Programming Assignment 4 – Virtual Memory Page Replacement Policies

Assigned: Tuesday, 11/1/2005, 2pm
Due: Tuesday, 11/15/2005 before 4pm

In this assignment, you will write a program that simulates the virtual memory page replacement algorithms described in section 8.2 of the required textbook by William Stallings (2004) *Operating Systems: Internals and Design Principles (5th Edition)*, pages 354 to 357: Optimal (OPT), Least Recently Used (LRU) and First-In-First-Out (FIFO). ***Graduates only (bonus for the others):*** Implement the single use-bit clock policy, too (CLOCK). This project contains three separate executables: a page replacement simulator `vmsim`, a page reference generator `vmgen`, and a page statistics program `vmstats` that produces a consolidated data file used for performance plots.

1. Specifications of the page replacement simulator `vmsim`

- ✓ `vmsim` must accept three command-line arguments in the following order: (a) the total number of physical memory frames (maximum 100), (b) an input filename where a sequence of page references is stored, (c) the chosen algorithm. The three (***grads:*** four) possible values for (c) are the strings `opt`, `lru` and `fifo` (***grads:*** and `clock`). (If `vmsim` is run with the wrong arguments or no arguments, it should print out usage instructions and exit.) Example:

  ```
  >vmsim 5 vmrefs.dat lru
  ```

- ✓ The format of the input file must be a simple ASCII sequence of integers in the range 0 to 99 separated by spaces, for example: `51 7 34 0 8 45 21`. No symbols or formatted text, just a pure sequence of space-separated integer numbers. This file can be created by hand or generated by the `vmgen` program (see section 2. below).

- ✓ `vmsim` will first read all the memory references from the input file and store them in a local array. Then, it will play back these references one by one and print out for each reference the current allocation state of physical memory frames in the following format:

  ```
  [51| 7|34|  |  ]
  ```

  This representation means that frames 0, 1 and 2 are occupied by pages 51, 7 and 34, while frames 3 and 4 are empty. It must start with the open square bracket `[`, end with the closed square bracket `]` and separate frames with the vertical bar `|`. One-digit page numbers should have an extra space to the left so that frames are always 2 characters wide (2 spaces for an empty frame; ***grads:*** do *not* display the use-bit asterisk in the CLOCK policy). Each page fault should be signaled by an F character two spaces to the right of the closed bracket, for example:

  ```
  [45| 7|34| 0| 8]  F
  ```

- ✓ After processing all the memory references, `vmsim` should finally print the total number of page faults and the miss rate (page faults divided by number of references). It should start counting page faults and page references only *after* all frames have been initially filled (see book, Fig. 8.15), e.g., with 3 frames that means starting at the 4th step. Use this printout format:

  ```
  Miss rate = 237 / 997 = 23.78%
  ```

2. Specifications of the page reference generator `vmgen`

- ✓ `vmgen` must accept three command-line arguments in the following order: (a) the range of page references (maximum 100), (b) the length of the sequence and (c), the name of the file that will be generated. (If `vmgen` is run with the wrong arguments or no arguments, it should print out usage instructions and exit.) Example:

```
>vmgen 10 200 vmrefs.dat
```

- ✓ `vmgen` will then generate a sequence of the desired length containing random page numbers uniformly drawn between 0 and the range minus one (i.e., 200 page numbers between 0 and 9 in the example above). Important: no page number in the sequence should be equal to the number that precedes it (hence, follows it, too). For example: `2 7 7 0 0 0 3 3 …` is not a valid sequence but `2 7 0 3 …` is. `vmgen` must write this sequence into the file given in input.

