Paper Name : Operating System

Prepared by : Debanjali Jana

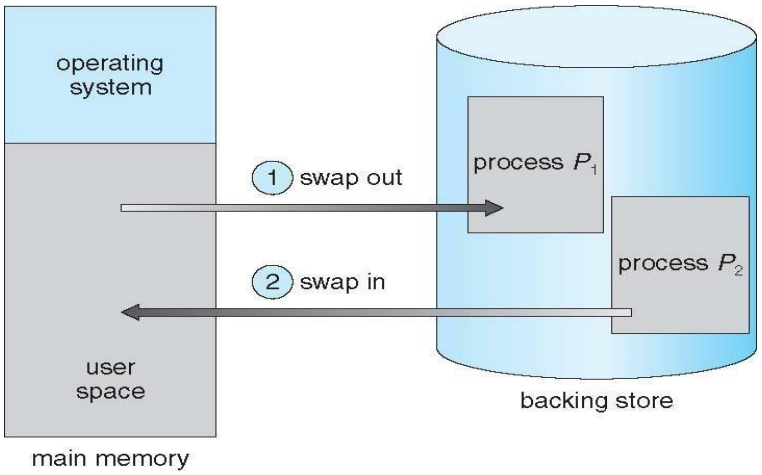
Swapping :

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
    - Total physical memory space of processes can exceed physical memory
- **Backing store** –. Swapping requires a backing store. The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is sometimes called roll out, roll in.

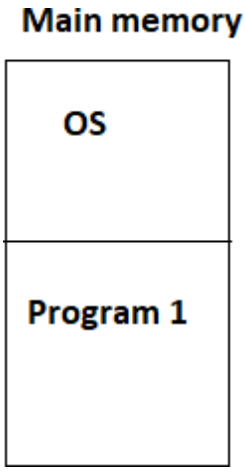
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a ready queue of ready-to-run processes which have memory images on disk

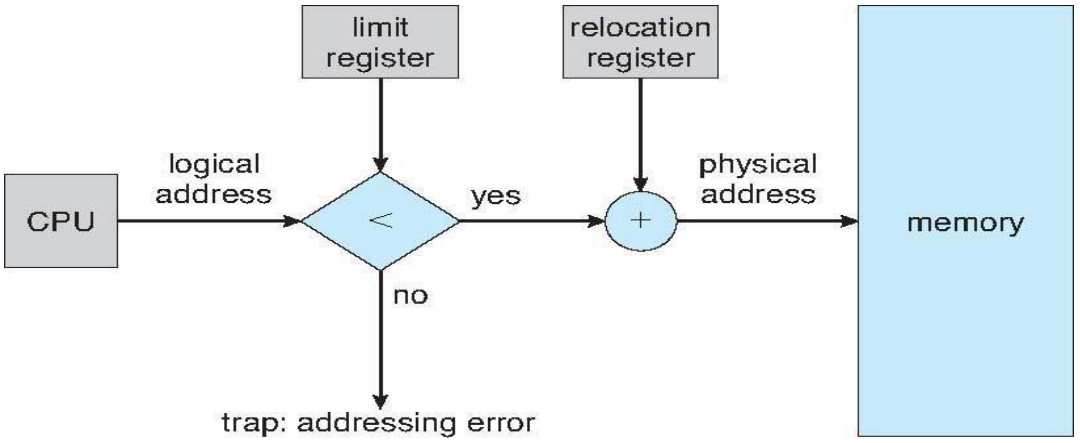
For example, assume a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the memory space that has been freed. In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. In addition, the quantum must be large enough to allow reasonable amounts of computing to be done between swaps.



Single partition allocation scheme:

The hardware support for implementing this scheme includes a relocation register and a limit register. The relocation register contains the value of the smallest physical address; the limit register contains the range of logical addresses (for example, relocation= 100040 and limit= 74600). With relocation and limit registers, each logical address must be less than the limit register; the MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory. When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Because every address generated by a CPU is checked against these registers, we can protect both the operating system and the other users' programs and data from being modified by this running process.





Multiple partition allocation scheme:

In this type of allocation, main memory is divided into a number of equal/unequal partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

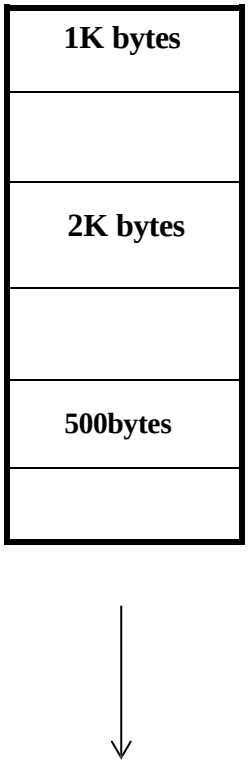
- Variable-partition sizes for efficiency (sized to a given process' needs)
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Process exiting frees its partition, adjacent free partitions combined
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)

Allocation Policies :

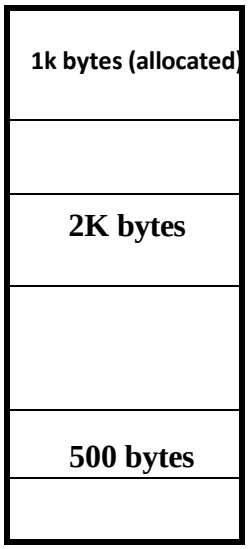
How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough. It allocates the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition. Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.

  To allocate *n* bytes, use the *first* available free block such that the block size is larger than *n.*

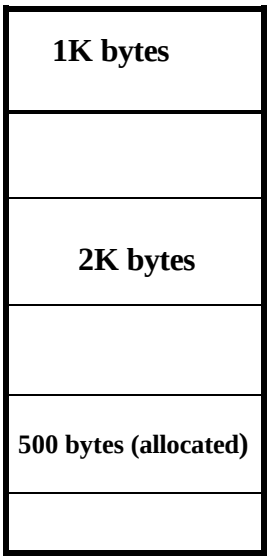


To allocate 400 bytes, we use the 1st free block available (smallest)



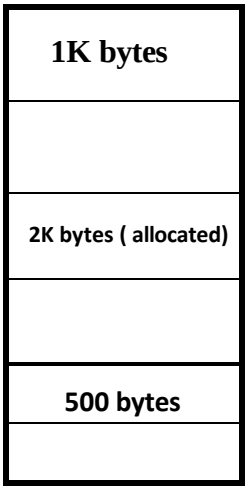
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; it searches the entire list, unless ordered by size, produces the smallest leftover hole thus considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.
  To allocate *n* bytes, use the *smallest* available free block such that the block size is larger than *n.*

To allocate 400 bytes, we use the 3rd free block available (smallest)



- **Worst-fit**: Allocate the ***largest*** hole; we must also search entire list unless it is sorted by size , produces the largest leftover hole.

To allocate 400 bytes, we use the 2nd free block available (largest)



To allocate *n* bytes, use the *largest* available free block such that the block size is larger than *n.*

Implementations have shown that both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization. Neither first fit nor best fit is clearly better than the other in terms of storage utilization, but first fit is generally faster.