COMO USAR:

Como rodar o cliente ?

No mesmo diretorio do projeto, rodar: `./cliente <port> <endereço>`

Como rodar o servidor ?

No mesmo diretorio do projeto, rodar: `./servidor <port>`

Como enviar dados ?

Digite exatamente assim no terminal: **`D 1 3044 -22.36589 -54.248964`**

No comando acima `D 1 3044 -22.36589 -54.248964` eu **salvo** um combustivel do tipo **Alcool** com o preço de **3044** nas coordenadas **-22.36589 -54.248964**.

- Primeiro Parametro:
    - D: Comando para salvar dados
- Segundo parametro
    - [0, 1, 2] Valores de 0 a 2
    - 0: Salvar combustivel Diesel
    - 1: Salvar combustivel Alcool
    - 2: Salvar combustivel Gasolina
    - Exemplo: `.. 1 ..` `.. 2 ..` `.. 0 ..`
- Terceito parametro
    - Preço, um inteiro de 4 Caracteres. Exemplo: `4068, 3014, 2047 1984`
- Quarto parametro
    - Coordenada x e y, Exemplo: `-22.36589 -54.248964`

Como pesquisar dados ?

Digite exatamente assim no terminal: **`P 2 2 -22.222700 -54.762646`**

No comando acima `P 2 2 -22.222700 -54.762646` eu **Pesquiso** um combustivel do tipo **Gasolina** e com raio de busca **dois** nas coordenadas **-22.222700 -54.762646**.

- Primeiro Parametro:
    - P: Comando para Pesquisar dados
- Segundo parametro
    - [0, 1, 2] Valores de 0 a 2
    - 0: Pesquisar combustivel Diesel
    - 1: Pesquisar combustivel Alcool
    - 2: Pesquisar combustivel Gasolina
    - Exemplo: `.. 1 ..` `.. 2 ..` `.. 0 ..`
- Terceito parametro
    - Raio, um inteiro que indica o raio de busca
- Quarto parametro
    - Coordenada x e y, Exemplo: `-22.36589 -54.248964`

SERVIDOR:

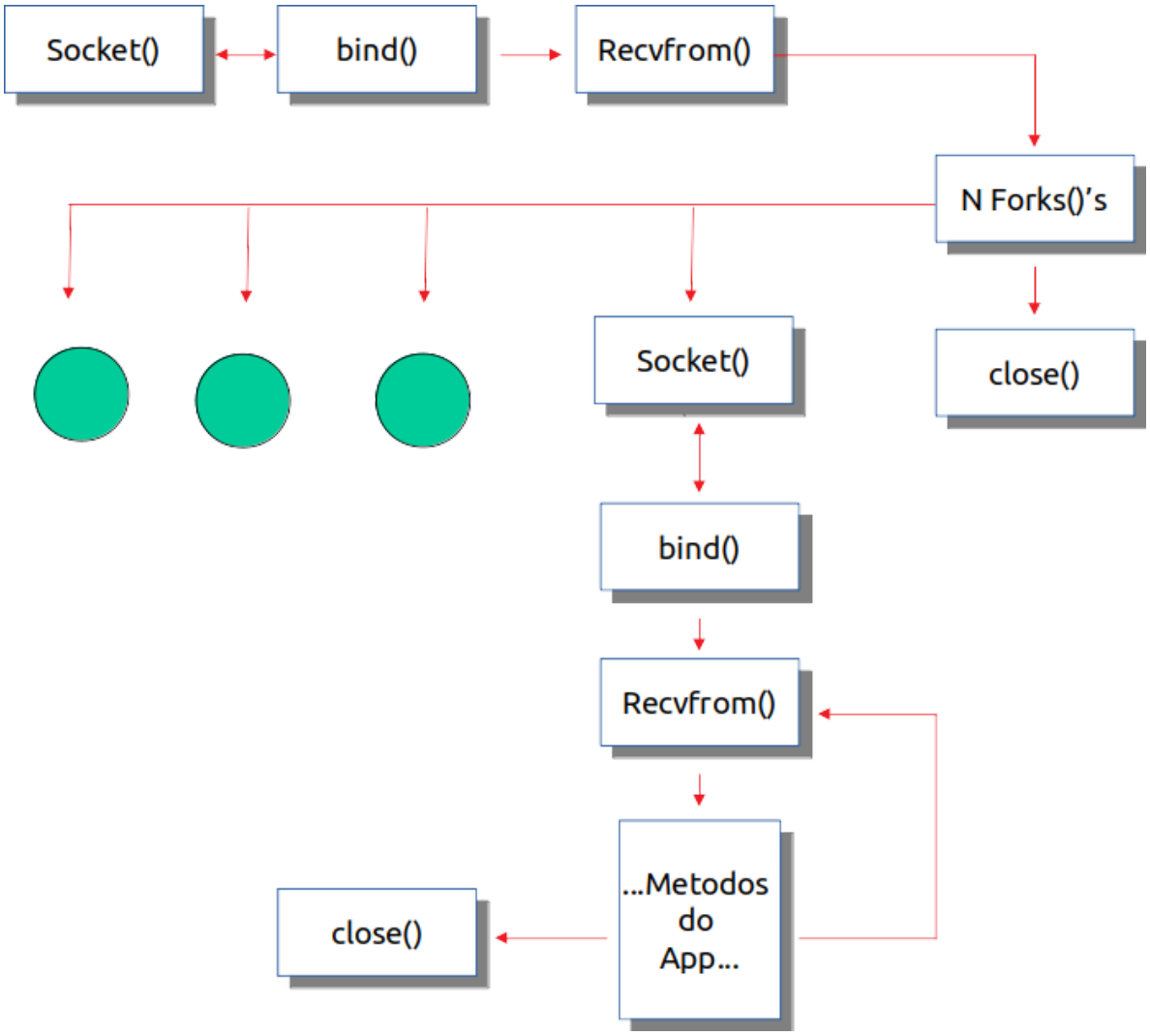
Problema:

O problema consistia em reproduzir de maneira sucinta o funcionamento do protocolo UDP, usando sockets e arquitetura *`cliente-servidor`*. A implementação seguiu uma especificação. O sistema desenvolvido é capaz de enviar o preço de vários postos de combustíveis e requisitar o preço mais barato em uma dada região. O sistema foi composto de dois programas, `cliente` e `servidor`.

