Client.c

int main() {

główna funkcja programu

int sockfd;

zmienna przechowująca deskryptor socketu, czyli identyfikator połączenia

struct sockaddr_in addr;

struktura przechowująca adres serwera (IP oraz port)

char buffer[100];

Bufor znaków do wysyłania i odbierania danych

sockfd = socket(AF_INET, SOCK_STREAM, 0);

Tworzenie socketu:

AF_INET — protokół IPv4

SOCK_STREAM — połączenie TCP

0 — domyślny protokół

addr.sin_family = AF_INET;

Ustawienie typu adresu na IPv4

addr.sin_port = htons(5050);

Ustawienie portu serwera na 5050, htons konwertuje port do formatu sieciowego

addr.sin_addr.s_addr = inet_addr("127.0.0.1");

Ustawienie adresu IP serwera.

127.0.0.1 oznacza localhost (ten sam komputer)

connect(sockfd, (struct sockaddr*)&addr, sizeof(addr));

Nawiązanie połączenia z serwerem o podanym adresie i porcie

while(1) {

pętla nieskończona (klient działa do momentu zakończenia programu)

scanf("%s", buffer);

wczytanie danych z klawiatury do bufora

c

Skopiuj kod

write(sockfd, buffer, strlen(buffer)+1);

Wysłanie danych do serwera przez socket.

+1 wysyła także znak końca stringa.

read(sockfd, buffer, sizeof(buffer));

odebranie odpowiedzi od serwera i zapisanie jej do bufora

if (strcmp(buffer, "exit") == 0)

break;

Jeśli serwer wyśle wiadomość „exit”, program kończy działanie pętli

close(sockfd);

zamknięcie połączenia sieciowego

server.c

int input_pipe[2];

int output_pipe[2];

Dwie tablice opisujące potoki:

input_pipe — do przekazywania danych do wątków

output_pipe — do odbierania wyników z wątków

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

Mutex używany do synchronizacji wątków, żeby nie korzystały z tych samych danych jednocześnie

void run_server() {

Funkcja odpowiedzialna za uruchomienie i działanie serwera

int server_fd, client_fd;

Deskryptory:

server_fd — socket serwera

client_fd — socket klienta

struct sockaddr_in addr;

Struktura przechowująca adres IP i port serwera

pipe(input_pipe);

pipe(output_pipe);

Tworzenie dwóch potoków do komunikacji między wątkami a serwerem

pthread_t t1, t2;

Zmienne przechowujące identyfikatory wątków

pthread_create(&t1, NULL, worker_thread, NULL);

pthread_create(&t2, NULL, worker_thread, NULL);

Utworzenie dwóch wątków roboczych, które wykonują funkcję worker_thread

server_fd = socket(AF_INET, SOCK_STREAM, 0);

Utworzenie socketu serwera: IPv4, TCP, domyślny protokół

addr.sin_family= AF_INET;

Ustawienie protokołu IPv4.

addr.sin_port = htons(5050);

Ustawienie portu serwera na 5050 (konwersja do formatu sieciowego)

addr.sin_addr.s_addr = INADDR_ANY;

Serwer nasłuchuje na wszystkich dostępnych interfejsach sieciowych

bind(server_fd, (struct sockaddr*)&addr, sizeof(addr));

Przypisanie adresu IP i portu do socketu serwera

listen(server_fd, 1);

Przełączenie socketu w tryb nasłuchiwania (maksymalnie 1 klient w kolejce)

client_fd = accept(server_fd, NULL, NULL);

Oczekiwanie na połączenie od klienta i jego zaakceptowanie

read(client_fd, buffer, sizeof(buffer));

Odczyt danych wysłanych przez klienta

write(input_pipe[1], buffer, strlen(buffer)+1);

Przekazanie danych z klienta do wątków roboczych przez potok

read(output_pipe[0], buffer, sizeof(buffer));

Odebranie przetworzonych danych z wątków roboczych

write(client_fd, buffer, strlen(buffer)+1);

Odesłanie odpowiedzi do klienta

if (strcmp(buffer, "exit") == 0)

break;

Jeżeli odpowiedź to „exit”, serwer kończy działanie pętli

close(client_fd);

Zamknięcie połączenia z klientem

close(server_fd);

Zamknięcie socketu serwera

worker.c

extern int input_pipe[2];

Deklaracja potoku wejściowego, który został utworzony w innym pliku (np. w serwerze)

extern int output_pipe[2];

