Trabalho Prático 2 - Medium

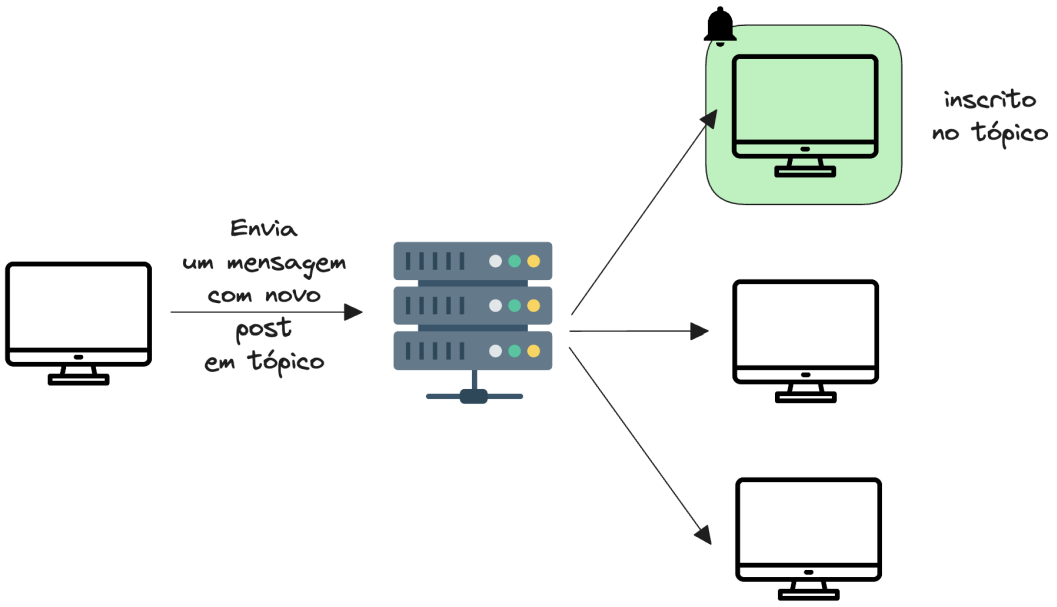
Entrega Individual

Data de Entrega: 08 de novembro de 2023, as 23:59

Introdução

O Medium é uma plataforma de publicação de conteúdo online que permite aos usuários criar e compartilhar artigos, histórias, poemas, código, etc. O conteúdo do Medium é organizado em tópicos, que são categorias de conteúdo que os usuários podem criar e seguir. Os usuários podem seguir tópicos para receber notificações sempre que houver um novo post para um tópico que eles seguem.

Neste trabalho, você irá implementar um blog, como o Medium, que permitirá que vários clientes se conectem ao servidor e publiquem posts. O servidor será responsável por armazenar os posts, os clientes conectados e os tópicos disponíveis para serem seguidos, enquanto os clientes poderão escrever posts, se inscrever em tópicos e receber notificações sempre que houver um post para um tópico específico no qual eles são inscritos.



Na imagem ilustrativa acima um cliente faz um novo post em um tópico, esse post é enviado para o servidor, e o servidor envia a notificação de que um novo post foi adicionado para os clientes que são inscritos naquele tópico.

Objetivo

Você desenvolverá 2 programas para o sistema de blogs utilizando apenas as funcionalidades da biblioteca de sockets POSIX e a comunicação via protocolo TCP. O código do servidor deverá utilizar múltiplas threads para a manutenção das múltiplas conexões. As próximas seções detalham o que cada entidade (servidor e usuário) deve fazer

Os objetivos deste trabalho são:

- Desenvolver um servidor utilizando a interface de sockets na linguagem C;
- Desenvolver um cliente utilizando a interface de sockets na linguagem C;
- Permitir que múltiplos clientes sejam conectados ao servidor;
- Fazer com que o servidor armazene as informações do blog;

Protocolo

O protocolo de comunicação entre o cliente e o servidor será modelado de tal forma que todas as operações sejam representadas pela seguinte estrutura.

