

1. Problem Statement

- 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.

- 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

1CPSC 351
Spring 2019

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).

- 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.

2CPSC 351

Spring 2019

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.

- Operational Details

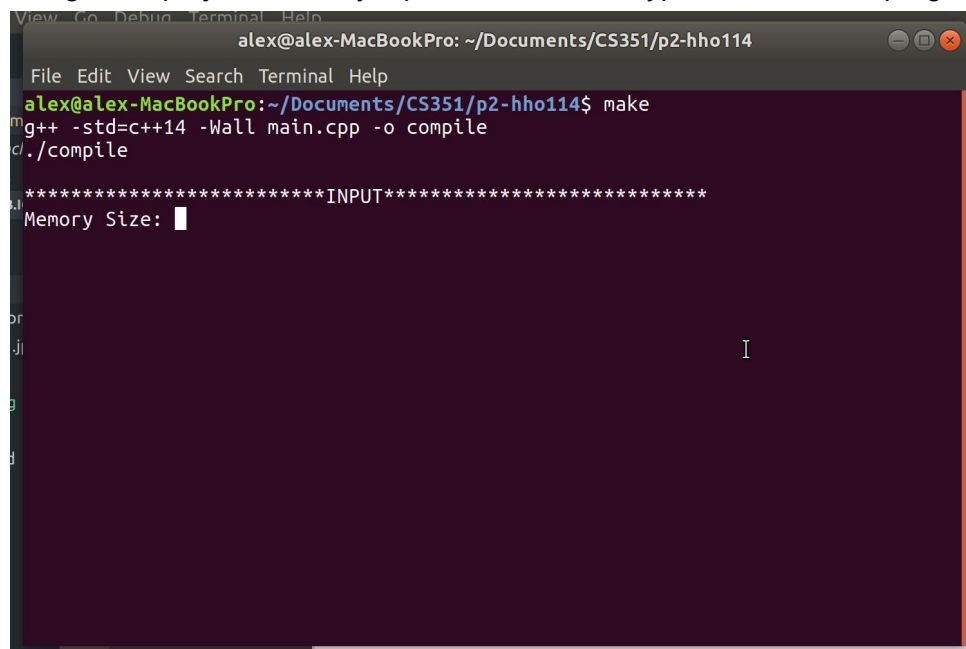
In this project, you can work alone or with two or three students together. No project team can have more than four members. All members of a team will get the same project grade.

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.

2. How to use your program

Navigate to project directory, open terminal, and type "make" to run program



```
alex@alex-MacBookPro: ~/Documents/CS351/p2-hho114
File Edit View Search Terminal Help
alex@alex-MacBookPro:~/Documents/CS351/p2-hho114$ make
g++ -std=c++14 -Wall main.cpp -o compile
./compile

*****INPUT*****
Memory Size: 1
```

Input the Memory size and page size:

```
alex@alex-MacBookPro: ~/Documents/CS351/p2-hho114
File Edit View Search Terminal Help
alex@alex-MacBookPro:~/Documents/CS351/p2-hho114$ make
g++ -std=c++14 -Wall main.cpp -o compile
./compile
*****INPUT*****
Memory Size: 2000
Page size (1: 100, 2: 200, 3: 400): 1
Enter name input file: in1.txt
```

Check the process of Memory Management and an average turn around time:

```
alex@alex-MacBookPro: ~/Documents/CS351/p2-hho114
File Edit View Search Terminal Help
./compile
*****INPUT*****
Memory Size: 2000
Page size (1: 100, 2: 200, 3: 400): 1
Enter name input file: in1.txt

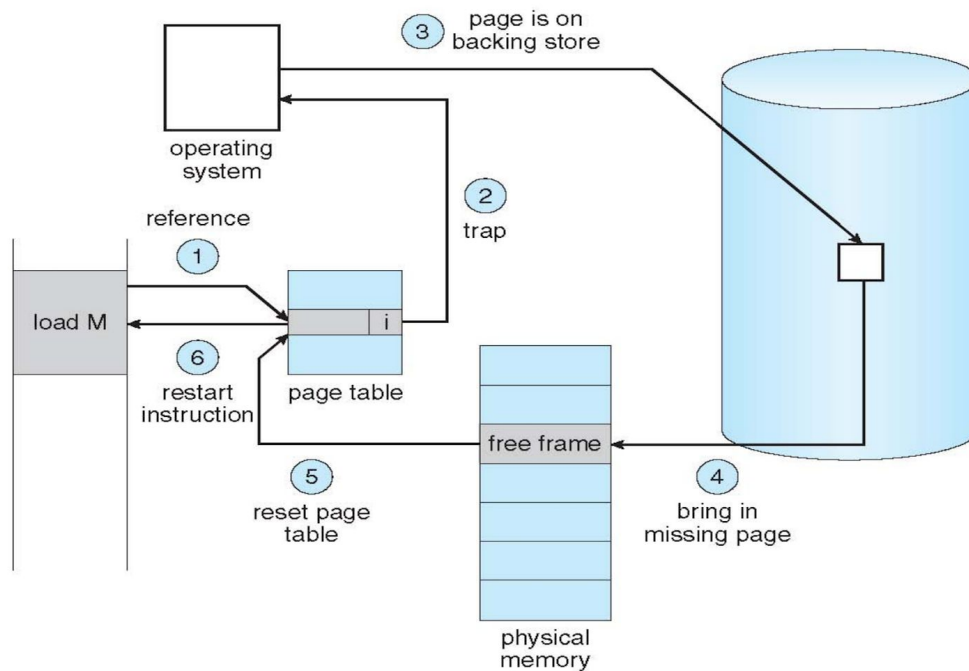
*****OUTPUT*****
t = 0: Process 1 arrives
    Input Queue:[1 ]
    Process 2 arrives
    Input Queue:[1 2 ]
    MM moves Process 1 to memory
    Input Queue:[2 ]
    Memory Map:
        0 - 99: Process 1, Page 1
        100 - 199: Process 1, Page 2
        200 - 299: Process 1, Page 3
        300 - 399: Process 1, Page 4
        400 - 1999: Free frame(s)
    MM moves Process 2 to memory
    Input Queue:[]
    Memory Map:
```

```
alex@alex-MacBookPro: ~/Documents/CS351/p2-hho114
File Edit View Search Terminal Help

1500 - 1599: Process 7, Page 5
1600 - 1699: Process 7, Page 6
1700 - 1799: Process 7, Page 7
1800 - 1899: Process 7, Page 8
1900 - 1999: Process 8, Page 1
Process 7 completes
Memory Map:
  0 - 99: Process 6, Page 1
 100 - 199: Process 6, Page 2
 200 - 299: Process 6, Page 3
 300 - 1899: Free frame(s)
 1900 - 1999: Process 8, Page 1
t = 2100: Process 8 completes
Memory Map:
  0 - 99: Process 6, Page 1
 100 - 199: Process 6, Page 2
 200 - 299: Process 6, Page 3
 300 - 1999: Free frame(s)
t = 3000: Process 6 completes
Memory Map:
  0 - 1999: Free frame(s)

Average Turnaround Time: 1175.00
alex@alex-MacBookPro:~/Documents/CS351/p2-hho114$
```

3. Design of your program



Two classes: Process, Frame

- Process include id, lifetime, time arrive, time into memory, is process active.
- Frame includes id, page number frame number, frame start time and end, is frame assigned

Three lists: Process list, Queue list, Memory Frame list

- Process list is a list which contains input processes from in1.txt
- Queue list is a list which contains arrive and waiting processes
- Memory frame list is a list which contains active process in frames

Implement the FIFO queue and frame list for memory management.