CS-570 Advanced Operating Systems
Assignment 3: Memory Management
Deadline: Wednesday 25th of April, 11:55pm

Instructions

- This is an individual assignment.
- You can use C, C++, Java, Python, C# or any other programming language that you are comfortable with.
- For full credit, you must follow the exact format of the input and output files as given in the examples.
- For late submissions, there is a 25% deduction per day.
- There will be viva at the end.
- You must submit your own work. Copying from any source is not allowed. All plagiarism cases will be forwarded to the Disciplinary Committee.

Deliverables

- At least three input files with different workloads.
- 1 output file that for each Memory Management Policy(MMP) (Total five output files).
- One report explaining which MMP works best in which case and your analysis about the different policies.
- Code.
- Readme file that explains how to run the simulation and any other assumptions, if any.

Overview

In this assignment you will code a simulator that explores the effects of limited memory and memory management policies. Your simulator will read policy information and the characteristics of the workload from input files and then will simulate the execution of the processes as well as the decisions of Memory Manager (MM). Your simulator will generate an output file with the trace of important events (explained below), as well as the memory map and the input queue status after each event. At the end of the simulation, your program will also print the average turnaround time per process. The details of the assignment are given below. Please read all the parts of this handout carefully before starting to work on your assignment.

Policies of Memory Manager

Your simulator will prompt the user for the size of the memory and the Memory Management Policy (MMP) and its associated parameters. The Memory Size (an integer) denotes the capacity of main memory in your simulation (This integer will denote the size in Kbytes). The MMP can be one of the following:

1. Variable-Size Partitioning
    a. First-Fit
    b. Best-Fit
2. Paging
3. Segmentation
    a. First-Fit
    b. Best-Fit

Note that for Variable-Size Partitioning and Segmentation there is a further policy parameter that can be either 1 or 2 representing the two approaches which are First-Fit and Best-Fit.

As an example, to define the memory size to be 2000 Kbytes and the MMP to be Variable-Size Partitioning with Best-Fit, the interaction would be the following (simulator prompts in boldface, user response in underlined italics):

Memory size> *2000*
Memory management policy (1- VSP, 2- PAG, 3- SEG)> *1*
Fit algorithm (1- first-fit, 2- best-fit)> *2*
If the policy is PAG, prompt for a page/frame size (rather than a fit algorithm).

