Servidor concorrente sobre TCP

Bruno Sobreira França (217787)
Thiago Danilo Silva De Lacerda (244712)

Abril 2024

1 Introdução

O socket atua como um endpoint para comunicação entre processos em uma rede de computadores. Ele utiliza os protocolos TCP e UDP, entre outros, para estabelecer canais de comunicação, abstraindo as complexidades da comunicação em rede e permitindo que dois processos distintos troquem dados de forma confiável ou não confiável pela rede. No presente trabalho é exploarado o uso de sockets em um servidor concorrente baseado em conexõs TCP de modo a atender um ou mais clientes .

O servidor tem como proposito armazenar informações sobre músicas e controlar o acesso a elas através de operações de escrita, leitura e exclusão, detalhadas ao longo deste relatório. As informações sobe as canções armazenadas são: identificador único, título, intérpretes, idioma, tipo de música (pop, rock, MPB, ...), refrão, ano de lançamento. Neste contexto, os clientes tem o papel de realizar diversos tipos de requisições a esses dados.

Ademais, será explanado ao longo do texto detalhes sobre a implementação e uso dos sockets, como também o racional adotado nas decisões realizadas durante o trabalho. Por fim, haverá demonstrações dos casos de uso, evidenciando as restrições dos programas implementados e os resultados obtidos.

2 Descrição geral e caso de uso

O sistema desenvolvido dividi-se em duas entidades distintas, Cliente e Servidor, com os respectivos papeis de, oferecer uma interface de uso aos usuários e gerênciar a lógica de acesso ao banco de dados de informações relacionadas a músicas. Essa separação, tem como objetivo permitir a execução dos papeis em maquinas distintas, nas quais idealmente o servidor possui uma capacidade de processamento muito maior do que o cliente, que por sua vez, aruaria semelhante a um proxy, enviando requisições do usúario ao servidor e apresentando eles as resposatas. Nota-se, que o servidor será responsavel por executar a mairor carga de trabalho do sistema.

Figura 1: Exemplo básico do fluxo de execução do cliente, no qual são utilizados os comandos SELECT, DELETE e INSERT

Para executar o programa cliente deve ser passados como parametros de execução o ip/hostname e a porta, cujo o servidor está escutando resquisições, a fim de identificar o computador e o processo em questão. Com a execução do programa cliente o usúario tem acesso a um terminal, no qual ele pode realizar prompts em um formato específico para inserir, selecionar ou deletar dados das músicas presentes no banco de dados do servidor. O formato desses prompts variam de acordo com a operação desejada.

Observa-se na figuara 1 um exemplo do fluxo de execução do cliente. Nesta figura, são utilizados os três dos quatro tipos de comandos possíveis: SELECT, cujo seleciona quais informações sobre as músicas será exebida, ou seja, as colunas da base de dados, como exemplo id e titulo; INSERT, utilizado para registrar uma nova música no banco de dados; DELETE, que possui a função de deletar uma música da base de dados, de acordo com um id fornecido. Além destes comandos, existe o comando HELP, que exibe ao usuário uma mensagem de ajuda explicando e exemplificando casos de uso. Tais comando tem um sintax específica, semelhantes a query SQL, que deve ser seguida de acordo com as regras a seguir:

```
INSERT 'id', 'titulo', 'interprete', 'idioma', 'tipo de musica', 'refrao', 'ano de lancamento'

SELECT colunas
```

```
SELECT colunas WHERE tipo_de_coluna='valor a ser filtrado'

SELECT colunas WHERE tipo_de_coluna='valor a ser filtrado' AND tipo_coluna='valor a ser filtrado'

SELECT colunas WHERE tipo_de_coluna='valor a ser filtrado' AND tipo_coluna='valor a ser filtrado'

DELETE id da musica a ser deletada
```

Listing 1: Sintax dos comandos a serem executados

Nota-se, com a descrição da sintax que o comando SELECT possuí uma variação ao adicionar, depois das colunas a serem selecionadas, a palavra WHERE. Com essa verssão do SELECT é possível filtrar os dados das músicas, a serem selecionados, de acordo com um ou mais valores dos campos da canção. Os filtros podems ser extendendidos através das condições AND e OR, que adicionam mais refinamento a busca na base de dados.

