Trabalho Prático 2

Redes de Computadores

Henrique Rotsen Santos Ferreira

2020100945

Documentação do Código:

Inclusões de Bibliotecas:

O código inclui diversas bibliotecas padrão do C, bem como um arquivo de cabeçalho personalizado chamado common.h. Este arquivo contém definições e declarações necessárias para o programa.

Definições de Constantes:

O arquivo common.h define constantes que representam o tipo de operação que pode ser realizado pelo cliente.

Struct BlogOperation:

A struct BlogOperation armazena informações sobre a mensagem enviada pelo cliente. Ela contém o id do cliente, tipo da operação, se é uma resposta do servidor, tópico da mensagem e o conteúdo da mesma.

Funções Auxiliares (declaradas em common.h):

*logexit(const char *msg)*: Imprime uma mensagem de erro e encerra o programa.

*addrparse(const char *addrstr, const char *portstr, struct sockaddr_storage *storage)*: Converte uma representação de endereço e porta para uma estrutura sockaddr_storage.

*addrtostr(const struct sockaddr *addr, char *str, size_t strsize)*: Converte um endereço de sockaddr para uma string legível.

*server_sockaddr_init(const char *proto, const char *portstr, struct sockaddr_storage *storage)*: Inicializa a estrutura sockaddr_storage com as informações de endereço do servidor.

Função main:

A função principal do programa. Ela lida com a configuração inicial, inicializações de sockets, no caso do servidor, dispara um thread para cada cliente. Assim temos um servidor que consegue comunicar 2 ou mais clientes. Já no caso do cliente ela envia as mensagens lidas do teclado.

Client thread:

Cada cliente tem seu próprio fluxo de execução, porém temos um thread que roda em paralelo com a main, apenas para receber as mensagens. Dessa maneira, toda resposta do servidor é mostrada imediatamente.

Inicialização e Configuração do Servidor:

O servidor é inicializado para aceitar conexões tanto via IPv4 quanto IPv6.

Loop de Aceitação de Clientes:

O servidor entra em um loop infinito para aceitar conexões de clientes.

Processamento de Mensagens dos Clientes:

O servidor recebe as mensagens e analisa o tipo de operação e quais ações devem ser tomadas para fazer a computação corretamente.

Fechamento de Conexões:

As conexões são fechadas apenas após o cliente dar EXIT. Nesse caso fecha-se o socket do cliente no “server.c” e o do servidor, no “client.c”.

Fechamento de Arquivos e Sockets:

Arquivos e sockets são devidamente fechados ao final do programa.

Desafios:

De modo geral, o TP pareceu bem explicado, com apenas certas ambiguidades que vi na documentação, mas nada que dificultasse a realização dele. O principal que era fazer a parte multithreading no cliente e no servidor e resolver questões relacionadas à sincronização.

Problemas e Dificuldades:

Durante o trabalho, tive alguns problemas no que tange às mensagens, percebi ao realizar testes com outros colegas que podia acontecer algumas dificuldades de sincronização, logo fiz uma mutex para as operações globais. Além disso percebi uma falta de constância na hora do cliente printar as mensagens, não consegui descobrir o que era exatamente, mas me parece que printar uma mensagem vazia (“”), sem um “\n”, pode levar a uma latência no próximo print.

Em geral, fiz bastantes testes e percebi que ao colocar um “\n” nas mensagens vazias ele parava de dar o problema, porém como não sei se os testes serão automáticos eu optei por deixar sem o “\n”.

Testes:

Fiz inúmeros testes no meu servidor e no cliente, a fim de testar todas as ocasiões descritas no problema. Os testes testaram tanto as conexões do meu TP em IPv4 e IPv6, quanto estas conexões em clientes e servidores de outros colegas.