Relatório do Projeto de Redes de Computadores

Anthony Gabriel Paulino de Moura (202407213)
Artur Ferreira Marques da Silva (202437820)
João Manoel Santos Lima (202407474)
Victor André Lopes Brasileiro (202407269)

Professor: Almir Pereira Guimarães

1 Descrição do Projeto

O projeto desenvolvido tem como objetivo demonstrar, de forma prática, os conceitos fundamentais de comunicação em redes de computadores, por meio da implementação de um sistema de chat em tempo real utilizando sockets TCP em Python.

A estrutura do sistema é baseada no modelo cliente-servidor, no qual um servidor central coordena a comunicação entre múltiplos clientes conectados simultaneamente. Cada cliente estabelece uma conexão direta com o servidor, que é responsável por receber as mensagens e retransmiti-las (broadcast) para todos os demais participantes, garantindo que todos os usuários compartilhem a mesma sala de conversa.

O sistema foi dividido em dois módulos principais:

- **server.py:** implementa o lado do servidor. É responsável por inicializar o socket TCP, escutar novas conexões, gerenciar os clientes conectados e distribuir as mensagens recebidas. Cada novo cliente é tratado em uma thread independente, garantindo que múltiplos usuários possam se comunicar simultaneamente sem que a execução de um bloqueie a do outro. O servidor também mantém uma lista com os sockets dos clientes conectados e um dicionário que associa cada socket ao respectivo nome de usuário, permitindo controlar entradas e saídas de forma dinâmica e segura.
- **client.py:** implementa o lado do cliente. Ele cria um socket TCP, conecta-se ao servidor e envia o nome do usuário logo após o estabelecimento da conexão. A partir desse ponto, o cliente é capaz de enviar e receber mensagens simultaneamente, utilizando também duas threads — uma dedicada ao envio de mensagens digitadas e outra à recepção de mensagens do servidor. Essa separação permite uma experiência interativa em tempo real, semelhante a um chat convencional.

O sistema foi desenvolvido para operar em ambiente local (localhost), utilizando o endereço IP `127.0.0.1` e a porta `5000`. Apesar disso, o mesmo código poderia ser facilmente adaptado para funcionar em redes externas, bastando alterar o IP do servidor.

Entre as principais características da aplicação, destacam-se:

- Comunicação bidirecional e concorrente, permitindo múltiplas conexões simultâneas.
- Utilização de threads para evitar bloqueios e garantir desempenho.
- Tratamento de exceções e remoção automática de clientes desconectados, garantindo estabilidade.

- Transmissão de mensagens em formato texto puro, com identificação de cada usuário.

2 Funcionalidades

2.1 Stack de Tecnologias

Utilizada a linguagem de programação Python e o modelo de programação é a programação concorrente baseada em threads.

2.2 Bibliotecas

As bibliotecas utilizadas foram a socket, que é uma lib fundamental para a comunicação em redes, permitindo a criação dos sockets, envio e recebimento dos dados através dos protocolos TCP/IP, etc. E a outra foi a `threading` que permite a administração e tratamento de vários clientes ao mesmo tempo, utilizando o modelo de concorrência por threads.

2.3 Arquitetura de Rede

Utilizado o modelo cliente-servidor. Centralizando, assim, os dados no documento `server.py`. Que, por sua vez, serve para receber as mensagens dos clientes, tratá-las e repassá-las para os outros clientes conectados ao servidor. O protocolo utilizado foi o TCP que adiciona uma camada a mais de confiabilidade e controle, junto com o IPv4 que dá o endereçamento adequado aos clientes.

2.4 Arquitetura de Concorrência

Modelo Multi-threaded por conexão (Uma thread para cada cliente conectado). Se um cliente travar ou demorar a responder a thread dele é bloqueada, enquanto o servidor continua a trabalhar com os outros clientes em threads separadas.

