PROJECT REPORT

MEMORY MANAGEMENT

DATE :

GROUP MEMBERS:

MEMORY MANAGEMENT

PURPOSE :

To simulate memory management of a simple system in order to better comprehend the complex memory management system of a multi-user operating system.

Understanding of Problem:

When a user executes a program, the operating system creates an address space for it to run in. This address space will include the instructions for the program itself, as well as any data it requires. In a system with memory protection, each process is restricted by the operating system to accessing only the memory in its own address space. However, the combined program memory requirements often exceed the system's amount of physical memory (RAM) installed on a computer. So, modern operating system use a portion of the hard disk called a swap file to extend the amount of available memory.

A computer memory must accommodate both the Operating System processes and user processes. The following outline describes what happens when a user starts a program.

1. The operating system creates an address space for the program to run in, which involves setting up appropriate page tables.

2. The program's text is mapped as read-only and data segments are mapped as read-write from the executable file on disk to the virtual address space.

3. The stack and heap are initialized.

4. Control is transferred to the program.

5. The CPU tries to access the first instruction. Since it is not in memory, a page fault is generated. This causes the operating system to intervene and allocate

one or more pages of physical memory and to copy the code in from the file.

6. The CPU then returns control to the program, which continues executing until it encounters another page fault. At this point, step 6 is repeated. If there is no free physical memory, the operating system will have to either discard (if read-only) or swap out (write data to swap file if read-write) pages before they can be reused. Typically an LRU (least recently used) algorithm is used for deciding which page(s) get tossed.

7. This pattern continues for the life of the program.

Steps for simulation of Memory Management:

ALLOCATION:

- Contiguous, Fixed, Equal Sized, Multiple-Partition Allocation
- Main memory is partitioned into a set of equal sized non-overlapping blocks. Any process whose size is less than or equal to a partition size can be loaded
  into the partition
  - In our simulation we are referring to our blocks as frames.
  - We have 100 frames that are 128 bytes in size.
  - We can change the amount and size of our frames.

- When a process arrives it is allocated a block of available contiguous memory, large enough to accommodate the process
- If all partitions are occupied, the operating system can swap a process out of a partition.

PLACEMENT ALGORITHM:

- The OS must decide which hole to fill when allocating memory to a process

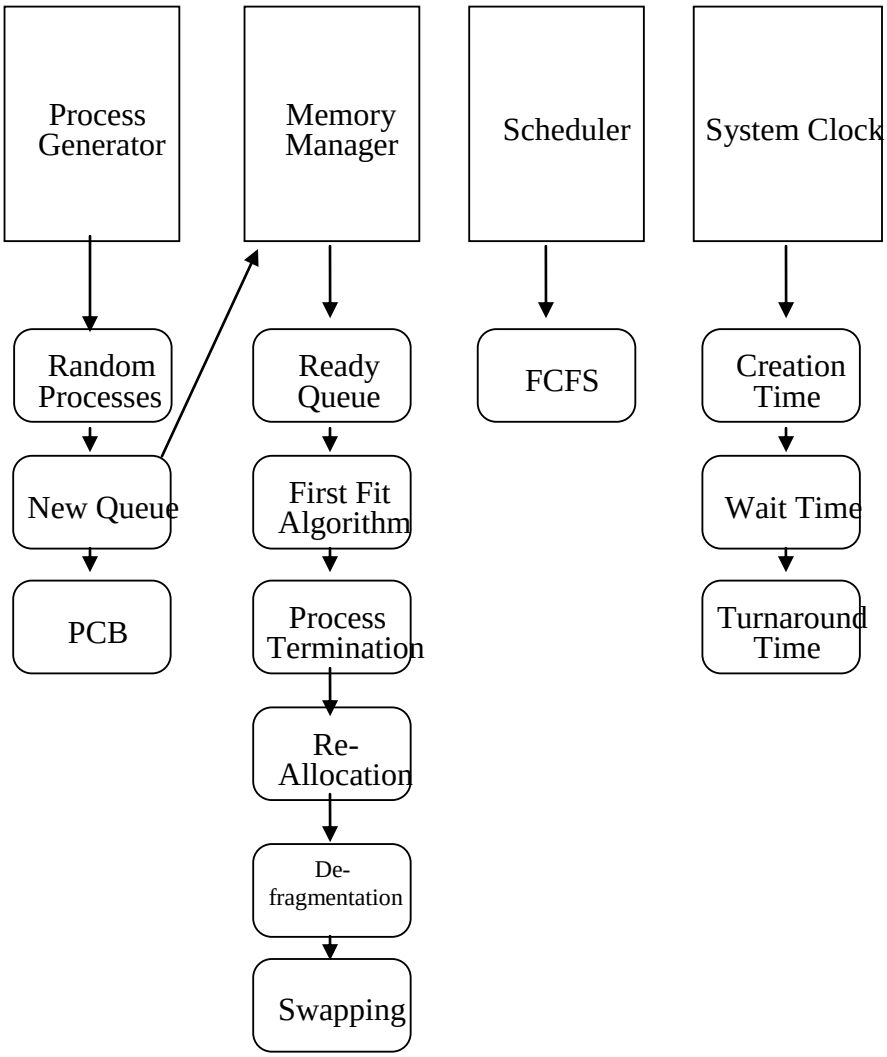
- Best fit: Find the smallest hole that satisfies the allocation

