Projeto 2 – MC833 – 1s 2021

Sistema Multi Cliente e Servidor UDP

Relatório

Nome: Victor Rigatto

RA: 178068

Introdução

Este relatório tem por objetivo descrever o processo de implementação de um sistema cliente-servidor por protocolo UDP, após o desenvolvimento do mesmo sistema por protocolo TCP, desenvolvidos em linguagem C, com funções de registro e armazenamento de informações de usuários, que serão melhor detalhadas nos próximos tópicos.

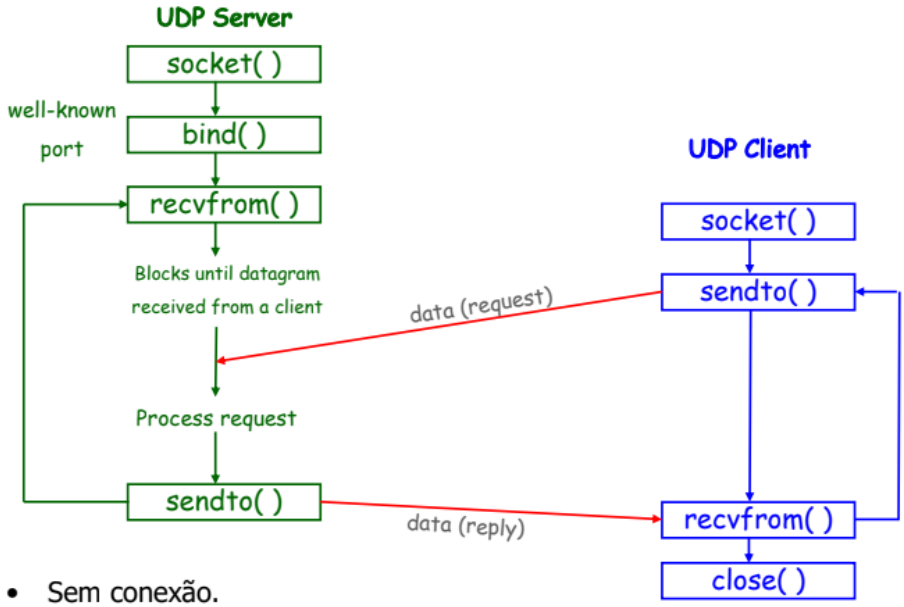
O desenvolvimento se deu através do entendimento do funcionamento da comunicação UDP e a implementação de socket nos aplicativos cliente e servidor, bem como as funções de leitura e escrita através dos sockets, e as respectivas funções de registro, exibição e manipulação de dados no lado servidor.

Como fonte de informações para a realização desse projeto, pode-se mencionar principalmente os slides disponibilizados em aula e informações adicionais obtidas na internet.

Descrição geral

O sistema foi desenvolvido em ambiente Linux, utilizando-se a ferramenta WSL2 do Windows 10 em conjunto com o Visual Studio Code. O sistema consiste de 2 arquivos, denominados client.c e server.c, aplicativos cliente e servidor respectivamente, além de data.dat, arquivo binário de registros criado automaticamente pelo servidor.

O aplicativo cliente foi estruturado para inicialmente tentar criar um socket UDP na porta arbitrária 5035. Os testes foram realizados rodando o cliente e o servidor localmente. Caso essa tentativa falhe, o programa informa um erro e encerra sua execução. A comunicação com o servidor se da pela implementação padrão de socket UDP, sem conexão, exemplificada nos slides da disciplina, utilizando a biblioteca padrão socket.h. De maneira análoga, o aplicativo servidor foi estruturado para inicialmente criar um socket de escuta na porta arbitrária 5035, para todos os endereços que quiserem enviar datagramas. Um esquema da conexão UDP e sua implementação em C encontram-se abaixo.



- Sem conexão.

