Operating Systems **CSI3131** Lab 4

Page replacement algorithms

Objective

Use a simulation to study the performance of page replacement algorithms studied in class.

Description (Take the time to read the entire document before starting the lab) A program for simulating a memory management system is provided to you. This program includes the following files (available from the file Lab4.zip):

a) *MemManage.java*: This file contains several classes to simulate the memory management system:
   a. The MemManage class: This simulation class creates four process objects (see the Process class) and simulates the execution of these processes using FIFO scheduling. Each process is executed for a random number of memory references (varies from process to process). The four processes have the following characteristics:
      i. PID = 100, number of virtual pages = 30
      ii. PID = 101, number of virtual pages = 24
      iii. PID = 102, number of virtual pages = 36
      iv. PID = 103, number of virtual pages = 32

      The simulation class counts the number of page faults during a simulation and at the end of the simulation in its output gives the number of page faults per 1000 references to memory, which will allow you to compare the performance of each algorithm from page replacement. The constructor of this class includes a parameter that defines which page replacement algorithm to use.
   b. The Process class: This class is used for the creation of process objects. More details are provided below because you have to manipulate several of its data structures (for example its array of pages).
   c. The Kernel class: A Kernel object is created when creating an instance of MemManage object. Among other things, it captures the specifications of the kernel. For example, an array of integers defines the number of physical memory frames. There are 32 physical frames available. You must NOT manipulate the Kernel object.
   d. The Seeds class: This class is used to create several seed values of the various random number generators used in the simulation program.

b) *MemManageExp.java*: This program contains a main method which runs a simulation for each page replacement algorithm. The simulations are long enough to give results and thus compare the performance of the four algorithms. The program's output should give results that vary between 45 and 60 page faults per 1000 memory references.

c) *FifoExp.java*, *ClockExp.java*, *LruExp.java*, *CountExp.java*: Each of these Java programs contains a main method for executing a simulation object (MemManage) for each of the page replacement algorithms. The simulation is short-lived and does not necessarily produce valid results. The programs allow you to debug the code separately for each page replacement algorithm (if you are generating output, short simulations will keep the length of the output manageable).
d) *KernelFunctions.java*: This Java program contains the KernelFunctions class which contains the methods which do the replacement of pages. The MemManage class calls two methods: memAccessDone whenever memory is accessed and pageReplacement whenever a page fault occurs. You must complete the methods in this class. You will also find in this Java file the class PgTblEntry which specifies the format of the entries of the page table (and used in the creation of the page tables). This class will be detailed later.
e) *colt.jar*: This file contains the various classes for creating the random number generators used in the simulation program. Make sure to put this file in the classpath.
f) *EvSchedSimul.jar*: The file contains several classes to perform a simulation. For example, you will notice that the MemManage class augments the EvSched class, an event scheduling simulation class. Make sure to put this file in the classpath.

The laboratory:

Part a: During the session of the first week of the lab, take the time to study the code provided and understand the organization of the Process class. Several of the concepts will be presented in class following the first lab session. Try to review lecture notes and text to understand the terms working set, page fault, page replacement, etc. Compile and run the provided code to test the FIFO page replacement algorithm. If you have time, try to understand one of the LRU or Clock algorithms and implement. Otherwise, wait until the second week to implement the algorithms (Part B and C).

Part b: Your first task is to complete the KernelFunctions class to perform and compare the three FIFO, CLOCK and LRU page replacement algorithms. Your results should show that LRU is the best performing (the smallest number of page faults per 1000) followed by CLOCK and finally that FIFO is the least efficient (the greatest number of page faults per 1000).

a) Complete the three methods, pageReplAlgorithmFIFO, pageReplAlgorithmCLOCK, and pageReplAlgorithmLRU. The doneMemAccess method will allow you to perform actions at each memory reference if necessary (for example to assign fields in page table entries).
b) DO NOT CHANGE the pageReplacement, addFrame, and pageReplAlgorithm methods. The first two methods assign physical memory frames to the process (defined by numAllocatedFrames in the Process class). Only after the maximum number of frames is assigned to the process, the page replacement methods will be called. The pageReplAlgorithm method will call the method corresponding to the algorithm chosen when the MemManage object was created.

c) The logProcessState method you are provided to debug your code. It can be used to print the process report from your page replacement code.

Part c: Here is the part where you explore your creativity. Your goal is to design, describe and realize a page replacement algorithm which should perform better than the CLOCK algorithm, but should be easier to perform than the LRU algorithm. You can add counters or other variables to the entries of page tables (ie, you can change the PgTblEntry class to support your implementation). Try to simplify the complexity of your algorithm to approach the simplicity of the CLOCK algorithm and move away from the complexity of the LRU algorithm. Be inspired by reading sections 9.4.5 and 9.4.6 of the Silberchatz text; you are encouraged to try different approaches and choose the best one. In addition to coding your algorithm, you must provide good documentation of your algorithm;

On the next page, you will find more information about the Process class and the data structures you are going to use.

The Process class:

The following data structures are used to perform the various page replacement algorithms.

a) `int pid;` The process identifier.

b) `public int numPages;` Gives the number of virtual pages used by the process.

c) `public PgTblEntry [] pageTable;`This array of PgTblEntry objects (see below) is the page array. Note that data necessary to support the page replacement (bit used, valid bit, time stamp, etc.) is there.

d) `public int [] workingSet;`This data structure lists the pages in the process's working set. This list is used to create page references during simulation.

e) `int numAllocatedFrames;` Defines the total number of frames that can be assigned to the process.

f) `int [] allocatedFrames;` This integer table gives the list of numbers of physical frames assigned to the process.

g) `int framePtr;`A pointer that can be used as an index in the allocatedFrames array. Can be used in page replacement methods. Its value is 0 after all frames are assigned to the process.

h) Other data structures are defined in the Process class to simulate memory access, and in particular to simulate the grouping of references. They are not presented here. If you're curious, check out the MemManage.java file.

The following class gives the format of the entries in the page table. It contains the fields necessary to perform the FIFO, LRU, and CLOCK page replacement algorithms. You can make changes to the class to achieve your Part B algorithm.

```
class PgTblEntry
{
 int frameNum;       // frame number
boolean valid; // valid bit boolean
used; // bit used double tmStamp; //
Time stamp }
```