MC833 Laboratório de redes de computadores
Relatório do projeto 1
Sistema de streaming de filmes usando TCP

Daniel de Sousa Cipriano RA: 233228

Abril 2025

1 Introdução

O objetivo desse projeto foi implementar um sistema de consulta para streaming de filmes, baseado em uma arquitetura cliente-servidor utilizando sockets TCP. Foi solicitada a construção de um sistema com banco de dados para consulta, utilizando bancos conhecidos como SQLite, MySQL, etc; ou utilizando arquivos de armazenamento como em formato JSON, CSV, etc. Os registros a serem armazenados no banco contém as seguintes especificações:

Campo	**Descrição**
Identificador	Número único para cada filme
Título	Nome do filme
Gênero	Pode ter um ou mais gêneros
Diretor	Nome do diretor
Ano de lançamento	Ano em que foi lançado

Foram implementadas sete operações possíveis que os clientes poderiam solicitar ao servidor. São elas:

Operação	**Descrição**
Cadastrar novo filme	Cliente insere dados do novo filme. O servidor gera um identificador único e armazena no banco
Adicionar novo gênero a um filme	Altera o gênero de um filme existente. Verifica existência no banco antes da alteração
Remover filme pelo identificador	Filtra e remove filme do banco pelo identificador fornecido
Listas IDs e títulos	Listar todos identificadores e títulos de filmes
Listar todas informações de filmes	Listar título, gênero, diretor, ano de lançamento e identificador de filme
Listar informações de um filme específico	Listar título, gênero, diretor, ano de lançamento e identificador do filme com o ID fornecido
Listar todos filmes de um gênero	Listar título, gênero, diretor, ano de lançamento e identificador de todos os filmes que se encaixam no gênero fornecido

O sistema segue alguns requisitos técnicos. São eles:

- O servidor deve ser concorrente por processos ou por threads
- O funcionamento do modelo cliente-servidor deve ser testado em máquinas físicas diferentes
- Deve-se implementar mecanismos de sincronização (mutex, semáforos, etc)

- O código deve ser implementado em linguagem C
- O código deve compilar sem erros em ambientes Linux

2 Arquitetura

2.1 Servidor

O servidor foi implementado utilizando sockets em C. Foi configurado para suportar uma fila de espera para conexão de no máximo cinco clientes e a concorrência foi desenvolvida com a utilização de threads de servidor. O servidor foi configurado para escutar em todas interfaces de rede disponíveis. Ao receber uma nova conexão, um novo socket para comunicação é atribuído e uma nova thread é designada para manipular a troca de informações com o cliente. O servidor escuta na porta 8080 e pode enviar ou receber um payload com tamanho máximo de 1024 bytes. O código de servidor foi implementado no arquivo server.c. Foi preferível a escolha de implementação de um servidor concorrente por threads, já que o banco de dados utilizado foi o SQLite, que não lida bem nativamente com concorrência de processos, além de maior simplicidade em manipular o banco com threads que compartilham o mesmo espaço de memória.

2.2 Cliente

Também são utilizadas threads no lado do cliente, cada thread recebe um socket e abre uma nova conexão com o servidor. Foi definido um número máximo de cinco clientes para teste, cada cliente recebe um valor pseudo-aleatório entre 1 e 7 que representa a operação solicitada. O cliente solicita conexão à porta 8080 do servidor e pode enviar ou receber um payload de no máximo 1024 bytes. O código de cliente foi implementado no arquivo client.c.

2.3 Ambiente de testes

Primeiramente foi configurado um ambiente de testes para simulação virtual de máquinas físicas separadas em uma mesma rede, utilizando contêineres Docker. Foram configurados dois contêineres, um armazena a estrutura de servidor e o outro armazena a estrutura de cliente. Os contêineres utilizam a imagem base GCC, que possui todas as dependências para execução de códigos em C[1]. Foi configurada uma rede em modo bridge que inclui ambos os contêineres, a faixa de IPs de sub-rede foi definida como 172.25.0.0/24, o contêiner de servidor possui o IP 172.25.0.10 e o contêiner de cliente possui o IP 172.25.0.11. Posteriormente, o sistema foi testado com cliente e servidor em máquinas fisicamente separadas do laboratório de computadores do Instituto de Computação da Unicamp.

3 Estrutura de armazenamento

Como estrutura de armazenamento, foi utilizado o banco de dados SQLite. Foram incluídos o header de biblioteca sqlite3.h e o arquivo sqlite3.c com a implementação das funções de banco do SQLite. Esses arquivos estão disponíveis para download na página oficial do SQLite[2]. A

implementação das operações de escrita e leitura no banco está no arquivo database.c, que inclui o header de definições database.h. O código de servidor inclui database.h e utiliza as funções implementadas em database.c. O SQLite utiliza padrão SQL para estruturar bancos, armazenando os dados em tabelas e dividindo os registros em linhas, com campos relacionados separados em colunas. Na implementação em database.c, o banco é inicializado criando o arquivo movies.db, que armazena o identificador único de cada registro como chave primária, seguido de colunas com as informações de cada registro, que no caso são os filmes com campos título, gênero, diretor e ano.

4 Descrição das operações

4.1 Operações de servidor

A implementação de servidor possui as seguintes funcionalidades, divididas por funções em código C.

4.1.1 main

Inicializa e encerra o banco de dados; configura e inicia escuta no socket de servidor; recebe conexões e divide threads para lidar com cada conexão individualmente. O código de servidor implementa a estrutura ServerThread, que possui os atributos sock(socket de servidor) e db(ponteiro para manipulação do banco de dados), representa as threads de servidor.

