**Aluna:** Letícia da Silva Macedo Alves ; **Matrícula:** 2018054443

**TP1 Redes de Computadores**

**1. Introdução**

A proposta do projeto é implementar um sistema servidor-clientes no modelo TCP/IP utilizando soquetes e que simula a interação de um treinador de pokemons com sua pokédex, um dispositivo que armazenará os nomes dos pokémons. Neste sistema, o servidor atua como a pokédex enquanto os clientes são os treinadores. Uma decisão foi considerar que apenas um cliente pode estar conectado ao servidor de cada vez.

O protocolo de comunicação estabelecido conta com cinco ações possíveis. Através dos comandos adequados, o treinador pode adicionar e remover pokémons, listar todos os pokémons presentes na pokédex ou ainda trocar um pokémon por outro (trocando seu nome na pokédex). Além disso, caso o cliente envie a mensagem *kill*, o servidor encerra as atividades do sistema. Se o cliente enviar algum comando que não seja algum desses, sua conexão é encerrada pelo servidor, que aguardará uma nova conexão de cliente.

Com o sistema em funcionamento, o servidor imprime as mensagens recebidas do cliente e o cliente as respostas enviadas pelo servidor após o processamento de cada requisição.

A seguir serão devidamente tratadas informações relevantes como as dificuldades e decisões para os problemas encontrados ao longo do projeto.

**2. Especificações**

**2.1. Tecnologias utilizadas**

O trabalho foi implementado na linguagem C, seguindo as especificações. A codificação foi realizada no Visual Studio Code utilizando a opção WSL (Windows Subsystem for Linux), a fim de garantir que o programa seja executado no sistema operacional Linux.

O compilador utilizado foi o gcc e, para verificar o uso correto da memória, sem vazamentos, foi utilizado o Valgrind.

**2.2. Intruções de compilação e execução**

O programa faz uso de um Makefile. Assim, para a compilação é necessário apenas um comando “make”, que gerará os arquivos binários necessários, na pasta raiz do projeto.

Para execução, serão necessários um terminal para o servidor e outro para o cliente. O servidor é iniciado pelo comando *./server* <*v4/v6> <porta>*. Já o cliente, se conecta ao servidor através de *./client 127.0.0.1 <porta>* ou *./client ::1 <porta>*, respectivamente (a depender se o protocolo usado é o IPv4 ou o IPv6, representados pelos tipos de endereço v4 e v6).

A <porta> para o recebimento de conexões é um número e deve ser a mesma para o servidor e para o cliente. Uma sugestão é, por padrão, usar a de número 51511.

**3. Desenvolvimento e implementação**

**3.1. Decisões de projeto e limites estabelecidos**

* Apenas um cliente pode estar conectado ao servidor de cada vez
* Cada mensagem possui no máximo 500 bytes
* A codificação dos textos das mensagens utiliza a tabela ASCII e apenas letras (minúsculas), números e espaços podem ser transmitidos
* Os nomes dos pokémons têm no máximo 10 caracteres e devem ser compostos apenas por letras minúsculas e números (a verificação da validade dos nomes é feita pela função *bool validarNome(char\* nomePokemon)* do arquivo *uteisServidor.c*)
* A pokédex é a mesma para todos os possíveis clientes
* A pokédex existe apenas durante a execução do servidor e, caso a execução seja encerrada, ela é perdida, juntamente com os pokémons armazenados
* A pokédex só armazena no máximo 40 pokémons ao mesmo tempo
* As operações são escritas em letras minúsculas
* Para as operações que exigem nomes de pokémons, assume-se que nomes serão passados ao comando (sendo válidos ou não). Assim, nunca será enviado um comando apenas “*add”*, sem que seja seguido por um nome, por exemplo
* Caso o cliente solicite uma operação desconhecida, ele será desconectado do servidor, sem receber nenhuma mensagem de erro

**3.2. Alterações iniciais do código base**

A implementação teve por base o código disponibilizado durante a aula do professor Ítalo Cunha. Essa aplicação já utilizava o protocolo TCP para aceitar endereços IPv4 e IPv6 e permitia que o cliente enviasse apenas uma mensagem ao servidor, que, por sua vez, enviava de volta ao cliente a mensagem que foi recebida e depois as conexões eram encerradas.

