Projeto 2 - Servidor Concorrente TCP e Iterativo UDP

Fernando dos Reis Santos Filho - RA: 234471
Lindon Jonathan S. dos S. P. Monroe - RA: 220407

MC833 1S/2024

Introdução

Este trabalho tem como objetivo demonstrar o uso de sockets na comunicação entre cliente e servidor através de um servidor concorrente TCP e um iterativo UDP. Na comunicação em rede, os sockets são elementos da camada de transporte responsáveis por permitir uma troca bidirecional de informações entre diferentes dispositivos dessa rede. Existem dois tipos de protocolos de sockets: TCP e UDP.

Os sockets TCP (Transmission Control Protocol) são um protocolo full-duplex, orientado à conexão que garante maior confiabilidade no transporte de dados. Quando um cliente deseja se conectar a um servidor TCP ele cria um socket TCP, especifica o endereço IP e o número da porta do servidor e solicita a conexão. O servidor, por sua vez, cria o seu próprio socket e envia uma mensagem de confirmação ao receber a solicitação de conexão por parte do cliente. Uma vez estabelecida a conexão, tanto cliente quanto servidor são capazes de compartilhar informações de forma bidirecional e sequencial, garantindo a entrega e confiabilidade na conexão.

Os sockets UDP (User Datagram Protocol) são um protocolo sem conexão, caracterizado pela sua simplicidade e eficiência na transmissão de dados. Ao contrário dos sockets TCP, não há necessidade de estabelecer uma conexão prévia entre o cliente e o servidor. Quando um cliente deseja enviar dados a um servidor UDP, ele cria um socket UDP e envia datagramas diretamente ao endereço IP e número da porta do servidor especificados. O servidor, por sua vez, também utiliza um socket UDP para receber esses datagramas. A principal vantagem dos sockets UDP é a baixa latência, pois não há overhead de estabelecimento de conexão ou confirmação de entrega. No entanto, isso implica que a entrega dos dados não é garantida, podendo ocorrer perda de pacotes ou recepção fora de ordem, o que torna o UDP ideal para aplicações onde a velocidade é crítica e a perda ocasional de dados é aceitável, como streaming de áudio, vídeo e jogos online.

O servidor desenvolvido neste projeto é um servidor de armazenamento, listagem e download de música capaz de suportar o acesso de múltiplos usuários.

Sistema: descrição geral e casos de uso

O sistema consiste em um servidor e cliente utilizando sockets TCP e UDP desenvolvidos utilizando a linguagem C. O servidor é capaz de armazenar informações sobre músicas, incluindo identificador único, título, intérprete/banda, idioma, gênero da música, refrão e ano de lançamento. Os clientes se conectam ao servidor e podem realizar diversas operações, tais como cadastrar uma nova música, remover uma música existente, listar músicas por ano, idioma, gênero, visualizar informações de uma música específica, entre outras.

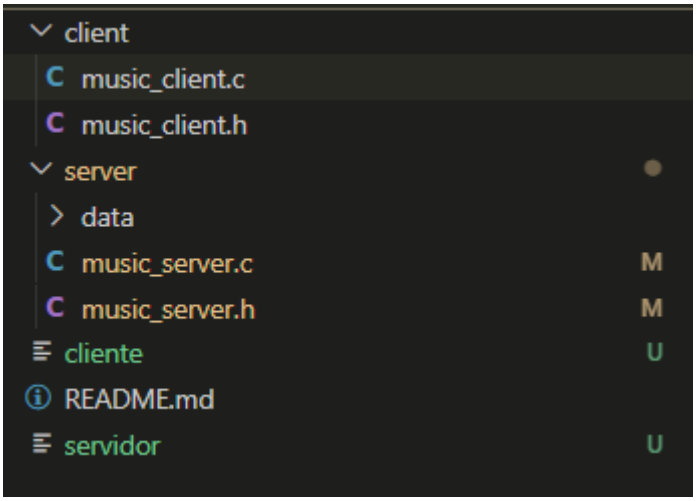

Figura 1: Representação do diretório do projeto

Armazenamento e Estrutura de Dados

Cada música é representada por uma estrutura de dados “Musica”, que armazena os seguintes atributos: identificador único, título, intérprete, idioma, gênero musical, ano de lançamento e refrão. Com esta estrutura que representa uma música a ser armazenada, o servidor possui duas formas de armazenamento desses dados, uma que simula um banco de dados e outra que simula uma memória RAM. Para garantir a persistência dos dados, o servidor armazena as informações das músicas em arquivos textos na pasta data. Assim, mesmo que o servidor reinicie, todas as músicas salvas até então continuarão disponíveis nas próximas sessões. Já para otimizar a busca das músicas existentes e criação do identificador único da músicas, o servidor possui um array da estrutura de dados “Musica”. Mesmo o array não sendo uma estrutura que armazena as informações apenas enquanto o servidor estiver em execução, não garantindo a persistência dos dados, ele se torna muito mais eficiente para realizar a busca das músicas do que abrir o diretório e buscar os arquivos na pasta data.

