

Trabalho Pratico 1 - Redes de Computadores

Adalberto Barbosa Vieira¹

¹Bacharelado em Sistemas de Informação
DCC - Universidade Federal de Minas Gerais 2018079756

adalberto.vieira@dcc.ufmg.br

Abstract. *This report seeks to showcase the execution of a suggested resolution for the Practical Assignment within the Computer Networks course and to engage in a discourse concerning the challenges faced.*

Resumo. *Esse relatório tem como objetivo apresentar a implementação de uma proposta de solução para o Trabalho Prático do curso de Redes de Computadores e discutir os desafios encontrados.*

1. Introdução

O desafio proposto no contexto do Trabalho Prático abarca a concepção e concretização de um sistema de solicitação de corridas, cuja operação ocorrerá entre um cliente e um servidor, viabilizada por meio de sockets.

2. Implementação

Para solucionar o problema, foi necessária a implementação de uma comunicação via socket entre o cliente e o servidor. O servidor foi responsável por gerenciar a lógica do cálculo da distância entre o motorista e o passageiro, enviando atualizações regulares ao passageiro (cliente) sobre a distância restante. Enquanto isso, o cliente exibia mensagens periódicas ao passageiro, baseadas nas respostas recebidas do servidor. Para calcular a distância entre as coordenadas, utilizou-se o método de Haversine.

Além da utilização de tipos "doubles", foi elaborada a seguinte estrutura de dados para a efetivação da comunicação entre o servidor e o cliente:

```
1 typedef struct {  
2     double latitude;  
3     double longitude;  
4 } Coordinate;
```

Durante o processo de implementação, duas variáveis, denominadas "message" e "response," são utilizadas para a manipulação das informações.

O programa esta dividido em tres categorias distintas. commun, server, client.

O arquivo commun possui os seguintes metodos.

```
1  
2 void logexit(const char *error_msg);  
3  
4 int addrparse( const char *addrstring, const char *portstr,  
5     struct sockaddr_storage *storage);  
6
```

```

7 void addrtostr(const struct sockaddr *addr, char *str, size_t strsize);
8
9 int server_sockaddr_init(const char *proto, const char *portstr,
10                          struct sockaddr_storage *storage);

```

- `logexit` Esta função é usada para imprimir uma mensagem de erro e encerrar o programa. Ela utiliza `perror()` para imprimir a mensagem de erro recebida como argumento.
- `addrparse` Esta função analisa e converte um endereço IP e uma porta para uma estrutura de endereço de soquete (`struct sockaddr_storage`). Ela recebe o endereço IP e a porta como strings, converte a porta para um número inteiro de 16 bits e converte o endereço IP para o formato de rede (big-endian). Em seguida, inicializa uma estrutura de endereço IPv4 ou IPv6, dependendo do tipo de endereço IP fornecido.
- `addrtostr` Esta função converte uma estrutura de endereço de soquete em uma string legível. Ela verifica se o endereço é IPv4 ou IPv6, converte o endereço IP e a porta para uma string legível e os formata em uma string.
- `server_sockaddr_init` Esta função inicializa uma estrutura de endereço de soquete para o servidor. Ela recebe o protocolo (IPv4 ou IPv6) e a porta como argumentos, converte a porta para um número inteiro de 16 bits e inicializa a estrutura de acordo com o tipo de endereço (IPv4 ou IPv6). Se o protocolo for "ipv4", ela inicializa uma estrutura `sockaddr_in` para IPv4; se for "ipv6", inicializa uma estrutura `sockaddr_in6` para IPv6.

Além das funções citadas a cima, também foram utilizadas as seguintes funções de sockets no server e client.

