Project 3: Virtual Memory Simulator[1]

Due: Friday, November 08, 2019 @11:59pm
Late: Sunday, November 10, 2019 @11:59pm with 10% reduction per late day

Table of Contents

[1] Based upon Project 3 of Dr. Misurda's CS 1550 course.

CS/COE 1550 – Introduction to Operating Systems

Project Overview

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 three 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 three algorithms for this project are:

> **Opt** – Simulate what the optimal page replacement algorithm would choose if it had perfect knowledge
>
> **FIFO** – Implement first-in, first-out
>
> **Aging** – Implement the aging algorithm that approximates LRU with an 8-bit counter. Do a refresh of the R bits every P CPU cycles.

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.

Project Details

You will write a program called `vmsim` that takes the following command line:

```
./vmsim –n <numframes> -a <opt|fifo|aging> [-r <refresh>] <tracefile>
```

The program will then run through the memory references of the file and decide 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: FIFO
Number of frames:       8
Total memory accesses:  %d
Total page faults:      %d
Total writes to disk:   %d
```

Implementation

We are providing three sample memory traces. The traces are available at `/u/OSLab/original/` in the files `gzip.trace.gz`, `swim.trace.gz`, and `gcc.trace.gz`. We will use more trace files to test your program.

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

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

Your simulator takes a command-line argument that specifies the trace file that will be used to compute the output statistics. The trace file will specify all the data memory accesses that occur in the sample program. Each line in the trace file will specify a new memory reference. Each line in the trace will therefore have the following three fields:

Access Type: A single character indicating whether the access is a load ('l') or a store ('s').

Address: A 32-bit integer (in unsigned hexadecimal format) specifying the memory address that is being accessed. For example, "0xff32e100" specifies that memory address 4281524480 is accessed.

CPU cycles since last memory access: Indicates the number of CPU cycles that elapsed **since** the last memory access (i.e., the one on the previous line in the trace). For example, if the 5th and 10th cycles in the program's execution are loads, and there are no memory operations between them, then the trace line for the second load has "4" for this field. For the first line in the trace file, the CPU cycles number indicates the cycle number at which the access occurs.

Fields on the same line are separated by a single space. Example trace lines look like:

```
l 0x300088a0 6
s 0x1ffffd00 0
```

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

```
unsigned int address;
char mode;
unsigned int cycles;

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

Notes

1. Implementing OPT in a naïve fashion will lead to unacceptable performance. It should not take more than **5 minutes** to run your program.
2. In case of a tie, in the OPT algorithm, break the tie by selecting the least recently used page. In the case of tie for the aging algorithm, break the tie based on the dirty flag then the based on the virtual page number.

Write Up

For Aging, you have a refresh parameter (measured in number of CPU cycles) 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 each of your three algorithm implementations (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.

For FIFO, with the three traces and varying the total number of frames from 2 to 100, determine if there are any instances of Belady's anomaly. Discuss in your writeup.

File Backups

One of the major contributions the university provides for the AFS filesystem is nightly backups. However, the `/u/OSLab/` partition on thoth 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` or `QEMU` to your `~/private/` directory frequently!

Loss of work not backed up is not grounds for an extension.

Requirements and Submission

You need to submit onto Gradescope:

- Your well-commented program's source
- A document (.DOC or .PDF) detailing the results of your simulation as described above
- **DO NOT** submit the trace files!

No matter which language you select, the autograder should be able to run your program as:

```
./vmsim –n <numframes> -a <opt|fifo|aging> [-r <refresh>] <tracefile>
```

Grading Sheet/Rubric

Item	Grade
Program runs with command-line parsing and correct output format	10%

as tested by an empty trace file.	
OPT implementation	20%
FIFO implementation	20%
Aging implementation	20%
Writeup (Determining the optimal value of the refresh parameter)	10%
Writeup (The 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)	10%
Writeup (FIFO Analysis)	10%