Relatório Trabalho Prático 1

Redes de Computadores

Vitor Ribeiro dos Santos – 2017023676

Introdução

Nesse trabalho foi criado um CRUD de um sistema de vacinação. Nessa interação cliente/servidor o cliente que pode ser do grupo SAÚDE ou grupo CIDADÃO. Com isso o cliente pode enviar informações de coordenadas dos postos de vacinação, remover as coordenadas, listar as coordenadas cadastradas ou procurar a coordenada mais próxima daquela informada.

Implementação

Foram escritos 4 arquivos para implementar as funções requeridas, na linguagem C. O servidor e o usuário, que possui a função main e suas respectivas funções de funcionamento, além de um arquivo com outras funções necessárias para sua execução e seu cabeçalho. Por fim, há um makefile que compila os arquivos e gera os executáveis.

O arquivo cliente.c executa a função do cliente. Primeiramente cria um storage para o socket do cliente e o inicializa com uma função que está implementada no arquivo funcoes.c e depois inicializa o próprio socket. Se conecta, então, ao servidor. O cliente então entra em um loop onde ele pode realizar as ações de listar, remover, adicionar ou pesquisar sobre uma coordenada. Caso ele envie o comando "kill" no terminal sua conexão é totalmente interrompida com o servidor.

O arquivo servidor.c executa a função do servidor. É criado um storage para o socket do servidor, que é inicializado em uma função que está em funcoes.c e depois é inicializado o próprio socket. A porta passada como entrada é atribuída ao servidor, e então ele passa a esperar conexões. Existe um loop para conexão de clientes, em que um storage e um socket são criados e inicializados para o cliente, depois o servidor aceita a conexão, ele aguarda o comando enviado pelo cliente e com base nesse comando ele realiza a ação nos dados armazenados de forma síncrona no próprio código.

O servidor limita a quantidade de bytes transmitidos na comunicação em 500 bytes e foi definido o armazenamento de 50 locais de vacinação. Após chegar nesse valor e retornado uma mensagem de limite excedido para o usuário.

Foram criadas 4 funções básicas no servidor para auxiliar na ação de acordo com o solicitado no cliente e alguma funções auxiliares para ajudar na manipulação dos dados em cada função.

No arquivo funcoes.c foram colocadas funções necessárias para os demais arquivos. logexit ajuda a encontrar a localização de erros na execução. add_parse inicializa os atributos do cliente e addtostr para ajudar no log e a função server_sockaddr_init para iniciar o servidor. A função send_message recebe as informações que foram geradas no buffer pelo servidor e envia a mensagem para o cliente.

Estruturas

Foram utilizadas estruturas sockaddr_storage para armazenar os dados da estrutura sockaddr tanto para o cliente quanto para o servidor, já que a estrutura sockaddr é pequena, e assim evitamos preocupações com seu tamanho durante a execução.

Após ser instanciada sockaddr_storage passa por um cast para sockaddr_in ou sockaddr_in6, dependendo do protocolo usado. Essa estrutura é inicializada na funções. No caso do servidor, de acordo com o argumento recebido ele consegue alterar em IPV4 ou IPV6.

Para as mensagens trocadas pelo servidor e cliente foram usados vetores de chars, em que foi aplicado uma lógica para indetificar caso tenha um "\n" o final de cada mensagem e o servidor conseguir responder as solicitações de acordo com o tipo de pacote enviado.

A estrutura principal criada para armazenar os dados foi uma Struct nomeada Local que armazena as variáveis x e y de um coordenada:

```
typedef struct
{
    int x;
    int y;
}Local;
```

Sendo assim foi criado uma vetor de 50 posições dessa estrutura para armazenamento das coordenadas:

```
Local vaccination_coordenates[50];
```

Desafios, Dificuldades e Soluções:

Durante o desenvolvimento do trabalho encontrei um problema enorme para a manipulação de strings (vetor de caractere). Foram gastas muitas horas para compreender como o buffer chegava para o servidor através da função recv() e como era enviado para o cliente na função send().

O maior desafio foi tratar o caso de múltiplas mensagens, pois não tinha compreendido como eu formaria um comando com uma informação possivelmente incompleta vinda pelo cliente.

A solução foi utilizar um buffer auxiliar que guarda o restante da informação vinda do cliente e um contador que é alimentado com a quantidade de comandos encontrados e enviando a tratando o comando sempre que se encontrar um comando um válido.