Para se adequar às especificações do TP1, o código teve que passar por várias alterações e foram adicionadas as funcionalidades exigidas pelo problema.

A pokédex foi implementada como uma lista duplamente encadeada de pokémons a fim de facilitar as operações de adição e remoção de pokémons. Há uma *struct Pokedex* que possui um atributo *numPokemons* e dois ponteiros, um que aponta para o *primeiro* e outro para o *último* pokémon dessa lista. Já a *struct Pokemon*, conta com um *nome* de, no máximo, 10 caracteres e um ponteiro para o *anterior* à ele e outro para o *próximo* pokémon da lista.

Além dos arquivos *server.c*, *client.c*, *common.h* e *common.c* padrões, foram usados 4 arquivos auxiliares para tornar o código mais legível e organizado, seguindo boas práticas de programação. São eles: *uteisServer.h*, *uteisServer.c*, *uteisClient.h* e *uteisClient.c*.

O objetivo do trabalho era fazer com que o servidor recebesse mensagens do cliente e tratasse devidamente as requisições recebidas. Dessa forma a pokédex pode ser manipulada e o cliente recebe as mensagens adequadas em cada cenário possível (válido ou inválido).

Primeiramente, para que o cliente não fosse desconectado logo após a primeira troca de mensagens, foi utilizado um loop *while* que garante que o treinador de pokémons possa enviar ao servidor quantas requisições de operações quiser enquanto o cliente ainda estiver conectado. A troca de mensagens dura até que um dos três cenários ocorra:

* O cliente envia uma mensagem com o comando *kill* (explicado mais à frente)
* O servidor é fechado. Então o cliente também será desconectado.
* O cliente é fechado. Neste caso, o servidor continua funcionando e aguardando uma nova requisição de conexão de um cliente.

O servidor também conta com um loop *while* que verifica a condição *clienteConectado* para garantir o mesmo funcionamento de múltiplas trocas de mensagem especificado acima.

Os buffers tanto do cliente quanto do servidor são arrays de char de tamanho 500 posições (tamanho máximo de uma mensagem).

**3.3. Operação Adicionar Pokémon**

A solicitação é feita através do comando ***add <nome>*** e no máximo quatro pokémons podem ser adicionados ao mesmo tempo em um mesmo comando, ou seja, o formato pode se estender até ***add <nome1> <nome2> <nome3> <nome4>***. Caso ocorra, os demais nomes passados após o quarto serão desconsiderados e não serão adicionados.

O servidor faz então as seguintes verificações ordem (para cada pokémon passado):

1. Se o nome do pokémon é válido (isto é, se possui até 10 caracteres e se é composto apenas por letras minúsculas e números)
   * Se não for válido, é retornada a mensagem ***invalid message***
2. Se a pokédex não está cheia (isto é, se possui até 39 pokémons)
   * Se estiver cheia, ***limit exceeded***
3. Se o pokémon em questão ainda não existe na pokédex

* Se já existe, ***<nome> already exists***

Caso passe em todas as verificações, é retornado ao cliente ***<nome> added***.

Como um exemplo de retorno, considerando o caso em que a pokédex possui 39 pokémons. O treinador deseja adicionar dois novos pokémons, de nomes pikachu e raichu, e envia ***add pikachu raichu*** ao servidor. Então a mensagem que será recebida pelo cliente após o processamento da requisição é ***pikachu*** ***added limit exceeded***, já que, ao adicionar o pikachu, a pokédex atingiu o limite de pokémons armazenados, que é 40.

**3.4. Operação Remover Pokémon**

A solicitação se dá através da mensagem ***remove <nome>***. Se mais de um nome for passado, só será removido o primeiro pokémon do comando. Os demais nomes serão desconsiderados e não serão removidos.

O servidor faz então as seguintes verificações em ordem:

1. Se o nome do pokémon é válido (isto é, se possui até 10 caracteres e se é composto apenas por letras minúsculas e números)
   * Se não for válido, é retornada a mensagem ***invalid message***
2. Se ele não existe na pokédex
   * Se de fato não existir, ***<nome> does not exist***

Caso passe em todas as verificações, é retornado ao cliente ***<nome> removed***.

