CAB 403

REPORT FOR ASSIGNMENT

Arik Intneam Mir

n9637567

Instruction:

The Client.c should be compiled using "**gcc -o Client Client.c** ", and the Server.c should be compiled by "**gcc -o Server Server.c -lpthread**"

To run the game, the Server should be run first. The server only takes port number as its parameter. But, if the user doesn't input any port number, the default port number 12345 is used.

And the client takes two parameters to run. These are ip address and Server's port number.

Task 1:

The entire playfield of the client-server based Minesweeper game was implemented in the "Client.c" file.

Client.c

- Firstly the initial stage of this game is to make sure that the Client connects with the Server. The Client takes hostname and the port number. So, for this the in the **int main()** function, the client creates a socket to connect to the Server. If the connection between the Client and the Server is successfully established, the **void introGame()** function starts working.
- In the **void introGame()** function, a welcome message shows up and tells the user to authenticate by entering his username and password.
- **int authenticateUser()** takes the username and the password for an user. And the user is only valid if, the entered username matches the information from the *Auhtentication.txt* file, which is located on the server. And it is checked by taking **int sockfd** as a parameter**,** which crosschecks with the server to see if its correct. If the user name is not authenticated, the socket closes down and tells the user that the Client is disconnecting from the Server.
- When the user is successfully authenticated, the **void gameEntry()** function begins to run. The **void gameEntry()** function gives the user 3 options, which are dealt via switch-case. 1st option is "Play Minesweeper", which then leads to run the **void startGame()** function. 2nd case option is "show Leaderboard", and this will take the user to the leaderboard of the game via the function **void showLeaderboard()** function, which intakes the parameter **sockfd,** and gets the results for the leaderboard from the Server. And if the 3rd case option is chosen, the game just quits.
- If the user has chosen 1st option and decided to play the game, the **void startGame()** function begins to work and **place_mine()** function is used for randomly placing 10

mines in the playfield, which are initially hidden from the user in the initial stage. The tiles are generated randomly using a do-while loop so that the one tile doesn't gets created at the same place as of another existing tile. In the **void startGame(),** at the same time, the clock for calculating the game's time is shown by using **clock_t** type function **clock().**

- **void printBoard()** is used for implementing the interface which comes to display after a new game has started. The interface shows the letters from A to I on the left side in a row basis and the numbers, 1 to 9 on a column basis. This function is called up in the **void startGame() function.**
- Below the interface of the game, the user has 3 options to function the game. The user can either Reveal a tile by using "R", put a flag using "P" or quit the game using "Q".
- If the user decides to reveal a flag using "R", the user will be asked to enter the tile coordinate, which takes a two values; an alphabet from the left side row and a number from the top column. If the same coordinates has been entered before in the game, the user gets an error message and asked to input new tile coordinates. If the coordinates entered by the user matches with one of the 10 hidden mine's coordinates in the playfield, the game is over and the user gets a message notifying him this information. The user can place a flag on any coordinates he wants by using "P", and the flags are represented on the playfield using "+" sign.
- Every time the player places a flag on the playfield, the **winner()** function is called to check if player has inserted all flags successfully without revealing any mines. If the player manages to open all the coordinates without revealing any mines, and when there are no mines left, the player is declared as a winner. And a message stating that the player has won is shown up afterwards along with the amount of time it took the player to win the game.
- The amount of time it took the player to play one set of game is calculated using **double duration**, which subtracts the present time in the game finishing stage **clock()** from the variable **start.**
- The **void find_adjacent()** function is used for finding adjacent tiles with mines hidden in it's coordinates. This function checks whether a tile has mine in it or not. If it doesn't have a mine in it and has non-zero number of adjacent tiles, the tile is revealed. And if there are no adjacent mines hidden around a tile, the function checks again until it finds a tile with a non-zero number of adjacent tiles by checking the 8 tiles surrounding it.
- When the user inputs a coordinate in the playfield, the **void expand()** function also reveals the other surrounding tiles, which doesn't contain mine in them. The helper functions **enqueue()** and **dequeue()** are used too for this functionality.
- The **bool complete()** checks on the completion of one round of the game. It only becomes true, only when all the tiles in the playfield has been revealed along with any mines been revealed with the game been played throughout successfully.
-

Server.c

- The Server establishes connection with the Client in the **main()** function. **int sock_fd** is created for listening for connection requests from the Client and **int new_fd** is used when a new Client wants to connect with the Server. The default port number for the Server is introduced as **MYPORT** with the value of 12345.
- After a successful connection is established between the Server and a Client, the **bool authenticateUser()** function is to verify if the username and password of a given user matches that of the texts from Authentication.txt. A **bool** variable **authenticated** is introduced, which returns false if the username and password is invalid.
- The algorithm of the leaderboard is also implemented in the Server. For the functionality, a **struct leaderboard** was declared with 4 variables- **char name[50], int gameTime, int gameWon** and **int gamePlayed.**
- The leaderboard was implemented in the function **void inputIntoLeaderboard(),** which uses the elements of **struct leaderboard** through a linked list to input data into the leaderboard. The leaderboard is displayed to using the function **void displayLeaderboard().**
- The function **void bsortDesc()** is used to show the winners name and their scores in a descending order.

Task 2:

- To create concurrent connection between multiple clients and the server, multiple processes are created, with one process for each client. The **authenticateUser()** function is called here to check the client id with the given username and password from the Authentication.txt.

Lackings:

- When a player wins, the is shown in the leaderboard, but it is not saved.
- When the game is run on a Linux environment, the game works, but the user authentication part of the game doesn't seem to work. But, if the authentication part is erased from the game, the game seems to work fine without it. Whereas, the game works perfectly fine along with the user authentication in a Mac environment.
- Task 3 hasn't been implemented.

- In Task 2, multithread wasn't implemented, and the critical section problem wasn't handled either.
- Though the leaderboard has been implemented in codes, but it couldn't be connected with the server as the data couldn't be passed.