Trabalho Prático Individual 01

Redes de Computadores

ALUNO: GUILHERME DRUMOND ROSA
MATRÍCULA: 2020041531

Introdução

Este projeto foi desenvolvido com o intuito de simular o funcionamento básico do aplicativo Uber, por meio da comunicação entre clientes e servidores com conexão TCP. Ao decorrer do projeto, foram utilizadas diversas funções da biblioteca de Sockets para a realização da conexão entre as entidades e a troca efetiva de mensagens. A linguagem utilizada foi C e o ambiente para o qual o projeto foi desenvolvido é Linux.

Explicando o Servidor

O Servidor desempenha a função de “motorista” e deve ficar sempre aguardando a solicitação de algum cliente. O código se inicia com a definição de uma função utilizada para calcular a distância em metros entre dois pontos recebidos como coordenadas (um para o cliente e outro para o servidor) e a definição da struct que armazena a latitude e longitude do Servidor. Fazemos então a declaração de diversas variáveis a serem utilizadas posteriormente e lemos a entrada recebida por linha de comando (tipo ipv4 ou ipv6 e a porta a ser atribuída ao servidor).

Dependendo do tipo de IP, a forma como fazemos a criação do socket apresenta pequenas variações mas a estrutura se mantém. Por meio da função socket, damos início ao socket referente ao servidor e realizamos sua abertura passiva, ou seja, ele ficará esperando a conexão de um cliente. Para isso, atribuímos os valores desejados a todos os atributos da struct de endereçamento, associamos o IP a porta do socket ao chamarmos a função bind e, por fim, chamamos a função listen para que o servidor trabalhe passivamente.

Entramos no primeiro loop mais externo e realizamos um accept, ou seja, servidor aceita a conexão com o cliente. A partir daí, servidor e cliente passam a trocar diversas mensagens em relação às coordenadas do cliente, se o motorista aceita ou não a corrida e, ainda, atualizando o progresso de uma corrida em andamento. Para que isso fosse possível, utilizamos a função send para envio de mensagens e recv para o recebimento. Sempre que a conexão com um cliente se encerrar, o servidor volta a escutar o meio.

Explicando o Cliente

O Cliente desempenha a função de “usuário” e é ele quem deve alcançar o processo servidor no momento da conexão. O código se inicia com a definição da struct que armazena a latitude e longitude do Cliente, seguida da declaração de diversas variáveis a serem utilizadas posteriormente. Lemos a entrada recebida por linha de comando (tipo ipv4 ou ipv6, endereço IP e porta).

Dependendo do tipo de IP, a forma como fazemos a criação do socket apresenta pequenas variações mas a estrutura se mantém. Por meio da função socket, damos início ao socket referente ao cliente e realizamos sua abertura ativa, ou seja, ele buscará a conexão com o servidor e dará início a ele. Para isso, atribuímos os valores desejados a todos os atributos da struct de endereçamento e, por fim, chamamos a função connect para que o cliente inicie a conexão com o servidor.

Entramos no loop referente à conexão e, a partir daí, servidor e cliente passam a trocar diversas mensagens em relação às coordenadas do cliente, se o motorista aceita ou não a corrida e, ainda, atualizando o progresso de uma corrida em andamento. Para que isso fosse possível, utilizamos a função send para envio de mensagens e recv para o recebimento. Sempre que a conexão com o servidor se encerrar, o socket do cliente é fechado.

Exemplo 1 (corrida completa)

1. Estado inicial da aplicação, servidor aguardando solicitação e cliente pode solicitar uma corrida ou sair.

```
gdrumondrosa@PCGui:/mnt/c/Developer/TP1_TCP_Redes$ ./bin/server ipv4 50501
Aguardando solicitação.
```

```
gdrumondrosa@PCGui:/mnt/c/Developer/TP1_TCP_Redes$ ./bin/client ipv4 127.0.0.1 50501
0 - Sair
1 - Solicitar corrida
```

2. Cliente solicita uma corrida e, agora, o servidor deve aceitar ou recusar.

```
Corrida disponível:
0 - Recusar
1 - Aceitar
```

```
0 - Sair
1 - Solicitar corrida
1
```

Servidor Cliente

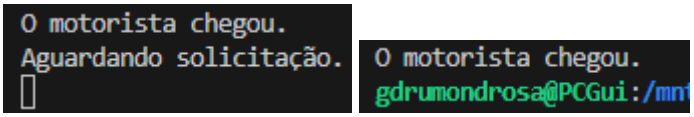
3. Servidor aceita a corrida e o cliente é informado da distância a cada 400m

```
0 - Recusar
1 - Aceitar
1
```

```
Motorista a 2499m
Motorista a 2099m
Motorista a 1699m
Motorista a 1299m
Motorista a 899m
Motorista a 499m
Motorista a 99m
```

Servidor Cliente

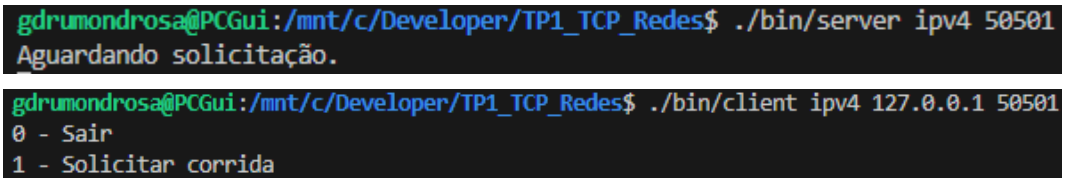
4. Ao chegar, a mensagem “O motorista chegou” aparece para as duas entidades. O servidor volta a escutar o meio e o usuário encerra.



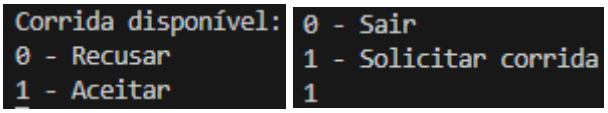
Servidor Cliente

Exemplo 2

1. Estado inicial da aplicação, servidor aguardando solicitação e cliente pode solicitar uma corrida ou sair.

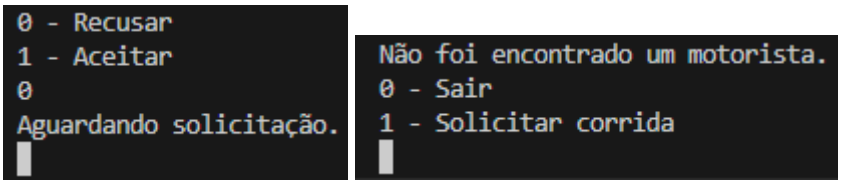


2. Cliente solicita uma corrida e, agora, o servidor deve aceitar ou recusar.



Servidor Cliente

3. Servidor recusa e imediatamente volta a escutar o meio, enquanto o cliente retorna ao menu principal após a mensagem “Não foi encontrado um motorista” ser exibida.



Servidor Cliente

Conclusão

Neste projeto, buscamos simular o funcionamento básico do aplicativo Uber, utilizando comunicação TCP entre clientes e servidores. O servidor, representando o papel de "motorista", permanece em espera constante por solicitações de clientes. Por sua vez, o cliente, assumindo o papel de "usuário", busca ativamente conectar-se ao servidor para solicitar corridas e ser atualizado do andamento do percurso. A integração desses componentes resultou em uma simulação prática do funcionamento de conexões desse tipo.