UNIVERSIDADE FEDERAL DE MINAS GERAIS

REDES DE COMPUTADORES

Trabalho Prático 01

JOÃO VITOR FERREIRA / 2021039654

BELO HORIZONTE

2024

SUMÁRIO

CODIGO SERVIDOR 3

Funcionamento do Servidor 3

Detalhes da Conexão 7

CODIGO CLIENTE 9

Funcionamento do Servidor 9

Detalhes da Conexão 11

TESTES 13

IPv4 13

Client 13

Server 21

IPv6 22

Client 22

Server 25

CODIGO SERVIDOR

Este servidor TCP implementado em C, que tem como principal funcionalidade simular a movimentação de um motorista em resposta a solicitações de clientes. O servidor é capaz de criar um socket e escutar por conexões de clientes. Ao estabelecer uma conexão, o servidor recebe as coordenadas geográficas do cliente e calcula a distância entre si e o cliente utilizando a fórmula de Haversine. Posteriormente, o usuário tem a opção de aceitar ou recusar a solicitação do cliente. Em caso de aceitação, o servidor envia ao cliente uma mensagem de confirmação e prossegue enviando atualizações da distância a cada 2 segundos até que a distância seja reduzida a zero. Em caso de recusa, o servidor informa ao cliente sobre a recusa e encerra a conexão. Este servidor suporta tanto o protocolo IPv4 quanto o IPv6.

Funcionamento do Servidor

A função `handle_client()`: `void handle_client(int clientSocket)`,que é chamada para cada cliente que se conecta ao servidor, recebe o descritor do socket do cliente como argumento. A primeira ação desta função é ler as coordenadas do cliente do socket usando a função `recv()` [figura 1]

```
// recv() receives data from the client and stores it in coordCli.
// Returns the number of bytes received or 0 if the connection was closed.
int bytes_received = recv(clientSocket, &coordCli, sizeof(coordCli), 0);
```

Figura 1

Em seguida, ela calcula a distância entre o servidor e o cliente utilizando a função `haversine()` [figura 2]

```
// Calculate distance between server and client
double distance = haversine(coordServ.latitude, coordServ.longitude, coordCli.latitude, coordCli.longitude);
```

Figura 2

Com base na opção escolhida pelo usuário, a função `handle_client()`[figura 3] pode aceitar ou recusar a solicitação do cliente. Se a solicitação for aceita, o servidor envia uma mensagem de confirmação para o cliente usando a função `send()` [figura 3] e entra em um loop que simula a movimentação do motorista, enviando atualizações da distância a cada 2 segundos até que a distância seja zero.

```c
if (option[0] == 1) // If request is accepted
{
    // Send confirmation message to client
    char message[] = "Motorista a caminho";
    send(clientSocket, message, strlen(message) + 1, 0);

    // Send distance updates to client every 2 seconds
    while (distance > 0)
    {
        send(clientSocket, &distance, sizeof(distance), 0);
        distance -= 400;
        sleep(2);
    }

    // Envia uma mensagem de chegada para o cliente
    distance = 0;
    send(clientSocket, &distance, sizeof(distance), 0);
    printf("O motorista chegou!\n");
}
```

Figura 3

Caso a solicitação seja recusada, a função `handle_client()` [figura 4] envia uma mensagem de recusa para o cliente usando novamente a função `send()` [figura 4] e fecha o socket do cliente.

```c
else
{
    // Send arrival message to client
    char message[] = "Não foi encontrado um motorista.";
    send(clientSocket, message, strlen(message) + 1, 0);
    close(clientSocket);
    // TODO libera a porta do server
    printf("Aguardando solicitação.\n");
}
```

Figura 4

Independente da decisão, a função imprime uma mensagem indicando a chegada do motorista ou que está aguardando uma nova solicitação. [figura 5]

Figura 5

A função `create_server_socket_and_listen()`: `void create_server_socket_and_listen(sa_family_t family, int port)` é responsável por criar um socket TCP e configurar o servidor para escutar conexões de clientes. Ela recebe como argumentos a família de endereços (AF_INET para IPv4 ou AF_INET6 para IPv6) e o número da porta.

