UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

RELATÓRIO DO PROJETO 1 - SOCKET TCP
DA DISCIPLINA MC833 - PROGRAMAÇÃO DE REDES DE COMPUTADORES

1º SEMESTRE DE 2024
TURMA A

PROFESSOR: EDMUNDO ROBERTO MAURO MADEIRA
ALUNO: RAONITON ADRIANO DA SILVA RA:186291

1. Introdução

Este relatório tem por objetivo detalhar o que foi realizado durante o desenvolvimento do projeto de Sockets TCP. Sockets são 'file descriptors' pelos quais diferentes processos podem se comunicar, estando estes processos no mesmo computador ou em computadores diferentes. O objetivo do projeto, então, foi desenvolver um cliente e um servidor, e realizar a comunicação entre eles.

A proposta do projeto era criar um sistema cliente/servidor, no qual o servidor armazenaria dados de músicas e o cliente faria requisições sobre os dados dessas músicas, tendo ainda um cliente com mais privilégios que seria capaz de, através de requisições ao servidor, cadastrar e remover músicas que estão cadastradas no banco de dados.

2. Objetivos

Os principais objetivos:

a. Implementar o algoritmo do cliente e do servidor

b. Implementar as funções de gerenciamento dos dados

c. Integrar e sincronizar o recebimento de e envio de dados por parte do servidor e do cliente.

E com isso exercitar e entender como funciona a transmissão e recebimento de dados no contexto do TCP.

3. Descrição geral

O projeto proposto era criar um cliente e servidor que fariam a comunicação através de sockets, utilizando-se da linguagem C, e realizando todo o desenvolvimento em um ambiente Linux devido às funcionalidades e bibliotecas já existentes, realizando o envio e recebimento de mensagens sob o protocolo de transmissão TCP(Transmission Control Protocol).

Temos dois programas que executam funções diferentes, sendo eles o `server.c` e o `client.c`.

Descrevendo as etapas mais importantes de ambos programas:

Server:

O server precisa chamar 4 funções para estar apto a trocar dados com o client, sendo elas:

socket() - abre um 'file descriptor' que será usado para a comunicação.

bind() - atribui o socket anteriormente criado a um ip e uma porta

listen() - coloca o socket em modo de espera, esperando uma conexão do cliente.

accept() - estabelece a conexão entre o cliente e o servidor, que a partir desse ponto estão prontos para mandar e responder requests.

No código abaixo estão trechos do código, no entanto, aqui estamos deixando de lado algumas verificações, mas que estão presentes no programa.

```C/C++
//SERVER -> server.c
```

```
//Socket
server_socket = socket(AF_INET, SOCK_STREAM, 0)...

//Garantindo que a sctruct estara zerada
memset(&server_Addr, 0, sizeof(server_Addr));

//Preenchendo as infos em server_Addr
server_Addr.sin_family = AF_INET;
server_Addr.sin_addr.s_addr = INADDR_ANY;
server_Addr.sin_port = htons(PORT);

//Bind
if(bind(server_socket, (struct sockaddr*)&server_Addr, sizeof(server_Addr)) ==
-1)...
//Listen
if(listen(server_socket, LISTENQUEUE) == -1)...

//Accept
while(1){
  client_addr_len = sizeof(client_Addr);
  client_socket = accept(server_socket, (struct sockaddr*)&client_Addr, &
  client_addr_len);
        .
        .
        .
     pid_t pid = fork();
     request(data, client_socket, pid, verificacao);
 }
```

Client:

Diferentemente do server, o client chama 2 funções, sendo elas:

socket() - abre um 'file descriptor' que será usado para a comunicação.

connect() - tenta se conectar com seu socket a uma port e ip.

```
C/C++
//Socket
client_socket = socket(AF_INET, SOCK_STREAM, 0);

//Insere as infos nos campos da struct sockaddr_in server_Addr
server_Addr.sin_addr.s_addr = inet_addr(ip);
server_Addr.sin_family = AF_INET;
server_Addr.sin_port = htons(PORT);

//Connect
```

```
if(connect(client_socket, (struct sockaddr*)&server_Addr, sizeof(server_Addr))
== -1)...
```

Somente após realizar os passos acima que o client e server realmente poderiam tentar tentar se comunicar. Caso a conexão fosse estabelecida, tanto o client como o server poderiam fazer uso de mais duas funções **send() e recv()**, eles trocariam informações através dessas funções até que um dos lados fechasse a conexão.

