Nome: Isabela Costa Souza

Matrícula: 2020054536

Explicação do Código (server)

```
static double haversine(double lat1, double lon1,
                        double lat2, double lon2) {
    double dLat = (lat2 - lat1) * M_PI / 180.0;
    double dLon = (lon2 - lon1) * M_PI / 180.0;
    lat1 = (lat1) * M_PI / 180.0;
    lat2 = (lat2) * M_PI / 180.0;

    double a = pow(sin(dLat / 2), 2) +
               pow(sin(dLon / 2), 2) * cos(lat1) * cos(lat2);
    double rad = 6371;
    double c = 2 * asin(sqrt(a));
    return rad * c * 1000;
}
```

Esse código implementa a fórmula de Haversine, que é uma fórmula utilizada para calcular a distância entre dois pontos em uma esfera, como a Terra. Essa fórmula é comumente usada em geolocalização para calcular distâncias entre coordenadas geográficas, como latitude e longitude.

A função recebe as coordenadas de latitude e longitude de dois pontos como entrada, calcula as diferenças em radianos entre as latitudes e longitudes dos dois pontos e o termo interno da fórmula de Haversine é calculado usando as diferenças de latitude e longitude, bem como os cossenos das latitudes.

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Uso: %s <ipv4 | ipv6> <porta>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int server_fd, new_socket, valread;
    struct sockaddr_in6 address6;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen;
```

Aqui, o servidor verifica se o número de argumentos da linha de comando é correto. Se não for, exibe uma mensagem de uso e encerra a execução. Também são sentadas as variáveis usadas para configurar o socket do servidor e manipular endereços de IP.

```
    if (strcmp(argv[1], "ipv4") == 0) {
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
            perror("socket failed");
            exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(atoi(argv[2]));
        addrlen = sizeof(address);
    } else if (strcmp(argv[1], "ipv6") == 0) {
        if ((server_fd = socket(AF_INET6, SOCK_STREAM, 0)) == 0) {
            perror("socket failed");
            exit(EXIT_FAILURE);
        }
        address6.sin6_family = AF_INET6;
        address6.sin6_addr = in6addr_any;
        address6.sin6_port = htons(atoi(argv[2]));
        addrlen = sizeof(address6);
    } else {
        fprintf(stderr, "Parâmetro inválido. Use 'ipv4' ou 'ipv6'.\n");
        exit(EXIT_FAILURE);
    }
```

Essa parte do código verifica se o primeiro argumento passado na linha de comando é "ipv4" ou "ipv6". Se for "ipv4", o código executa as configurações para endereços IPv4. Se for "ipv6", executa as configurações para endereços IPv6. Se for qualquer outra coisa, o programa emite uma mensagem de erro e sai.

Quando entra em algum dos ifs, um socket é criado com a função socket(). O primeiro argumento AF_INET ou AF_INET6 indica o tipo de endereço (IPv4 ou IPv6). O segundo argumento SOCK_STREAM indica que é um socket TCP. O terceiro argumento 0 indica que o protocolo padrão para o tipo de socket e endereço especificado será usado.

Dependendo do tipo de endereço (IPv4 ou IPv6), as configurações de endereço e porta são atribuídas às estruturas address ou address6. O endereço IP é definido como INADDR_ANY ou in6addr_any, o que indica que o servidor irá vincular-se a todas as interfaces disponíveis no sistema. A porta é definida com base no segundo argumento passado na linha de comando.

```
if (setsockopt(server_fd, SOL_SOCKET,
               SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
       perror("setsockopt");
       exit(EXIT_FAILURE);
}

if (strcmp(argv[1], "ipv4") == 0) {
   if (bind(server_fd,
            (struct sockaddr*)&address, sizeof(address))<0) {
           perror("bind failed");
           exit(EXIT_FAILURE);
       }
} else if (strcmp(argv[1], "ipv6") == 0) {
       if (bind(server_fd,
                (struct sockaddr*)&address6, sizeof(address6))<0) {
           perror("bind failed");
           exit(EXIT_FAILURE);
       }
}

if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
}
```

Essa parte do código configura o socket do servidor para reutilizar endereços e portas, faz o bind do socket a um endereço e porta específicos e o coloca em um estado passivo de escuta por conexões entrantes.

O primeiro if configura o socket para reutilizar o endereço e a porta, permitindo que o socket seja vinculado a um endereço e porta que estejam em uso por outro socket. Isso é útil para evitar erros como "Endereço já em uso" ao reiniciar o servidor.

Os próximos ifs associam o socket a um endereço e porta específicos. Se o tipo de endereço for IPv4, o socket é vinculado à estrutura address, que contém as informações do endereço IPv4. Se for IPv6, o socket é vinculado à estrutura address6, que contém as informações do endereço IPv6.

Por fim, o último if coloca o socket em um estado passivo de escuta por conexões entrantes. O argumento 3 especifica o número máximo de conexões pendentes que podem ser enfileiradas para o socket. Se houver mais conexões entrantes do que o número especificado, elas serão recusadas.

