CS 1550 – Project 3: VM Simulator
Due: Sunday, November 13, 2016, by 11:59pm

Project Description
In class, we have been discussing various page replacement algorithms that an Operating System implementer may choose to use. In this project, you will compare the results of four different algorithms on traces of memory references. While simulating an algorithm, you will collect statistics about its performance such as the number of page faults that occur and the number of dirty frames that had to be written back to disk. When you are done with your program, you will write up your results and provide a graph that compares the performance of the various algorithms.

The four algorithms for this project are:

Opt – Simulate what the optimal page replacement algorithm would choose if it had perfect knowledge

Clock – Use the better implementation of the second-chance algorithm

Aging – Implement the aging algorithm that approximates LRU with an 8-bit counter

Working Set Clock – See the section below.

You may write your program in C/C++, Java, Perl, or Python as long as it runs on thoth.cs.pitt.edu.

Implement a page table for a 32-bit address space. All pages will be 4KB in size. The number of frames will be a parameter to the execution of your program.

How it Will Work
You will write a program called vmsim that takes the following command line:

./vmsim –n <numframes> -a <opt|clock|aging|work> [-r <refresh>] [-t <tau>] <tracefile>

The program will then run through the memory references of the file and display the action taken for each address (hit, page fault – no eviction, page fault – evict clean, page fault – evict dirty).

When the trace is over, print out summary statistics in the following format:

```
Algorithm: Clock
Number of frames:       8
Total memory accesses:  1000000
Total page faults:      181856
Total writes to disk:   29401
```

Implementation
We are providing two sample memory traces. We will grade with two additional ones. The traces are available at /u/OSLab/original/ in the files bzip.trace.gz and gcc.trace.gz

Each trace is gzip compressed, so you will have to copy each trace to your directory under /u/OSLab/ and then decompress it like:

```
gunzip bzip.trace.gz
```

In the resulting trace file is a sequence of lines, each having a memory address in hexadecimal followed by a R or W character to indicate if that access was a read or a write. For example, gcc.trace trace starts like this:

```
0041f7a0 R
13f5e2c0 R
```

If you are writing in C, you may parse each line with the following code:

```
unsigned int address;
char mode;

fscanf(file, "%x %c", &addr, &mode);
```

Please Note

Implementing OPT in a naïve fashion will lead to unacceptable performance. It should not take more than 5 minutes to run your program.

Working Set Clock Algorithm

Use the line number of the file as your virtual time.

Use the tau you specify on the command line to determine that a page is out of the working set.

We will use a time periodic refresh as in NRU to reset the valid pages' R bits back to zero. Use the command line parameter to set this time period. When resetting the R bit to zero, record the current virtual time in your page table entry for that page.

We will use the following algorithm variant. On a page fault with no free frames available:

1. Scan the page table looking at valid pages, each time continuing from where you left off previously
2. If you find one that is unreferenced and clean, evict it and stop.
3. Along the way, if you encounter a page that is unreferenced, older than tau, and dirty, write it out to disk and mark it as clean.
4. If you get through the whole page table with no unreferenced, clean pages, evict the page with the oldest timestamp

Write Up

For Aging, you have a refresh parameter to set. Try to find a good refresh period that works well. You do not need to find the absolute minimum, just approximately how long to wait. Plot your results in a graph and discuss why your choice of refresh seemed to be the best.

For WSClock, you have two parameters to vary: refresh and tau. Use the refresh that was good for Aging and try to determine a good choice of tau. Plot your results. Vary the refresh using the tau you determine and explain what seems to be a good choice for tau and refresh.

For each of your four algorithms (with Aging using the proper refresh you determined), describe in a document the resulting page fault statistics for 8, 16, 32, and 64 frames. Use this information to determine which algorithm you think might be most appropriate for use in an actual operating system. Use OPT as the baseline for your comparisons.

File Backups

One of the major contributions the university provides for the AFS filesystem is nightly backups. However, the /u/OSLab/ partition is not part of AFS space. Thus, any files you modify under your personal directory in /u/OSLab/ are not backed up. If there is a catastrophic disk failure, all of your work will be irrecoverably lost. As such, it is my recommendation that you:

Backup all the files you change under /u/OSLab to your ~/private/ directory frequently!

Loss of work not backed up is not grounds for an extension. YOU HAVE BEEN WARNED.

Requirements and Submission

You need to submit:

- Your well-commented program's source
- A document (.DOC or .PDF) detailing the results of your simulation with a graph plotting the number of page faults versus the number of frames and your conclusions on which algorithm would be best to use in a real OS
- **DO NOT** submit the trace files.

Make a tar.gz file as in the first assignment, named USERNAME-project3.tar.gz

Copy it to ~jrmst106/submit/1550 by the deadline for credit.