Projeto Final – Chat

Relatório – Requisitos funcionais

Enzo barros – Thiago Akimoto – Vitor Lisboa

Relatório

1. Utilização dos Sockets e Conexão TCP:

- Usamos os sockets para permitir a comunicação entre o cliente e o servidor de chat. Escolhemos o protocolo TCP (Transmission Control Protocol), que garante uma comunicação confiável e orientada a conexão.
- No código do servidor, configuramos o socket com socket.AF_INET (para endereços IPv4) e socket.SOCK_STREAM (para TCP). Para que o servidor aceite conexões TCP e troque mensagens de forma confiável com os clientes.
- O servidor usa 'bind' para associar o socket a um IP específico (no caso, 127.0.0.1, que representa o localhost) e uma porta (9999). Em seguida, ele chama listen() para aguardar conexões de clientes.
- No lado do cliente, o socket também é configurado como AF_INET e SOCK_STREAM para conectar-se ao servidor usando TCP. A função connect() é usada para estabelecer a conexão com o servidor.

2. Tratamento de Broadcast:

- Como o protocolo TCP não suporta broadcast, o tratamento de broadcast foi implementado no código do servidor.
- Quando o servidor recebe mensagem de um cliente, ele verifica se a mensagem é para todo mundo ou se é privada. Se for para todo mundo, o servidor usa a função broadcast para enviar a mensagem a todos os clientes conectados. Menos para quem esta enviando a mensagem.
- Essa função envia a mensagem individualmente para cada cliente (exceto o remetente), simulando um broadcast em TCP.

3. Threads no Cliente e no Servidor:

- A threads é o que permite que o servidor e o cliente façam ações ao mesmo tempo. Receber e enviar mensagem ao mesmo tempo.
- No servidor, uma nova thread é criada para cada cliente que se conecta, ou seja ele continua aceitando outras conexões enquanto os clientes que já estão conectados consigam enviar mensagens.
- No cliente, criamos uma thread separada para receber mensagens do servidor enquanto o usuário pode enviar mensagens pela interface gráfica. Isso faz com que a interface não trave.

4. Comunicação Privada (Unicast):

- Criamos uma função que permite o envio de mensagem privada entre os usuários usando o elemento ‘ /‘ e depois o nome do usuário.
- Quando o servidor recebe uma mensagem que começa com /, ele interpreta a mensagem como privada e usa a função unicast para enviar essa mensagem apenas ao cliente destinatário, sem notificá-la aos outros clientes.

Requisitos Funcionais

1. **Autenticação do Usuário:**
    - O sistema deve solicitar um nome de usuário para identificação no chat, que será exibido para outros usuários ao entrar ou sair.
2. **Notificação de Entrada e Saída:**
    - Todos os clientes devem ser notificados quando um novo usuário entra ou sai do chat. A entrada e saída de cada usuário são tratadas e comunicadas por meio do servidor usando mensagens de broadcast.
3. **Envio de Mensagens Públicas (Broadcast):**
    - Usuários devem poder enviar mensagens que serão visíveis para todos os outros usuários conectados. Essa funcionalidade foi implementada usando a função broadcast, que envia a mensagem para todos os clientes ativos, exceto o remetente.
4. **Envio de Mensagens Privadas (Unicast):**
    - Usuários devem poder enviar mensagens privadas para um destinatário específico. Essa função é ativada ao digitar /<nome_destinatário> mensagem, e o servidor redireciona a mensagem apenas para o destinatário específico.
5. **Interface Gráfica para o Cliente:**
    - O sistema deve possuir uma interface gráfica para facilitar o uso, onde o usuário possa digitar mensagens e visualizar a conversa. A interface deve ser implementada com a biblioteca Tkinter e inclui uma área para o histórico do chat e um campo para a entrada de mensagens.
6. **Comunicação Confiável com TCP:**
    - O sistema deve utilizar o protocolo TCP para garantir que as mensagens cheguem ao destino sem erros, pois o TCP fornece uma camada de verificação de erro e confiabilidade.