Trabalho Prático 1
Servidor de Mensagens Publish/Subscribe

Jean George Alves Evangelista - 2018047021

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanalves@dcc.ufmg.br

1. Introdução

O objetivo deste trabalho foi realizar a implementação de um servidor e clientes para trocas de mensagens utilizando o paradigma de comunicação *publish/subscribe* na linguagem de programação C/C++. O servidor recebe mensagens dos clientes e repassa as mensagens para os clientes interessados naquela mensagem (inscritos em determinadas *tags*). O cliente informa ao servidor em quais tags possui interesse e a partir disso recebe as mensagens que contém suas *tags* de interesse;

2. Implementação

A implementação foi feita utilizando como base o código disponibilizado no Moodle, referente às aulas do tópico “Introdução à Programação em Redes”, ministradas pelo professor Ítalo. Portanto, o código final segue boa parte da estrutura criada nas aulas: um programa para o cliente e um programa para o servidor, bem como algumas funções comuns utilizadas por ambos os programas. Além disso, foi adicionado um arquivo para manipulação do terminal. A seguir segue uma breve explicação das principais partes da implementação e das alterações realizadas no código original do professor.

2.1 Cliente

Apesar do programa do cliente ter sido feito como base o modelo disponibilizado algumas alterações nele foram necessárias no decorrer da implementação:

- Mudança de .c para .cpp: foi necessário porque o código utilizado para manipulação do terminal e atendimento do requisito de *input* e output simultâneos utilizava recursos de C++.
- Alteração do escopo do *socket* utilizado no cliente para global: foi necessário porque o socket passou a ser utilizado em funções a parte no arquivo e não apenas na função *main*. Embora possa não ser uma das melhores alternativas, se mostrou bastante simples e evitou complicações de passagens por parâmetro.
- Mudança na obtenção dos dados via terminal: Anteriormente os dados eram recebidos via *fgets* e agora são recebidos se utilizando dos recursos da classe *Console* importada.
- Criação de um método para recebimento de mensagens: foi necessário para atender o requisito do envio/recebimento de mensagens de maneira concorrente. Esse método foi atribuído a uma *thread* e é executado em paralelo a leitura dos dados via console. Também foram feitas algumas alterações no loop de recebimento para receber até uma quebra de linha, bem como foram adicionadas validações em cima da mensagem recebida.
- Mudanças no envio de mensagens: Além de tratar a string lida de maneira diferente foi feita uma alteração em um dos parâmetros do método *serend* da biblioteca *socket.h*, para não enviar o caractere delimitador de fim de *string* ‘\0’.

2.2 Servidor

Muitas alterações foram realizadas no arquivo *server-mt.c* original. Dentre as principais estão:

- Alteração para sempre trabalhar com IPV4.
- Mudança na estrutura *client_data*: foi necessário adicionar atributos para armazenamento das *tags* dos cliente e uma *flag* indicativa para saber se a conexão desse cliente está aberta ou não.
- Criação de um array de *client_data*, de escopo global: Todo cliente que se conecta é adicionado nesse array e o programa gerencia o envio/recebimento de mensagens e inscrição/desinscrição de tags por meio dessa variável.
- Mudanças na *main*: Foram poucas, se resumem-se a instanciação da variável global de controle de clientes, inserção no *array* a cada cliente conectado e mudança no atributo que é passado para função executada na *thread* de clientes.

A execução do código do servidor se dá basicamente do método *client_thread*, executado pela thread definida na *main*. O método executa dois loops, um mais externo e um interno. O interno é executado até atingir uma das seguintes condições:

- Receber uma mensagem inválida, nesse caso o cliente é desconectado e o loop é interrompido
- Receber a mensagem "##kill", nesse caso todos os clientes são desconectados e o processo é encerrado
- Receber o caractere '\n', nesse caso o loop se encerra.

O loop externo executará enquanto receber dados do cliente. Nesse loop é feita uma validação se o primeiro caractere da mensagem é '+' ou '-', que indicam inscrição e desinscrição. Caso seja, é chamado um método que valida, trata, inscreve/desinscreve a tag para o cliente e envia a mensagem correta no fim. Caso não seja, um método de tratamento de mensagens é chamado, que verifica se a mensagem possui o caractere '#' (indicativo de uma *tag*), caso possuir a mensagem é enviada para todos os clientes que possuem aquela *tag*. O controle de *tags* e de *sockets* dos clientes é realizado utilizando o *array* global citado anteriormente.

2.3 Common

No projeto existe um arquivo *common.c*, que reúne métodos úteis para o cliente e para o servidor. Nada do código original foi alterado, somente foram incluídos dois métodos para validação de caracteres e mensagem.

2.4 Console

Foi incluída uma estrutura pronta para manipulação da entrada de dados de maneira concorrente ao recebimento de mensagens do servidor. Por não ser o foco do trabalho julga-se que não é necessário entrar em detalhes desse trecho. O código original está referenciado na seção de referência.

3. Desafios, dificuldades e imprevistos do projeto

A implementação do trabalho como um todo foi um grande desafio, uma vez que a maneira como a comunicação em rede é realizada numa linguagem de mais baixo nível como C é bastante complicada de ser entendida a primeira vista.

Dentre as dificuldades enfrentadas na implementação, as principais foram (1) conciliar a entrada e saída de dados; e (2) trabalhar com *threads*, algo que não foi visto na disciplina e requer um conhecimento prévio. Além disso, cabe citar que foi necessário realizar diversas alterações no código original para atender os requisitos e muitas dessas alterações não foram triviais e demandaram bastante tempo para entendimento. Por último, inclui-se a dificuldade em testar todos os cenários possíveis descritos na especificação, sendo necessário manipular o teste disponibilizado para realmente confirmar se os programas funcionam corretamente.

5. Conclusão

Os programas construídos foram testados em diversos casos de teste criados com base na especificação e de acordo com os resultados obtidos é possível concluir que a implementação do que foi especificado foi realizada com êxito. O trabalho como um todo permitiu ter uma visão melhor de como é realizada programação em rede e os conhecimentos adquiridos certamente poderão ser utilizados no futuro em outras bibliotecas/linguagens.

Referências

Simultaneous input and output in a console. StackOverflow. Disponível em: <https://stackoverflow.com/questions/55414228/simultaneous-input-and-output-in-a-console>. Acesso em: 14 jan. 2021.

C++ Language. Disponível em: <http://www.cplusplus.com/doc/tutorial/>. Acesso em: 03 jul. 2019.

Aulas da disciplina “Redes de Computadores”. UFMG Virtual. Disponível em: <virtual.ufmg.br>. Acesso em: 14 jan. 2021.