Programming Network Sockets

Trevor Ritchie

Project Overview

This project implements a multi-client TCP chat room application using Python's `socket` and `threading` libraries. The system consists of a server (`chat_server.py`) that manages multiple client connections and a client application (`chat_client.py`) that allows users to send and receive messages in real-time.

How the Server Manages Multiple Clients

The server uses **threading** to handle multiple clients concurrently. When the server starts, it creates a TCP socket, binds it to a host and port, and enters a listening state. The main server loop continuously accepts new connections using `server_socket.accept()` , which blocks until a client attempts to connect.

Threading Architecture

When a client connects, the server immediately spawns a new thread to handle that specific client:

```
client_thread = threading.Thread(
    target=handle_client,
    args=(client_socket, client_address),
    daemon=True
)
client_thread.start()
```

Each thread runs the `handle_client()` function independently, allowing the server to process multiple clients simultaneously without blocking. The use of **daemon threads** ensures that these threads automatically terminate when the main program exits.

Thread-Safe Client Management

To prevent race conditions when multiple threads access shared data, the server maintains a global `clients` list protected by a `threading.Lock`:

```
clients_lock = threading.Lock()

with clients_lock:
    clients.append((client_socket, username))
```

This lock ensures that only one thread can modify the clients list at a time, preventing corruption when clients join or leave simultaneously. The `broadcast()` function also uses this lock when iterating through all connected clients to send messages.

Client Lifecycle

1. **Connection**: Client connects → Server accepts → New thread created
2. **Username Request**: Thread prompts for and receives username
3. **Message Loop**: Thread continuously receives messages using `recv(1024)` and broadcasts them
4. **Disconnection**: When `recv()` returns empty bytes, the client has disconnected → Thread cleans up and exits

---

Username and Timestamp Handling

Username Management

When a client first connects, the server sends a prompt requesting a username. The `handle_client()` function waits for this username before proceeding:

```
client_socket.sendall("Please enter your username: ".encode('utf-8'))
username = client_socket.recv(1024).decode('utf-8').strip()
```

If the client sends an empty username, the server assigns a default name using the client's port number (`Guest_12345`). Each client's username is stored alongside their socket connection in a tuple `(connection_socket, username)` within the global clients list.

Timestamp Implementation

Every message is prefixed with a timestamp generated by the `get_timestamp()` function:

```
def get_timestamp():
    return datetime.now().strftime('[%H:%M:%S]')
```

This creates a consistently formatted timestamp like `[14:30:45]`. The timestamp is prepended to all messages, including:

- Chat messages: `[14:30:45] Alice: Hello everyone!`
- Join notifications: `[14:30:45] Bob has joined the chat!`
- Leave notifications: `[14:30:45] Charlie has left the chat.`

This ensures all participants can see when events occurred, which is crucial for understanding conversation flow in an asynchronous environment.

Message Broadcasting

When a message is received from a client, it's formatted with both timestamp and username before being broadcast to all other connected clients (excluding the sender):

```
formatted_message = f"{get_timestamp()} {username}: {message}"
broadcast(formatted_message, client_socket)
```

The `broadcast()` function iterates through all connected clients and uses `sendall()` instead of `send()` to ensure complete message delivery, as `send()` may only transmit part of the data.

Client Implementation

The client uses a **dual-threaded architecture**:

1. **Receive Thread**: Continuously listens for incoming messages from the server using `socket.recv()` and displays them
2. **Main Thread**: Captures user input and sends messages to the server using `socket.sendall()`

This design allows users to receive messages at any time while typing, creating a responsive chat experience. The global `running` flag coordinates shutdown between both threads, ensuring clean termination when the user types 'quit' or the server closes the connection.

Proposed Improvements

While the current implementation works well for text-based chat, there are two significant improvements that would enhance the application's functionality and security.

1. Application-Layer Protocol

Implementing a **structured application-layer protocol** with proper message headers, similar to HTTP, would greatly enhance the application's capabilities.

Current Limitation

The application currently sends raw UTF-8 encoded text with newline delimiters. This approach has limitations:

- Cannot send binary data (images, files)
- Messages containing newlines could cause parsing issues
- No support for different message types (commands vs. chat messages)
- No guarantee of complete message delivery if network conditions cause fragmentation

Proposed Solution

Implement a three-layer message protocol:

1. **Fixed-Length Header (2 bytes)**: Contains the length of the JSON header
2. **JSON Header**: Metadata including content-type, content-length, and encoding
3. **Message Content**: The actual data payload

This would allow the chat application to support:

- **Private messaging**: Using action headers like `{"action": "private", "recipient": "Bob"}`
- **File transfers**: Supporting `content-type: "binary/image"` for sending images
- **Commands**: Distinguishing `/help` commands from regular chat messages
- **Reliable delivery**: Knowing exact message length prevents incomplete reads

This protocol design would make the application more robust and extensible, enabling future features like emoji reactions, message editing, and multimedia support without breaking backward compatibility.