Primeiramente, a função cria um socket TCP utilizando a função `socket()` [figura 6]

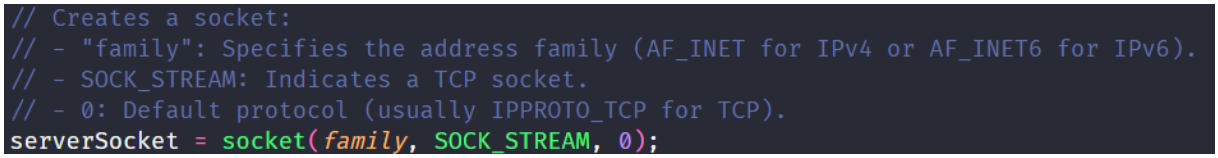


Figura 6

Em seguida, configura a estrutura de endereço do servidor com base na família de endereços especificada e associa o socket ao endereço usando a função `bind()` [figura 7] e [figura 8]

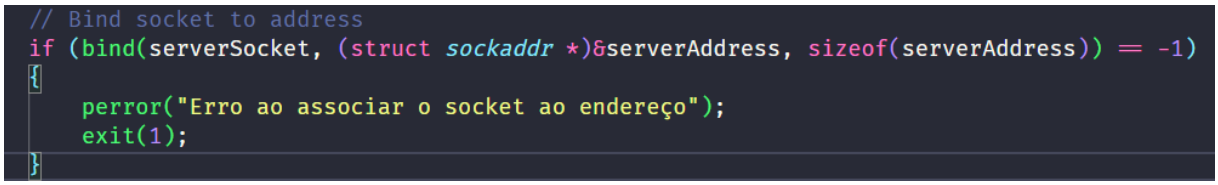


Figura 7

Figura 8

Posteriormente, a função coloca o socket em modo de escuta utilizando a função `listen()` [figura 9] permitindo que o servidor aceite conexões de clientes.

```c
// Listen for connections (maximum of 5 queued connections)
if (listen(serverSocket, 5) == -1)
{
    perror("Erro ao ouvir por conexões");
    exit(1);
}
```

Figura 9

A função então entra em um loop infinito, onde aceita uma conexão de cliente usando a função `accept()`[figura 10] e chama a função `handle_client()` [figura 10] para lidar com a comunicação com o cliente.

```c
if (family == AF_INET)
{
    clientAddressLength = sizeof(clientAddress);
    clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress, &clientAddressLength);
}
else if (family == AF_INET6)
{
    clientAddressLength = sizeof(clientAddress6);
    clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress6, &clientAddressLength);
}

if (clientSocket == -1)
{
    perror("Erro ao aceitar conexão");
    continue;
}

handle_client(clientSocket);
```

Figura 10

A função `main()` `int main(int argc, char **argv)` é a principal função do programa. Ela começa verificando se o número correto de argumentos foi fornecido, especificamente a versão IP e a porta. A versão IP é então validada para garantir que seja "ipv4" ou "ipv6". Após essas verificações iniciais, a função `create_server_socket_and_listen()` [figura 11] é chamada com a família de endereços apropriada e o número da porta passados como argumentos.

```c
// Create server socket and listen for connections based on IP version
if (strcmp(ipVersion, "ipv4") == 0)
{
    create_server_socket_and_listen(AF_INET, port);
}
else if (strcmp(ipVersion, "ipv6") == 0)
{
    create_server_socket_and_listen(AF_INET6, port);
}
```

Figura 11

Detalhes da Conexão

- `socket()`[figura 6]: Esta função é utilizada para criar um socket. Ela recebe três argumentos: a família de endereços (por exemplo, AF_INET para IPv4), o tipo de socket (por exemplo, SOCK_STREAM para TCP) e o protocolo (geralmente 0 para o protocolo padrão). A função retorna um descritor de socket que pode ser usado em chamadas de função subsequentes que operam em sockets.
- `bind()`[figura 7 e 8]: Esta função associa o socket criado a um endereço específico e porta. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para a estrutura de endereço que contém detalhes do endereço e porta, e o terceiro argumento é o tamanho da estrutura de endereço. Isso permite que o servidor receba conexões de clientes nesse endereço e porta especificados.
- `listen()`[figura 9]: Após o bind, esta função coloca o socket em modo de escuta, indicando que o servidor deve aceitar conexões de entrada dos clientes. O primeiro argumento é o descritor do socket e o segundo argumento especifica o número máximo de conexões pendentes que podem ser enfileiradas antes de começarem a serem rejeitadas.
- `accept()`[figura 10]: Esta função é usada para aceitar uma conexão de entrada de um cliente. Ela retorna um novo descritor de socket que pode ser usado para comunicação com o cliente. O primeiro argumento é o descritor do socket de escuta, o segundo argumento é um ponteiro para uma estrutura de endereço que será preenchida com o endereço do cliente, e o terceiro argumento é um ponteiro para o tamanho da estrutura de endereço.