- `setsockopt` reutilizar endereços de socket locais mesmo que eles já estejam em uso.
- `bind` Esta função associa o socket a um endereço. Ela recebe o descritor de arquivo do socket (`s`), a estrutura de endereço (`address`) e o tamanho da estrutura de endereço.
- `accept` : Esta função aceita uma conexão de entrada em um socket de escuta. Ela recebe o descritor de arquivo do socket de escuta (`s`), uma estrutura para armazenar as informações do cliente (`client_sockaddr`) e um ponteiro para o tamanho da estrutura.
- `listen` Esta função coloca o socket em modo de escuta, permitindo que ele aceite conexões de entrada. O segundo argumento especifica o tamanho da fila de conexão pendente.
- `recv` Esta função recebe dados de um socket. Ela recebe o descritor de arquivo do socket (`s`), um ponteiro para armazenar os dados recebidos, o tamanho dos dados esperados e opções de recebimento.
- `send` Esta função envia dados por um socket.
- `close` Esta função fecha o socket.

3. Desafios

O primeiro e mais urgente obstáculo manifestou-se na necessidade de adquirir proficiência na implementação de sockets em C. Embora tenha representado uma oportunidade única

para concretizar os conhecimentos teóricos obtidos em sala de aula, a assimilação das nuances da linguagem C revelou-se uma tarefa custosa e demorada. O domínio dos conceitos e a aplicação prática exigiram um tempo considerável de pesquisa e experimentação.

Além disso, a garantia do fluxo apropriado entre as mensagens e respostas constituiu-se em mais um desafio notável. Foi imperativo revisar, em diversas ocasiões, a sequência de execução do código a fim de assegurar que a troca de informações transcorresse sem problemas.

A preocupação com a endianness e a gestão do protocolo IPv6 também geraram complexidades no decorrer do desenvolvimento. Uma falha na implementação exigiu esforços de retrabalho para corrigi-la, o que, por sua vez, acrescentou uma camada adicional de complexidade.

Esses desafios, embora significativos, proporcionaram valiosas oportunidades de aprendizado e crescimento, permitindo o aprimoramento das habilidades práticas e conceituais no domínio das redes de computadores e da programação em C.

4. Prints ipv4

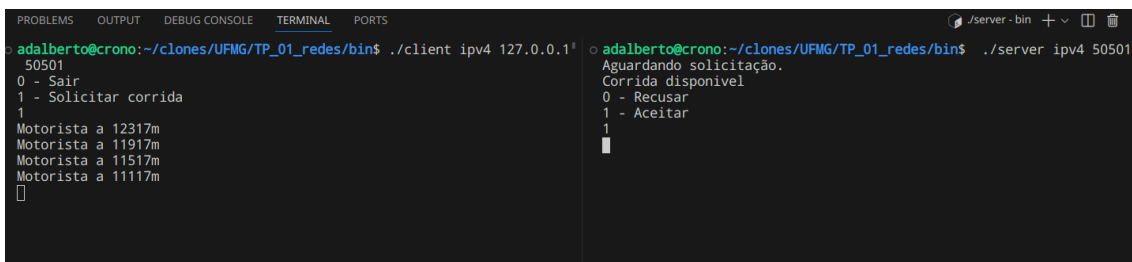
Inicia o programa



```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv4 127.0.0.1
50501
0 - Sair
1 - Solicitar corrida
[]

adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv4 50501
Aguardando solicitação.
[]
```

