Trabalho Prático 2 - Medium

Isabella Vignoli Gonçalves - 2020006655
Redes de Computadores - Turma TN - Prof. Marcos Augusto
Universidade Federal de Minas Gerais

Introdução

O trabalho prático “Medium” da disciplina de Redes de Computadores trata de uma plataforma de publicação de conteúdo, ou seja, um blog. Utilizando funcionalidades da biblioteca de sockets POSIX e comunicação via protocolo TCP, a implementação do projeto permite que vários clientes se conectem a um servidor, publiquem posts em tópicos, recebam atualizações sobre novos posts em tópicos em que estão inscritos, visualizem a lista de tópicos disponíveis e realizem operações como inscrição e cancelamento de inscrição em tópicos.

Estrutura de Implementação

No contexto do projeto, o sistema é dividido em três partes principais: uma voltada para o **servidor** (*server.h e server.c*), que implementa as ações do servidor, gerenciando clientes, tópicos e postagens e utiliza threads para lidar com múltiplas conexões simultâneas; outra que implementa a lógica do **cliente** (*cliente.c*), interagindo com o servidor por meio de sockets e com uma thread para receber atualizações assíncronas do servidor; e, por fim, o **common** (*common.h e common*.c) que contém funções e estruturas compartilhadas por ambos.

Servidor

O servidor inicia configurando um socket usando “*server_sockaddr_init*” para aceitar conexões de clientes. O servidor associa o socket a um endereço usando *bind* e fica aguardando conexões com a função *listen*. A estrutura principal é a struct Blog, que contém arrays de *struct Topic* e *struct Client*, uma variável para contagem do número de tópicos no blog (*topics_count*), uma variável para contagem do número de clientes (*clients_count*) e um array de booleanos para os clientes (*bool_clients*). Esse último atributo é utilizado para identificar os clientes que estão conectados, facilitando a atribuição de ID para o menor inteiro disponível no momento.

Quando um cliente se conecta, o servidor cria uma nova thread (*threadsClient*) para gerenciar a comunicação com esse cliente específico. O uso de threads é fundamental para lidar com a concorrência de múltiplos clientes conectados simultaneamente ao servidor. Cada thread do cliente é responsável por aceitar comandos do cliente enquanto aguarda atualizações do servidor, o que é feito através da função “*receive_all*”. Essa função assegura que todos os bytes da mensagem sejam recebidos corretamente

O servidor trata cada tipo de operação, “publish”, “subscribe”, “unsubscribe”, “list topics” e “exit”, por meio da função *operationType*. A comunicação entre cliente e servidor ocorre por meio de trocas de mensagens usando a estrutura *BlogOperation*.

Cliente

Após estabelecer uma conexão com o servidor usando a função *connect* de sockets, o cliente cria uma estrutura *struct BlogOperation* representando uma solicitação de conexão ao servidor e a envia e o servidor responde com uma confirmação, atribuindo um ID único ao cliente.

O cliente inicia uma thread (*waitingThread*) para receber atualizações assíncronas do servidor, enquanto a thread principal continua aguardando comandos do usuário. Quando uma mensagem é recebida, a função *serverResponseHandler* é chamada para processar e exibir a resposta apropriada. A função *serverData* garante que se a mensagem recebida indicar que o cliente deve se desconectar, uma mensagem é impressa no servidor, o socket do cliente é fechado e a thread se encerra. A thread criada por essa função permite que o cliente receba atualizações em tempo real sem bloquear a thread principal, que entra em um loop infinito, aguardando comandos do usuário por meio da função *fgets*. Os comandos do usuário são processados pela função *inputCommand*, analisando a entrada do usuário e determinando o tipo de operação a ser realizada.

Common

Aqui são implementadas as funções compartilhadas entre o cliente e o servidor. Isso inclui funções para implementar o socket, de *logexit,* que é usada para imprimir mensagens de erro críticas no console e encerrar o programa quando ocorre um erro fundamental, como falha na inicialização do soquete ou na conexão, e uma função para criar uma estrutura do tipo Blog Operation (struct Blog Operation createBlogOperation).

```
struct BlogOperation {
   int client_id;
   int operation_type;
   int server_response;
   char topic[50];
   char content[2048];
}
```

Na struct BlogOperation, o “client_id” indica o ID do cliente que é conectado; o “operation_type” correponde a todas as ações que podem ser realizadas; o “server_responde” que define se a ação transmitida corresponde a uma resposta

vinda do servidor; o topic[50] identifica o nome do tópico que recebe a ação; e o content[2048] que indica o conteúdo do post feito no blog.

