Memory Organization

Memory Management:

The main working principle of digital computer is Von-Neumann stored program principle. First of all we have to keep all the information in some storage, mainly known as main memory, and CPU interacts with the main memory only. Therefore, memory management is an important issue while designing a computer system.

On the other hand, everything cannot be implemented in hardware, otherwise the cost of system will be very high. Therefore some of the tasks are performed by software program. Collection of such software programs are basically known as operating systems. So operating system is viewed as extended machine. Many more functions or instructions are implemented through software routine. The operating system is mainly memory resistant, i.e., the operating system is loaded into main memory.

Due to that, the main memory of a computer is divided into two parts. One part is reserved for operating system. The other part is for user program. The program currently being executed by the CPU is loaded into the user part of the memory. The two parts of the main memory are shown in the figure 3.2.

In a uni-programming system, the program currently being executed is loaded into the user part of the memory.

In a multiprogramming system, the user part of memory is subdivided to accommodate multiple processes. The task of subdivision is carried out dynamically by operating system and is known as memory management.

Fig 3.2. Main memory

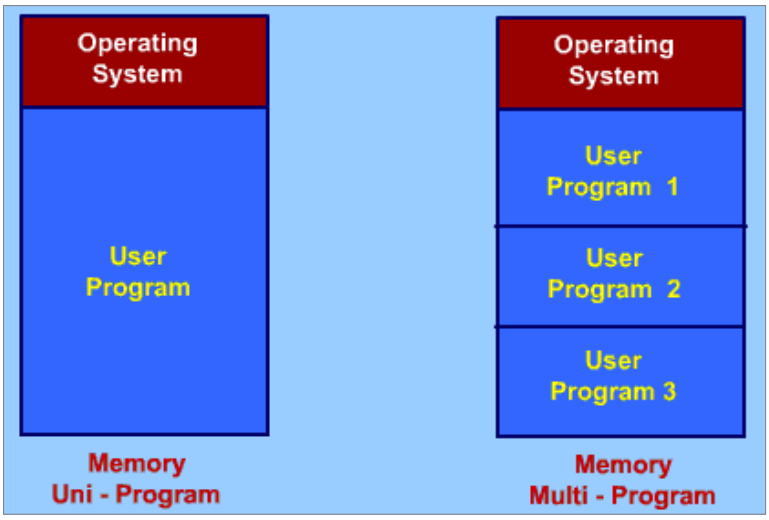
Efficient memory management is vital in a multiprogramming system. If only a few process are in memory, then for much of the time all of the process will be waiting for I/O and the processor will idle. Thus memory needs to be allocated efficiently to pack as many processes into main memory as possible.

When memory holds multiple processes, then the process can move from one process to another process when one process is waiting. But the processor is so much faster then I/O that it will be common for all the processes in memory to be waiting for I/O. Thus, even with multiprogramming, a processor could be idle most of the time.

Due to the speed mismatch of the processor and I/O device, the status at any point in time is referred to as a state.

In an **uniprogramming system**, main memory is divided into two parts: one part for the operating system and the other part for the program currently being executed.
In **multiprogramming system**, the user part of memory is subdivided to accommodate multiple processes.
The task of subdivision is carried out dynamically by the operating system and is known as **memory management**. In uniprogramming system, only one program is in execution. After completion of one program, another program may start. In general, most of the programs involve I/O operation. It must take input from some input device and place the result in some output device.
Partition of main memory for uni-program and multi program is shown in figure3.4

To utilize the idle time of CPU, we are shifting the paradigm from uniprogram environment to multiprogram environment.
Since the size of main memory is fixed, it is possible to accommodate only few process in the main memory. If all are waiting for I/O operation, then again CPU remains idle.
To utilize the idle time of CPU, some of the process must be off loaded from the memory and new process must be brought to this memory place. This is known **swapping**.



3.4. Partition of main memory for uni-program and multi program.

What is swapping?

1. The process waiting for some I/O to complete, must store back in disk.
2. New ready process is swapped in to main memory as space becomes available.
3. As process completes, it is moved out of main memory.
4. If none of the processes in memory are ready,
    - Swapped out a block process to intermediate queue of blocked process.
    - Swapped in a ready process from the ready queue.

