CS 375 Operating Systems **Spring 2015**

PROGRAM 2 (AND LAB 5)

The goal of Program 2 and Lab 5 is to write a crude simulator of a Virtual Memory system, designed to illustrate the performance (in terms of page fault rate only) of three different page replacement policies, across different kinds of "input." You will work in teams of two; you may select your partner, or I will put you together with someone during lab on April 2nd.

Your program should take two input parameters from the command line:

(1) the size of the memory system to simulate, measured in number of memory frames, and
(2) the name of an input file that contains a sequence of operations that simulate adding and removing processes from your simulation, and having those processes make memory references, as described below.

Thus, your program should be invoked as follows:

```
$ ./program2 <frames-of-memory> <input-file>
```

The input file contains lines as follows:

```
START <process-number> <address-space-size>
TERMINATE <process-number>
REFERENCE <process-number> <virtual-page-number>
```

A START line indicates that a new process is starting, is uniquely identified by `<process-number>`, and contains `<address-space-size>` pages of memory. You may assume that no process with this process number exists at the time your program encounters a START line.

A TERMINATE line indicates that process number `<process-number>` has completed and will not make any more memory references. You may assume that a process with this process number exists at the time your program encounters a TERMINATE line.

A REFERENCE line indicates that process `<process-number>` accesses page number `<page-number>` within its own virtual address space. If the page-number is less than 1 or greater than the number of pages in the identified process's virtual address space, or if no process with the indicated number is currently being simulated, you should print an error message, ignore the reference, and continue. For this assignment, we care only about your programs' performance on well-formed and meaningful input.

Your program should read these lines of input, in order, and very crudely simulate the operation of a virtual memory system, just by having the system pretend to share `<frames-of-memory>` frames of memory across the processes in your simulation. You should implement three different page replacement algorithms: RANDOM, LRU, and LFU.

A short explanation on page replacement appears below, we will talk about it more during lab on April 2nd, and we will cover the material in more detail during the week of April 13th. You will not need to implement the page replacement algorithms for the code that you submit for Lab 5.

Page Replacement

As you know, running processes share physical memory for their address spaces. When the total amount of physical memory is less than the amount that is needed to support all active processes' address spaces, some frames must be shared by multiple processes. When a process generates a reference to part of its virtual address space, the virtual memory system of the OS must copy some page out to disk and use that freed up frame to store the page that is being accessed. This "demand paging" influences the performance of programs and the system as a whole. The fewer times the system moves data to and from disk, the faster the memory accesses will be, and the better the performance of the processes that make them. Decisions about which page of memory to copy out to disk, when a process makes a reference to a non-resident page, significantly influence performance when memory pressure is high. A *page replacement algorithm* selects the page of memory for replacement.

You will implement the following algorithms:

- RANDOM: This algorithm selects a random frame number to evict, when a page of memory is needed for a memory reference.
- LRU: Least Recently Used. This algorithm selects the page that was least recently accessed.
- FIFO: First In First Out: This algorithm selects for removal the page that has been resident in memory for the longest period of time.

Your program should simulate the effect that the memory reference string contained in the input file would have on a virtual memory system with each of these three page replacement algorithms.

The output of your program should consist of three numbers, representing the *page fault rates* of the three different algorithms. The page fault rate of a system is the number of times that a memory reference refers to a page of virtual memory that is not resident in physical memory (i.e. the number of faults), divided by the total number of memory references.

Format your output as follows, and print it to **`stdout`**:

```
Page Fault Rates:
RANDOM: <x>%
LRU: <y>%
FIFO: <z>%
```

Replace **`<x>`**, **`<y>`**, and **`<z>`** with the page fault rates of the three page replacement algorithms.

Please do not misunderstand the scope of this assignment. You are not implementing a full blown system, you are not creating processes or threads to support the processes named in the input file, you are not actually storing pages of virtual memory on a disk, or using frames of real memory, you are not even performing translation of virtual addresses to physical addresses (although you do need to "map" per-process page numbers to physical memory frame numbers… this can be done with a simple map or array). You just need to maintain enough information in your program to calculate the page fault rates that would occur, if the string of memory references were generated by a set of processes to a real virtual memory system.

Lab 5

You will turn in a "checkpoint" toward successful completion of Program 2, as Lab 5. This checkpoint should include a program that at least:

- Reads the input file in the appropriate format.
- Maps virtual memory references from processes into frames of physical memory, and keeps track of whether and where each process's virtual memory pages are stored in memory. When a process generates a REFERENCE to a page of memory that is not resident, and when there is available memory, then your program should allocate a frame to the process making the request.
- Supports START and TERMINATE. In particular, when a process terminates, all frames of physical memory that are being used for the terminating process's address space should be freed up.

One thing your Lab 5 submission does *not* have to do, is select victim pages when memory is completely used. If physical memory “fills up” to the point that there are no free frames of memory, your program can simply count this as a page fault, but not implement any algorithm to replace the memory, for now. Your Lab 5 submission may also operate on only very small hand-made input files, to illustrate its correct operation.

One team member should submit your Lab 5 code to Blackboard, in the format we use for Labs and Programs. Make sure it is clear who your partner is.

Program 2

Your Program 2 submission should extend your Lab 5 code to include the three page replacement algorithms, and to accurately report page fault rates.

Another important aspect of Program 2 is your design and implementation of a program that *generates input files* for your crude Virtual Memory simulator. You should design your solution to this assignment to enable your code to investigate the effectiveness of the three page replacement algorithms across memory reference strings that exhibit different characteristics. LRU is designed to work well when programs exhibit significant locality of reference within their address space. Some algorithms work better than others when memory pressure his high. Some programs generate memory references in phases, where references to one area of a virtual address space are frequent for the first part of the program, but then the code moves on to access another part of the virtual address space during a subsequent phase. Design and implement a program that takes input parameters (you decide what they should be), and generates input files for your virtual memory assessment program. Do this such that you could use the generated input files to successfully evaluate the effectiveness of different page fault algorithms (not necessarily just the three that you are implementing as part of this assignment), in terms of page fault rates. You may decide to vary the number of processes that make memory references, the degree to which the memory reference strings exhibit locality of reference, the total amount of memory required by all of the processes in the input file, etc.