Documentação TP1

Redes de Computadores

Nome: Lucca Silva Medeiros
Matrícula: 2019054773

1) Introdução

O trabalho prático consiste na implementação de uma solução que permita a instalação, desinstalação e monitoramento de equipamentos de comutação de rede através de um software. Dado este cenário, foi implementado um novo protocolo chamado “Flamingo” que executa em um sistema cliente-servidor, simulando a interação entre o agente (cliente), responsável por fazer as solicitações que controlam e monitoram os equipamentos de rede, e o controlador (servidor), responsável por executar as mensagens solicitadas pelo cliente.

2) Implementação

A implementação do trabalho se deu através de duas etapas. A primeira consistiu na implementação do estabelecimento da comunicação entre cliente e servidor. Para isso foi utilizado POSIX sockets na linguagem C, utilizando a biblioteca padrão de sockets TCP. Essa etapa não apresentou muitas dificuldades para ser implementada, visto que o material de apoio presente na proposta do trabalho foi suficiente para entender todas as etapas que envolvem uma comunicação desse tipo:

- **Servidor**: Inicialização (socket), Abertura Passiva (bind, listen e accept), Comunicação (recv e send) e Finalização (close).

- **Cliente**: Inicialização (socket), Abertura Passiva (connect), Comunicação (send e recv) e Finalização (close).

O único desafio enfrentado durante essa etapa foi a respeito da permanência da conexão entre cliente e servidor. O protocolo foi desenhado visando que: dado o estabelecimento da conexão, o cliente deve ser capaz de enviar quantas mensagens quiser (desde que sejam mensagens válidas), até que se encerre a comunicação através da mensagem *exit*. Com isso, a primeira versão implementada encerrava a comunicação entre cliente e servidor toda vez que o servidor enviava uma mensagem de resposta.

Logo, foi necessário ler a documentação da biblioteca, e fazer uma modificação para atender o requisito do protocolo. Foi adicionado mais um loop no servidor, o qual fica responsável por manter a conexão ativa após o estabelecimento da mesma. Esse loop só é encerrado caso o cliente envie a mensagem de saída ou envie uma mensagem inválida.

A segunda etapa consistiu na implementação dos comandos do protocolo, são eles:

1. **Instalar Switch** - *add sw <switch_id> in <rack_id>*
2. **Desinstalar Switch** - *rm sw <switch_id> in <rack_id>*
3. **Ler dados do Switch** - *get <switch_id> in <rack_id>*
4. **Listar equipamentos de um Rack -** *ls <rack_id>*

Para garantir que cada comando respeite as regras de negócio e atenda os possíveis erros e restrições, foi pensada a seguinte metodologia de implementação. Para cada comando do protocolo temos:

1) **Validar o comando**

Essa parte consiste em tratar a consistência da mensagem recebida. Ela é responsável por verificar se a mensagem está digitada corretamente respeitando os devidos espaçamentos, quebra de linha e demais configurações estabelecidas pelo protocolo.

Essa etapa também, após confirmar que tudo foi digitado corretamente, armazena os parâmetros passados como o número do switch e o número do rack. Caso esses valores estejam fora do escopo do servidor (como rack_id e switch_id maiores que 4, por exemplo), essa etapa também retornará os devidos erros para esses casos. Em resumo, validar o comando garante que a mensagem passada para a próxima etapa seja passível de ser executada.

2) **Tentar o comando**

Uma vez recebido os parâmetros devidamente tratados pela etapa de validação do comando, essa etapa tenta de fato executar a requisição do cliente. Logo para cada comando temos uma tentativa e execuções específicas:

Instalar Switch: Tenta de fato instalar os switches no rack, verificando se o rack da mensagem já possui os respectivos switches instalados. Caso algum já esteja instalado, invalida a mensagem alertando o cliente.

Desinstalar Switch: Tenta de fato desinstalar o switch no rack, verificando se o rack da mensagem já possui o switch instalado. Caso não tenha um switch daquele tipo previamente instalado, invalida a mensagem alertando o cliente.

Ler dados do Switch: Tenta de fato ler o switch no rack, verificando se o rack da mensagem já possui o switch instalado. Caso não tenha um switch daquele tipo previamente instalado, invalida a mensagem alertando o cliente.

Listar equipamentos de um Rack: Tenta de fato ler todos os switches do rack, verificando se o rack da mensagem já possui pelo menos um switch instalado. Caso não tenha nenhum switch previamente instalado, invalida a mensagem alertando o cliente.

Em resumo, tentar o comando garante que a mensagem passada para a próxima etapa seja possível de ser executada.

3) **Efetuar o comando**

Uma vez recebido os parâmetros devidamente tratados pela etapa de tentativa do comando, a etapa final é de fato executar a requisição do cliente e responder com a mensagem de sucesso.

Visando simplificar a implementação, a estrutura de dados escolhida para representar o servidor foi a seguinte:

```
struct Rack_s { int sw[4]; int count; } Rack_default = {{0,0,0,0},0};
typedef struct Rack_s Rack;
Rack server[MAX_RACKS];
```

O servidor comporta um array de racks, onde cada rack é um struct que possui dois atributos: **count**, contador de quantos switches foram instalados no rack, e **sw[ ]**, array que representa a instalação dos switches no rack em questão.

Exemplos:
- **server[0].sw[0] == 1** - Implica que no rack 01 tem um switch do tipo 01 instalado.
- **server[1].sw[2] == 0** - Implica que no rack 02 não tem um sw do tipo 03 instalado.
- **server[2].count == 2** - Implica que o rack 03 possui 2 switches instalados.

Os maiores desafios na implementação dessa etapa foram: manipulação de strings e caracteres em C (objetivo que foi superado com muita pesquisa e leitura de documentação e exemplos da linguagem) e a garantia de que todas as exceções de erros fossem devidamente tratadas. Esse segundo foi superado definindo a abordagem descrita acima para cada comando suportado pelo protocolo.

3) **Conclusão**

O desenvolvimento deste trabalho possibilitou a aplicação prática de diversos conceitos estudados durante a disciplina de redes de computadores. Com a junção das duas etapas de desenvolvimento descritas acima foi possível implementar o protocolo de comunicação proposto e assim entender melhor como funciona a comunicação na prática via sockets no formato cliente-servidor.