But swapping is an I/O process, so it also takes time. Instead of remain in idle state of CPU, sometimes it is advantageous to swapped in a ready process and start executing it. The main question arises where to put a new process in the main memory. It must be done in such a way that the memory is utilized properly.

Partitioning:

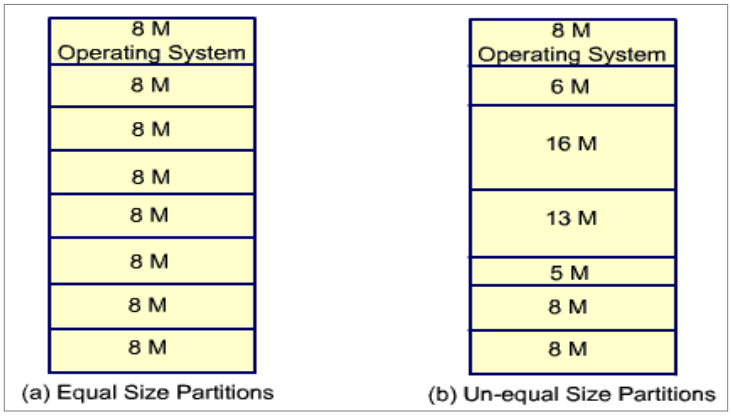
Splitting of memory into sections to allocate processes including operating system. There are two schemes for partitioning:

- Fixed size partitions
- Variable size partitions

Fixed sized partitions:

The memory is partitioned to fixed size partition. Although the partitions are of fixed size, they need not be of equal size. There is a problem of wastage of memory in fixed size even with unequal size. When a process is brought into memory, it is placed in the smallest available partition that will hold it.

Equal size and unequal size partition of fixed size partitions of main memory is shown in Figure3.5.



Even with the use of unequal size of partitions, there will be wastage of memory. In most cases, a process will not require exactly as much memory as provided by the partition.
For example, a process that require 5-MB of memory would be placed in the 6-MB partition which is the smallest available partition. In this partition, only 5-MB is used, the remaining 1-MB cannot be used by any other process, so it is a wastage. Like this, in every partition we may have some unused memory. The unused portion of memory in each partition is termed as hole.

Variable size Partition:

When a process is brought into memory, it is allocated exactly as much memory as it requires and no more. In this process it leads to a hole at the end of the memory, which is too small to use. It seems that there will be only one hole at the end, so the waste is less.
But, this is not the only hole that will be present in variable size partition. When all processes are blocked then swap out a process and bring in another process. The new swapped in process may be smaller than the swapped out process. Most likely we will not get two process of same size. So, it will create another whole. If the swap- out and swap-in is occurring more time, then more and more hole will be created, which will lead to more wastage of memory.
There are two simple ways to slightly remove the problem of memory wastage:

1. Join the adjacent holes into one large hole, so that some process can be accommodated into the hole.
2. From time to time go through memory and move all holes into one free block of memory.

During the execution of process, a process may be swapped in or swapped out many times. it is

obvious that a process is not likely to be loaded into the same place in main memory each time it is swapped in. Furthermore if compaction is used, a process may be shifted while in main memory.

A process in memory consists of instruction plus data. The instruction will contain address for memory locations of two types:

- Address of data item
- Address of instructions used for branching instructions

These addresses will change each time a process is swapped in. To solve this problem, a distinction is made between logical address and physical address.

