DCC

DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático Individual 01

Disciplina: Redes de Computadores
Objetivo: Este trabalho prático proporcionará aos alunos uma experiência prática no desenvolvimento de sistemas distribuídos simples, além de oferecer insights sobre os desafios e considerações envolvidos na criação de aplicativos cliente-servidor.
Linguagem: C
Tipo de conexão: TCP
Valor: 25 pontos
Data de entrega: xx/xx/2024

Descrição das atividades:

Neste trabalho prático, os alunos devem desenvolver uma aplicação console simulando o funcionamento básico do aplicativo Uber, com um cliente fazendo o papel do usuário e um servidor fazendo o papel do motorista. O objetivo é proporcionar aos alunos uma compreensão prática de como funciona a comunicação entre clientes e servidores com conexão TCP.

Requisitos:

- Servidor (Motorista):
  - Deverá conter a seguinte struct com os valores especificados de coordenadas:
    - ```
      typedef struct {
          double latitude;
          double longitude;
      } Coordinate;
      // Criando uma coordenada
      Coordinate coordServ = {-19.9227,-43.9451};
      ```
  - O papel do servidor será “escutar” à espera de uma conexão. Enquanto espera, deve ser exibida uma mensagem “Aguardando solicitação.” na tela. (Figura 1)
      ```

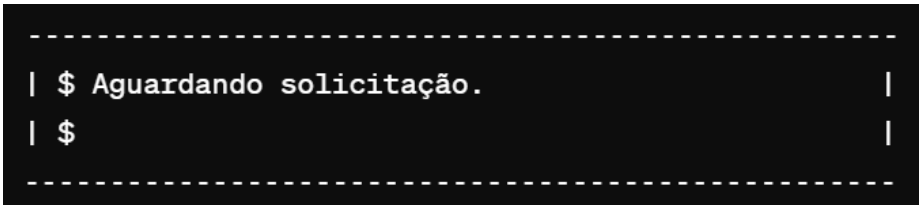

Figura 1 - Servidor em espera.

- Ao receber uma conexão do cliente, o terminal do servidor dará a opção de aceitar ou recusar uma corrida: (Figura 2)

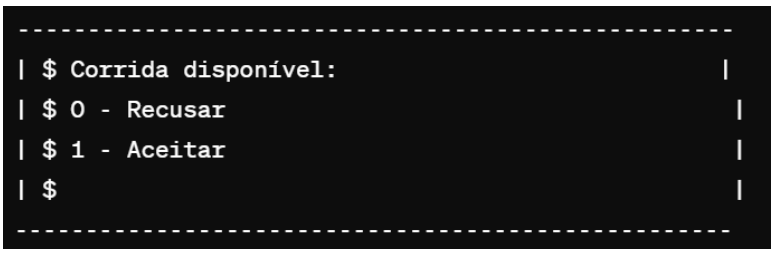

Figura 2 - Servidor ao estabelecer uma conexão.

  - Caso aceite:
    - O servidor receberá as coordenadas do cliente. Deve ser calculado a distância em metros entre suas coordenadas e as coordenadas do cliente. (Ver dicas).
    - O servidor deverá enviar mensagens consecutivas ao cliente. Essas mensagens irão simular a distância do motorista ao cliente.
    - Deve-se iniciar com o valor calculado entre o Servidor e o Cliente. A cada 2000ms, essa distância deve ser decrescida de 400m e uma nova mensagem enviada ao cliente atualizando a distância. (Figura 6) Na distância 0 (zero) a mensagem deverá ser “O motorista chegou!”. Ao mesmo tempo, a mesma mensagem deverá ser exibida no terminal do Servidor. (Figura 3)

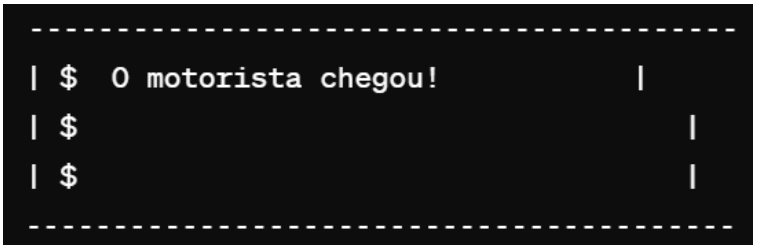

Figura 3 - Servidor após enviar todas as mensagens ao cliente.

- Caso recuse, o servidor deverá enviar uma mensagem ao cliente com os dizeres “Não foi encontrado um motorista.” (Figura 5). Após isso, a conexão deverá ser desfeita.
- Em ambos os casos, ao final, o servidor voltará a escutar esperando uma nova conexão.

- Cliente (Usuário):
  - Deverá conter a seguinte struct, utilizando sua latitude e longitude com 4 casas decimais de precisão:

```
typedef struct {
    double latitude;
    double longitude;
} Coordinate;
// Criando uma coordenada
Coordinate coordCli = {suaLatitude, suaLongitude};
```

  - A latitude e longitude podem ser consultadas no Google Maps.
  - O papel do cliente será buscar uma conexão com o servidor. Seu menu inicial terá duas opções, Sair ou Buscar Motorista (Figura 4):

```
--------------------------------
| $ 0 - Sair                   |
|     1 - Solicitar corrida    |
| $                            |
--------------------------------
```

Figura 4 - Menu inicial do cliente.

