CS 346 – Lab 9 – Page Replacement Simulation

Today we will simulate and compare some page replacement strategies. A true simulation would require us to consider millions of frames and billions of memory references, but we don't have that kind of time to work on this project. We will do everything in miniature. For example, we will assume we have a small number (e.g. 3 or 4) frames in RAM where pages can be swapped in and out.

Create a new directory lab09. From my directory /home/www/chealy/cs346/lab09, please copy the files Sim.java, input1.txt, Test2.java, Test3.java and Test4.java to your directory. You will make changes to the Java files.

This lab has 2 parts. First, you will modify Sim.java to implement the LRU page replacement strategy. Second, you will write short Java programs to generate input for Sim.java.

Part One

The page replacement algorithms implemented in Sim.java are:

- Clairvoyant (already done for you)
- FIFO (already done for you)
- Second chance FIFO (already done for you)
- LRU

Modify Sim.java to complete the implementation of the LRU page replacement policy. Note that you will also need to count the total number of page faults caused with LRU, so that we can compare its performance with the other policies.

Your Java program will require 2 kinds of input:

- The number of available frames, which will be specified at the command line
- The reference string (list of page number requests). This will be read in from standard input. (Standard input may be redirected from a file by the shell.)

In other words, when you run your program, you need to enter a command such as:

```
java Sim 4 < input1.txt
```

The essential output from your program will be 4 numbers: the number of page faults resulting from each page replacement algorithm.

The text file input1.txt has been provided for you for testing Sim.java. This is one of the examples we went over in class.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

I recommend that you initially test your program with small amounts of input like this, and print diagnostic output whenever your program detects a page fault: show which page gets evicted.

Part Two

Let's generate more realistic input data for our page replacement simulator. You have already been using input1.txt, so let's create three more input files, input2.txt, input3.txt, and input4.txt. Each of these input files will be created by its own little Java program, Test2.java, Test3.java, and Test4.java, respectively.

Test Run #2 (Test2.java)

The purpose of Test2.java is to print a reference string as output. Let's say we want to simulate the memory references that result from accessing a 2-dimensional array in various ways. For example, let's consider a 7x7 array of integers. Each integer occupies 4 bytes, and

let's say our **page size is 32 bytes**. (Of course in reality all these numbers would be much larger.) We want to access the individual array elements in row-major order, column-major order, and triangularly, like this:

```
for i = 0 to 6
  for j = 0 to 6
    access a[i, j]
for j = 0 to 6
  for i = 0 to 6
    access a[i, j]
for i = 0 to 6
  for j = 0 to i
    access a[i, j]
for j = 0 to 6
  for i = 0 to j
    access a[i, j]
```

You may use any base address you wish, such as 0. Let's assume that the entries of the array are stored in row-major order, so that the array element addresses are as follows:

```
Address = base address of array + 4 * (7*i + j)
Page number = Address / 32
```

When your run this program, redirect its output to a file called input2.txt so that it can be later used as input to your page replacement simulator. This technique is also useful for test runs 3 and 4.

Test Run #3 (Test3.java)
Same as test run #2, but we will approximately double the dimensions, and only do triangular access to the array. The array will be 15x15 of integer, and the **page size will be 64 bytes**.

In Test3.java, we simulate these data accesses. Redirect the output to input3.txt.

```
for i = 0 to 14
  for j = 0 to i
    access a[i, j]
for j = 0 to 14
  for i = 0 to j
    access a[i, j]

Address = base address of array + 4 * (15*i + j)
Page number = Address / 64
```

Test Run #4 (Test4.java)
For your reference string, generate 1,000 random numbers in the range 1-5. Note that we want to experience page faults only occasionally, so we make sure that the number of referenced pages is 1 more than the number of available frames. For instance, if we had 8 frames, we'd want to experiment with page numbers 1-9. Redirect the output to input4.txt.

Findings of experiments

For 4 page frames, and using the 4 input text files, enter the number of page faults here:

Algorithm	input1.txt	input2.txt	input3.txt	input4.txt
Clairvoyant				
FIFO				
2nd chance FIFO				
LRU				

When you are finished, please submit all the source files that you modified, as well as the results of your experiments.