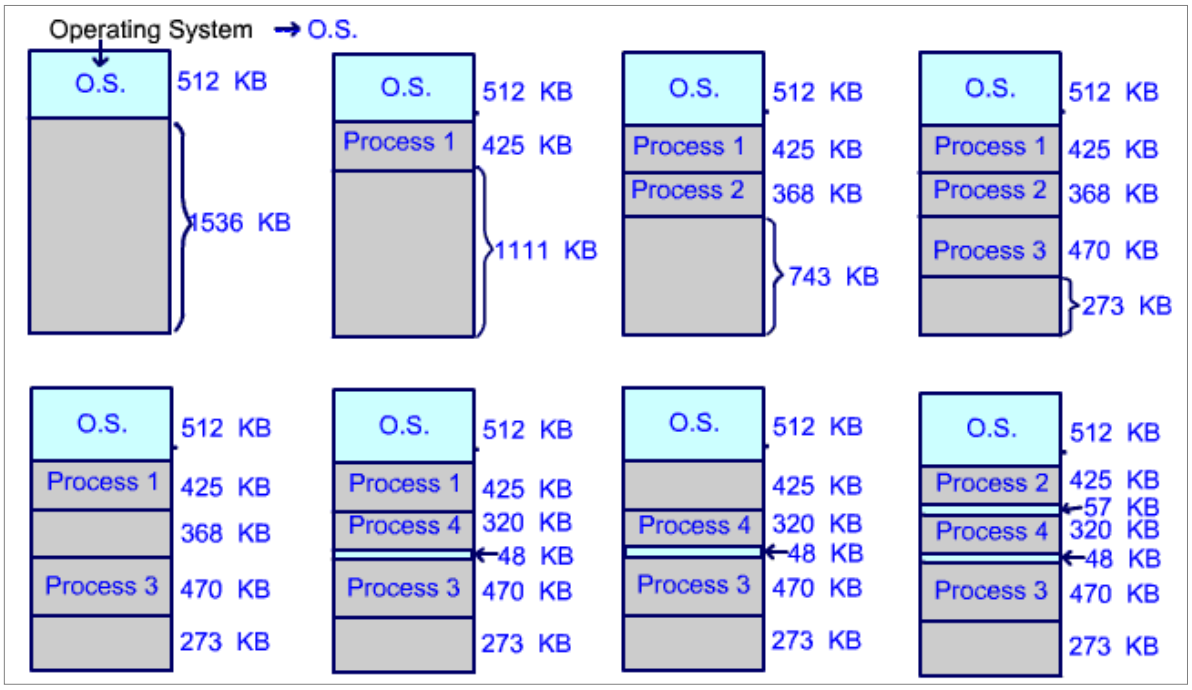
- *Logical address is expressed as a location relative to the beginning of the program. An instruction in the program contains only logical address.*
- *Physical address is an actual location in main memory.*

When the processor executes a process, it automatically converts from logical to physical address by adding the current starting location of the process, called its base address to each logical address.

Every time the process is swapped in to main memory, the base address may be different depending on the allocation of memory to the process.

Consider a main memory of *2-MB* out of which 512-KB is used by the Operating System. Consider three processes of size *425-KB*, *368-KB* and *470-KB* and these three processes are loaded into the memory. This leaves a hole at the end of the memory. That is too small for a fourth process. At some point none of the process in main memory is ready. The operating system swaps out process-2 which leaves sufficient room for new process of size *320-KB*. Since process-4 is smaller than process-2, another hole is created. Later a point is reached at which none of the processes in the main memory is ready, but proces-2, so process-1 is swapped out and process-2 is swapped in there. It will create another hole. In this way it will create lot of small holes in the memory system which will lead to more memory wastage.

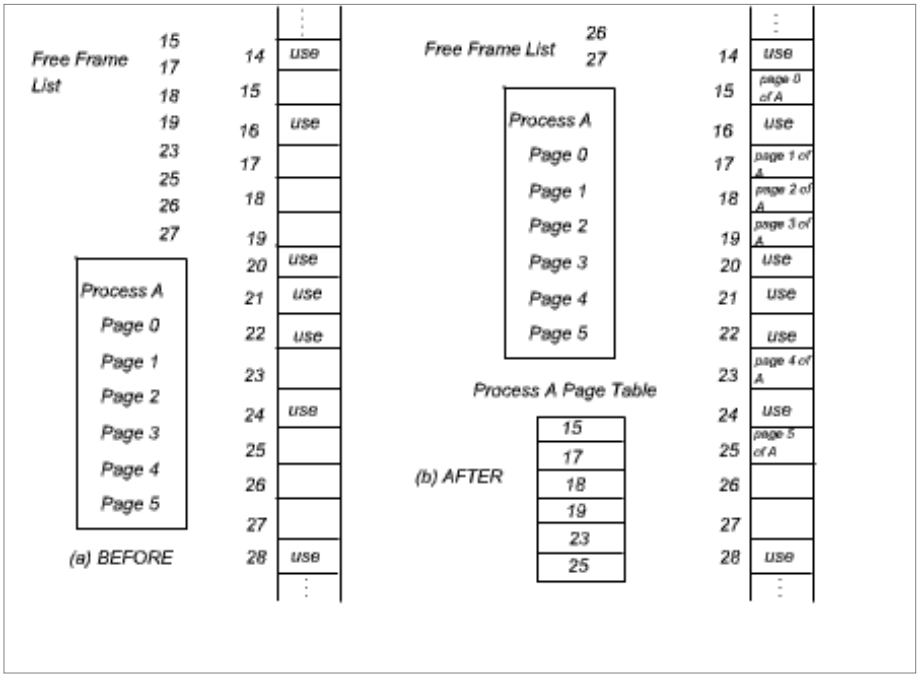
The effect of dynamic partitioning that creates more whole during the execution of processes is shown in the Figure3.6.