Panorama geral e soluções de implementação:

Como o problema a ser resolvido é uma "simulação" do protocolo `UDP`, haviamos necessidade de estabelecer um par um para um 1:1, ou seja, um unico cliente se comunica exclusivamente com um unico socket no servidor. O protocolo `UDP` aceita simultaneas mensagens em um unico socket. então o primeiro desafio foi estabelecer essa cardinalidade.

Soluções



A Solução é simples, usamos um unico socket para para receber conexões de qualquer cliente. Ápos o cliente se "conectar", a função `recvfrom` retorna o tamanho de bytes da mensagem recebida, verificamos com um `if` se numero de bytes recebido é maiores que zero, se sim criamos um `processo` para aquele cliente, e nessa bifurcação estabelecemos o par 1 para 1. O diagrama acima ilustra o que foi dito. Criamos um novo socket para o cliente, fazemos o bind do cliente com novo socket criado. Apartir desse momento, todas as mensagens futuras enviadas do cliente, vão ser recebidas pelo socket a qual foi feito o bind.

```
 while (true) {
        dados = recvfrom (...); // Recebe dados do cliente
        if(dados > 0)  {
            puts("Um cliente mandou mensagem\n");

            // Cria processo novo para o cliente
            if(fork() == 0) {
                inicia_cliente(...); // inicia o processo do novo cliente
                close(socketServidor); // fecha o socket
                exit(0);
            }
        }
    }
```

Nesse trecho de código, há um `while` infinito recebendo conexões, apos `recvfrom` receber mensagens, criamos um `processo` com o uso da função `fork()`, se o processo for o processo filho, inicializamos o cliente.

Vamos agora ver um pouco da função `inicia_cliente()`

```
void inicia_cliente(sock_In *cliente, int port) {

    novo_socket = criaSock();
    bindSock(&servidor, novo_socket,  cliente->sin_addr.s_addr, port);
    /// Mais código...

    while (true) {
        puts("Aguardando mensagens...\n");
        dados = recvfrom(...);

        /// Mais código...

        if (dados > 0)
            trata_solicitacao(...);

        /// Mais código...
    }
}
```

Nessa função, recebedos uma estrutura cliente e um inteiro chamado porta, nessa estrutura teremos informações do cliente que acabou de tentar estabelecer a "conexão", já a porta, é a porta a qual o servidor fara o `bind`, note que o cliente precisara dessa mesma porta para conseguir mandar mensagens para esse socket. Omitimos nas explicações acima o motivo da porta, mas assim que o servidor cria o `fork()` o `S.O` gera uma porta aleatoria e enviamos essa porta para o cliente e o fluxo continua o mesmo. Pegamos essa porta gerada aleatoria que será compartilhada com o cliente e fazemos o bind com o servidor. Em seguida, entranos no `while` e esperamos mensagens do cliente.

Vamos dar uma olhada na função `trata_solicitacao` e concluir essa explicação.

```
void trata_solicitacao(...) {

    int tipo_operacao = trata_mensagem(frame_recebido);

    /// "char *string = frame_recebido->pacote.string"
    check = checksum_verifica(string, strlen(string));


    // Se o valor do checksum_verification
    // for diferente do valor recebido no
    /// campo checksum da struct recebida, quer dizer
    // que o pacote veio corrompido, reenviamos
    // o pacote e retornamos sem continuar o codigo

    if(check != frame_recebido->checksum) {
        sendto(
            socketServidor,
            (const char *)&(frame_enviar),
            sizeof(Frame),
            MSG_CONFIRM,
            (sock_addr*)&cliente,
            sizeof(cliente)
        );
        return;
    }

    // Se tipo_operacao for igual a um, significa que o cliente quer salvar
dados
    tipo_operacao == 1 ? salvar_dados(...) : pesquisa_dados(...);
    send_ack(...);

}
```

A Função trata a string recebida, verifica se os dados estão corretos, verifica o tipo de operação e redoreciona conforme for a solicitação do usuario. a função `salvar_dados`, salva no arquivo os dados enviado pelo cliente, já a função `pesquisa_dados` retorna para o cliente as informações que ele deseja.