Desafios

Um dos principais desafios enfrentados no desenvolvimento do projeto foi compreender quais estruturas deveriam ser implementadas pelo servidor e quais deveriam fazer parte do cliente, principalmente no que tange as mensagens a serem impressas no terminal. Nesse contexto, a leitura e o processamento dos comandos de ação no terminal do cliente também foram uma dificuldade encontrada, mas que foram solucionadas pela função *inputCommand* no arquivo "*cliente.c"*.

Pensando na atribuição de IDs aos clientes quando uma conexão era iniciada, um ponto que demandou mais tempo e atenção foi garantir que caso um cliente se desconectasse, o próximo cliente a se conectar receberia o menor ID disponível e não o próximo número na sequência. Para isso, foi necessário criar um array de booleanos que identificasse os clientes atuais. A partir disso, se tornou possível buscar qual o primeiro ID disponível para ser utilizado.

No que diz respeito às postagens realizadas no blog, inicialmente foi difícil pensar em como funcionaria o processo de notificação de usuários inscritos em um determinado tópico quando uma postagem nova ocorresse. Nesse contexto, era necessário determinar uma implementação tal qual fosse impresso no terminal de todos os clientes inscritos que um novo post havia sido feito, indicando o cliente que o realizou e exibindo o conteúdo do post em questão. Além disso, um outro fator que também seria considerado é que caso o cliente deixasse de ser inscrito nesse tópico ele deveria parar de receber notificações de novos posts, problema que foi recorrente durante o processo. Isso foi resolvido utilizando a solução de criação de array de booleanos com o ID dos tópicos, assim como apresentado acima.

Devido ao fato de que vários clientes poderiam se conectar ao servidor e enviar comandos simultâneos para realizar ações no blog, a comunicação entre o cliente e o servidor exigiu mais cuidado e demandou um maior entendimento da manipulação de threads e sockets. A implementação da parte de threads foi um dos maiores desafios do código, mas, ao final, foi criada uma thread para receber mensagens do servidor, o que permitiu que o cliente continuasse a aceitar comandos do usuário enquanto esperava por atualizações do servidor. Esse processo precisou ser tratado em ambos cliente e servidor por meio das funções *threadsClient* e *serverData*. Ademais, a sincronização da comunicação entre cliente e servidor utilizando a função "receive_all" foi uma grande dificuldade, já que em muitas tentativas as informações e mensagens necessárias não estavam chegando corretamente, isto é, a estrutura BlogOperation não estava sendo atualizada juntamente com as ações que estavam sendo realizadas.

A listagem e criação de tópicos também foi desafiadora já que inicialmente alguns tópicos que eram criados não apareciam na lista dos tópicos existentes. O entendimento de que para tratar o caso em que nenhum tópico existia, isto é, quando era necessário imprimir a mensagem "no topics avaliable" no servidor era

preciso criar uma função que sempre buscasse pelo ID do tópico em questão no array de tópicos também gerou uma série de erros no início.

Por fim, uma parte importante que foi alvo de grande atenção ao longo do projeto foi garantir que ao encerrar a conexão com o servidor o cliente em questão fosse desconectado adequadamente, isto é, que a mensagem informando esse acontecimento fosse impressa no servidor, assim como o terminal do cliente fosse encerrado juntamente com o seu socket e a thread terminada.

Conclusão

Durante o desenvolvimento do trabalho foi possível entender melhor o funcionamento de threads associadas à utilização de sockets, em um contexto de comunicação entre cliente e servidor. A implementação do código permitiu a aprendizagem de estratégias e mecanismos para assegurar a sincronização das mensagens enviadas e ações realizadas dentro do blog. Portanto, foi necessário estabelecer uma lógica de processamento do input vindo do cliente de tal forma que as atualizações corretas ocorressem na estrutura BlogOperation após a realização das ações identificadas.

Instruções para execução e compilação

Após descompactar o arquivo, basta acessar o diretório em que o programa está armazenado com o comando "***cd <diretorio>***". Para compilar o programa basta digitar o comando "***make***" no terminal. Para executar o servidor utiliza-se o seguinte comando:

./bin/server <v4/v6> <porta>

Já para executar o cliente utiliza-se o comando abaixo:

./bin/client <IP> <porta>

É importante ressaltar que esse mesmo comando será utilizado para todos os terminais de cliente que forem ser conectados, sempre mantendo a mesma porta para que a conexão seja realizada com o mesmo servidor.