2. SSL/TLS Encryption

Currently, all messages are transmitted in **plain text** over the network, making them vulnerable to eavesdropping and man-in-the-middle attacks. Any user on the same network could use packet capture tools like Wireshark to read all chat messages.

Generative AI Use

For this project, I utilized **Claude Sonnet 4.5** integrated within the Zed editor as a development assistant. I provided Claude with the full assignment instructions and the Real Python socket programming tutorial as context to help guide the implementation.

Claude assisted with:

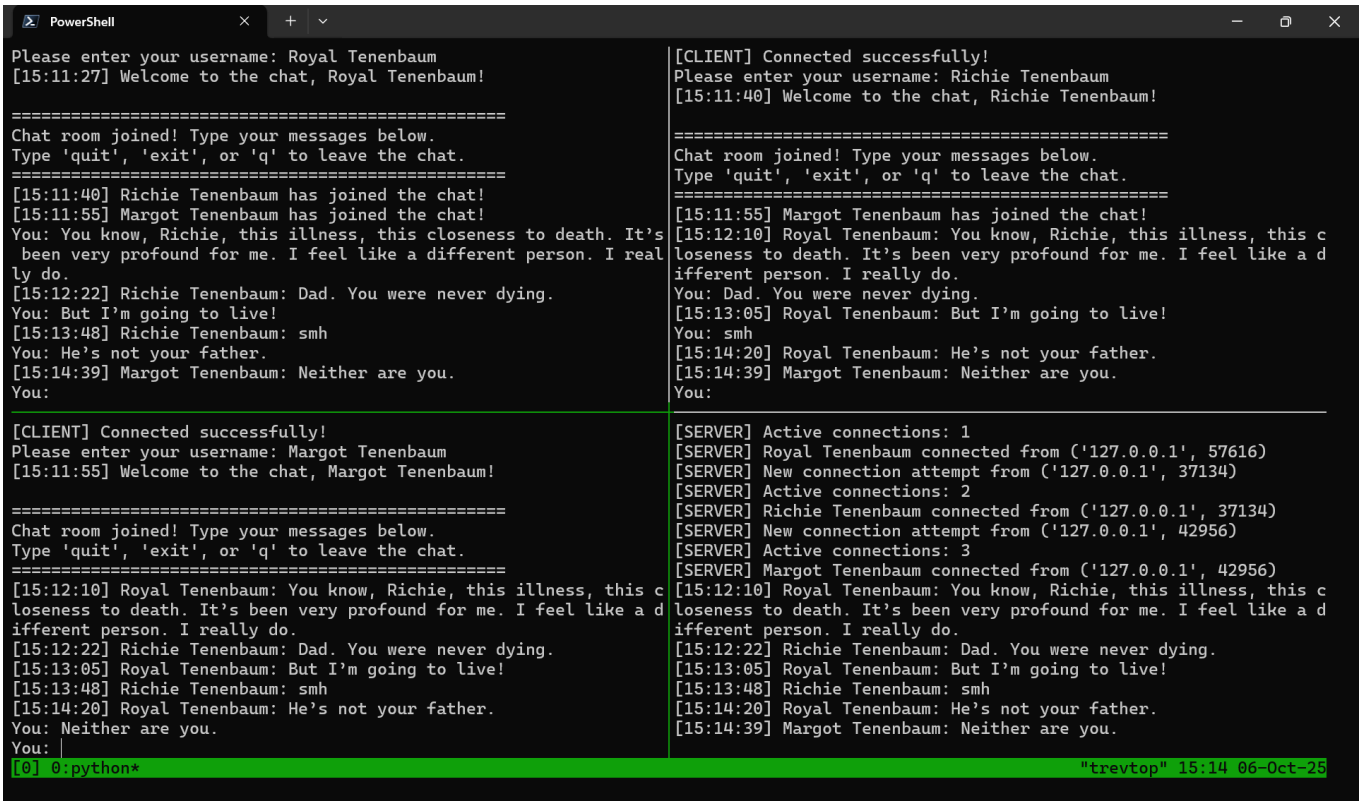
- Generating the initial code structure for both `chat_server.py` and `chat_client.py`
- Implementing best practices for socket programming (thread-safe operations, proper exception handling)
- Adding comprehensive comments and docstrings throughout the code
- Suggesting improvements based on the tutorial's advanced concepts

