COP4600.001C11: Operating Systems
Project Three: Virtual Memory and Memory Management

June 23, 2011

1 Instructions

This project involves developing a rudimentary virtual memory simulator. You are given a file containing a simple memory trace of a program, and you must develop a simple pure-demand paging virtual memory simulator that reads the program trace, runs one of several page replacement aglorithms on the trace, and reports various statistics for each page replacement algorithm. The simulator must track pages in memory, page faults, and disk writes. See section 9.4, and specifically 9.4.1, of the textbook for more information on important concepts for this project. For this project you may work alone or with one other classmate. If you work with a classmate, make only one submission, with both names included clearly in the report. Yes, this project requires a report.

2 The Trace File

Please download the `trace` file that accompanies these specifications. In it you will find exactly 1Mi entries of the form

```
9c2823cf w
9c281ea3 r
9c282a52 w
...
```

The trace file represents memory accesses that a fictitious process makes during execution. It holds 1Mi entries, and each entry is an (`address, action`) pair, specifying in hexadecimal, the 32-bit virtual memory address the program is accessing, and the action (either read or write) performed at that virtual memory address. Your program must run under the assumption that this file exists in the same directory as your source files and make file, and that it is named `trace`.

3 Syntax

Write a program with invocation syntax

```
$ ./proj3 <numframes> <pagesize> <lru|mru|lfu|mfu|rand|opt|clock>
```

`numframes` is the number of frames of physical memory, and `pagesize` is the size of each memory frame (equal to the size of a virtual memory page) in kibibytes (KiB). You do not type the `<`s, `>`s, or `|`s, of course. Your project must support at least three of the page replacement algorithms listed below:

- `lru`: least recently used
- `mru`: most recently used
- `lfu`: least frequently used
- `mfu`: most frequently used
- `rand`: random
- `opt`: opt
- `clock`: clock

Note the invocation syntax above and make sure your program does not throw a segmentation fault if the correct number of arguments are not given. There is one exception to this rule: Your program must print the algorithms supported, space-separated, and exit if it is run with zero arguments. Your program must use the syntax given above for specifying the page replacement algorithm.

4 Output

Your program should produce the following statistics each time it is run with correct arguments.

- Number of frames (number of frames of physical memory)
- Page size in bytes
- Algorithm
- Number of disk reads (page faults)
- Number of disk writes

Please see the following sample output to see how to structure your printf() format strings. Test commands similar to these will be run on your submission. If the user does not supply your program with a valid argument, simply print `incorrect arguments`, and quit (note the exception with zero arguments).

```
$ make all
$ ./proj3
  lru mru rand

$ ./proj3 0
  incorrect arguments

$ ./proj3 0 1
  incorrect arguments

$ ./proj3 12 4 arc
  incorrect arguments

$ ./proj3 12 4 mru
  // This is for a different tracefile.
  // Do not compare your numbers to these.
  12 frames
  4KiB pagesize
  most recently used algorithm
```

```
  48379 page faults
  18288 disk writes

$ ./proj3 32 256 opt
  // This is for a different tracefile.
  // Do not compare your numbers to these.
  32 frames
  256KiB pagesize
  optimal algorithm
  3918 page faults
  1992 disk writes
```

5 Conventions

Your submission will receive severe penalties if it does not adhere to the following conventions.

5.1 Required Files

Please submit a zip file named `lastname-firstname-proj3.zip` (all letters lowercase!) containing the following files.

```
Makefile
proj3.c
proj3.pdf
```

The `proj3.pdf` file is your report.

5.2 Requirements of the makefile

Your makefile should support the following commands when run from the directory containing your source files:

```
$ make all
$ make clean
```

The `all` command should produce your executable file named `proj3`. The `clean` command should remove all temporary and compiled files, leaving only the source files and report (if applicable) required of you for this project.

6 Report

Your team must submit a single report in pdf format. If you worked with another classmate, include both names at the top of the report. I am not providing a report template file, but you can use any standard doc, odf, or LaTeX template file you find online from, e.g., IEEE or ACM. If you find another standard template somewhere that you can use, you may do so if you provide a link to the source.

You have a great deal of flexibility with your report, but you must include sufficiently extensive treatment of the following three items:

- Explanations of your implementations of the algorithms. Include figures if you must. Someone with moderate knowledge of C should be able to recreate your algorithms from your descriptions.
- Comparison of the performance of all algorithms on the trace file. Include discussion, graphics like charts and/or plots, and reasoning for the relative performance of each algorithm.
- Answer the following questions:
    1. By which metrics do you judge a page replacement algorithm?
    2. Of the three page replacement algorithms you implemented, can you call one of them "the best." If so, why? If not, why not?
    3. Why did you choose the algorithms that you chose to implement?
    4. For each of the algorithms given (*not* just the ones you implemented), give a high-level example of a workload with which it would perform best, and give an example of a workload with which it would perform poorest.

7 Grading

The following deductions apply to this 100-point project

- -100 if it does not compile with `$ make all`
- -100 for late submission (don't forget about your token!)

- -20 for each algorithm incorrectly implemented less than the three required (i.e. only implementing two algorithms correctly will result in -20)
- -10 if an edge case produces a segmentation fault
- -30 for no report in pdf format
- -10 for insufficient treatment of each of the three main sections of the report (see 6)

Partial credit is possible where appropriate. You can earn up to 30 points extra credit if you integrate into your program working set (see textbook section 9.6.2) size detection. If you attempt this, please include a section in your report on your implementation. I will give partial credit for earnest attempts at the extra credit. I can tell you that Δ, the working set window, changes abruptly many times during execution, and it changes at constant intervals. Figure 9.22 and the information in the blue box surrounding it will be especially helpful to you if you decide to attempt the extra credit.