UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

RELATÓRIO DO PROJETO 2 - SOCKET UDP
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 UDP. 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 2, foi desenvolver um cliente e um servidor, utilizando como base o projeto 1 que utilizava sockets TCP, no entanto no projeto 2 utilizaria-se sockets UDP para realizar a comunicação entre eles os processos.

A proposta do projeto era criar um sistema cliente/servidor, no qual o servidor armazenaria dados de músicas e os arquivos mp3 das músicas, o cliente faria requisições sobre os dados dessas músicas, tendo ainda uma opção de download, o papel do servidor seria atender as requisições do client.

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 sob os protocolos TCP e UDP.

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. O o envio e recebimento de mensagens deveria ser realizado sob o protocolo de transmissão TCP(Transmission Control Protocol) e UDP(User Datagram Protocol).

Haviam duas possibilidades de desenvolvimento e a opção escolhida foi a que implementaria 2 consultas sob o protocolo TCP e 1 função de transmissão de arquivo sob UDP.

Diferentemente do projeto 1 no projeto 2 seria necessário apenas 2 funções de listagem dos dados que haviam sido salvos no projeto 1, utilizando o protocolo TCP e mais uma função de download das músicas que foram cadastradas no projeto 1, o download do arquivo .mp3 utilizando o protocolo UDP.

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()$_{TCP}$ - abre um 'file descriptor' que será usado para a comunicação TCP
bind()$_{TCP}$ - atribui o socket anteriormente criado a um ip e uma porta
setsockopt() - configuração para utilizarmos a porta mais de uma vez e para fazer mais de um bind na mesma porta
listen()$_{TCP}$ - coloca o socket em modo de espera, esperando uma conexão do cliente.
socket()$_{UDP}$ - abre um 'file descriptor' que será usado para a comunicação UDP
bind()$_{UDP}$ - atribui o socket anteriormente criado a um ip e uma porta, nesse caso específico, um bind para a mesma porta e ip que fizemos o bind()$_{TDP}$
accept() - estabelece a conexão entre o cliente e 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);

//Reuseaddr - permite que use novamente uma porta que foi usada poucos
instantes antes
if(setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse))
== -1)...

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

//Socket UDP
udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

//Bind UDP
if(bind(udp_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) ==
-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, udp_socket, client_addr, 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;

Tanto para o TCP quanto para o UDP, sendo que no client tudo é feito no protocolo TCP, somente na opção de download que criamos o socket UDP, e fechamos assim que a transferência é concluída.

```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)...

//Na opcao de DOWNLOAD - Socket UDP
udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

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

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 4 funções **send() e recv() sob o protocolo TCP**, e **sendto() e recvfrom() sob o protocolo UDP.** As quatro funções exercem papéis parecidas dentro do contexto de cada protocolo.

O client e o server 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

ESCOLHA UMA OPCAO:
[1] - Listar todas as musicas e informacoes.
[2] - Listar musicas por estilo musical
[3] - Fazer download
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. As mensagens no terminal do servidor são dessa forma:

```
Socket tcp criado com sucesso.
Bind realizado com sucesso.
Server tcp is listening --- Port: 9877
Socket udp criado com sucesso.
Bind realizado com sucesso.
Client Accept realizado.
Client pid 42938 - opt: 1
Client Accept realizado.
Client pid 42940 - opt: 2
Client pid 42940 - enviou ano: Rnb
Client pid 42938 - opt: 3
Client pid 42938 - enviou id: 0
Transferencia concluida - 'Cidade Maravilhosa.mp3' para o Client pid: 42938
Client pid 42938 - opt: 3
Client pid 42938 - enviou id: 1
Transferencia concluida - 'Asa Branca.mp3' para o Client pid: 42938
Client pid 42940 - opt: 3
Client pid 42940 - enviou id: 4
Transferencia concluida - 'Garota de Ipanema.mp3' para o Client pid: 42940
Client pid 42940 - opt: 0
```

4. Compilação

Para compilar e executar no mesmo computador:
Abra 2 terminais no diretório da pasta projeto2/
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 para executar o programa: `./client 127.0.0.1`

Para compilar e executar em computadores diferentes:
Os passos iguais aos acima para o terminal que executará o server, a diferença é que no terminal do client que vai executar em outro computador 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`

Makefile
Há um arquivo makefile na pasta projeto2/ para facilitar a compilação dos arquivos.
Caso esteja rodando o server e o client na mesma máquina:
com os 2 terminais abertos no diretório do projeto2/ basta digitar em um dos terminais: `make`
e então é um deles executar o `/server` e no outro `./clientIP`

5. Casos de uso

Caso de Uso 01: “Executar o programa client.c.”

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 client.c .

Fluxo Principal

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

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: ./client IP
        - 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: “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!”

Caso de Uso 03: “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 2 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 04: “Fazer Download de uma música dado seu 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 3 e pressiona enter
- O sistema exibe a mensagem: “Digite o id da musica que deseja baixar”
- O usuário digita uma entrada e pressiona enter.
- O sistema exibe:
  Titulo da musica
  Nome do arquivo: Titulo da musica.mp3
  Download de Titulo da musica.mp3 concluido

Pós-condições

O sistema mostra novamente o menu.

Exceções:

- Usuário digita uma entrada não numérica quando é esperado o id da música desejada
  - O sistema exibe a mensagem: ”Caracter Invalido!”
- Usuario digita um id que nao esta cadastrado no banco de dados
  - O sistema exibe a mensagem: “id ‘x’ nao encontrado!”

Caso de Uso 5: “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.

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++
```

```
├─ projeto2/                <- 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,..."
|    └─ songs/              <- Pasta onde estao armazenadas as musicas
|    |
|    └─ downloads/          <- Pasta onde estao armazenadas as musicas
|                             recem baixadas
|
└──────────────────────────────────────────────────────────────────────
```

