TCP/IP BASED PLC CONNECTION TO DOOCS

Gerhard Grygiel, Michael Böhnert, DESY Hamburg

Abstract

In industrial control systems the Ethernet based communication is gaining a more dominant role. For several tasks and subsystems of the TTF VUV-FEL (Vacuum-Ultraviolet Free Electron Laser at DESY) PLCs (Programmable Logic Controllers) are used in reliable low level controls. These PLCs are connected via TCP/IP communication to DOOCS [2] (Distributed Object Oriented Control System). A class library on the DOOCS server side implements the interface to the PLC. It contains functions for the TCP/IP communication and the methods to exchange different data structures of a PLC. The developed class library allows script based server generation for different PLC applications. The DOOCS server processes are running on Unix/Linux computers. In case of problems, the network-debugging on Unix/Linux computers is easy. There are many network analyzing tools available. This paper also discuses the error handling and TCP/IP configuration. At start-up or after connection lost special precautions have to be taken to resynchronize and to protect the valid data. TCP/IP configuration parameters are online available as well as communication statistics.

INTRODUCTION

At the outset of industrial revolution relays were used to operate automated machines, and these relays were interconnected using wires inside the control panel. Nowardays PLCs (Programmable Logic Controllers) are used for all kind of factory automatisation. In large systems like TTF VUV-FEL a growing number of distributed PLCs are installed. Therefore a DOOCS library was developed which hides all the complexity of communication and programming from the developer. For the developer it is quite enough to fill out a definition file. The automated server generation script then produces a full featured DOOCS server.

GENERAL PROCESS

PLC Process (Siemens S7 [3]):

The PLC can receive raw data in a TCP/IP packet or send raw data in a TCP/IP packet from/to the DOOCS server. A timer in the main PLC program cycle (OB1) triggers the send and receives function to the communication processor. Data blocks for send and receive needs to be defined before.

The PLC collects the 'data to send' in a data block. Whenever the timer calls FC5 (send to CP) the data are transported to the communication processor. If no connection is established the FC5 ignores the data. If the connection is established and the remote side is not available the FC5 returns an error. After a successful send the FC6 can be used to check if there is new data for the PLC to process. These data could be setpoints, commands or alive telegrams.

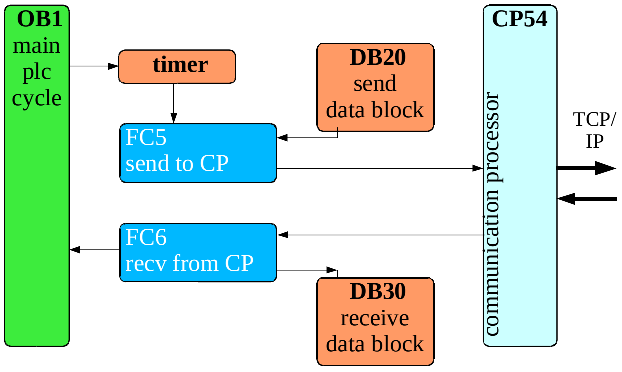


Figure 1: The PLC Process.

DOOCS Server Process:

The DOOCS Server starts one thread for receiving data from the PLC and another thread for sending data to the PLC. Both communication threads are using the same TCP/IP socket and both threads are able to initiates the connection. A mutex is used to take care that only one thread at a time talks to the PLC.

For the connection the SOCK_STREAM option is used. It provides a sequenced, reliable, two-way, connection-based byte stream. The maximum data size of the socket buffer needs also to be defined. To be able to define timeout values for the connection process, it is necessary to set the socket to non-blocking IO and use instead the select [4] call to check if the connection is established. If the connection is done the socket has to switch back to blocking IO. To prevent not to overrun the PLC, an adjustable time will be spent for waiting.

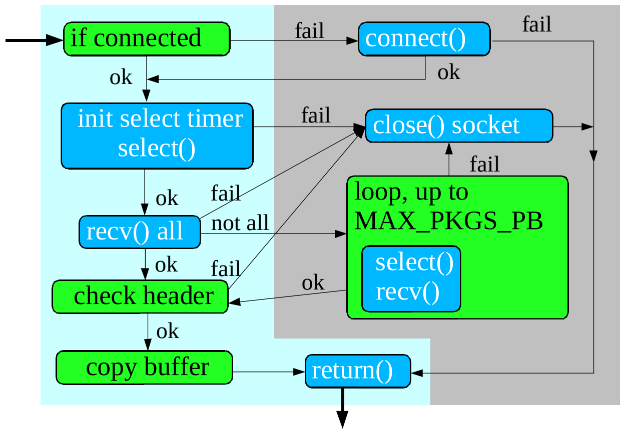


Figure 2: The DOOCS read thread.

