MC833 - Relatório do Projeto 2

Nome: Felipe Escórcio de Sousa - **RA:** 171043
Nome: Ricardo Ribeiro Cordeiro - **RA:** 186633

Junho de 2021

1 Introdução

O intuito deste trabalho é prover uma primeira experiência com a programação de um servidor UDP iterativo. Para isso foi proposto este projeto que implementa um servidor UDP básico, baseado no servidor TCP implementado no Projeto 1, e que também realiza o cadastro, listagem, edição e exclusão de dados de alunos formados em uma universidade fictícia. Também é proposto a programação de uma plataforma também simples a ser usada pelo usuário, implementando UDP também, que realiza as requisições ao servidor, localmente ou remotamente.

2 Descrição geral e casos de uso

Primeiramente, para rodar são necessários os seguintes requisitos:

- LibMongoC
- Docker-Compose ou MongoDB rodando localmente

E para um guia simples para instalação das mesmas, consulte o apêndice B deste relatório.

Já para compilar basta utilizar o comando Make na pasta do projeto ou nas pastas específicas do cliente ou servidor para compilação de apenas alguma parte.

O servidor é configurado para aceitar datagramas UDP. A partir do estabelecimento do socket, o mesmo espera pelas operações a serem enviadas pelo client. Por padrão a porta configurada é a `3490`, mas para que se configure a porta a ser utilizada basta passar a porta como parametro na hora da execução do programa e a url do banco de dados junto da porta usada por este, por exemplo:

```
$ ./server 4242 mongodb://localhost:27017
```

E se for necessário rodar uma instância do banco de dados localmente, sem instalá-lo, basta rodar:

```
$ docker-compose up -d
```

Para que seja possível rodar o servidor é necessário que de alguma forma haja alguma conexão com algum banco MongoDB, local ou remoto.

O cliente, por sua vez, pode apenas mandar datagramas ao servidor, podendo enviar comandos e dados para o lado do servidor. Há também a possibilidade deste ver um menu com ajuda, só que desta vez não há a opção terminar a operação do seu lado, mas apenas de fechar o processo (não há conexão). Para se configurar um IP de uma máquina remota (podendo ser IPv4 ou IPv6) e a porta, basta passar como parâmetro o endereço seguido da porta na hora da execução, por exemplo:

```
$ ./client 123.0.1.23 1234
```

As operações realizadas pelo servidor são:

1. Registrar perfil
2. Adicionar novas experiências à um perfil existente
3. Listagem por curso
4. Listagem por habilidade
5. Listagem por ano de graduação
6. Listagem de todos os perfis
7. Encontrar por email
8. Deletar perfil

Sendo as operações de criação, edição e remoção de registros restrita ao administrador do sistema. O username que precisa ser enviado para realizar estas operaçãoes é apenas `admin`, por simplicidade.

O Cliente é quem envia mensagens requisitando cada operação, utilizando um menuzinho textual simples, além das operações locais de consultar ajuda e fechar o processo. O servidor não tem poderes de gerar comandos para o cliente.

3 Armazenamento e estrutura de dados

Primeiramente, a razão da escolha de MongoDB para gerenciamento de dados sobre outras implementaçãoes se dá ao fato de simplificar razoavelmente o tratamento dos dados, dadas as naturezas dos dados e operações

requeridas pelo enunciado do trabalho. Porém, apesar de deixar o tratamento dos dados mais simples, a instalação do driver necessário requereu algumas adaptações no comando de compilação, necessitando incluir as libs manualmente e linkar os executáveis.

Uma das principais features do MongoDB, por ser um banco de dados não relacional, é salvar todos os dados como documentos JSON, de fácil leitura, edição e gerenciamento.

A principal simplicação que essa escolha trouxe foi no envio de mensagens, essas podendo ser convertidas automaticamente em strings e enviadas, além de facilitar a edição e deleção. As informações serão formatadas da forma de um BSON, dessa maneira:

```
{
    email: <string>,
    name: <string>,
    surname: <string>,
    residence: <string>,
    graduation: <string>,
    graduationYear: <string>
    skills: <string>,
    experiences: <array<string>>
}
```

Sendo essas limitadas apenas pelos tamanhos máximos de vetor estabelecidos no programa cliente, ou no arquivo compartilhado. No geral, apenas para formatação de mensagens ou de tratamento no banco de dados que os dados são trabalhados diretamente como JSONs, sendo no envio e recebimento, convertidos em strings. Devido a isso, não há definições de structs ou tipos novos.

