Idea programu:

Program składa się z 5 plików: main.c, server.c, worker.c, server.h, worker.h.

Działanie:

1. Klient wysyła tekst do serwera przez sieć
2. Server odbiera tekst
3. Server przekazuje tekst do workera za pomocą pipe
4. Worker przetwarza dane – zmienia pierwszą na “X”
5. Worker odsyła wynik do servera przez pipe
6. Server odsyła wynik z powrotem do klienta

main.c:

main.c jest bardzo prosty – tylko uruchamia serwer

server.c:

int pipe_in[2]; int pipe_out[2]; pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER – tworzymy pipe do wysyłania do workerów, pipe do odbioru od workerów, mutex do synchronizacji wątków

void start_server() - uruchomienie serwera

int server_fd - gniazdo serwera; client_fd - połączenie z konkretnym klientem

struct sockaddr_in addr - struktura, w której zapisujemy: adres IP, numer portu, typ połączenia

char buf[100] - bufor na tekst, który przychodzi od klienta i wraca z workera

pipe(pipe_in); pipe(pipe_out) - tworzymy dwa potoki komunikacyjne

pthread_t t1, t2; pthread_create(&t1, NULL, worker_thread, NULL); pthread_create(&t2, NULL, worker_thread, NULL) - tworzymy dwa wątki workerów

server_fd = socket(AF_INET, SOCK_STREAM, 0) - tworzymy gniazdo sieciowe(socket TCP)

addr.sin_family= AF_INET; addr.sin_port = htons(1234); addr.sin_addr.s_addr = INADDR_ANY - Ustawienia serwera: IPv4, port 1234, nasłuchiwanie na wszystkich interfejsach

bind(server_fd, (struct sockaddr*)&addr, sizeof(addr)); listen(server_fd, 1) - server zaczyna nasłuchiwać na porcie

client_fd = accept(server_fd, NULL, NULL) - server czeka na połączenie klienta

read(client_fd, buf, sizeof(buf)) - odczyt danych od klienta

write(pipe_in[1], buf, strlen(buf)+1) - server przekazuje dane do workera przez pipe

read(pipe_out[0], buf, sizeof(buf)) - server odbiera przetworzone dane od workera

write(client_fd, buf, strlen(buf)+1) - server wysyła wynik z powrotem do klienta

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

break; - jeśli klient wysłał "exit", kończymy pracę

close(client_fd); close(server_fd) - zamykanie połączeń sieciowych

worker.c:

extern int pipe_in[2]; extern int pipe_out[2]; extern pthread_mutex_t mutex - worker korzysta ze zmiennych zdefiniowanych w server.c: pipe do odbioru danych, pipe do wysyłania danych, mutex do synchronizacji

void* worker_thread(void* arg) - funkcja workera uruchamiana jako osobny wątek

char buf[100] - bufor ta tekst

read(pipe_in[0], buf, sizeof(buf)) - worker czyta dane wysłane przez server przez pipe

pthread_mutex_lock(&mutex) - blokada mutexu – tylko jeden wątek może teraz pracować na danych

if (strcmp(buf, "exit") == 0) - jeśli otrzymano "exit", kończymy działanie

write(pipe_out[1], buf, strlen(buf)+1) - worker odsyła "exit" do servera

pthread_mutex_unlock(&mutex) - zwalnia mutex i kończy pętlę

buf[0] = 'X' - worker zmienia pierwszą literę na “X”

write(pipe_out[1], buf, strlen(buf)+1) - odsyła zmodyfikowany tekst do servera przez pipe

pthread_mutex_unlock(&mutex) - zwalnia mutex

server.h:

plik server.h zawiera deklarację funkcji start_server(), dzięki czemu może być ona używana w pliku main.c

worker.h:

plik worker.h zawiera deklarację funkcji workera, która jest uruchamiana jako osobny wątek przy pomocy biblioteki pthread

client.c:

- łączy się z serwerem przez sieć (localhost, port 1234)
- wysyła tekst wpisany przez użytkownika do serwera
- odbiera odpowiedź z serwera
- wypisuje odpowiedź na ekran

int sock - deskryptor gniazda

struct sockaddr_in addr - struktura z adresem serwera (IP + port)

char buf[100] - bufor na tekst

sock = socket(AF_INET, SOCK_STREAM, 0) - tworzysz socket: AF_INET - IPv4, SOCK_STREAM - TCP, 0 - domyślny protokół (TCP)

addr.sin_family = AF_INET - adres też jest IPv4

addr.sin_port = htons(1234) - htons zamienia kolejność bajtów na sieciową, port serwera - 1234

addr.sin_addr.s_addr = inet_addr("127.0.0.1") - adres IP serwera: 127.0.0.1(localhost)

connect(sock, (struct sockaddr*)&addr, sizeof(addr)) - tutaj następuje połączenie z serwerem

write(sock, buf, strlen(buf)+1) - dane wysłane do serwera przez socket: sock - gdzie, buf - co, strlen(buf)+1 - długość razem z \0

read(sock, buf, sizeof(buf)) - odbiera odpowiedź z serwera do tego samego bufora

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

break; - jeśli serwer odeśle "exit", to klient kończy pętlę

close(sock) - zamyka połączenie sieciowe

Wynik działania:

server uruchomiony jest na porcie 1234. client wysyła kolejne teksty: hello, cyber.

server odsyła odpowiedzi zmodyfikowane przez workera: Xello, Xyber.

po wpisaniu “exit” program kończy działanie