Simulation of Memory Management

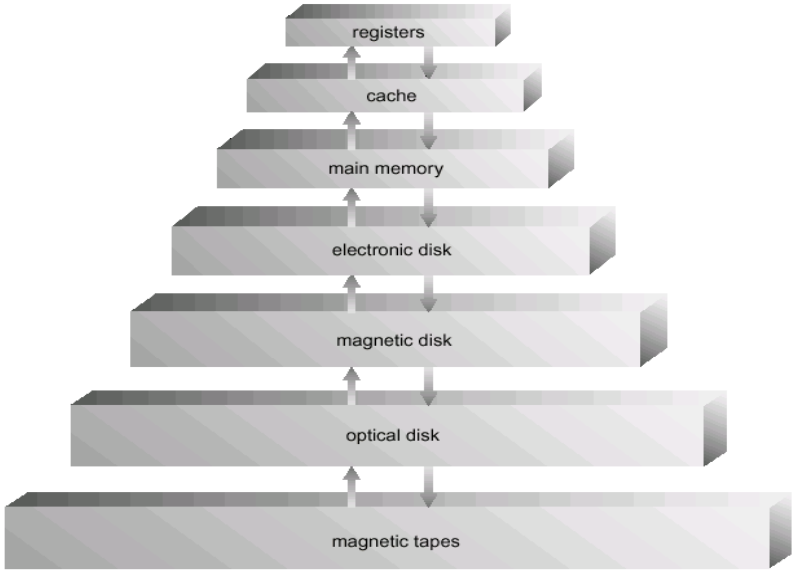
Memory Management:

Memory management is the management of the contents of processor memory (or storage). Applications and data must be in processor memory before they can be used. As the workload on a system increases, memory use also increases. Memory management is concerned with optimizing the use of processor memory to try to keep the applications and data that are being used in memory. Memory is divided into sections, called frames and allocated contiguously. When the process needs them, they must be allocated to the process. Memory is allocated to the processes that are in the ready queue.

We need memory management for following reasons:

- Memory is a scarce resource.
- Management of memory hierarchy
    - Ease of programming
    - Efficient use of memory
    - Multiprogramming
    - Security

Storage Hierarchy:



Storage systems organized in hierarchy according to speed, cost, and volatility. In this hierarchy, our project is dealing with the main memory and processes.

Memory Allocation:

Main memory usually divided into two partitions:

One for operating system which is usually held in lower memory with interrupt vector and user processes held in higher memory.

When a process requests memory, the process scheduler allocates memory from a block of available memory called *HOLE* which are scattered throughout memory.

The algorithms that are used for memory allocation are:

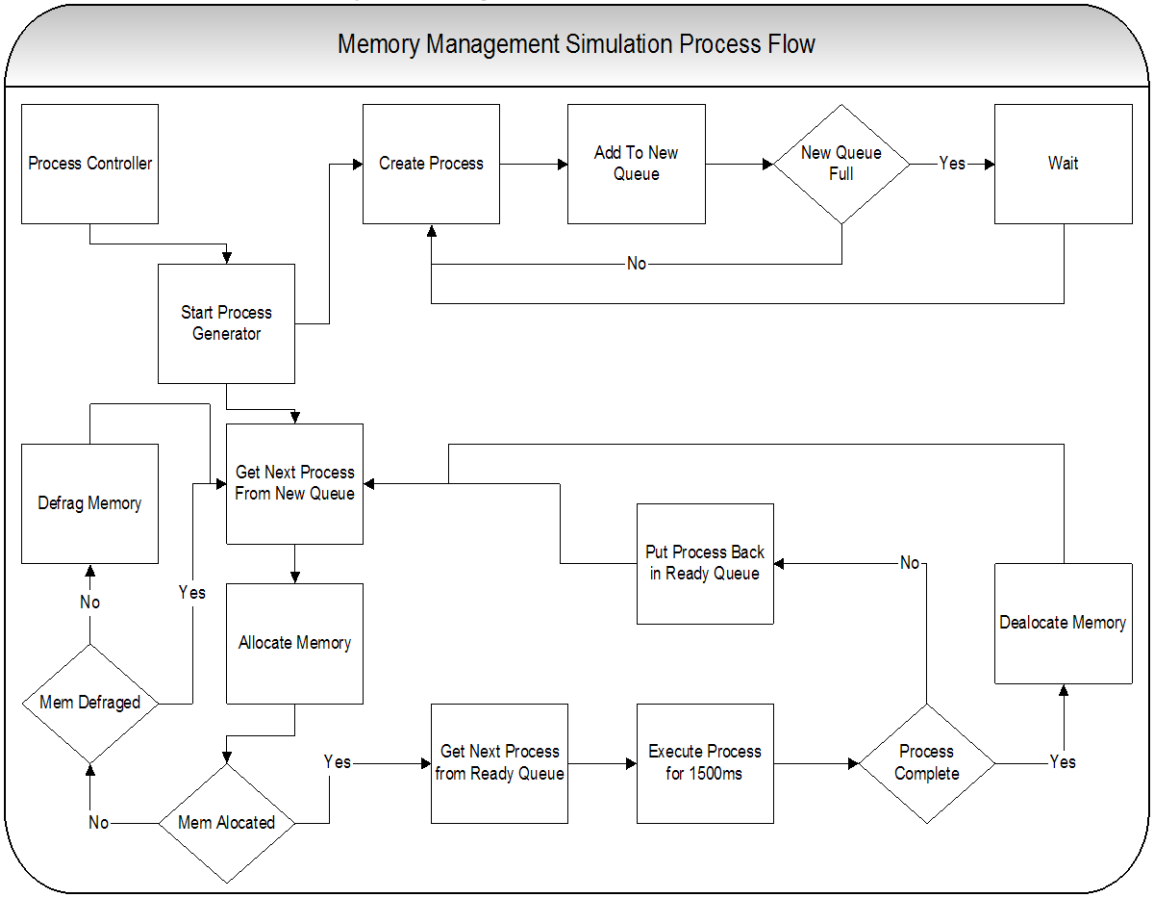
- First Fit: Allocates the first available memory block that is big enough.
- Best Fit: Allocates the smallest available memory block that is big enough.
- Worst Fit: Allocates the largest available memory block.

