Servidor iterativo sobre UDP

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

Maio 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 iterativo baseado em socket DGRAM de modo a atender um ou mais clientes.

O servidor tem como proposito armazenar músicas e controlar o acesso a elas e de suas inforamções 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 e caminho, no qual a música foi armazenada no servidos. 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. Além disso, será feito uma comparação entre as bases de código do servidor UDP, implementado no present trabalho, e o servidor TCP, implementado no trablalho anterior. 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, que armazena as músicas e suas informações. 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, atuaria 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.

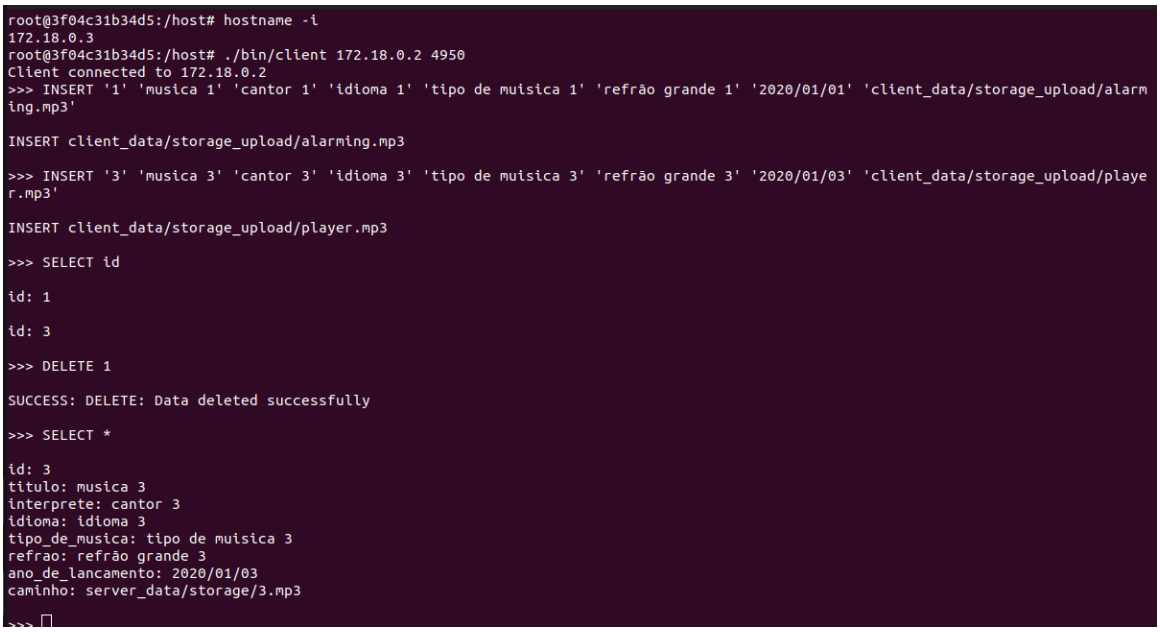

Figura 1: Exemplo básico do fluxo de execução do cliente, no qual são utilizados os comandos SELECT, DELETE e INSERT

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 as 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 cinco 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; 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, existem os comandos DOWNLOAD e HELP, que respectivamente tem a função de baixar uma música em path especifico do cliente e exibir aos 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' 'caminho
      do arquivo da m sica'

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'