No contexto do projeto, as mensagens trocadas são: o cliente fazendo requisições de acordo com opções existentes em um menu que é enviado pelo servidor ao cliente

Após se conectarem, o cliente receberá esse menu no terminal, podendo então fazer suas requisições.

Cliente conectado --- Port: 9877 Nivel de privilegio: comum ESCOLHA UMA OPCAO: [1] - Listar todas as musicas e informacoes. [2] - Buscar musica por id. [3] - Listar musicas por ano de lancamento. [4] - Listar musicas por idioma e ano de lancamento. [5] - Listar musicas por estilo musical [0] - Sair. Digite a opcao:	Cliente conectado --- Port: 9877 Nivel de privilegio: super ESCOLHA UMA OPCAO: [1] - Listar todas as musicas e informacoes. [2] - Buscar musica por id. [3] - Listar musicas por ano de lancamento. [4] - Listar musicas por idioma e ano de lancamento. [5] - Listar musicas por estilo musical [6] - Cadastrar nova musica [7] - Remover musica pelo id [0] - Sair. Digite a opcao:

O server é capaz de receber conexões simultâneas, fazendo um fork() para um novo accept() de um client diferente, mesmo que o primeiro client ainda esteja ativo e fazendo requisições.

4. Compilação

Para compilar e executar no mesmo computador:

Abra 2 terminais no diretório da pasta projeto1/

Em um terminal execute o comando: `gcc server.c -o server`

E depois para executar o programa: `./server`

No outro terminal execute o comando: `gcc client.c -o client`

E depois podemos executar o programa de 2 formas:

execute: `./client 127.0.0.1` ou `./client 127.0.0.1 super`

Para compilar e executar em computadores diferentes:
Os passos iguais os acima para o terminal que executará o server, a diferença é que no terminal do client que vai executar em outro computar ao invés de digitar 127.0.0.1, você digitará o IP do computador onde o server está executando no terminal do cliente execute: `./client IP` ou `./client IP super`

5. Casos de uso

Caso de Uso 01: "Executar o programa como cliente sem privilégios."

Atores
Usuário
Sistema

Pré-condições
O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa.

Fluxo Principal
- O usuário digita ./client IP
- O sistema exibe a mensagem "Cliente conectado --- Port: X"
- O sistema exibe a mensagem "Nível de privilégio: comum"

Pós-condições
O sistema exibe o menu.
O usuário está apto a interagir com o sistema.

Exceções:
- Entrada inválida
    - O sistema exibe a mensagem:
        - "uso para cliente comum: ./client IP
        - uso para cliente super: ./client IP super
        - substitua IP pelo IP da maquina onde ./server esta rodando
        - ou por 127.0.0.1 caso esteja rodando na mesma maquina"
        -
    - O sistema encerra a execução.

Caso de Uso 02: "Executar o programa como cliente com privilégios."

Atores
Usuário
Sistema

Pré-condições

O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa.

Fluxo Principal

- O usuário digita ./client IP super
- O sistema exibe a mensagem “Cliente conectado --- Port: X”
- O sistema exibe a mensagem “Nível de privilégio: super”

Pós-condições

O sistema exibe o menu com as opções extras.
O usuário está apto a interagir com o sistema.

Exceções:

- Entrada inválida
    - O sistema exibe a mensagem:
      “uso para cliente comum: ./client IP
      uso para cliente super: ./client IP super
      substitua IP pelo IP da maquina onde ./server esta rodando
      ou por 127.0.0.1 caso esteja rodando na mesma maquina”
    - O sistema encerra a execução.

Caso de Uso 03: “Listagem de todas as músicas e informações.”

Atores

Usuário
Sistema

Pré-condições

O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no client-side.
O usuário deve estar visualizando o menu exibido no terminal do client.

Fluxo Principal

- O usuário digita 1 e pressiona enter
- O sistema exibe uma lista com as informações: "id, Musica, Autor, Idioma, Estilo, Refrão, Ano" de todas as músicas cadastradas.

Pós-condições

O sistema mostra novamente o menu.