- Worst fit: Use the biggest free region
- Next fit: like first fit, but search starting from the last allocation

- First fit: Use the first region in memory that satisfies the allocation

- First-fit and best-fit better than worst-fit in terms of speed and storage utilization.
- For our simulation we are utilizing the first fit placement algorithm.

FLOW CHART :



The Process Generator:

- creates random processes with random memory requirements at random times.
- defines the execution time required by the process.
- places random processes into the new queue.
- creates and maintains information about each process which is the process control block or PCB.
- The number of frames required is determined by dividing the amount of memory required by the frame size.
    - number of frames = (required frames / frame size)

The memory manager:

- loads the randomly created processes in ready queue.
- allocates the first available memory block of frames to each process (first fit algorithm)
- terminates each process when the scheduler sends a termination The space is then re-allocated back to main memory.
- de-fragments the holes left in memory.
- controls the swapping process.

Scheduler

- The scheduler utilizes the FCFS algorithm
- First Come First Serve algorithm
- Each process that enters the ready queue is executed based on its arrival to the queue.
- Only one process can be executed at any given time.
- The process that is executing must complete execution in entirety before another process can be executed.

System Clock

- To determine when the process finishes its execution.
- To measure calculations such as: total wait time per process, total turnaround time per process, average wait time for all processes and average turnaround time for all processes.
  - Waiting time – amount of time a process has been waiting in the ready queue
  - Turnaround time – amount of time to execute a particular process

FRAGMENTATION:

- Fragmentation makes available memory useless by breaking it into dis-contiguous pieces too small to use. There are 2 categories of memory fragmentation:

- **External Fragmentation**

  - ♦ Total memory space exists to satisfy a request, but it is not contiguous.

  - ♦ Reduce external fragmentation by compaction (de-fragmentation)
  - ♦ Shuffle memory contents to place all free memory together in one large block.

- **Internal Fragmentation**

Allocated memory may be slightly larger than requested memory; this size difference is internal to a partition, but not being used.

SWAPPING:

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.

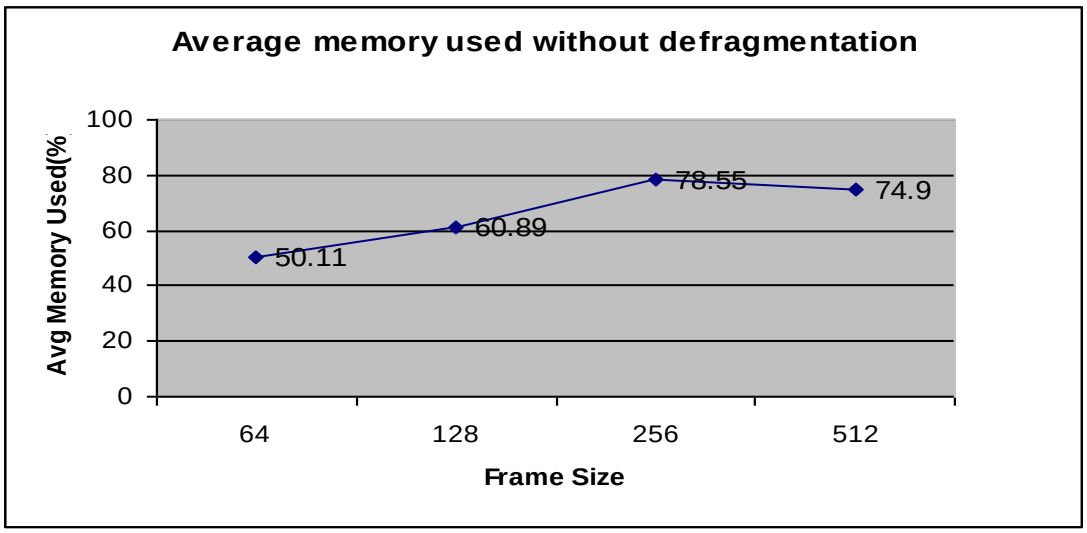
- For our simulation we are illustrating the swapping procedure. When we swap we are creating a new process with the highest priority which overrides all other