DOWNLOAD id da musica a ser baixada

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 *

//Fazer o download de uma arquivo de musica do servidor
DOWNLOAD 'valor do id'
```

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. Percebe-se nesta figura que o servidor atende iterativamente cada requesição realizada presente na figura 1.

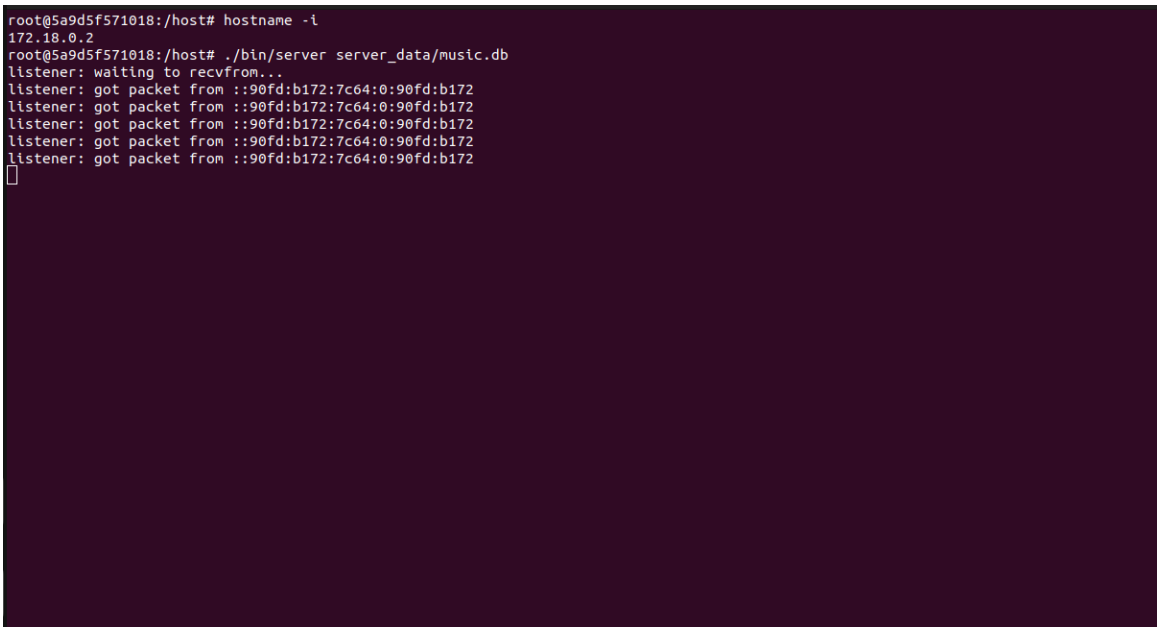

Figura 2: Exemplo do fluxo de execução do servidor, no qual ele recebe multiplas requisições de um usuário

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, foram criadas as biblioteca udp_messeage_exchange_wrapper e udp_file_exchang_wrapper, externa a essas aplicações, com a finalidade de criar funções que tratam possíveis erros no envio e recebimento de mensagens e arquivos através das funções sendto() e recvfrom() da biblioteca sys/-socket.h. Essas biblitotecas se encontra no path libs/.

3.1 UDP_message_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 UDP_message_exchange_wrapper.c. Essas funções formam uma camada de abstração para enviar e receber mensagens.

A função _recv_message_w() lê dados de um socket 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. 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, buscando transferir toda a mensagem que ela recebe como parametro. Ela envia iterativamente dados até que toda a mensagem tenha sido colocada na rede, caso aconteça algum erro neste processo, ela finaliza o programa.

Por fim, a função send_message_w() orquestra o envio de uma mensagem. Ela converte o comprimento da mensagem em uma representação de string, envia este comprimento primeiro em busca de permitir 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, lidando com possiveis falhas no processo de transmissão de mensagen. Elas encapsulam tarefas comuns associadas ao envio de mensgens, com a finalidade simplificar a implementação dos aplicativos server e cliente.

3.2 UDP_file_exchange_wrapper

Esse biblíoteca consiste em um par de funções, recv_file_w() e send_file_w(), internas ao arquivo de código UDP_file_exchange_wrapper.c. Essas funções formam uma camada de abstração para enviar e receber arquivos.

A função send_file_w é responsável por enviar um arquivo. Primeiro, ela abre o arquivo especificado pelo caminho path em modo de leitura binária. Em seguida, calcula o tamanho total do arquivo e determina quantos pacotes serão necessários para enviar o arquivo completo, considerando o tamanho definido de cada pacote (PACKETSIZE). A função então lê o arquivo em blocos de dados, preenche uma estrutura FilePacket com os dados lidos e informações sobre o pacote atual e total de pacotes, e envia esses pacotes sequencialmente através do socket. Após enviar cada pacote, a função insere um pequeno atraso (usleep) para evitar sobrecarregar a rede. Ao final, o arquivo é fechado.

A função recv_file_w é projetada para receber um arquivo e salvá-lo no caminho especificado. Inicialmente, ela abre um arquivo em modo de escrita binária. Ela utiliza um array de pacotes (packetArray) para armazenar temporariamente os pacotes recebidos e um contador para monitorar o número total de pacotes esperados. A função usa select para implementar um mecanismo de timeout, assegurando que a operação não fique bloqueada indefinidamente caso algum pacote não seja recebido. Quando um pacote é recebido, ele é armazenado na posição correspondente do array. A função verifica repetidamente se todos os pacotes esperados foram recebidos e, uma vez que todos são recebidos, escreve os dados dos pacotes no arquivo na ordem correta. Finalmente, o array de pacotes é liberado da memória e o arquivo é fechado.

Juntas, essas funções fornecem um mecanismo para a troca de arquivos, lidando com possiveis falhas no processo de transmissão. Elas encapsulam essa lógica, com a finalidade simplificar a implementação dos aplicativos server e cliente.

3.3 Client

