Trabalho prático 2 - Redes de Computadores
Sistema de múltiplas conexões

Nome: Alexis Duarte Guimarães Mariz
Matrícula: 2019006337

O objetivo do sistema é permitir a comunicação em tempo real entre múltiplos usuários na rede local, usando sockets. O sistema foi implementado em C e suporta IPv4 e IPv6.
Para compilar o programa, execute o comando "make", sem nenhum parâmetro adicional.

Após compilado, para executar o programa, seguindo a especificação do trabalho, primeiro deve-se digitar no terminal:
./**server v4 5151** ou **./server v6 5151**, onde 5151 é uma porta escolhida para rodar o servidor.
Em seguida, deve-se digitar em outra aba do terminal:
./user 127.0.0.1 5151 ou **./user ::1 5151**, de acordo com a versão IPv4 ou IPv6 escolhida para rodar o Servidor. Esse passo pode ser repetido várias vezes e o servidor aceita até 15 conexões simultâneas.

user.c
O arquivo user.c implementa a lógica do cliente. Ele inicia a conexão com o servidor, envia mensagens para o servidor e recebe respostas. Ao iniciar sua execução, o cliente envia um REQ_ADD ao servidor. O cliente utiliza duas threads, uma para enviar mensagens ao servidor e outra para receber as mensagens. Além disso, o arquivo user.c implementa várias funções para lidar com a comunicação com o servidor:

- parse_opening_server_response(): Analisa a resposta inicial do servidor após a conexão ter sido estabelecida.
- command_parse(): Analisa a mensagem recebida do servidor e realiza a ação apropriada com base nessa mensagem.
- listen_socket(): Executa em uma thread separada e escuta as mensagens recebidas do servidor.
- parse_opening_server_response(): Analisa a resposta inicial do servidor após a conexão ter sido estabelecida.
- parse_command_to_send(): Lê os comandos do usuário do teclado, analisa os comandos e os envia ao servidor.
- main(): A função principal do programa. Ela cria um socket, se conecta ao servidor, envia o comando REQ_ADD para o servidor e inicia as threads de envio e recebimento.

server.c
O arquivo server.c implementa o servidor para o sistema de chat. A função main no arquivo inicializa o servidor e espera por conexões de clientes. Quando uma conexão de cliente é aceita, o servidor cria uma nova thread para lidar com a comunicação com esse cliente, permitindo que o servidor atenda a vários clientes simultaneamente.
O servidor implementa um conjunto de funções para lidar com os diferentes comandos e mensagens que podem ser recebidos dos clientes:

- error(): Envia uma mensagem de erro para o cliente, com seu código de erro.
- msg(): Encaminha a mensagem recebida para o destinatário especificado ou para todos os usuários se o destinatário for -1.
- res_list(): Envia a lista de usuários conectados ao cliente.
- ok(): Envia uma
- req_rem(): Processa uma solicitação de um cliente para ser removido da rede.

- req_add(): Processa uma solicitação de um novo cliente para se juntar à rede.
- command_parse(): Analisa o comando recebido do cliente e chama a função apropriada para processar o comando. Nota: O servidor nunca recebe as mensagens com IdMsg 7 e 8.
- client_thread(): Função que é executada em uma nova thread para cada cliente conectado. Ela recebe e processa comandos do cliente chamando command_parse().

common.c
O arquivo common.c implementa as funções:

- logexit()
- addrparse()
- addrtostr()
- server_sockaddr_init()
- select_file()
- send_file()
- command_parse()
- message_parse()

A função logexit encerra o programa com uma mensagem de erro personalizada caso ocorra algum erro. As funções addrparse, addrtostr e server_sockaddr_init implementam a lógica dos sockets. Essa parte não tive dificuldades, pois segui o tutorial no YouTube de Introdução à Programação em Redes do professor Ítalo Cunha(https://www.youtube.com/playlist?list=PLyrH0CFXIM5Wzmbv-lC-qvoBejsa803Qk), disponibilizado no Moodle.
Além disso, a funçãosendCommand() implementa a lógica de envio de comandos e mensagens tanto para o servidor, quanto para o cliente.

Formato de Transmissão de Dados na Rede

Os dados são transmitidos na rede no seguinte formato:

length(4bytes)	IdMsg(4bytes)	IdSender(4bytes)	IdReceiver(4bytes)	Message(2032bytes)

Os primeiros 4 bytes definem o número de bytes transmitidos na rede, os próximos 4 bytes são o IdMsg, que é o identificador da mensagem, IdSender(4bytes), que é o identificador do remetente, IdReceiver, que é o identificador do destinatário e Message é a mensagem real que está sendo transmitida.

- No primeiro trabalho foi utilizado uma string “\end” para definir o fim da mensagem. No entanto, para esse trabalho optei por definir que os primeiros 4 bytes serão usados para definir o número de bytes transmitidos na rede.
- Como não existe um OK para confirmação de mensagem enviada, o servidor envia a mensagem também para o remetente, para que o cliente que enviou possa imprimir a sua própria mensagem enviada apenas caso ela seja enviada com sucesso. Isso evita que o remetente imprima sua mensagem caso ele tente enviar uma mensagem para um usuário inexistente, por exemplo.
Uma grande dificuldade que tive foi a passagem do vetor de conexões ativas como ponteiro nos parâmetros das funções. Isso levou algumas horas para debugar, além de outros problemas, como a necessidade de duas threads no programa do usuário, que eu não havia previsto no início do projeto.