Legenda:

- Arquivos presentes na pasta antes de compilar pela primeira vez
- A pasta **songs/** e' o banco de dados, onde os arquivos .mp3 estão armazenados
- A pasta **downloads/** será criada após a primeira vez que a opção 3 for realizada com sucesso.

No projeto 2 os arquivos nSongs.csv, songId.csv e songData.csv já existem, 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 seria utilizada inúmeras vezes, optou-se por manter um arquivo com essa quantidade salva. o arquivo nSongs.csv 'e atualizada a cada inserção ou exclusão de musica. **No entanto, o cadastro e exclusão são funcionalidades do projeto 1.**

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. **No entanto, o cadastro e exclusão são funcionalidades do projeto 1.**
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. **No entanto, o cadastro e exclusão são funcionalidades do projeto 1.**
A estrutura é:

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

songs/:
É uma pasta onde estão os arquivos .mp3 disponíveis para download e que tem suas informações cadastradas no arquivo songData,csv.

downloads/:
É uma pasta 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. **No entanto o cadastro era uma funcionalidade do projeto 1.**
A estrutura é:

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.

As transmissões das músicas via socket udp foram feitas utilizando uma estrutura struct Packet, ambos os lados, client e server possuem essa struct definida.

```
C/C++
typedef struct Packet{
    int id;
    char data[MAXSTR];
}Packet;
```

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);
        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);
            request(data, client_socket, udp_socket, client_addr, pid,
verificacao);
        }else
```

```
            close(client_socket);
}
```

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 válida a 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++
//############ SERVER ############
if(strncmp(buffer, "1", strlen(buffer)-1) == 0){
//BUSCA TODAS AS MUSICAS E INFO
            .
            .
            .
}else if(strncmp(buffer, "2", strlen(buffer)-1 ) == 0){
//BUSCA MUSICAS POR ESTILO MUSICAL
            .
            .
            .

}else if(strncmp(buffer, "3", strlen(buffer)-1 ) == 0){
//OPCAO DE DOWNLOAD
            .
            .
            .
}
```

A maior parte da comunicação entre server e client é feita sob socket TCP. O socket UDP é usado para fazer a transmissão do arquivo que o cliente optou, e nesse caso, tanto as mensagens de texto, quanto os dados da música são enviadas em UDP.

O protocolo UDP não garante confiabilidade, portanto os dados podem não chegar e caso cheguem, podem chegar fora de ordem. Por esse motivo, implementamos um checksum simplificado, ou seja, o client envia um ACK como resposta ao server.

Funcionamento simplificado:

Através da Struct Packet o server envia um identificador e os dados lidos do arquivo .mp3, o primeiro identificador enviado é o `packet.id==0`.

No lado do client a variável `check == -1`, o client recebe o Packet e verifica se `(packet.id - check) == 1`, se sim, escreve no arquivo .mp3 do client, e envia de volta para o server o id recebido.

O server neste momento verifica se o id recebido de volta é igual ao que foi enviado, se sim, o cliente recebeu o pacote corretamente, então incrementa o valor do id, e envia a próxima leitura ao cliente; se não, envia novamente o mesmo pacote e aguarda o retorno do cliente;

Isso garante que todos os pacotes sejam entregues e na ordem correta.

8. Discussão

Tivemos muitos problemas com a transmissão UDP, pois a princípio não estavam funcionando os envios, tentamos com a função select() para conseguir lidar com a transmissão UDP e TCP ao

mesmo tempo, mas voltamos para o que gostaríamos anteriormente, que era: no server os socket TCP e UDP serem aberto no início da execução e no client o socket TCP ser aberto no início da comunicação e o socket UDP só ser aberto no momento do recebimento da música, após isso o socket UDP seria fechado e a comunicação voltaria a ser pelo socket TCP.