- Ao clicar em sair:
  - o programa será encerrado.
- Ao clicar em “Buscar Motorista”
  - O cliente iniciará a conexão com o servidor enviando suas coordenadas.
  - Após conexão bem sucedida, ele aguardará a resposta do servidor e exibirá na tela.
  - Caso a corrida tenha sido recusada, o cliente deverá voltar ao menu inicial. (Figura 5)

```
------------------------------------------------------------
| $ Não foi encontrado um motorista.                       |
| $ 0 - Sair                                                 |
| $     1 - Solicitar corrida                                |
------------------------------------------------------------
```

Figura 5 - Exibição para corridas recusadas.

- Caso contrário, deverá exibir todas as mensagens e após a última mensagem (“O motorista chegou!”) o programa deverá ser encerrado. (Figura 6)

Figura 6 - Exibição das respostas do servidor.

- Documentação:
  - A documentação deve ser composta de:
    - Pelo menos 3 (três) páginas.
    - Deve estar em formato PDF.
    - Deve conter o nome completo e matrícula do aluno.
    - Deve conter uma seção explicando o código do Servidor e do Cliente. Os trechos que envolvem conexão devem ser explicados detalhadamente. Isso inclui deixar explícito o funcionamento de métodos utilizados de qualquer biblioteca de rede.
    - Deve conter uma seção com prints dos testes simulando o caso em que o motorista aceita ou recusa a corrida. Todas as etapas devem estar explícitas nos prints. Cada print deve conter uma legenda com sua descrição.

Detalhes de Implementação:

- Servidor e Cliente devem estar em arquivos separados.
- O trabalho deve ser entregue utilizando a linguagem C (Não é permitido o uso de C++).
- O mesmo deverá ser desenvolvido para máquinas Linux (Não utilizem bibliotecas que não sejam compatíveis).
- Todas as linhas do código que remetem à conexão devem ser comentadas explicando sua função.
- Os alunos devem implementar um mecanismo de comunicação adequado entre os clientes e o servidor, como sockets TCP/IP.

- Deve ser utilizado IPv4 e IPv6. Desta forma, o servidor deve ser configurado com o parâmetro que define o tipo de IP. O segundo parâmetro deverá ser a porta utilizada. Exemplo:
    - $ ./server ipv4 50501
    - $ ./server ipv6 50501

O cliente também deverá receber parâmetros. Primeiro a versão do IP (ipv4 ou ipv6), em seguida o IP do servidor e por fim a porta. Exemplo:

- $ ./client ipv4 127.0.0.1 50501
- $ ./client ipv6 ::1 50501

- **Dicas:**
    - Para o cálculo de distância entre coordenadas, pode-se utilizar a fórmula de Haversine. (https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/).
    - Link para playlist de introdução à programação de programas de rede do professor Ítalo Cunha (https://www.youtube.com/watch?v=tJ3qNtv0HVs&list=PLyrH0CFXIM5Wzmbv-IC-qvoBejsa803Qk&index=1)
    - Utilize a metodologia de divisão e conquista. Divida o trabalho em problemas menores e solucione-os por partes.

Exemplos de execução:

Exemplo1:

Servidor	Cliente	Obs
$ Aguardando solicitação	$ 0 - Sair 1 - Solicitar Corrida	Cliente seleciona solicitar corrida
$ Corrida disponível: 0 - Recusar 1 - Aceitar		Servidor seleciona aceitar
	$ Motorista a 900m	
	$ Motorista a 500m	
	$ Motorista a 100m	
$ O motorista chegou!	$ O motorista chegou!	
$ Aguardando solicitação.	<Programa é encerrado>	

Exemplo 2:

Servidor	Cliente	Obs
$ Aguardando solicitação	$ 0 - Sair 1 - Solicitar Corrida	Cliente seleciona solicitar corrida
$ Corrida disponível: 0 - Recusar 1 - Aceitar		Servidor seleciona recusar.
$ Aguardando solicitação	$ Não foi encontrado um motorista.	
	$ 0 - Sair 1 - Solicitar Corrida	

Entrega:

- Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.
- Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.
- O Makefile deve compilar o cliente em um binário chamado “client” e o servidor em um binário chamado “server” dentro de uma pasta chamada "bin" na raiz do projeto.
- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP1_MATRICULA.zip
- Pontuação:
    - Servidor (8 pontos):
        - Escutar conexão, efetuar conexão e receber coordenadas (4pts)
        - Calcular distância entre coordenadas (1pt)
        - Tratamento correto das opções de Aceitar/Recusar corrida (1pt)
        - Retorno ao cliente (2pts)
    - Cliente (8 pontos):
        - Buscar conexão com o servidor (2pts)
        - Conectar e enviar coordenadas (3pts)

        - Tratamento correto da resposta do servidor (3pts)
    - Documentação (9 pontos):
        - Descrição do código com explicações explícitas (6pts)
        - Prints (com descrições) do programa em funcionamento. (3pts)
- Penalidade por atraso:
    - Para cada dia de atraso na entrega do trabalho ocorrerá uma penalidade de 25% da nota. Conforme tabela abaixo:

Dia	Penalidade
1 dia de atraso	25%
2 dias de atraso	50%
3 dias de atraso	75%
4 ou mais dias de atraso	100% (Nota zero no trabalho)