**Assignment 2: Memory Management**

**Purpose:**

The purpose of this project is to stimulate the memory management process and effects of limited memory. Our simulator will read the input files and then will simulate the execution of the processes according to total memory block size and page size by MM(Memory manager). It will generate the output into terminal window itself. The output contains memory map and input queue status with turnaround time.

|  |
| --- |
|  |
|  |  |

**Input file:**

in1.txt(Hardcoded into main.cpp)

**Output:**

Generated into terminal.

Validate it with files provided as sample out1.txt, out2.txt, out3.txt for in1.txt provided.

**Pseudocode:**

**//Start of main**

Main(){

read input file in1.txt

processQueue(queue)🡪 add processes as they are read in from input file

inputQueue(vector)🡪 processes in input queue (actually a vector) are processed front-to-back, removing from this "queue" and adding to MMU if MMU has enough space for that process

criticalPoints(vector)🡪 keep track of points where we have to write to output

processesInMemory(vector)🡪 keep track of which processes are currently in memory map

memblocks(vector)🡪 MMU, composed of vector of memblocks

keep track of which processes are currently in memory map

Select page size(1=100,2=200,3=400)

Initialize MMU with page size and memory size

addUniqueCriticalPoint() //add arrival time to criticalPoints if arrival time not already in criticalPoints

//go through criticalPoints and process arrival/completion events

While criticalPoints not empty{

// move process from processQueue to inputQueue if criticalPoint matches arrival time, else it must be a completion time

printInputQueue() //Prints the input queue

}

for(all values in processesInMemory){

// check if any processes in MMU completed processing then free up memory from MMU and erase it

//print which process completes processing

printMMU() //Prints Memory Map

}

For(all values in inputQueue){

// check if there is enough memory in MMU for any process in inputQueue

// move process from inputQueue to processesInMemory

// move iterator back after erasing front element

addUniqueCriticalPoint()//add arrival time to criticalPoints if arrival time not already in criticalPoints

Print which processes moves to memory

// decrease currentProcess.memoryNeed until all needed pages are used

printInputQueue() //Prints the input queue

printMMU() //Prints Memory Map

}

If only processes are arriving {

printMMU() //Prints Memory Map

}

// keep criticalPoints sorted in ascending order

//Print total turnaround time

}

**//End of Main**

**//Funtion** **printInputQueue**(){

// argument passed by value, so original inputQueue not changed

// Function prints the input queue with proper format

}

**//Funtion** **printMMU**(){

// Function prints the memory map with proper format

}

**//Funtion** **addUniqueCriticalPoint(**){

This function add critical points of time for process start/end/processing period

}