- `send()`[figura 3 e 4]: Esta função é usada para enviar dados através do socket para o cliente conectado. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para os dados a serem enviados, o terceiro argumento é o tamanho dos dados, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes enviados ou -1 em caso de erro.
- `recv()`[figura 1]:Esta função é usada para receber dados do cliente conectado através do socket. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para um buffer onde os dados serão armazenados, o terceiro argumento é o tamanho do buffer, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes recebidos ou -1 em caso de erro.
- `close()`[figura 4 e 5]: Esta função é usada para fechar um descritor de socket, liberando os recursos associados a ele.

CODIGO CLIENTE

Este código implementa um cliente TCP em C que se conecta a um servidor e solicita uma corrida. O cliente começa por criar um socket e tentar conectar-se ao servidor. Uma vez estabelecida a conexão, o cliente envia suas coordenadas para o servidor. Em seguida, o cliente aguarda a resposta do servidor. Se a solicitação de corrida for aceita, o cliente recebe atualizações de distância do servidor até que a distância seja reduzida a zero, indicando a chegada do motorista. Se a solicitação for recusada, o cliente é informado de que nenhum motorista foi encontrado. Este cliente suporta tanto o protocolo IPv4 quanto o IPv6.

Funcionamento do Servidor

A função `create_and_connect_socket()`: `void create_and_connect_socket(sa_family_t family, char *ipAddress, int port)` é responsável por criar um socket TCP e conectar-se ao servidor. Ela recebe como argumentos a família de endereços (*AF_INET* para IPv4 ou *AF_INET6* para IPv6), o endereço IP do servidor e o número da porta.

Primeiramente, a função cria um socket TCP utilizando a função `socket()` [figura 12].

```
// Creates a socket:
// - "family": Specifies the address family (AF_INET for IPv4 or AF_INET6 for IPv6).
// - SOCK_STREAM: Indicates a TCP socket.
// - 0: Default protocol (usually IPPROTO_TCP for TCP).
clientSocket = socket(family, SOCK_STREAM, 0);
```

Figura 12

Em seguida, configura a estrutura de endereço do servidor com base na família de endereços especificada [figura 13] e [figura 14].

```
if (family == AF_INET)
{
    ((struct sockaddr_in *)&serverAddress)->sin_family = family;
    ((struct sockaddr_in *)&serverAddress)->sin_port = htons(port);
    inet_pton(family, ipAddress, &((struct sockaddr_in *)&serverAddress)->sin_addr));
    addr_size = sizeof(struct sockaddr_in);
}
```

Figura 13

```
else if (family == AF_INET6)
{
    ((struct sockaddr_in6 *)&serverAddress)->sin6_family = family;
    ((struct sockaddr_in6 *)&serverAddress)->sin6_port = htons(port);
    inet_pton(family, ipAddress, &(((struct sockaddr_in6 *)&serverAddress)->sin6_addr));
    addr_size = sizeof(struct sockaddr_in6);
}
```

Figura 14

Posteriormente, a função tenta conectar o cliente ao servidor usando a função `connect()` [figura 15].

```
// Attempts to connect the client socket to the server address:
// - clientSocket: The socket descriptor created earlier.
// - (struct sockaddr *)&serverAddress: The server address structure.
// - addr_size: The size of the address structure.
if (connect(clientSocket, (struct sockaddr *)&serverAddress, addr_size) == -1)
{
    perror("Erro ao conectar ao servidor");
    exit(1);
}
```

Figura 15

Após estabelecer a conexão, o cliente envia suas coordenadas para o servidor usando a função `send()` [figura 16].

```
// Sends the "coordCli" structure (containing client coordinates) to the server.
send(clientSocket, &coordCli, sizeof(coordCli), 0);
```

Figura 16

O cliente então aguarda a resposta do servidor, que é processada e exibida [figura 17].

```
recv(clientSocket, serverResponse, MAX_SIZE, 0);
```

