Report on Computer Networking Quiz Application

By: Mohamed Elsayed

A. Detailed Description of the Application

1. Overview

The project is a client-server application that facilitates a Computer Networking quiz. The system involves two main components:

- **Server:** Responsible for serving questions to clients, evaluating answers, and sending back the results.
- **Client:** Provides a graphical user interface (GUI) for users to interact with the quiz, select answers, and view the results.

2. Models, Frameworks, and Protocols Used

- **Socket Programming:** Utilized for communication between the client and server over TCP/IP.
- **Pandas Library (Server):** Used for reading and managing quiz questions from an Excel file.
- **Tkinter Library (Client):** Provides the GUI for user interaction.
- **Multithreading:** Enables the server to handle multiple clients simultaneously and allows the client to run a timer in the background.
- **JSON Serialization:** Facilitates the transfer of structured data (questions and answers) between the client and server.

3. Application Workflow

1. The server loads quiz questions from an Excel file and waits for client connections.
2. When a client connects, the server randomly selects five questions and sends them to the client.
3. The client displays these questions using a Tkinter-based GUI.
4. The user answers each question within a given time limit. The client sends the answers back to the server upon completion or timeout.
5. The server evaluates the answers and returns the score to the client.
6. The client displays the final result to the user.

B. Functions and Classes Description

Server-Side

Main Components

- **`questions_df = pd.read_excel("ExamQuestions.xlsx")`**
  - Reads quiz questions from an Excel file.
  - Stores questions and options in a Pandas DataFrame.
- **`questions = questions_df.to_dict(orient="records")`**
  - Converts the DataFrame to a list of dictionaries, making it easier to handle in the application.
- **`handle_client(Client_connection)`**
  - Handles client interactions in a separate thread.
  - Sends selected questions to the client.
  - Receives answers and evaluates them.
  - Sends the final score back to the client.
- **Multithreading**
  - Each client connection is handled in a separate thread to support concurrent users.

Client-Side

Main Components

- **GUI Setup**
  - `Tk()` creates the main window.
  - `Label`, `Radiobutton`, and `Button` widgets display the quiz questions and options.
- **`timer()` Function**
  - Runs a countdown timer for the quiz.
  - Ends the quiz and sends answers to the server if time runs out.
- **`update_ui(question_index)` Function**
  - Updates the displayed question and options for the user to answer.
- **`Send_Answers()` Function**
  - Sends the user's answers to the server.
  - Displays the result received from the server.
- **`next_question()` Function**
  - Saves the current answer and loads the next question.
  - Sends answers if the quiz is completed.

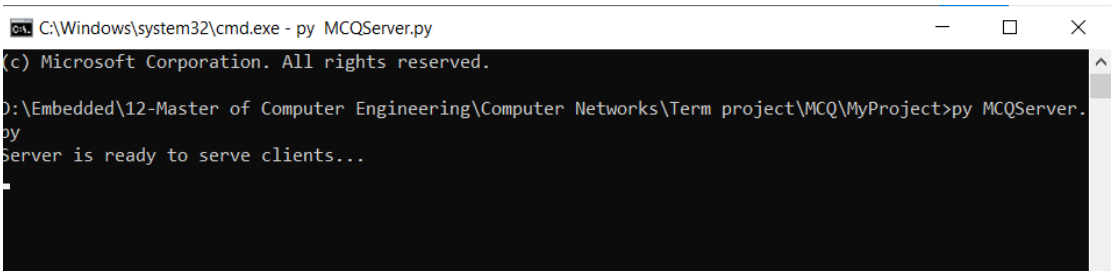
Networking

- **Server:**
  - Listens for connections using `socket.listen()`.
  - Uses `socket.accept()` to accept and handle client connections.
- **Client:**
  - Connects to the server using `socket.connect((HOST, PORT))`.
  - Receives questions and sends answers via JSON-encoded messages.

C. Screenshots of the Running Program

1. Server Terminal

- **Before Client Connection:** Displays a message waiting for client connections.



- **After Client Connection:** Logs the received answers and the calculated score.

