TCP Client Script

The **TCP client script** creates a graphical chat client using Python's `socket`, `threading`, and `tkinter` modules. It connects to a TCP server on a specified IP and port, sends the user's chosen username, and enables real-time messaging. A background thread continuously listens for incoming messages from the server without freezing the GUI. Users can send messages via a text entry box, and all messages (sent and received) are displayed in a scrollable chat window. The script also handles connection errors and ensures the socket is properly closed when the user exits the application, ensuring smooth and responsive communication.

Here is a detailed explanation for the TCP client script of the multi-client chat application.

Imports and Constants

These are modules we imported for the TCP client script

```
import threading
import tkinter as tk
from socket import AF_INET, SOCK_STREAM, socket, timeout
from tkinter import messagebox, scrolledtext
```

- `socket`: For network communication over TCP.
- `AF_INET`: IPv4.
- `SOCK_STREAM`: TCP connection (reliable, stream-based).
- `timeout`: Used to avoid indefinite blocking when receiving messages.
- `tkinter`: For the GUI.
- `threading`: Enables receiving messages while the GUI remains responsive.

The following constants and global variables are used in the script.

```
# GUI constants
DARK_GREY = "#121212"
MEDIUM_GREY = "#1F1B24"
OCEAN_BLUE = "#464EB8"
WHITE = "white"
FONT = ("Helvetica", 17)
BUTTON_FONT = ("Helvetica", 15)
SMALL_FONT = ("Helvetica", 13)
```

GUI styling constants for consistent theming.

```
# Socket constants
SERVER_IP = "127.0.0.1"
SERVER_PORT = 65432
```

- **SERVER_IP = "127.0.0.1"**: The server runs on localhost (loopback address).
- **SERVER_PORT = 65432**: Port number for the server to listen on.

Socket Initialization

In the first few lines we created a TCP client socket. and set a timeout to ensure the `recv()` won't block forever, this is needed to keep the GUI from freezing during shutdown or error.

```
# Create the client socket object
client = socket(AF_INET, SOCK_STREAM)
client.settimeout(1)  # Set timeout to prevent blocking

stop_threads = False  # Global flag to stop threads
```

Displaying Messages in the GUI

The `display_message()` function is used to display incoming and outgoing messages in the GUI. It also temporarily sets the text box to writable state, displays the message, then disables it again (to prevent user editing).

```
def display_message(message: str):
    """Function to write messages on the GUI"""

    message_box.config(state=tk.NORMAL)
    message_box.insert(tk.END, message + "\n")
    message_box.config(state=tk.DISABLED)
```

Arguments:

- `message`: The message to be displayed.

Returns:

- Nothing.

Connecting to the server

The `connect_to_server()` function is used to connect with the server. In addition it also gets the username of the client from the GUI and sends that to the server.

```
def connect_to_server():
    """Function to connect with the server"""

    global stop_threads
    stop_threads = False  # Reset flag in case of reconnection

    try:
        client.connect((SERVER_IP, SERVER_PORT))
        display_message("server ~ Successfully connected to the server")
    except Exception as e:
        messagebox.showerror(
            title="Connection Error",
            message=f"Unable to connect to server at {SERVER_IP}:{SERVER_PORT}\n\n{e}",
        )
        return

    username = username_textbox.get()
    if username:
        client.sendall(username.encode("utf-8"))
        main_window.title(f"{username}")
    else:
        messagebox.showerror(title="Invalid username", message="Username cannot be empty")
        return

    threading.Thread(target=listen_for_messages_from_server, daemon=True).start()

    username_textbox.config(state=tk.DISABLED)
    username_button.config(state=tk.DISABLED)
```

Arguments:

- None (uses GUI inputs and global `client` socket)

Returns:

- None

Purpose:

- Initiates a connection to the server using `connect()`.
- Send the username (read from the GUI input).
- Starts a **thread** to listen for incoming messages.

Key Concepts:

```
client.connect((SERVER_IP, SERVER_PORT))
```

- Establishes a TCP connection to the server.
- If the server isn't running, an exception is caught, and a GUI error is shown.

```
client.sendall(username.encode("utf-8"))
```

- Sends the username to the server once connected.

- Encoding is necessary because sockets transmit bytes, not strings.

```
threading.Thread(target=listen_for_messages_from_server, daemon=True).start()
```

- Starts a background thread to listen for server messages without blocking the GUI.

