TP1 - Pokédex

Jackson Nunes Silva - 2018054532

Novembro 2021

1 Introdução

O objetivo deste trabalho prático foi assimilar e exercitar os conceitos sobre programação com sockets e protocolos de comunicação vistos na disciplina Redes de Computadores. Para tal, foi proposta a implementação de um servidor de mensagens funcionando como uma Pokédex, cujo cliente é um treinador, que pode adicionar, remover, consultar e trocar pokémons.

2 Implementação

O programa foi implementado em linguagem C e divido em 4 arquivos: *server.c*, que possui toda a implementação do servidor e das estruturas de dados da Pokédex, sem modularização; *client.c*, que possui a implementação do cliente; *common.h* e *common.c*, que apenas implementam duas funções parecidas para inicializar a estrutura necessária *sockaddr_storage* para os dois tipos de endereço (IPv4 e IPv6), conforme o código visto em aula.

2.1 Estrutura de Dados

Para representar a Pokédex no servidor, um *struct* foi criado contendo dois elementos: um ponteiro para uma lista de pokémons e um inteiro, que guarda a quantidade de pokémons que aquele registro contém. A lista encadeada foi implementada de forma simples, de modo que cada nó possui um vetor do tipo *char* contendo o nome do pokémon representado e um ponteiro para o próximo nó. Para viabilizar as possíveis ações do cliente, foram criadas as funções básicas de uma lista, como inserir, remover e trocar pokémons, bem como um procedimento auxiliar que verifica se existe determinado pokémon na estrutura (*has_element()*).

Todo o controle de acesso à Pokédex é realizado inteiramente por uma única função, *access_pokedex()*, que recebe a ação desejada (mensagem enviada pelo cliente) e retorna um ponteiro do tipo *char* contendo a resposta já pronta para ser enviada de volta. Os comandos **add**, **remove**, **list** e **exchange** são executados e retornam a mensagem de confirmação ou a mensagem de erro (conforme os casos da especificação), o comando **kill** retorna a mensagem "kill"e um comando desconhecido retorna a mensagem "invalid". Desse modo, para cada chamada da função de acesso, o servidor verifica se o retorno é uma string "kill", caso em que encerra a própria execução e a do cliente ou "invalid", que apenas desconecta o cliente; caso contrário, a resposta é enviada.

2.2 Comunicação Cliente-Servidor

O protocolo para a comunicação entre o cliente e o servidor foi implementado conforme ensinado em aula. Portanto, a sequência de execução do servidor é: inicia socket e faz a abertura passiva com os comandos **bind** e **listen**, esperando um cliente se conectar com a função **accept**. Assim que um cliente se conecta, entra em um *loop* de troca de mensagens

até que o cliente caia ou envie comando para desconectar. Como não foi implementado um sistema de *threads*, o servidor atende apenas um cliente por vez.

Já em relação ao cliente implementado, o processo é semelhante: inicializa-se o *socket* e faz a abertura ativa com o comando **connect**. O restante é idêntico, pois entra no mesmo loop de envio e recebimento de mensagens com o servidor.

3 Desafios e Conclusão

Em geral, foi um trabalho tranquilo de se fazer, uma vez que não exigia a implementação de estruturas de dados complexas. O foco de desafio foi, portanto, no entendimento de como funciona a comunicação entre cliente e servidor, bem como a implementação prática utilizando a biblioteca *socket.h*. Por ter uma inicial insegurança em relação ao tema, foi decidido não implementar o acesso de múltiplos clientes ao mesmo tempo usando *threads*, visto que poderia dificultar a compreensão do básico da matéria.

No entanto, considero que ficou bem claro o funcionamento deste protocolo, o que já me da uma base para poder implementar o acesso de vários clientes simultâneos. Nesse aspecto, a realização desta tarefa foi muito positiva, possibilitando a visualização da teoria na prática e uma compreensão ampliada sobre o tema.