Universidade Federal de Minas Gerais
Escola de Engenharia
DCC023 - Redes de Computadores
Trabalho prático 03: servidor de mensagens
Júlia de Backer Pacífico - 2019021476
Agosto de 2021

I. Introdução

Inspirado na popularização dos serviços de comunicação instantâneos, o propósito do terceiro trabalho prático proposto na disciplina de Redes de Computadores foi da criação de um sistema de compartilhamento de mensagens multimídias, utilizando apenas as funcionalidades equivalentes às da biblioteca de *sockets* POSIX. Assim, foram implementados 3 programas segundo o modelo servidor cliente: dois clientes, um do tipo exibidor e outro do tipo emissor, e um servidor responsável pela transmissão das mensagens trocadas na rede.

II. Implementação

O trabalho foi implementado por meio da ferramenta WSL -Windows Subsystem for Linux, em linguagem C++, devido à sua eficiência para projetos em baixo nível. Afora o servidor, um programa exibidor, cujo papel é de apenas exibir as mensagens a ele entregues, deve ser o primeiro executado, seguido de um cliente emissor que será a ele associado.

II.i) Arquitetura

Foi criada uma estrutura comum *header*, que contém os elementos de um cabeçalho de 8 bytes utilizado para a comunicação na rede — tipo de mensagem, identificador de origem, identificador de destino e o número de sequência da mensagem; cada um deles do tipo *unsigned short*, de tamanho de 2 bytes. O identificador do servidor assume valor $2^{16} - 1$, enquanto os identificadores dos emissores podem assumir valores de 1 a $2^{12} - 1$ e, o dos exibidores, valores de 2^{12} a $2^{13} - 1$. O identificador de cada cliente é atribuído pelo servidor em resposta a uma mensagem do tipo "OI", e os valores para cada tipo de mensagem, cujos papéis são predeterminados na proposta deste trabalho, são informados na Tabela 1.

II.ii) Servidor

Com base nos exemplos implementados por BEEJ[2] , o servidor criado utiliza da função *select()* para abrir múltiplos sockets e tem uma lista de espera para até 10 clientes. Assim que conectado a um deles, o servidor aguarda por uma mensagem de cabeçalho. Se receber uma mensagem do tipo "OI", o servidor busca pelo campo de identificação do cliente remetente para interpretá-lo como um emissor ou exibidor: clientes do tipo exibidor enviam o cabeçalho da mensagem com o identificador (*id*) nulo, enquanto clientes emissores enviam um valor maior que zero. Caso o identificador da mensagem "OI" não corresponda a nenhuma das interpretações possíveis, o servidor identifica um programa indesejado e interrompe a comunicação após enviar uma mensagem do tipo "ERRO". Do contrário, o servidor atribui um identificador ao cliente conectado conforme as especificações fornecidas.

Tipo de mensagem	Valor de campo
OK	1
ERRO	2
OI	3
FLW	4
MSG	5
CREQ	6
CLIST	7
FILE	8
FILE_CHUNK	9

Tabela 1 — tipos de mensagem.

Para a base de dados do servidor, foi criada uma estrutura *client* que armazena dois parâmetros: o número do socket do cliente conectado e seu identificador. Assim, a cada nova conexão, o servidor cria um novo *client* e preenche seus devidos campos após atribuir-lhe um número de identificação disponível por meio de uma função criada para essa finalidade. Com isso, o *client* recém-criado é concatenado em um *vector*. Portanto, existe no servidor um *vector* para clientes emissores e outro para clientes exibidores. Após o tratamento da mensagem "OI", o servidor envia uma mensagem de confirmação do tipo "OK" informando o identificador atribuído ao cliente comunicante e aguarda novas mensagens.

Os tipos de mensagem são identificados por meio de um comando *switch-case* que tem como parâmetro um campo do *header* recebido, e este modelo é padrão para identificar o tipo de mensagem recebida em cada um dos três programas implementados. Ao identificar uma mensagem do tipo "FLW", o servidor busca pelo identificador do destinatário no *vector* de exibidores para repassar a mensagem antes de liberar seu *id* na base de dados ao fechar a conexão. Em seguida, o servidor remove o identificador do remetente do *vector* de emissores. Já para mensagens do tipo "MSG", é esperado, após o cabeçalho, que o cliente emissor envie um inteiro de 2 bytes informando o tamanho do buffer que conterá a mensagem de texto antes do buffer em sí. Essas duas informações são logo repassadas ao programa exibidor, e é esperada uma resposta em cabeçalho do tipo "OK" ou "ERRO", que é posteriormente encaminhada do servidor ao emissor.

