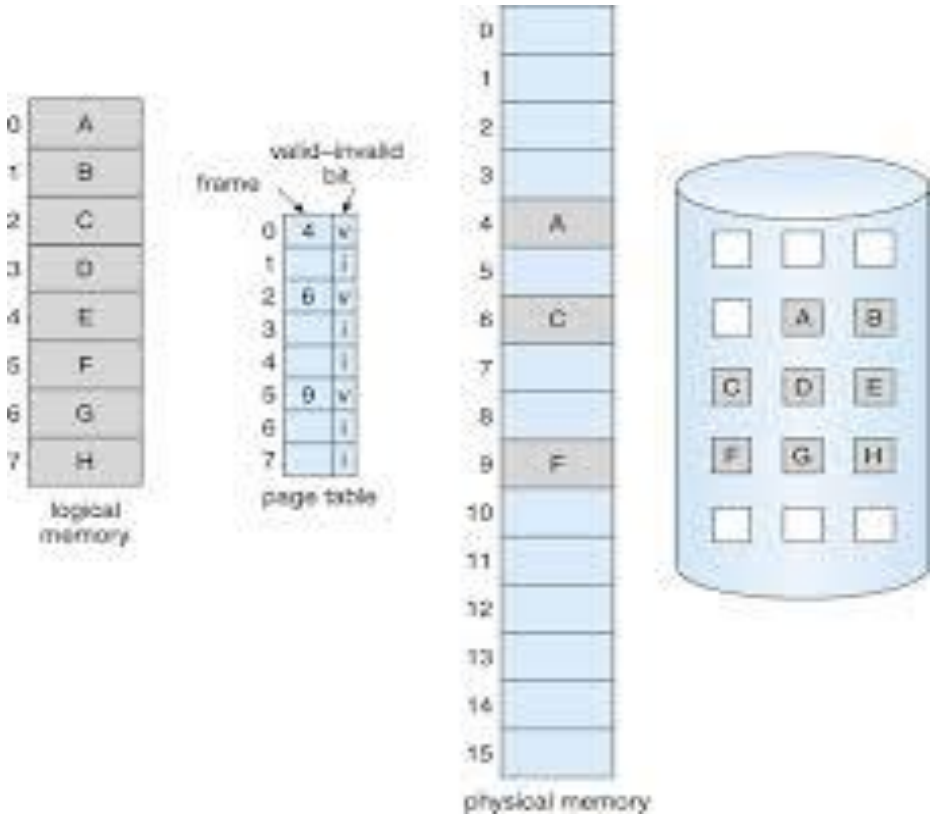

Fig 3: Page Table when some pages are not in main memory

Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

Initially only those pages are loaded which will be required the process immediately. The pages that are not moved into the memory are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page. When the process requires any of the pages that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table to indicate that the page is now in memory.

- We restart the instruction that was interrupted by the illegal address trap. Now the process can access the page as if it had always been in memory.