Ao ser executado, o programa cliente inicializa variáveis e estruturas de dados necessárias para se troca de mensagens com o servidor por meio de um socket UDP. 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 obter o endereço do servidor. Caso essa operação 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 udp_message_exchange_wrapper. O cliente então aguarda para receber

uma resposta do servidor usando a função recv_message_w(), também da biblioteca wrapper. Se não houver resposta, é imprimido uma mensagem de erro e o programa é terminado, caso contrário o binário duplica a resposta recebida e usa strtok para separar o comando da resposta. Se o comando é "INSERT", ele obtém o caminho do arquivo a ser enviado, chama send_file_w para enviar o arquivo e imprime uma mensagem confirmando o envio. Se o comando é "DOWNLOAD", obtém o caminho do arquivo no servidor, ajusta o caminho local para salvar o arquivo, chama recv_file_w para receber o arquivo e imprime uma mensagem confirmando o download. Para outros comandos, simplesmente imprime a resposta recebida do servidor. Após processar a resposta, o código libera a memória alocada para a resposta e a duplicata da resposta..

O loop continua indefinidamente, permitindo que o usuário envie várias requisições 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.

3.4 Server

O arquivo `server.c` é o componente principal da aplicação de servidor, responsável por lidar com as conexões de entrada, comunicar-se com os clientes e coordenar operações de banco de dados. Ao executar o servidor, ele inicializa os componentes necessários, como configuração de sockets e conexão com o banco de dados. Em seguida, ele começa a escutar requisições na porta especificada, neste caso, a 4950.

O código começa incluindo as bibliotecas necessárias e definindo algumas constantes. Na função main, ele configura os parâmetros para a criação do socket, utilizando a estrutura addrinfo para especificar que será um socket UDP (SOCK_DGRAM) e permitindo tanto IPv4 quanto IPv6 (AF_UNSPEC). Em seguida, chama getaddrinfo para obter as informações de endereço do servidor. Se ocorrer um erro, imprime uma mensagem e encerra a execução.

Em seguida, tenta criar um socket iterando sobre a lista de endereços retornada por getaddrinfo. Se a criação do socket falhar, imprime um erro e tenta o próximo endereço. Se nenhum socket puder ser criado, imprime uma mensagem de erro e sai. Após a criação bem-sucedida do socket, vincula-o (bind) ao endereço obtido.

O servidor entra em um loop infinito onde espera por mensagens de clientes, utilizanado recv_message_w. Após receber uma mensagem, identifica o tipo de solicitação (como INSERT, DELETE, SELECT, DOWNLOAD ou HELP) e chama a função apropriada para processar a solicitação.

- `insert_request`: Lê os dados da solicitação de inserção, verifica a formatação correta, insere os dados no banco de dados e inicia a transferência de arquivo do cliente.
- `download_request`: Lê o ID do arquivo solicitado, verifica se o arquivo existe, e inicia a transferência do arquivo para o cliente.
- `delete_request`: Lê o ID do arquivo a ser deletado, verifica a existência do arquivo, remove o arquivo e deleta o registro correspondente do banco de dados.
- `select_request`: Executa uma consulta no banco de dados com base nos parâmetros fornecidos e retorna os dados solicitados ao cliente.
- `help_request` e `wrong_request`: Fornecem ajuda sobre os comandos disponíveis ou informam sobre solicitações inválidas.

A função de processamento pode interagir com o banco de dados SQLite, realizando operações específicas de acordo com a solicitação do cliente, e envia uma resposta de volta ao cliente usando send_message_w ou arquivos por meio da send_file_w.

O servidor utiliza funções das bibliotecas personalizadas udp_message_exchange_wrapper.h e udp_file_exchange_wrapper.h para enviar e receber mensagens e arquivos. Isso inclui funções como send_message_w, recv_message_w, send_file_w e recv_file_w, que encapsulam a lógica de comunicação.

Em resumo, a aplicação do servidor escuta requisições, lida com as solicitações de cada cliente processando-as e interagindo com o banco de dados, e envia respostas de volta para os clientes.

