**ECE 5650 – Project 1**

Developing an Online Quiz System Using Low-Level Socket Programming



**By:**

David Baron-Vega – GF7068

Brian McCoy – HB5610

Enesia Hamza – GP8542

Hadi Yassine – GV8096

**Source Codes**

***Server Code:***

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

***Client Code:***

***A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated***

***Cross-functional Communication Code:***

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

**Procedure**

The provided code implements an online client-server quiz system. The system allows students and instructors to interact with the server through sockets. The client code is responsible for connecting to the server, submitting user type (instructor or student), providing access credentials, and sending various commands, such as taking a quiz or viewing grades. The server, on the other hand, establishes a socket connection, authenticates users based on user’s IDs and passwords, and handles quiz-related functions, including shuffling questions, grading, and storing grades in a file.

**Client-Side Code:**

The client-side code establishes a socket connection to the server. It prompts the user to input whether they are an instructor or a student, sending this information to the server. The user then provides the access credentials which include an access ID and password. After authentication, the client can submit various commands to the server, such as taking a quiz or viewing grades. These commands are sent using the established socket connection.

**Server-Side Code:**

The server-side code sets up a socket to listen for incoming connections. It handles user authentication based on the predefined user IDs and passwords. After successful authentication, the server distinguishes between instructors and students. For students, it facilitates the quiz-taking process, including shuffling questions, receiving user answers, grading, and storing grades in a file. For instructors, it allows them to view grades stored in the file.

**Cross-functional Code:**

The cross-functional code provides the essential building blocks for the client and server functionalities. Various utility functions were defined to handle socket communication, client-server interaction, and authentication processes. This code includes the quiz development functions that define quiz-related tasks, including question shuffling, quiz taking, grading, and storing and retrieving grades in and from the file defined.

**\*\*\*** Below is a breakdown of the code, showing how the code runs, and explaining all the functions used and defined.

***Server Side***:

**Imports:**

* socket: Provides low-level networking interface.
* random: Used for generating random numbers.
* funs: A source file containing various functions used in the code.

**Initialization:**

* quizLength: Number of questions in the quiz.
* qQ and qA: The sets of quiz questions and answers defined in the “funs” source code.
* team & lead: Dictionaries containing user IDs and corresponding passwords for students and instructors.
* fileName: The name of the file where grades will be stored (gradeStored.txt).
* port: Obtains a port number using the “getPort” function in the “funs” source code.
* serverSocket: Establishes a server socket using the “establishServer” function in the “funs” source code.

**Main Loop:**

* The code enters an infinite loop (while True), waiting for client connections.
* Once a connection is established, it enters a nested loop (while access == False), prompting for user credentials until authentication is successful.
* Once authenticated, it checks the type of user (messageAccess) and enters corresponding loops for students ("s") or instructors ("i").

**Student Section (messageAccess = "s"):**

* If the user requests to take a quiz (takeQuiz), it shuffles the quiz questions, allows the user to answer them, grades the quiz, and stores the grade.
* The user can request their grade (getGrade), and it sends the grade back.
* The student user can exit the loop (exit).

**Instructor Section (messageAccess = "i"):**

* If the instructor requests to view grades (viewGrades), it sends the contents of the grade file (gradeStored.txt).
* The instructor user can also exit the loop (exit).

***Client Side***:

**Imports:**

* socket: Provides low-level networking interface.
* funs: A source file containing various functions used in the code.

**Initialization:**

* port: Obtains a port number using the ”getPort” function from the funs source file.
* clientSocket: Establishes a client socket using the “establishClient” function from the funs module.
* addy: This variable is set to the value of “clientSocket”. It is used as an address for communication.

