--------------------------------------------------------------------------

+-------------------------+

| CS 450 |

| PROJECT: PT SIM |

| DESIGN DOCUMENT |

+-------------------------+

---- GROUP ----

**Jordan Edginton <edgintoj@sonoma.edu>**

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission or

>> bug notes, please give them here.

>> Please cite any offline or online sources you consulted while

>> preparing your submission, other than man pages, course

>> text, lecture notes, and course staff.

**Software Used: Jetbrains Clion C++ 20**

**I ended up giving up on the toggle and created 2 different main.cpp and page table class files (hence why there are 2 of each, 1 for a and 1 for b). All functions work as expected.**

**The only outside source I used was for the <bitset> library to make the conversion from decimal to binary easier. Other than that, I used no outside sources.**

PAGE TABLE SIMULATOR

====================

---- EXPLORE THE PROBLEM ----

>> A1: Given the following description of a page table:

>>

>> 7 8 32

>> 0 1 0 0

>> 1 1 4 0

>> 1 1 5 1

>> 1 1 2 0

>>

>> Translate the following sequence of address requests using the method

>> described for Part A.

>>

>> 0x05 **Disk**

>> 0x7F **SEGFAULT**

>> 0x3B **162**

>> 0x7F **SEGFAULT**

>> 0x40 **160**

>>

>> A2: It is ok if your program just reads every row of the input file,

>> but it is possible to compute how many rows you might expect.

>> Show a computation to determine how many rows are in the page table

>> using the first row of the input file shows above: 7 8 32

**Size of blocks / bytes = 2^5 / 2^3 = 2^2 = 4**

---- DATA STRUCTURES ----

>> A3: Copy here the declaration of each new or changed `struct',

>> `struct' member, global or static variable, `typedef', or enumeration.

>> Identify the purpose of each in 2--25 words.

>> Recall the instructions required at least one data structure.

**class Page\_table{**

**...**

**private:**

**vector<int> valid // our valid column of the page table   
vector<int> perm // our permissions column of the page table  
vector<int> frame // our frame column of the page table  
vector<int> used // our used column of the page table**

**p\_size // physical memory size**

**v\_size // virtual memory size**

**t\_size // table size**

**offset // how many bits is the offset**

**}**

---- FUNCTIONS ----

>> A4: Provide a prototype and documentation for each function

>> you have introduced to support this portion of the project.

>> Use the Google Style Guide for function documentation.

>> Recall the instructions required at least two functions

>> in your project, and these should be reflected in A4 and/or B3.

**Main.c:**

**vector<int> read\_input(); // reads input from the redirected file from the user via multiple cin statements**

**Page\_table:**

**public:**

**void read\_table(char\* &input) // reads the table from the file labeled in the command line argument**

**void simulate\_a(vector<int> inp) // simulates the page table using a vector created from user inputs (read\_input())**

**private:**

**int bin\_to\_int(string bin); // converts a binary string to an integer, used for printing the addresses**

---- ALGORITHMS ----

>> A5: Describe your general strategy for managing bit-wise

>> transformations of data, and relevant support functions you used

>> to accomplish this.

**When it came to bit-wise operations I created my own functions for converting: binary to decimal (bin\_to\_int()),binary to hexadecimal (bin\_to\_hex()), and hexadecimal to decimal(hex\_to\_decimal()).**

**For decimal to binary, I used the <bitset> library. The library allowed me to convert an integer into a binary string, which allowed me to do all of the bitwise transformations easier. I couldn’t figure out how to change it back though… so I just created my own conversion functions to do so.**

CLOCK REPLACEMENT SIMULATOR

===========================

---- EXPLORE THE PROBLEM ----

>> B1: Given the following description of a page table:

>>

>> 7 8 32

>> 0 1 0 0

>> 1 1 4 0

>> 1 1 5 1

>> 1 1 2 0

>>

>> Translate the following sequence of address requests using the method

>> described for Part B.

>>

>> 0x05 **PAGEFAULT** **133**

>> 0x7F **SEGFAULT**

>> 0x3B **162**

>> 0x7F **SEGFAULT**

>> 0x40 **160**

>>

---- DATA STRUCTURES ----

>> B2: Copy here the declaration of each new or changed `struct',

>> `struct' member, global or static variable, `typedef', or enumeration.

>> Identify the purpose of each in 2--25 words.

>> Do not repeat anything already described in A3.

**Everything in class Page\_tableb is the same as Page\_table, except for the name of simuate\_a() has been changed to simulate\_b().**

---- FUNCTIONS ----

>> B3: Provide a prototype and documentation for each function

>> you have introduced to support this portion of the project.

>> Use the Google Style Guide for function documentation.

>> Do not repeat anything already described in A4.

**Everything in class Page\_tableb is the same as Page\_table, with the exception of one small part of simulate\_b() that changes the DISK output to the PAGEFAULT output and implements second chance replacement.**

---- ALGORITHMS ----

>> B4: Describe (i) the data structure you used to search through the frames

>> following the clock rotation, and (ii) reason through the number of bits

>> you would need if you were using a space-efficient representation

>> (in particular, describe how might implement a row of the table in C).

**In simulate\_b() I used a variable *int clok* to keep track of the current clock position within the table. After every Pagefault I would check the used vector to see if the number there is greater than 0. If so, we subtract 1 from used and add 1 to clok. When a 0 is found, we print PAGEFAULT and calculate the address. If clok reaches 32 we reset it to 0.**

**I kept track of my rows as indexes in multiple vectors, with each vector representing a column (frame, valid, perm, and used). I think if I were to implement this in C I would do it the same way, with the only difference being calculating the size needed for the array before filling the array.**

---- RATIONALE ----

>> B5: Did you need to handle any ambiguous scenarios or corner cases

>> for the Clock algorithm, left unspecified in the algorithm's

>> description? For example. how does your program behave when

>> there is a page table and no valid entries to evict?

>> Explain any judgements you used in implementing

>> unclear or unspecified behavior.

**I will admit that I did not have the handling for a page with no valid entries (but why would you have permissions not set to 0 if it isn’t valid)…**

**NOW: When there are no valid entries one of two things will happen: if all have permissions at 0, it will print as many SEGFAULT’s as there are inputs. If there is one that doesn’t have permissions at 0, then after it iterates through the entire table it will print an error and exit.**

SURVEY QUESTIONS

================

Answering these questions is optional, but it will help us improve the

course in future quarters. Feel free to tell us anything you

want--these questions are just to spur your thoughts. You may also

choose to respond anonymously in the course evaluations at the end of

the quarter.

>> In your opinion, was this assignment, or any one of the problems

>> in it, too easy or too hard? Did it take too long or too little time?

>> Did you find that working on a particular part of the assignment gave

>> you greater insight into some aspect of OS design?

>> Is there some particular fact or hint we should give students in

>> future quarters to help them solve the problems? Conversely, did you

>> find any of our guidance to be misleading?

>> Any other comments?

**Extra credit: do this with the LRU algorithm instead of 2nd chance.**