Operating Systems **CSI3131** ***Lab 4*** *Winter 2018*

Page Replacement Algorithms

Objective

To use a simulation for evaluating various page replacement algorithms studied in class.

Description (Please read the complete lab document before starting.)

You are being provided with a simulation program of a memory management system that consists of the following files (available from the Lab4.zip file):

a) *MemManage.java*: This file contains a number of classes to simulate memory management:
   a. MemManage Class: The simulation class that creates four process objects (see the Process class) and simulates the execution of the four processes using FIFO scheduling. Each process is executed for a random number of memory accesses (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 monitors the number of page faults and at the end outputs the number of page faults per 1000 memory references to allow you to compare the performance of each page replacement algorithm. The constructor of this class contains a parameter that defines the page replacement algorithm.
   b. Process Class: This class is used to create process objects. This class is defined in more detail later since you shall be manipulating many of its data structures (for example its page table).
   c. Kernel Class: One kernel object, "kernel" is instantiated when a MemManage object is instantiated. It provides the kernel specifications. For example an integer array defines the available physical frames. There are 32 physical frames available. You will NOT be manipulating the kernel object.
   d. Seeds Class: This is used to create a number of seeds used to initialise the various random number generators used in the simulation program.

b) *MemManageExp.java*: This Java program provides a *main* method to run a simulation for each of the three page replacement algorithms. The simulations are sufficiently long to produce results and can be used to compare the performance of the three page replacement algorithms. Note that the resulting output should be on the order of 50 to 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 for each of the page replacement algorithms. They run the simulation for a very short period and do not produce valid results. They are provided for your convenience to allow you to debug your code (if you debug using logging, then the short runs produces manageable output) separately for each page replacement algorithm.
d) *KernelFunctions.java*: This Java program contains the class *KernelFunctions* which provides the necessary methods for page replacement. The *MemManage* class invokes two methods: *memAccessDone* each time a memory access is completed and *pageReplacement* each time a page Fault occurs. You will be completing the methods in this class. Also found in this Java file is the class *PgTblEntry* specifying the format of the page table entries (and used to create the page table). More on this class later.
e) *colt.jar*: This file contains the various classes for creating various random number generators used in the simulation program. Be sure to include this file in "classpath".
f) *abcmod.jar*: This file contains the various classes that provide simulation functionality. For example, notice that *MemManage* is extends the *EvSched* class, an event scheduling simulation class. Be sure to include this file in "classpath" (or imported into your favourite Java development tool).

The Lab:

Part A: During the first week, take the time to study the code provided and to understand how the Process class is organized. Many of the concepts shall be presented during a lecture following the first lab session. Try to review the course notes and text book to understand terms such as working sets, page faults, page replacement algorithms, etc. Compile and run the code to test out the FIFO page replacement algorithm. If you have time try to understand one of the algorithms LRU or Clock and implement. Otherwise wait until the second week of the lab to complete the algorithms (Parts B and C).

Part B: Your next task is to complete the *KernelFunctions* class to implement and compare the three page replacement algorithms FIFO, CLOCK and LRU. Your results should show that LRU has the best performance (least number of page faults per 1000) followed by CLOCK, and finally that FIFO has the poorest performance (most page faults per 1000).

a) Complete the three methods, *pageReplAlgorithmFIFO*, *pageReplAlgorithmCLOCK*, and *pageReplAlgorithmLRU*. The method *doneMemAccess* has also been provided if you need to perform actions each time a memory access is completed (e.g. for updating the fields of the related page table entry).
b) DO NOT change the methods *pageReplacement*, *addFrame*, and *pageReplAlgorithm*. The first two methods will allocate physical memory frames to the process (defined by array *allocatedFrames*) until the all frames have been allocated to the process (defined by *numAllocatedFrames* in the Process class). Only after all frames have been allocated to the process, will the page replacement algorithm methods be invoked. The method *pageReplAlgorithm* invokes that method corresponding to the algorithm selected when *MemManage* was instantiated.
c) The method *logProcessState* has been provided for debugging purposes. It can be used to print the state of a process within your page replacement code.

Part C (if you have time): This is the part where you can unleash your creativity. Your goal is to design, describe and implement a page replacement algorithm that should hopefully improve upon the CLOCK algorithm, while still being cheaper to implement then the LRU algorithm. You can use additional counters/variables in the page table entries (i.e. you may make changes in the PgTblEntry class to support your implementation), but try to keep the cost of your algorithm closer to the cost of CLOCK then the cost of LRU. Take your inspiration by reading sections 9.4.5 and 9.4.6 from the textbook, trying several approaches and selecting the best is strongly encouraged.

Additional details on the Process class, PgTblEntry and structures you will be manipulating are provided below:

Process Class:

The following data structures provide the means to implement the various page replacement algorithms:

a) `int pid;` The process pid number used to identify the process.

b) `public int numPages;` Defines the number of virtual pages defined for the process.

c) `public PgTblEntry [] pageTable;` This is the process page table. It is an array of PgTblEntry objects (see below). Note that all necessary data for supporting page replacement are provided in the page table (used bit, time stamp, etc.).

d) `public int [] workingSet;` This array provides the list of the process pages in its current working set. This list is used to create the page references during simulation.

e) `int numAllocatedFrames;` This integer defines the total number of frames that can be allocated to the process.

f) `int [] allocatedFrames;` An integer array that provides a list of frame numbers, that is the number of the frames allocated to the process.

g) `int framePtr;` A pointer, that can be used to index into the *allocatedFrames* array. Provided to support the page replacement. Set to 0 after all frames have been allocated to the process.

h) A number of other data structures are defined in the Process class used to simulate memory access, and in particular locality of reference. These are not presented here. If you are curious, consult the *MemManage.java* file.

The following class specifying the format of the page table entries is provided to you, it should be sufficient for using with FIFO, LRU and CLOCK page replacement algorithms, you may add additional fields to support your algorithm for Part B.

```
class PgTblEntry
{
   int frameNum;    // Frame number
   boolean valid;   // Valid bit
   boolean used;    // Used bit
   double tmStamp;  // Time Stamp
}
```