Foi presuposto na descrição do trabalho, que determinadas operações deveriam ser realizadas pelo sistema. Dado as característica mais gerais dos prompts disponíveis do trabalho que está sendo aprensentado, mapeou-se quais prompts devem ser realizado a fim de executar as operações exigidas. Esse mapemento está listado a seguir

```
//cadastrar uma nova musica utilizando um identificador
INSERT 'id', 'titulo', 'interprete', 'idioma', 'tipo de musica', 'refrao', 'ano de lancamento'

//remover uma musica a partir de seu identificador
DELETE 'valor do id'

//listar todas as musicas (identificador, titulo e interprete) lancadas em um determinado ano.
SELECT id, titulo, interprete WHERE ano_de_lancamento='valor'

//listar todas as musicas (identificador, titulo e interprete) em um dado idioma lancadas
//num certo ano.
SELECT id, titulo, interprete WHERE ano_de_lancamento='valor' AND idioma='valor'

//listar todas as musicas (identificador, titulo e interprete) de um certo tipo
SELECT id, titulo, interprete WHERE tipo_de_musica='valor'

//listar todas as informacoes de uma musica dado o seu identificador;
SELECT id, titulo, interprete WHERE id='valor'

//listar todas as informacoes de todas as musicas
SELECT *
```

Listing 2: Mapeamento das operações exigidas em relação aos comandos para executa-las no sistema

Para dar início a execução do servidor deve ser passado como parametro de execução o path do arquivo do banco de dados, como é possível notar na figura 2. Observa-se também nesta figura, a característica concorrente do servidor que o torna capaz de atender a requisições de diversos usuários simultaneamente.

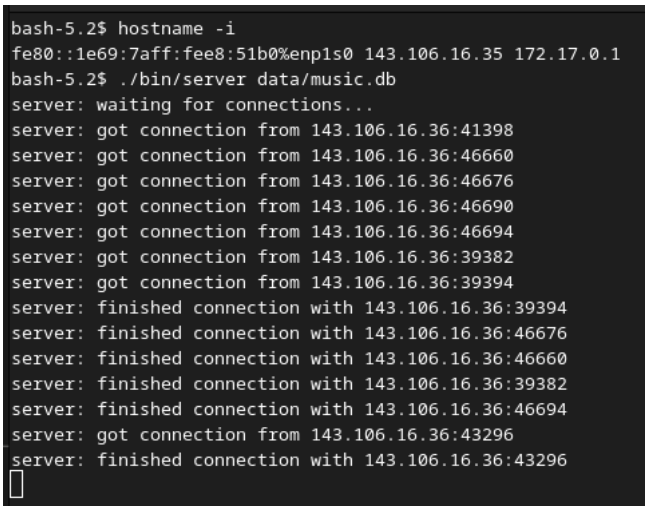

Figura 2: Exemplo do fluxo de execução do servidor, no qual ele recebe requisições de multiplos usúarios

3 Implementação

O sistema client-server foi desenvolvido em C, utilizando a biblietoeca libc em conjunto a SQlite3, com a finalidade de utilizar funções prontas e padronizadas e realizar querys a uma base de dados. Com uso desta linguagem, foram escritos dois programas distintos para atender as funcionalidade das entidades cliente e servidor, respectivamentes presentes no path /src/cliente e /src/server. Além disso, foi criado a biblioteca tcp_exchange_messeage_wrapper, externa a essas aplicações, com a finalidade de criar funções que tratam possíveis erros no envio e recebimento de mensagens através das funções send() e recv() da biblioteca sys/socket.h. Essa biblitoteca se encontra no path libs/exchange_messeage_wrapper/tcp_exchange_messeage_wrapper/.

3.1 TCP_exchange_wrapper

Esse código consiste em um par de funções, recv_message_w() e send_message_w(), juntamente com suas funções de suporte, _recv_message_w() e _send_message_w(), internas ao arquivo de código TCP_exchange_wrapper.c. Essas funções formam uma camada de abstração para enviar e receber mensagens sobre uma conexão TCP.

A função _recv_message_w() lê dados de um socket TCP em pedaços até receber um comprimento especificado de dados. Ela recebe iterativamente dados até acumular o comprimento desejado no buffer fornecido. Ao concluir, ela garante que os dados recebidos sejam terminados em nulo e retorna o total de bytes recebidos, ou um valor menor ou igual a zero se ocorrer um erro;

A função recv_message_w() encapsula o processo de receber uma mensagem de um socket TCP. Primeiro, ela recebe o comprimento da mensagem esperada e, em seguida, aloca memória para armazenar a mensagem com base neste comprimento. Após recuperar os dados da mensagem, ela retorna um ponteiro para a mensagem recebida ou NULL se ocorrer um erro durante a recepção.