Figura 17

Finalmente, a função `main()` `int main(int argc, char **argv)` é a principal função do programa. Ela começa verificando se o número correto de argumentos foi fornecido, especificamente a versão IP, o endereço IP do servidor e a porta. A versão IP é então validada para garantir que seja "ipv4" ou "ipv6". Após essas verificações iniciais, a função `create_and_connect_socket()` [figura 18] é chamada com a família de endereços apropriada, o endereço IP do servidor e o número da porta passados como argumentos.

```c
// Request ride based on IP version
if (strcmp(ipVersion, "ipv4") == 0)
{
    create_and_connect_socket(AF_INET, ipAddress, port);
}
else if (strcmp(ipVersion, "ipv6") == 0)
{
    create_and_connect_socket(AF_INET6, ipAddress, port);
}
else
{
    printf("Versão de IP inválida\n");
    exit(1);
}
```

Figura 18

Detalhes da Conexão

- `socket()`[figura 12]: Esta função é utilizada para criar um socket. Ela recebe três argumentos: a família de endereços (por exemplo, AF_INET para IPv4), o tipo de socket (por exemplo, SOCK_STREAM para TCP) e o protocolo (geralmente 0 para o protocolo padrão). A função retorna um descritor de socket que pode ser usado em chamadas de função subsequentes que operam em sockets.
- `connect()`[figura 15]: Esta função é usada para conectar o socket do cliente ao servidor. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para a estrutura de endereço que contém detalhes do endereço e porta do servidor, e o terceiro argumento é o tamanho da estrutura de endereço.
- `send()`[figura 16]: Esta função é usada para enviar dados através do socket para o servidor conectado. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para os dados a serem enviados, o terceiro argumento é o tamanho dos dados, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes enviados ou -1 em caso de erro.

- `recv()`[figura 17]: Esta função é usada para receber dados do servidor conectado através do socket. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para um buffer onde os dados serão armazenados, o terceiro argumento é o tamanho do buffer, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes recebidos ou -1 em caso de erro.
- `inet_pton()`[figura 13 e 14]: Esta função converte endereços IP de notação textual para formato binário. O primeiro argumento é a família de endereços (por exemplo, AF_INET para IPv4), o segundo argumento é uma string contendo o endereço IP em notação textual, e o terceiro argumento é um ponteiro para uma estrutura que será preenchida com o endereço IP em formato binário. A função retorna 1 se a conversão for bem-sucedida ou -1 em caso de erro.

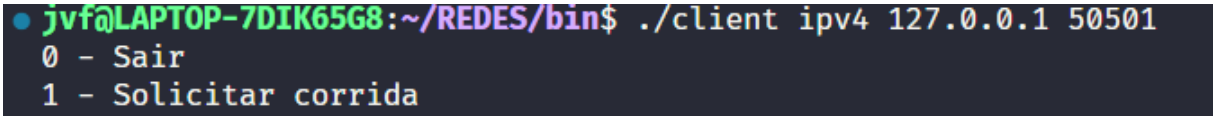
TESTES

IPv4

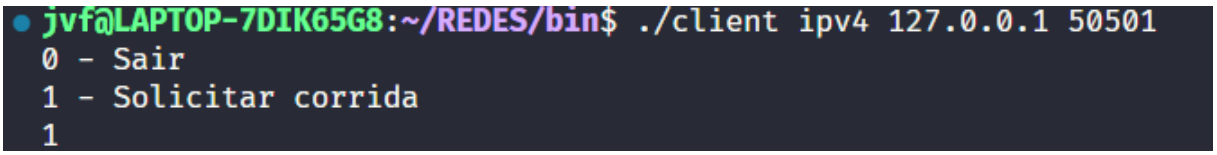
Para os testes do IPv4 foram utilizadas as coordenadas {-19.9391, -43.9398} que correspondem ao endereço do Departamento de Ciência da Computação – ICEx – UFMG que é Rua Reitor Pires Albuquerque, ICEx – Pampulha, Belo Horizonte – MG, 31270–901