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

Abaixo descrição de cada uma das propriedades:

- **client_id**: Essa propriedade armazena o id do cliente que fez a ação no blog. Esse ID deve ser gerado no servidor no momento da conexão e o cliente deve armazená-lo para as futuras interações.
- **operation_type**: Essa propriedade é um enum com todas as ações que serão suportadas pelo programa.
- **server_response:** Essa propriedade especifica se a mensagem transmitida é uma resposta do servidor ou não. Isso é importante porque o operation_type é o mesmo tanto para a ação do usuário quanto para a resposta do servidor. Para ilustrar isso, imagine que o cliente realiza a operação de listar tópicos, e o servidor responde com uma lista de todos os tópicos disponíveis. A resposta do servidor incluirá o campo 'content' preenchido com a lista de tópicos e a propriedade 'server_response' definida como 1; o operation type de ambos é o mesmo (3 para o caso de listagem de tópicos).

- **topic:** Essa propriedade é usada quando uma ação é feita em um tópico específico. Por exemplo, postagem em um tópico, ou inscrição em um tópico.
- **content:** Essa propriedade é usada quando a operação trafega algum conteúdo. Por exemplo, um novo blog post trafega um conteúdo, bem como a resposta de listagem de tópicos enviada pelo servidor

Operation types suportados

Operation type	**ID**
Nova conexão	**1**
Novo post em um tópico	**2**
Listagem de tópicos	**3**
Inscrição em um tópico	**4**
Desconectar do servidor	**5**

Obs.: Não existe uma operação para criar tópicos. O tópico é criado no momento da publicação em um tópico inexistente, ou quando um usuário se inscreve em um tópico inexistente.

Caso de uso

Para melhor exemplificar o uso do protocolo vamos imaginar os seguintes cenários:

Cenário 1

O servidor está ativo, e um novo cliente foi adicionado ao servidor. Nesse caso a estrutura será preenchida da seguinte forma:

```
struct BlogOperation operation;
operation.client_id = 0;
operation.operation_type = 1;
operation.server_response = 0;
strcpy(operation.topic, "");
strcpy(operation.content, "");
```

Repare que o cliente está como zero porque essa é uma mensagem enviada pelo cliente para o servidor no momento de uma nova conexão. Nesse caso o servidor irá definir o id desse cliente e irá retorná-lo. Ao receber essa mensagem o servidor saberá que a operação é de nova conexão e que ele precisa definir um id para esse usuário e responder com a seguinte estrutura.

```
struct BlogOperation operation;
```

```
operation.client_id = 1;
operation.operation_type = 1;
operation.server_response = 1;
strcpy(operation.topic, "");
strcpy(operation.content, "");
```

O cliente irá guardar o próprio id enviado pelo servidor e nas próximas mensagens trafegadas na rede ele irá se identificar com esse id.

Cenário 2
O cliente de ID 2 está conectado e deseja se inscrever em um tópico.

```
struct BlogOperation operation;
operation.client_id = 2;
operation.operation_type = 4;
operation.server_response = 0;
strcpy(operation.topic, "frases_tolkien");
strcpy(operation.content, "");
```

Dado que esse tópico não existe, ele deve ser criado e armazenado do lado do servidor, além disso, o cliente 2 deve ser inscrito a esse tópico.

Cenário 3
O servidor está ativo, o cliente de ID 1 está conectado, e o cliente de ID 2 está inscrito no tópico *frases_tolkien.* O cliente 1 deseja adicionar um novo conteúdo a este tópico:

```
struct BlogOperation operation;
operation.client_id = 1;
operation.operation_type = 2;
operation.server_response = 0;
strcpy(operation.topic, "frases_tolkien");
strcpy(operation.content, "Mas quão poderosa, quão estimulante
para a própria faculdade que a produziu, foi a invenção do
adjetivo: nenhum feitiço ou encantamento em Feéria é mais
potente.");
```

Nesse caso, o servidor irá enviar essa operação para o cliente 2 da seguinte forma.

```
struct BlogOperation operation;
operation.client_id = 1;
operation.operation_type = 2;
operation.server_response = 1;
strcpy(operation.topic, "frases_tolkien");
```

```
strcpy(operation.content, "Mas quão poderosa, quão estimulante para a própria faculdade que a produziu, foi a invenção do adjetivo: nenhum feitiço ou encantamento em Feéria é mais potente.");
```

Repare que temos o cliente 1 como o originador da operação, e essa operação sendo marcada como uma resposta do servidor e sendo enviada para o cliente de ID 2.