Paging:

- Both unequal fixed size and variable size partitions are inefficient in the use of memory. It has been observed that both schemes lead to memory wastage. Therefore we are not using the memory efficiently. There is another scheme for use of memory which is known as **paging.**
- In this scheme, the memory is partitioned into equal fixed size chunks that are relatively small. This chunk of memory is known as frames or page frames.
- Each process is also divided into small fixed chunks of same size. The chunks of a program is known as pages.
- A page of a program could be assigned to available page frame.
- In this scheme, the wastage space in memory for a process is a fraction of a page frame which corresponds to the last page of the program.
- At a given point of time some of the frames in memory are in use and some are free. The list of free frame is maintained by the operating system.
- Process A, stored in disk, consists of pages. At the time of execution of the process A, the operating system finds six free frames and loads the six pages of the process A into six frames.
- These six frames need not be contiguous frames in main memory. The operating system maintains a page table for each process.
- Within the program, each logical address consists of page number and a relative address within the page.
- In case of simple partitioning, a logical address is the location of a word relative to the beginning of the program; the processor translates that into a physical address.
- With paging, a logical address is a location of the word relative to the beginning of the page of the program, because the whole program is divided into several pages of equal length and the length of a page is same with the length of a page frame.
- A logical address consists of page number and relative address within the page, the process uses the page table to produce the physical address which consists of frame number and relative address within the frame.
- The Figure shows the allocation of frames to a new process in the main memory. A page table is maintained for each process. This page table helps us to find the physical address in a frame which corresponds to a logical address within a process.

The conversion of logical address to physical address is shown in the figure3.8 for the Process A.