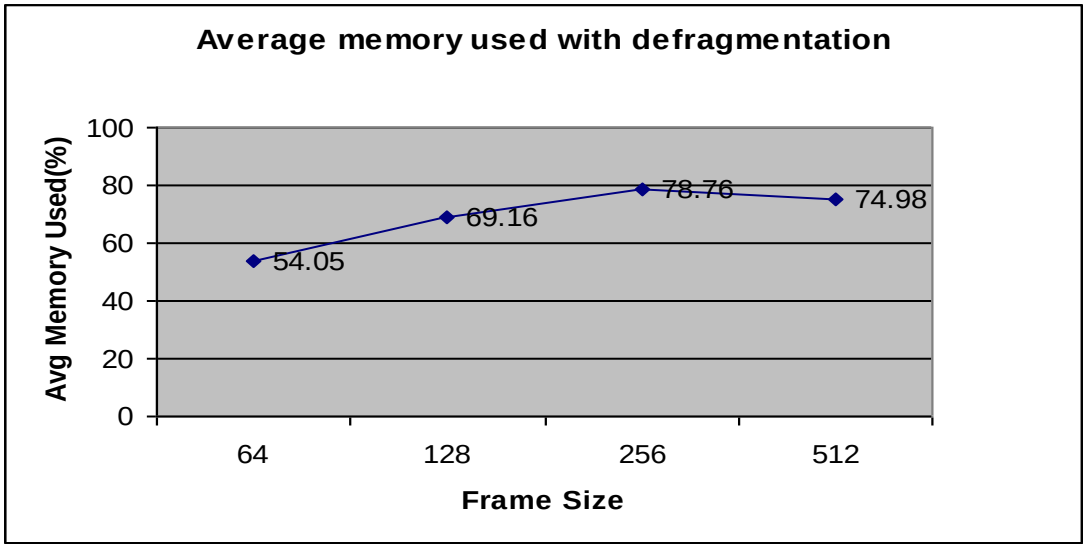
processes in the new queue and it goes directly to ready queue.

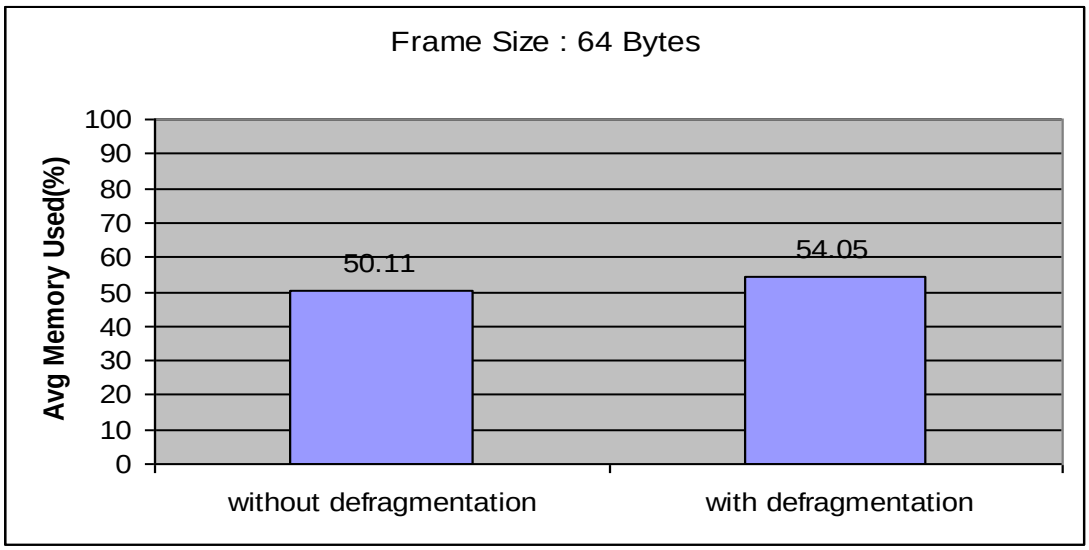
- If there is no space available in main memory then swapping will pull out process from memory and place into new q to make main memory available for new highest priority process

