**RELATORIO**

*Luan Estevinho, Laura Guillarducci*

**Utilização dos Sockets**

O sistema de chat foi desenvolvido utilizando sockets para permitir a comunicação entre o servidor e os clientes conectados. Os sockets desempenham um papel fundamental no envio e recebimento de mensagens em rede, sendo configurados de forma específica para garantir a confiabilidade da troca de dados.

**No Servidor:**

O servidor cria um socket configurado para utilizar o protocolo **TCP**, que é indicado por **SOCK\_STREAM**. Esse protocolo foi escolhido por ser orientado à conexão, garantindo que as mensagens cheguem na ordem certa e sem duplicações. Após a criação, o socket é associado a um endereço IP e uma porta com o comando **bind()**. A partir daí, o servidor entra em modo de espera por novas conexões com **listen()**. Assim que um cliente tenta se conectar, o método **accept()** é chamado, permitindo que o servidor gerencie a comunicação individualmente com o cliente.

**No Cliente:**

O cliente, por sua vez, cria um socket com as mesmas configurações e solicita a conexão ao servidor por meio do método **connect()**. Depois de estabelecida a conexão, o cliente pode enviar mensagens ao servidor e, ao mesmo tempo, receber mensagens enviadas por outros usuários conectados.

**Tratamento de Broadcast**

Uma funcionalidade importante do sistema é o envio de mensagens para todos os participantes conectados, conhecida como broadcast. Sempre que um cliente envia uma mensagem geral, o servidor repassa essa mensagem a todos os outros clientes por meio de uma função chamada **broadcast()**. Essa função percorre a lista de clientes ativos no servidor e utiliza o método **sendall()** para enviar a mensagem. Embora o protocolo **TCP** não suporte nativamente a funcionalidade de broadcast, ela foi simulada no código por meio do uso de uma lista que armazena os clientes conectados. Caso o envio falhe (por exemplo, se o cliente desconectou), o cliente problemático é removido da lista, garantindo que a comunicação continue funcionando para os demais participantes.

**Uso de Threads**

**No Cliente:**

Para que o cliente possa enviar mensagens e receber as mensagens do servidor ao mesmo tempo, foi necessário implementar o uso de threads. Uma thread separada é responsável por manter o cliente sempre escutando as mensagens enviadas pelo servidor, sem que isso interrompa as ações do usuário.

**No Servidor:**

O servidor utiliza threads para gerenciar cada cliente conectado. Assim, cada vez que um cliente se conecta, uma nova thread é criada e dedicada exclusivamente a ele. Isso garante que o servidor consiga atender vários clientes ao mesmo tempo, sem que a interação de um bloqueie o funcionamento de outro.

Essa abordagem baseada em threads é essencial para o funcionamento simultâneo e dinâmico do sistema, tanto no cliente quanto no servidor.

**Mensagens Unicast**

O sistema também possui suporte a mensagens privadas. Para isso, o cliente pode utilizar um comando específico no formato **@nome mensagem**, onde “nome” é o destinatário da mensagem. O servidor, ao identificar que uma mensagem possui esse formato, utiliza a função **unicast()** para enviar o texto apenas ao cliente mencionado. Caso o destinatário não seja encontrado, o remetente é avisado.

**Protocolo de Comunicação**

O sistema utiliza o protocolo **TCP**, que foi configurado durante a criação dos sockets com **SOCK\_STREAM**. Esse protocolo foi escolhido por sua confiabilidade, pois garante que os dados cheguem ao destino sem perdas e na ordem correta. Essa característica é essencial em um sistema de chat, onde cada mensagem deve ser entregue com precisão.

Apesar de o **TCP** não oferecer suporte direto a broadcast, foi possível implementar essa funcionalidade por meio de uma abordagem que utiliza a lista de clientes conectados. Com isso, as mensagens gerais são encaminhadas para todos os participantes, permitindo uma comunicação fluida no chat.