Exceções:

- Entrada inválida
    - O sistema exibe a mensagem de erro: Opcao invalida ou nao permitida para esse nivel de privilegio!”

Caso de Uso 04: “Listagem das informações de músicas em determinado id.”

Atores

Usuário

Sistema

Pré-condições
O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no client-side.
O usuário deve estar visualizando o menu exibido no terminal do client.

Fluxo Principal

- O usuário digita 2 e pressiona enter
- O sistema exibe a mensagem: "Digite o id:"
- O usuário digita uma entrada e pressiona enter
- O sistema exibe uma lista com as informações: "id, Musica, Autor" da musica com o id digitado anteriormente.

Pós-condições
O sistema exibe novamente o menu.

Exceções:

- Id não encontrado
    - O sistema exibe a mensagem: ”Nenhum dado encontrado”

Caso de Uso 05: “Listagem das informações de todas as músicas dado um determinado ano de lançamento”.

Atores
Usuário
Sistema

Pré-condições
O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no client-side.
O usuário deve estar visualizando o menu exibido no terminal do client.

Fluxo Principal

- O usuário digita 3 e pressiona enter
- O sistema exibe a mensagem: “Digite o ano de lancamento”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe uma lista com as informações: "id, Musica, Autor” de todas as músicas com ano de lançamento digitado anteriormente.

Pós-condições
O sistema mostra novamente o menu.

Exceções:

- Ano não encontrado
    - O sistema exibe a mensagem: ”Nenhum dado encontrado”

Caso de Uso 06: “Listagem das informações de todas as músicas dado um determinado idioma e ano de lançamento.”

Atores
Usuário
Sistema

Pré-condições
O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no client-side.
O usuário deve estar visualizando o menu exibido no terminal do client.

Fluxo Principal

- O usuário digita 4 e pressiona enter
- O sistema exibe a mensagem: “Digite o idioma”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o ano de lancamento:”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe uma lista com as informações: "id, Musica, Autor” de todas as músicas dado um determinado idioma e ano de lançamento ambas informações digitadas anteriormente..

Pós-condições
O sistema mostra novamente o menu.

Exceções:

- Ano && idioma não encontrados
    - O sistema exibe a mensagem: ”Nenhum dado encontrado”

Caso de Uso 07: “Listagem das informações de todas as músicas dado um determinado estilo musical”.

Atores
Usuário
Sistema

Pré-condições
O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no client-side.
O usuário deve estar visualizando o menu exibido no terminal do client.

Fluxo Principal

- O usuário digita 5 e pressiona enter
- O sistema exibe a mensagem: “Digite o estilo da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe uma lista com as informações: "id, Musica, Autor” de todas as músicas com o estilo musical digitado anteriormente.

Pós-condições
O sistema mostra novamente o menu.

Exceções:

- Estilo musical não encontrado
    - O sistema exibe a mensagem: ”Nenhum dado encontrado”

Caso de Uso 08: “Cadastrar nova musica”.

Atores

Usuário

Sistema

Pré-condições

O sistema do servidor deve ter sido executado e estar rodando no server-side.

O usuário deve ter compilado o programa e estar executando no modo cliente com privilégios

O usuário deve estar visualizando o menu exibido no terminal do client com as opções extras.

Fluxo Principal

- O usuário digita 6 e pressiona enter
- O sistema exibe a mensagem: “*O id sera gerado automaticamente*”
- O sistema exibe a mensagem: “Digite o titulo da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o nome do autor/cantor da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o idioma da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o estilo da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o ano de lancamento da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “Digite o refrao da musica”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “A musica foi cadastrada”

Pós-condições

A música e cadastrada no banco de dados

O sistema mostra novamente o menu.

Caso de Uso 09: “Cadastrar nova musica”.

Atores

Usuário

Sistema

Pré-condições

O sistema do servidor deve ter sido executado e estar rodando no server-side.

O usuário deve ter compilado o programa e estar executando no modo cliente com privilégios

O usuário deve estar visualizando o menu exibido no terminal do client com as opções extras.

Fluxo Principal

- O usuário digita 7 e pressiona enter
- O sistema exibe a mensagem: “Digite o id da musica a ser removida”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe a mensagem: “A musica com o ‘id’ foi removida!”