```
while(1) {
    printf("Aguardando solicitação\n");
    if ((new_socket = accept(server_fd,
                             (struct sockaddr *)&address,
                             (socklen_t*)&addrlen))<0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    int acceptDrive = 0;
    printf("Corrida disponível:\n0 - Recusar\n1 - Aceitar\n");
    scanf("%d", &acceptDrive);

    char* message = "Não foi encontrado um motorista";
    if(acceptDrive == 1) message = "Motorista encontrado";
    send(new_socket, message, strlen(message), 0);
```

Essa parte do código aguarda uma conexão e quando a mesma chega, dá ao motorista a opção de aceitar ou recusar a corrida. A resposta do motorista é enviada ao cliente.

```
    if(acceptDrive == 1) {
        Coordinate coordCli;
        read(new_socket, &coordCli, sizeof(Coordinate));

        double distancia = haversine(coordCli.latitude,
                                     coordCli.longitude,
                                     coordServ.latitude,
                                     coordServ.longitude);
        if(distancia <= 0) {
            printf("O motorista chegou!\n");
            send(new_socket, "O motorista chegou!",
                 strlen("O motorista chegou!"), 0);
            break;
        }
```

Se o motorista aceita a corrida, ele espera o cliente enviar suas coordenadas e calcula a distância entre os dois. Se a mesma for zero, o motorista chegou.

```
while (distancia > 0) {
   if (distancia <= 400) {
        printf("O motorista chegou!\n");
        send(new_socket, "O motorista chegou!",
             strlen("O motorista chegou!"), 0);
        break;
   } else {
       distancia -= 400;
       char message[50];
       sprintf(message, "Motorista a %.2f metros", distancia);
       send(new_socket, message, strlen(message), 0);
       printf("%s\n", message);
       sleep(2);
   }
```

A cada dois segundos, o motorista decresce a distância em 400 metros e envia a mesma para o usuário, até chegar a zero e o último envio de mensagem ser “O motorista chegou!” e o socket fechar a conexão.

Explicação do Código (client)

```
int main(int argc, char *argv[]) {
   while(1) {
       int callDriver;
       printf("0 - Sair\n1 - Solicitar Corrida\n");
       scanf("%d", &callDriver);

       if(callDriver == 0) {
           return 0;
       }

       if (argc != 4) {
           fprintf(stderr, "Uso: %s <ipv4 | ipv6> <IP do servidor>
                   <porta>\n", argv[0]);
           exit(EXIT_FAILURE);
       }
```

O código espera a resposta do usuário sobre a corrida, se ele escolher sair, o programa termina. Em seguida, no segundo if, é verificado o formato dos argumentos passados ao programa.

```
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            printf("\n Erro na criação do socket\n");
            return -1;
        }

        if (strcmp(argv[1], "ipv4") == 0) {
            memset(&serv_addr, '0', sizeof(serv_addr));
            serv_addr.sin_family = AF_INET;
            serv_addr.sin_port = htons(atoi(argv[3]));
            addrlen = sizeof(serv_addr);
            if(inet_pton(AF_INET, argv[2], &serv_addr.sin_addr)<=0) {
               printf("\nEndereço inválido/Endereço não suportado \n");
               return -1;
            }
        } else if (strcmp(argv[1], "ipv6") == 0) {
            memset(&serv_addr6, '0', sizeof(serv_addr6));
            serv_addr6.sin6_family = AF_INET6;
            serv_addr6.sin6_port = htons(atoi(argv[3]));
            addrlen = sizeof(serv_addr6);
            if(inet_pton(AF_INET6, argv[2],&serv_addr6.sin6_addr)<=0) {
               printf("\nEndereço inválido/Endereço não suportado \n");
               return -1;
            }
        } else {
            fprintf(stderr, "IP inválido. Use 'ipv4' ou 'ipv6'.\n");
            return -1;
        }
```

No primeiro if, um socket é criado usando a função socket(). O primeiro argumento AF_INET especifica que é um socket IPv4, e SOCK_STREAM indica que se trata de um socket TCP. O terceiro argumento 0 indica que o protocolo padrão para o tipo de socket e endereço especificado será usado.

Nos próximos ifs, verifica-se se o primeiro argumento passado na linha de comando é "ipv4" ou "ipv6". Dependendo do tipo de endereço, as configurações de endereço e porta são atribuídas às estruturas serv_addr (para v4) ou serv_addr6 (para v6).

Se o tipo de endereço for IPv4, a estrutura serv_addr é configurada. memset() é usada para limpar a estrutura. AF_INET indica que é um endereço IPv4. htons() converte a porta de formato de host para formato de rede. inet_pton() converte o endereço IP de apresentação para formato binário. Se o tipo de endereço for IPv6, a estrutura serv_addr6 é configurada de forma semelhante, mas para endereços IPv6.