3.7. Allocation of free frames

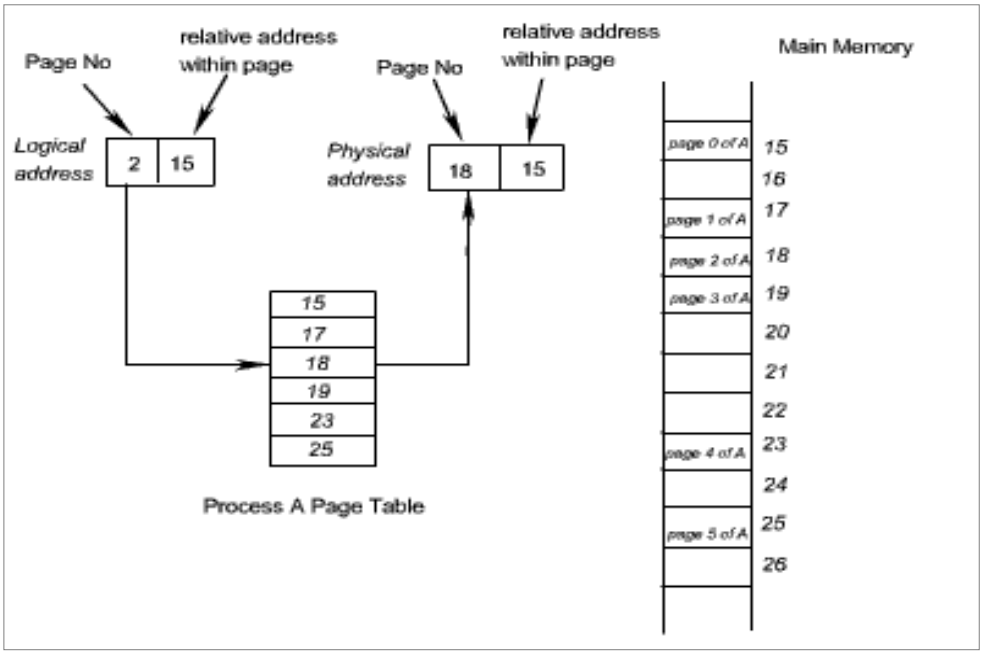


Fig 3.8.Translation of Logical Address to Physical Address

This approach solves the problems. Main memory is divided into many small equal size frames. Each process is divided into frame size pages. Smaller process requires fewer pages, larger process requires more. When a process is brought in, its pages are loaded into available frames and a page table is set up.

The translation of logical addresses to physical address is shown in the Figure.

Virtual Memory

- The concept of paging helps us to develop truly effective multiprogramming systems.
- Since a process need not be loaded into contiguous memory locations, it helps us to put a page of a process in any free page frame. On the other hand, it is not required to load the whole process to the main memory, because the execution may be confined to a small section of the program. (eg. a subroutine).
- It would clearly be wasteful to load in many pages for a process when only a few pages will be used before the program is suspended.
- Instead of loading all the pages of a process, each page of process is brought in only when it is needed, i.e on demand. This scheme is known as *demand paging.*
- Demand paging also allows us to accommodate more process in the main memory, since we are not going to load the whole process in the main memory, pages will be brought into the main memory as and when it is required.
- With demand paging, it is not necessary to load an entire process into main memory.
- This concept leads us to an important consequence – It is possible for a process to be larger than the size of main memory. So, while developing a new process, it is not required to look for the main memory available in the machine. Because, the process will be divided into pages and pages will be brought to memory on demand.
- Because a process executes only in main memory, so the main memory is referred to as real memory or physical memory.
- A programmer or user perceives a much larger memory that is allocated on the disk. This memory is referred to as virtual memory. The program enjoys a huge virtual memory space to develop his or her program or software. The execution of a program is the job of operating system and the underlying hardware. To improve the performance some special hardware is added to the system. This hardware unit is known as Memory Management Unit (MMU).
- In paging system, we make a page table for the process. Page table helps us to find the physical address from virtual address.
- The virtual address space is used to develop a process. The special hardware unit, called Memory Management Unit (MMU) translates virtual address to physical address. When the desired data is in the main memory, the CPU can work with these data. If the data are not in the main memory, the MMU causes the operating system to bring into the memory from the disk.

A typical virtual memory organization is shown in the Figure3.9.

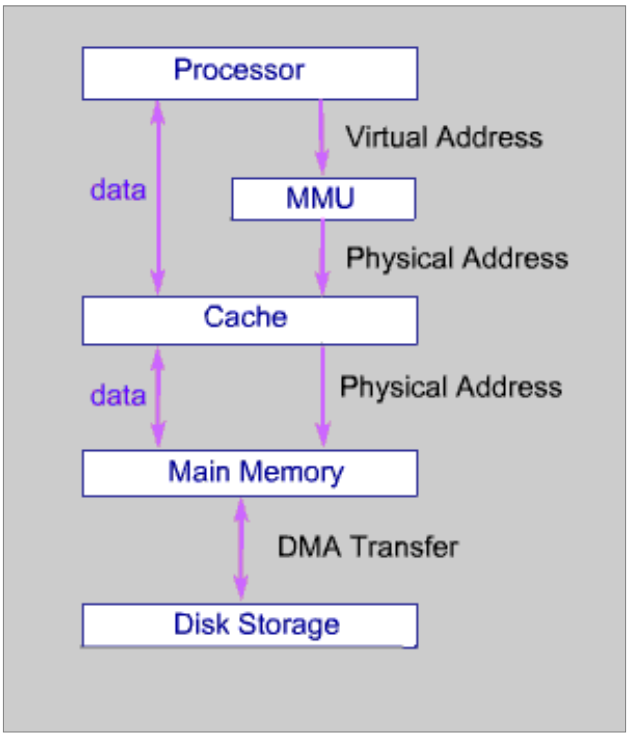


Fig3.9: virtual memory organization.