2.5 Funções do Servidor (server.py)

- **Broadcast():** Função que pede uma mensagem e um cliente como argumentos. Ela irá repassar esta mensagem para todos os clientes com exceção do remetente.
- **Handle_Cliente():** Função principal do código. Nela se encontram o loop principal de mensagens e o tratamento de mensagens. No início da função é perguntado o nome de usuário do cliente e então o servidor anuncia, através da função `Broadcast()`, que este usuário entrou no servidor. E então, a partir disso, se inicia o loop de mensagens, onde o servidor recebe mensagens, faz um tratamento, e faz um `Broadcast()` para os outros clientes.
- **iniciar_servidor():** Função que inicializa e configura o servidor. Primeiro criamos um socket TCP. Associamos à um IP e porta e inicializamos as escutas (`listen()`). Inicia-se um loop em que o servidor irá aceitar novas conexões e adicionar os clientes em uma lista de clientes (sockets). E então associamos e iniciamos uma thread para tratar cada cliente.

Observação: Vale salientar que em todas as funções há blocos de código para tratamento de erro, ou seja, se em qualquer momento da execução houver um erro de conexão, o cliente associado à esse erro será desconectado.

2.6 Funções do Cliente (client.py)

- **receber_mensagens():** Recebe as mensagens do servidor, trata e exibe no terminal.

- **enviar_mensagens():** Lê uma mensagem do teclado, trata e envia para o servidor.
- **iniciar_cliente():** Inicialmente cria um socket TCP, IPv4 e se conecta ao servidor. Então, recebe-se a mensagem de boas-vindas e pedido de inserção do nome de usuário. Trata-se essa mensagem e exibe no terminal. Lê-se o nome no terminal, trata e envia ao servidor. Inicializa uma thread para receber as mensagens.

3 Protocolos Implementados

3.1 Transmission Control Protocol (TCP)

O TCP (Transmission Control Protocol) é o protocolo de transporte utilizado pelo sistema e é o responsável por garantir uma comunicação confiável, ordenada e orientada à conexão entre cliente e servidor.

No projeto, cada conexão iniciada entre o cliente e o servidor passa por um processo de estabelecimento de sessão, que internamente realiza o *three-way handshake* (SYN, SYN-ACK, ACK). Esse processo assegura que ambas as partes estejam prontas para iniciar a troca de dados.

Após estabelecida a conexão, o TCP oferece as seguintes garantias:

- **Confiabilidade:** garante que todos os dados enviados cheguem ao destino, retransmitindo pacotes perdidos quando necessário.
- **Ordenação:** os pacotes são entregues na mesma sequência em que foram enviados.
- **Controle de fluxo:** evita sobrecarga no receptor, ajustando dinamicamente a taxa de envio.
- **Controle de congestionamento:** adapta o envio de pacotes às condições da rede, prevenindo quedas de desempenho.

No código, o uso de `socket.SOCK_STREAM` indica explícitamente que o socket utilizará o protocolo TCP. As principais funções aplicadas são:

- `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`: cria um socket TCP baseado no protocolo IPv4.
- `bind()` e `listen()`: usadas pelo servidor para associar o socket ao endereço e colocar o servidor em modo de escuta.
- `accept()`: aceita novas conexões, retornando um novo socket para comunicação individual com cada cliente.
- `connect()`: utilizada pelo cliente para se conectar ao servidor.
- `send()` e `recv()`: responsáveis pelo envio e recebimento de fluxos de bytes, respectivamente.

Essas operações são executadas dentro de threads paralelos, o que garante que múltiplos usuários possam enviar e receber mensagens de forma simultânea. A escolha do TCP se justifica pelo tipo de aplicação: um chat interativo, no qual a perda de mensagens ou a entrega fora de ordem comprometeria diretamente a comunicação entre os usuários.

3.2 Internet Protocol (IP)

O IP (Internet Protocol) atua na camada de rede e é responsável por endereçar e encaminhar pacotes entre dispositivos.

Embora o sistema funcione em ambiente local, o IP é essencial para identificar unicamente o servidor e os clientes participantes da comunicação.

O endereço IP utilizado (`127.0.0.1`) é conhecido como *loopback*, e serve para permitir que o cliente e o servidor se comuniquem dentro da mesma máquina. Esse tipo de endereço é amplamente usado em testes e simulações, pois elimina dependências de rede física.