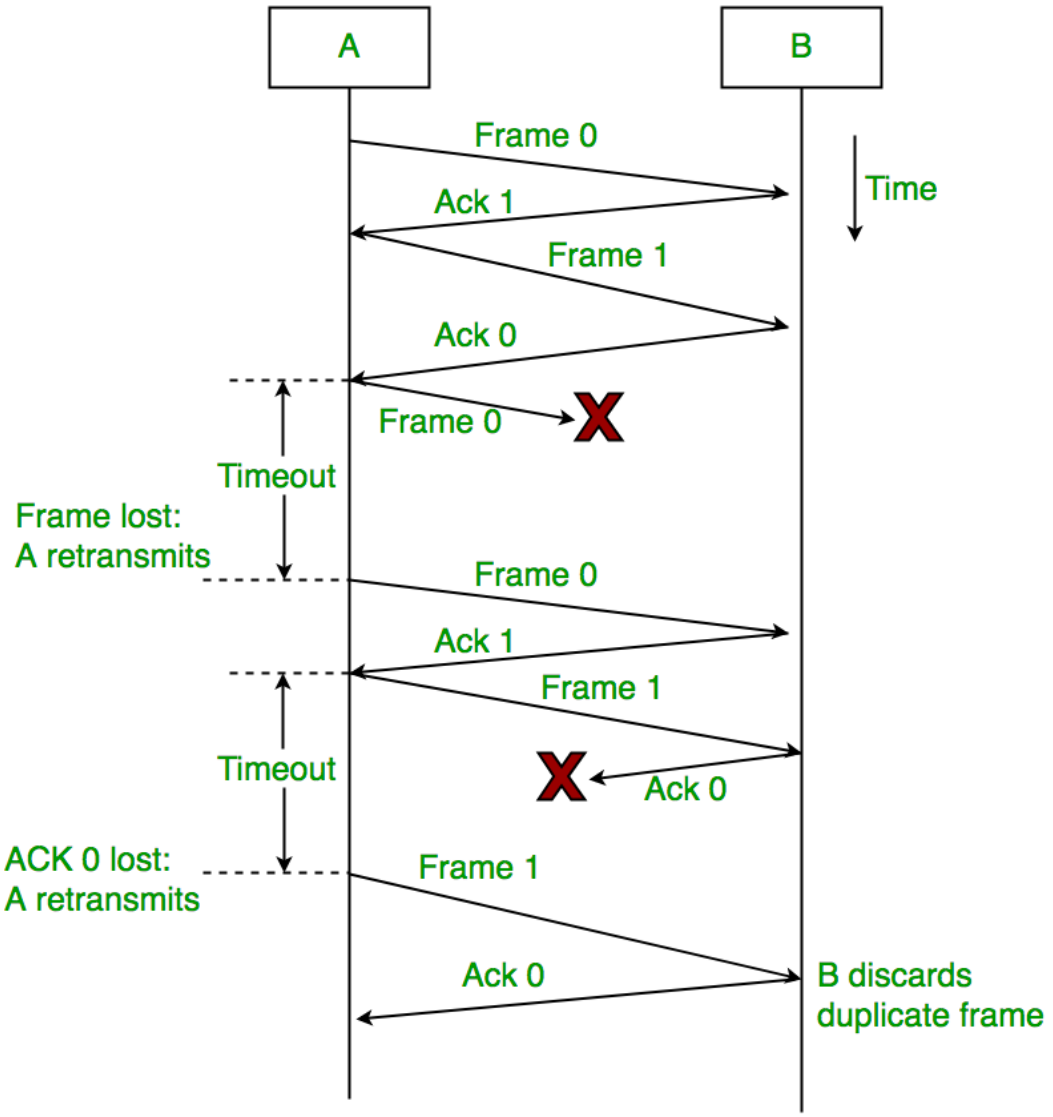
Chekcsum:

```c
uint32_t checksum_verifica(char *data, size_t len) {
    uint32_t a = 1, b = 0;
    size_t index;

    // Processar cada byte dos dados em ordem
    for (index = 0; index < len; ++index) {
        a = (a + data[index]) % MOD_ADLER;
        b = (b + a) % MOD_ADLER;
    }

    return (b << 16) | a;
}
```

> **Checksum** ou **soma de verificação** é um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo. A implementação acima é o algoritmo Adler-32, a explicação do mesmo esta na aba de *Algoritmos usados na implementação do trabalho*

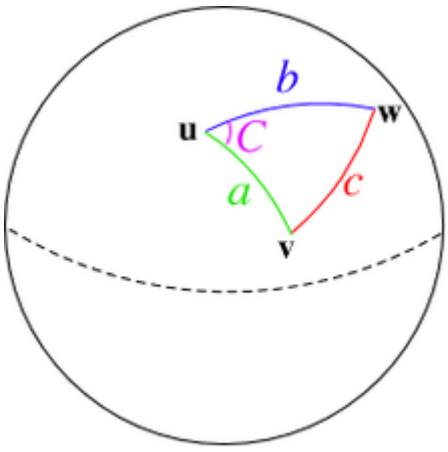
Algotitmos usados na implemetação do trabalho:

1. **Stop and Wait**
    1. Usado na comunicação orientada por conexão.
    2. Oferece erro e controle de fluxo
    3. implementa principalmente o conceito de protocolo de janela deslizante com tamanho de janela 1



2. **Haversine**

    1. A fórmula haversina determina a distância do grande círculo entre dois pontos em uma esfera, dadas as suas longitudes e latitudes.



3. **Adler-32**

    1. Adler-32 é um algoritmo de soma de verificação que foi inventado por Mark Adler em 1995, e é uma modificação do checksum de Fletcher O Adler-32 é mais confiável que o Fletcher-16 e um pouco menos confiável que o Fletcher-32 .

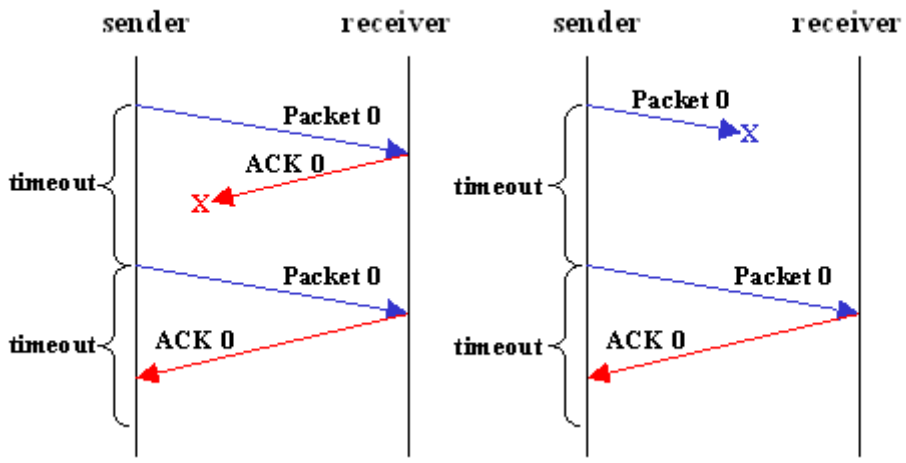
2. Uma soma de verificação Adler-32 é obtida calculando duas somas de verificação A e B de 16 bits e concatenando seus bits em um inteiro de 32 bits. A é a soma de todos os bytes no fluxo mais um e B é a soma dos valores individuais de A de cada etapa.

