RELATÓRIO PROJETO FINAL CHAT – CONECTIVIDADE EM SISTEMAS CIBERFÍSICOS

Estudantes:

- Allanys Schultz Moraes
- Gabriel Skarbek Lyra
- Gustavo Araujo de Paula

Utilização de Sockets

- O projeto utiliza o protocolo TCP para estabelecer uma conexão confiável entre o servidor e os clientes. No lado do servidor (servidor.py), o socket é configurado utilizando (socket.AF_INET) para IPv4 e (socket.SOCK_STREAM) para o protocolo TCP. Após ser criado, o servidor é vinculado ao endereço (127.0.0.1) e à porta (12345), começando a escutar por conexões com listen().

- No cliente (cliente.py), o socket é configurado da mesma maneira e conecta-se ao servidor usando connect(). Cada mensagem enviada ou recebida é transmitida como uma string codificada em UTF-8.

- O broadcast no servidor é implementado na função (broadcast), que percorre a lista de clientes conectados e envia a mensagem a todos, exceto o remetente. Como o TCP não suporta broadcast nativo, isso é tratado através da manutenção de uma lista de sockets conectados.

Tratamento de Broadcast

- O broadcast é implementado no lado do servidor para garantir que todos os usuários conectados recebam as mensagens globais. Na função (broadcast), cada cliente na lista (clientes) recebe a mensagem, desde que não seja o remetente. A mensagem é enviada utilizando send() e, caso ocorra uma falha, o cliente problemático é removido, além disso mensagens de entrada e saída de usuários são tratadas como broadcasts.

Utilização de Threads

- O uso de threads é essencial para garantir que tanto o servidor quanto os clientes possam executar tarefas simultâneas, como:

- **No servidor**: Cada cliente conectado é gerenciado por uma thread separada, criada dentro da função start_server utilizando threading.Thread. Essa abordagem permite que o servidor atenda múltiplos clientes ao mesmo tempo, escutando suas mensagens e realizando as operações apropriadas (broadcast, mensagens privadas etc.).

- **No cliente:** Uma thread dedicada (threading.Thread) é responsável por receber mensagens do servidor de forma contínua na função receive_messages. Isso evita que o cliente fique bloqueado enquanto aguarda mensagens e permite que o usuário continue interagindo com a interface.

Identificação dos Usuários

- Cada usuário precisa informar seu nome ao conectar-se ao servidor. Esse nome é enviado ao servidor, que o armazena em um dicionário (usernames), associando o nome ao respectivo socket. Isso permite identificar quem está enviando mensagens e implementar mensagens privadas utilizando o prefixo (@), quando um cliente se conecta, seu nome é transmitido a todos os usuários via broadcast. Da mesma forma, sua saída do chat é notificada aos outros.

Requisitos Atendidos

- **Identificação dos usuários:** os usuários precisam informar seus nomes antes de acessar o chat. O nome é obrigatório, e mensagens privadas podem ser enviadas utilizando o formato (@nome).

- **Broadcast:** Todas as mensagens públicas são transmitidas a todos os usuários conectados, utilizando a função broadcast.

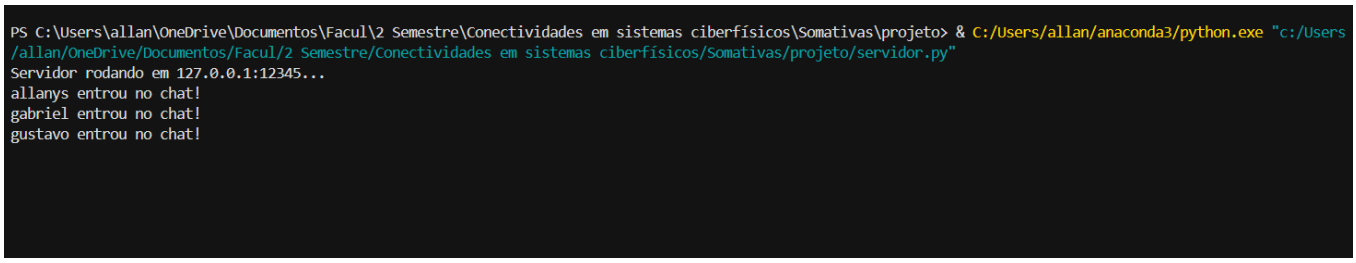
- **Mensagens Privadas:** Um sistema de mensagens privadas é implementado. Caso a mensagem comece com (@nome), ela é enviada apenas para o destinatário especificado.

- **Conexões TCP:** O projeto utiliza o protocolo TCP, garantindo a entrega confiável das mensagens.

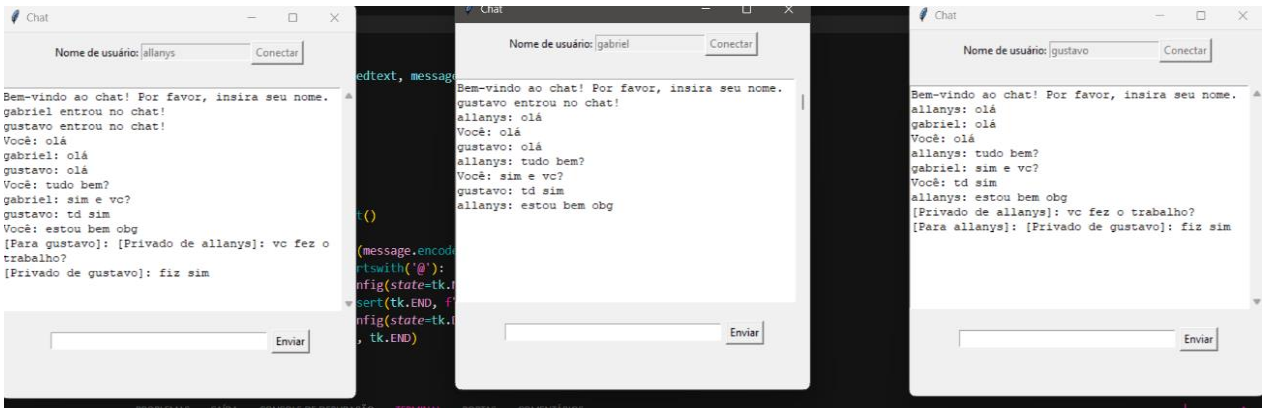
- **Notificações de Entrada e Saída:** Os usuários recebem notificações quando alguém entra ou sai do chat, este projeto atende aos requisitos especificados, fornecendo um chat funcional com funcionalidades de mensagens públicas e privadas, notificações de status e uma e uma interface gráfica no cliente.

Interface Gráfica do Cliente

- **Conexão com o Servidor:**



- **Interface do Cliente:**



- **Deslogando do Servidor:**