Por outro lado, a função _send_message_w() envia dados por um socket TCP em pedaços até transmitir com sucesso toda a mensagem. Ela envia iterativamente dados até que toda a mensagem tenha sido enviada, lidando com quaisquer erros que possam ocorrer durante o processo de

transmissão.

Por fim, a função send_message_w() orquestra o envio de uma mensagem através de um socket TCP. Ela converte o comprimento da mensagem em uma representação de string, envia este comprimento primeiro para garantir que o receptor saiba o tamanho da mensagem esperada e, em seguida, envia os dados reais da mensagem.

Juntas, essas funções fornecem um mecanismo para a troca de mensagens através de sockets TCP em um programa C, lidando com as complexidades da transmissão e recepção de dados enquanto garantem a sincronização adequada entre remetente e destinatário. Elas encapsulam tarefas comuns associadas à comunicação TCP, com a finalidade simplificar a implementação dos aplicativos server e cliente.

3.2 Client

Ao ser executado, o programa cliente inicializa variáveis e estruturas de dados necessárias para se conectar a um servidor por meio de uma rede TCP. Ele recebe dois argumentos da linha de comando: o nome do host ou endereço IP do servidor e o número da porta para se conectar.

Com tais argumento o binário executado tenta conectar-se ao servidor. Caso a conexão seja bem-sucedida, o cliente entra em um loop de comunicação. Dentro deste loop, ele solicita ao usuário que insira uma mensagem com printf(">>>") e lê a mensagem usando fgets() para o buffer msg, que possui o tamanho de 2000 bytes.

Assim que a mensagem é obtida, o cliente a envia para o servidor usando a função send_message_w(), que faz parte da biblioteca tcp_exchange_message_wrapper.h, cujo trata possívei erros relacionados a envio de mensagens via send() da bilioteca sys/socket.h.

O cliente então aguarda para receber uma resposta do servidor usando a função recv_message_w(), também da biblioteca wrapper, com o proposito de realizar o recebimento da mensagens com o tratamento de possíveis erros ao utilizar a função recv() (também do sys/socket.h). Ao receber a resposta, ele imprime a mensagem no console para o usuário ver.

O loop continua indefinidamente, permitindo que o usuário envie várias mensagens de ida e volta entre o cliente e o servidor. Esse processo garante um canal de comunicação contínuo e interativo entre o cliente e o servidor, que pode ser finalizado pelo usuário através do pressionamento conjunto das teclas ctrl e c.

3.3 Server

O arquivo server.c é o componente principal da aplicação de servidor. Ele é responsável por lidar com as conexões de entrada, comunicar-se com os clientes e coordenar as operações do banco de dados. Quando você executa o servidor, ele inicializa os componentes necessários, como configurar sockets e abrir o banco de dados. Em seguida, ele começa a ouvir conexões de entrada em uma porta especificada, no caso a 3490.

Quando um cliente se conecta ao servidor, o servidor aceita a conexão e bifurca um processo filho para lidar com aquele cliente específico. Isso permite que o servidor lide com vários clientes simultaneamente sem esperar que o pedido de um cliente termine antes de processar o pedido de outro cliente.

Dentro do processo filho, o servidor inicia um loop onde espera continuamente receber mensagens do cliente. Ao receber uma mensagem, ele identifica o tipo de solicitação (como INSERT, DELETE, SELECT ou HELP) analisando a mensagem.

Dependendo do tipo de solicitação recebida, o servidor chama diferentes funções de processamento definidas no arquivo process_request.c. Essas funções de processamento lidam com a lógica específica

para cada tipo de solicitação. Por exemplo:

Para uma solicitação de INSERT, o servidor insere dados no banco de dados SQLite. Para uma solicitação de DELETE, ele remove dados do banco de dados. Para uma solicitação de SELECT, ele recupera dados do banco de dados. Para uma solicitação de HELP, fornece informações sobre operações disponíveis.

Cada função de processamento interage com o banco de dados SQLite usando funções definidas no arquivo database.c. Essas funções executam consultas SQL no banco de dados, realizando operações como criar tabelas, inserir registros, selecionar dados ou excluir registros.

Uma vez que a função de processamento tenha concluído sua tarefa, ela gera uma mensagem de resposta, que é então enviada de volta para o cliente. Esta mensagem de resposta pode conter o resultado da operação, como mensagens de sucesso ou falha, ou os próprios dados solicitados.