Pós-condições

A música é removida do banco de dados
O sistema mostra novamente o menu.

Exceções:

- id não encontrado
    - O sistema exibe a mensagem: ”id ‘id’ nao encontrado!”.

Caso de Uso 10: “Sair”.

Atores

Usuário
Sistema

Pré-condições

O sistema do servidor deve ter sido executado e estar rodando no server-side.
O usuário deve ter compilado o programa e estar executando no modo cliente com privilégios
O usuário deve estar visualizando o menu exibido no terminal do client, vale tanto para o cliente comum e para o cliente super.

Fluxo Principal

- O usuário digita 0 e pressiona enter

Pós-condições

A execução termina.

6. Armazenamento e estruturas de dados

a. A organização dos arquivos na pasta:

```
C/C++


    ├─ projeto1/                <- Pasta raiz do projeto
    |    ├─ server.c            <- implementacao do server
    |    ├─ client.c            <- implementacao do client
    |    ├─ configAndCRUD.h     <- implementacao das funcoes
    |    ├─ headerIncludes.h    <- includes, defines e prototipos das
    |    |                         funcoes
    |    ├─ Makefile            <-
    |    ├─ nSongs.csv          <- armazena o numero que representa a
    |    |                         a quantidade de musicas
    |    ├─ songId.csv          <- armazena um numero que representa id
    |    |                         da proxima musica a ser cadastrada
    |    └─ songdData.csv       <- armazena os dados de cada musica
    |                              cadastrada, "id, musica, autor,..."
    |
    └──────────────────────────────────────────────────────────
```

Legenda:

- Arquivos presentes na pasta antes de compilar pela primeira vez
- Arquivos adicionados à pasta após compilar

O programa verifica se os arquivos nSongs.csv, songId.csv e songData.csv já existem, caso não existam eles serão criados, caso já existam, as informações contidas neles são carregadas para a memória do programa.

nSongs.csv:
É um arquivo csv que mantém atualizado o número de músicas cadastradas no arquivo songsData.csv, o intuito é não ter que contar a quantidade de linhas todas as vezes para saber o número de músicas cadastradas. E como essa informação teria que ser inúmeras vezes, optou-se por manter um arquivo com essa quantidade salva.
A estrutura é:

```
Unset
Quantidade de dados
0
```

songId.csv:
É um arquivo csv que mantém um número que será usado pelo programa quando for cadastrar uma nova música. Então o primeiro número existente será o 0. No momento do cadastro de uma nova música, esse arquivo é lido, o valor salvo para ser o id daquela

música, e o valor que está no songId.csv é incrementado. Garantindo assim um id diferente para cada música, funcionando como uma chave primária.
A estrutura é:

```
Unset
SongId
7
```

songData.csv:
É um arquivo csv que mantém os dados das músicas, o arquivo inicia vazio, mas a cada novo cadastro de músicas, elas vão sendo salvas no arquivo csv.
A estrutura é:

```
Unset
Id,Titulo,Interprete,Idioma,Tipo,Refrao,Ano
10,Teste,MC testinho,Portugues,Funk,Teste hoje para nao testar amanha,2024
```

b. Estrutura de dados
Os dados que foram salvos nos .csv, serão carregados para um vetor de struct Data todas as vezes que o programa server for executado, a struct Data tem a seguinte estrutura:

```
C/C++
typedef struct Data{
    int  id;
    char title[STR];
    char auth[STR];
    char lang[STR];
    char style[STR];
    char chorus[MAXSTR];
    int year;
}Data;
```

O vetor de struct Data é alocado dinamicamente e altera o seu tamanho a cada nova adição de música ou exclusão.

7. Detalhes de implementação

Uma vez que o server esteja esperando por conexões, ele entra num laço while até que e sempre que um client se conecta o server faz um **fork()** e chama uma função que lidará com as requisições do client. Nesse ponto um processo filho foi criado, enquanto o server que é o pai volta a aguardar por novas conexões.

```C/C++
while(1){
        client_addr_len = sizeof(client_Addr);
        client_socket = accept(server_socket, (struct sockaddr*)&client_Addr,
&client_addr_len);
        if(client_socket == -1){
            perror("Erro: accept().\n");    //Em caso de erro ao aceitar um
client, nao eh necessario matar o programa.
            continue;                   //ou seja, o while pode seguir
esperando por novos accepts
        }else
            printf("Client Accept realizado.\n");

        pid_t pid = fork();
        if(pid == -1){
            perror("Erro: fork()");
            close(client_socket);
            continue;
        }else if(pid == 0){
            close(server_socket);               //fecha o server_socket do fork
            request(data, client_socket, pid, verificacao);
        }else
            close(client_socket);
}
```

Na função request o server verifica se o client é comum ou super, pois isso implica na permissão para realizar algumas das requisições.