Mensagens do tipo "CREQ" são encaminhadas ao exibidor como mensagens do tipo "CLIST". Ao receber uma requisição deste tipo, o programa servidor cria uma variável N de 2 bytes que armazena a soma do tamanho dos dois *vectors* que contém o dado dos clientes; portanto, N contém o total de clientes conectados. Posteriormente, é criado um simples array de N elementos que armazena o identificador de todos os clientes conectados ao servidor. Dessa forma, o servidor suporta o *boradcast* de mensagens CLIST se o campo do destinatário especificado pelo emissor for nulo; do contrário, elas são enviadas apenas ao exibidor associado. Em resposta, o servidor espera uma mensagem "OK", que será repassada ao cliente.

II.iii) Exibidor

O programa exibidor deve ser executado antes da inicialização do cliente emissor. Assim, ele recebe como parâmetro de execução obrigatório o par de endereço no formato "IP:port". Assim que conectado ao servidor, o exibidor envia uma mensagem do tipo "OI", com o campo de identificação em valor nulo com número de sequência também zero. Em seguida, aguarda uma mensagem de confirmação do tipo "OK" que contenha o número de identificação que o exibidor utilizará na rede. Caso seja recebida uma mensagem de erro, o socket da comunicação é fechado e o programa tem execução terminada; do contrário, o programa exibidor aguarda pelo cabeçalho que precede novas mensagens em um loop intermitente até que seja recebida uma mensagem do tipo "FLW", que encerra a conexão do socket. Ademais, mensagens do tipo "MSG" e "CLIST" tem seu recebimento confirmado por uma mensagem "OK" enviada ao remetente cujo identificador é mostrado no cabeçalho.

II.iv) Emissor

Além do par de endereço "IP:port" necessários à execução do do emissor, este programa cliente também deve receber como parâmetro o identificador de um emissor que será a ele associado. Assim, ao enviar uma mensagem tipo "OI" ao servidor, o emissor espera a validação do *id* fornecido. Em caso negativo, o programa recebe uma mensagem tipo "ERRO", fechando o socket e encerrando sua execução. Ademais, caso a mensagem seja aceita pelo servidor, o programa emissor incrementa o número de sequência da mensagem e inicia um laço infinito para envio de mensagens. A cada laço, o programa emissor incrementa o número de sequência da mensagem, sendo ele o único na comunicação passivo de alterar este valor.

O programa emissor, apesar de não exibir as mensagens recebidas, conta com uma simples interface de interação com o usuário, com a qual pode-se selecionar o tipo de mensagem enviada e o destinatário apropriado (se apenas o exibidor associado ou se para todos os demais exibidores na rede).

Com o recebimento de mensagens de confirmação, o cliente emissor imprime um informativo quando sua mensagem for devidamente entregue.

III. Discussão

O projeto pode ser compilado por meio de um makefile criado adicionalmente, sendo imprescindível o header file *common.h* criado para modularização de algumas funções de manipulação do cabeçalho de mensagem e de funcionamento do *socket parse* dos clientes implementados.

Dentre os principais desafios encontrados, o maior deles foi sincronizar os clientes e o servidor durante a comunicação, o que, ao fim da prática, infelizmente foi um desafio não completamente solucionado. Por esse motivo, o programa consta algumas falhas de comunicação após o envio de pelo menos duas mensagens. Além disso, o programa servidor é passível de muitas melhorias, dentre elas, uma modularização mais eficiente. O programa servidor também não sucede em tratar mensagens de envio de arquivo; por isso, mensagens do tipo "FILE_CHUNK" retornam uma mensagem de erro ao cliente exibidor.

IV. Conclusão

Concluiu-se, findada a execução deste trabalho, que uma das questões mais pertinentes na programação em redes é a compatibilidade entre os tipos enviados e recebidos na redes, bem como a trivialidade da sincronia entre o envio e o recebimento de mensagens pelos comunicantes.Para além da prática com a linguagem C++, a implementação deste trabalho também foi proveitosa para compreender de forma prática os conceitos trabalhados ao longo das aulas. Apesar das dificuldades supracitadas, o servidor implementado apresentou alguns resultados satisfatórios..

V. Fontes

- **[1] "Sockets In C", Michael J. Donahoo, Kenneth L. Calvert;**
- **[2] "Beej's Guide to Network Programming", Brian "Beej Jorgensen" Hall;**
- **[3] "Linguagem C: Completa e Descomplicada", André Backs.**