We have used the first fit algorithm to allocate memory.

De fragmentation:

Memory is de fragmented by Shuffling processes around to eliminate "holes" in memory space, in order to allow new process to be loaded into memory. Defragger itself is a process which takes up system resources. It has the whole memory space mutually exclusively, when it's de fragmenting.

Simulation Of Memory Management Process Flow Chart:



Classes used in Memory Management Simulation:

The Process Class

The Process class extends the thread class and implements the run method which is a busy loop with a 1ms cycle. This cycle continues for the total run time of the process. This loop can be interrupted at any time by calling the pause method. The pause method causes the process thread to wait until the notify method is called. When a process completes its total run cycle it sets a Boolean flag "Return" to true. Each process has a PCB class object to hold state information. When a process is created a TotalRunTime with a minimum of 1000 and a maximum of 25000 is randomly assigned, as well as a random TotalMemoryRequired with a minimum of 4096 bytes and a maximum of 409600 bytes.

The PCB Class

The process control block is used to store the process state data.
The PCB class has two methods – GetNumFramesUsed and SetFrames.

GetNumFramesUsed returns an integer value representing the number of frames used by the process. SetFrames accepts two variables, the first is the start frame and the second is the end frame. The SetFrames method sets the corresponding variables in the PCB object.

The PCB contains six public variables that hold the state information of the process, which are discussed below.

The Process ID is a unique integer from 0 to 99. This is assigned at the time the PCB is created. TotalMemoryRequired is the total memory needed by the process in bytes. TotalRunTime is the total time the process will run for before completion. CurrentRunTime is the total time a process has been running. StartTime is the time a process is started (not implemented in our simulation). StartFrame is the first memory frame assigned to the process by the memory manager. EndFrame is the last memory frame assigned to the process by the memory manager.

The ProcessGenerator Class

The ProcessGenerator class extends the thread class and implements the run method, which is a loop that creates processes and adds them to the new queue. This loop waits once the new queue contains five processes. The wait is interrupted when a process is removed from the new queue so that a new process may be generated.

The ProcessScheduler Class

The ProcessScheduler Class has one method, Schedule, which retrieves the next Process from the new queue and attempts to allocate memory for it. If no memory can be allocated for the Process, then the memory defragmenter is run if the memory has not yet been defragmented. The ProcessScheduler then retrieves the next Process from the new queue and attempts to allocate memory for it. This continues until all the Processes in the new queue have been visited or a Process is successfully allocated memory.

The ProcessController Class

The ProcessController Class is much like an operating system kernel. The Process controller is responsible for starting the ProcessGenerator, processing the ready queue in a round robin fashion, and executing the scheduler before each process is executed. The ProcessGenerator is started by calling the start method, which instantly returns. After the ProcessGenerator is started the scheduler is given a chance to run. When the scheduler completes the ProcessController removes the next Process from the ready queue and executes it. The Process is allowed to execute for either 1500 ms or the remainder of its total run time, whichever is less.