4 Divisão do código

Mais uma vez, a codificação é separada principalmente em 4 arquivos `".c"` em específico, são esses:

4.1 server.c

Esse arquivo é responsavel por implementar as interações do servidor, é neste arquivos que recebemos os datagramas do cliente, e coordenamos o fluxo do programa, seja resolvendo no próprio `server.c` ou enviando requisições para o banco de dados.

4.2 db.c

Esse arquivo é responsável por implementar as funções de comunicação com o banco de dados (MongoDB). Utilizando a bíblioteca `libmongoc` podemos implementar as operações básicas de um CRUD como: Registrar um Perfil, Listar por determinado parâmetro, adicionar uma nova experiência à um perfil existente e deletar um perfil.

4.3 client.c

Esse arquivo é responsável por ser a gerar o programa de interface com o usuário, além de se comunicar com o servidor, de modo que ao receber algum dado, realiza o feedback com o dado requisitado pelo o usuário.

4.4 shared.c

Esse arquivo é responsável por manter as funções, constantes e definições compartilhadas entre o cliente e o servidor, evitando replicação de código e tornando o processo de debug e correção mais simples.

5 Implementação

A implementação dos sockets se deu por funções já implementadas por bibliotecas nativas do C. A configuração, tanto para o cliente quanto para o servidor, acontece de maneira muito semelhante, primeiro, é criado um socket mas apenas o servidor é ligado à uma porta . No caso do servidor, ele passa a escutar este socket, esperando por datagramas que podem chegar, quando algum chega, ele o trata. Para o cliente, após isso, este já pode começar a operar as funções estabelecidas.

Operação	**Descrição**	**Formato dos dados**
0	Inserir novo perfil	[OpCode][Id][Dados]
1	Inserir novas experiências em um perfil existente	[OpCode][Id][Email] [Dados]
2	Listagem por curso	[OpCode][Curso]
3	Listagem por habilidade	[OpCode][Habilidade]
4	Listagem por ano de graduação	[OpCode][Graduação]
5	Procurar por email	[OpCode][Email]
6	Listar todos os perfis	[OpCode]
7	Deletar um perfil	[OpCode][Id][Email]

É interessante comentar que o programa rodou em máquinas que utilizam de extremidade *Little-Endian* (sistema operacional Linux em x86), assim como a utilizada pela rede(modelo OSI), tal que é necessário atentar isso ao ler dados inteiros ou mesmo caracteres de forma correta. Não foi possível testar a comunicação utilizando dois computadores com extremidades diferentes para aferir se pode ser necessário fazer a reordenação dos bits dos dados recebidos.

6 Diferenças notadas entre TCP e UDP

6.1 Confiabilidade

De início podemos ressaltar que esse segundo projeto geralmente não é pensado para ser feito utilizando o protocolo UDP, já que o mesmo não garante a confiabilidade necessária na transmissão de dados que o escopo do projeto exige. De toda forma, utilizamos UDP nesse segundo projeto, apenas para fins didáticos porém com total consciência de que não é o protocolo mais recomendado para os requisitos.

Intrinsicamente podemos ver que o protocolo TCP se demonstra mais confiável que o UDP, já que o protocolo UDP não garante a confiabilidade na transmissão dos dados. Nesse projeto julgamos que a confiabilidade seja vital, já que há transmissão de dados cadastrais, sendo importante que os dados sejam trafegados sem perdas de uma ponta a outra. Além disso, outra característica do UDP é que não é possível assegurar que os datagramas chegarão na mesma ordem enviada, o que pode não ser um problema para um projeto pequeno como esse, mas que pode impactar em projetos de grande escala.

Outro ponto importante é a possibilidade do cliente e o servidor ficarem bloqueados, pelo lado do servidor ficará bloquedo até receber alguma informação vinda do cliente, sendo importante ressaltarmos que isso não é um problema já que o mesmo não teria nenhuma outra tarefa, e assim que recebe um informação ele executa o fluxo normal. Porém, o mesmo não é verdade do lado do cliente, já que ao se encontrar bloqueado o cliente aguarda uma resposta, de modo que nos casos onde o request é perdido, ou o reply é perdido o cliente não recebe a mensagem tornando-se bloqueado até que receba a mensagem solicitada, cabendo a nós tratarmos esse comportamento adicionando configurações para definir um tempo para timeout (configurado em 2,5s de limite). Nem sempre pro cliente é garantido que a operação requisitada pode ser executada, tornando a aplicação no geral menos confiável para o usuário.