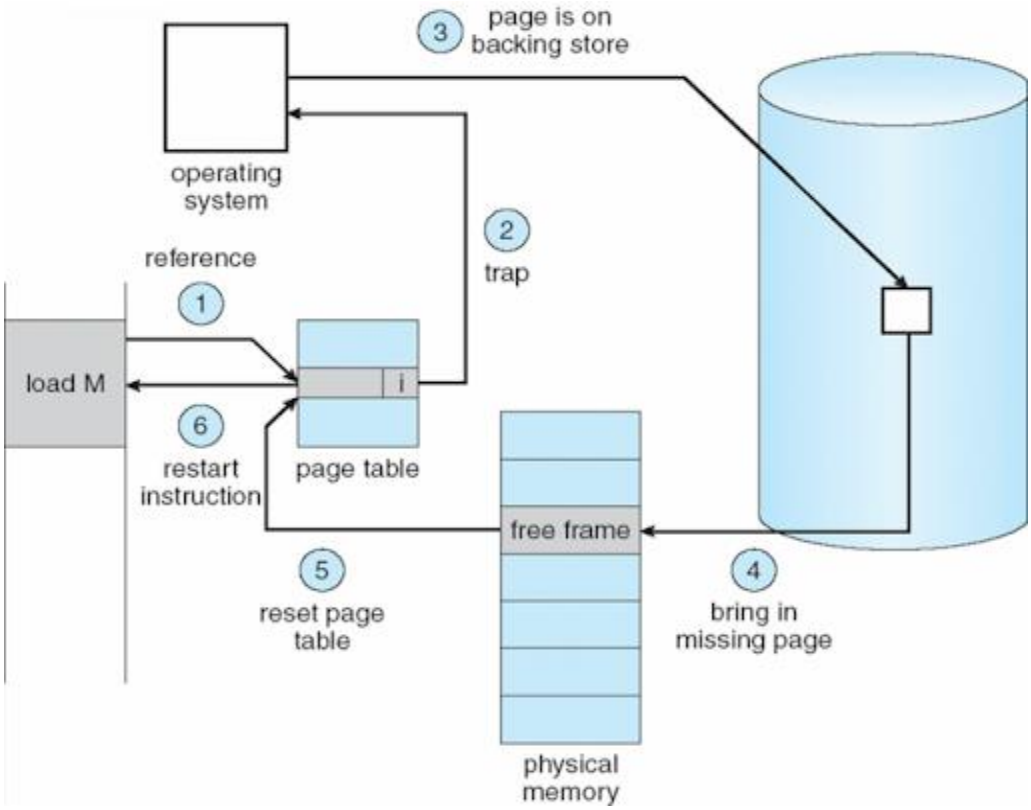

Fig 4: Steps in handling a page fault

Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

The page replacement is used to make the frame free if they are not in used. If no frame is free then other process is called in.

Advantages of Demand Paging

- Large virtual memory.
- More efficient use of memory.
- Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging

- Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of an explicit constraint on jobs address space size.

Thrashing: the phenomenon of moving pages from primary to secondary storage or vice-versa consumes a lot of computer's energy but accomplishes very little useful results. This situation is called thrashing. A process is in thrashing if it is spending more time in paging instead of their execution.

The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilization would fall drastically. The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing.

This phenomenon is illustrated in figure 5 in which CPU utilization is plotted against degree of multiprogramming. As degree of multiprogramming increases, CPU utilization goes on increasing although more slowly until a maximum is reached. After this point of degree of multiprogramming is increased then thrashing is occurred and CPU utilization drops sharply after this point.

At this point, to increase CPU utilization and stop thrashing degree of multiprogramming should be reduced.

