Memory Management – (cont…2)

Logical and Physical Address Space

The computer interacts via logical and physical addressing to map memory. **Logical Address** is the one that is generated by CPU and it is also referred to as **Virtual Address**. The program perceives this address space.

Physical address is the actual address understood by computer hardware, i.e. the memory unit. Logical to Physical address translation is taken care by the O/S.

The term **Virtual Memory** refers to the abstraction of separating **Logical Memory** (the memory as seen by the Program/Process) from **Physical Memory** (memory as seen by the Processor). Because of this separation, the programmer needs to be aware of only the logical memory space while the O/S maintains two or more levels of physical memory space.

In compile-time and load-time address binding schemes, these two tend to be the same. These differ in execution-time address binding scheme and the MMU (Memory Management Unit) handles translation of these addresses.

MMU (Memory Management Unit) is a hardware device that maps logical address to the physical address. It maps the virtual address to the real store location. The simple MMU scheme adds the relocation register contents to the base address of the program that is generated at the time it is sent to the memory.

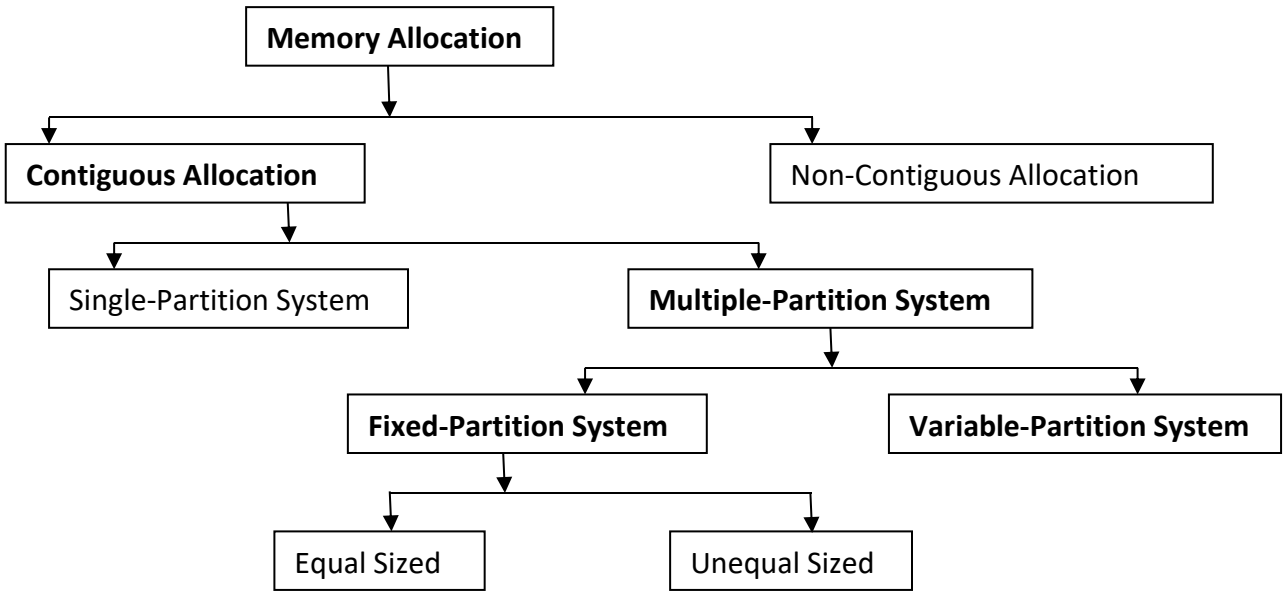
The entire set of logical addresses forms logical address space and set of all corresponding physical addresses makes physical address space.

Memory Allocation Strategies

In practical scenario, Operating Systems can be divided into several categories as shown below:

1. Single process system

2. Multiple process system – Fixed Partition memory and Variable Partition Memory.

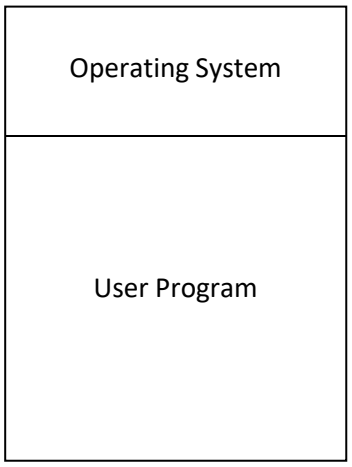


In an uniprogramming system, main memory is divided into two parts: One part is the operating system and the other part contains the program currently being executed. In multiprogramming system, the user part of memory is subdivided to accommodate multiple processes.