Ainda em confiabilidade notamos que qualquer processo por enviar datagramas para a porta do cliente, sendo uma possível insegurança. O ponto disso se dá devido no receive from o cliente não checa se o ip que enviou o dado é o mesmo com o qual o cliente enviou a requisição, gerando essa abertura na segurança. Cabe assim, ao desenvolvedor checar se o endereço de ip recebido é o mesmo para qual foi enviada a requisição. Também não é garantido ao cliente saber mesmo se o servidor está funcionando ou se as mensagens estão apenas se perdendo.

6.2 Nível de abstração

Um outro ponto comporativo a ser análisado, é a relação entre Stream trazida pelo protocólo TCP e o Datagrama trazido pelo protocólo UDP. Pelo TCP temos a transmissão dos dados via Stream, de modo que longos fluxos de dados são quebrados em pacotes e depois remontados na outra ponta, por outro lado quando falamos sobre comunicação UDP, notamos que o envio de dados é feito por datagramas de modo que por ser um protocolo orientado a pacotes, a aplicação quebra os dados em pacotes e os envia para a outra ponta, sem a necessídade de ser feita a remontagem.

Nota-se também duas maneiras de tratar conexões, na tratativa de acesso concorrente, utilizada no TCP podemos ver que o servidor trata simultaneamente várias conexões, enquanto o UDP trata de forma iterativa, tratando uma conexão de cada vez. Note que assim como nos outros casos, há situações onde cada uma das tratativas tem seus beneficios, por exemplo, no caso do servidor iterativo, podemos ver beneficios quando temos serviços de curta duração, devolvendo de forma rápida e eficiênte uma requisição de baixo custo para o servidor.

7 Conclusão

Fica claro nesse projeto que cada protocolo tem sua melhor empregabilidade, desenvolver utilizando tanto TCP quando UDP nos permite compreender as diferenças de cada um dos protocolos e seus dividos usos. O trade-off mais importante para nosso projeto se passa principalmente na confiabilidade e segurança durante a transmissão de dados.

Para um projeto de cadastros de dados pessoais, é de extrema importante que os dados cadastrais do usuário sejam recebidos de forma integra e completa, fazendo sentido "pagar"o preço necessário da maior complexidade implicada pelo TCP. Todas as diretrizes adicionais do TCP como o processo de conexão, por exemplo, permitem que o TCP seja a melhor

solução empregada nesse tipo de projeto. Importante deixar claro que de forma alguma isso nos diz que um protocolo seja sempre mais efetivo que o outro, sendo sempre importante fazer o comparativo com o caso empregado. Um contra-exemplo onde o uso de UDP é mais efetivo seria caso estivessemos implementando um projeto de streaming, ou um projeto de comunicação como serviços semelhantes ao do skype/google meet, onde a informação deve chegar de forma rápida e com alta tolerancia na perda de dados. Análisando os requisitos do projeto compreendemos que utilizar o protocolo UDP seria a melhor solução para esse caso.

Concluimos que ambos projetos atendem seu objetivo de explicitar as principais diferenças nos protolocos, seus prós e seus contras, trazendo uma amostra do que seria um projeto real, assim como a importância de um engenheiro de redes ter o embasmento teórico e prático necessário que o permita tomar a decisão do melhor protocolo a ser empregado em cada caso de projeto.

A Referências

1. http://beej.us/guide/bgnet/translations/bgnet_ptbr.pdf
2. http://mongoc.org/libmongoc/current/tutorial.html

B Instalação de dependências

Link para instalação do `docker-compose`:

`https://docs.docker.com/compose/install/`

Link para instalação da lib do banco de dados:

`http://mongoc.org/libmongoc/current/installing.html`

É possível que seja necessário também realizar a instalação dos pacotes `*-dev`(Ubuntu) ou `*-devel`(Fedora).

C Possíveis problemas com a instalação do driver do banco de dados

É possível que as os headers da lib do banco de dados tenha sido instalado em algum diretório diferente dos usados na construção do programa, para isso verifique em que diretório dentro de `/usr` estão contidos os headers das libs e altere nos Makefiles necessários.