4.1.2 get_id

Gera um identificador inteiro utilizando o método da divisão para gerar chaves de hash com strings. Recebe informações do filme a ser cadastrado(título, gênero, diretor e ano), concatena as strings em uma única string e aplica o método da divisão para gerar um ID inteiro. Esse método garante que a probabilidade de repetição de identificadores para diferentes entradas, seja muito baixa.

4.2 receive_request

Manipula a requisição do cliente para gerar a resposta de servidor. A entrada vinda do cliente é dividida em tokens, utilizando funções de manipulação de strings da biblioteca string.h. A opção de operação solicitada pelo cliente é filtrada e o servidor chama a função equivalente do banco de dados para realizar o pedido e retornar o resultado como resposta.

4.3 thread_handler

Função de thread do servidor. Chama a função receive_request(), sincronizada com um mutex, para evitar condições de corrida no acesso compartilhado ao banco de dados.

4.4 Operações de cliente

A implementação de cliente possui as seguintes funcionalidades, divididas por funções em código C.

4.4.1 main

Configura os sockets de cliente, distribuindo um socket para cada thread de cliente e iniciando a conexão com o servidor. Imprime um menu que orienta o usuário acerca do modelo de entrada. O código de cliente implementa a estrutura Client, que possui os atributos id(marca identificador de cliente), op(marca operação a ser solicitada ao servidor), sock(socket de cliente), server_addr(estrutura de servidor, fornecida na conexão), server_len(tamanho em bytes da estrutura de servidor).

4.4.2 print_log

Imprime um registro para confirmação de conexão com o servidor.

4.4.3 capitalize_string

Converte todas letras de uma string em maiúsculas. Usada para padronizar a requisição enviada ao servidor.

4.4.4 initialize_client

Inicializa estrutura de cliente, com os atributos para conexão e a operação a ser realizada, definida como um valor pseudo-aleatório entre 1 e 7.

4.4.5 connect_thread

Função de thread de cliente. Realiza conexão com o servidor e envia requisição de acordo com a operação desejada. A entrada de requisição segue o padrão $x|a|b|c|...|$ onde x representa o número da operação e a,b,c,... representam os argumentos necessários.

4.5 Operações de banco de dados

A implementação de banco de dados possui as seguintes funcionalidades, divididas por funções em código C.

4.5.1 initialize_database

Inicializa o banco de dados.

4.5.2 close_database

Encerra banco de dados.

4.5.3 is_database_empty

Verifica se o banco de dados está vazio.

4.5.4 add_movie

Adiciona registro de filme ao banco de dados.

4.5.5 genre_update

Atualiza gênero de um registro de filme do banco de dados.

4.5.6 delete_movie

Remove registro de filme do banco de dados.

4.5.7 basic_list

Lista IDs e títulos de todos registros de filme no banco de dados.

4.5.8 all_list

Lista todas informações de todos registros de filme no banco de dados.

4.5.9 id_list

Lista todas informações de um registro de filme no banco de dados, filtrando por um ID específico.

4.5.10 genre_list

Lista todas as informações de todos os registros de filme no banco de dados, filtrando por um gênero específico.

5 Detalhes da implementação

5.1 Bibliotecas

Além das bibliotecas padrão da linguagem C, foram utilizadas as seguintes bibliotecas na implementação[3]:

- sys/types.h, que fornece tipos de dados bem definidos
- sys/socket.h, que fornece funções para criação e manipulação de sockets
- unistd.h, que fornece funções de leitura e escrita
- netinet/in.h, que fornece funções, estruturas e constantes para manipulação de endereços de rede

- arpa/inet.h, que fornece funções de conversão de endereços de rede
- errno.h, que fornece funções e constantes para manipulação de erros
- ctype.h, que fornece funções de conversão de caracteres
- string.h, que fornece funções para manipulação de strings
- pthread.h, que fornece tipos bem definidos e funções para manipulação de threads

5.2 Design de comunicação

A comunicação entre cliente e servidor ocorre através do envio de buffers através de funções de escrita e leitura em socket. O cliente recebe a entrada via arquivo padrão de entrada(stdin), armazena em um buffer e escreve no socket para enviar ao servidor. O servidor lê a requisição, realiza a operação necessária, armazena a resposta em um buffer e escreve de volta para o cliente. O cliente recebe a resposta e escreve no arquivo padrão de saída(stdout). As funções de leitura e escrita padrão da biblioteca unistd.h não possuem controle bem definido de quantidade de bytes a serem escritos ou lidos. Portanto, foi necessária a implementação de funções auxiliares que controlam de forma eficiente a leitura e escrita de dados. Ambos os códigos, de servidor e de cliente, possuem as funções read_all que garante a leitura total de buffer, read_line que garante a leitura de uma linha de buffer e write_all que garante a escrita total de buffer.

6 Conclusão

Por fim, o sistema foi testado em ambiente Docker e em máquinas físicas diferentes, com resultado de funcionamento correto conforme o esperado. A construção do projeto proporcionou um aprendizado prático, com ênfase no funcionamento do modelo cliente-servidor, na comunicação por protocolo TCP via socket e nos desafios de sincronização em um servidor que utiliza multithreading.

Referências

[1] Disponível em: <https://hub.docker.com/_/gcc>.

[2] SQLite Download Page. Disponível em: <https://sqlite.org/download.html >.

[3] The Open Group Base Specifications Issue 8. Disponível em: <https://pubs.opengroup.org/onlinepubs/9799919799/>.