Modbus communication C/C++ code without third party library

Implementing Modbus TCP communication in C/C++ without using a third-party library requires handling the low-level details of the Modbus protocol, including constructing the appropriate message frames and managing TCP socket communication.

Let's walk through a complete example of reading and writing Modbus registers using direct TCP socket communication.

**Basic Requirements**

1. **TCP Socket Setup**: Create and manage a TCP connection.
2. **Modbus TCP Framing**: Construct and parse Modbus protocol messages.

**Example Code**

Below is a basic example to help you understand Modbus TCP communication without any third-party library.

**1. Include Necessary Headers**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

C++

Copy code

**2. TCP Socket Initialization**

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

C++

Copy code

**3. Modbus TCP Request/Response Functions**

**3.1 Write Register Function**

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06;

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

C++

Copy code

**3.2 Read Registers Function**

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03;

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

C++

Copy code

Copy

Good response

Bad response

continue

Certainly, let's continue from where we left off with the readRegisters function and then complete the main program.

**Continued: Read Registers Function**

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[9 + i \* 2] << 8) | response[10 + i \* 2];

}

return true;

}

C++

Copy code

**Main Function**

Now, putting everything together in the main function:

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the socket

close(sockfd);

return 0;

}

C++

Copy code

**Complete Example: All Together**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

// Function prototypes

int initializeSocket(const char\* ip, int port);

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer);

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

C++

Copy code

Copy

Good response

Bad response

continue

Certainly, here is the continuation and completion of the previous example:

// Close the socket

close(sockfd);

return 0;

}

// Function to initialize a TCP socket

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

// Function to write a single register

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06; // Function code for writing a single holding register

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

// Function to read multiple registers

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03; // Function code for reading holding registers

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if

C++

Copy code

Copy

Good response

Bad response

continue

Alright, let's continue with the rest of the readRegisters function and complete this Modbus TCP client example.

**Continued: Read Registers Function**

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

if (byteCount != 2 \* numRegisters) {

std::cerr << "Unexpected byte count: " << byteCount << std::endl;

return false;

}

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[9 + i \* 2] << 8) | response[10 + i \* 2];

}

return true;

}

C++

Copy code

**Complete Example: All Together**

Putting all parts together, the complete example code would be:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

// Function prototypes

int initializeSocket(const char\* ip, int port);

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer);

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the socket

close(sockfd);

return 0;

}

// Function to initialize a TCP socket

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

// Function to write a single register

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06; // Function code for writing a single holding register

frame[8] = (register

C++

Copy code

Copy

Good response

Bad response

continue

Certainly, here is the continuation of the writeRegister function and the complete example in one place.

**Continued: Write Register Function**

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

C++

Copy code

**Complete Example: All Together**

Here is the complete example:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

// Function prototypes

int initializeSocket(const char\* ip, int port);

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer);

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the socket

close(sockfd);

return 0;

}

// Function to initialize a TCP socket

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

// Function to write a single register

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

C++

Copy code

Copy

Good response

Bad response

continue

Certainly, here is the continuation and completion of the writeRegister function, followed by the readRegisters function and the complete example:

**Continued: Write Register Function**

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06; // Function code for writing a single holding register

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

C++

Copy code

**Function: Read Registers**

Here is the complete readRegisters function:

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03; // Function code for reading holding registers

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

if (byteCount != 2 \* numRegisters) {

std::cerr << "Unexpected byte count: " << byteCount << std::endl;

return false;

}

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[9 + i \* 2] << 8) | response[10 + i \* 2];

}

return true;

}

C++

Copy code

**Complete Example: All Together**

Here is the complete example with both functions writeRegister and readRegisters together:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

// Function prototypes

int initializeSocket(const char\* ip, int port);

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer);

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

C++

Copy code

Copy

Good response

Bad response

continue

Certainly, here is the continuation of the main function and complete example code with initializeSocket, writeRegister, and readRegisters:

**Continued: Complete Main Function**

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the socket

close(sockfd);

return 0;

}

// Function to initialize a TCP socket

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

// Function to write a single register

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06; // Function code for writing a single holding register

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

// Function to read registers

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID

C++

Copy code

Copy

Good response

Bad response

continue

Sure, here is the continuation and completion of the readRegisters function, and the complete example together:

**Continued: Read Registers Function**

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03; // Function code for reading holding registers

