CIS*2500: Assignment 3

Due April 1, 9:00 A.M.

The Big Picture

Computer simulations are used in everything from gaming to product testing. A simulation models a situation in software and then the program 'simulates' what would happen in real life given the boundaries set for the simulation. For example, traffic flow on streets and highways can be simulated to determine how to time the traffic lights, spread of contaminants and chemical weapons can be simulated to facilitate disaster planning, ecosystems can be simulated to predict the effect of human intervention (i.e. hunting or forestry) on the wildlife populations.

For this assignment, you will write a simulation for a movie theatre ticket line. The simulation will calculate information about how long it takes to get people seated for the movie.

Your program will read data in from data files and then construct simulated lines of ticket buyers. Each buyer will take a different amount of time to make purchases. Some of the ticket buyers will also want to purchase snacks. Some buyers may only wish to purchase snacks. Your program will correctly calculate the amount of time each buyer spends in line (both ticket line and snack line) and will be able to correctly print the list of movie attendees in the order in which they enter the theatre. (We will assume that all movie-goers stand patiently in line and do not jump lines or push in front of anyone).

Buyers will start in one of two ticket line ups and then will move to either the snack line or the seating line after ticket purchase. Snack purchasers will move to the seating line after their snack purchase.

Musts-You must have these working to get a mark greater than zero

- You must have your code in the correct folders with a working makefile and README

- You must have a list testing program and a linked list module
- You must read in at least one of the ticket-purchase lines into a list and be able to print it out again.
- You must use your linked list.
- You must use dynamically allocated memory properly.
- Your program must compile without errors or warnings on the SOCS computers.
- Your list testing program must start without segfaulting.

Expectations for the programs (you will write two programs and a linked list module)

- Linked List Module
  - Your linked list module will be tested using a program that Judi writes
  - the program will test every operation of your list and will try to find ways to break your linked list by sending null values, wrong values, etc.
  - program the linked list defensively
- Linked List Testing
  - Linked list implements all of the list functionality
  - tests boundary conditions for each function in the list module
  - prints out information about what is being tested as it runs
- Simulation
  - Accepts two files names via argv and argc and uses those files to create two lines of ticket buyers
  - Correctly simulates the flow of ticket buyers through the two ticket lines so that a buyer stands in line for the correct number of seconds and then moves to the appropriate next line (snacks or seating)
  - The simulation clock should start at 0 seconds. The two ticket lines must be processed in parallel for this simulation (but you should not spawn different threads or processes).
  - Keeps track of elapsed time for every ticket buyer.
  - Writes updates to the screen so that the user knows what is going on

- Calculates statistics at the end of the simulation to show maximum time in line, minimum time in line, and average time in line (not counting the seating line)
- Prints a nicely formatted report file called report.txt that gives the details about each ticket buyer and the statistics. The report file must be written to the docs folder of the assignment. See below for more information about the format of that file.

Expectations for your code

- Submission folder has src/ include/ and bin/ subdirectories that are used properly
- Source code is divided properly into .c and .h files
- Each .c file has the required header (see the policies document)
- Source code is properly indented
- Comments about function input/output and purpose are in .h files
- Comments about algorithm logic are in .c files
- Variable and function names are meaningful and are in camelCase
- A plain text file called README is in the root folder. It contains information about running and using your program as well as any known limitations of the program.
- You have a makefile that will compile your code and place the executable in the bin folder.
- Your code must compile with no error or warning messages using the -ansi and -Wall flags in gcc
- Your makefile has multiple targets that can compile either of your two programs
- Your function names match the function names given in the original .h file
- You use a struct as the node in your linked list module
- You use the linked list effectively to represent the four different lines of patrons.

Details about data files

- There will be two data files for this assignment, each containing the information for one ticket line.

- A single line of text in the file represents one movie-goer. The fields in the file are enumerated below.
    1. a patron ID. will be 6 alphanumeric characters.
    2. The amount of time (in seconds) that the patron will take to make a purchase. This time will be the same for ticket purchases and food purchases.
    3. a boolean that indicates whether the patron wishes to purchase snacks (0 == no, 1 == yes).
    4. a boolean that indicates whether the patron wishes to attend the movie (0 == no, 1 == yes). Patrons who do not attend the movie but who wish to purchase snacks will take up time in the snack line but will not join the seating line up.
- You need do only minimal error checking for input errors. The input file used for grading will not contain intentional errors.

Program Output

- Your program must write a file called report.txt in the docs subfolder of your assignment directory.
- All times are reported in hh:mm:ss format
- The first three lines of the file must contain the statistics about the simulation run. They must be formatted exactly as shown.
    1. Max:maxTimeHere
    2. Min:minTimeHere
    3. Avg:averageTimeHere
- The remaining lines of the file must be a list of each patron in the seating line, in the order that the patrons arrived in the line.
- Each line must contain a single patron-string followed by a newline (no spaces).
- a patron-string consists of the Patron ID followed by a colon followed by the total time that patron spent in line in hh:mm:ss

- For example the patron-string *45BAWF:00:15:45* shows that the patron spent zero hours, 15 minutes and 45 seconds in line. Time spent in the seating line is not counted in the total.

Extras that are worth bonus marks

- Add a command line flag to your simulation to tell it to read all of the patrons from a single file. If the flag is set, have your simulation place Patrons in the most advantageous ticket line to try to minimize the time spent in line.
- Raspberry Pi people: Provide some kind of gui to visualize the simulation. You could use ncurses or gtk or even tkl if you wanted.
- make graphs or charts from the output of your progam by writing a nicely formatted output file for R and calling R from within your program. Write the charts to file.
- Raspberry Pi people: Keep a journal or log as you work on the pi. Detail everything you needed to install on it to do the assignment. Document any difficulties you had as well as any really cool things you found out about the Pi.

Deliverables: Things you must hand in

1. (via git) your A3 folder with all required subfolders, c files and header files, README and Makefile
2. (via bucky) your collaboration statement
3. your test file with the data you used to test your program

Notes

1. Assignments that do not compile will be given a grade of zero
2. Assignments that compile with warnings may be given a grade of zero
3. Assignments will be compiled and marked using a linux terminal (same configuration as the Thornborough lab). Your code must compile and run correctly on that configuration.
4. Assignment three will be partially graded with an automated script. If you fail to follow the format for writing the text file, the script will fail you even if you have the right answer.
5. All work in this course is to be done independently. Submissions will be examined electronically for similarity.

Grading Procedures

Grading Guidelines TBA