Documentação - TP2

1. Introdução

O problema proposto foi implementar um sistema de blog, semelhante a plataforma Medium, que permitirá com que vários clientes se conectem a um servidor, publiquem posts e recebam notificações de publicações em tópicos que seguem.

Para a implementação do jogo, criou-se um cliente e um servidor. O servidor é responsável por guardar os posts publicados no blog e os usuários inscritos em cada post. O cliente é responsável por publicar os textos e participar dos tópicos nas quais estão inscritos.

2. Funcionamento

Essa seção visa descrever, de maneira geral, o funcionamento do programa, tanto em relação ao servidor quanto ao cliente.

O servidor é responsável por criar a conexão e ouvir os clientes que se conectam a ele. Ele realiza esse processo com um número limitado de clientes, usando para isso o *multithreading*. Para cada cliente conectado, o servidor recebe um determinado comando, como salvar um post ou listar os tópicos criados, e realiza uma operação específica. O servidor identifica cada cliente com um id específico e único.

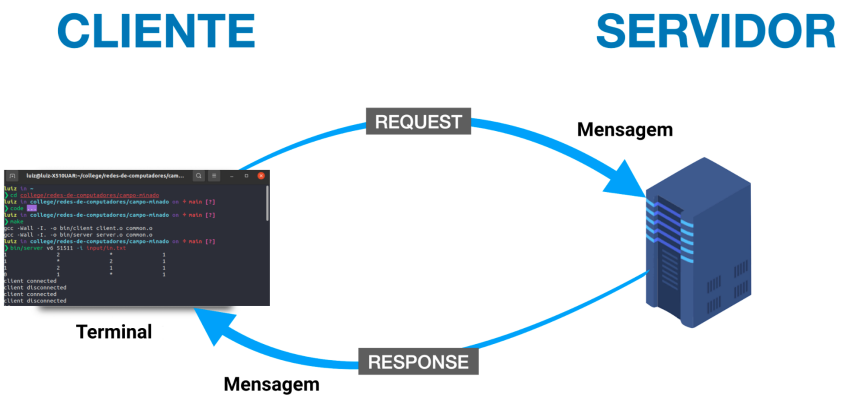
Cada post enviado para o servidor pertence a um tópico, e os clientes recebem uma notificação se um post for publicado em um tópico na qual está inscrito. Para armazenar esses dados, foram utilizadas duas tabelas *hash*, na qual a chave é o nome do tópico e o valor de uma é o conteúdo do post e da outra é o id do *socket* do cliente inscrito naquele tópico. Essa abordagem permite uma recuperação eficiente dos dados associados a um determinado tópico e a identificação rápida dos clientes inscritos em um tópico específico.

Já o cliente é responsável por mandar as requisições ao servidor e de armazenar o valor do id que o servidor encaminha para cada um. O cliente pode

publicar um post no servidor, listar os tópicos, se inscrever e desinscrever em determinados tópicos (para receber notificações de posts feitos) e fechar a conexão.

Para enviar um comando ao servidor, receber as respostas e receber as notificações, o cliente utiliza da função *select*. Essa função monitora múltiplos descritores de arquivo, esperando até que um ou mais deles estejam prontos para leitura, escrita ou que ocorra uma exceção. Isso permite que criemos um fluxo no cliente na qual o mesmo pode receber uma resposta do servidor e enviar um comando sem uma ordem específica.

Abaixo, é descrito o fluxo das mensagens, bem como a estrutura que descreve o formato delas.




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

2.1. Compilação

A compilação do programa ocorre com a execução do comando ***make*** no diretório raiz do projeto. Com isso, tanto o binário do servidor quanto do cliente serão criados na pasta ***bin***. Para executá-los, basta rodar os comandos:

1) bin/server <versao_do_ip> <porta>
2) bin/client <ip> <porta>

3. Discussão

O desenvolvimento de um blog, permitindo a interação entre múltiplos clientes e um servidor através de *sockets* na linguagem C, é um desafio empolgante, mas está longe de ser uma tarefa trivial. Durante o processo de criação desse programa, diversos empecilhos surgiram, aumentando a complexidade de desenvolvimento do mesmo. Neste texto, são apresentados alguns desafios encontrados durante o processo

3.1. Comunicação em Rede com múltiplos clientes:

A comunicação entre o múltiplos clientes e o servidor é a principal funcionalidade do sistema. Para tal, foi utilizado o *multi-threading*, que permite que o servidor responda de forma concorrente aos clientes. Trabalhar com esse tipo de abordagem requer alguns cuidados. O acesso concorrente ao armazenamento de posts e informações do cliente requer uma cuidadosa sincronização para evitar condições de corrida. Além disso, é necessário criar um fluxo que permita que cada cliente receba corretamente a resposta do servidor.

3.2. Tratamento de Erros:

Erros são inevitáveis em qualquer aplicativo de rede. Lidar com erros de forma eficaz e fornecer mensagens de erro claras para os usuários é vital para manter o funcionamento correto do sistema. No trabalho em questão, lidar com o acesso concorrente dos usuários ao servidor e lidar com erros de armazenamento dos posts e dos ids na tabela hash foram tratados, bem como erros de conexão entre o cliente e o servidor, garantindo a confiabilidade do blog.

3.3. Fluxo de entrada e saída no cliente:

O cliente pode receber e enviar mensagens ao servidor sem uma ordem específica. Para que isso seja possível, é necessário um *listener* para checar qual a próxima ordem a ser executada. Isso permite que o servidor possa enviar eventos de postagem em um determinado tópico para os clientes que estejam inscritos nele a qualquer momento. No entanto, tal fluxo flexível de entrada e saída também apresenta desafios, como a necessidade de uma cuidadosa manipulação de eventos assíncronos e a garantia de que o cliente possa gerenciar corretamente as

respostas do servidor e enviar mensagens ao mesmo, sem uma dependência entre as duas ações.

3.4. Armazenamento dos dados no servidor:

Como discutido anteriormente no documento, a tabela hash foi usada para armazenar os dados de posts e clientes inscritos nos tópicos. A implementação dessas estruturas de dados requer atenção à sincronização, especialmente considerando a natureza concorrente do sistema, onde múltiplos clientes podem interagir simultaneamente com o servidor. A garantia de que as operações de leitura e escrita nas tabelas hash sejam atomicamente seguras é essencial para evitar condições de corrida que poderiam comprometer a consistência dos dados armazenados.

4. Conclusão

Em conclusão, a implementação bem-sucedida do sistema de blog em C, com interação entre múltiplos clientes e um servidor, destaca a importância da abordagem cuidadosa em conceitos como multithreading, tratamento de erros e fluxo de entrada/saída assíncrono. A gestão eficiente de sincronização entre clientes e servidor, juntamente com uma robusta manipulação de erros, garante a confiabilidade do sistema mesmo em situações adversas. O resultado é um sistema capaz de oferecer uma experiência interativa e colaborativa aos usuários, mantendo a integridade e confiabilidade das operações realizadas.