Caio Teles Cunha 2020006434

1. Introdução

A proposta do Trabalho Prático 2, da disciplina de Redes, foi implementar um servidor e um cliente, utilizando sockets, que pudessem se comunicar e executar um funções parecidas com a criação de um fórum online.

O servidor ficou com a responsabilidade de armazenar os tópicos que os usuários podem escrever, bem como os posts inscritos em cada tópico, lidar com os comandos enviados pelo cliente e sinalizar para os usuários inscritos nos tópicos que um novo post foi publicado.O servidor também deveria ser implementado de forma a receber múltiplos clientes, já que os fóruns online funcionam assim. Já o cliente foi responsável por exibir os erros caso o servidor recebidos do servidor, tratar o input do usuário e envia-lo corretamente ao servidor e receber as respostas do servidor de maneira constante exibindo no terminal quando necessário.

2. Implementação

O programa foi desenvolvido em um ambiente operacional linux, na linguagem C.

3. Estrutura de dados

Cinco estruturas de dados razoavelmente simples foram utilizadas para a implementação do programa.

A primeira é a descrita na especificação do trabalho chamada de BlogOperation. Essa estrutura é a passada pelo socket para fazer a comunicação entre cliente e servidor. Possui 3 campos de inteiros: um para o id do cliente, outro para o tipo de operação e outro para definir se é resposta do servidor ou não. Possui também 2 vetores de strings, um para o título do tópico e outro para o conteúdo do post.

A segunda estrutura foi definida apenas no servidor, é a postTopic. Ela é utilizada para armazenar um tópico de um post. Ela possui um vetor de char content que é responsável por armazenar a string que é o conteúdo do post. Possui também um ponteiro para um nextPost. Dessa forma, implementei para que cada tópico tivesse uma fila encadeada de posts.

A terceira estrutura, também definida apenas no servidor, é a Topic. Ela é utilizada para armazenar um tópico. Possui um vetor de char para o título do tópico, um vetor de inteiros para armazenar se o cliente com aquele id está inscrito no tópico, possui um apontador para um postTopic e um apontador para Topic. Dessa forma, no servidor, os tópicos são armazenados em uma fila encadeada. O servidor possui um apontador global para o primeiro elemento dessa fila.

A quarta e quinta estruturas,receiverClientData e clientData, são estruturas auxiliares que foram usadas para passar informações necessárias às threads criadas, tanto no cliente, quanto no servidor.

4. Discussão dos desafios

O primeiro desafio encontrado foi tornar o código compatível com os protocolos ipv6 e ipv4. O protocolo ipv4 tem um endereço de 32 bits, já o protocolo

ipv6 possui um endereço de 128 bits. A biblioteca usada possui estruturas diferentes para lidar com cada endereço, então foi necessário criar uma função que, sabendo qual o protocolo, inicializa as estruturas necessárias. É necessário utilizar o struct sockaddr como uma interface para as as structs sockaddr_in (ipv4) e sockaddr_in6 (ipv6). O struct sockaddr_storage também foi usado devido à sua funcionalidade de armazenar qualquer um dos endereços e depois escolhemos o tipo do endereço específico. As manipulações dessas structs para que o programa funcionassem de maneira correta foram complexas de entender e aplicar. Outra coisa que foi desenvolvida foi o entendimento de host byte order e network byte order. A porta, por exemplo, deve estar sempre em network byte order (big-endian) ao passar para a struct sockaddr_storage.

O segundo desafio encontrado foi tornar possível que o servidor recebesse múltiplos clientes ao mesmo tempo. Para resolver esse problema, segui a implementação usando threads, com a biblioteca pthreads, devido à minha familiaridade com a biblioteca por causa da matéria de sistemas operacionais. O servidor, assim que recebe uma conexão e aceita essa conexão, cria uma thread para lidar com a conexão e volta para o estado de aguardar conexão. Dessa forma o servidor pode atender vários clientes ao mesmo tempo. Devido à especificação do trabalho, mantive o número máximo de threads possíveis a serem criadas com o número 10.

O terceiro desafio for garantir a coerência dos dados. Como as várias threads trabalham sobre a mesma fila encadeada de tópicos, seria possível que duas threads,por exemplo, tentassem criar um tópico ao mesmo tempo e acabassem sobrescrevendo uma a outra. Dessa forma, dados dos usuários seriam perdidos e o programa não armazenaria corretamente os tópicos e posts. Para garantir então essa coerência foi usado um mutex, de forma que somente uma thread por vez pode acessar a fila encadeada de tópicos evitando que os dados não sejam armazenados corretamente.

O quarto desafio encontrado foi do lado do cliente. O trabalho pede que os clientes recebam os posts criados nos tópicos em que estão escritos, no entanto o cliente também deve poder digitar a qualquer momento um novo comando. A solução para esse desafio foi tornar o cliente também multithread. Criei uma thread responsável por receber as respostas do servidor e a main ficou por conta de receber constantemente os inputs do usuário e enviar para o servidor. Dessa forma, foi possível cumprir a especificação do trabalho.

4. Conclusão

Em resumo, o trabalho foi uma ótima forma de praticar e solidificar os conhecimentos recebidos em aula sobre a matéria de redes. Também foi de suma importância para o aprendizado da programação em sockets e como eles funcionam. Pude também neste trabalho aplicar conhecimento de outras matérias, como sistemas operacionais com as threads e mutex. Acredito que o trabalho foi essencial para me ajudar a compreender a matéria e me ajudar no estudo da mesma.