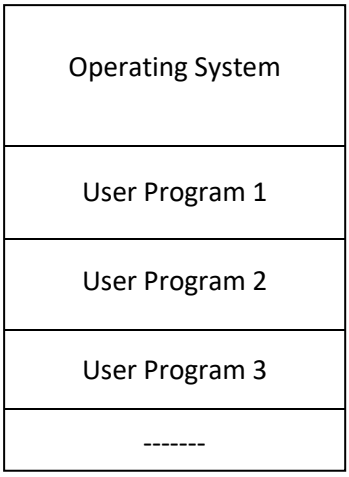
In uniprogramming system, only one program is in execution and only after completion of this program, another program may start. But in general, most of the programs involve I/O operations and it takes input from some input devices or displays the result in some output devices.

To utilize the idle time of CPU, we have shifted the paradigm from uniprogram environment to multiprogram environment, so that there exist a good mix of CPU-bound processes along with I/O bound processes and maximum utilization of resources gets established.

Partition of main memory for single process system and multi-process system is shown below:



Uniprogramming / Single Process system



Multi-programming / Multi-Process system

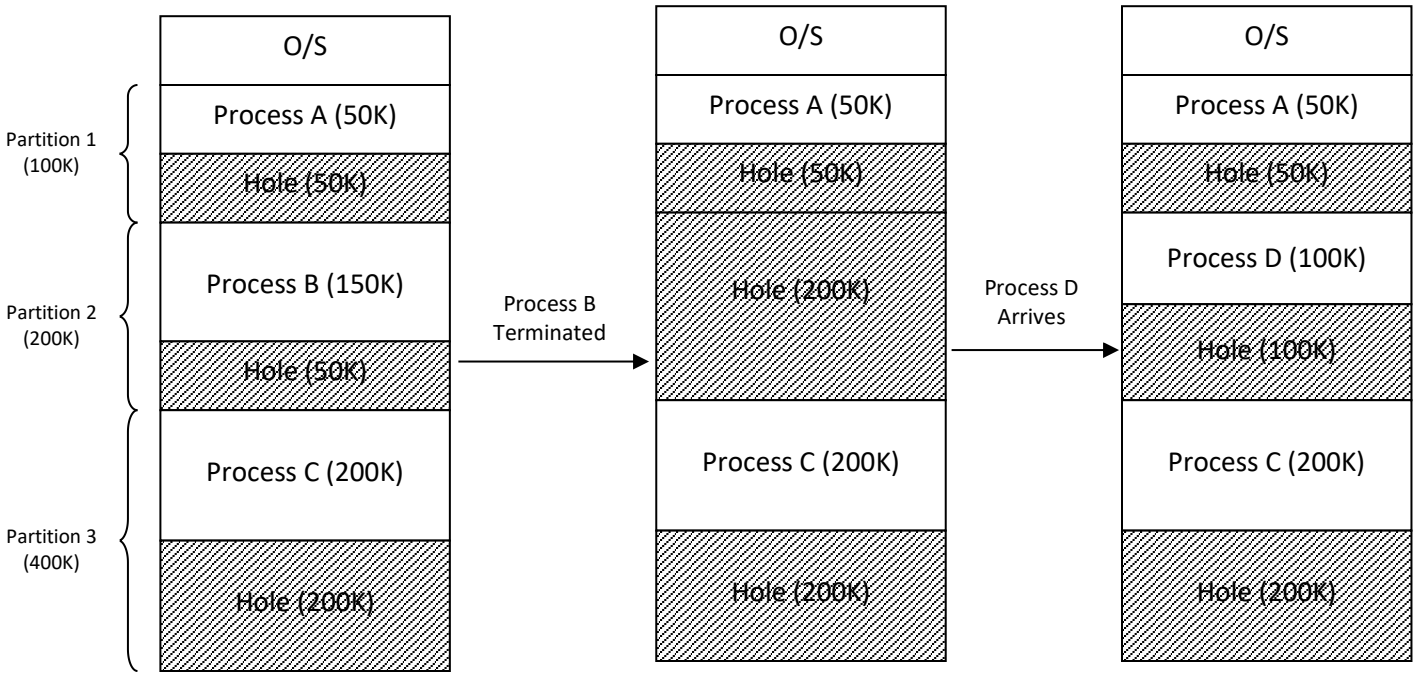
Multiple Partition System – Fixed Sized Partition

This is also known as **static partitioning scheme**. The entire memory is divided into n (possibly unequal) fixed-sized partitions having fixed boundaries, each of which can hold exactly one process. The degree of multiprogramming is dependent on the number of partitions. We can have one queue per partition or just a single queue for all the partitions.

Initially, whole memory is available for user processes and is like a large block of available memory. O/S keeps details of available memory blocks and occupied blocks in tabular form. O/S also keeps track on memory requirements of each process.

