ASSIGNMENT #5

Virtual Memory Simulation

CSCI 480 100 points + 15 bonus Spring 2024

Check Blackboard for due date. Due by 11:59 PM.

Write a program in C++ or C which simulates a paging-based virtual memory system.

Specification:

As we know, the virtual memory mechanism called demand paging divides the virtual memory into evenly sized pages and the real physical memory into frames of the same size. The page table does the address mapping, which contains one entry for each virtual address page. Each entry has at least two fields: a Boolean flag indicating whether there is physical memory present for that virtual page, and the frame id of physical memory for the virtual page if the frame is present. The number of entries in the page table is equal to the number of virtual pages in the address range of the system.

In this assignment, you assume the page size (frame size) is 1K and the system has 9K virtual memory. You can hardcode the page size and virtual memory size. By default, you can also assume that the system has 4K physical memory, but this should be a variable and your assignment will be written generically for different physical memory sizes (it will be an argument from the command line as explained later).

Assume a program (that does some tax calculation) consisting of a driver block of code (main()), 5 functions, and 3 blocks of memory containing arrays of numbers. Each of these units will be assigned to a specific virtual page. Each function or an array is 1K (1 page in size). Their starting virtual addresses are provided as below:

Table 1: The virtual memory layout for the program.

Data	Starting Virtual addresses/Virtual page number
net[]	7k/7
activity[]	6k/6
holding[]	5k/5
print()	4k/4
update()	3k/3
tax()	2k/2
load()	1k/1
main()	0k/0

The program is about loading some data for tax calculation. The flow goes like this:

m; l; h; m; a; n; m; u; h; n; a; m; t; n; m; p; n;

where *m* is for main(), *l* is for load() etc. Since we assume each unit is one page, the above is essentially the page reference string.

This program will be loaded and run by your assignment in your simulated virtual memory.

Detailed assignment specification:

- The driver program executes the flow. It takes in a command line argument for number of frames. The default argument is 4. --- It is provided with the assignment. You can change it although likely not necessary.
- The algorithm will be *pure demand paging*, which means the physical memory was initially empty and the frames fill from 0 up as the program executes.
- For each page reference, the program will try to access the page based on the page table. Depending on if it is already in the memory, you would need to decide if there is a page fault. (Note that in the case of page fault, the page will be accessed after being swapped in.)
- When the physical memory is full, you need to implement a page replacement algorithm. You will code the First in First Out (FIFO) algorithm. You need to swap in the page to replace one that is not currently used by the above algorithms. You do *not* need to consider the possibility of first swapping out in this assignment. After page replacement, you need to update the page table correspondingly by making the original mapping for the frame invalid, and making the new mapping valid.
- You will print out some information in the console when there is a page fault. At the end, you print out the layout of page table and the physical memory, as well as the total number of page faults (See example output).
- Change the physical memory size from 4K to 5K by changing the command line argument from 4 to 5. Run the program again.

Bonus Credit (15 points):

Implement the Least Recently Used (LRU) for page replacement. Print out the same required info as above (memory layout for 4K physical memory and 5K physical memory). ~~This will be submitted via a different submission link in Blackboard.~~ Bonus credit does not allow late submission. You can code the logic of FIFO and LRU in the same program and just change some variables/parameters for different algorithms.

Output (FIFO with 4 frames):

```
Welcome!

FIFO with 4 physical frames

PageTable: page fault occurred

Physical: Swap In: main

Physical: Accessed frameID: 0 contains: main


PageTable: page fault occurred

Physical: Swap In: load

Physical: Accessed frameID: 1 contains: load


PageTable: page fault occurred

Physical: Swap In: holding

Physical: Accessed frameID: 2 contains: holding


Physical: Accessed frameID: 0 contains: main


PageTable: page fault occurred
```

```
Physical: Swap In: activity
Physical: Accessed frameID: 3 contains: activity

PageTable: page fault occurred
Physical: Swap In: net
Physical: Accessed frameID: 0 contains: net

PageTable: page fault occurred
Physical: Swap In: main
Physical: Accessed frameID: 1 contains: main

PageTable: page fault occurred
Physical: Swap In: update
Physical: Accessed frameID: 2 contains: update

PageTable: page fault occurred
Physical: Swap In: holding
Physical: Accessed frameID: 3 contains: holding

Physical: Accessed frameID: 0 contains: net

PageTable: page fault occurred
Physical: Swap In: activity
Physical: Accessed frameID: 0 contains: activity

Physical: Accessed frameID: 1 contains: main

PageTable: page fault occurred
Physical: Swap In: tax
Physical: Accessed frameID: 1 contains: tax

PageTable: page fault occurred
Physical: Swap In: net
Physical: Accessed frameID: 2 contains: net

PageTable: page fault occurred
Physical: Swap In: main
Physical: Accessed frameID: 3 contains: main

PageTable: page fault occurred
Physical: Swap In: print
Physical: Accessed frameID: 0 contains: print

Physical: Accessed frameID: 2 contains: net

Now print snapshots:

Physical Memory Layout:
Frame: 0 contains: print
```

```
Frame: 1 contains: tax
Frame: 2 contains: net
Frame: 3 contains: main

Page Table Snapshot:
Page Index: 0 : Physical Frame Index: 3 : In Use: true
Page Index: 1 : Physical Frame Index: -1 : In Use: false
Page Index: 2 : Physical Frame Index: 1 : In Use: true
Page Index: 3 : Physical Frame Index: -1 : In Use: false
Page Index: 4 : Physical Frame Index: 0 : In Use: true
Page Index: 5 : Physical Frame Index: -1 : In Use: false
Page Index: 6 : Physical Frame Index: -1 : In Use: false
Page Index: 7 : Physical Frame Index: 2 : In Use: true
PageTable: Current number of page faults: 13
```

Programming Hints: (See a different doc)

Administration:

Submission is the same as before: e.g. you will submit a compressed file through Blackboard. See details from previous assignments on how to make and name the compressed file.

~~Extra credit needs to be submitted via a different submission link in Blackboard.~~

In your Makefile, if you do not extra credit, you only need one executable output “your-zid_project5”. On the other hand, if you do the extra credit, then you need to make sure your compilation produces TWO executable files called “your-zid_project5” and “your-zid_project5_p2”. For a student with z1234567 as their zid, the executable would be *z1234567_project5 and z1234567_project5_p2*.

Note that the directory must be called: “z1234567_project5_dir”, all in lowercase.

Notify your TA that you’ve done the extra credit part by adding a line of text when submitting your assignment in Blackboard (“I did extra credit.”) so that s/he will do the corresponding testing and grading.