Obs: Esses são apenas alguns exemplos de como essas mensagens serão trafegadas, e a ordem em que esses cenários podem acontecer pode variar.

Comandos

Tabela de comandos

Comando feitos pelo cliente	Descrição do comando	Terminal de outro cliente	Servidor
conexão do cliente	A nova conexão não tem nenhum input do cliente, apenas a execução do programa client.	-	Imprime que o cliente foi conectado e o id desse cliente. ***client 01 connected***
> publish in frases **> Eu descobri em mim mesmo desejos os quais nada nesta Terra pode satisfazer.**	O cliente irá digitar publish in frases, dará o enter, e em seguida poderá digitar o texto que será enviado.	O cliente inscrito no tópico irá imprimir a frase com o id do cliente que postou a mensagem O formato da string é: "new post added in <topic> by <client_id>\n" seguido do conteúdo do post. **new post added in frases by 01** **Eu descobri em mim mesmo desejos os quais nada nesta Terra pode satisfazer.**	O servidor irá imprimir que existe um novo post no tópico e irá redirecionar para os clientes inscritos naquele tópico. ***new post added in frases by 01***
subscribe x	O cliente irá digitar subscribe seguido do nome do tópico. Esse tópico será criado no servidor caso não exista.	-	Armazena a inscrição de um tópico para o cliente e imprime que o cliente foi inscrito no tópico. ***client 01 subscribed to x***
unsubscribe x	O cliente irá digitar unsubscribe seguido do nome do tópico.	-	Remove a inscrição de um tópico para o cliente e imprime que o cliente não está mais inscrito no tópico. ***client 01 unsubscribed to x***

list topics	imprime a lista de tópicos separados por ponto e vírgula Exemplo: ***frases; perguntas; piadas*** Caso nenhum tópico exista será impresso "no topics available"	-	Envia todos os tópicos existentes no servidor para o cliente.
exit	O cliente será desconectado do servidor. Os demais clientes conectados continuarão conectados.	-	Imprime que o cliente foi desconectado e remove ele da lista de clientes, e desvincula ele dos tópicos. ***client 01 was disconnected***

Exemplo de execução

Esta seção apresenta alguns exemplos de execuções do sistema. A fim de facilitar a explicação, as tabelas a seguir detalham o passo a passo dos comandos de entrada (em negrito) e as informações que devem ser impressas em tela em cada instante de tempo. A Tabela 1 apresenta um cenário de conexão de dois usuários (Terminal 1 e 2) com o servidor (Terminal 0)

Nesse primeiro exemplo no terminal 1 o cliente recebe o id 1, porém posteriormente ele se desconecta e o id 1 é atribuído ao próximo cliente conectado

Tempo	Servidor	Terminal 1	Terminal 2
T0		**./client 127.0.0.1 51511**	
T1	client 01 connected		
T2		**> list topics**	
T3		no topics available	
T4		**> subscribe in piadas**	
T5	client 01 subscribed to piadas		
T6		**> exit**	
T7	client 01 disconnected		
T8			**./client 127.0.0.1 51511**
T9	client 01 connected		

T10			**> list topics**
T11			piadas
T12			**> subscribe in welcome**
T13	client 01 subscribed to welcome		
T14		**./client 127.0.0.1 51511**	
T15	client 02 connected		
T16		**> subscribe in welcome**	
T17	client 02 subscribed to welcome		
T18			**> publish in welcome**
T19			**> bem vindos**
T20	new post added in welcome by 01		
T21		new post added in welcome by 01 bem vindos	new post added in welcome by 01 bem vindos

Abaixo um exemplo de conexão com 4 clientes em que ocorre a subscription de mais de um cliente aos tópicos.