Sending Messages

The `send_message()` function is used to send messages to the server, from the GUI.

```
def send_message():
    """Function for sending messages in the GUI"""

    message = message_textbox.get()
    if message:
        try:
            client.sendall(message.encode("utf-8"))
            message_textbox.delete(0, tk.END)
        except Exception as e:
            messagebox.showerror(title="Error", message=f"Failed to send message: {e}")
    else:
        messagebox.showerror(title="Empty message", message="Message cannot be empty")
```

Arguments:

- None

Returns:

- None

Purpose:

- Grabs the text from the input field and sends it to the server.

Listening for Messages

The `listen_messages_from_server()` function is used to continuously listen for messages from the server.

```
def listen_for_messages_from_server():
    """Function to listen for messages from the server"""

    global stop_threads
    while not stop_threads:
        try:
            message = client.recv(2048).decode("utf-8")
            if message:
                display_message(message)
        except timeout:
            continue  # Prevent blocking when stopping threads
        except Exception:
            break  # Exit thread when socket is closed
```

Arguments:

- None

Returns:

- None (runs indefinitely in a thread)

Purpose:

- Continuously listens for new messages from the server using `recv()`.

Key Concepts:

```
message = client.recv(2048).decode("utf-8")
```

- Waits for up to 2048 bytes of data.
- Decodes bytes into a UTF-8 string.
- When `timeout` occurs, it silently continues (to allow safe shutdown).
- Messages are added to the GUI using `add_message()`.
- If an exception occurs (server disconnect, network issue), the loop exits and the thread terminates.

Handling Window Closing

The `on_closing()` function is used to shutdown the client when the GUI window is closed.

```
def on_closing():
    """Handle window closing event."""

    global stop_threads
    stop_threads = True  # Stop the listening thread

    try:
        client.shutdown(2)  # Shut down both send and receive operations
        client.close()  # Close the socket connection
    except Exception as e:
        print(f"Error closing socket: {e}")

    main_window.destroy()  # Close the Tkinter window
```

Arguments:

- None

Returns:

- None (runs indefinitely in a thread)

Purpose:

- Handles GUI close events (window X or Ctrl+C).
- Gracefully shutdown the socket and thread.
- Ensures both sending and receiving are disabled before closing the socket.
- Closes the GUI window.

Building the GUI

The next few lines are all used to build the GUI of the chat application

```
# Build the GUI
main_window = tk.Tk()
main_window.geometry("600x600")
main_window.title("Messenger Client")
main_window.resizable(False, False)

# Bind the close event
main_window.protocol("WM_DELETE_WINDOW", on_closing)

# Create the GUI layout
top_frame = tk.Frame(main_window, width=600, height=100, bg=DARK_GREY)
middle_frame = tk.Frame(main_window, width=600, height=400, bg=MEDIUM_GREY)
bottom_frame = tk.Frame(main_window, width=600, height=100, bg=DARK_GREY)

top_frame.grid(row=0, column=0, sticky=tk.NSEW)
middle_frame.grid(row=1, column=0, sticky=tk.NSEW)
bottom_frame.grid(row=2, column=0, sticky=tk.NSEW)

username_label = tk.Label(top_frame, text="Enter username:", font=FONT, bg=DARK_GREY, fg=WHITE)
username_label.pack(side=tk.LEFT, padx=10)

username_textbox = tk.Entry(top_frame, font=FONT, bg=MEDIUM_GREY, fg=WHITE, width=23)
username_textbox.pack(side=tk.LEFT)

username_button = tk.Button(
    top_frame, text="Join", font=BUTTON_FONT, bg=OCEAN_BLUE, fg=WHITE, command=connect_to_server
)
username_button.pack(side=tk.LEFT, padx=15)

message_textbox = tk.Entry(bottom_frame, font=FONT, bg=MEDIUM_GREY, fg=WHITE, width=38)
message_textbox.pack(side=tk.LEFT, padx=10)

message_button = tk.Button(
    bottom_frame, text="Send", font=BUTTON_FONT, bg=OCEAN_BLUE, fg=WHITE, command=send_message
)
message_button.pack(side=tk.LEFT, padx=10)

message_box = scrolledtext.ScrolledText(
    middle_frame, font=SMALL_FONT, bg=MEDIUM_GREY, fg=WHITE, width=67, height=26.5
)
message_box.config(state=tk.DISABLED)
message_box.pack(side=tk.TOP)
```

Running the Client

Finally the last few lines starts the Tkinter event loop, making the GUI run. The if statement ensures the GUI runs only when the script is executed, not imported.

```
if __name__ == "__main__":
    # Run the Tkinter main loop
    main_window.mainloop()
```