UNIVERSITY OF WISCONSIN
Computer Sciences Department

CS 537
Fall 2020

Barton Miller

Programming Assignment #4: Page Replacement Simulator

Handed out: Friday, November 13
~~Due: Wednesday, December 2, 5pm~~
Now due: Thursday, December 3, 5:30pm

Simulating Page Replacement Algorithms

The goal of this assignment is to evaluate several page replacement algorithms. You will use real memory trace data from UNIX systems to test these algorithms.

Your assignment is to write a program that reads in the trace data and simulates the page replacement scheduling. You will keep track of various performance statistics and print these out at the completion of the trace file.

A. Running your Program

You will build three executables called "537pfsim-fifo", "537pfsim-lru", and "537pfsim-clock". The trace file will be named as the a value on the command line. For example:

```
537pfsim-lru -p 8192 -m 2 tracefile1
```

The parameters are described below in the Simulator Parameters section.

B. The Trace Files

The trace files contain a sequence of records that report on memory references made by the running process (and not the operating system) Each record is a line in the file that describes one memory reference. The record has two decimal numbers, the process ID (PID) and virtual page number (not the full address, but just the VPN). Each number in the trace file is a long integer ("unsigned long"). For example, a line in the trace file might be:

```
1234 10000
```

which indicates that process 1234 referenced page 10000.

The trace files can be found on AFS at `~cs537-1/public/proj4`.

Some of the trace files are large, in the millions of lines, so you will have to be conscious of efficiency issues in your program design. This means that any **linear time** algorithm will cause your program to run unacceptably slow.

C. Program Information

Your program will be structured as a continuous loop that reads trace records and advances the simulated time.

C.1. Important Events

Your program will maintain a notion of current time. The "clock" in your simulator will be a variable that holds the value of the current time. the clock tick will be 1 ns. The clock will start at time zero (0) and advances each time a process makes a memory reference and also while waiting for a disk I/O to complete.

Several things can happen while a simulator is running:

1. A successful (mapped) memory reference will be made. In this case, you simply advance the simulated time by 1 ns.
2. The process will have a page fault, so the process will block until I/O is completed.
3. The process that is currently running completes. This happens after you execute the last memory reference in the trace file for that process. In this case, you need to update the various performance statistics (see below) and remove the process from any run/ready queues.
4. A disk I/O will complete: The process that completed its I/O will be considered to be runable again.

C.2. Simulator Details

Here are some important details:

1. The traces in the files don't include information about whether the references are reads or writes. For simplicity, assume that all references are reads (you don't have to handle dirty pages).
2. Memory references take 1 ns. Transferring a page from disk to memory takes 2 ms.
3. Memory starts out empty; all page frames are free. You are tracking only the memory used by processes and not considering any memory used by the operating system kernel.
4. There is only one disk, so requests for pages must be queued. The disk can do only one read at a time.
5. Some simplifying assumptions:
    - Not that you should not account for time to handle a disk I/O. Assume that it takes zero time to start a disk I/O or handle its completion.
    - Do not try to account for process context switch time.
6. You will need to make two passes over the trace file. The first pass will include finding all the PID's, and marking the start and end of execution (first and last memory reference) of each process. A process is considered to have terminated (freeing all of its memory) after its last memory reference in the trace file.
7. Do not read all the traces into a data structure at one time; process the traces from the file. Real trace files have billions of records and this does not work in the real world.
8. If a process has a page fault then it is *blocked* until the page is loaded into memory; do **not** process any further memory references from that process until the page is loaded. You should, however continue to process memory references from other unblocked processes. This means that you will need to have one pointer into the trace file for each active process.
    - To move around in the file, you can use the `fseek()` library call
    - To get the value of your current position in a file, you can use `ftell()`.
9. When a process blocks for a page fault, you have to remember where you were in the trace file for that process. You then start processing references from another process.
10. Important rule: In general, you are always processing the earliest unprocessed trace in the file for a runable process.

C.3. Scheduling Algorithms

The details of the particular scheduling algorithm (you will implement three) should be isolated in a single module. All your program, except for the scheduling algorithm, should be the same for the different versions.

1. The first version of your program will implement global FIFO page scheduling. Pages are first brought into memory will be discarded first.