   No início de uma execução Adler-32, A é inicializado em 1, B em 0. As somas são feitas no módulo 65521 (o maior número primo menor que 2^{16}). Os bytes são armazenados em ordem de rede (big endian), B ocupando os dois bytes mais significativos

Retransmição de mensagens:

A retransmição de mensagen foi implementada seguindo o procotolo stop-and-wait, ou seja, a retransmição da mensagem é feita após um estouro do timeoute, a expressão "estouro do timeout" significa que um tempo limite de espera se esgotou, então é retransmitido a mensagem. essa retranmição tem um limite, no cliente e no servidor, esse limite é 5. Ou seja, apos 5 retransmições de mensagem e nenhuma resposta, o lado que retranmitiu a mensagem encerra a "conexão".

Abaixo um diagrama para melhorar o entendimento:



Aqui temos um `sender` e um `receiver`, no lado esquerdo da mensagem o `packet 0` é enviado, o `receiver` responde com `ack 0`, esse ack não chega no tempo determinado, então uma retranmição é ativada assim retranmityindo o `packet 0`. O Servidor nota que já recebeu o `packet 0` e reenvia o `ack 0`. Quando a mensagem chega e o ack também, o fluxo normal continua.

Conclusoes finais:

O Maior desafio durante a construção e implementação do servidor, foi estabelecer um par 1:1 entre o `socket do servidor` e `socket do cliente` em novo um processo.

CLIENTE:

Problema:

O problema a ser tratado no lado do cliente é simples, precisamos enviar e receber dados para o servidor. Como foi uma implemetação "fechada", o cliente rigosoramente precisou seguir as regras do servidor para envio e recebimento da mensagens. Um ponto importante é o pedido de conexão do cliente ao servidor, o que será explicado a seguir.

Solução:

```
int main(int argc, char **argv) {

    sockfd = criaSocket(...); // Cria socket
    servidor.sin_port = handshaking(..); // Estabelece conexão com servidor

    int proxSeq = 0; // Faz o controle de qual pacote enviar e qual ack receber

    while (true) {
        // Se o ack é o correto, envia o proximo pacote
        if(proxSeq == frame_recebido.ack) {
            /* ************************************************************
            *  Pega mensagem do cliente, seta pacote e envia o pacote para
            *  o cliente.
            *  Seta o proximaSequencia
            ************************************************************/

        } else // Reenvia o pacote anterior


        escutaResposta(...); // Nesse ponto se o TimeOut estorar, reenvia o
pacote,
        recvfrom(...);

        if(tamanho_recebido > 0) {
            if(frame_recebido.ack == proxSeq)  // Trata mensagem recebida do
servidor
                trate(...);
            else  // Se o ACK recebido não é o esperado, Reenvia o pacote
                sendto(...);
        }

    }
}
```

Uma explicação sucienta desse trecho de código: Primeiro passao, cria um novo socket, logo em seguida atribui a esse socket a porta que vai vim da função `handshaking`. Inicia um `while` infinito para aguardar mensagens. a cada mensagem enviada, estarta a função `escutaResposta()` que vai definir um `timeout` e ficar escutando os eventos.

A Função `handshaking` apenas envia uma solicitação para o servidor pedindo uma porta, o servidor retorna para essa função um inteiro com a porta a qual o cliente deve se conectar.

Panorama Geral e soluçãoes de implementação:

No lado do cliente, assim como no lado do servidor, precisamos fazer o controle de qual pacote foi recebido e qual pacote foi enviado, para isso, a mesma solução foi impelementada, o protocolo stop and wait.

As seguintes estruturas definidas no lado do servidor, foram adotadas também pelo cliente:

```
struct pacote {
    char string[2000];
};

struct frame {
    int ack;
    int tipo;
    int porta;
    int proxSeq;
    uint32_t checksum;
    Pacote pacote;
};
```

Essas estruturas definem os dados que irão percorer pela rede, na primeira é a mensagem propriamente dita enviada pelo cliente ou servidor.

Já a segunda estrutura, `frame` .É uma estrutura de controle, ou seja, cada variavel passada é para controle do pacote.

- Ack, é uma variavel de controle. Serve para saber se o pacote recebido é o certo;
- Tipo, é o inteiro que indica qual pacote está percorrendo pela rede;
- Porta, é usada para que o servidor envie a porta de conexão a qual o cliente devera mandar mensagem;
- proxSeq, é uma variavel de controle. Serve para saber qual o proximo pacote devera ser recebido;
- cheksum, soma de verificação da mensagem;
- Pacote pacote, é a string a ser enviada;

Algoritmos usados na implementação do trabalho:

Os algoritmos usados no lado do cliente são os mesmos usados no servidor.

Retransmição de mensagens:

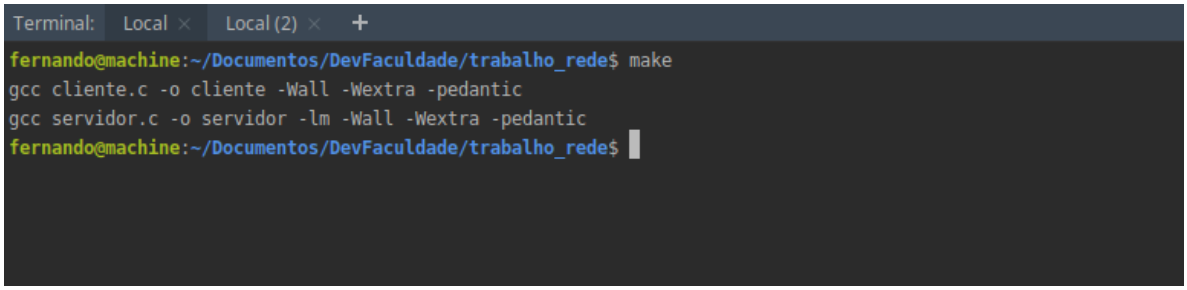
Os algoritmos e metodos de retranmição no usados no lado do cliente são os mesmos usados no servidor.