**Authentication Loop:**

* The code enters an infinite loop (while True), prompting for the type of access (student or instructor) until a valid type is provided.
* The access type is obtained using the “submitTypeRequest” function from the funs source file.
* If the access type is not "s" (student) or "i" (instructor), the loop continues until a valid type is provided.
* If a valid type is provided, the code proceeds to submit credentials using the “submitCredentials” function.
* The access status is then obtained using the “mailbox” function and opened using the “openMail” function.
* If access is granted, the loop is exited.

**Student Section (accessType == "s"):**

* Once access is granted for a student, a command list is printed, and the code enters a loop for processing student commands.
* The student can take a quiz by answering true/false questions.
* The user answers questions interactively until three questions are answered or the user decides to exit.
* After completing the quiz, the user can request their grade.
* The student user can exit the loop.

**Instructor Section (accessType == "i"):**

* Once access is granted for an instructor, a command list is printed, and the code enters a loop for processing instructor commands.
* The instructor can view grades.
* The instructor user can exit the loop.

***Cross-functional Communication***:

**Socket Functions:**

* getPort: Returns a predefined port number.
* postMan: Sends a message (mail) through the specified socket (sock).
* mailBox: Receives a message from the specified socket and decodes it.
* openMail: Prints the received message.

**Client Functions:**

* establishClient: Creates a client socket and connects it to the specified server (localhost and the provided port).
* submitTypeRequest: Takes user input to determine if the client is an instructor or student and sends this information to the server.
* submitCredentials: Takes user input for access ID and password, sending this information to the server.
* submitChoice: Takes user input for various commands and sends them to the server.
* authentication: Checks if the provided user ID and password match the predefined team dictionary.

**Server Functions:**

* establishServer: Creates a server socket, binds it to a specific port, and listens for incoming connections.

**Quiz Development Functions:**

* quizQuestions: Returns a list of predefined quiz questions.
* quizAnswers: Returns a list of the predefined correct answers to the quiz questions.
* shuffler: Shuffles the order of quiz questions and answers.
* actuateQuiz: Sends quiz questions to the client, receives user answers, and returns the user's answers.
* grader: Grades the quiz based on user answers and returns the calculated grade.
* gradeWriter: Adds user ID and grade to a file.
* gradeReader: Reads and prints the contents of the grade file.

**Testing**

We ran the program on python 3.11 to check for any bugs in the code. The steps and results are shown below.

* Switch to the directory where the code files are located (serverProject folder).
* Connect the server side in one terminal and client side in another using the commands “python3 server.py” & “python3 client.py” respectively to their code files.

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer program

Description automatically generated

Once the server is connected and ready, we can launch the instructions on the client’s side. The program runs as follows:

1. The program prompts the user to type “i” for instructor or “s” for student.
2. It prompts the user to enter the access ID and password.
3. The program authenticates the inserted credentials.
4. If credentials are valid, the system prints a “granted” message.
5. The system prompts the user to insert a command ( takeQuiz or exit) from the list.
6. It prints the first question in the quiz and prompts the user to input an answer (T or F).
7. After answering all 3 quiz questions, the system prompts the user to insert an action (getGrade or exit) from the command list.
8. If “getGrade” command was inserted, the system calculates the grade and prints it indicating that the quiz is finished.
9. The system saves the grades in the file (gradeStored.txt) for instructor to view.

Note that at any point during the quiz, the user can type “exit” and the program terminates.

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**Completion & Self-Critique**

* The program meets all requirements:
  + The program supports only one quiz.
  + The quiz contents are stored on the server.
  + The points assigned to each quiz are divided equally among the quiz questions.
  + The quiz supports only true/false questions.
  + The user can take the quiz multiple times during the same session.
  + The quiz is not time limited.
  + All commands are case-insensitive.
  + The client and server programs check for the validity of inputs and take the proper actions.
  + The quiz data structure contains more questions than presented to a student.
  + The program runs in an infinite loop, awaiting user’s input and proceeds accordingly.
* The program runs correctly all the time without any issues.
* The program has been tested numerous times, and always runs smoothly implying the same procedure.
* The code is neat, organized, and well documented.