Tempo	**Servidor**	**Terminal 1**	**Terminal 2**	**Terminal 3**	**Terminal 4**
T0		**./client 127.0.0.1 51511**			
T1	client 01 connected				
T2			**./client 127.0.0.1 51511**		
T3	client 02 connected				
T4		**> subscribe to piadas**			
T5	client 01 subscribed to piadas				
T6				**./client 127.0.0.1 51511**	

T7	client 03 connected				
T8				**> list topics**	
T9				piadas	
T10					**./client 127.0.0.1 51511**
T11	client 04 connected				
T12					**> subscribe to piadas**
T13	client 04 subscribed to piadas				
T14			**> publish in piadas**		
T15			**> Era uma vez um pintinho que se chamava relam, toda vez que chovia relam piava**		
T16	new post added in piadas by 02				
T17		new post added in piadas by 02 Era uma vez um pintinho que se chamava relam, toda vez que chovia relam piava			new post added in piadas by 02 Era uma vez um pintinho que se chamava relam, toda vez que chovia relam piava
T18				**> exit**	
T19	client 03 disconnected				

Exemplo de cliente se subscrevendo mais de uma vez a um tópico

Tempo	**Servidor**	**Terminal 1**
T0		**./client 127.0.0.1 51511**

T1	client 01 connected	
T4		**> subscribe to piadas**
T5	client 01 subscribed to piadas	
T6		**> subscribe to piadas**
T7		error: already subscribed

Exemplos de terminais de execução

Conexão de dois clientes ao servidor

Ordem de execução

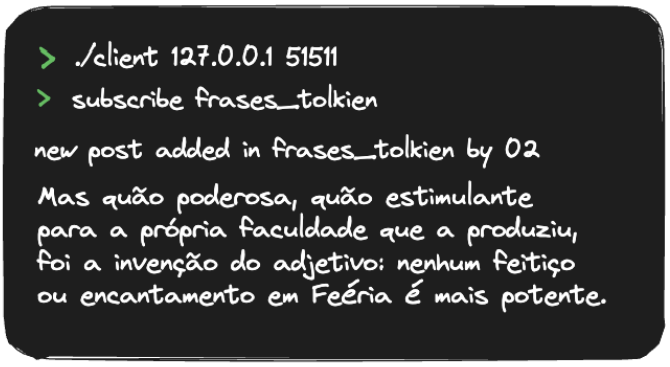
- Servidor inicia a execução
- Cliente 01 é conectado
- Cliente 02 é conectado

Obs: Ao se conectar ao servidor os clientes ainda não possuem um id. Eles passarão a ter id após definição do servidor.

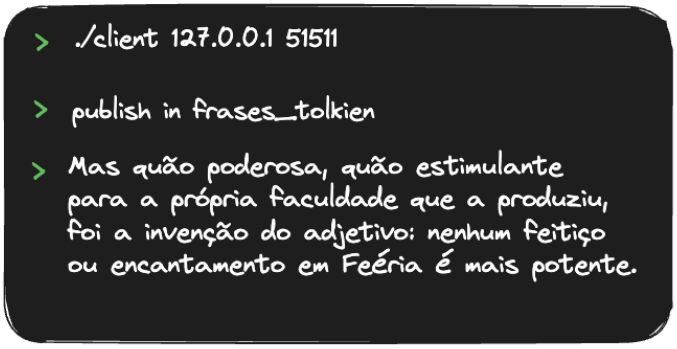
Inscrição do cliente 01 ao tópico frases_tolkien

server

client 01



client 02



Ordem de execução

- Servidor inicia a execução
- Cliente 01 é conectado
- Cliente 02 é conectado
- Cliente 01 subscreve no tópico frases_tolkien, e esse tópico ainda não existe, então ele será criado.
- Cliente 02 publica um conteúdo no tópico frases_tolkien
- Cliente 01 recebe notificação de que conteúdo foi publicado pelo usuário 02.

Obs: O usuário 02 não está inscrito no tópico, mas ele pode publicar nele, porém não recebe notificação de nova publicação nesse tópico e também não é automaticamente inscrito nele.

Detalhes de implementação

- O servidor será responsável por guardar os clientes conectados e o ID deles.
- O cliente será responsável por armazenar seu próprio id depois de recebê-lo do servidor e enviá-lo no campo ***client_id*** sempre que for fazer uma operação
- O servidor será responsável por guardar todos os tópicos e os clientes conectados a esses tópicos. Não será avaliado no trabalho a maneira que esses tópicos serão armazenados, apenas se foram armazenadas

corretamente e se as notificações foram encaminhadas corretamente para os clientes inscritos nos tópicos.