2. The second version of your program will implement global LRU (least recently used) scheduling. Pages are kept in memory according to their overall wall clock reference time. When all the page frames are filled and a new page is references, the page that is referenced the furthest in the page is removed.
3. The third version of your program will implement a global Clock algorithm.

C.4. Simulator Parameters

Your simulator can take the following values as command line parameters. These values are:

Page size (`-p` option):
: This value is the number of bytes and must be a power of two. So a parameter of `-p 512` means that the page size is 512 bytes. The default value if this option is not specified is 4096.

Real memory size (`-m` option):
: This value is the number of MB. So a parameter of `-m 4` means that the memory size is 4MB. The default value if this option is not specified is 1 (MB).

C.5. Performance Data

Your simulator will keep trace of several performance statistics and print out these results when the simulator completes. These statistics are:

Average Memory Utilization (AMU):
: For each clock tick, examine how many pages frames are occupied and average this over each clock tick that the simulator runs.

Average Runable Processes (ARP):
: This value is an average of the number of processes that are running (or runable). This value is averaged over each clock tick for which the simulator runs.

Total Memory References (TMR):
: This is simply a count of the total number of memory references in the trace file.

Total Page Ins (TPI):
: This is the total number of page faults (resulting in disk transfers into memory).

Running Time:
: This is the total number of clock ticks for the simulator run.

D. Software Design Issues

Good design on this assignment will save you, literally thousands of lines of code. The page replacement algorithm should be encapsulated in a PageAlgorithm module. In one version of the program, this module will do something simple, like a FIFO scan of pages. In another version, it may need much more complicated bookkeeping information to track last reference times for LRU.

All other parts of your program should be the same, so you can re-use them for the different versions.

You have plenty of time for this assignment, but don't delay in getting started! Work on a design and and initial structure for your module and then talk with your TA.

Some of the modules that you might want to build will be for input, processes, page tables, page frame table, paging device (disk queue), and statistics.

Your makefile will have build rules for the three separate programs: `537pfsim-fifo`, `537pfsim-lru`, and `537pfsim-clock`.

E. Deliverables

You can work individually or in a group of two. In either case, you will turn in a single copy of your program, clearly labeled with the name and logins of both authors.

You will turn in your programs, including all .c and .h files and your makefile. Also include a README file which describes a little bit about what you did for this project.

Note that you must run your programs on the Linux systems provided by the CS Department. You can access these machines from the labs on the first floor of the Computer Sciences Building or from anywhere on the Internet using the ssh remote login facility. If you do not have ssh on your Windows machine, you can download this client:

> SSHSecureShellClient-3.2.9.exe

Your program should run on the test trace files we provide. These files can be found in `~cs537-1/public/proj4`

You should run your simulator with page sizes of 512 and 4096 bytes and with physical memory sizes of 1 MB, 4MB, and 32 MB. That's a total of 6 runs per trace file per algorithm. (Of course, you'll probably want to use a shell script to run all these different variations.)

F. Handing in Your Assignment

Your CS537 handin directory is `~cs537-1/handin/`*`your_login`* where *`your_login`* is your CS login. Inside of that directory, you need to create a `proj4` subdirectory (unless the TAs created one for you already).

Copying your files to this directory is accomplished with the `cp` program, as follows:

```
shell% cp *.[ch] makefile README ~cs537-1/handin/your_login/proj4
```

You can hand files in multiple times, and later submissions will overwrite your previous ones. To check that your files have been handed in properly, you should list `~cs537-1/handin/`*`your_login`*`/proj4` and make sure that your files are there. The handin directories will be closed after the project is due.

Whether you are working individually or in pairs, you should:

1. Submit only one copy of your code.
2. Create a file called partner.txt in each of of your `proj4` directories. It should have a line in the file for each person who worked on the code (so, 1 or 2 lines). Each line will have you name, CS login and NetID.

G. Original Work

This assignment must be the original work of you and your project partner. Unless you have explicit permission from Bart, you may not include code from any other source or have anyone else write code for you.

Use of unattributed code is considered plagiarism and will result in academic misconduct proceedings (and "F" in the course and a notation on your transcript).

Last modified: Wed Nov 18 09:06:39 CST 2020 by bart