4 Comparação entre os projetos UDP e TCP

Nota-se, através da seção 3, que a estrutura do projeto UDP mantém-se semelhante àquela do TCP. Novamente, a carga de trabalho principal do sistema encontra-se no lado do servidor, que basicamente processa todas as requisições do usuário, permitindo que o cliente tenha poucas responsabilidades. Contudo, agora é preciso que o lado do usuário implemente uma lógica que discrimine as solicitações de transferência de arquivos daquelas que apenas enviam e recebem uma mensagem. Tal lógica é necessária para permitir a transferência de arquivos por um novo par de funções auxiliares: recv_file_w e send_file_w.

Este par de funções não está presente no projeto TCP. Conforme explicado na subseção 3.2, essas funções têm o propósito de facilitar o envio de arquivos por meio dos métodos recvfrom e sendto. Esses métodos implementam uma lógica de fragmentação dos dados dos arquivos em pacotes para transmissão. Além disso, essas funções gerenciam o recebimento de pacotes fora de ordem e tratam transações interrompidas caso os pacotes deixem de chegar dentro de um limite de tempo estabelecido. Essa abordagem visa garantir a integridade e a confiabilidade das transferências de arquivos, uma vez que permite lidar com possíveis interrupções ou problemas de ordem na transmissão dos dados.

Em relação às operações disponibilizadas pelo sistema, nota-se que, no lado do servidor, a operação DOWNLOAD foi adicionada para permitir que o usuário baixe canções do servidor. Além disso, observa-se que as operações INSERT e DELETE do servidor foram modificadas de modo a lidar, respectivamente, com o upload de arquivos por parte do usuário para inserir músicas na base de dados e para deletar o arquivos de uma música na base de dados, quando requerido pelo usuário.

Além dessas adições, observa-se que os trechos de código destinados à preparação dos sockets, tanto no cliente quanto no servidor, foram modificados para permitir o uso de sockets UDP. Com o mesmo propósito, as funções send_message_w e recv_message_w foram adaptadas.

Em suma, o sistema tornou-se mais complexo devido às alterações feitas com o propósito de permitir o envio e recebimento de arquivos. A introdução dessas funcionalidades adicionais exigiu ajustes significativos no código, especialmente na configuração e manipulação dos sockets, bem como na implementação de lógicas para lidar com a transferência de arquivos.

5 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 corrente na materia, como exemplo o desenvolvimento das funções recv_file_wrapper() e send_file_wrapper(). Portanto exite a possibilidade de haver bugs nestes componentes e limitações indesejadas. Possíveis bugs ou fontes de bugs 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. As funções send_message_w e recv_message_w são exemplos.
- 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.
- Caso o cliente forneça um caminho para um arquivo inexistente na operação de INSERT, ele terá a execução de seu programa finalizado

6 Conclusão

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

O código do lado do servidor (server.c) configura um socket de udp e aguarda requisições dos clientes. Ele utiliza o SQLite para operações de banco de dados e incorpora funções para processar solicitações de clientes, como inserir, excluir, baxar e selecionar informações de musicas ou o proprio arquivo de música.

No lado do cliente (client.c) envia-se mensagens inseridas pelo usuário e exibe-se respostas recebidas do servidor. Quando o clinte realiza uma requisição que exige a transferência de um arquivo como DOWNLOAD e INSERT um lógica especifica é aplicada para controlar essa transação. Em requesições como o SELECT, apenas as informações requiridas são exibidas.

Quando executados em conjunto, o par cliente-servidor permite que os usuários façam requisições 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 da transmissão de mensagens e arquivos.

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.

7 Referências

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