Conclusoes Finais:

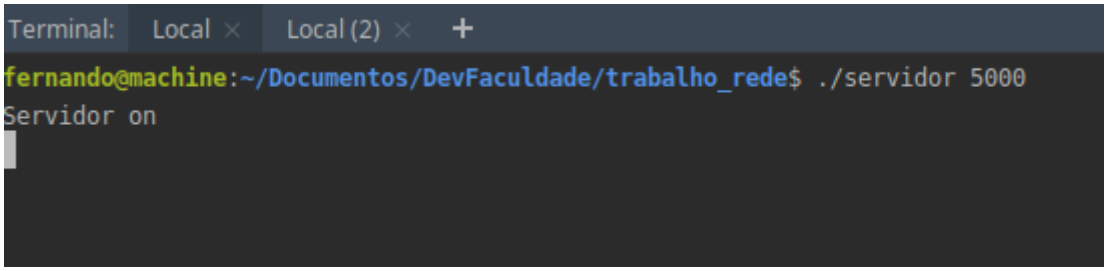
A implementação do cliente foi rápida e facil, ela é muito similar a implementação do servidor, apenas com suas diferenças simples, nada que fuja do padrão do servidor.

TESTES:

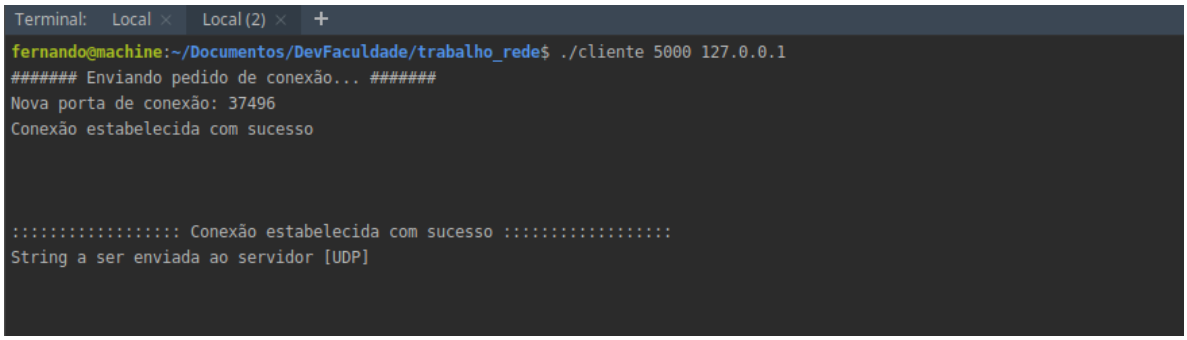
Teste de compição com MAKE:



Teste do Servidor:



Teste do cliente:



```
fernando@machine:~/Documentos/DevFaculdade/trabalho_rede$ ./servidor 5000
Servidor on
:::::::::::::::::: Um cliente acaba de fazer uma solicitação de conexão... Criando novo processo ::::::::::::::::::

S.O esta gerando uma nova porta!!
Porta gerada: >> 37496

######### Enviando nova porta para cliente solicitante #########



Conexão do cliente na porta 37496 estabelecida com sucesso, PID do processo: 31313

:::::::::::::: Identificador do cliente: 31313
:::::::::::::: Aguardando mensagens...
```

Teste entre cliente e servidor:

Os testes serão realizados apartir da cordenada: **(-22.224759, -54.782080)**, imagem no mapa.

