Relatório - Servidor de Mensagens

Igor Roiz Teixeira

1. Introdução

A proposta do trabalho prático era desenvolver um sistema de troca de mensagens entre cliente e servidor, onde o cliente envia comandos para a manipulação de switches dentro dentro de racks, e o servidor executa essa manipulação e retorna uma mensagem para o cliente informando se a operação foi bem sucedida ou não.

O sistema deveria ser construído utilizando o protocolo TCP, na linguagem C e utilizando a biblioteca POSIX. Esse relatório descreve o processo de desenvolvimento desse trabalho e os principais problemas encontrados.

2. Desenvolvimento

O desenvolvimento do sistema foi dividido em três partes: comunicação, onde foi estabelecida a comunicação cliente-servidor e a troca de mensagens entre eles, interpretação, onde os comandos passados pelo cliente eram interpretados, e execução, onde os comandos passados eram executados realmente executados. Ao final do desenvolvimento de cada parte eram executados testes para garantir o funcionamento do sistema até então e as devidas correções eram aplicadas antes de prosseguir, fazendo assim um desenvolvimento incremental.

2.1. Fase de Comunicação

Nessa fase, foram desenvolvidos os códigos de cliente e servidor com o objetivo de que no final uma troca de mensagens simples ocorresse, com o cliente enviando uma *string* e o servidor devolvendo outra de confirmação de recebimento.

Essas funcionalidades foram desenvolvidas baseada nas video aulas e códigos disponibilizados pelo professor. Nesse processo, o servidor é inicializado primeiro e cria um *socket*, podendo utilizar tanto IPv4 quanto IPv6 conforme passado pelo usuário e então aguarda a conexão. O cliente então

realiza o mesmo processo, se conecta ao servidor e ambos lançam uma confirmação de conexão.

O servidor então passa a aguardar uma mensagem do cliente. Para essa parte, foi criado um *loop* infinito em ambas as partes. Primeiro o cliente lê uma mensagem passada pelo usuário e envia para o servidor que estava aguardando. O servidor imprime essa mensagem no seu próprio *log* e envia uma mensagem de confirmação para o cliente que ao receber também a mostra no seu *log*. Essa sequência é repetida até que o cliente envia a mensagem “exit”, ao receber isso o servidor sai do *loop* de troca de mensagens, envia uma última mensagem confirmando o encerramento da conexão e fecha o *socket*. Após receber essa confirmação o cliente também sai do *loop* e fecha o seu *socket*.

Apesar de ser a primeira experiência criando um sistema de troca de mensagens, essa fase não apresentou maiores dificuldades durante o desenvolvimento.

2.2. Fase de Interpretação

Nessa fase, a mensagem recebida pelo servidor é enviada para outra função, chamada “*execComando”* que irá tratá-la. Para isso foi utilizada a função “*strtok”* que divide uma *string* em várias partes com base em um delimitador passado como parâmetro, nesse caso foram usados o caractere espaço e o “\n” que pegava o final da mensagem.

Usando estruturas condicionais os comandos eram analisados a cada palavra, para os dígitos referentes a switches e racks é usada uma função *“isDigit”* que confere se aquela string se refere a um dígito com zero a esquerda como especificado nas instruções. Se a qualquer momento dessa checagem for encontrada um erro de escrita da mensagem (fora dos parâmetros descritos no trabalho), a função devolve para a função principal o número inteiro um que aciona a condição de encerrar o loop de troca de mensagens e encerrar a conexão enviando antes uma mensagem de erro para o cliente.

Caso as mensagens estejam corretas os números referentes aos switches e os racks são extraídos para variáveis inteiras e são repassadas para funções que executam as operações com esses elementos.

2.3. Fase de execução

A principal dificuldade no desenvolvimento do trabalho foi nessa fase, em relação a decisão de qual estrutura de dados usar para manipular racks e switches. Primeiro foi considerado usar listas encadeadas para ambos, uma dentro da outra, ou apenas para os racks, e os switches seriam representados por um vetor. No final, por conta da complexidade, e com base nas especificações, foi decidido criar um vetor do tipo *struct rack* que contém o identificador do rack e um vetor de inteiros para os switches que recebe zero para representar uma posição vaga para instalar um switch, ou o identificador do equipamento instalado. Essa estrutura possui uma série de funções para a sua manipulação.

As funções de execução então usam os números de identificadores dos racks e switches que recebem como inteiros para testar os critérios especificados na descrição do trabalho, como pré-existência de equipamentos e espaço para instalação dos equipamentos. Caso qualquer erro seja encontrado, uma mensagem para notificar o cliente é escrita, e a execução dessa função é interrompida para retornar a troca de mensagens. Passando por todas essas avaliações, a operação solicitada é executada utilizando as funções da estrutura dos racks, e a mensagem de confirmação da execução é escrita seguindo os parâmetros determinados.

3. Conclusão

O trabalho teve um bom fluxo de desenvolvimento seguindo o modelo incremental e realizando os testes necessários ao final de cada fase para garantir seu bom funcionamento, esse planejamento foi feito previamente permitindo facilitar as pesquisas necessárias em cada parte. O maior gargalo foi a decisão por quais estruturas de dados usar na fase de execução, todas as citadas foram experimentadas, mas o modelo de desenvolvimento permitiu mudar de uma para outra sem interferir na execução das outras fases já estáveis.