CPSC 351 Operating System Concepts Project #3

Due: 12:55 P.M. on April 30, 2014

1. Overview

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.

2. Basic functions of the simulator

Input file: *in1*

- *Out1* for file *in1* using Paging with memory size = 2000 and page size = 100
- *Out2* for file *in1* using Paging with memory size = 2000 and page size = 200
- *Out3* for file *in1* using Paging with memory size = 2000 and page size = 400

Your simulator will prompt the user for the size of the memory 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 is:

> 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.

As an example, to define the memory size to be 2000 kbytes, the interaction would be the following (simulator prompts in **boldface**, user responses in *italics*):

> **Memory size>** *2000*
>
> **Page Size (1: 100, 2: 200, 3: 400)>** *3*

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.

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. You simply sum these integers to get the overall space requirement.

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 it is not 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

Please make sure to refer to the input file 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).

3. 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 (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

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, …, 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. MM should not bring the pages of a process partially: all of its pages must be brought at once. Advanced virtual memory techniques, including demand paging 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, it may have to choose between the lower end and the upper end of the region. In this case, always use the lower end.
- 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; explain it in your write-up.

4. Operational Details

In this project, you can work alone or with two or three students together. No project team can have more than four members. All members of a team will get the same project grade.

During the grading process, your program will be compiled and run with various test files with the same format, so it is your responsibility to make sure that your program works correctly.

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.

5. Submission Guideline:

You have to submit your package on TITANium before 12:55 P.M. on April 30, 2014 (one copy per person/team).

- **This assignment MUST be completed using C or C++ on Linux.**
- You may work in groups of 3.
- Please hand in your source code electronically (do not submit .o or executable code)
- You must make sure that the code compiles and runs correctly.
- Write a README file (text file, do not submit a .doc file) which contains
  - Your Section#, Name and Email Address.
  - The programming language you used (i.e. C or C++).
  - A write-up explaining the high-level design of your program.
  - How to execute your program.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as p3-[userid] for assignment 3, e.g. p3-ytian).
- Tar the contents of this directory using the following command. tar cvf [directory name].tar

  [directory name] E.g. tar -cvf p3-ytian.tar p3-ytian/
- Use TITANIUM to upload the tared file you created above.

Do not forget to include any information needed to compile and run your program. If we are not able to compile or run your program, it cannot be graded; so include all the information. As before, if your submission involves more than one file, you must tar and compress the files before submitting. However, please do not use Windows-based compression programs such as WinZip.

6. Grading Guideline:

- Program compiles: 10'
- Correctness: 75'
- README file: 15'
- **Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.**

7. Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in

programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.

8. Final Notes

- Do not postpone the project to very last days.
- You are encouraged to work with a partner, but choose your partner wisely. Efficient cooperation with your partner can provide a better project with less effort only if you are well-coordinated. If you have a partner, go over the project specification with him/her as soon as possible and decide on the individual responsibilities.
- Carefully review this specification, your lecture notes and the textbook before starting to code to make sure that you thoroughly understand the system execution mechanism and the rules of the specific memory management policies. Otherwise, you may have an incorrect implementation.
- You can direct your programming and system questions to our online discussion forum or me(ytian@fullerton.edu)