ASSIGNMENT

Que1server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 9999

#define BUFFER\_SIZE 1024

// Function to reverse a string

void reverse\_string(char \*str) {

int len = strlen(str);

for (int i = 0; i < len / 2; i++) {

char temp = str[i];

str[i] = str[len - i - 1];

str[len - i - 1] = temp;

}

}

// Function to handle client connections

void handle\_client(int client\_socket) {

char buffer[BUFFER\_SIZE];

int bytes\_received;

while ((bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0)) > 0) {

// Null terminate the received data

buffer[bytes\_received] = '\0';

// Reverse the received string

reverse\_string(buffer);

// Send the reversed string back to the client

send(client\_socket, buffer, strlen(buffer), 0);

}

// Close the client socket

close(client\_socket);

}

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_addr\_len = sizeof(client\_addr);

// Create a TCP socket

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize server address structure

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the specified port

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_socket, 5) == -1) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

// Accept a new connection

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (client\_socket == -1) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Connection from %s:%d\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

// Fork a new process to handle the client

pid\_t pid = fork();

if (pid == -1) {

perror("Fork failed");

exit(EXIT\_FAILURE);

}

if (pid == 0) { // Child process

// Close the server socket in the child process

close(server\_socket);

// Handle the client

handle\_client(client\_socket);

// Close the client socket in the child process

close(client\_socket);

// Exit the child process

exit(EXIT\_SUCCESS);

} else { // Parent process

// Close the client socket in the parent process

close(client\_socket);

}

}

// Close the server socket

close(server\_socket);

return 0;

}

Que1client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"

#define PORT 9999

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

// Create a TCP socket

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize server address structure

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

server\_addr.sin\_port = htons(PORT);

// Connect to the server

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

// Input a string to send to the server

printf("Enter a string to reverse: ");

fgets(buffer, BUFFER\_SIZE, stdin);

buffer[strcspn(buffer, "\n")] = 0; // Remove newline character from the input

// Send the string to the server

send(client\_socket, buffer, strlen(buffer), 0);

// Receive the reversed string from the server

int bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0';

printf("Reversed string received from server: %s\n", buffer);

// Close the socket

close(client\_socket);

return 0;

}

Que2server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <math.h>

#include <pthread.h>

#define MAX\_MSG\_SIZE 1024

double evaluate\_expression(char \*expr) {

char \*token;

double result;

char operator;

// Get the first token

token = strtok(expr, " ");

// Convert the first token to a double

sscanf(token, "%lf", &result);

// Get subsequent tokens

while ((token = strtok(NULL, " ")) != NULL) {

// Check if the token is an operator

if (strchr("+-\*/", token[0]) != NULL) {

operator = token[0];

} else {

// Convert the token to a double and apply the operator

double operand;

sscanf(token, "%lf", &operand);

switch (operator) {

case '+':

result += operand;

break;

case '-':

result -= operand;

break;

case '\*':

result \*= operand;

break;

case '/':

result /= operand;

break;

default:

printf("Invalid operator\n");

return NAN; // Not-a-Number

}

}

}

return result;

}

// Function to handle client connections

void \*handle\_client(void \*arg) {

int client\_socket = \*((int \*)arg);

char buffer[MAX\_MSG\_SIZE];

while (1) {

// Receive expression from client

recv(client\_socket, buffer, MAX\_MSG\_SIZE, 0);

printf("Expression received: %s\n", buffer);

// Check for exit condition

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

printf("Client disconnected\n");

close(client\_socket);

pthread\_exit(NULL);

}

// Evaluate expression

double result = evaluate\_expression(buffer);

// Convert result to string

sprintf(buffer, "%.2lf", result);

// Send result to client

send(client\_socket, buffer, strlen(buffer), 0);

printf("Result sent: %s\n", buffer);

}

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

fprintf(stderr, "Usage: %s <port>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

// Create socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt))) {

perror("setsockopt failed");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(atoi(argv[1]));

// Bind socket to address and port

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %s\n", argv[1]);

while (1) {

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Connection accepted from %s:%d\n", inet\_ntoa(address.sin\_addr), ntohs(address.sin\_port));

// Create a new thread to handle the client

pthread\_t tid;

if (pthread\_create(&tid, NULL, handle\_client, (void \*)&new\_socket) != 0) {

perror("Thread creation failed");

exit(EXIT\_FAILURE);

}

// Detach the thread

pthread\_detach(tid);

}

return 0;

}

Que2client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define MAX\_MSG\_SIZE 1024

int main(int argc, char \*argv[])

{

    if (argc != 3)

    {

        fprintf(stderr, "Usage: %s <server\_hostname> <port>\n", argv[0]);

        exit(EXIT\_FAILURE);

    }

    struct sockaddr\_in server\_addr;

    int sock\_fd;

    char buffer[MAX\_MSG\_SIZE];

    if ((sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

    {

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(atoi(argv[2]));

    if (inet\_pton(AF\_INET, argv[1], &server\_addr.sin\_addr) <= 0)

    {

        perror("Invalid address/ Address not supported");

        exit(EXIT\_FAILURE);

    }

if (connect(sock\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

    {

        perror("Connection failed");

        exit(EXIT\_FAILURE);

    }

    while (1)

    {

        printf("Enter arithmetic expression (or 'quit' to exit): ");

        fgets(buffer, MAX\_MSG\_SIZE, stdin);

        buffer[strcspn(buffer, "\n")] = 0;

        send(sock\_fd, buffer, strlen(buffer), 0);

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

        {

            printf("Exiting...\n");

            break;

        }

        recv(sock\_fd, buffer, MAX\_MSG\_SIZE, 0);

        printf("Result: %s\n", buffer);

    }

    close(sock\_fd);

    return 0;

}