MEMORY MANAGEMENT

Introduction

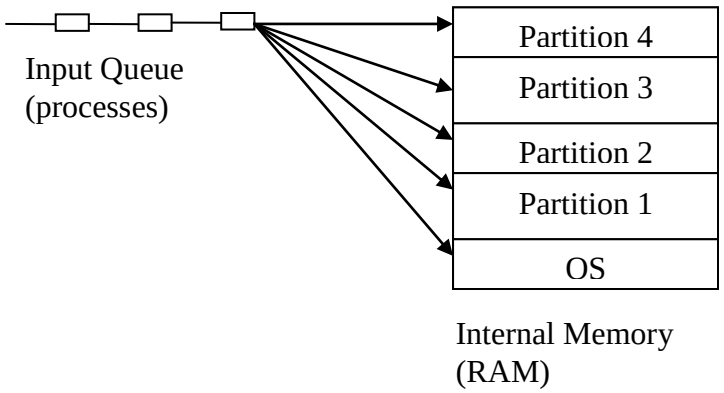
That part of the OS that manages memory is called the memory manager. Its job is to keep track of which parts of memory are in use, allocate memory to processes when they need it and manage the swapping of processes between the main memory and the hard disk when the memory is not big enough to hold all the processes.

One of the earliest memory management policies was to use virtual memory. The basic idea behind virtual memory is that the combined size of the program, its data and the stack may exceed the amount of physical memory available for it. What the OS does is to keep those parts of the program currently in use in memory and the rest on the disk. The two most common technique of memory are paging and segmentation.

In early systems, another more straightforward method was to use overlays. A large program would be split by the programmers into pieces called overlays. The first overlay will be loaded in memory, start running and when it was done it would call another overlay in memory. Although the OS did the actual work of swapping overlays in and out of memory, the splitting of the program into small modules was the programmer's responsibility.

Memory Maps / Memory Allocation Methods

a) Fixed Partition Memory

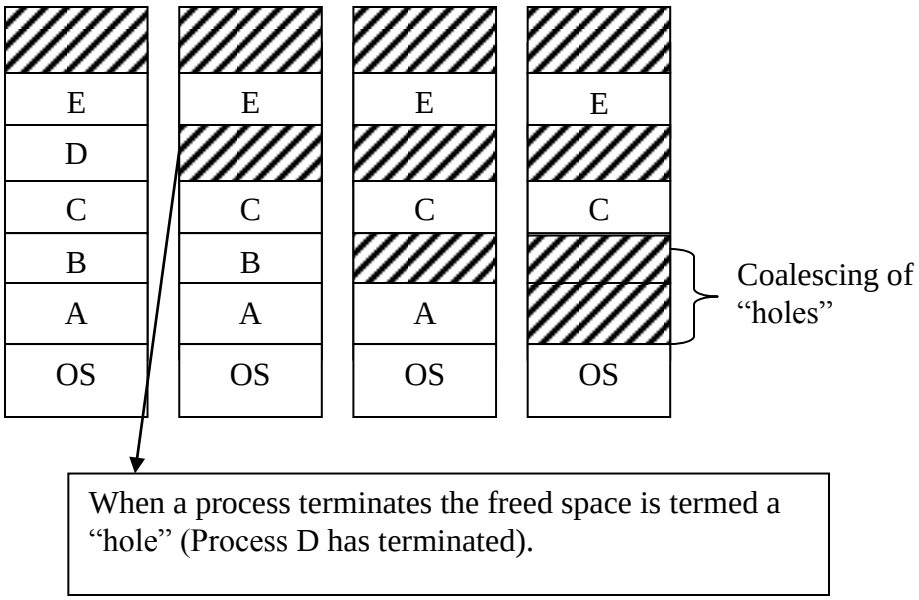


In a fixed memory partition system, the memory is divided into unequal length partitions and, as each partition becomes free, the job closest to the front of the queue that fits into the partition will be loaded into it and executed.

Note that since a job will not exactly fit a partition size, there will always be some wasted space referred to as **internal fragmentation**. Another disadvantage is that a process may not be able to run because it is larger than the largest available partition.

b) Variable Partition Memory

In this scheme, the partitions are allowed to be variable in size at load time. This is done in order to allocate the process the exact amount of memory it requires. Processes are loaded into consecutive areas until the memory is filled. When a process terminates the space it occupied is freed and becomes available for other processes.



The distribution of the free memory space (holes) in the way shown in the diagram is called **external fragmentation**.

It is possible to combine all the holes into one big chunk by moving the processes downward as far as possible. This is known as **memory compaction** and involves a substantial amount of processing since all active process would have to be suspended during the reshuffling.

Relocation and Protection

Multi programming introduces two problems that must be solved, namely *relocation* and *protection*.

Relocation can be solved in two ways:

1) by a **relocating loader**, which modifies the actual jump instructions so that the effect of the load address is taken into account.

