**Explanation of the Server and Client Code**

**Checksum Explanation:**

The checksum is a simple error-checking mechanism. It ensures that the message transmitted between the client and server is received without alteration. The checksum is calculated as a 16-bit one's complement sum of the bytes in the message, and both the sender and receiver independently compute and compare it.

**Server Code Walkthrough**

**calculate\_checksum(message)**

* **Purpose**: This function computes a 16-bit one's complement checksum for the given message.
* **How it works**:
  1. The message is converted to bytes.
  2. The sum of these bytes is taken modulo 65536.
  3. The result is inverted (bitwise negation) to get the final checksum.
* **Usage**: This ensures that the server can verify if a message received from the client has been altered in transit.

**corrupt\_message(message)**

* **Purpose**: This function introduces random errors into a message.
* **How it works**:
  1. A random index in the message is chosen.
  2. The character at that index is replaced with a randomly chosen ASCII character (between 32 and 126).
* **Usage**: It simulates the possibility of transmission errors for testing error detection using the checksum.

**start\_server(host='0.0.0.0', port=65432)**

* **Purpose**: This is the main server function that handles the connection with the client.
* **Main Steps**:
  1. **Socket setup**: The server creates a TCP/IP socket (socket.AF\_INET, socket.SOCK\_STREAM), binds it to the provided host and port, and begins listening for incoming connections.
  2. **Connection handling**: Once a client connects, the server accepts the connection and the communication begins.

**Main server loop:**

* **Receiving data**:
  + The server continuously waits for incoming data from the client using conn.recv(1024).decode().
  + If no data is received (i.e., the client disconnected), the server breaks the loop and closes the connection.
* **Checksum validation**:
  + If the received data ends with 5 digits (data[-5:]), it implies that the message has an attached checksum.
  + The server extracts the message and received checksum, computes its own checksum, and compares the two.
  + If the checksums match, the server sends a success message, otherwise, it sends an error message.
* **Sending a response**:
  + After processing the client's message, the server reads user input from the console.
  + If the message is not empty or "QUIT", it sends the message with a checksum back to the client.
  + With a 40% probability, the message can be corrupted to simulate transmission errors.
* **Quit command**:
  + If the user types "QUIT", the server sends a "QUIT" message to the client and gracefully closes the connection.

**Closing the connection:**

* When the server detects "QUIT", or if the client disconnects, the connection is closed (conn.close()) and the server socket is also closed (server\_socket.close()).

**Client Code Walkthrough**

**calculate\_checksum(message)**

* The client uses the same checksum function as the server to compute the checksum for outgoing messages and validate incoming messages from the server.

**corrupt\_message(message)**

* The client also uses the same function as the server to simulate errors in its outgoing messages for testing error detection.

**start\_client(host, port)**

* **Purpose**: This is the main client function that handles the connection with the server.
* **Main Steps**:
  1. **Socket setup**: The client creates a TCP/IP socket and connects to the server.
  2. **Message exchange**: The client communicates with the server, sending and receiving messages.

**Main client loop:**

* **Sending the first message**:
  + The client asks the user for the first input (input("You (client): ")), computes its checksum, and sends the message along with the checksum to the server.
* **Receiving data from the server**:
  + The client waits for a response from the server using client\_socket.recv(1024).decode().
  + If the server disconnects, the client breaks the loop.
* **Checksum validation**:
  + If the received data ends with 5 digits, the client extracts the message and checksum, computes the checksum, and compares the two. Based on the comparison, it determines if the message was received correctly.
* **Sending a response**:
  + After processing the server's response, the client reads user input.
  + If the input is not empty or "QUIT", it sends the message to the server.
  + With a 40% probability, the message can be corrupted before sending.
* **Quit command**:
  + If the user types "QUIT", the client sends a "QUIT" message to the server and gracefully closes the connection.

**Closing the connection:**

* When the client detects "QUIT" from the server or the server disconnects, the connection is closed (client\_socket.close()).

**How to Execute the Code**

**1. Prerequisites**

* Ensure that Python is installed on your system (version 3.x).
* The socket and random libraries are part of the Python Standard Library, so no additional installations are required.

**2. Setting Up the Code**

You need to run both the **server** and **client** scripts. Since these scripts represent two different sides of a network communication system, they should be run in separate terminal windows or on separate machines (or even on the same machine with different terminal windows).

**Steps to Execute the Server Code**

**1. Create the Server File\***

**- Create a new Python file and name it something like `server.py`.**

**- Copy and paste the server code into this file.**

**2. Run the Server Code:**

**- Open a terminal window.**

**- Navigate to the directory where `server.py` is located.**

**- Run the server with the following command:**

**python server.py**

**- The server will start and begin listening for connections on the specified host (`0.0.0.0`) and port (`65432`).**

**You should see an output like:**

**Server listening on 0.0.0.0:65432**

**- Once a client connects, the server will display:**

**Connected by ('client\_ip\_address', client\_port\_number)**

**Steps to Execute the Client Code**

**1.Create the Client File:**

**- Create a new Python file and name it something like `client.py`.**

**- Copy and paste the client code into this file.**

**2. Run the Client Code:**

**- Open a \*\*new terminal window\*\* (while the server is running).**

**- Navigate to the directory where `client.py` is located.**

**- Run the client with the following command:**

**python client.py**

**- The client will ask you to input the server's IP address.**

**- If the server is running on your local machine, you can enter `localhost` or `127.0.0.1`.**

**Enter the server IP address: 127.0.0.1**

**- You can now start exchanging messages between the client and server.**

**Testing the Client-Server Communication**

**1. Client Sends a Message:**

**- After the client connects to the server, it will prompt you to input a message.**

**You (client): Hello Server**

**- The client will calculate a checksum, optionally corrupt the message with a probability, and send it to the server.**

**2. Server Receives the Message:**

**- The server receives the message, checks if it has been corrupted by comparing checksums, and sends a response back:**

**Client: Hello Server**

**- If there is a checksum mismatch due to corruption, the server will send an error message.**

**3. Server Sends a Message:**

**- The server will ask for input after receiving a message from the client.**

**You (server): Hi Client**

**- The server will compute a checksum for its message, and send it to the client with possible corruption.**

**4. Client Receives the Message:**

**- The client will validate the server's message based on the checksum and notify if the message is correct or corrupted.**

**---**

**Closing the Connection**

**To stop the communication:**

**1. Either the server or client can type `QUIT`.**

**You (client): QUIT**

**2. When `QUIT` is sent:**

**- The other side will receive the `QUIT` message.**

**- Both the client and server will terminate the connection and close their respective sockets.**

**You should see a message like:**

**Closing connection...**