California State Polytechnic University, Pomona
Computer Science Department
CS4310: Operating Systems
Spring 2019
Instructor: Prof. Tony Diaz

Program 3 - Due May 10

(Total: 100 pts)

The purpose of this project is to have you write a simulation 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 decisions of Memory Manager (MM). The simulator will generate an output file with the trace of important "events," 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. Below we provide the specifications and details about the project. Please read all the parts of this hand-out carefully before starting to work on your project.

Basic functions of the simulator

Your simulator will prompt the user for the size of the memory and then the memory management policy (and associated parameters) information. The Memory Size (an integer) denotes the capacity of the main memory in your simulation (it can be interpreted as a multiple of Kbytes). The Memory Management Policy can be:

1. VSP: Variable-Size Partitioning (Contiguous Allocation). In this case, there will also be a policy parameter can be either 1, 2 or 3, encoding the algorithm used for choosing among eligible "memory holes" when moving a process to the memory (1: First-Fit, 2: Best-Fit, 3: Worst-Fit).
2. PAG: Paging. In this case, the policy parameter will denote the page size (consequently, also the frame size). You can assume that the Memory Size will always be a multiple of the page (hence, frame) size.
3. SEG: Segmentation. In this case, the policy parameter can be either 1, 2 or 3, denoting the algorithm used for choosing among eligible "memory holes" when moving individual segments to the memory (1: First-Fit, 2: Best-Fit, 3: Worst-Fit).

As an example, to define the memory size to be 2000 kbytes and the memory policy to be Variable-size partitioning with Worst-fit, the interaction would be the following (simulator prompts in **boldface**, user responses in *italics*):

> **Memory size:** *2000*
> **Memory management policy (1- VSP, 2- PAG, 3- SEG):** *1*
> **Fit algorithm (1- first-fit, 2- best-fit, 3- worst-fit):** *3*

If the policy is PAG, prompt for a page/frame size (rather than a fit algorithm).
At this point, you will 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 file will be listed in arrival order. If two processes have the same arrival time, the process listed first in the file goes on the input queue first. See the online examples for details.

The Address Space line is a sequence of one or more integers separated by a single blank. The first integer gives the number of 'pieces' of memory on the line. This sequence denotes the total size of the address space of the process. For VSP and PAG, you simply sum these integers to get the overall space requirement. For SEG, the list of numbers is the size of the individual segments of the process. As an example, if Memory Management Policy is PAG and Address Space is "3 500 300 200", then the process has a total address space of 1000 K for the purposes of your simulation. If Memory Management Policy is SEG then the process has three segments: the first segment is 500 K, while the second and third segments are 300 K and 200 K, respectively.

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 screen) 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

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. Please make sure to visit Blackboard for sample input files and the corresponding output files that illustrate the expected format. When the simulation terminates (when there are no more processes in the input queue or time reaches 100,000).

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 put to Input Queue. Processes in Input Queue will be ordered according to their arrival times (FIFO 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 Memory Management Policy and it will update the input queue. Then it will check whether the head of the updated input queue (new head) can be also 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, etc..., and in that order.

- Throughout the project, you will assume that the entire address space of a process must be brought to the main memory for execution. For PAG and SEG, MM should not bring the pages/segments of a process partially: all of its pages and segments must be brought at once. Advanced virtual memory techniques, including demand paging, demand segmentation and page replacement issues are beyond the scope of this project. Consequently, your program can simply ignore any process that it encounters a process whose total address space is larger than 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/page/segment, 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.
- Since the focus of this project 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; but like any other design decision, explain it in your write-up.

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

NOTE: For full credit, you must follow the format of the input file as given above; during the grading process we will use our own input files in that format to test your program. 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.

You MAY work in groups of 2 on this assignment.
Simply let me know who your partner is, before the due date.

What to turn in:

- A project report analyzing your code and explaining your solution
  (submit a .doc or .pdf file to Blackboard, each member needs to write their own report)

- A softcopy of all of your source code
  (submit it to Blackboard, only one member of your group needs to submit your code)