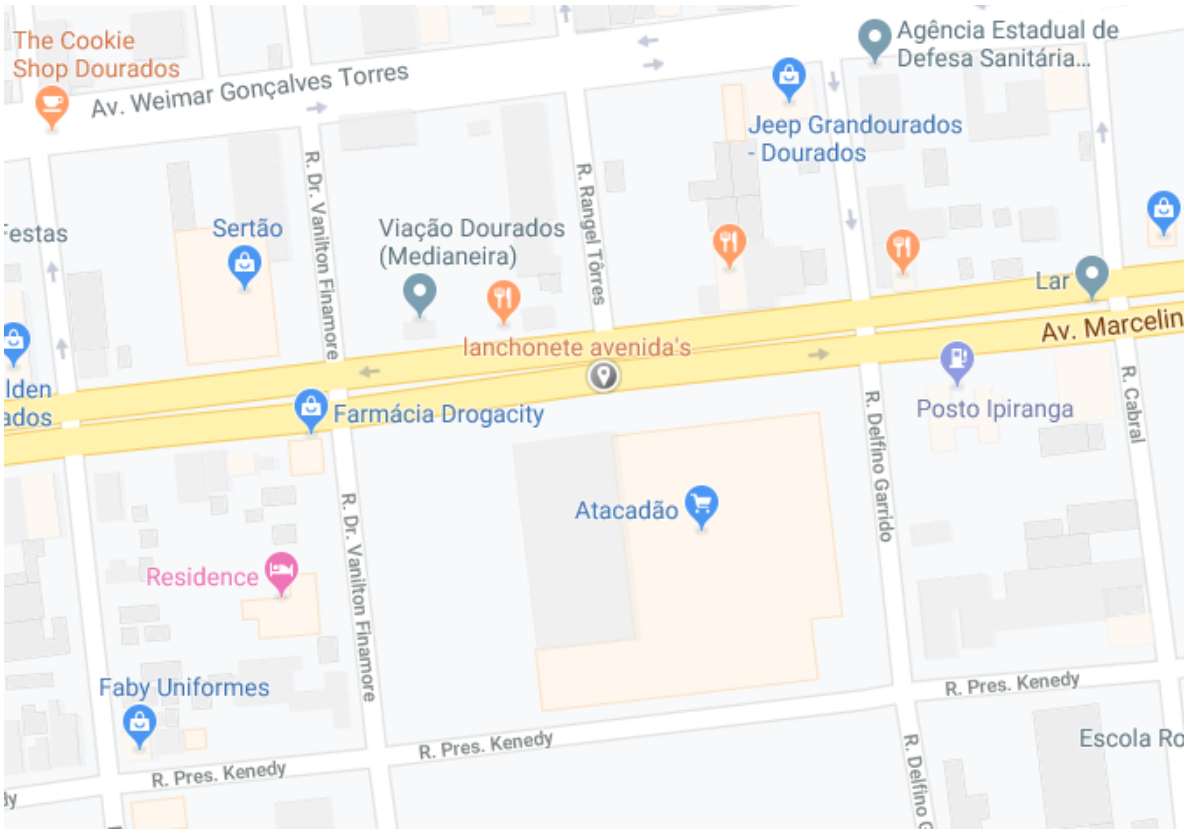


Tabela de postos de combustiveis com preço do mesmo e distancias da origem (-22.224759, -54.782080). Distancia calcula no site: https://www.sunearthtools.com/pt/tools/distance.php

-22.224540 -54.780474 (Auto posto hipiranga)	Distancia da origem (x:y)	Preço
Gasolina	167.1 m	3,025
Álcool	167.1 m	2,099
Diesel	167.1 m	3,075

-22.224532 -54.779901 (Auto posto União)	Distancia da origem (x:y)	Preço
Gasolina	225.8 m	3,075
Alcool	225.8 m	3,001
Diesel	225.8 m	4,001

-22.225299 -54.786935 (Auto posto Paulistão 2)	Distancia da origem (x:y)	Preço
Gasolina	503.5 m	2,095
Alcool	503.5 m	2,091
Diesel	503.5 m	4,051

-22.218506 -54.787282 (Posto Shell Monte Alegre)	Distancia da origem (x:y)	Preço
Gasolina	877.8 m	4,095
Alcool	877.8 m	2,001
Diesel	877.8 m	3,021

-22.222177 -54.757705 (Autoposto Taurus)	Distancia da origem (x:y)	Preço
Gasolina	2.5261 km	1,095
Alcool	2.5261 km	3,081
Diesel	2.5261 km	1,921

Cliente 1 inserindo valores no servidor:

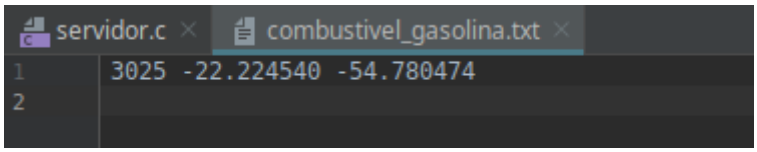
Cliente:

```
String a ser enviada ao servidor [UDP]
D 2 3025 -22.224540 -54.780474

[ACK 1] Recebido
:::::::::::::::: >> Resultado da sua busca:

"------------> Dados Salvos!

String a ser enviada ao servidor [UDP]
```

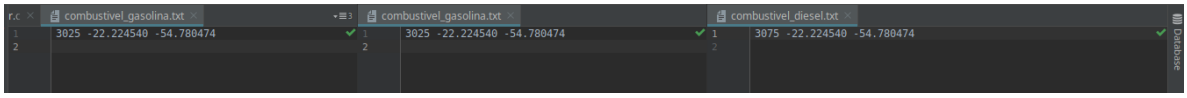
Servidor:

```
=========================================================
Pacote Recebido:
        Tipo ======> 0
        Mensagem ======> D 2 3025 -22.224540 -54.780474
        Checksum ======> 1550845373
=========================================================


Valor do checksum no servidor: 1550845373


$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ Comando do cliente: SALVAR DADOS $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


tipo combustivel: 2
mensagem: 3025 -22.224540 -54.780474
:::::::::::::: Enviando mensagem de confirmação para o cliente
Pacote recebido: 0
Pacote que devo receber: 0
:::::::::::::: [ACK 1] Enviado com sucesso.
```

Arquivo no servidor:



Resultado final dos saves do primeiro cliente:



Cliente 2 inserindo valores no servidor:

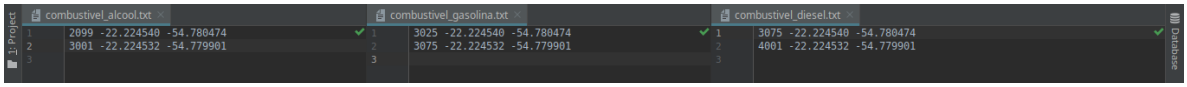
Cliente:

```
######## Enviando pedido de conexão... #######
Nova porta de conexão: 36732
Conexão estabelecida com sucesso




:::::::::::::::::: Conexão estabelecida com sucesso ::::::::::::::::::
String a ser enviada ao servidor [UDP]
D 2 3075 -22.224532 -54.779901

[ACK 1] Recebido
:::::::::::::::: >> Resultado da sua busca:

"------------> Dados Salvos!

String a ser enviada ao servidor [UDP]
```

Servidor:

```
Conexão do cliente na porta 36732 estabelecida com sucesso, PID do processo: 412

:::::::::::::: Identificador do cliente: 412
:::::::::::::: Aguardando mensagens...

============================================================
Pacote Recebido:
        Tipo ======> 0
        Mensagem ======> D 2 3075 -22.224532 -54.779901
        Checksum ======> 1561920966
============================================================


Valor do checksum no servidor: 1561920966


$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ Comando do cliente: SALVAR DADOS $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


tipo combustivel: 2
mensagem: 3075 -22.224532 -54.779901
:::::::::::::: Enviando mensagem de confirmação para o cliente
Pacote recebido: 0
Pacote que devo receber: 0
:::::::::::::: [ACK 1] Enviado com sucesso.



:::::::::::::: Identificador do cliente: 412
:::::::::::::: Aguardando mensagens...
```