Na pilha TCP/IP, o IP trabalha em conjunto com o TCP da seguinte forma:

- O IP define o endereço de origem e destino dos pacotes (no caso, o servidor e o cliente).
- O TCP organiza e garante a entrega confiável desses pacotes.

Assim, o IP é o responsável por fazer a entrega até o destino correto, enquanto o TCP garante que a mensagem chegue inteira, na ordem e sem duplicação.

Ainda que o IP não apareça explicitamente no código Python, ele é usado implicitamente pelo sistema operacional para rotear os pacotes entre os sockets conectados, de acordo com o par (endereço IP, porta) definido em cada conexão.

4 Melhorias Possíveis

Apesar de o chat TCP em Python cumprir bem sua função básica de comunicação entre vários clientes via terminal, existem várias melhorias que poderiam tornar o projeto mais estável, eficiente e próximo de um sistema real de mensagens.

Uma das principais seria a implementação de um protocolo de mensagens mais estruturado, definindo um delimitador (como `'\n'`) ou um prefixo indicando o tamanho de cada mensagem. Essa mudança impediria que mensagens maiores que 1024 bytes fossem fragmentadas ou corrompidas durante o envio e recebimento, garantindo uma comunicação mais confiável entre os clientes.

Outra melhoria relevante seria o uso de mecanismos de sincronização, como `threading.Lock()`, para proteger o acesso à lista de clientes no servidor. Atualmente, várias threads podem modificar essa lista ao mesmo tempo, o que pode gerar inconsistências e falhas inesperadas. Com o uso de locks, seria possível evitar esses problemas e garantir que o servidor gerencie as conexões de forma segura.

Além disso, é importante tratar exceções de forma mais específica, substituindo o `except` genérico por erros mais direcionados, como `ConnectionResetError`, `BrokenPipeError` e `socket.timeout`. Essa prática tornaria a depuração mais simples e deixaria o sistema mais transparente, especialmente se combinada com a adição de logs detalhados para cada evento ou erro.

Outra melhoria possível seria permitir configurar o IP e a porta do servidor por meio de argumentos de linha de comando ou variáveis de ambiente. Isso facilitaria a execução em diferentes redes e dispositivos, sem precisar alterar o código manualmente. Por fim, seria interessante implementar autenticação de usuários, verificação de nomes duplicados e até suporte a mensagens privadas ou histórico de conversa, elevando o projeto de um simples chat de terminal para uma aplicação mais completa e próxima de um serviço de chat real.

5 Dificuldades Encontradas

Durante o desenvolvimento do chat TCP, uma das maiores dificuldades foi compreender como funcionam os sockets TCP e as threads trabalhando em conjunto. O uso simultâneo de múltiplas conexões e a comunicação entre elas exigem atenção especial para evitar bloqueios ou travamentos, principalmente quando o servidor precisa lidar com várias mensagens ao mesmo tempo.

Outra dificuldade importante foi garantir a comunicação correta entre os clientes. Como o TCP é um protocolo orientado a fluxo, as mensagens podem chegar fragmentadas ou em blocos diferentes do que foi enviado. Isso exigiu um cuidado extra para entender o comportamento do `recv()` e do `send()`, além de lidar com a desconexão inesperada de clientes.

Também houve desafios relacionados à gestão das threads. Cada cliente conectado gera uma nova thread no servidor, e controlar o encerramento dessas threads de forma limpa, especialmente quando um cliente fecha a conexão sem aviso, não é uma tarefa simples. A ausência de locks e validações específicas pode gerar exceções e comportamentos imprevisíveis.

Por fim, a falta de autenticação e de controle de nomes dificultou a organização do chat quando vários usuários se conectavam ao mesmo tempo. Sem uma forma de validar ou diferenciar os clientes, era fácil ocorrer confusão nas mensagens. Mesmo com essas dificuldades, o projeto serviu como uma ótima introdução prática aos conceitos de redes, comunicação TCP e programação concorrente em Python.