**3.5. Operação Consultar Pokédex**

Para ver uma lista com todos os pokémons contidos na pokédex em um dado momento, basta enviar o comando ***list***. O servidor verifica se a pokédex está vazia. Se estiver, retorna a mensagem ***none***, senão, os nomes dos pokémons são enviados no formato ***<nome1> <nome2> (...) <nomeN>***.

Quaisquer outros caracteres enviados depois do comando *list* serão ignorados.

**3.6. Operação Trocar Pokémon**

Esse comando troca o nome de um pokémon por outro na pokédex através de ***exchange <nome1> <nome2>***, sendo o primeiro nome correspondente ao pokémon que se deseja trocar e o segundo nome é o pokémon que será adicionado na pokédex no lugar do primeiro.

O servidor faz então as seguintes verificações em ordem:

1. Se o nome de ambos os pokémons são válidos (isto é, se possuem até 10 caracteres e se são compostos apenas por letras minúsculas e números)
   * Se pelo menos um dos nomes for inválido, é retornada a mensagem ***invalid message***
2. Se o pokémon a ser trocado existe na pokédex
   * Se não existir***, <nome1> does not exist***
3. Se o segundo pokémon passado para o comando ainda não existe na pokédex
   * Se ele já existe, ***<nome2> already exists***

Caso passe em todas as verificações, é retornado ao cliente ***<nome1> exchanged***.

**3.7. Comando kill**

Há ainda uma forma de encerrar tanto o cliente quanto o servidor. Para isso, basta que o cliente envie ***kill***. O servidor, ao receber tal comando, fechará a conexão do cliente (que deixará de executar) e depois encerrará sua própria execução.

**4. Principais desafios e problemas encontrados**

Os maiores desafios do trabalho foram: modificar a aplicação inicial para permitir a troca de múltiplas mensagens entre servidor e cliente e debugar o código. Durante os testes manuais de execução deparei-me diversas vezes com o famoso “segmentation fault” e a lógica por detrás desse erro às vezes não era nada simples e levava longos minutos até encontrar o problema.

Houveram também, ao longo do processo, erros de compilação pouco intuitivos envolvendo ponteiros, que exigiram uma boa revisão na linguagem C.

**5. Testes realizados**

O sistema foi submetido, primeiramente, a testes manuais após o fim da implementação de cada funcionalidade. Para cada possível cenário de envio de mensagens ao servidor, foram testadas todas as possibilidade de mensagem, inválidas ou inválidas, a fim de garantir que as respostas enviadas do servidor para o cliente (mensagens de sucesso e mensagens de erro em caso de operações inválidas) estavam de acordo com o esperado.

Ao final da implementação do sistema como um todo, o código também passou pelos testes disponibilizados pelas monitoras no GitHub através do link [*https://github.com/dekinks/TestesTP1*](https://github.com/dekinks/TestesTP1). O resultado foi a obtenção de sucesso em todos os casos de teste. Logo, o servidor desenvolvido mostrou-se compatível com o ambiente de testes tanto para os cenário nos quais cada mensagem é, provavelmente, enviada em apenas um pacote quanto naqueles em que a mensagem é enviada em mais de um pacote (recebida pelo servidor por mais de um *recv*).

**6. Considerações finais e conclusão**

Os maiores desafios são os que conferem os maiores aprendizado. Através deste projeto pude aplicar os conceitos vistos até o momento na matéria e entender mais sobre o funcionamento de um sistema cliente-servidor. Além disso, revisei várias especificidade da linguagem C e busquei codificar utilizando boas práticas de programação, como manter o código organizado e escolher nomes de funções e atributos descritivos e autoexplicativos. Aprendi bastante com minhas dificuldades.

Acredito que o sistema foi bem sucedido, visto que passou por todos os testes. Para cada mensagem enviada pelo cliente, o servidor é capaz de tratá-la, realizando as operações correspondentes e retornando ao treinador de pokémons a mensagem adequada (de sucesso ou de erro).

**7. Materiais de referências consultados**

* Livro: Linguagem C Completa e Descomplicada, do autor André Backes
* Capítulos 2 e 3 do livro sobre programação com sockets (disponibilizados no Moodle)
* Playlist de programação com sockets do professor Ítalo Cunha (<https://www.youtube.com/playlist?list=PLyrH0CFXIM5Wzmbv-lC-qvoBejsa803Qk>), na qual um código base para um sistema simples servidor-cliente foi apresentado