Resultado final dos saves do segundo cliente:



Cliente 3 inserindo valores no servidor:

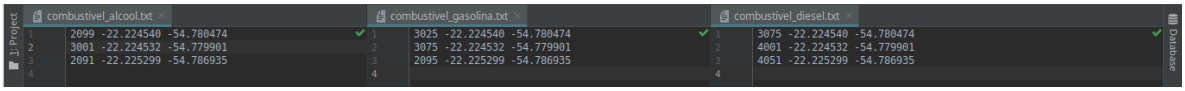
Cliente:

```
####### Enviando pedido de conexão... #######
Nova porta de conexão: 50028
Conexão estabelecida com sucesso




:::::::::::::::::: Conexão estabelecida com sucesso ::::::::::::::::::
String a ser enviada ao servidor [UDP]
D 2 2095 -22.225299 -54.786935

[ACK 1] Recebido
:::::::::::::::::: >> Resultado da sua busca:

"------------> Dados Salvos!

String a ser enviada ao servidor [UDP]
```

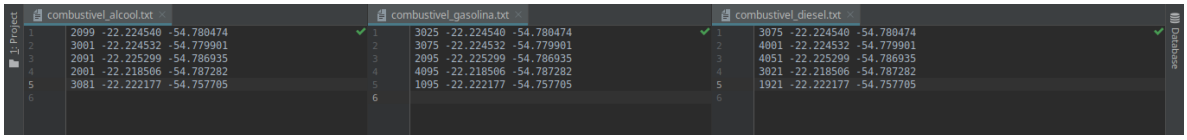
Servidor:

```
Conexão do cliente na porta 50028 estabelecida com sucesso, PID do processo: 695

:::::::::::::: Identificador do cliente: 695
:::::::::::::: Aguardando mensagens...

============================================================
Pacote Recebido:
        Tipo ======> 0
        Mensagem ======> D 2 2095 -22.225299 -54.786935
        Checksum ======> 1573520855
============================================================


Valor do checksum no servidor: 1573520855


$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ Comando do cliente: SALVAR DADOS $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


tipo combustivel: 2
mensagem: 2095 -22.225299 -54.786935
:::::::::::::: Enviando mensagem de confirmação para o cliente
Pacote recebido: 0
Pacote que devo receber: 0
:::::::::::::: [ACK 1] Enviado com sucesso.



:::::::::::::: Identificador do cliente: 695
:::::::::::::: Aguardando mensagens...
```

Resultado final dos saves do terceiro cliente:



Resultado final de todos os clientes apos saves no arquivo:



Cliente 1 Pesquisando Dados:

Pesquisa realizada: `P 2 1 -22.224759 -54.782080`

Resultado esperado: (Auto posto Paulistão 2) com gasolina ao preço de 2,095 e distancia de 0.5 Km

```
String a ser enviada ao servidor [UDP]
P 2 1 -22.224759 -54.782080

[ACK 1] Recebido
:::::::::::::::: >> Resultado da sua busca:

"-----------> O Posto de combustivel mais proximo de ti esta localizado a 0.50Km e se encontra nas coordenadas: (-22.225299;-54.786935) com valor de R$2.095
String a ser enviada ao servidor [UDP]
P 0 1 -22.224759 -54.782080
```

Pesquisa realizada: `P 0 1 -22.224759 -54.782080`

Resultado esperado: (Posto Shell Monte Alegre) com diesel ao preço de 3,021 e distancia de 0.8 Km

```
String a ser enviada ao servidor [UDP]
P 0 1 -22.224759 -54.782080

[ACK 0] Recebido
:::::::::::::::: >> Resultado da sua busca:

"-----------> O Posto de combustivel mais proximo de ti esta localizado a 0.88Km e se encontra nas coordenadas: (-22.218506;-54.787282) com valor de R$3.021
```

Pesquisa realizada: `P 2 10 -22.224759 -54.782080`

Resultado esperado: (Autoposto Taurus) com diesel ao preço de 1,095 e distancia de 2.5 Km

```
P 2 10 -22.224759 -54.782080

[ACK 1] Recebido
:::::::::::::::: >> Resultado da sua busca:

"-----------> O Posto de combustivel mais proximo de ti esta localizado a 2.53Km e se encontra nas coordenadas: (-22.222177;-54.757705) com valor de R$1.095
```

Tabela de log no servidor das consultas:

```
:::::::::::::: Enviando pacote com resposta para cliente
Pacote recebido: 1
Pacote que devo receber: 1
:::::::::::::: [ACK 0] Enviado com sucesso.



:::::::::::::: Identificador do cliente: 31313
:::::::::::::: Aguardando mensagens...

============================================================
Pacote Recebido:
        Tipo ======> 0
        Mensagem ======> P 2 10 -22.224759 -54.782080
        Checksum ======> 1377961319
============================================================


Valor do checksum no servidor: 1377961319

++++++++++++++++++++++++++++++++++++++++++++++
         Pacote de pesquisa:
                Raio: 10
                Tipo de combustivel: 2
                Latitude origem: -22.224759
                Longitude origem: -54.782080
++++++++++++++++++++++++++++++++++++++++++++++
```

Cliente 2 Pesquisa dados:

