**RMS**

S CODE  #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/wait.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

// Function to perform the requested arithmetic operation

float perform\_operation(int num1, int num2, char operator, int \*error) {

float result;

\*error = 0;

switch (operator) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

if (num2 != 0) {

result = (float) num1 / num2;

} else {

\*error = 1; // Division by zero error

result = 0;

}

break;

default:

\*error = 1; // Invalid operator error

result = 0;

break;

}

return result;

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

int num1, num2, error;

char operator;

// Create socket

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

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Bind the socket to the network

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(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 %d...\n", PORT);

while (1) {

// Accept a new connection

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

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Fork a new process to handle the client

if (fork() == 0) {

// Child process handles the client

close(server\_fd); // Child process doesn't need the server socket

// Read the data from the client

read(new\_socket, buffer, BUFFER\_SIZE);

sscanf(buffer, "%d %c %d", &num1, &operator, &num2);

// Perform the operation

float result = perform\_operation(num1, num2, operator, &error);

// Prepare the response

if (error == 1) {

sprintf(buffer, "Error: Invalid operation.");

} else {

sprintf(buffer, "Result: %.2f", result);

}

// Send the result back to the client

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

// Close the client socket

close(new\_socket);

// Terminate the child process

exit(0);

} else {

// Parent process continues to accept new connections

close(new\_socket); // Parent doesn't need the client socket

}

// Reap zombie processes

while (waitpid(-1, NULL, WNOHANG) > 0);

}

return 0;

}

C CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

int num1, num2;

char operator;

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

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

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

// Convert IPv4 address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

// Get two integers and an operator from the user

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter an operator (+, -, \*, /): ");

scanf(" %c", &operator);

printf("Enter second number: ");

scanf("%d", &num2);

// Send numbers and operator to server

sprintf(buffer, "%d %c %d", num1, operator, num2);

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

// Receive the result from server

read(sock, buffer, BUFFER\_SIZE);

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

// Close the socket

close(sock);

return 0;

}