- **Atente-se para o fato de que um novo post acontece da seguinte maneira: primeiro o cliente digita que irá publicar em um tópico e em seguida o programa irá esperar uma entrada do usuário na entrada padrão.**
- As mensagens devem ser encaminhadas apenas para os clientes inscritos nos tópicos e o usuário que enviou a mensagem só irá recebê-la se estiver inscrito no tópico.
- Considere a criação de tópicos sem separação de espaços. (Não será testada a criação de tópicos com espaços)
- O usuário pode se inscrever apenas uma vez nos tópicos. Caso tente se inscrever mais de uma vez um erro será exibido: *error: already subscribed*
- Um cliente **SEMPRE** pode publicar em tópicos inexistentes, ou se inscrever em tópicos inexistentes. Nesses casos eles serão criados.
- A atribuição do id será feita pelo menor inteiro disponível. Ou seja, se tivermos 4 clientes conectados e o cliente 2 e 3 desconectam, então o próximo a conectar receberá o id 2.
- Só serão testados até 10 clientes.
- A mensagem de erro deve ser enviada no content da resposta do servidor.

Desenvolvimento

O projeto requer a implementação de dois componentes essenciais: um servidor e um cliente, ambos baseados no protocolo **TCP**. É crucial garantir que ambos os componentes sejam compatíveis **tanto com o IPv4 quanto com o IPv6**, proporcionando flexibilidade na escolha do endereço IP.

Para configurar o servidor corretamente, este deve ser capaz de receber, seguindo rigorosamente essa **ordem, o tipo de endereço desejado (v4 para IPv4 ou v6 para IPv6) e um número de porta espe**cificado na linha de comando. Recomenda-se a utilização da porta 51511 para manter a padronização do projeto.

Da mesma forma, os clientes precisam receber, também rigorosamente nessa ordem, o endereço IP do servidor e o número de porta para estabelecer a conexão com sucesso. Certifique-se de que ambos os componentes sejam configurados de acordo com essas diretrizes para uma integração eficaz e a comunicação adequada entre servidor e cliente.

A seguir, um exemplo de execução dos programas em dois terminais distintos:

IPv4:

```
no terminal 1: ./server v4 51511
no terminal 2: ./client 127.0.0.1 51511
no terminal 3: ./client 127.0.0.1 51511
```

IPv6:

```
no terminal 1: ./server v6 51511
no terminal 2: ./client ::1 51511
no terminal 3: ./client ::1 51511
```

Dicas e cuidados:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle.
- Playlist de programação com sockets do professor Ítalo Cunha
- O guia de programação em rede do Beej (http://beej.us/guide/bgnet/) tem bons exemplos de como organizar um servidor
- Implemente o trabalho por partes. Por exemplo, implemente o tratamento das múltiplas conexões, depois crie os formatos das mensagens e, por fim, trate as mensagens no servidor ou cliente.
- Procure resolver os desafios da maneira mais simples possível.

Avaliação

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C (não sendo permitida a utilização de C++)** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo **(você pode testar com as implementações dos seus colegas)**. Procure escrever seu código de maneira clara, com comentários pontuais e bem identados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática
Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor.

Para a correção os seguintes testes serão realizados **(com IPv4 e IPv6)**:

- Conexão de múltiplos clientes **10%**
- Atribuição correta de ids **10%**
- Fechamento de conexão **10%**
- Encaminhamento correto das publicações para os clientes inscritos nos tópicos **20%**
- Impressão correta das mensagens **10%**

- Unsubscribe nos tópicos **10%**
- Impressão correta das mensagens no servidor **10%**
- Documentação **20%**

Informações importantes

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte Arial tamanho 12, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.

Atenção: Será dado grande importância a parte da *discussão dos desafios encontrados*, pois as dificuldades ao longo do projeto sempre existem.

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.

Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado “**client**” e o servidor em um binário chamado “**server**” dentro de uma pasta chamada "**bin"** na raiz do projeto.
- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP2_MATRICULA.zip

Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega.
Obs: Não serão aceitos trabalhos com atraso.