Memory Management

We know that the memory in the computer system is a large array of words or bytes, each location with its own address. Interaction is achieved through a specific reads/writes of specific memory address. The CPU fetches the program from the hard-disk and stores it in the memory. if a program is to be executed, it must be mapped to absolute addresses and loaded into memory.

In a multiprogramming environment, in order to improve the CPU utilization, performance and throughput, several processes must be kept in the memory. Different algorithms are used to manage the operations of the memory. The main tasks of the O/S in connection to memory management are:

- Keep track of which part of the memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available and
- Allocate and de-allocate memory space are needed.

The part of the O/S that performs this vital task of memory management is known as memory manager.

Overlays and Swapping

Programs reside in the disk in form of executable files, which needs to be brought into the memory and placed within a process. Such processes form the ready queue, from which the selected process gets the CPU time for execution.

During these stages, addresses may be represented as source code address or in symbolic form (e.g. LABEL). Compiler will bind these symbolic address to relocatable addresses. The linker will bind these relocatable address to absolute addresses.

Now before we load the program in the memory, we must bind the memory addresses that the program is going to use. **Binding** is basically assigning which address the code and data are going to occupy. This Binding can be done during the compile-time, Load-time or during Execution-time.

Compile-time: If memory location is known before hand, absolute address can be generated.

Load-time: If memory location is not known, this relocatable address may be generated during load-time.

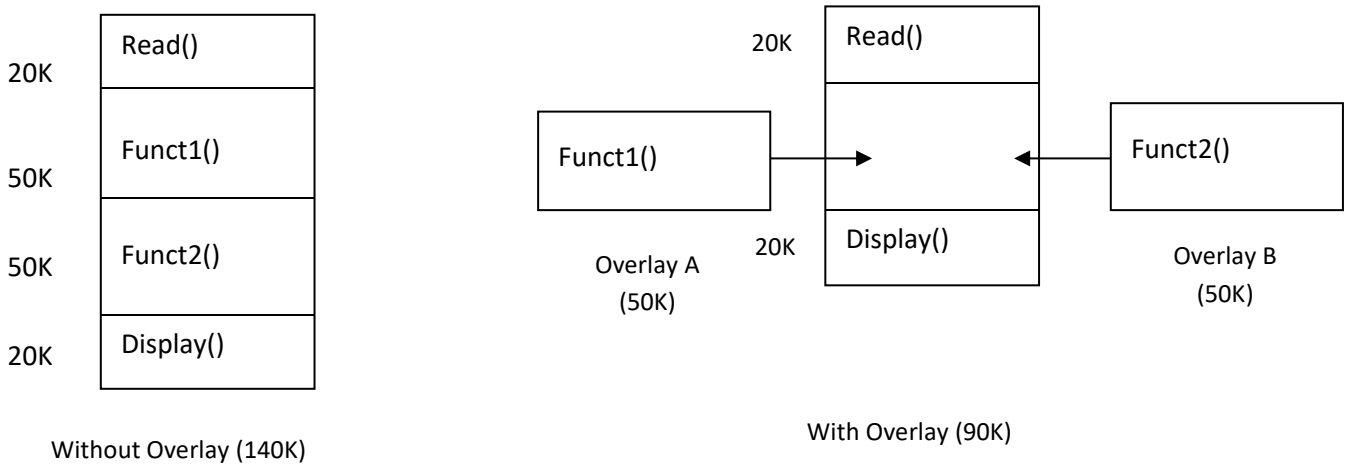
Execution-time: In this case the binding is delayed until run-time, as because the process can be moved during its execution. We need hardware support for address mapping (the base and limit registers).

A large program may have several modules in it. For better memory utilization, all modules can be kept on disk in a relocateable format and only main program is loaded into memory and executed. Only when required, the other modules are called, loaded and address is updated. Such schemes is called **Dynamic Loading**.

Overlays

Similar to Dynamic loading, we can use the concept of **Overlays**. Here the entire program or application is divided into instructions and data sets such that when one instruction set is needed it is loaded in memory and after its execution is over, the space is released. The other instructions are loaded into the same space which has been released. Such instructions which can be loaded and unloaded by a program are called overlays.

Thus **Overlays** are part of a single application, which has been loaded at same origin where previously some other part(s) – or overlays – were residing. A program based on overlay scheme must have a root piece, which is always memory resident and a set of overlays.



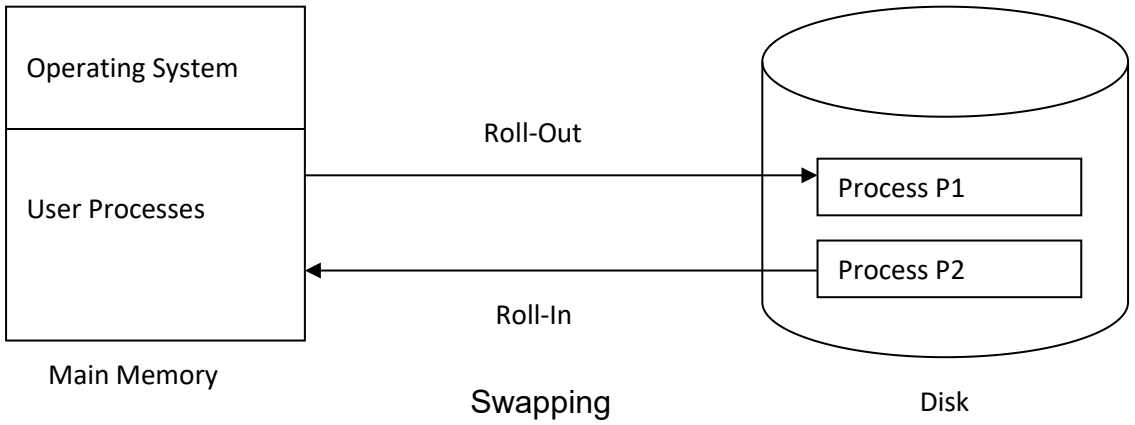
Example of Overlay

Overlay gives a program a way to extend limited main storage. It must be noted that the overlays should be written in such a way that they follow the rule of mutual exclusion and they should not call each other.

Swapping

Swapping is an approach for memory management by bringing each process in entirety, running it and then putting it back on the disk, so that another program may be loaded into that space.

Swapping is a technique that lets us use the disk file as an extension of memory. Lower priority user processes are swapped back to the disk and this is called **Roll-out Swapping**. Swapping the process back to memory when some event occurs – may be in a different location, is called **Roll-in Swapping**.



Swapping

Advantages of Swapping:

- Allows high degree of multiprogramming
- Allows dynamic relocation. Address binding is done during execution time, so that we can swap in the processes at different locations.
- Can be easily applied to priority-based scheduling algorithms to improve performance.
- Better Memory utilization.