Client

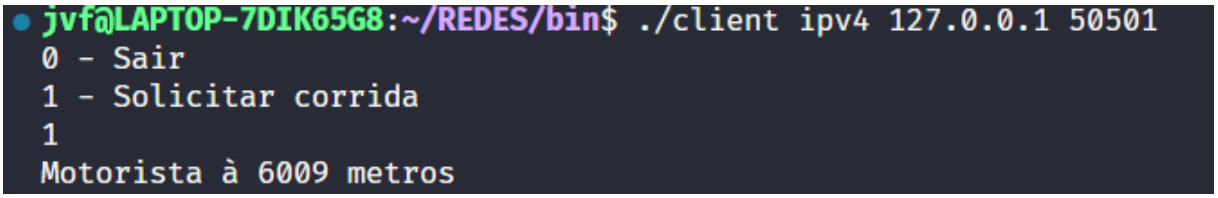
Corrida Aceita



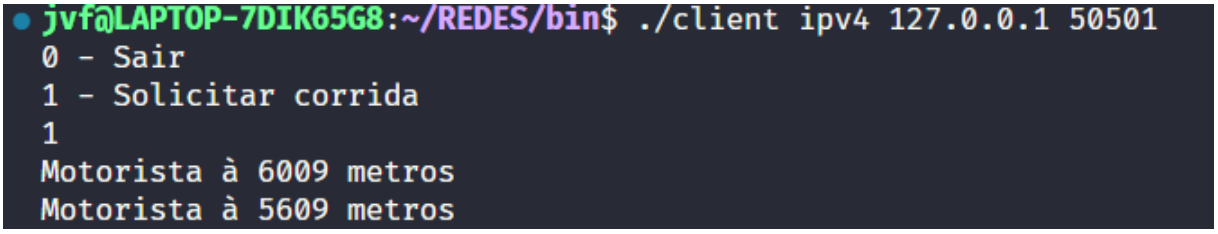
Interface de escolha de ação



Usuário escolha solicitar corrida



Servidor aceita corrida e 1ª resposta de distância do servidor



2ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
```

3ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
```

4ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
```

5ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
```

6ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
```

7ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
```

8ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
```

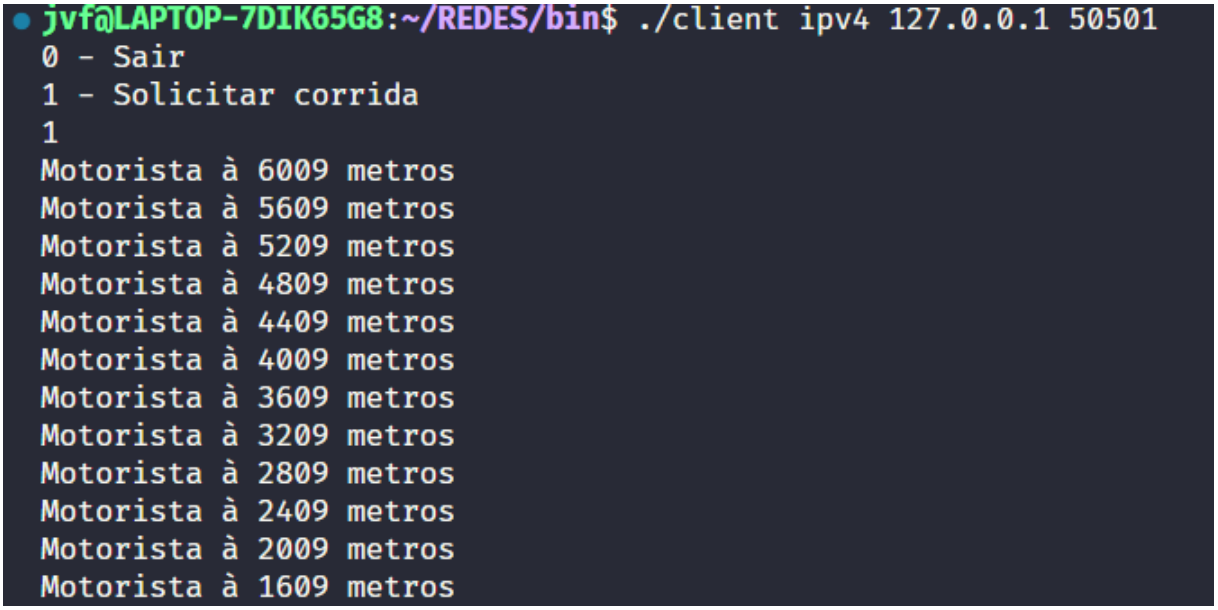
9ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
```

