Project - 2
CSC 4103 – Operating Systems
Due: December 1st @11:59pm

Your task in this project will be to implement the **Memory Management** component of an Operating System using OSP. Please read all instructions carefully before starting the project.

1. Preparation

Before beginning your work, read the corresponding sections from the OSP book (Chapter 1.6 from the OSP 1.0 or Chapter 5 from OSP 2.0) carefully. You may also need to revisit the related materials in your text book (Silberschatz) to get the conceptual background necessary for the project. For this purpose you should read especially Chapter 9.4 from Silberschatz for different page replacement policies.

2. Programming Task - Implement Memory Page Replacement Policies in OSP:

The objective of this task is to implement the Memory module with two different page replacement policies in OSP, and to discuss simulation results on various cases. In the Memory module you have to **implement and compare FIFO and LRU** (Section 9.4.2 and 9.4.4) page replacement policies.

A brief discussion (around 1 page) on statistics of different situations is needed in this task. The discussion should be attached to your source program. You should run your program on various simulation parameters and explain different statistics among the results. You should also try different degree of prepaging and discuss the difference in the results from both policies.

The statistics of interest are:

- Number of pages swapped in/out (the most important parameter for this task)
- Turnaround time
- Waiting time
- Throughput

You should design your program to ask for a directive from the terminal as to whether to run under FIFO or LRU algorithms.

Files needed for this task can be found in directory:

- **C version:** /classes/cs4103/cs4103_kos/project-2/prj2_C
- **Java version:** /classes/cs4103/cs4103_kos/project-2/prj2_Java

Create a directory called "*prog2*" in your home directory, and copy all the files in the above directory into your new directory.

3. Implementation

For C programmers:

You are to complete the following functions in the file *memory.c:*

1. memory init ( ): Called once to allow for the initialization of internal data structures; the body of this routine can be left empty, if no initialization is needed.

2. prepage (pcb): Prepages the process specified in the argument.

3. int start cost (pcb): Calculates the cost of prepaging in terms of the number of pages to be swapped in or out, times the cost of page transfer.

4. deallocate (pcb): Called by the terminate and kill handlers to deallocate pages occupied by the terminated process represented by pcb.

5. get page (pcb, page id): Implements page allocation and replacement; if replacement, decides which frame to replace; brings the desired page into main memory and saves the old frame contents on the drum, if needed; calls siodrum( ) of SIMCORE to do the actual page transfer; sets valid/invalid and other ags.

6. lock page (iorb) & unlock page (iorb): Called to lock/unlock the specified page; memory locking is used to protect pages involved in active I/O operations from being swapped out; locking/unlocking is done by incrementing/decrementing the lock count field of the corresponding frames.

7. refer (logic addr, action): Called by SIMCORE to simulate memory access by cpu; logic addr is a logical address within the virtual memory of the current process; it is converted to a physical address using the page table pointed to by PTBR; the action parameter indicates whether this is a store operation, which changes the memory contents, or a load operation, which does not; this information is needed for "dirty bit" optimization.

For Java Programmers:

You fill need to implement the following methods in 4 different files:

1. Inside PageFaultHandler.java:
   a. **public static int do_handlePageFault(….):** The core method responsible for handling a fault. If a swap in or swap out operation is required, the page fault handler must request the operation.

2. Inside PageTable.java:
   a. **public PageTable(….):** Page table constructor.
   b. **public void do_deallocateMemory():** Frees up main memory occupied by

the task. Then unreserves the freed pages, if necessary.

3. Inside PageTableEntry.java:
   a. **public PageTableEntry(….):** Page table entry constructor.
   b. **public int do_lock(….):** This method increases the lock count on the page by one.
   c. **public void do_unlock():**This method decreases the lock count on the page by one.

4. Inside FrameTableEntry.java:
   a. **public FrameTableEntry(….):** Frame table entry constructor.

Remark: It should be pointed out that the OSP functions referred above simulate work that would be done by hardware in a real system. In this task, the prepaging policy is to bring pages starting from the first page of the process to the minimum number between total number of pages in the process and degree of prepaging.

4. Running your Simulator

After finishing the programming task, type UNIX command "make" in your directory. Your module will automatically compile and link into the OSP system. An executable file named **OSP** *(or a jar file if you are using java)* will be created in your current directory. To run your program, just type "OSP" *(or using your java virtual machine)*. The system will prompt you to input simulation parameters and will run with them. Use command "OSP.demo" *(or Demo.jar)* you can get the running result of a fine tuned program provided by OSP. You may compare the result with that of yours and adjust your program accordingly.

5. How to Submit

After you finish each task of your project, you can submit it by typing *"p_copy 2"*. Your program will be copied automatically to the instructor's account. After submission, you still can modify your program and hand in again. A new submission will overwrite the previous one.

6. Requirements

You are also expected to submit the program with clear and sufficient comments and explanations. Presentation will be counted as a part of grading.

All the programs should be submitted by **December 1st, Tuesday @11:59pm.**