Documentação TP1 - Redes de Computadores

Guilherme Luiz Lara Silva
2019054633

Introdução

O trabalho prático utiliza de uma situação fictícia, a qual uma empresa provedora de infraestrutura em nuvem precisa reduzir a complexidade das suas operações. Temos 4 entidades principais e diversas regras para levarmos em consideração na solução final. As entidades em questão são:

1. Switch
2. Rack
3. Agente
4. Controlador

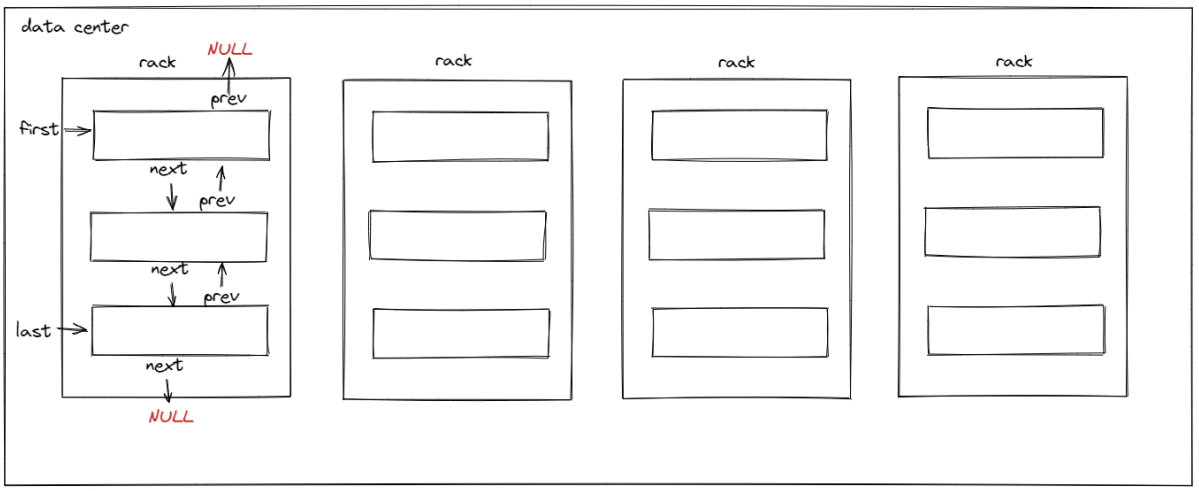
A solução final, consiste em na implementação de um protocolo chamado Flamingo, o qual deve suportar 4 operações:

1. Instalar switch
2. Desinstalar switch
3. Ler dados do switch
4. Listar equipamentos de um rack

Estruturas

A estruturação do código, conforme solicitado pela documentação, foi feita em 3 arquivos, *client.c, server.c* e *common.c*. Além destes, foi usado o arquivo common.h, para a declaração de funções e estruturas necessárias no arquivo *common.c*. A lógica do programa rodeia 3 structs principais, *dataCenter*, *rack* e *switch*.
A struct *dataCenter* é composta por 3 elementos, *size, max_size e racks*. O campo size reflete o atual tamanho do *dataCenter*, que compõe uma quantidade finita de racks, representada por *max_size*. Sempre que um rack é adicionado, o campo size é incrementado, e existe a lógica para que size nunca seja maior que *max_size*. Já o campo racks é um vetor de 4 elementos alocado de forma dinâmica do tipo *struct rack*.
A struct rack, assim como *dataCenter*, também tem os campos *max_size* e *size*, que possuem as mesmas respectivas funções, porém dessa vez o valor de max size é 3 ao invés de 4. Além disso, também temos o campo *id,* que armazena o id do rack, os campos *first* e *last*, do tipo *struct switchRack **, que são necessários para criar uma estrutura de lista encadeada, utilizada para que possamos iterar sobre os switches do rack.
Por fim, temos a struct switchRack, que representa os switches armazenados nos racks anteriormente citados. Essa struct possui os campos *id* (int), *message* (int), *prev* e *next* (*struct *switchRack*). Estes dois últimos citados são necessários para a criação de uma estrutura de lista encadeada, também necessária para a iteração entre switches “irmãos”.