Deklaracja potoku wyjściowego do odsyłania danych z wątku

extern pthread_mutex_t mutex;

Deklaracja mutexa, który służy do synchronizacji dostępu do wspólnych zasobów

void* worker_thread(void* arg) {

Funkcja wykonywana przez wątek roboczy

read(input_pipe[0], buffer, sizeof(buffer));

Odczyt danych z potoku wejściowego, który pochodzi od serwera

pthread_mutex_lock(&mutex);

Zablokowanie mutexa, aby tylko jeden wątek naraz miał dostęp do wspólnych danych

if (strcmp(buffer, "exit") == 0) {

Sprawdzenie, czy odebrana wiadomość to „exit”

write(output_pipe[1], buffer, strlen(buffer)+1);

Przekazanie informacji „exit” dalej do serwera

pthread_mutex_unlock(&mutex); Odblokowanie mutexa.

buffer[0] = 'X';

Modyfikacja danych — pierwszy znak wiadomości zostaje zmieniony na 'X'

write(output_pipe[1], buffer, strlen(buffer)+1);

Wysłanie przetworzonej wiadomości do potoku wyjściowego

pthread_mutex_unlock(&mutex);

Odblokowanie mutexa po zakończeniu pracy na danych

main.c

Program uruchamia serwer TCP, który nasłuchuje na porcie 5050 i komunikuje się z jednym klientem. Odebrane wiadomości są przekazywane do wątków roboczych, gdzie są prosto modyfikowane, a następnie odsyłane z powrotem do klienta. Synchronizacja wątków odbywa się za pomocą mutexa, a wysłanie komunikatu „exit" kończy działanie programu.

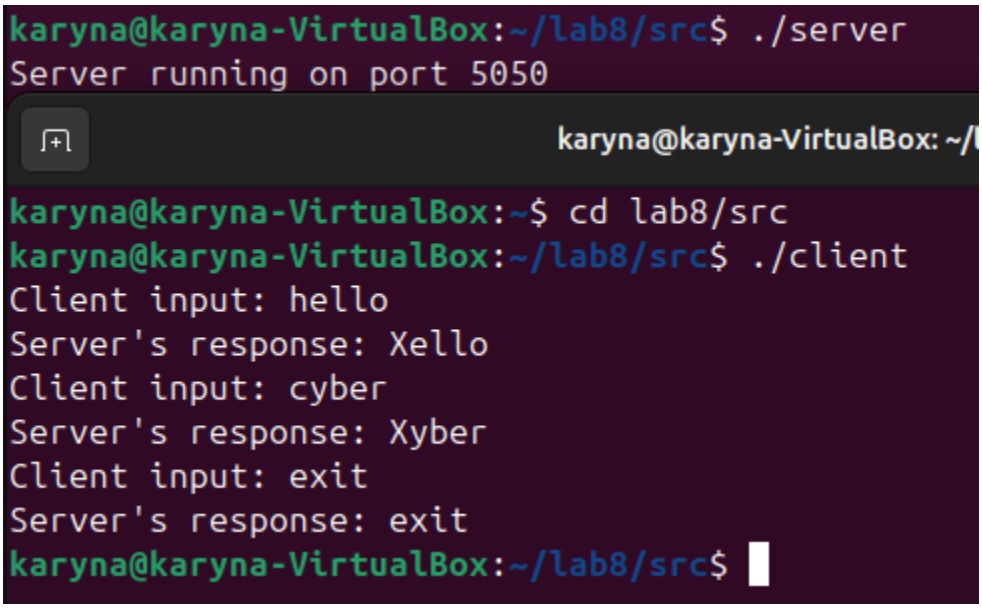
server.h

Plik zawiera deklarację funkcji `run_server`, która uruchamia działanie serwera. Chroniony jest dyrektywami `ifndef`, aby nie został dołączony wielokrotnie.

worker.h

Plik nagłówkowy zawiera deklarację funkcji wątku roboczego `worker_thread` i zabezpieczenie przed wielokrotnym dołączeniem pliku.

Wynik:



Klient łączy się z serwerem na porcie 5050 i wysyła wiadomości. Serwer przekazuje je do wątków roboczych, które zmieniają pierwszy znak wiadomości na „X" i odsyłają z

powrotem. Serwer wysyła zmodyfikowaną wiadomość do klienta. Proces powtarza się do momentu, aż klient lub wątek wyśle komunikat „exit”, wtedy połączenie zostaje zamknięte.