Dessa forma, sempre que o servidor inicia ele carrega todas as músicas na pasta data no array de “Musica” e executa as operações de inserção, remoção ou listagem das músicas. Na operação de inserção de música o identificador é calculado como a soma do tamanho do array mais um e os dados são armazenados tanto no array quanto em um arquivo texto cujo nome é o identificador único. Para a remoção de uma música pelo seu identificador, basta realizar um pop no array e apagar o arquivo data com o respectivo identificador. Já para as operações de listagem, basta realizar a busca das músicas a serem listadas diretamente no array sem precisar abrir o diretório data.

```c
typedef struct Musica {
    int id;
    char titulo[100];
    char interprete[100];
    char idioma[50];
    char genero[50];
    int ano;
    char refrao[MAXDATASIZE];
} Musica;
```

Figura 2: Estrutura de dados do armazenamento das músicas

Implementação do Servidor

O servidor foi pensado de forma a resolver três pontos principais: permitir que clientes adicionem, manipulem e baixem as músicas que estão sendo coordenadas e armazenadas pelo servidor, permitir que isso seja feito por mais de um cliente simultaneamente e que o cliente não precise estar sendo rodado na máquina local.

Para isso foram utilizados sockets TCP e UDP, que permitem tanto uma conexão confiável com os clientes quanto uma transmissão de baixa latência, e a técnica de multithreading, para permitir que vários clientes se conectem sem um processo interferir no outro. Para entender melhor os detalhes da implementação, é necessário que seja abordado tópico por tópico.

Servidor

O servidor começa com a criação de um socket TCP, o que vai permitir a comunicação com os clientes, e seu endereço é configurado de forma a aceitar conexões em qualquer IP do host. A porta é definida dinamicamente, de forma que não foi definido um

valor fixo para a mesma. Isso se reflete ao realizar a conexão do cliente, em que é preciso informar tanto o IP do servidor quanto a porta.

O servidor faz uma associação desse socket TCP criado com o essa porta e IP e começa a escutar as conexões, aceitando essas conexões em um loop infinito, e para cada conexão ele cria uma nova thread com o socket para conectar o cliente.

Cada uma dessas threads executa a função *client_handler*, que envia o menu de opções para o cliente, recebe a opção escolhida pelo cliente, e invoca a função *handle_client_choice*, que contém o tratamento para cada uma das 9 opções disponíveis no menu.

Essas opções envolvem manipulação das músicas que o servidor armazena, como adição, remoção e exibição das músicas já armazenadas a partir de diferentes critérios e também o upload e download de arquivos mp3 da música cadastrada. Todos esses dados estão contidos na variável global que contém os dados das músicas, mas também em arquivos .txt e mp3 na máquina do usuário, na pasta server/data.

Adicionando a opção de Download de músicas .mp3 do servidor utilizando UDP

Ao escolher a opção de download de músicas (Opção 9 no nosso menu), as funções receive_udp_file e send_file_udp são iniciadas. Estas são responsáveis por criar os sockets UDP, e, no caso do download, transferir o arquivo mp3 que está no servidor em uma sequência de pacotes de dados para o client.

Como a concorrência é gerenciada por múltiplas threads e as músicas são armazenadas em struct global, podemos ter múltiplos clientes conectados, que acessam e modificam a mesma lista de músicas.

Cliente

O cliente funciona conectando-se ao servidor e realizando solicitações e recebendo respostas, conseguindo visualizar e modificar as informações de músicas armazenadas. Para isso, primeiramente o cliente cria um socket TCP e utiliza o IP e a porta do servidor fornecidos para iniciar a conexão. O IP do servidor pode ser o local (como 127.0.0.1), caso o servidor esteja rodando na mesma máquina, ou o IP obtido com comandos como *ipconfig* (Windows) ou *ip a* (Ubuntu).