Embora do lado do client também tenha estruturas condicionais que verificam o que foi digitado e enviam para o servidor, quem realmente avalia se é válido da entrada é o server. Ele avalia e manda um retorno ao client de acordo com a requisição, essa comunicação termina quando o client digita 0.

```C/C++
if(strncmp(buffer, "1", strlen(buffer)-1) == 0){                    //BUSCA
TODAS AS MUSICAS E INFO
            memset(buffer, 0, MAXSTR);
            printAllDataInfo(data, client_socket);

        }else if(strncmp(buffer, "2", strlen(buffer)-1 ) == 0 ){
//BUSCA MUSICA POR UM ID INFORMADO
            memset(buffer, 0, MAXSTR);
```

```
            strcpy(buffer, "Digite o id:");
            send(client_socket, buffer, strlen(buffer),0);
            memset(buffer, 0, MAXSTR);

            recv(client_socket, buffer, MAXSTR, 0);

            printf("Client pid %d - enviou id: %s", getpid(), buffer);

            //printEssencialData(data, int client_socket, int flag, int id, int
year, char *lang, char *style)
            printEssencialData(data, client_socket, 0, atoi(buffer), 0, NULL,
NULL);
            memset(buffer, 0, MAXSTR);


        }
```

Um detalhe implementação importante foi a escol

```C/C++
//TRECHO DA FUNCO writeSong() no arquivo fileSetingAndLoad.h
int n = nSongsFromCSV(), newN;
if(n == 0){
    data = (Data *)malloc(1* sizeof(Data));
}else if(n >= 1){
    newN = n;
    data = (Data *)realloc(data, (newN+1) * sizeof(Data));
}
```

8. Discussão

No decorrer do desenvolvimento do projeto pode-se notar que algumas escolhas teriam impactos significativos no desempenho quando múltiplos clientes estivessem fazendo requisições ao mesmo tempo.

Optou-se por armazenar os dados em arquivos .csv pela facilidade de manipular os arquivos, mas essa tarefa se mostrou trabalhosa e também impacta no desempenho. Por esse motivo é usado um vetor de struct para armazenar todos os dados e facilitar as buscas e consultas.

Ainda existem aspectos que podem ser melhorados, tal como a busca, uma vez que os os elementos são sempre inseridos no final do vetor e o id que normalmente é a chave, sempre está em ordem crescente. Sendo assim poderia ter sido usado outra estrutura de busca para quando a chave fosse o id, tal como a busca binária. Tal como poderíamos ter criado um banco de dados sql, que seria um jeito mais robusto de manipular as inserções e exclusões, como também as buscas.

Para evitar problemas de diferenças entre os dados da memória e os dados dos .csv a cada alteração como cadastramento e exclusão de música, é necessário alterar todos arquivos .csv e num cenário no qual vários clientes estão fazendo alterações, isso será um problema.

9. Conclusão

O objetivo de implementar um client/server passando pelas etapas de criação dos sockets até a etapa de envio e recebimento de mensagens. O sistema é capaz de responder às requisições do cliente e armazenar dados vindos do cliente no banco de dados do servidor. Embora o desempenho possa ser afetado, para poucos clientes não é perceptível perda de desempenho.

A partir desse projeto pode-se notar quão importante e robusto é o sistema de redes e o processo de request e responses entre client e server. Mesmo tarefas simples demandam alto grau de robustez e quão importantes são para o uso diário.

10. Referências

As principais referências:

a. Massachusetts Institute of Technology
   https://web.mit.edu/6.031/www/fa19/classes/23-sockets-networking/

b. Treina Web
   https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets

c. Beej's Guide to Network Programming
   https://beej.us/guide/bgnet/html/

Bem como as aulas da disciplina de redes.