Lorenzo Zafra – 1395521
CMPUT 379 Assignment 4 Project Report

Objectives:

The objective of the assignment is to learn how to develop a simulator program and the quirks that comes with such programs. This assignment specifically focuses a simulator for page fault algorithms. In addition, this assignment helps us understand the different types of page replacement algorithms and which one is the best algorithm in certain types of cases.

Design Overview:

- .c files – a4vmsim.c, strategy.c, stack.c, pagetable.c
  - **strategy**.c contains all the page fault algorithms. The functions inside the file is the handler for each strategy
  - **stack**.c contains all the functions and functionality of a stack that is needed by the algorithms.
  - **pagetable**.c creates a pagetable for the simulator.
  - **a4vmsim.c** the main program which initializes the needed data structures and runs the simulation.
- I created a stack for the following uses:
  - LRU – keeps track of all the page numbers. Least recently used will always be at the tail if the least recently referenced is moved to the top.
  - MRAND – keeps track of the last k=3 referenced pages
  - SEC – instead of implementing a queue, I used the stack as a queue by popping from the tail (dequeue for a queue)
- Created an enum for all the error codes that a user may encounter.
- Created a struct array for the pagetable. This struct contains info such as virtual address, physical address, valid bit, reference bit and modified bit.
- Created an array to emulate the physical memory. This array just holds the virtual address of the page.

Assumptions:

- Simulated system is a 32-byte system.
- System is little endian

Project Status:

At time of submission, the assignment is working as specified. I had the biggest trouble playing with pointers and struct arrays. I found that I was getting a lot of segmentation faults but that was fixed.

I also had issues at the start reading the bits from STDIN. I realized after that I was interpreting the bytes after its read backwards.

More issues came up when implementing the algorithms. At first, I thought that I would only need the stack for the LRU algorithm but after having troubles with implementing a queue for the second chance algorithm, I upgraded my stack and gave it more functions so that I could emulate it as a queue.

Testing and Results:

Testing the reading of bytes from the mrefgen program was done by concatenating bits and printing them out. I compared them using the “od” command to make sure that the bits are being read correctly.

To test the various algorithms, I first made sure that the basic functionality is working. i.e., I made sure that the operation from the reference string is being read correctly and that the write count and accumulator final value are both correct. After I verified its correctness, I unit tested my stack to make sure that it is working as intended as it will be used for all of the strategies in my design.

To test that the selection of page faults is working, I changed various parameters in the mrefgen program and ran all strategies to compare which is efficient in such situations. In general, the LRU replacement algorithm should be more efficient than MRAND which is what I got from the tests. Also, by increasing the page size, I expected that the number of page faults should decrease which is the result from my testing.

Also by increasing the ratio of writes, I should expect a lot of flushes which is what I got from testing.

Additionally, by increasing the locality and focus, I should except the algorithms to have much less page faults.

Acknowledgements:

- APUE 3rd Edition
- https://linux.die.net/
- http://www.geeksforgeeks.org/stack-data-structure-introduction-program/
- https://en.wikipedia.org/wiki/Page_replacement_algorithm
- https://stackoverflow.com/questions/3638431/determine-if-an-int-is-a-power-of-2-or-not-in-a-single-line
- https://stackoverflow.com/questions/3407012/c-rounding-up-to-the-nearest-multiple-of-a-number

How to Run:

Compile both mrefgen.c and a4vmsim.c. a4vmsim can be compiled by using the makefile & running “make” and mrefgen.c can be compiled using:

```
#  Compilation: gcc -o mrefgen mrefgen.c -lm
```

Then, there are two ways:

1. Generate synthetic reference strings by

   ./mrefgen -m memsize -l [0-9] -f [0-9] -w <0-1> -n numref > [filename]

2. Run the simulator and feed it the generated reference string

   ./a4vmsim pagesize memsize strategy < [filename from step1]

Or we can combine both of these steps

./mrefgen -m memsize -l [0-9] -f [0-9] -w <0-1> -n numref > | ./a4vmsim pagesize memsize strategy