Após a conexão, o cliente entra em um loop onde lê as mensagens do servidor e responde a solicitações específicas, como a escolha da sua opção do menu ou os dados da música. O envio dessas respostas é feito com write(). Como os clientes são totalmente independentes e o servidor os gerencia em threads diferentes, um cliente não interfere no outro e todos acessam os dados globais das músicas.

Adicionando a opção de Upload de músicas .mp3 do cliente utilizando UDP

Como já citado anteriormente, o socket UDP é utilizado para a transferência de arquivos .mp3 entre o cliente e o servidor. No caso do upload de músicas, ao escolher a opção 1 do nosso menu, para adicionar uma música, o cliente pode também escolher, após inserir os dados da música, como de costume, adicionar um arquivo .mp3 que deve estar na pasta

client/data. Caso o arquivo não exista, o sistema avisa e ignora a tentativa. Dessa maneira, ao adicionar os dados de uma música ao servidor, o cliente pode também adicionar o arquivo da música, e inclusive solicitar baixá-lo do servidor em seguida, tornando a experiência completa

Comparação das Tecnologias

Ao desenvolver esse projeto, fez-se necessário compreender as particularidades dos sockets TCP e UDP em termos de complexidade de implementação, confiabilidade e tempo de execução para decidir onde implementar cada socket.

Comparando a complexidade de implementação, os sockets TCP geralmente resultam em um código mais complexo e extenso. Isso se deve à necessidade de estabelecer conexões, gerenciar sessões, transmitir pacotes perdidos e controlar o fluxo de dados. Em contraste, a implementação de sockets UDP é mais simples e resulta em um código menor, já que não há necessidade de se estabelecer um “handshake” para estabelecer a conexão.

Em termos de confiabilidade, os sockets TCP são altamente confiáveis, pois garantem que os dados cheguem ao destino e na ordem correta através de confirmações (ACKs) e retransmissões de pacotes perdidos. Por outro lado, os sockets UDP oferecem menos confiabilidade, já que não garantem a entrega dos pacotes, a ordem ou a integridade. Pacotes podem ser perdidos, duplicados ou recebidos fora de ordem, o que é aceitável para aplicações onde a velocidade é mais crítica que a confiabilidade.

Em relação ao tempo de execução, os sockets TCP têm um overhead maior devido à necessidade de estabelecer e manter a conexão, bem como de confirmar e transmitir pacotes. Isso pode resultar em maior latência, especialmente devido ao processo de "handshake" inicial e controle de fluxo. Os sockets UDP, por sua vez, têm um menor tempo de execução devido à ausência de conexão, confirmações e retransmissões. A latência geralmente é menor, proporcionando uma comunicação mais rápida, o que é crítico para aplicações em tempo real.

Tendo em vista as particularidades de cada socket, o socket TCP foi desenvolvido para as funcionalidades que exigem confiabilidade na transmissão ao inveś de velocidade, como foi o caso das funcionalidades de cadastro e listagem das músicas. Já o socket UDP foi utilizado na operação de download das músicas. Por se tratar de uma operação com grandes arquivos, necessitando de uma velocidade rápida de download, a baixa latência do socket UDP é o ideal para essa funcionalidade.

Conclusão

O servidor desenvolvido nesse projeto demonstra a aplicação prática do uso de sockets TCP e UDP na comunicação cliente-servidor. Conforme apresentado no projeto, utilizando-se do endereço IP e número da porta tanto do servidor quanto do cliente, é possível estabelecer uma conexão TCP segura, confiável e bidirecional entre servidor e cliente para o cadastro e armazenamento de músicas ao lado de um socket UDP de baixa latência, ideal para o download de músicas. Com o uso da técnica de multithreading múltiplos clientes se conectam simultaneamente no mesmo servidor. Com isso, o projeto cumpre o objetivo de demonstrar e fortalecer o entendimento desses conceitos básicos de rede e de concorrência.

Referências

[1] GeeksforGeeks. Socket Programming in C/C++. Disponível em: https://www.geeksforgeeks.org/socket-programming-cc/. Acesso em: 13 de mar. de 2024

[2] Beej's Guide to Network Programming. Disponível em: https://beej.us/guide/bgnet/. Acesso em: 13 de mar. de 2024

[3] GeeksforGeeks. UDP Client Server using connect | C implementation. Disponível em: https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/. Acesso em: 15 de maio. de 2024