Servidor TCP Concorrente

https://github.com/isabelatelles/mc833

ISABELA TELLES FURTADO DOSWALDO
RA 170012

THAMIRIS FLORINDO COELHO
RA 187506

I. INTRODUÇÃO

O intuito deste trabalho é implementar uma conexão TCP entre cliente e servidor. Para isso, criamos um sistema simples de cadastro de filmes em exibição. O servidor é responsável por armazenar informações sobre os filmes, tais como: título, sinopse, gênero, salas em exibição; enquanto o cliente realiza operações de consulta e armazenamento sobre esses dados. Além disso existem dois tipos de usuários: o administrador e o cliente padrão, de modo que algumas operações podem ser executadas somente pelo administrador.

Este trabalho está separado nas seguintes seções: I. Introdução, II. Descrição Geral e Casos de Uso, III. Armazenamento e Estruturas de Dados, IV. Detalhes de Implementação, V. Conclusões e Referências.

II. DESCRIÇÃO GERAL E CASOS DE USO

O servidor é capaz de suportar múltiplos clientes concorrentemente, pois cria vários *sockets* que conseguem se conectar à porta, de forma que cada um deles se comunica com um cliente. Cada cliente, por sua vez, possui apenas um *socket*, e faz uma requisição tratada por outro *socket* no lado do servidor.

O sistema foi implementado levando em consideração os seguintes casos de uso:

1) Cadastrar um novo filme
2) Remover um filme a partir de seu identificador
3) Listar o título e salas de exibição de todos os filmes
4) Listar todos os títulos de filmes de um determinado gênero
5) Retornar o título do filme que possui determinado identificador
6) Retornar todas as informações do filme que possui determinado identificador
7) Listar todas as informações de todos os filmes

No lado do cliente, é possível escolher um caso de uso para ser realizado. O primeiro e segundo caso de uso, entretanto, são restritos ao usuário administrador.

Neste trabalho o cliente e o servidor foram hospedados em maquinas físicas diferentes, portanto os pacotes trafegam na rede.

Para executar os programas foi feito um Makefile. É preciso rodar os seguintes comandos caso deseja seja rodar, ambos, o cliente e servidor em sua máquina local:

```
$ make client_local PORT=XXXX
$ make server PORT=XXXX
```

Se, por outro lado, deseja rodar o cliente e o servidor em máquinas diferentes, é preciso rodar os seguintes comandos:

```
$ make client_remote IP=XXX.XX.XX
  PORT=XXXX
$ make server PORT=XXXX
```

III. ARMAZENAMENTO E ESTRUTURAS DE DADOS

Para armazenar os dados do servidor, criamos um banco de dados MySQL com apenas duas tabelas: **filme**, cujas colunas são ID, TÍTULO, SINOPSE e GÊNERO; e **sala de exibição**, cujas colunas são NÚMERO DA SALA e a chave estrangeira ID DO FILME. O banco de dados foi modelado dessa maneira para existir a possibilidade de um filme ser exibido em mais de uma sala de exibição ao mesmo tempo. Para fins de teste, o banco foi populado com oito filmes e oito salas de exibições, uma para cada filme. Todas as consultas, inserções e remoções feitas no banco de dados são realizadas pelo servidor e encontram-se no arquivo *database_queries.c*.

As estruturas de dados utilizadas para armazenar os dados retornados pelas consultas podem ser vistas abaixo, bem como no arquivo *database_queries.h*.

```
typedef struct MovieData {
  int id;
  char title[150];
  char synopsis[500];
  char genre[45];
} Movie;

typedef struct ExhibitionRoomData {
  int room_number;
  char movie_title[150];
} ExhibitionRoom;
```

Algumas operações de consulta, entretanto, retornam apenas o título de um filme, ou uma lista de títulos de filmes. Nessas operações não foram utilizadas tais estruturas de dados, pois

não foi julgado necessário, de forma que utilizou-se apenas um vetor do tipo char, caracterizando uma *string*, e uma matriz do tipo char, caracterizanto um vetor de *strings*.

IV. Detalhes de Implementação

A. Comunicação Cliente/Servidor

A comunicação entre cliente e servidor se deu através de uma conexão TCP, de modo que ao se conectarem realizam *handshake*, assim temos certeza que não perderemos informação durante a conexão. Para cada novo processo, o servidor realiza um *fork*, isto foi feito para que o servidor seja concorrente por processos.