Memory Management Policy (MMP) Details

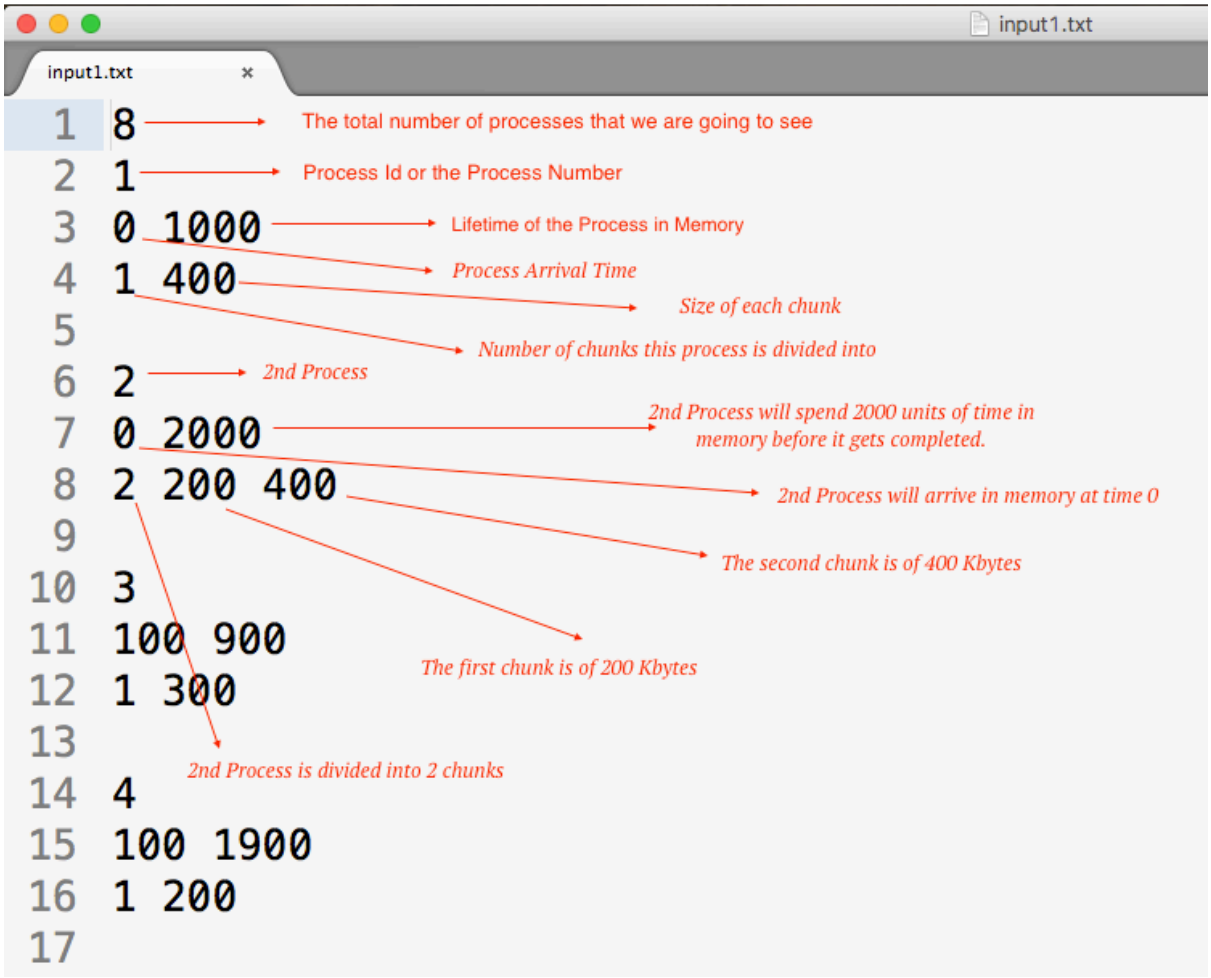
A process's memory requirements will be specified as a sequence of different sized chunks (see the format of the input file below for details.) A process can only be loaded into the memory if there is sufficient free space to load all its chunks (more details in the next section).

VSP: Variable-Size Partitioning (Contiguous Allocation). In this case, you will sum individual chunks and load all the chunks as one process into memory. All these chunks must be placed contiguously in the memory as one big chunk.

PAG: Paging. In this case, you will sum individual chunks of the process. Then divide total address space into *pages* of the specified page/frame size. These individual *pages* (of the same size) may or may not be loaded contiguously into memory. However, all these *pages* belonging to a single process should be loaded simultaneously into memory.

SEG: Segmentation. In this case, the individual chunks of a process will be considered as different *segments* of a process. These individual *segments* belonging to a single process (of possibly different sizes) may or may not be loaded contiguously into memory. However, all these *segments* belonging to a single process should be loaded simultaneously into memory.

Format of the input workload File



Your program will also prompt the user for the name of a workload file. This file will first have an integer value N that tells you how many processes are defined in the file. The characteristics of each individual process will include a unique ***Process Id*** on the first line, the time it is submitted to the system (***Arrival Time***) and its lifetime after it is admitted to the main memory (***Lifetime in Memory***) on the second line, and finally, its memory requirements (***Address Space***). These process specifications will be separated by a *single blank line*. Processes in the workload file will be listed in arrival order. If two processes have the same arrival time, the process listed first in the workload file goes on the input queue first. For details please see the examples in the "Examples" folder

The ***Address Space*** line is a sequence of one or more integers separated by a single blank. The first integer gives number of chunks of memory on the line. This sequence denotes the total size of the Address Space of the process.

Note that since the memory is limited, there is no guarantee that a process will be admitted to the memory as soon as it arrives; thus it may have to wait until the system can accommodate its memory requirements. The ***Lifetime in Memory*** information for a given process defines how long the process will run ONCE IT HAS BEEN GIVEN SPACE IN MAIN MEMORY. If a process is submitted to the system at time = 100 with ***Lifetime in Memory*** = 2000, but isn't admitted to the memory until time = 1500, then it will complete at time = 1500 + ***Lifetime in Memory*** = 3500. The memory space for a process will be freed by the memory manager when it completes.