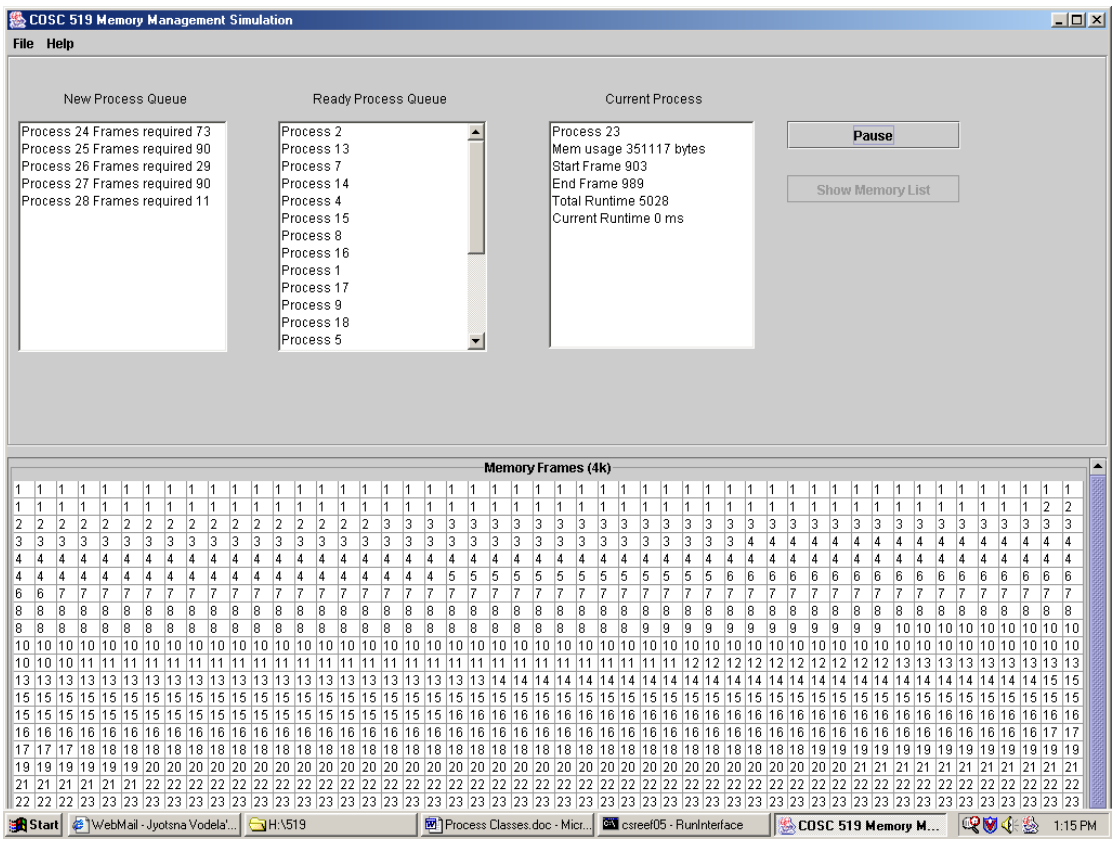
The Defragmentor Class:

The Defragmentor class takes the processes in ready queue and sorts them to store all processes in the order they physically appear in the memory space, then the memory has been defragged and allocated to the processes in ordered process array.

The Memory Manager Class:

The memory manager class takes PCB as the input parameter it is used to allocate and deallocate memory using attributes of PCB .The memory is simulated as a linear linked list. It sets the Start Frame and End Frame parameters using "First Fit" algorithm.

Application Interface for Memory Management:



The Interface shows the status of each process from being generated to the end of the process and memory used by the process.

Explanation of each field in the interface:

New Process Queue:

Displays the processes that are sit in the new queue and frames needed for each process.

Ready Process Queue:

Displays the processes that are in ready queue which are allocated memory and executed using round robin algorithm.

Current Process:

Displays current process being executed by CPU.

It displays following processor specifications:

1) Process Id: Process Identification number being generated.
2) Memory Usage (in bytes):Amount of memory required by each process being generated.
3) Start Frame: It shows the starting frame of process when it is loaded into the main memory.
4) End Frame: It shows the last frame of process when it is loaded into the main memory.
5) Total Runtime: It shows the amount of time it takes to run the process.
6) Current Runtime: It shows amount of time currently ,it takes to run the process .

Memory Frames(4K):

Displays allocated memory frames of each 4k to the processes.

Conclusion:

Memory Management is one of the main tasks of the operating system. In this we can implement many of the features. Due to time constraints ,it's a bit impossible for us to make a detailed and advanced memory management implementing concepts like virtual memory and paging.