Project 3: Virtual Memory Simulator[1]

Due: Friday, November 13th, 2020 @11:59 pm
Late: Sunday, November 15st, 2020 @11:59 pm with 10% reduction per late day

Table of Contents

PROJECT OVERVIEW **2**
PROJECT DETAILS **2**
IMPLEMENTATION 3
IMPORTANT NOTES 3
WRITE UP 3
EXTRA CREDIT 4
Extra Credit Task 1 (5 points): *4*
Extra Credit Task 2 (5 points): *4*
FILE BACKUPS 5
REQUIREMENTS AND SUBMISSION **5**
GRADING SHEET/RUBRIC **5**

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

Project Overview

In class, we have discussed various page replacement algorithms that an Operating System implementer may choose to use. We have also discussed local and global page replacement policies. In particular, in **local page replacement**, each process is allocated a certain number of physical memory frames, and when a page is to be evicted, the victim page is selected among the pages of the same process. In this project, you will simulate the Second Chance page replacement algorithm by reading traces of memory references that were generated by two processes while running on a 32-bit system and collecting relevant metrics. Because it is local page replacement algorithm, each process has its own frames (i.e., a percentage of the total available physical memory). While simulating the 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 ways to split physical memory up between the two processes. The page size, the total number of frames, and the memory split will be **command-line arguments** to the execution of your program.

You may write your program in C/C++, Java, Perl, or Python as long as it runs on thoth.cs.pitt.edu with the "./vsim" command specified below.

Project Details

Your virtual memory simulator, called `vmsim`, takes the following command line arguments:

```
./vmsim –n <numframes> -p <pagesize in KB> -s <memory split> <tracefile>
```

The memory split between the two processes is provided in the form *a:b*, where *a* and *b* are positive integers > 0 that represent the ratios of each process's memory allocation. For example, a 1:2 split means that the second process gets twice as many frames as the first process (i.e., the first process gets one third and the second two thirds). The sum of a and b evenly divided the total number of frames.

The program/simulator will run through the memory references of the trace file and decide the action taken for each address (memory hit, page fault with no eviction, page fault and evict clean page, page fault and evict dirty page).

When the trace is over (that is, after dealing with all the memory references for both simulated processes), `vmsim` prints out summary statistics **in the following format (as you know, gradescope grading requires very strict formatting**:

```
Algorithm: Second Chance
Number of frames: 8
Page size: 8 KB
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 `1.trace.gz`, `2.trace.gz`, and `3.trace.gz`. We will use more trace files to test your program when grading; these traces are for you to start testing 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 1.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 occurred while running two processes. Each line in the trace file will specify a new memory reference and the process that made the access. 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'). The 's' mode modifies the address and sets the dirty bit to true.

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 (in decimal) is accessed.

Process: Either 0 or 1, representing which of the two processes made the memory access.

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

```
l 0x012ff200 0
s 0xfe7fefc8 1
```

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

```
unsigned int address;
char mode;
int process;

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

Important Notes

1. As mentioned in class, when a dirty page is evicted, it has to be written to disk; if the page is clean, the evicted page is just abandoned.
2. If you are using Python, name your file vmsim and add the following shebang line at the beginning of the file: `#!/usr/bin/env python`

Write Up

Describe in a document the resulting page fault statistics for the following memory splits: 1:1, 1:3, 3:1, 3:5, 5:3, 7:9, 9:7. Run your simulation with the following memory configurations:

- a total of 8 frames and a page size of 4 KB
- a total of 1024 frames and a page size of 4KB
- a total of 8 frames and a page size of 4 MB
- a total of 1024 frames and a page size of 4 MB

Extra Credit

You can do the following tasks for extra credits.

Extra Credit Task 1 (5 points):

Read 64-bit memory trace files. 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'). The 's' mode modifies the address and sets the dirty bit to true.
>
> **Address:** A 64-bit integer (in unsigned hexadecimal format) specifying the memory address that is being accessed. For example, "0x7894ff32e100" specifies that memory address 132581332017408 (in decimal) is accessed.
>
> **Process:** Either 0 or 1, representing which of the two processes made the memory access.

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

```
l 0x4567ab12ff200 0
s 0x111234ffe7fefc8 1
```

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

```
unsigned long int address;
char mode;
int process;

fscanf(file, "%c %lx %d", &mode, &addr, &process);
```

We are providing three sample 64-bit memory traces. The traces[2] are available at `/u/OSLab/original/` in the files `1-64.trace.gz`, `2-64.trace.gz`, and `3-64.trace.gz`. We will use more trace files to test your program when grading; these traces are for you to start testing your program.

Extra Credit Task 2 (5 points):

Find a case for Belady's anomaly in any of the trace files with any of the configurations and report it in your writeup.

[2] The 64-bit trace files for the extra credit are adapted from https://www.cis.upenn.edu/~milom/cis501-Fall12/traces/trace-format.html

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` 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!

The autograder should be able to run your program as:

```
./vmsim –n <numframes> -p <pagesize in KB> -s <memory split> <tracefile>
```

Grading Sheet/Rubric

Item	Grade
Program runs with command-line parsing and correct output format as tested by an empty trace file.	10%
Second Chance implementation	30%
Local Page Replacement	30%
Writeup	30%
Extra Credit Task 1	5%
Extra Credit Task 2	5%