CI –Comunicações Industriais – Industrial Communications
2022/2023 – 1st semester

Modbus TCP protocol stack for TCP clients (and servers)

Introduction: Modbus is one of the most used communication protocols in industry tailored to the supervision and control of remote processes. Among other features, the Modbus protocol can operate over the Internet (or an intranet) using TCP/IP, which results in a high level of flexibility with respect to the connection of equipment. In this assignment we will implement a Modbus TCP protocol stack over TCP/IP sockets, building a library of functions that manage the communication over Modbus. The work is individual, can be carried out remotely and should be concluded in the respective class of the 10th to the 14th of October, in which there will be an assessment.

Objectives: The main objectives of this assignment are:

•Understand the concepts of layer, service and protocol in the scope of industrial networks.

•Materialize these concepts by implementing a library of functions to develop Modbus TCP clients using TCP sockets. (We still keep some guidelines to develop servers for those willing to go further, but we just require the development of a client.)

•Deepen the knowledge of the Modbus protocol.

Duration: 4 weeks, including the assessment.

Workplan:

•Week 1 – Introduction to Modbus, use standard applications, inspect layers with Wireshark.
See specific guide (*Communicating with Modbus*) plus documentation below
Exercising TCP sockets in C. Creation of a simple TCP client and server application.
See specific guide (*Creating and using TCP sockets in C*)

•Weeks 2-4 – Development of a Modbus TCP library in C for a Modbus Client.
See the remainder of this guide.
In the last lab session, the student will develop and submit (Moodle) a simple program.

Documentation: To carry out this assignment, it is mandatory to read information on Modbus. We suggest the following order of readings:

1.Modbus chapter – it has a compact description of the Modbus protocol overall. It may be enough for an initial simplified implementation. However, several details for complete implementation will require the following readings:

2.Modbus Application Protocol Specification – detailed description of Modbus data model and its functions. Focus on chapters 3-5 and 7 (Exceptions).

3.Modbus Messaging on TCP/IP Implementation Guide – detailed description of Modbus implementation over TCP/IP. Read chapters 1-3. Chapter 4 is recommended for those willing to carry on with more advanced parts of the work.

Support Modbus client/server applications: In order to analyze how ModbusTCP works (week 1) and to test your work (in weeks 2-4), it is possible to use standard Modbus programs, either clients or servers. For example, once you build a very simple client, you can test it right away with a standard Modbus server. Similarly, if you have developed a server, you can test it using a standard Modbus client.

- Freeware (Textual interface, only. Have to be invoked through the command line)
  - Digslave Modbus Slave Simulator - Modbus server.
  - Modpoll Modbus Master Simulator - Modbus client.
- Limited licences (Free trial for 21 days. For Windows. Graphical interface)
  - Modbus Slave - Modbus server.
  - Modbus Poll - Modbus client.

Week 1 – Introduction to Modbus TCP

This preliminary work aims at introducing Modbus TCP from the point of view of a communication protocol. It is important to read the accompanying literature on Modbus and Modbus TCP, particularly introduction and communication frames (see accompanying slides).

To exercise the use of Modbus we will use two Windows applications that are already available in the workbench computers, namely *ModbusPoll* and *ModbusSlave*.

First, configure the server (*ModbusSlave*) with the needed Modbus parameters (type (register/coil) and number of memory positions, their addresses and functions). Note that these Modbus memory positions will be available to be read/written by clients through the network. Then, configure the communication parameter, namely, to work over TCP on port 502. This is the standard Modbus port.

Second, configure the client (*ModbusPoll*) to access the server using TCP on the *loopback* IP address. Configure to issue requests to the server on demand (later, change to periodic requests). Exercise with different functions and different number of items (registers or coils) to read/write and their addresses. When you change the requests to periodic, the client will continually send the configured request to the server. In case of a read request, if you change a value of a read position in the server, the client will automatically show that change.

In the client, switch the loopback IP address with the IP address of the computer of the next workbench. Repeat for different functions

Now, launch the Wireshark network sniffer and configure it to filter ModbusTCP packets. With the ModbusPoll accessing the ModbusSlave server periodically, capture a few packets. Inspect the two upper layers, namely Modbus and ModbusTCP, and compare with what you have read in the accompanying literature.

Weeks 1- and homework – Developing and using TCP sockets in C

Using the remainder of week 1 and continuing as homework, we will exercise the creation and use of TCP sockets as programming interfaces to communication channels. These will then be used in the remainder of the first assignment.

Using the Cygwin environment (Linux-based) that is installed in the lab computers, write a distributed client-server application in the C language, in which the server implements a string echo function, i.e., it receives ASCII strings and returns them without modification, while the client simply sends an ASCII string to the server, waits for the answer, prints it on the screen and terminates.

Note that the client and the server will be two different processes. The easiest way to manage these two processes is to open two Cygwin windows and develop each one on its own window. The processes will communicate via the loopback address of the IP protocol stack, i.e., the IP address 127.0.0.1.

After having this simple client-server application running in your computer, try crossing the client and server with those of other student by adequately changing the IP address and port (alternatively, you can install the server in another computer). Then return to the local configuration with the loopback address to continue the work.

Now, add some processing to the server side by returning the received string but with all lower-case characters converted to upper case.

As a homework, change the client and the server to work in a continuous fashion. The client repeatedly reads a string from the keyboard, sends it to the server, waits for the answer and prints it, until the character '#' is input. The server should equally work in a cycle, reading, updating and writing the strings.

Development environment: The programming language shall be the C language compiled with gcc/Linux. In the lab computers running Windows we will use Cygwin that offers a Linux-like runtime environment, but within Windows. You may use other C programming environment (e.g., in Windows) but the sockets library and interface may be slightly different.

Support literature: Some reading on programming TCP sockets is also recommended, particularly the following book, available in the library, has many ready to use examples:

- D. Comer, Internetworking with TCP/IP, Volume 3: Client-Server Programming and Applications

Weeks 2 to 4 – Developing a Modbus TCP library in C

A Modbus client (server) will generally support a limited set of Modbus functions. In this work, the following Modbus functions should be implemented:

- 03 (0x03) Read Holding Register
- 16 (0x10) Write Multiple Registers

The Modbus TCP protocol stack should be organized in two layers, namely ModbusAP (application) and ModbusTCP (session). It is mandatory to follow the usual Request-Confirmation-Indication-Response protocol typical of communication across layers in client-server interactions. The functions used in each layer should be kept in two separate source files, namely *ModbusAP.c* and *ModbusTCP.c* with the corresponding header files *ModbusAP.h* and *ModbusTCP.h*. See the accompanying slides.

Then the client (and server, for those willing to go further) program should also be developed in a separate source file, namely *client.c* (and *server.c*), which use the API provided in *ModbusAP.c*

The client should be able to interact with a standard Modbus application, for example, the ModbusSlave application used in the first week. It should be able to read and write registers in there.