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

if (byteCount != 2 \* numRegisters) {

std::cerr << "Unexpected byte count: " << byteCount << std::endl;

return false;

}

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[9 + i \* 2] << 8) | response[10 + i \* 2];

}

return true;

}

C++

Copy code

**Complete Example: All Together**

Here is the complete example with both writeRegister and readRegisters functions, along with the initializeSocket and main function:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

// Function prototypes

int initializeSocket(const char\* ip, int port);

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer);

int main() {

const char\* ip = "192.168.1.100"; // Replace with the IP of your Modbus server

int port = 502; // Modbus TCP standard port

int sockfd = initializeSocket(ip, port);

if (sockfd == -1) {

return -1;

}

uint8\_t slaveID = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(sockfd, slaveID, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(sockfd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(sockfd, slaveID, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(sockfd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the socket

close(sockfd);

return 0;

}

// Function to initialize a TCP socket

int initializeSocket(const char\* ip, int port) {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

std::cerr << "Failed to create socket: " << strerror(errno) << std::endl;

return -1;

}

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

server

C++

Copy code

Copy

Good response

Bad response

continue

Sure, let's continue with the initializeSocket function and complete the example:

**Function to Initialize a TCP Socket**

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, ip, &serverAddr.sin\_addr) <= 0) {

std::cerr << "Invalid address: " << ip << std::endl;

close(sockfd);

return -1;

}

if (connect(sockfd, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Connection failed: " << strerror(errno) << std::endl;

close(sockfd);

return -1;

}

return sockfd;

}

// Function to write a single register

bool writeRegister(int sockfd, uint8\_t slaveID, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x06; // Function code for writing a single holding register

frame[8] = (registerAddress >> 8) & 0xFF;

frame[9] = registerAddress & 0xFF;

frame[10] = (value >> 8) & 0xFF;

frame[11] = value & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

// Function to read registers

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03; // Function code for reading holding registers

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

if (byteCount != 2 \* numRegisters) {

std::cerr << "Unexpected byte count: " << byteCount << std::endl;

return false;

}

for (int i =

C++

Copy code

Copy

Good response

Bad response

continue

}

uint8\_t response[12];

if (recv(sockfd, response, sizeof(response), 0) != sizeof(response)) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x06) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

return true;

}

// Function to read registers

bool readRegisters(int sockfd, uint8\_t slaveID, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t\* buffer) {

uint8\_t frame[12];

static uint16\_t transactionID = 0;

transactionID++;

// Construct Modbus TCP frame

frame[0] = (transactionID >> 8) & 0xFF;

frame[1] = transactionID & 0xFF;

frame[2] = 0x00;

frame[3] = 0x00;

frame[4] = 0x00;

frame[5] = 0x06;

frame[6] = slaveID;

frame[7] = 0x03; // Function code for reading holding registers

frame[8] = (startAddress >> 8) & 0xFF;

frame[9] = startAddress & 0xFF;

frame[10] = (numRegisters >> 8) & 0xFF;

frame[11] = numRegisters & 0xFF;

if (send(sockfd, frame, sizeof(frame), 0) == -1) {

std::cerr << "Failed to send frame: " << strerror(errno) << std::endl;

return false;

}

// Calculate the expected response length

int responseLength = 9 + 2 \* numRegisters;

uint8\_t response[responseLength];

if (recv(sockfd, response, responseLength, 0) != responseLength) {

std::cerr << "Failed to receive response: " << strerror(errno) << std::endl;

return false;

}

if (response[7] != 0x03) {

std::cerr << "Invalid response function code: " << static\_cast<int>(response[7]) << std::endl;

return false;

}

int byteCount = response[8];

if (byteCount != 2 \* numRegisters) {

std::cerr << "Unexpected byte count: " << byteCount << std::endl;

return false;

}

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[9 + i \* 2] << 8) | response[10 + i \* 2];

}

return true;

}

C++

Copy code

**Summary**

This code demonstrates how to perform Modbus TCP operations on a remote server. It includes functions to initialize a socket, write to a register, and read multiple registers. The main function showcases a practical use case of these functions:

1. Initialize the socket.
2. Write a value to a specific register.
3. Read a set number of registers starting from a specified address.
4. Print the values read from the registers.
5. Close the socket.

If you have specific requirements or modifications needed for your project, please let me know!