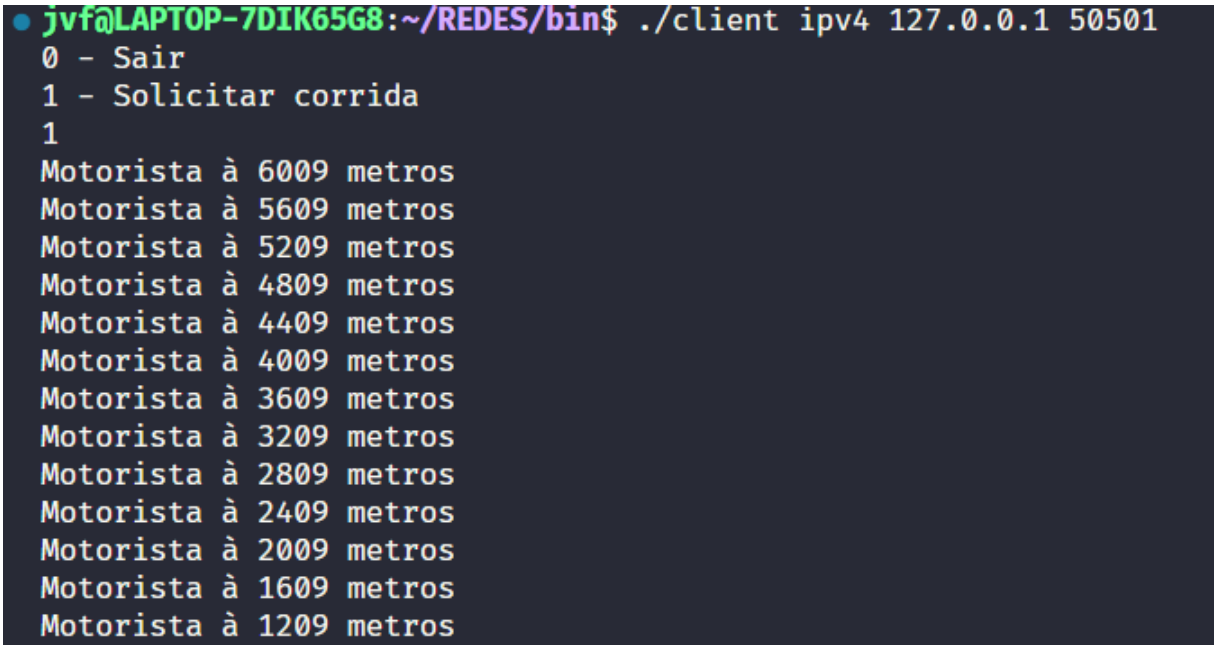
10ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
```

11ª resposta de distância do servidor



12ª resposta de distância do servidor



13ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
Motorista à 1609 metros
Motorista à 1209 metros
Motorista à 809 metros
```

14ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
Motorista à 1609 metros
Motorista à 1209 metros
Motorista à 809 metros
Motorista à 409 metros
```

15ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
Motorista à 1609 metros
Motorista à 1209 metros
Motorista à 809 metros
Motorista à 409 metros
Motorista à 9 metros
```

16ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
Motorista à 1609 metros
Motorista à 1209 metros
Motorista à 809 metros
Motorista à 409 metros
Motorista à 9 metros
O motorista chegou!
```

Resposta de chegada do motorista

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 6009 metros
Motorista à 5609 metros
Motorista à 5209 metros
Motorista à 4809 metros
Motorista à 4409 metros
Motorista à 4009 metros
Motorista à 3609 metros
Motorista à 3209 metros
Motorista à 2809 metros
Motorista à 2409 metros
Motorista à 2009 metros
Motorista à 1609 metros
Motorista à 1209 metros
Motorista à 809 metros
Motorista à 409 metros
Motorista à 9 metros
O motorista chegou!
jvf@LAPTOP-7DIK65G8:~/REDES/bin$
```

Programa se encerra

Corrida Recusada

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
```

Usuário escolha em solicitar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Nao foi encontrado um motorista
```

Servidor recusa corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
1
Nao foi encontrado um motorista
0 - Sair
1 - Solicitar corrida
```

Interface de escolha de ação

Server

Corrida Aceita

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
```