B. Mensagens / Requisições

A fim de facilitar a identificação das requisições feitas pelo cliente ao servidor categorizamo-as do seguinte modo:

Categorias	Requisição	Mensagem *
1	Cadastrar um novo filme	[OP][struct movie][número da sala]
2	Remover um filme a partir do seu identificador	[OP][ID do filme]
3	Listar o título e salas de exibição de todos os filmes	[OP]
4	Listar todos os títulos de filmes de um determinado gênero	[OP][gênero do filme]
5	Retornar o título do filme que possui o identificador X	[OP][ID do filme]
6	Retornar todas as informações do filme que possui o identificador X	[OP][ID do filme]
7	Listar todas as informações de todos os filmes	[OP]

* OP é um *enum* atribuído a categoria.

Note que a primeira informação da mensagem é o *enum* referente a operação requisitada pelo cliente. Essa será a primeira informação analisada pelo servidor para identificar qual requisição foi feita, o restante da mensagem é decodificada diferentemente de acordo com a operação desejada.

As mensagens trocadas entre cliente e servidor são transmitidas por um vetor de bytes (tipo char), a fim de facilitar a implementação esse vetor possui um tamanho fixo de 6000 bytes. Desse modo copiamos a memória do vetor recebido para estruturas de dados desejada de acordo com a operação.

As mensagens do servidor para o cliente são compostas pelo vetor de bytes contendo as estruturas esperadas pelo cliente em cada uma das opções, conforme a tabela a seguir:

Categorias	Resposta	Mensagem
1	Identificador do novo filme cadastrado	[ID do filme]
2	1 - Filme removido com sucesso; 0 - Falha na remoção do filme	[1/0]
3	Lista com títulos e salas de exibição de todos os filmes	[tamanho do vetor][vetor de structs ExhibitionRoom]
4	Lista com todos os títulos de filmes de um determinado gênero	[tamanho do vetor][vetor de títulos]
5	Título do filme com identificador X	[título]
6	Informações do filme com identificador X	[struct Movie]
7	Lista com todas as informações de todos os filmes	[tamanho do vetor][vetor de structs Movie]

Nas categorias 3, 4 e 7 foi necessário enviar também a quantidade de elementos que a mensagem continha, pois como dito anteriormente usamos um tamanho fixo de vetor para troca de mensagens e o cliente precisava saber quando era necessário parar de ler as informações.

C. Saída

Utilizamos a saída padrão para imprimir as operação requisitada pelo cliente ao servidor e para imprimir de forma textual a resposta da requisição dada pelo servidor. Caso ocorra algum erro durante as operações esse erro é impresso na saída de erro.

A requisição é feita pelo cliente escolhendo uma das opções mostradas a baixo e o cliente escreve cada um dos parametros que será mandado para o servidor:

```
Choose one option:

0 - Register new movie
1 - Remove a movie of ID
2 - List titles and exhibition rooms of
    all movies
3 - List all movie titles of a GENRE
4 - Return movie title of ID
5 - Return movie informations of ID
6 - List all movies
```

Após o cliente fazer a requisição ao servidor, no servidor a saída mostra a opção que deve executar, o que foi recebido do cliente e o que está enviando ao cliente, como podemos ver a seguir:

```
Socket successfully created..
Socket successfully binded..
Server listening..
OP: 3

RECEIVED
genre: romance

SENT
title: Orgulho e Preconceito
title: Para Todos os Garotos que Ja Amei
```

Após enviar para o cliente a mensagem contendo o retorno de sua requisição, o cliente então imprime na tela a saída correspondente ao que foi recebido.

```
RECEIVED
Title: Orgulho e Preconceito
Title: Para Todos os Garotos que Ja Amei
```

V. Conclusão

A implementação do TCP se mostrou confiável, como era esperado. Durante a comunicação todas as mensagens enviadas foram recebidas devidamente, sem perda de informações. Realizar este trabalho foi importante para reforçar a teoria de TCP aprendida em sala de aula.

Referências

[1] R. Joshi. Example of client-server program in c. [Online]. Available: https://www.programminglogic.com/example-of-client-server-program-in-c-using-sockets-and-tcp/

[2] Y. Shukla. Tcp server-client implementation in c. [Online]. Available: https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/