3. Specifications of the page statistics program `vmstats`

- ✓ `vmstats` must accept four command-line arguments in the following order: (a) the minimum number of frames (no less than 2), (b) the maximum number of frames (no more than 100), (c) the frame number increment (positive), (d) the input filename containing the references. (If `vmstats` is run with the wrong arguments or no arguments, it should print out usage instructions and exit.) Example:

```
>vmstats 5 40 10 vmrefs.dat
```

- ✓ `vmstats` will loop over the three (***grads:*** four) page replacement methods: `opt`, `lru` and `fifo` (***grads:*** and `clock`) and for each method, it will loop over the number of frames, starting at the mininum and applying the increment until it exceeds the maximum (i.e., 5, 15, 25, 35, in the example above). For each {method, number of frames} pair it will calculate the page fault rate using the reference file given in input and print out a one-line message containing this rate. Use this printout format:

```
lru, 15 frames: Miss rate = 237 / 997 = 23.78%
lru, 25 frames: Miss rate = 163 / 997 = 16.35%
...
```

- ✓ Therefore, the implementation of both `vmsim` and `vmstats` must rely on a common utility simulation engine (entry-point function) with the difference that `vmsim` is going to run the simulation once and in verbose mode (displaying all the allocation steps and the final miss rate; see 1.), whereas `vmstats` is going to run the simulation repeatedly and silently (displaying only the final miss rate for each pair in the loops). In accordance with this design, please keep the simulation engine in a source file distinct from the `vmsim` and `vmstats` capstone code.

- ✓ As a by-product, `vmsim` must also generate one consolidated results file containing the matrix of all the calculated miss rates. The results file must be named `vmrates.dat` and must be formatted in the following way: it should contain exactly four (***grads:*** five) lines of *N* space-separated numbers, where *N* is the number of numbers of frames (4 in the example above). The first line must be the sequence of numbers of frames (`5 15 25 35` in the example above),

the other three (***grads:*** four) lines must be the sequences of rate values for each number of frame. These sequences must be in the following order: `opt`, `lru` and `fifo` (***grads:*** and `clock`). `vmrates.dat` should not contain formatted messages or symbols, just four (***grads:*** five) lines of pure space-separated numerical sequences with line feeds between sequences.

Specifications of the plots

- ✓ Once you have generated `vmrates.dat`, you must use the graphing tool of your choice to plot the three (***grads:*** four) miss rate curves, similarly to Figure 8.17 of the textbook. This plot file should be a JPEG or PNG image called `vmrates.jpg` or `vmrates.png`.

- ✓ Benchmark parameters and a test reference input file will be sent to you next week. Along with your source and documentation files, you will have to submit the `vmrates.dat` stats file and the plot image file resulting from the execution of `vmstats` against these parameters.

Project Requirements, Submission & Required Documentation

- ✓ Follow all the requirement and submission guidelines *exactly* as prescribed in Stallings' book on pages 154 to 156, replacing the name of the executable `myshell` with `vmsim`, `vmgen` and `vmstats`, and naturally any mention of "shell" and shell features with these programs' features. I remind you that Stallings' guidelines include, among other things: appropriately structuring and properly commenting the code (3.), not including object code or executables in your submission (5.), naming the makefile exactly `makefile` (6.), naming the readme file exactly `readme`, and writing a reasonably detailed UNIX-like readme documentation (2.). Your documentation should describe the problem's background, the programs' usage and parameters, and a sample of a typical outcome.

- ✓ Your programs *must* compile without warnings and run properly under Gentoo Linux, the system used in the ECC lab (check their Web site for hours). You may develop and test your programs on your own UNIX machine (you can also produce the stats file and plots from your machine), but it is your responsibility to make sure that they also work properly on the Gentoo installation of ECC, under the default ECC Lab shell. There will be a penalty if they don't.

- ✓ Place all necessary files and documents in a compressed tarball using tar and gzip (`.tar.gz` or `.tgz`) or, alternatively, a zip file (`.zip`) (please do not use the `bzip2` format). Email this file to the instructor doursat@unr.edu within two weeks, before Tuesday, 11/15/2005 at 4pm. Do not turn in printouts in any form. Late assignments will be marked down according to the late policy published on the course Web page.

- ✓ Do not show, exchange or copy code with anyone, and do not look for a solution on the Web! Plagiarism *will* be detected and severely penalized. This must remain an individual effort.

- ✓ Please keep clean code organization, layout and coding conventions, as these will also be an important part of the grade.

Thank you and good luck!