Cliente será pesquisado apartir de outra coordenada: **-22.233206, -54.764215**

Pesquisa realizada: `P 2 1 -22.233206 -54.764215`

Resultado esperado: NÃO HÁ POSTOS PERTO DE VOCÊ

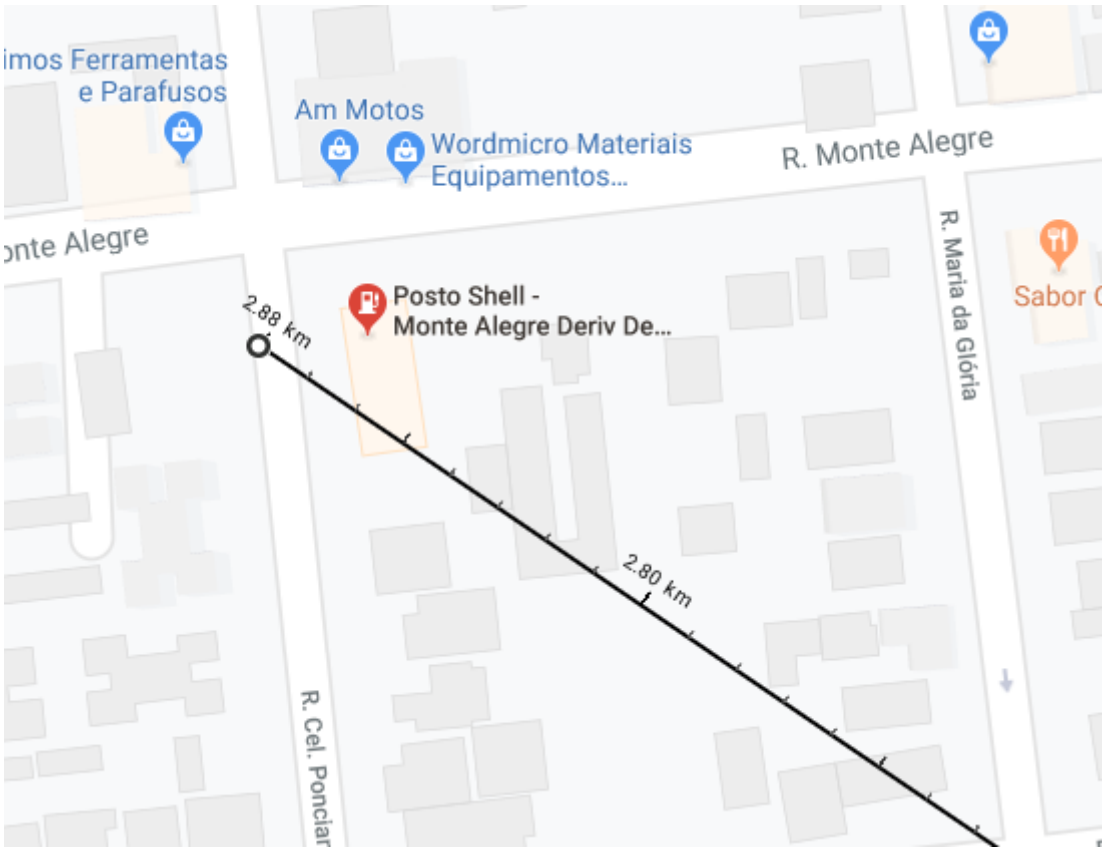
```
String a ser enviada ao servidor [UDP]
P 2 1 -22.233206 -54.764215

[ACK 1] Recebido
:::::::::::::::: >> Resultado da sua busca:

Ops, parece que não temos nenhum posto de combustivel perto de você :(
Tente aumentar o raio de busca ou alterar o tipo de combustivel
```

```
++++++++++++++++++++++++++++++++++++++++++++++
        Pacote de pesquisa:
                Raio: 1
                Tipo de combustivel: 2
                Latitude origem: -22.233206
                Longitude origem: -54.764215
++++++++++++++++++++++++++++++++++++++++++++++




Tabela de valores de todos os postos de tipo 2
Latitude: -22.224540, Longitude -54.780474, Preço: 3.025000
Latitude: -22.224532, Longitude -54.779901, Preço: 3.075000
Latitude: -22.225299, Longitude -54.786935, Preço: 2.095000
Latitude: -22.218506, Longitude -54.787282, Preço: 4.095000
Latitude: -22.222177, Longitude -54.757705, Preço: 1.095000

Não Há postos de combustiveis perto da sua região


:::::::::::::: Enviando pacote com resposta para cliente
Pacote recebido: 0
Pacote que devo receber: 0
:::::::::::::: [ACK 1] Enviado com sucesso.



:::::::::::::: Identificador do cliente: 412
:::::::::::::: Aguardando mensagens...
```

Pesquisa realizada: `P 1 10 -22.233206 -54.764215`

Resultado esperado: (Posto Shell Monte Alegre) com diesel ao preço de 2,001 e distancia de 2.88 Km

```
String a ser enviada ao servidor [UDP]
P 1 10 -22.233206 -54.764215

[ACK 0] Recebido
:::::::::::::::: >> Resultado da sua busca:

"-----------> O Posto de combustivel mais proximo de ti esta localizado a 2.88Km e se encontra nas coordenadas: (-22.218506;-54.787282) com valor de R$2.001
```



Referencias bibliograficas:

1. Protocolo Stop and Wait
   - [x] Geeks For Geeks - https://www.geeksforgeeks.org/stop-and-wait-arq/
   - [x] Google Imagens - https://www.google.com.br Pesquisa: Stop and Wait Retransmission

2. Distancia entre dois pontos em uma esfera
   - [x] Movable - https://www.movable-type.co.uk/scripts/latlong.html
   - [x] Geeks For Geeks - https://bit.ly/2XWTUGz

3. UDP Checksum
   - [x] Wikipedia - https://en.wikipedia.org/wiki/Adler-32
   - [x] Stack Overflow - https://stackoverflow.com/questions/3463976/c-file-checksum

4. Uteis
   - [x] Beej - http://beej.us/guide/bgnet/html/single/bgnet.html

5. Servidor UDP com Fork
   - [x] Justskins - http://www.justskins.com/forums/fork-udp-server-236700.html

- [x] Linuxforums - https://bit.ly/2LfTyVJ