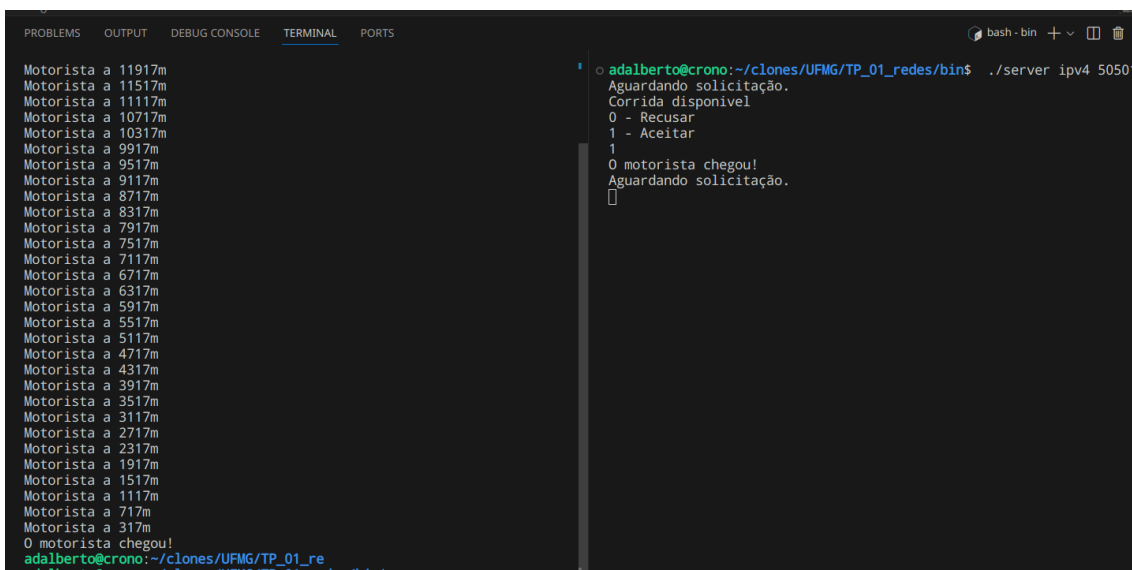
Envia requisito pelo client e aceite do servidor



```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv4 127.0.0.1
50501
0 - Sair
1 - Solicitar corrida
1
Motorista a 12317m
Motorista a 11917m
Motorista a 11517m
Motorista a 11117m
[]

adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv4 50501
Aguardando solicitação.
Corrida disponível
0 - Recusar
1 - Aceitar
1
[]
```

Programa executado



```
Motorista a 11917m
Motorista a 11517m
Motorista a 11117m
Motorista a 10717m
Motorista a 10317m
Motorista a 9917m
Motorista a 9517m
Motorista a 9117m
Motorista a 8717m
Motorista a 8317m
Motorista a 7917m
Motorista a 7517m
Motorista a 7117m
Motorista a 6717m
Motorista a 6317m
Motorista a 5917m
Motorista a 5517m
Motorista a 5117m
Motorista a 4717m
Motorista a 4317m
Motorista a 3917m
Motorista a 3517m
Motorista a 3117m
Motorista a 2717m
Motorista a 2317m
Motorista a 1917m
Motorista a 1517m
Motorista a 1117m
Motorista a 717m
Motorista a 317m
0 motorista chegou!
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$

adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv4 50501
Aguardando solicitação.
Corrida disponível
0 - Recusar
1 - Aceitar
1
0 motorista chegou!
Aguardando solicitação.
[]
```

5. Prints ipv6

Inicia o programa

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida

```

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv6 50501
Aguardando solicitação.

```

Envia requisito pelo client e aceite do servidor

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista a 12317m
Motorista a 11917m

```

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv6 50501
Aguardando solicitação.
Corrida disponível
0 - Recusar
1 - Aceitar
1

```

Programa executado

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Motorista a 12317m
Motorista a 11917m
Motorista a 11517m
Motorista a 11117m
Motorista a 10717m
Motorista a 10317m
Motorista a 9917m
Motorista a 9517m
Motorista a 9117m
Motorista a 8717m
Motorista a 8317m
Motorista a 7917m
Motorista a 7517m
Motorista a 7117m
Motorista a 6717m
Motorista a 6317m
Motorista a 5917m
Motorista a 5517m
Motorista a 5117m
Motorista a 4717m
Motorista a 4317m
Motorista a 3917m
Motorista a 3517m
Motorista a 3117m
Motorista a 2717m
Motorista a 2317m
Motorista a 1917m
Motorista a 1517m
Motorista a 1117m
Motorista a 717m
Motorista a 317m
0 motorista chegou!
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$
```

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv6 50501
Aguardando solicitação.
Corrida disponível
0 - Recusar
1 - Aceitar
1
0 motorista chegou!
Aguardando solicitação.

```

Envia requisito pelo client e não aceite do servidor

```
adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./client ipv6 ::1 50501
0 - Sair
1 - Solicitar corrida
1
Não foi encontrado um motorista.
0 - Sair
1 - Solicitar corrida
█

adalberto@crono:~/clones/UFGM/TP_01_redes/bin$ ./server ipv6 50501
Aguardando solicitação.
Corrida disponível
0 - Recusar
1 - Aceitar
0
Aguardando solicitação.
█
```