Your simulator must generate output to the output file (and to the screen as well, if possible) that explicitly shows the trace of important events that modify the memory contents and input queue, including:

- Process arrival
- Process admission to the main memory
- Process completion

- Basically, whenever there is a change in the ***Memory Map*** or the ***Input Queue***, your program must also display the updated contents of the memory map and/or input queue. The simulation terminates when all the processes have successfully been executed or time reaches 100,000 units.

Implementation

For this simulation, you can keep an integer (or long) variable representing your "virtual clock". Then you can increment its value until all the processes have completed, making appropriate memory management decisions along the way as processes arrive and complete. Each arriving process will be first placed in the ***Input Queue***. Processes in ***Input Queue*** will be ordered according to their arrival times (FCFS ordering). Whenever a process completes or a new process arrives, the memory manager (MM) must be invoked. In case of a completion, MM will first adjust the ***Memory Map*** to reflect the fact that the memory region(s) previously allocated to the process is/are now free. After that, (and also when a new process arrives) MM will check to see if it can move the process at the head of the input queue to the memory. If so, it will make the allocation as specified by MMP and it will update the ***Input Queue***. Then it will check whether the new process at the head of the updated ***Input Queue*** (new head) can also be moved to the main memory, and so on. Even if the current commitments in memory do not allow MM to admit the process at position X of ***Input Queue***, MM should try to load other processes at positions X + 1, X + 2, X + 3, ... and in that order.

Throughout the implementation, you will assume that the entire ***Address Space*** of a process must be brought to the main memory for execution. Advanced virtual memory techniques, including demand paging, demand segmentation and page replacement issues are beyond the scope of this assignment. Consequently, your program can simply ignore any process that it encounters whose total address space is larger than the memory size. Note that a large process may have to wait in ***Input Queue*** until sufficient space is available in memory; that is a normal scenario.

When the MM is allocating a free memory region (hole) to one *process*(for VSP)/*page*(for PAG)/*segment*(for SEG), it may have to choose between the lower end and the upper end of the region. In this case, always use the lower end. As an example, if there is a hole between

the memory addresses 200 K and 500 K, and MM is moving a segment of length 100 K to that hole, the segment must be put between the addresses 200 K and 300 K, effectively yielding a new, smaller hole between the memory addresses 300 K and 500 K.

The ***turnaround*** time for each process will be computed as the difference between its completion time and its **Arrival Time**.

NOTE: Even though at first it may look like you have to implement five different allocation algorithms, in reality the algorithms are very similar to each other. As an example, implementing First-Fit after you complete Best-Fit should be straightforward. Carefully thinking about the relationships among the schemes can save significant coding time!

More Details

The difference between First-Fit and Best-Fit approach is that while placing a chunk in a memory hole First-Fit approach would fit the chunk as soon as it finds a memory hole that can accommodate this chunk whereas in the case of Best-Fit the chunk is placed in the *Smallest Possible* memory hole that can accommodate this chunk. For example, we have a hole between memory addresses 100 K and 300 K and another hole from memory addresses 500 K and 600 K and a chunk of 100 Kbytes is to be brought in the memory. The First-Fit approach would place the chunk between the addresses 100 K and 200 K, effectively yielding a smaller hole between the memory addresses 200 K and 300 K. The Best-Fit approach, on the other hand, would place this chunk between the addresses 500 K and 600 K, effectively filling up a memory hole.

You can assume that the input file will contain information about at most 20 (twenty) processes. The maximum lengths of simulation time and memory size that can be indicated in the input file are 100,000 and 30,000, respectively. For SEG, assume that a process can have at most 10 segments.

As already discussed the simulation terminates when all the processes have successfully been executed or the time reaches 100,000 units. For the time interval (virtual clock), you can use 1 micro second as unit time.

Since the focus of this assignment is on memory management, do not be concerned about the details of CPU scheduling or I/O device management. Just assume that the process will be completed after staying in memory for **Lifetime in Memory** time units.

Feel free to adopt any convenient data structure to represent your ***Memory Map, Input Queue, Process*** etc.

Good modular code along with proper commenting carries bonus marks. ☺

*Bonus marks may only be used if the obtained marks are less than the total marks.

Marks Breakdown

1. Variable-Size Partitioning ***(2+2 points)***
    a. First-Fit
    b. Best-Fit
2. Paging ***(2 points)***
3. Segmentation ***(2+2 points)***
    a. First-Fit
    b. Best-Fit