Unit: III
Lecture: 7
(Memory Management)
Virtual Memory

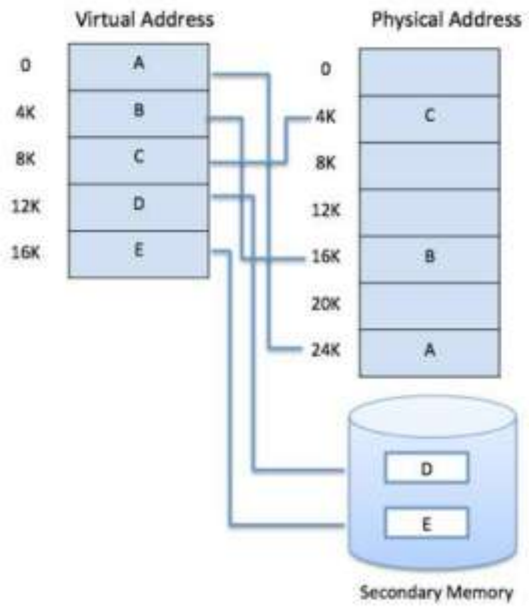
Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory.

Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory (RAM) to disk storage.

Today, most personal computers come with at least 8 GB of RAM. But sometimes, this is not enough to run several programs at one time. This is where virtual memory comes in. Virtual memory frees up RAM by swapping data that has not been used recently over to a storage device, such as hard drive or solid-state drive.

Virtual memory is important for improving system performance, multitasking and using large programs.

Virtual memory automatically manages two levels of the memory hierarchy, representing the main memory and the secondary storage, in a manner that is invisible to the program that is running. Virtual memory enables a program to ignore the physical location of any desired block of its address space; a process can simply seek to access any block of its address space without concern for where that block might be located. If the block happens to be located in the main memory, access is carried out smoothly and quickly else the virtual memory has to bring the block in from secondary storage and allow it to be accessed by the program.



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.

Demand Paging:

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

Page Fault: A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.

The process of loading the pages into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps:

- If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
- The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
- The OS will search for the required page in the logical address space.
- The required page will be brought to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.
- The page table will be updated accordingly.
- The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Advantages:

- More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- A process may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

Thrashing:

At any given time, only a few pages of any process are in the main memory and therefore more processes can be maintained in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme.

In the steady-state practically, all of the main memory will be occupied with process pages, so that the processor and OS have direct access to as many processes as possible. Thus 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. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

Thus, thrashing is a situation in the system which occurs when the system will spend more time only on the page replacement thereby decrease in CPU utilization and thus the time taken to complete the execution of the process will increase.