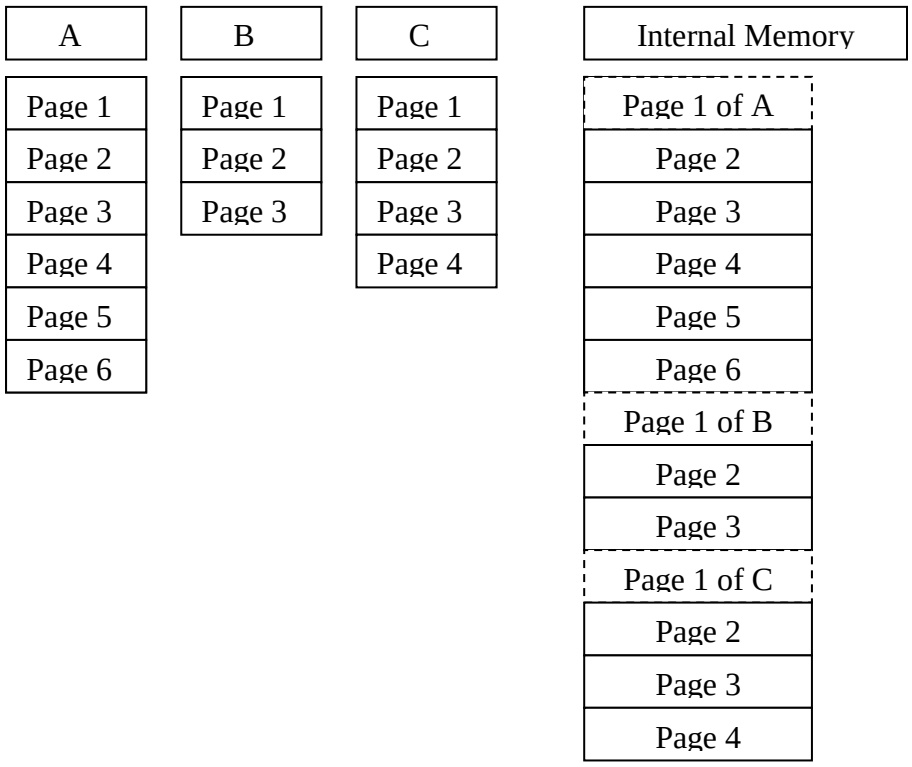
2) by the use of a **base register** (also called a **relocation register**). The base register is a CPU register which is loaded with the starting address in memory of where the program is in memory. All "jumps" are then made relative to this address, i.e. the *effective address = value of base + jump address*.

Note: Some systems also have a limit register. This would be loaded with the length of the partition (area in memory for storing the process). Addresses are then checked against the limit register to make sure they do not attempt to address memory outside the current partition.

Note also that the base register technique has the advantage over relocating loaders that a program can be moved in memory after it has started execution. After it has been moved all that needs to be done to make it ready to run is change the value of the base register.

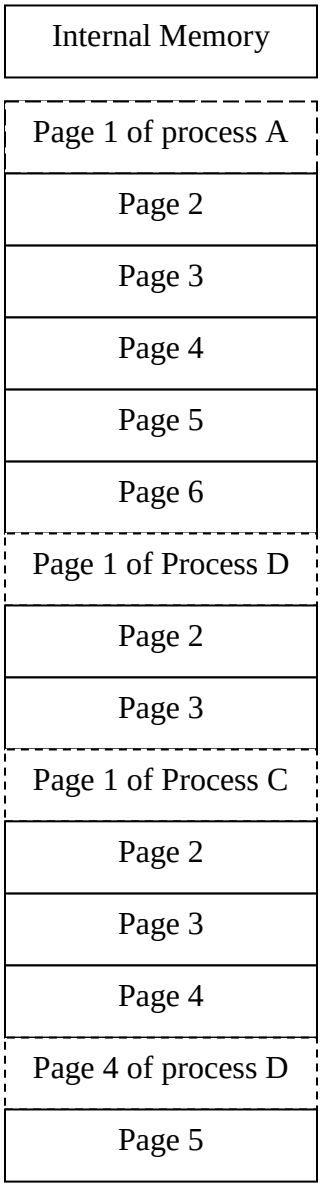
Paging

In a paged system each process is divided into a number of fixed size chunks called *pages*. The memory space is viewed as a set of *page frames* of the same size. The loading process now involves transferring each process page to some memory page frame.

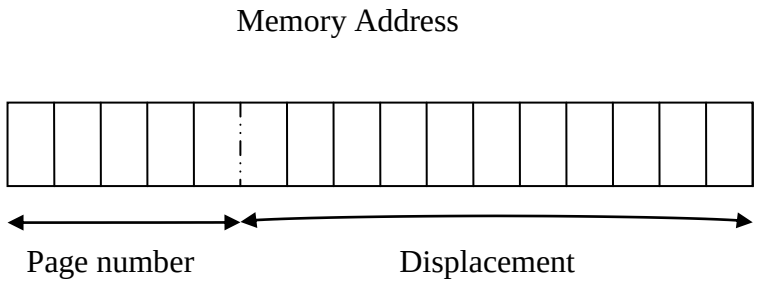


The above diagram shows three processes of different sizes. They are loaded in memory in consecutive locations.

If process B terminates and anther process (say process D) consisting of 5 pages needs to be loaded into memory, then we may end up with the situation shown below



With paging, an address is divided into 2 parts, a *page number* and a *displacement*, or *offset*. This is called **page addressing**.



In the above example, the top 5 bits are used for the page number and 11 bits are used for the displacement.
Hence with such a system, a maximum of 2^5 or 32 pages can be supported. Each page can be up to 2^{11} or 2048 memory locations long.

Note that the relocation problem can be solved by simply changing the page number when a page is loaded into a different page frame.