Após enviar a resposta, o processo filho fecha a conexão com o cliente e termina, devolvendo o controle de volta ao processo pai. O processo pai continua a ouvir novas conexões, repetindo o processo para cada novo cliente que se conecta.

Para o envio e recebeminto de mensagens o server faz uso das funções recv_message_wrapper() e send_message_wrapper() da biblioteca tcp_exchange_message_wrapper, a fim de realizar respectivamente o recebimento e envio de menssagens, por meio do recv() e send() da biblioteca sys/socket.h com os devidos tratamentos necessários para garantia do recebimento e envio das mansagens.

Em resumo, a aplicação do servidor ouve conexões de entrada, lida com as solicitações de cada cliente processando-as e interagindo com o banco de dados, e envia respostas de volta para os clientes. Esta arquitetura permite o tratamento simultâneo de vários clientes.

4 Restrições

Ao longo da realização do projeto optou-se por realizar concessões na implementação dos programas client-server, a fim de focar em partes consideradas mais importantes para ao assunto da materia, como exemplo o desenvolvimento das funções recv_message_wrapper() e send_message_wrapper(). Portanto exite a possibilidade de existir bugs nestes componentes e limitações indesejadas. Algumas indetificadas são:

- Embora o código inclua tratamento básico de erros, pode não cobrir todos os cenários de erro possíveis. Por exemplo, ele pode não lidar abrangentemente com erros relacionados a falhas na alocação de memória ou comportamento inesperado do socket
- O código do Cliente pressupõe um tamanho fixo de buffer de mensagem de 2000 bytes (char msg[2000]). Algo suficiente para realização dos testes, todavia impõe uma limitação no tamanho máximo das requições que podem ser enviadas ao servidor.
- O código utiliza operações de I/O bloqueantes, o que significa que o cliente e o servidor podem ser bloqueados enquanto aguardam o envio ou recebimento de dados. Isso poderia potencialmente levar a problemas de desempenho, especialmente em cenários com alto tráfego de mensagens ou tempos de resposta longos
- As funções de wrapper (send_message_w() e recv_message_w()) assumem um formato de mensagem específico onde o comprimento da mensagem é enviado antes do conteúdo da mensagem real. Este formato deve ser consistente e acordado tanto pelo cliente quanto pelo servidor para uma troca de mensagens adequada.

- Com a finalidade de facilitar a implementação todos os campos das músicas são do típo char *, o que permite o armazenamento de dados incoerentes, como exemplo inserir um texto no campo ano de lançamento.
- O típo do tamanho da mensagem ao ser enviado como parte do "cabeçalho"da mensagem é char * de tamnaho de 200 bytes, o que poderia permitir em tese o envio/recebimento de uma mensagem com até 10^{200} digítos de tamanho, o que pode proporcionar algum erro ao utílizar esse limite máximo.

5 Conclusão

Em conclusão, os códigos desenvolvidos consistem em uma aplicação cliente-servidor funcional com um wrapper para troca de mensagens TCP. Quando executados em conjunto, esses componentes permitem a comunicação entre um cliente e um servidor por meio de uma conexão TCP.

O código do lado do servidor (server.c) configura um socket de servidor, aguarda conexões de entrada e lida com solicitações de clientes. Ele utiliza o SQLite para operações de banco de dados e incorpora funções para processar solicitações de clientes, como inserir, excluir e selecionar dados do banco de dados.

No lado do cliente (client.c), o código estabelece uma conexão com o servidor, envia mensagens inseridas pelo usuário e exibe respostas recebidas do servidor. Ele utiliza uma biblioteca de wrapper (tcp_exchange_message_wrapper.h e tcp_exchange_message_wrapper.c) para enviar e receber mensagens pela rede, garantindo uma comunicação eficiente e confiável.

Quando executados em conjunto, o par cliente-servidor permite que os usuários troquem mensagens de forma interativa, com o cliente enviando solicitações ao servidor e o servidor respondendo conforme necessário. As funções de wrapper facilitam essa comunicação ao lidar com detalhes de formatação e transmissão de mensagens.

Em suma, enquanto o código fornecido serve como base para construir uma aplicação cliente-servidor simples com funcionalidade básica de banco de dados, são necessárias melhorias e otimizações adicionais para atender a requisitos de mundo real afim de garantir robustez e escalabilidade.

6 Referências

- https://beej.us/guide/bgnet/html/
- https://beej.us/guide/bgc/html/split/
- https://beej.us/guide/bgnet0/html/split/
- https://www.sqlite.org/cintro.html
- https://man7.org/linux/man-pages/index.html