Address Translation

- The basic mechanism for reading a word from memory involves the translation of a virtual or logical address, consisting of page number and offset, into a physical address, consisting of frame number and offset, using a page table.
- There is one page table for each process. But each process can occupy huge amount of virtual memory. But the virtual memory of a process cannot go beyond a certain limit which is restricted by the underlying hardware of the MMU. One of such component may be the size of the virtual address register.
- The sizes of pages are relatively small and so the size of page table increases as the size of process increases. Therefore, size of page table could be unacceptably high.
- To overcome this problem, most virtual memory scheme store page table in virtual memory rather than in real memory.
- This means that the page table is subject to paging just as other pages are.
- When a process is running, at least a part of its page table must be in main memory, including the page table entry of the currently executing page.
- A virtual address translation scheme by using page table is shown in the Figure 3.10.

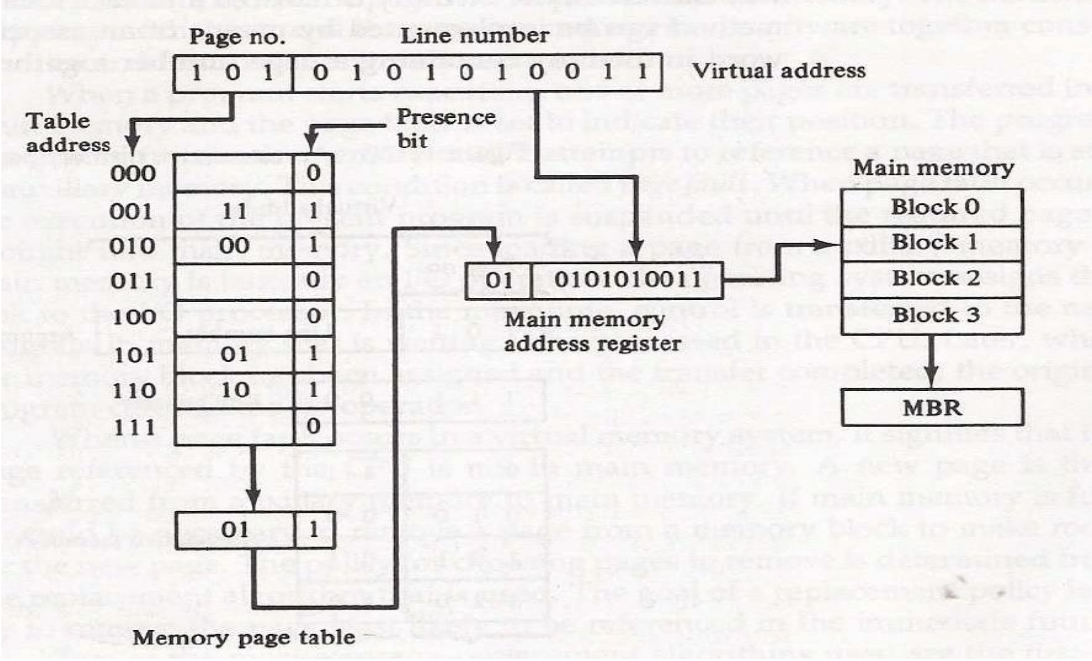


Fig3.10: Memory table in a Paged system.

- In a memory hierarchy system, programs and data first stored in Auxiliary memory. Portion of program and data are brought in to main memory as they are needed by the CPU.
- "Virtual Memory" is a concept used in some large computer systems that permits the user to construct programs through a large memory space were available, equal to the totality of auxiliary memory.
- Virtual Memory systems provides a mechanism for translating program generated address in to correct main memory locations. This is done dynamically, while programs are being executed in the CPU.
- An address used by the programmer will be called "virtual address" and the set of such address the address space.
- An address in main memory is called a location (or) physical address. The set of such locations is called memory space.