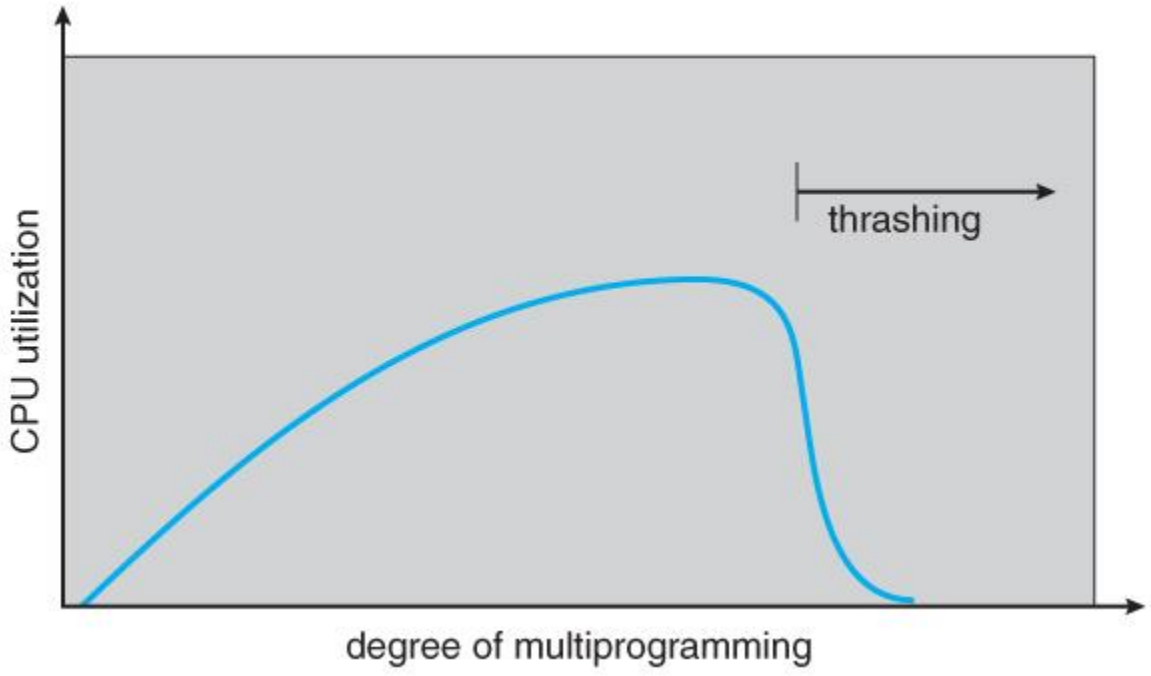

Fig 5: Thrashing

To prevent from thrashing a strategy called as page-fault frequency (PFF) is used. Since thrashing has a high rate of page fault. So we can control the page-fault rate. When it is too high, we know that the process needs more frames. Similarly if the page fault rate is too low, then the

process has too many frames. We can establish upper and lower bounds on the desired page fault rate (Figure 6). If the actual page fault rate exceeds the upper limit, we allocate that process to another frame. If the page fault rate falls below the lower limit, we remove a frame from that process. Thus we can directly measure and control the page fault rate to prevent thrashing.

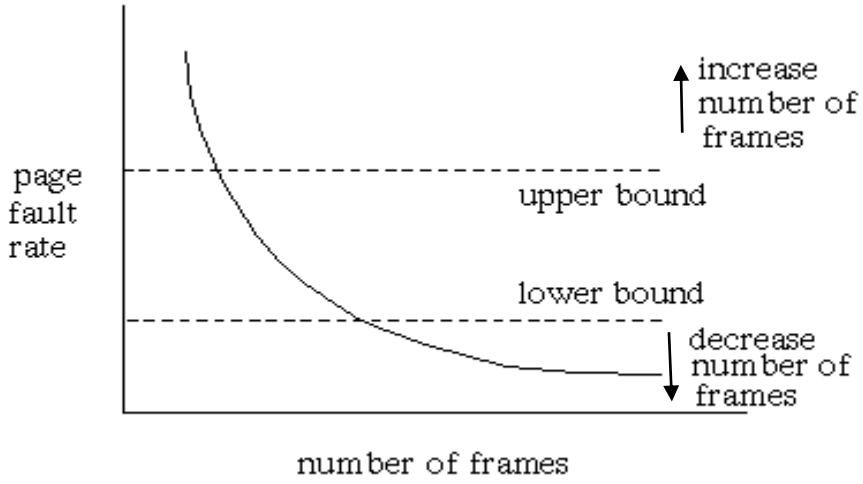

Fig 6: Page fault frequency (PFF)

References:

(1) Abraham Silberschatz, Galvin & Gagne, Operating System Concepts, John Wiley & Sons, INC.

(2) Harvay M.Deital, Introduction to Operating System, Addition Wesley Publication Company.

(3) Andrew S.Tanenbaum, Operating System Design and Implementation, PHI

(4) Vijay Shukla, Operating System, S.K. Kataria & Sons