Esquema de comunicação UDP sem conexão entre cliente e servidor

```
1 #include "unp.h"
2
3 int main(int argc, char **argv)
4 {
5         int sockfd;
6         struct sockaddr_in servaddr, cliaddr;
7         sockfd = Socket (AF_INET, SOCK_DGRAM, 0);
8         Bzero (&servaddr, sizeof(servaddr));
9         servaddr.sin_family = AF_INET;
10        servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
11        servaddr.sin_port = htons (SERV_PORT);
12        Bind (sockfd, (SA *) &servaddr, sizeof(servaddr));
13        dg_echo (sockfd, (SA *) &cliaddr, sizeof(cliaddr));
14 }
```

Estrutura padrão com informações do servidor para socket de datagrama

Abaixo encontra-se uma foto da implementação em C do processo de criação de socket e conexão com o servidor do aplicativo cliente. Os comentários descrevem as funções utilizadas.

```c
//Limpa estrutura do socket
bzero(&servaddr, sizeof(servaddr));

//Socket de domínio de internet IPv4, UDP e protocolo padrão (0)
//Cria socket para acesso à camada de transporte, retorna um descritor de socket
sockfd=socket(AF_INET, SOCK_DGRAM, 0);

if (sockfd < 0){
    printf("Nao pode criar socket.\n");
    return 0;
}

//Especifica o domínio de internet IPv4 e conexão na porta 5035
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(5035);

//Conecta socket
connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
```

Implementação de socket do cliente em C

Após o sucesso da criação do socket UDP, o aplicativo cliente exibe um menu com as funções disponíveis. Para cada opção, a função com um número identificador é transmitida para o servidor e, em seguida, as informações pertinentes. Ao contrário do que foi realizado no trabalho TCP, optou-se por transmitir e receber os dados do registro em estruturas inteiras, presupondo-se que o cliente e servidor são executados em arquiteturas compatíveis. Essa fraqueza no sistema pode ser solucionada através de serialização, o que não foi implementado neste trabalho. Esse processo se repete até que o cliente escolha encerrar o programa.

Mensagens podem ser trocadas entre o cliente e servidor utilizando-se das funções sendto(envia datagramas pelo socket) e recvfrom (recebe datagramas pelo socket).

```c
//Estrutura para os dados do arquivo de registros
struct data{
    char email[20];
    char nome[20];
    char sobrenome[20];
    char cidade[20];
    char curso[20];
    char habilidades[40];
    char experiencias[40];
};
struct data Data;
```

Estrutura de registro iguais no cliente e no servidor para transmissão

```c
//Preenche estrutura do registro
printf("Insira um email: ");
fgets(Data.email, 1000, stdin);
printf("Insira um nome: ");
fgets(Data.nome, 1000, stdin);
printf("Insira um sobrenome: ");
fgets(Data.sobrenome, 1000, stdin);
printf("Insira uma cidade: ");
fgets(Data.cidade, 1000, stdin);
printf("Insira um curso: ");
fgets(Data.curso, 1000, stdin);
printf("Insira as habilidades: ");
fgets(Data.habilidades, 1000, stdin);
printf("Insira as experiencias: ");
fgets(Data.experiencias, 100, stdin);
//Envia estrutura do registro para o servidor
sendto(sockfd, &Data, sizeof(struct data), 0, (struct sockaddr*)&servaddr, len);
```

Implementação do envio de um registro em estrutura completa pelo cliente

A implementação do servidor é semelhante ao cliente. A primeira etapa é a criação de um socket para escuta de datagramas de qualquer endereço de cliente. O recebimento de uma função inicia os próximos processos dentro do servidor.

```c
//Limpa estrutura do socket
bzero(&servaddr, sizeof(servaddr));

//Socket de domínio de internet IPv4, UDP e protocolo padrão (0)
//Cria socket para acesso à camada de transporte, retorna um descritor de socket
sockfd = socket(AF_INET, SOCK_DGRAM, 0);

if (sockfd < 0){
    printf("Nao pode criar socket.\n");
    return 0;
}

//Especifica o domínio de internet IPv4 e recebe qualquer endereço na porta 5035
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htons(INADDR_ANY);
servaddr.sin_port = htons(5035);

//Aloca um "nome" (endereço local) para o socket
//sockfd é o descritor do socket criado
bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```

Implementação de socket do servidor em C

A escolha e subsequente envio das funções, seguidos de eventuais envios ou recebimentos de estruturas, está representado abaixo.

