UNIVERSIDADE FEDERAL DE MINAS GERAIS
NOME: Eduardo Henrique Dias Melgaço
MATRÍCULA: 2017023501

Redes de Computadores
Trabalho Prático 1 - Documentação

Implementação

Para construir uma arquitetura publish-subscribe foi necessário a criação de um servidor que abre uma conexão via TCP na porta 51511 e uma implementação de cliente, que gera N clientes que podem se conectar a essa porta com a finalidade de troca de mensagens com o servidor. A implementação, criação dos sockets e conexão entre o cliente e servidor utilizou como base o código disponibilizado em aula com um exemplo de servidor multithread.

Observando a implementação do servidor, é possível perceber que ele inicializa a conexão um socket, que escuta a porta que é passada por parâmetro e, após essa etapa de inicialização, ele entra em um loop esperando conexões de clientes através da função *accept*. Ao receber uma conexão de um cliente, ele abre uma thread que será executada à parte, sem interromper a thread principal. Isso faz com que ele prossiga em sua execução e busque novas conexões, onde em cada nova conexão uma nova thread será aberta

Na execução da thread de um novo cliente, criamos um laço de repetição que recebe a mensagem enviada pelo cliente, checa se ela contém apenas caracteres válidos. Caso a mensagem não seja válida o cliente será desconectado instantaneamente, mas sem afetar a execução do servidor e impedir novas conexões. Quando a mensagem é válida, o servidor deve tratar os seguintes tipos de mensagem:

- **Adição de uma nova tag:** A adição de uma nova tag é feita em cima de uma variável global, que é um *map,* onde a chave é um inteiro e o valor é uma string. A chave é composta pelo socket do cliente, e a string do valor armazena todas as tags inseridas, que são envolvidas por identificadores "<" e ">". Exemplo de item na estrutura de dados: 4: "<dota><overwatch>". Os identificadores foram utilizados para demarcar o começo e o fim de uma tag, com a finalidade de evitar erros.
- **Remover uma tag:** O processo de remoção de uma tag é bem similar ao de inserção. Para remover uma tag, a função busca a string "<tag>" dentro da string de tags e caso ela exista, ela é removida da string de listagem de tags.
- **A mensagem é ##kill:** Quando a mensagem for igual à "##kill", o servidor itera sobre a chave de todos os clientes, recupera seus socket e fecha a conexão de um à um. Além disso, foi criado a flag global ***should_disconnect*****,** que é utilizada para encerrar o loop de todas as threads.
- **Envio de uma mensagem:** O envio de uma mensagem é feito quando nenhum dos critérios acima são satisfeitos. O envio de mensagem é feito através da função ***send_message_to_subs****, q*ue processa a mensagem para buscar todas as tags e checa se alguma tag está presente em algum dos clientes. Em caso positivo, a mensagem é enviada para esse cliente.

Analisando o código do servidor é possível perceber que o servidor utiliza uma série de funções complementares que são definidas no módulo "server-utils.cpp". Essas funções são criadas com o intuito de formatar tags, checar validade da tag e retornar as tags de uma mensagem.

A implementação do cliente é bem mais simples, ele abre uma conexão com socket e utiliza duas threads. Uma para receber mensagens do servidor, que é a thread principal, e outra para enviar mensagens ao servidor. A thread principal foi utilizada para o recebimento de mensagens do servidor

pela necessidade de desconectar e encerrar a execução do cliente, quando o servidor manda um sinal de término.

Desafios

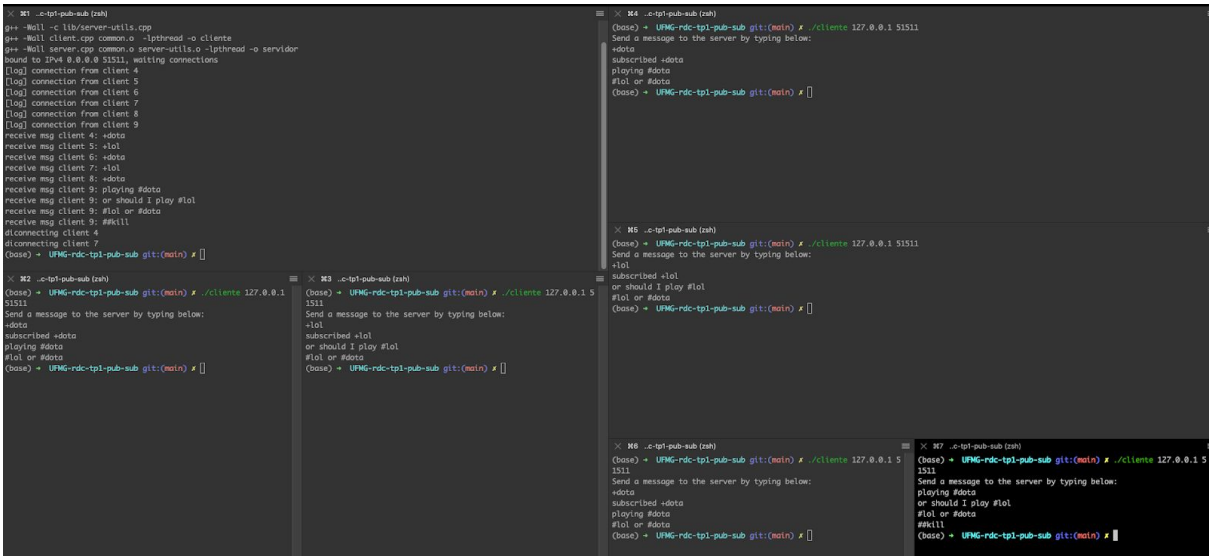
No decorrer da implementação desse trabalho prático me deparei com diversos desafios, que me auxiliaram no aprendizado à medida que eu sempre tive que revisar o material do curso e entender mais a fundo novos conceitos da matéria para solucioná-los.

A parte mais desafiante é entender o fluxo de execução padrão de um código em redes. Isto é, entender qual a forma que devemos usar o recv e o send de forma a não atrapalhar o fluxo de execução do código, além de entender as APIs de socket do POSIX. Outro desafio enfrentado foi o de processar a mensagem enviada por um cliente e reconhecer as tags e para onde elas devem ser enviadas.

Além dos fatores listados acima, outro que dificultou o desenvolvimento foi a minha inexperiência com C e C++. Isso me custou algumas horas resolvendo erros de ponteiros e de alocação de memória e "segmentation fault". Isso foi um dos motivadores para utilizar uma estrutura de dados simples para armazenar as tags de um cliente. Utilizar uma string para armazenar todos os dados, permitiu que eu evitasse erros de memória e utilizasse algoritmos simples para fazer a inserção, busca e remoção de tags.

Resultados

Após a conclusão da da implementação, o resultado obtido atendeu os requisitos solicitados e se mostrou efetivo frente a bateria de testes disponibilizada. A seguir é exibido uma captura de tela do terminal, exibindo o funcionamento do projeto, que pode ser percebido através dos logs de cada terminal:



Ao darmos zoom no exemplo acima, percebemos que todos os clientes se inscrevem na tag #dota ou #lol e que as mensagens são encaminhadas com sucesso e por fim o processo é encerrado com o comando ##kill.