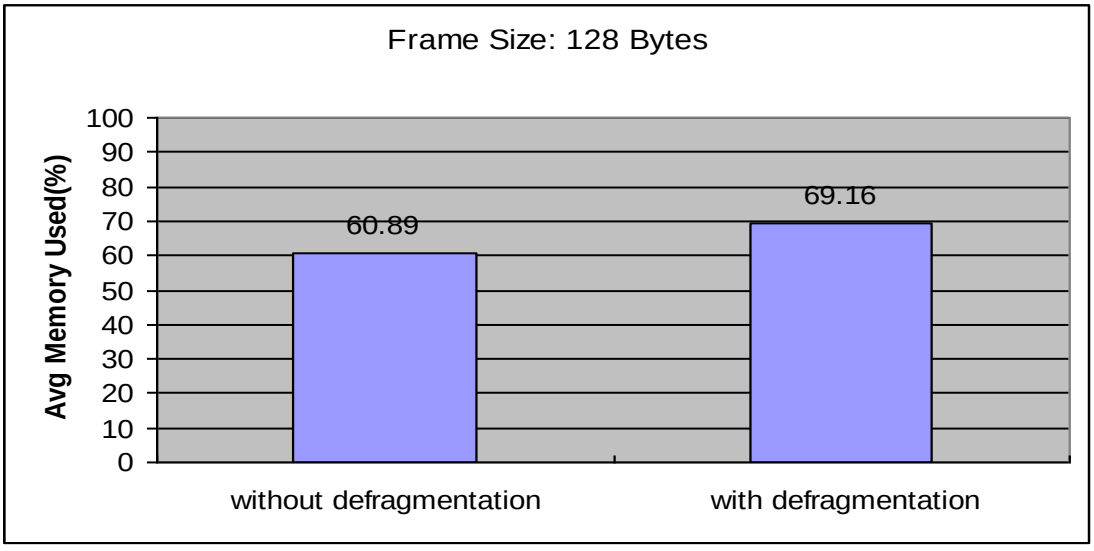
GRAPH EXPLATION:

The processes generated by the process generator are from 0 to 6000 bytes in size. The average memory used(%) for our program is based on the total process size in the ready queue at every given time. For example, suppose we have 100 frames with 128 bytes of each frame. If we have 3 processes in the ready queue with size 400, 1600 and 2000, then the average memory used will be 32%. With bigger size of the frame will the memory manager bring more processes into the ready queue(since we have more space in our main memory). Notice that 256 bytes is the optimal frame size for our project. If we make the frame size to 512 bytes, then there are 51200 bytes in the main memory. It is too big compare to our process size. It will be a waste of the memory because we don't have enough processes to make the memory full. As you can see this effect on the chart that the average memory used decreases when the frame size becomes to 512.




Average memory used with defragmentation
Avg Memory Used(%)
100
80
60
40
20
0
54.05
69.16
78.76
74.98
64
128
256
512
Frame Size


Frame Size : 64 Bytes
Avg Memory Used(%)
100
90
80
70
60
50
40
30
20
10
0
50.11
54.05
without defragmentation
with defragmentation


Frame Size: 128 Bytes
Avg Memory Used(%)
100
90
80
70
60
50
40
30
20
10
0
60.89
69.16
without defragmentation
with defragmentation

conclusion:

Memory management is a huge project, at present owing to the time and ability limit, it is a bit impossible for us to make a detailed and advanced memory management system. What we have done so far is a simulation of a simple Memory management. In our project we have divided the total memory into certain frames based on the given frame size, assigned frames needed based on the process size and run one process at a time based on FCFS. The processes executing states and total memory utilization can also be clearly indicated. For the best optimization of memory we used external de-fragmentation (compaction). Sometimes we need to swap a process out of the memory when there is not enough space for new process with a higher priority and then place it in the memory to complete the execution (roll out and roll in).

By doing this project we understand how the actual memory management works for the real computer system, since we are not generating addresses from the CPU so we didn’t implement the paging concepts.