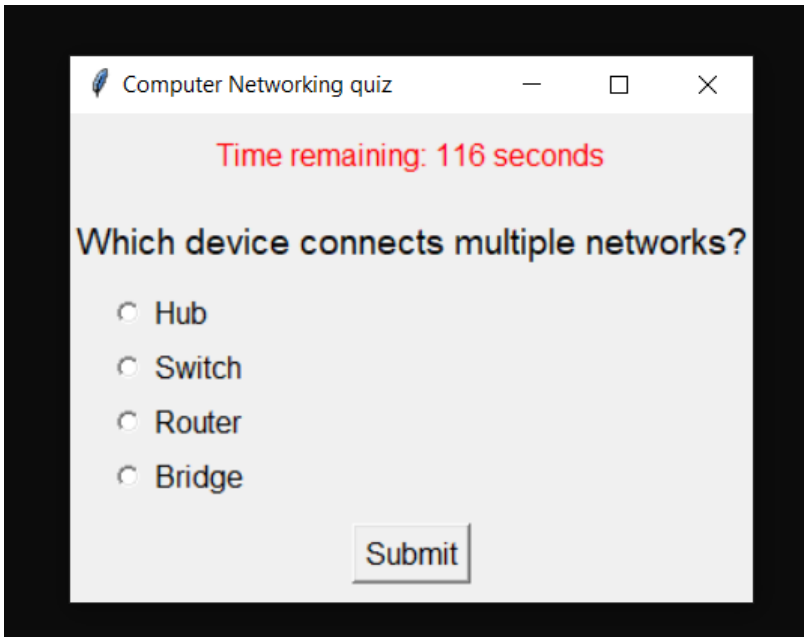
```
Server is ready to serve clients...
Recieved connection from client : ('127.0.0.1', 32569)
Recieved answers from ('127.0.0.1', 32569) are : ['B', 'B', 'D', 'C', 'C']
```

- **Results Sent:** Confirms the result sent to the client.

```
Recieved answers from ('127.0.0.1', 32569) are : ['B', 'B', 'D', 'C', 'C']
Sent result to ('127.0.0.1', 32569): {'score': 2, 'total': 5}
Connection with ('127.0.0.1', 32569) closed.
```

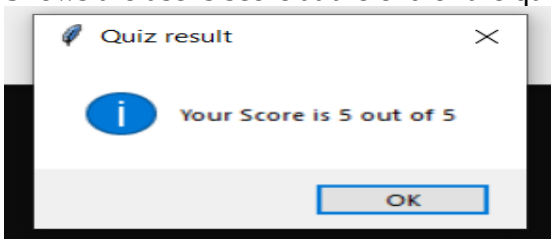
2. Client GUI

- **Quiz Interface:**
  - Displays a question with multiple-choice options.
  - Timer counts down in real-time.
  - "Submit" button submits the current answer.



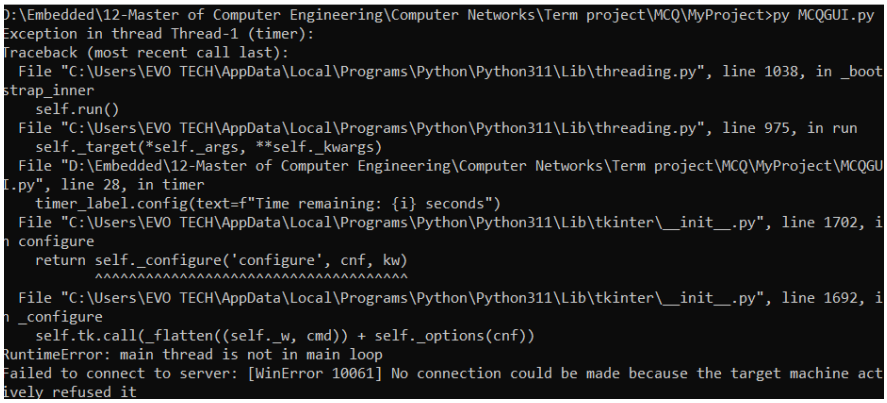
- **Result Popup:**

- Shows the user's score at the end of the quiz.



3. Client Terminal

- Logs messages such as:
  - Connection to the server.
  - Errors if the connection fails.



```
D:\Embedded\12-Master of Computer Engineering\Computer Networks\Term project\MCQ\MyProject>py MCQGUI.py
Exception in thread Thread-1 (timer):
Traceback (most recent call last):
  File "C:\Users\EVO TECH\AppData\Local\Programs\Python\Python311\Lib\threading.py", line 1038, in _boot
strap_inner
    self.run()
  File "C:\Users\EVO TECH\AppData\Local\Programs\Python\Python311\Lib\threading.py", line 975, in run
    self._target(*self._args, **self._kwargs)
  File "D:\Embedded\12-Master of Computer Engineering\Computer Networks\Term project\MCQ\MyProject\MCQGU
I.py", line 28, in timer
    timer_label.config(text=f"Time remaining: {i} seconds")
  File "C:\Users\EVO TECH\AppData\Local\Programs\Python\Python311\Lib\tkinter\__init__.py", line 1702, i
n configure
    return self._configure('configure', cnf, kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\EVO TECH\AppData\Local\Programs\Python\Python311\Lib\tkinter\__init__.py", line 1692, i
n _configure
    self.tk.call(_flatten((self._w, cmd)) + self._options(cnf))
RuntimeError: main thread is not in main loop
Failed to connect to server: [WinError 10061] No connection could be made because the target machine act
ively refused it
```

Conclusion

This project demonstrates a practical implementation of client-server communication using Python, incorporating GUI development and threading. The server efficiently handles multiple clients, while the client provides an intuitive interface for users to interact with the quiz.