Interface de espera de requisição do cliente

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
1
```

Servidor escolhe aceitar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
1
O motorista chegou!
```

Após enviar atualizações de distância envia que o motorista chegou

Corrida Recusada

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
```

Interface de espera de requisição do cliente

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
0
```

Servidor escolhe recusar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv4 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
0
Aguardando solicitação.
```

Interface de espera de requisição do cliente

IPv6

Para os testes do IPv6 foram utilizadas as coordenadas {-19.939015762357613, -43.939794194394366}; que correspondem ao endereço da dti digital que é Rua Levindo Lopes, 357 – Savassi, Belo Horizonte – MG, 30140–170

Client

Corrida Aceita

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
```

Usuário escolha em solicitar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
```

Servidor aceita corrida e 1ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
```

2ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
Motorista à 1151 metros
```

3ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
Motorista à 1151 metros
Motorista à 751 metros
```

4ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
Motorista à 1151 metros
Motorista à 751 metros
Motorista à 351 metros
```

5ª resposta de distância do servidor

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
Motorista à 1151 metros
Motorista à 751 metros
Motorista à 351 metros
O motorista chegou!
```

Resposta de chegada do motorista

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista à 1951 metros
Motorista à 1551 metros
Motorista à 1151 metros
Motorista à 751 metros
Motorista à 351 metros
O motorista chegou!
jvf@LAPTOP-7DIK65G8:~/REDES/bin$
```

Programa se encerra

Corrida Recusada

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$  ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$  ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
```

Usuário escolha em solicitar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$  ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Nao foi encontrado um motorista
```

Servidor recusa corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$  ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Nao foi encontrado um motorista
0 - Sair
1 - Solicitar corrida
```

Interface de escolha de ação

Server

Corrida Aceita

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
```

Interface de espera de requisição do cliente

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
```

Interface de escolha de ação

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
1
```

Servidor escolhe aceitar corrida

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
1
O motorista chegou!
```

Após enviar atualizações de distância envia que o motorista chegou

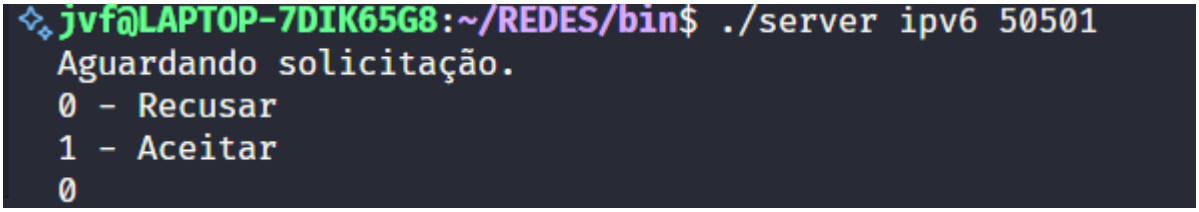
Corrida Recusada

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
```

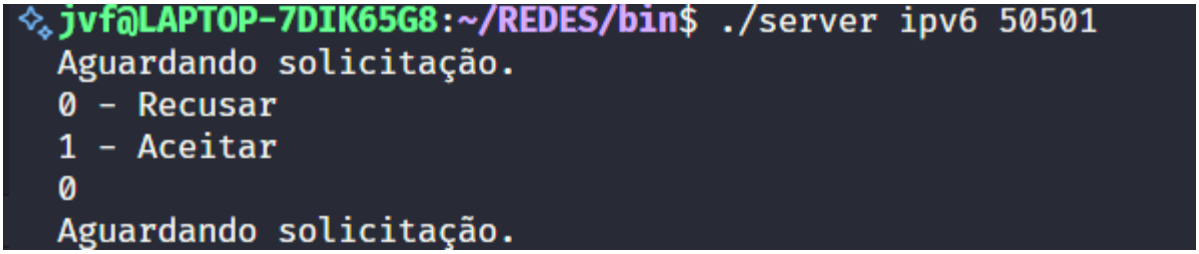
Interface de espera de requisição do cliente

```
jvf@LAPTOP-7DIK65G8:~/REDES/bin$ ./server ipv6 50501
Aguardando solicitação.
0 - Recusar
1 - Aceitar
```

Interface de escolha de ação



Servidor escolhe recusar corrida



Interface de espera de requisição do cliente