O modelo abaixo representa bem o que foi explicado nos parágrafos acima.



Para acessarmos um rack, usamos o seu índice, que possui o mesmo valor que seu id - 1. Dessa forma, o acesso ao rack é O(1) e não O(n).
Sempre que necessário iterar sobre os switches de um rack, usamos um ponteiro auxiliar que a cada iteração é assignado para apontar para o próximo elemento da lista, até que ele chegue até o último elemento, e portanto, acabe apontando para NULL.

Checagens e Operações

Todas as 4 operações do protocolo flamingo foram implementadas. Antes de falarmos sobre as operações, vamos citar rapidamente as checagens existentes. Cada checagem será listada abaixo e a letra que a representa, será importante no momento de fazermos a relação de quais checagens são feitas em cada operação.

a. Rack não existe
b. Switch não existe
c. Rack vazio
d. Switch já instalado na Rack
e. Limite de Rack excedido
f. Tipo de Switch desconhecido

Muito bem, agora em relação às funcionalidades:

1. Instalar switch

Para que um switch seja instalado, é feito uma checagem dos itens ***d***,***e*** e ***f*** através do método *checkAddSwitchToRack.* Caso a checagem passe, faremos a adição dos switches para o rack em questão através do método *addSwitchToRack.*

2. Desinstalar switch

Para que um switch seja removido, é feito uma checagem dos itens ***d*** e ***f*** através do método *checkRemoveSwitchFromRack.* Caso a checagem passe,

faremos a remoção dos switches para o rack em questão através do método *removeSwitchFromRack.*

3. Ler dados do switch

   Para que um switch seja lido, é feito uma checagem dos itens ***a*** e ***b*** através do método *checkGetSwitchesData.* A checagem do item a foi feita por conta própria, já que não era necessária de acordo com a documentação. Caso a checagem passe, faremos a leitura dos switches para o rack em questão através do método *getSwitchesData.*

4. Listar equipamentos de um rack

   Para que os switches sejam listados, é feito uma checagem dos itens ***a*** e ***c*** através do método *checkListSwitchesFromRack.* Caso a checagem passe, faremos a remoção dos switches para o rack em questão através do método *listSwitchesFromRack.*

Desafios

Os maiores desafios desse TP certamente foram relacionados ao uso da linguagem C em si. O fato de termos de criar do zero as estruturas de dados a serem utilizadas foi algo que tomou bastante tempo, tempo este que poderia ter sido poupado caso usássemos uma linguagem como C++, que já oferece algumas estruturas como Vector e Map dentro da biblioteca padrão da própria linguagem.
Além disso, seria possível fazer um parse e assert mais preciso dos comandos enviados pelo cliente caso usássemos alguma linguagem com uma biblioteca de expressões regulares que usasse uma sintaxe mais recente. É possível fazer com que C tenha expressões regulares com uma sintaxe mais moderna e amigável, porém é necessária a instalação de uma biblioteca externa.
Outro desafio foi a questão de fazer a checagem antes da execução de um comando pelo fato de ter optado por usar variáveis do tipo char*. Toda vez que uma verificação era feita, o valor da variável, que seria usado para de fato executar o comando em questão, não era mais válido pois havia sido alterado pelo método que fazia a checagem. Dessa forma, tornou-se evidente a necessidade de criarmos uma cópia do valor original antes de fazermos a checagem, para que na hora da operação, tenhamos os valores intactos em uma variável.

Conclusão

É de suma importância exaltarmos a necessidade de aprendizado prático em um ambiente que preza tanto pelo teórico e abstrato, como é a universidade.
A oportunidade de execução de um trabalho como este, que visa a implementação prática da comunicação entre cliente e servidor, utilizando protocolos de rede - que foram vistos durante as aulas - é muito enriquecedor de um ponto de vista didático. Ver o programa dar certo ou errado, perceber as razões por trás de cada erro e fazer uma relação com o que foi dado em sala foi muitas vezes elucidador.