```
        if (strcmp(argv[1], "ipv4") == 0) {
            if (connect(sock, (struct sockaddr *)&serv_addr,
                        sizeof(serv_addr)) < 0) {
                printf("\n Falha na conexão \n");
                return -1;
            }
        } else if (strcmp(argv[1], "ipv6") == 0) {
            if (connect(sock, (struct sockaddr *)&serv_addr6,
                        sizeof(serv_addr6)) < 0) {
                printf("\n Falha na conexão \n");
                return -1;
            }
        }
```

Aqui deve ser estabelecida a conexão com o servidor. Se o tipo de endereço for IPv4, a função connect() é usada para estabelecer uma conexão com o servidor remoto usando o socket sock e a estrutura serv_addr, que contém as informações do endereço IPv4 do servidor. Se a conexão falhar, uma mensagem de erro é exibida e o programa termina com código de erro -1.

Se o tipo de endereço for IPv6, a função connect() é usada de forma semelhante para estabelecer uma conexão com o servidor remoto usando o socket sock e a estrutura serv_addr6, que contém as informações do endereço IPv6 do servidor. Da mesma forma, se a conexão falhar, uma mensagem de erro é exibida e o programa termina com código de erro -1.

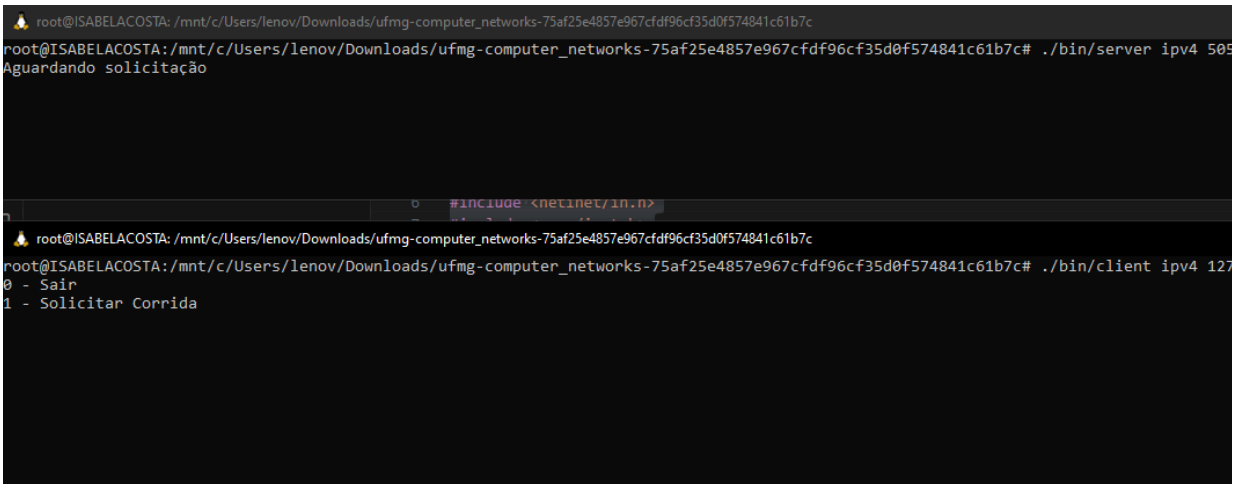
```
        int continueDrive = 0;
        char buffer[1024] = {0};
        while(1) {
            valread = read(sock, buffer, 1024);
            if(strstr(buffer, "Motorista encontrado")) {
                continueDrive = 1;
            } else {
                printf("Não foi encontrado um motorista\n");
            }
            break;
        }
```

O cliente espera a resposta do motorista, se ele aceitar a corrida o código continua, senão o loop é quebrado e o usuário tem que enviar uma nova solicitação.

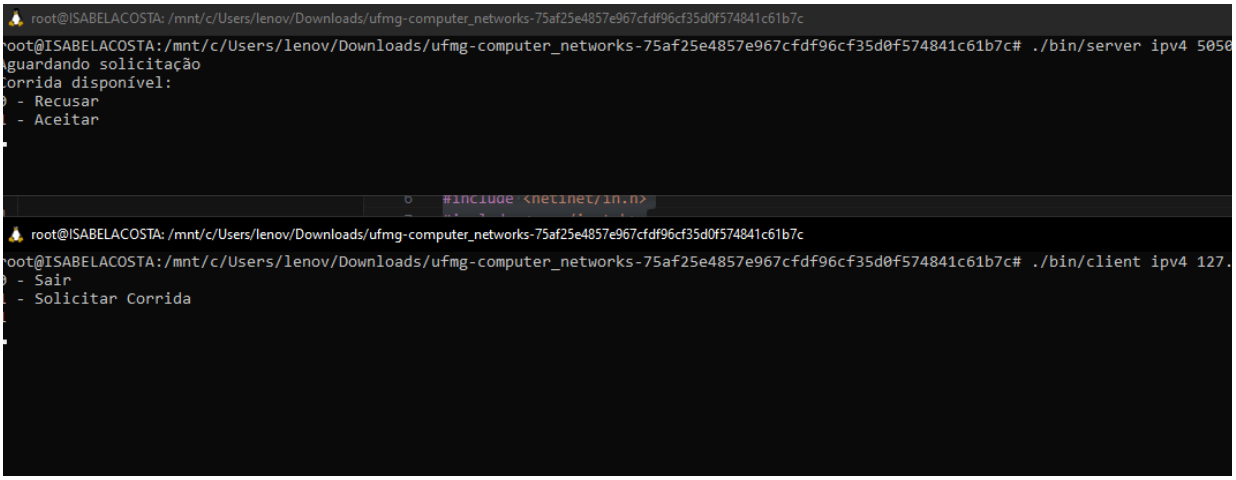
```
        if(continueDrive == 1) {
            send(sock , &coordCli, sizeof(Coordinate) , 0 );

