Project 4: Trace-Driven Simulation of Page Replacement
CS 460

Introduction

This assignment should be programmed entirely in C and must compile and run on the lab's Linux environment or similar. Copying & pasting code from anywhere is considered cheating and results in an automatic F.

Background

If total memory requirements exceed the physical (main) memory, it is necessary to replace pages from memory to free frames for new pages. Two replacement algorithms are First In First Out (FIFO) and Least Recently Used (LRU). Refer to the lecture slides for more information about the two algorithms.

Programming Task

Write a program that implements the FIFO and LRU page-replacement algorithms. A sample page reference sequence is provided in the file "pageref.txt". Report the number of page faults and latency incurred by each algorithm (see details below). Implement the replacement algorithms so that the number of available page frames can vary as specified (e.g., 20). Assume that demand paging is used, i.e., page frames are initially free.

Requirements

Input Parameters

The program should accept three input parameters to specify:

1. The algorithm ("FIFO" or "LRU")
2. The number of physical page frames available (will fit in an `unsigned char`)
3. The input file that contains the page-reference squeuence

Example: `$ ./simulate FIFO 20 pageref.txt`

Input File Format

A sample input file is available on Canvas. Your program will be tested with files of the same format. The format of the input file is as follows:

- Each line has two fields, separated by whitespace
- The first field is either "R" or "W", meaning a read or a write to the page, respectively
  - These letters will always be given in uppercase
- The second field is the virtual page number
  - You may assume the page number fits comfortably in an `unsigned char`
- The file will contain no more than $2^{16} - 1$ lines (i.e. it can be indexed with a `short`)
- You may assume:
  - You will be given a valid file name
  - The file does not contain blank lines
  - All lines are correctly formatted

Output

The program should generate the following statistics:

1. The total number of page references
2. The total number of page misses
3. The total number of time units for page misses
4. The total number of time units for writing back the dirty (modified) page

The number of time units expended per page access is found as follows:

- Pages hits consume 0 time units
- Loading a page in after a miss requires 5 time units
- If a modified page is evicted, writing it out requires 10 time units
- Thus, a page miss which evicts a dirty page has a total cost of 15 units

Output Format

Output must be formatted as follows to be understood correctly by Autolab:

- Any line of text beginning with a pound / hash (#) will be ignored
- There should be exactly four lines with are not ignored
- These lines must correspond to the statistics given in the previous section *in that order*
  - Autolab will be looking for strings of arbitrary text ending in a number
    - Something like this regular expression: `/.*(\d+)$/`
  - Make sure the part of the line that Autolab ignores is still useful for a person

Bonus Tasks

You may choose to implemented additional algorithms for some extra credit. Each algorithm is worth 10 points.

1. Implement Optimal page replacement (aka MIN)
   - Algorithm is specified at the command line with OPT
2. Implement Clock algorithm
   - Specified at the command line with CLK
   - Implement a single-hand approach

Submission

Submission will be *via Autolab* (`https://autolab.encs.vancouver.wsu.edu/`). Turn in a single *zip file* containing:

- A bunch of .c files (with, optionally, some headers)
- A Makefile to compile your source
  - You must compile successfully with the -Wall and -Werror flags
  - You must include:
    - A default rule which compiles your program without running it
    - A `run` rule which runs your program *without arguments*
    - a `clean` rule which removes any intermediate files and the executable

Grading

Autolab will handle primary grading for this assignment. This includes functionality testing as well as testing for memory errors with Valgrind. We will also manually examine code for good style.