```c
case 1:
    printf("Opção [1] recebida.\n");
    //Recebe estrutura do registro do cliente
    n = recvfrom(sockfd, &Data, sizeof(struct data), 0, (struct sockaddr*)&cliaddr, &len);
    //Executa função de registro em arquivo
    writeFile(Data);
    break;
```

Implementação do recebimento de uma estrutura de registro completa pelo servidor

```c
//Função para escrever os dados no arquivo de registros
void writeFile(struct data Data){
    FILE *file;
    file = fopen("data.dat","a");
    if(file){
        //Escreve dados no arquivo binário estrutura por estrutura
        fwrite (&Data, sizeof(struct data), 1, file);
    } else{
        printf("Nao pode abrir arquivo.\n");
    }
    fclose(file);
}
```

Implementação da escrita em arquivo binário da estrutura completa recebida

Casos de uso

Abaixo encontram-se fotos de casos de uso do sistema cliente-servidor. Adotou-se a exibição do recebimento de funções no console do servidor, para fins de visualização.

```
[1] Cadastrar novo perfil.
[2] Adicionar experiencia profissional.
[3] Listar um perfil.
[4] Listar todos os perfis por curso.
[5] Listar todos os perfis por habilidade.
[6] Sair.
Escolha uma opcao:
```

Terminal do cliente mostrando o menu e servidor aguardando um datagrama de função

```
Escolha uma opcao:
3
Insira um email: oi@oi.com

Email: oi@oi.com
Nome: victor
Sobrenome: rigatto
Cidade: campinas
Curso: engenharia
Habilidades: algumas
Experiencias: algumas
```

```
Opção [3] recebida.
```

Terminal do servidor confirmando o recebimento da opção 3 e cliente recebendo dados

Armazenamento

Optou-se por utilizar a manipulação direta de arquivos binários em C, abrindo, lendo e escrevendo no arquivo definido como data.dat, no mesmo diretório do aplicativo servidor. O servidor recebe do cliente os dados no formato de estrutura, com todos os dados de registro inseridos. Essa estrutura é então inteiramente escrita no arquivo, e posteriormente inteiramente lida em outras funções.

```c
//Função para buscar os dados no arquivo de registros
void searchFile(char *str){
    FILE *file;
    file = fopen("data.dat","r");
    if(file){
        //Ler dados do arquivo binário estrutura por estrutura
        while(fread(&Data, sizeof(struct data), 1, file)){
            //Compara strings da busca desejada pelo cliente, se encontrar envia os dados para o cliente
            if(strcasecmp(Data.email,str) == 0 || strcasecmp(Data.curso,str) == 0 || strcasecmp(Data.habilidades,str) == 0){
                //Envia estrutura do registro para o cliente
                sendto(sockfd, &Data, sizeof(struct data), 0, (struct sockaddr*)&cliaddr, len);
            }
        }
    } else{
        printf("Nao pode abrir arquivo.\n");
    }
    fclose(file);
}
```

Função para busca dos registros em arquivo binário

Conclusão

A realização desse projeto permitiu um entendimento prático importante da teoria que estudamos em MC832, especificamente da comunicação TCP e, posteriormente neste trabalho, a comunicação UDP entre cliente e servidor. Assim como no primeiro trabalho, a implementação teve desafios de nível técnico interessantes, como a manipulação dos dados para transmissão, que neste trabalho levou-se em consideração as limitações da comunicação UDP, que não estabelece conexão e não garante o envio completo dos dados como no TCP. Um passo importante nessa direção seria a possibilidade de transmitir estruturas inteiras e checar o seu recebimento, que pode ser feita através de serialização, para fins também de compatibilidade entre plataformas e arquiteturas distintas.

Outro ponto importante foi a elaboração do armazenamento, análoga ao primeiro trabalho As operações com arquivos binários em C è um tanto quanto complexa, especialmente para funções de registro e alteração. Um sistema com banco de dados, como o SQL relacional, poderia ser mais interessante, porém acreditou-se não ser necessário.

Diferentemente do primeiro trabalho, esta implementação também permite a utiliação de múltiplos clientes, pois não há o estabelecimento de uma conexão entre eles e o servidor, há apenas o envio de datagramas e o seu recebimento.