

CS 1550 – Project 5: Virtual Memory Simulator

Due: Sunday, December 10, 2024, by 11:59pm

Project Description

In class, we discuss 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

Clock – Implement the clock version of the second-chance algorithm

LRU – Evict the least recently used page

You may write your program in C/C++, Java, or Python, as long as it compiles and runs on the versions of the compiler/interpreter on thoth.cs.pitt.edu.

Implement a **two-level page table** for a **32-bit address space** with **512 entries in the root**. All **pages will be 8KB** 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|lru> <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 simulating the trace is over, print out summary statistics in the following format:

```
Algorithm: %s
Number of frames: %d
Total memory accesses: %d
Total page faults: %d
Total writes to disk: %d
Number of page table leaves: %d
Total size of page table: %d
```

Implementation

We are providing two sample memory traces. We will grade with two additional ones. The traces are available on thoth in /u/OSLab/original as the files `ls.trace.gz` and `cc.trace.gz`

These files will need to be decompressed using `gunzip`.

The resulting trace files have been produced using a valgrind tool called “lackey” that logs program memory accesses. It prints memory data access traces that look like this (without the explanatory comments):

```
I 0023C790,2 # instruction fetch at 0x0023C790 of size 2
I 0023C792,5
  S BE80199C,4 # data store at 0xBE80199C of size 4
I 0025242B,3
  L BE801950,4 # data load at 0xBE801950 of size 4
I 0023D476,7
  M 0025747C,1 # data modify at 0x0025747C of size 1
I 0023DC20,2
  L 00254962,1
  L BE801FB3,1
I 00252305,1
  L 00254AEB,1
  S 00257998,1
```

Every instruction executed has an "I" (instruction) event representing its fetch from RAM (a memory access). When executed, some instructions do additional memory accesses and are followed by one or more "L" (load), "S" (store) or "M" (modify – a load followed by a store at the same address) events indented by one space. Some instructions do more than one load or store, as in the last two examples in the above trace.

Modify is for an instruction like `INC` (increment) that both loads and stores in the same step. Count it as both a load and a store of the specified address.

Some lines are not in this format as they are other output from the program or the tool. Ignore lines not in the format. Do not assume the file will always have a particular header – skip it based upon the contents.

For simplicity, you may treat each memory accesses as falling within a single page, basically ignoring the size and assuming every access is at the specified address and of size 1.

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.

Write Up

For each of your three algorithms, 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

Backup all of your project files 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 (.DOCX 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 named USERNAME-project5.tar.gz

Upload your file to Canvas by the deadline for credit.