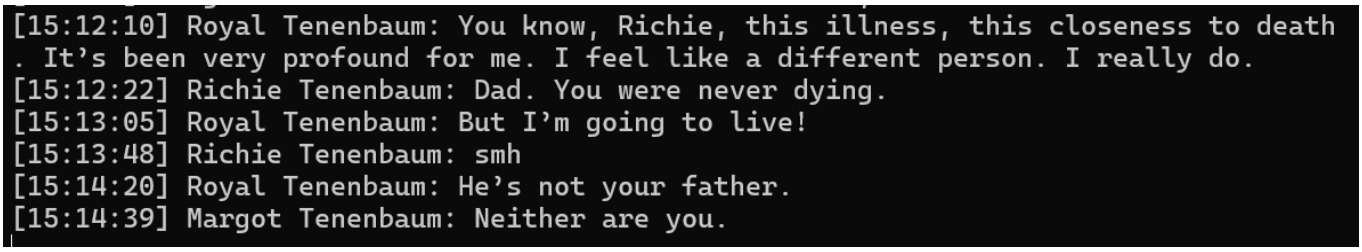
However, I maintained full oversight of the development process by:

- **Reviewing all generated code** line-by-line to ensure I understood every function and method
- Verifying that the implementation met the assignment requirements
- Testing the application with multiple clients to confirm functionality
- Making informed decisions about which features to implement and which to discuss as future improvements

This approach allowed me to learn socket programming concepts more efficiently while ensuring I fully comprehended the code I submitted.

Royal Tenenbaums Scene


PowerShell
Please enter your username: Royal Tenenbaum
[15:11:27] Welcome to the chat, Royal Tenenbaum!
==========
Chat room joined! Type your messages below.
Type 'quit', 'exit', or 'q' to leave the chat.
==========
[15:11:40] Richie Tenenbaum has joined the chat!
[15:11:55] Margot Tenenbaum has joined the chat!
You: You know, Richie, this illness, this closeness to death. It's been very profound for me. I feel like a different person. I really do.
[15:12:22] Richie Tenenbaum: Dad. You were never dying.
You: But I'm going to live!
[15:13:48] Richie Tenenbaum: smh
You: He's not your father.
[15:14:39] Margot Tenenbaum: Neither are you.
You:
[CLIENT] Connected successfully!
Please enter your username: Richie Tenenbaum
[15:11:40] Welcome to the chat, Richie Tenenbaum!
==========
Chat room joined! Type your messages below.
Type 'quit', 'exit', or 'q' to leave the chat.
==========
[15:11:55] Margot Tenenbaum has joined the chat!
[15:12:10] Royal Tenenbaum: You know, Richie, this illness, this closeness to death. It's been very profound for me. I feel like a different person. I really do.
You: Dad. You were never dying.
[15:13:05] Royal Tenenbaum: But I'm going to live!
You: smh
[15:14:20] Royal Tenenbaum: He's not your father.
[15:14:39] Margot Tenenbaum: Neither are you.
You:
[CLIENT] Connected successfully!
Please enter your username: Margot Tenenbaum
[15:11:55] Welcome to the chat, Margot Tenenbaum!
==========
Chat room joined! Type your messages below.
Type 'quit', 'exit', or 'q' to leave the chat.
==========
[15:12:10] Royal Tenenbaum: You know, Richie, this illness, this closeness to death. It's been very profound for me. I feel like a different person. I really do.
[15:12:22] Richie Tenenbaum: Dad. You were never dying.
[15:13:05] Royal Tenenbaum: But I'm going to live!
[15:13:48] Richie Tenenbaum: smh
[15:14:20] Royal Tenenbaum: He's not your father.
You: Neither are you.
You:
[SERVER] Active connections: 1
[SERVER] Royal Tenenbaum connected from ('127.0.0.1', 57616)
[SERVER] New connection attempt from ('127.0.0.1', 37134)
[SERVER] Active connections: 2
[SERVER] Richie Tenenbaum connected from ('127.0.0.1', 37134)
[SERVER] New connection attempt from ('127.0.0.1', 42956)
[SERVER] Active connections: 3
[SERVER] Margot Tenenbaum connected from ('127.0.0.1', 42956)
[15:12:10] Royal Tenenbaum: You know, Richie, this illness, this closeness to death. It's been very profound for me. I feel like a different person. I really do.
[15:12:22] Richie Tenenbaum: Dad. You were never dying.
[15:13:05] Royal Tenenbaum: But I'm going to live!
[15:13:48] Richie Tenenbaum: smh
[15:14:20] Royal Tenenbaum: He's not your father.
[15:14:39] Margot Tenenbaum: Neither are you.
[0] 0:python*
"trevtop" 15:14 06-Oct-25


[15:12:10] Royal Tenenbaum: You know, Richie, this illness, this closeness to death. It's been very profound for me. I feel like a different person. I really do.
[15:12:22] Richie Tenenbaum: Dad. You were never dying.
[15:13:05] Royal Tenenbaum: But I'm going to live!
[15:13:48] Richie Tenenbaum: smh
[15:14:20] Royal Tenenbaum: He's not your father.
[15:14:39] Margot Tenenbaum: Neither are you.