            while(1) {
                valread = read(sock, buffer, 1024);
                if (strstr(buffer, "O motorista chegou!") ||
                    strcmp(buffer, "O motorista chegou!") == 0) {
                    printf("O motorista chegou!\n");
                    return 0;
                }
                printf("%s\n", buffer);
                sleep(2);
            }
        }
```

O cliente envia suas coordenadas para o motorista e a cada dois segundos espera por um update na distância do mesmo, printando-a na tela.

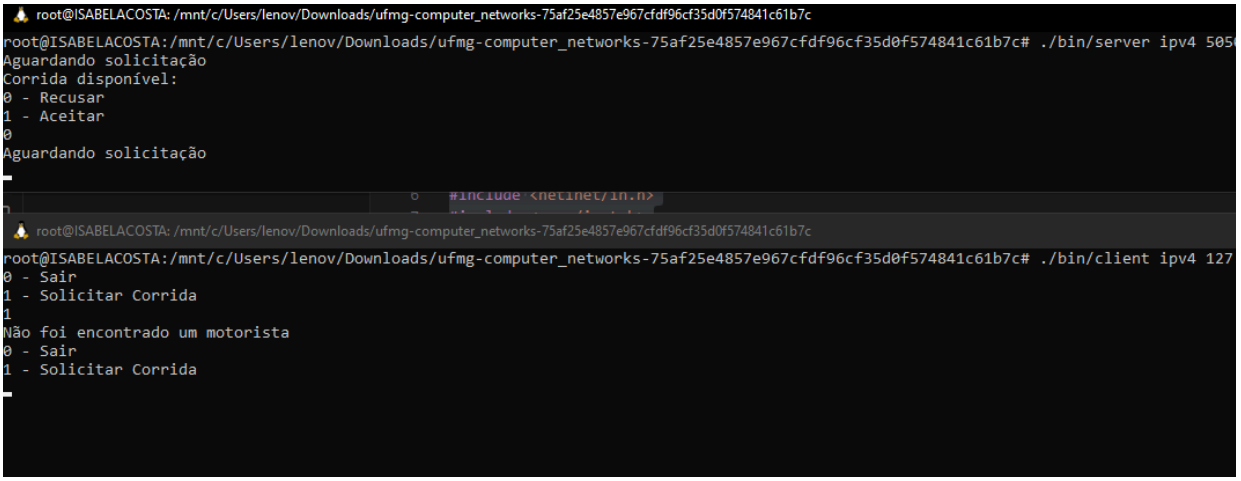
Rodando o código (motorista recusa a corrida)



O cliente se conecta com sucesso ao servidor

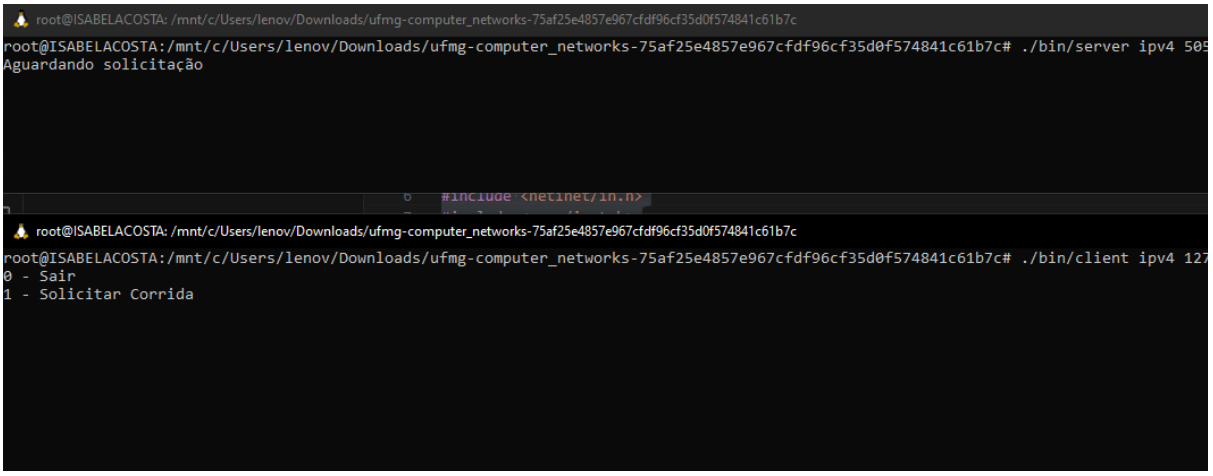


