# Documentação do TP1 de Rede de Computadores

Melkzedeck Miranda - 2018070775

## Introdução

Com base no contexto de pandemia da Covid-19, é de suma importância que seja usada uma tecnologia de Redes de Computadores para ajudar a combater o avanço de novas infecções, gerenciando postos de vacinação com a independência de um computador central físico. O Trabalho Prático 1 consiste em um sistema servidor cliente para troca de mensagens a fim de que as pessoas possam encontrar um local de vacinação a partir da sua posição geográfica.Os requisitos consistem em ser aplicado usando o protocolo TCP/IP e usar quatro tipos de mensagens para comunicação: *add* e *rm* para adicionar e remover, respectivamente, um local de vacinação, *list* para listar os locais de vacinação e *query* para encontrar o posto de vacinação mais próxima.Essa é a base do protocolo que será implementada no TP.

## Implementação

A implementação do software consiste em duas partes: comunicação em Rede e gerenciamento dos locais dos postos de saúde. Para a comunicação em Rede, foram feitas quatro Classes: Adress, Tclient, STclient e Tserver, usando a interface POSIX e soquetes de rede na linguagem C. O restante do trabalho foi implementado em C++.

A classe *Adress* serve para tratamento de endereços IP (IPv4 e IPv6) e portas usadas na comunicação na rede.

```
    /*Construtor da classe Adress usando a versão ('4'-(IPv4) ou '6'-
(IPv6))
     e a porta(string) como argumento - lado do servidor*/
    Adress::Adress(char v, std::string port_s);

    /*Construtor usando o endereco(string) e porta(string) - lado cliente*/
    Adress::Adress(std::string addrstr, std::string port_s);
```

A classe *Tclient* serve para inicialização, conexão e encerramento no final do programado soquete, e envio e recebimento de mensagens visando a comunicação do lado do Cliente.

```
    //Construtor do socket com endereco
    Tclient::Tclient(const Adress&);

    /* define um tamanho maximo de fluxo de mensagem da classe
    Tamanho default: 1024*/
    static void Tclient::setTAM(const int& t);

    /* fluxo de saída de string:
    envia para o socket a string
    e retorna a qntd de bits enviadas*/
```

```
    ssize_t Tclient::operator<<(const std::string&);

    /* fluxo de entrada de string: recebe uma msg do socket,
    coloca em uma string e retorna a qntd de bytes recebidas*/
    ssize_t Tclient::operator>>(std::string&);
```

A classe *STclient* (Server-Tclient) é uma classe derivada da classe Tcliente e é usada no lado do servidor,atuando na comunicação do servidor com o cliente. A motivação do uso da herança consiste na reutilização da interface por conta de similaridade em suas funcionalidades, e por ter uma única funcionalidade a mais que a classe pai.

```
class STclient : public Tclient{
    Adress addr_;
public:
    STclient(int, Adress);
    STclient& operator=(const STclient&);
    Adress addr();
};
```

A classe *Tserver* serve para inicializar, ligar (bind) a um endereço e deixar o soquete na escuta (listen) para novas conexões, visando a comunicação no lado do servidor e, no término de sua utilização, o fechamento do descritor do soquete. Quando um objeto da classe Tserver identifica uma nova conexão, ele retorna um objeto da classe STclient para que a comunicação seja estabelecida.

```
    //Construtor do socket com endereco
    Tserver::Tserver(Adress&);

    // Espera conexao e retorna o cliente conectado
    STclient Tserver::waitConection();
```

O gerenciamento dos postos de saúde é usado integralmente no programa *server*,e o programa *client* serve para que o usuário consiga enviar as mensagens pela rede. O servidor se comunica com somente um único cliente e após a sua desconexão, o servidor poderá se conectar novamente com um cliente mantendo seus dados,finalizando somente quando o cliente mandar a mensagem “kill”.

Foi criado um tipo de dado para representar os locais de vacinação e usado um contêiner para armazená-los:

```
//lado do servidor
typedef struct local{
    int x;
    int y;
}local;
std::vector<local> mundo;
```

# Desafios

Os principais desafios enfrentados nesse primeiro TP foi a familiarização de programar usando sockets e endereços de rede. A parte que lida especificamente com o protocolo TCP não foi muito desafiadora, pois depois de feita a conexão (com todas as etapas do servidor e do cliente), garantindo que não houve erro no processo, o restante é enviar e receber dados e depois fechar a conexão. Mas se familiarizar até mesmo com o próprio conceito de socket e descritor de arquivo, e entender o que são e como funcionam a parte de dados de endereço não foi uma tarefa fácil. Houve muitos imprevistos e desafios (além do comentado no parágrafo acima) no caminho mas, como deixei a documentação para o final do trabalho, não irei conseguir me lembrar de todos eles. Entretanto, dois deles foram mais importantes.

**Desafio de um dos construtores**

Um dos desafios foi fazer o construtor *Adress* usando um ponteiro sockaddr:

```
    //Construtor usando ponteiro do struct sockaddr
    Adress::Adress(struct sockaddr* addr);
```

A função accept() tem como argumento um ponteiro sockadd. A ideia desse construtor era facilitar a obtenção do endereço de quem se conectou usando esse ponteiro. Com paciência e checando a documentação dos socketes e do *struct sockaddr* constantemente, cheguei ao resultado final. Em termos de dificuldade, essa parte foi a que mais me deu trabalho, pois, além de considerar as duas famílias de IPs, foi a parte que inicialmente mais deu erro e a que mais tive que pensar e pesquisar.

**Imprevisto no recebimento dos pacotes**

Um imprevisto que aconteceu foi na comunicação dos sockets, no fluxo de entrada. A principal função responsável era escrita da seguinte maneira:

```
ssize_t Tclient::operator>>(std::string& str){
    char *msg;
    msg = new char[tam_max];
    ssize_t cnt = recv(socket_, msg, tam_max, 0);
    str = std::string(msg);
    delete msg;
    return cnt;
}
```

E esse código estava gerando lixo após a mensagem recebida, modificando o pacote. Para resolver, adicionei na quinta linha do código acima, o quanto de bytes que a string formaria usando o cnt, que informa quantos bytes foram recebidas:

```
    //cnt = num de bytes rebidos
    str = std::string(msg, cnt);
```

```
    //erro corrigido
```

Embora a solução seja simples, custou tempo, paciência e pesquisa para achar uma solução, pois estava estagnado em procura da resposta (sendo que era a única coisa em todo o trabalho que faltava corrigir para finalmente finalizar o trabalho).

## Conclusão

Esse primeiro trabalho prático foi desafiador. Entender o funcionamento dos soquetes na programação não é algo trivial. Precisa de tempo e paciência para abstrair os conceitos e entender o seu funcionamento. E esse desafio na minha opinião foi agregador, pois tive que pensar em resolver problemas presentes nos requisitos do TP, e pensar em como corrigir bugs que aconteciam e que às vezes, à primeira vista, não tinham o porquê de acontecerem.