As processes enter into the input queue and when sufficient space for it is available, process is allocated space and is loaded. After its execution is over it releases its occupied space and O/S fills this space with other processes from the input queue.

The block of available memory is known as a Hole. Holes of various sizes are scattered throughout the memory. When a process arrives, it is allocated memory from a hole that is large enough to accommodate it.



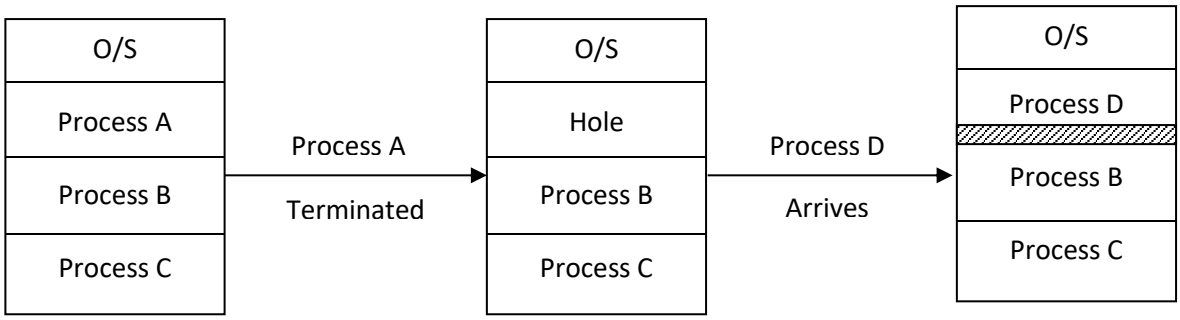
Fixed Sized Partition Scheme

This scheme suffers from fragmentation problem. Storage fragmentation occurs either because the user processes do not completely accommodate the allotted partition or partition remains unused. Any program, however small it is, occupies an entire partition.

This phenomenon, in which there is wasted space internal to a partition is known as **internal fragmentation**.

Multiple Partition System – Variable Sized Partition

This is also known as **dynamic partitioning scheme**, where the boundaries are not fixed. Processes accommodate memory according to their requirement. There is no wastage, as partition size is exactly same as the size of the user process. Initially when processes start, this wastage can be avoided but later on when they terminate they leave holes in the main memory. Other processes may avail these, but eventually they become too small to accommodate new jobs.

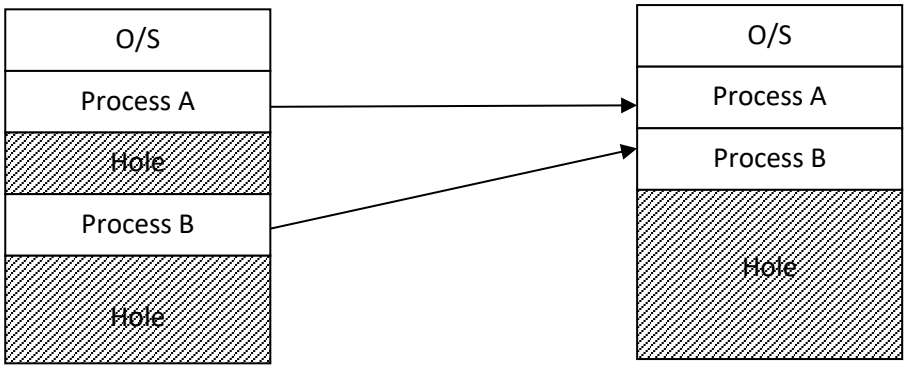


Variable sized Partitions

The program of Fragmentation remains in this case also. As time goes on and processes are loaded and removed from memory, fragmentation increases and memory utilization declines. This wastage of memory, which is external to partition is known as **external fragmentation.**

External fragmentation can be removed by **coalescing holes** and **storage compaction**. **Coalescing holes** is the process of merging existing holes adjacent to a process that will terminate and free its allocated space. Thus new adjacent holes and existing holes can be viewed as a single large hole and can be efficiently utilized.

There is another possibility that holes are distributed throughout the memory. For utilizing such scattered holes, all occupied areas of memory are shuffled to one end and all free memory space are left into a single large block which can further be utilized. This mechanism is known as **Storage Compaction**.



Storage Compaction

Storage Compaction also has the following limitations:

- It requires extra overheads in terms of resource utilization and large response time
- Compaction is required frequently, as jobs terminate rapidly.
- Compaction is only possible if dynamic relocation (relocation at run-time) is being used.

So in order to solve this fragmentation problems, we can either compact the memory making large free memory blocks, in expense of large overheads in terms of resource utilization, or implement **paging scheme** which allows a program's memory to be **non-contiguous**, thus permitting a program to be allocated physical memory wherever it is available.