Universidade Federal de Minas Gerais

Departamento de Ciência da Computação

Curso de Redes 2020/2º

Aluno: Marcos Vinicius Moreira Santos

Relatório: Trabalho Prático - Aplicação Publish/Subscribe

Introdução

O código foi dividido em 2 modulos, uma biblioteca de código que está em localizada em src/netowrk que abstrai um servidor e cliente TCP. Essa abstração foi usada para a implementação da aplicação Publish/Subscribe permitindo que as regras de negócio da aplicação fiquem mais limpas e de facil entendimento.

As regras de negócio da aplicação Publish/Subscribe está definida nos arquivos *.c e *.h localizadas dentro do diretório 'src/'.

Compatibilidade:

A aplicação foi testada somente em sistemas **UNIX**, e portanto a compatibilidade com outros sistemas não é garantida.

Network:

Introdução:

Os arquivos localizados em src/network são os arquivos que abstraem a implementação de uma estrutura de servidor e cliente para comunicação sobre a network utilizando o protocolo TCP/IP. Além disso, também foi implementado estruturas de dados para programação assíncrona como threads e mutexes.

A biblioteca foi construida utilizando UNIX sockets e POSIX threads além da biblioteca padrão do C99.

Servidor TCP:

Em src/network/tcp_server.c foi definida uma estrutura de dados e metodos que abstraem um servidor TCP. A abstração em questão utiliza a thread principal da aplicação para esperar por conexões, e quando uma solicitação de conexão é feita por um cliente, essa mesma thread é responsável por criar uma nova thread que irá ser responsável por manter a conexão com o cliente ativa, qualquer mensagem enviada pelo cliente portanto será recebida e tratada dentro dessa thread.

Em caso de aplicações em que a ordem que o servidor irá processar as requisições de diferentes clientes importa, é possivel instanciar o servidor tcp com a flag **TCP_SERVER_SYNC**, essa flag fará com o que servidor execute a primeira requisição que chegar antes que a thread de um outro cliente com requisição pendente comece sua execução.

O comportamento do servidor é definido utilizando-se o método:

```
void tcp_server_t_set_request_handler(struct tcp_server_t* server,
tcp_server_t_request_handler handler);
```

Este método irá definir como o servidor deverá processar uma mensagem qualquer recebida de algum cliente.

Estes metodos precisam seguir a interface:

```
void(*)(struct request_t, struct reply_t);
```

As estruturas request_t e reply_t são estruturas que armazenam informações uteis como o cliente que fez a requisição, o servidor que está processando a requisição, a mensagem de payload e o tamanho da mensagem enviada pelo cliente.

Cliente TCP:

Foi implementado em src/network/tcp_client.c uma estrutura que abstrai a funcionalidade de um client TCP.

No cliente não foi necessário nenhuma thread extra, foi escolhido a implementação de um cliente que utiliza *'non blocking sockets'*. Essa escolha foi tomada para permitir que o programador utilizando o cliente possa fazer o uso dos recursos computacionais da forma que ele achar melhor baseado na aplicação sendo desenvolvida.

Detalhes de implementação:

Para a implementação das estruturas do servidor, foram utilizadas estruturas de dados assincronas, como threads e mutexes, ambas essas estruturas foram abstraidas utilizando a biblioteca pthreads em src/network/async.c.

Foi feito o uso de UNIX sockets para a implementação do servidor e cliente assim como outras classes da biblioteca padrão como strings, stdlib e stdio. No caso do cliente foi utilizado um *'non blocking socket'* atualizando suas flags para incluir a flag ***O_NONBLOCK*** após sua inicialização.

O uso de threads foi necessário para garantir que um servidor possa servir a mais de um cliente por vez, portanto para cada cliente o servidor cria uma thread unica que será responsável por escutar as mensagens enviadas por esse cliente.

Para armazenar as conexoes ativas com o servidor, foi utilizado uma estrutura de lista encadeada que pode ser utilizada simultaneamente por multiplas threads, sincronizada através do uso de mutexes, onde cada nó armazena as informações references à uma conexão/cliente.

Aplicação Publish/Subscribe:

Toda a lógica relacionada à aplicação de Publish/Subscribe está definida sobre os arquivos *.c e *.h dentro do diretório src/.

A aplicação solicitada define quatro ações de usuário:

1. Cadastrar o cliente em um canal enviando uma mengagem iniciada com o caractere '+'.
2. Descadastar o cliente de um canal enviando uma mengagem iniciada com o caractere '-'.
3. Encerrar a execução do servidor e todas as suas conexoes enviando uma mensagem "##kill".
4. Enviar uma mensagem de texto para o servidor que irá redirecionar a mesma mensagem para qualquer cliente que esteja interessado. Essa ação pode ser realizada enviando uma mensagem que não se encaixa em nenhuma das condições das ações anteriores.

O Servidor:

O servidor é responsável por tratar as requisições feitas pelos clientes. Foi definido um handler, adicionado ao servidor através do método:

```
void tcp_server_t_set_request_handler(struct tcp_server_t* server,
tcp_server_t_request_handler handler);
```

Por sua vez, esse handler verifica a mensagem enviada pelo cliente e, baseado nas quatro condições descritas acima, decide alguma ação que será executado sobre a mensagem.

O servidor utiliza apenas duas estruturas de dados auxiliares, um mapa do tipo 'string'->'lista de inteiros' que irá armazenar em qual canal('string') um determinado grupo de clientes('lista de inteiros') estão interessados, chamaremos essa estrutura de tabela de canais, e uma estrutura de tabela hash que armazenará inteiros e que será utilizada para armazenar quais os clientes que já receberam uma dada mensagem para evitar que clientes cadastrados em dois canais diferentes recebam uma mensagem duplicada em caso de essa mesma mensagem ter sido enviada para ambos os canais.

O Cliente:

O cliente é responsável por 2 coisas:

1. Receber a entrada do usuário
2. Enviar a entrada do usuário para o servidor.
3. Receber as mensagens do servidor

Para receber a entrada do usuário, foi implementada em src/terminal.h algumas funções uteis no manejo de entradas de usuário pelo terminal, como funções não bloqueantes que verificam se existem alguma entrada pendente. A partir disso, a lógica do cliente passa a ser bastante simples, basta testar se existe entrada pendente do usuário, caso exista, a mensagem é enviada ao servidor.

Em caso de nenhuma mensagem pendente, o cliente tenta receber mensagens do servidor utilizando o método:

```
int tcp_client_t_receive(struct tcp_client_t* client, char* message, int
length);
```

Este método retorna -1 quando não existem mensagems pendentes e 1 quando existe pelo menos uma mensagem pendente do servidor. No segundo caso, é salva no 2º parametro chamado *message* uma mensagem com tamanho igual ao indicado pelo 3º parametro *length*.

Conclusão:

Durante a implementação desse Trabalho foi implementado um Servidor e Cliente que utilizam o protocolo TCP/IP, o que possibilitou entender com mais detalhes o funcionamento deste Protocolo.

Também foi implementado uma aplicação por cima do servidor e cliente TCP abstraidos dentro de src/network que se resume a uma aplicação Publish/Subscribe. Durante a implementação desta aplicação foi possível o melhor entendimento das aplicações que utilizam comunicação sobre a netowrk, e em especial, foi possivel obter "insights" sobre o funcionamento de sistemas como Apache Kafka e RabbitMQ.