Como herança do projeto 1 a estrutura de ler os arquivos .csv se manteve, no entanto não realizamos nenhum cadastro ou exclusão, logo em termos de desempenho não realizamos atualizações nos arquivos .csv e nem nos dados carregados para a memória. Com isso, o projeto 2 troca mensagens em TCP e UDP e a principal função do sistema é realizar as buscas e retornar uma resposta ao cliente, logo o desempenho tem sua maior dificuldade no algoritmo escolhido para a busca, isso desconsiderando o fato de que temos que transmitir o arquivo de música e o que também leva tempo.

Após ter as transmissões funcionando, o grande problema foi entender o motivo de as músicas estarem chegando apenas metade do arquivo, e nesse ponto implementar o sistema de ACK foi essencial no entanto, mesmo garantido que todos os pacotes fossem transmitidos e recebidos, o arquivo final apresentava pequenos sons de 'glitch', ou seja, mesmo garantido a entrega do pacote, algumas perdas foram percebidas, e infelizmente não conseguimos corrigir.

9. Comparação entre o Projeto 1 - TCP e o Projeto 2 - TCP/UDP

O projeto 1 necessitou de mais atenção, devido a implementação de todo sistema de criação e alteração dos arquivos .csv, como também o carregamento desses dados dos .csv's para a memória do programa que aumentaram a complexidade do desenvolvimento, e só a partir disso que desenvolvemos a comunicação entre o server e o client sob socket TCP. A escolha por armazenar os dados em arquivos .csv trouxeram uma dificuldade, mas que a princípio se mostrou mais simples que fazer utilizando banco sql.

O projeto 2 utilizou toda a base criada no projeto 1 e isso fez com que o desenvolvimento fosse mais rápido, principalmente porque o server TCP já funcionava, e bastava implementar a função de download utilizando socket UDP. Os arquivos .csv no projeto 2 são os mesmos do projeto 1 e já estão criados, e não realizamos nenhuma alteração. Então a preocupação passou a ser: conseguir enviar dados utilizando o socket UDP e conseguir receber as músicas de forma satisfatória. Uma vez que os elementos são sempre inseridos com id maior no projeto 1(temos a garantia que as músicas sempre estão em ordem crescente pelo id), o projeto 2 utilizou essa característica para melhorar a busca, deixando de ser linear e passando a ser binária. O número de músicas é muito pequeno para ter diferença notável entre a busca linear e busca binária, mas pensando num cenário com milhares de músicas, teríamos a busca linear no pior caso sendo `O(n)` e a busca binária no pior caso sendo `O(lg(n))` que é muito melhor .

Em relação ao **tamanho do código**, mantivemos todas as funções do projeto 1, embora nem todas sejam utilizadas no projeto 2, e adicionamos mais 2 funções.

A opção 3 do menu que é enviado ao usuário que é referente ao download é tratada por um if, tanto no server quanto no client, e esse if especificamente no server é uma função muito grande, quebrar essa função em mais partes, modularizando as tarefas, trariam mais clareza e simplicidade para o código.

O projeto 2 e o projeto 1 são **confiáveis**, em relação a tratamento de erros, entradas incorretas e quando houver falha no envio de um pacote de dados da música, o ACK que foi implementado garante a entrega da música ao cliente. O projeto 1 tinha muito mais funcionalidades e portanto muitos pontos que poderiam fazê-lo apresentar comportamento inesperado, problemas que foram tratados dentro do que foi observado, o projeto 2, mais simples que o primeiro, se mostrou robusto

para tratar os erros, embora as músicas cheguem com algumas perdas de dados que podem ser percebidas ao ouvir a música ou utilizando o diff através do terminal

10. Conclusão

O objetivo do projeto 2 era implementar um client/server passando pelas etapas de criação dos sockets até a etapa de envio e recebimento de mensagens, e tendo uma opção de download de arquivo .mp3. O sistema é capaz de responder às requisições do cliente utilizando TCP e UDP, sendo que o UDP foi usado somente na etapa de download.

Através desse projeto, pode-se notar as diferenças entre o protocolo TCP e o UDP, cada um com suas vantagens e desvantagens, sendo que o UDP é muito melhor quando o dado precisa ser transmitido rapidamente e quando não há grandes problemas em perder alguns pacotes, já o TCP devido ao handshake, controle de fluxo, sistema de ACK's, etc., é confiável e muito melhor de ser utilizado quando não se pode perder dados na transmissão, no entanto com isso acaba sendo mais lento que o UDP.

O objetivo foi atingido satisfatoriamente, o sistema realiza todas as funcionalidades propostas, exercitando ambos os protocolos e conseguindo notar suas diferenças empiricamente.

11. 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/

d. Scaler
https://www.scaler.com/topics/udp-server-client-implementation-in-c/

e. Geeks for geeks
https://www.geeksforgeeks.org/tcp-and-udp-server-using-select/

f. Die Net - Linux man page
https://linux.die.net/man/3/fd_set

Bem como as aulas da disciplina de redes.