O cliente seleciona “Solicitar corrida” e a corrida aparece para o motorista

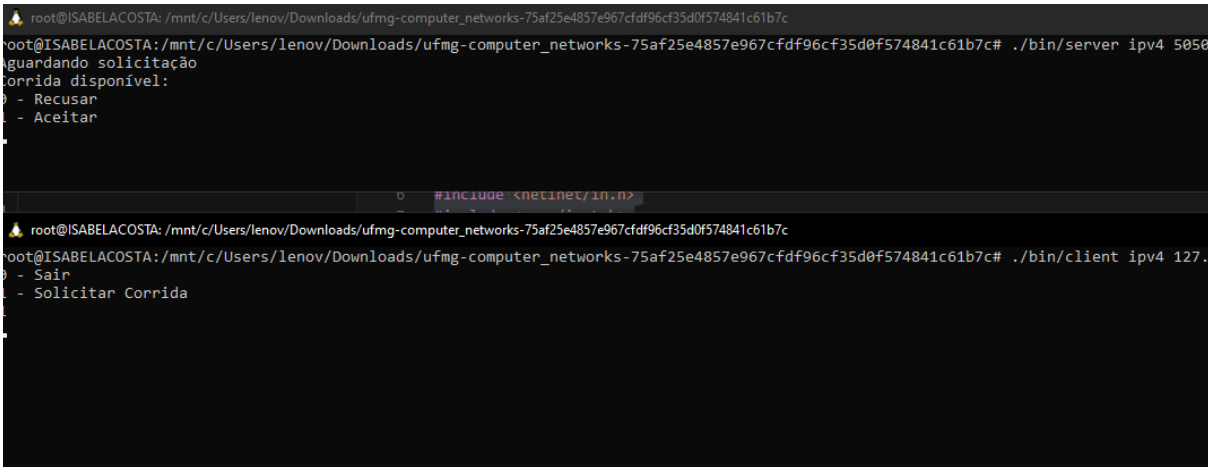


O motorista seleciona “Recusar”, a conexão entre os dois é encerrada, o cliente recebe uma mensagem informado que não foi possivel encontrar um motorista e volta a ter as opções de solicitar uma corrida ou sair do programa

Rodando o código (motorista aceita a corrida)



O cliente se conecta com sucesso ao servidor



O cliente seleciona “Solicitar corrida” e a corrida aparece para o motorista

O motorista aceita a corrida e a mesma começa, sendo que a cada 2 segundos a distância entre o motorista e o cliente diminui, até ele chegar. Quando o motorista chega, ele volta a aguardar solicitação e o cliente sai do programa.