The receiving thread checks a communication flag. If this is ok it defines a timeout value for a specified time to get data from the PLC. The select call is used to check for new data. The select call returns in the moment when the communication processor from the PLC gets new data. But that does not mean the complete data block is available in the communication processor, it only means that the data transport from the PLC to the communication processor has started. Therefore it is very important to wait some micro seconds (adjustable) on the DOOCS server side before the data receiving starts. Otherwise it is possible to get 50 byte in 50 separate TCP/IP packages. If receiving fails or not enough data was received the old connection will be closed and a new connection process starts. For the standard PLC DOOCS communication we have defined a so called data header. In the first two data words the data size and the last part of the own IP address is set. The data header check can be switched on or off. On the PLC side the header will be checked in addition. If all checks are ok the data is copied and converted in to the proper byte order to be available in a DOOCS data block and for archiving. Also some statistic values over the communication are stored.

The DOOCS server collects all commands for the PLC and puts them to a send queue. The sending thread gets the data from the send queue. As long as data is in the send queue the sending thread delivers the data to the PLC. The queue mechanism guarantees that no command will be lost. In the send thread there is also an adjustable wait implemented which avoid an overrun of the PLC with data packets.

DATA FLOW AND TIMING

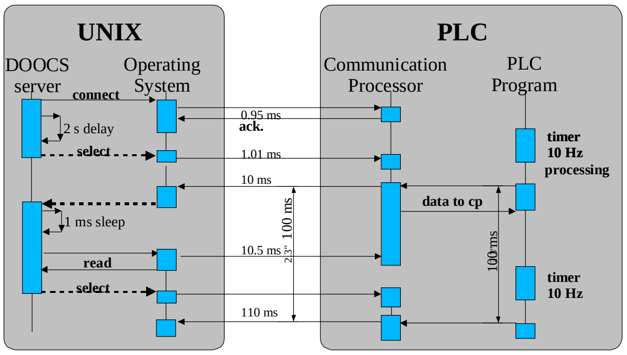


Figure 3: 10 Hz. 200 Byte.

For the beam inhibit system at VUV-FEL a connection with 10 Hz and 200 bytes per packet is in operation. Tests with this beam inhibit system PLC points out that the limit seems to be at 160 Hz with 200 bytes per packet.

DOOCS SERVERS

The DOOCS server model builds up a particular hardware device into a C++ object. This means, that the software describes the properties of this hardware device in a one to one relation. The server does all data processing and calculations. Libraries archive and configuration files are on the local hard disk. At start-up, the server creates as much as needed objects from a configuration file. The two main classes are:

Eq_fct class: is the container for all properties, an equipment function represents a device or only one location of a device.

D_fct class: describes a single property/value in an equipment function.

A DOOCS server is a standalone UNIX process which is permanently running. On the network it talks Sun RPC/XDR and TINE protocol. All items related to permissions, configurations, client connections and a lot more are done inside in a server library. The server programmer takes only care on the mapping from hardware to DOOCS properties. In DOOCS terminology a peace of hardware is represented as an equipment function (eq_fct) and all properties, that are accessible from the network, are represented as data functions (d_fct).

THE DOOCS PLC CLASS LIBRARY

A “D_plcnet” class is responsible for the PLC connection. It holds the configuration parameters and manages the data connection to the PLC. The data processing runs in a separate thread for every PLC. There are many ways of data setting. Data setting can be done on demand, when new data arrived or in the update function of the DOOCS server which is called by an adjustable timer.

For all major PLC data types corresponding classes are available in the DOOCS PLC class library.

For example to communicate to a float in the PLC the following constructor in the device server is required:

D_float_plcnet("FLOAT1 my float value", PLC, this, RO data_word);

- FLOAT1: DOOCS property name.
- PLC: pointer to the D_plcnet class which holds the data block to represent.
- this: pointer to the DOOCS container for the properties (eq_fct class).
- RO: a value of 1 means this property is read only. It dose not send data to the PLC.
- data_word is the starting position from the beginning of the PLC data block.

The D_float_plcnet class represents a PLC word which means two bytes. It is able to do polynomial data corrections, for instance to convert a PLC input value 0-65536 to 0-10 bar. The history is also managed in this class. For all major data types PLC classes are available (bit, int, float(16bit), iee32float). The programmer don't care about data representing or archiving, it is only necessary to define a variable for every PLC value which will come up in the DOOCS property list and to define one variable for the PLC itself. It is also possible to write a DOOCS server which represents more than one PLC.

ERROR HANDLING

The error handling is done on the equipment function level (device/location). Every data function (property/value) can produce an error (error bits, hi or low value errors...). If an equipment function has more than one error source per location, for instance two or more data functions which provides errors independently, the reset error needs a special handling. There is only one error-value per equipment function. Every data function holds own error and severity value internally and the corresponding gets and set public member functions. The error condition in the device instance reflects the logical or of all data functions.

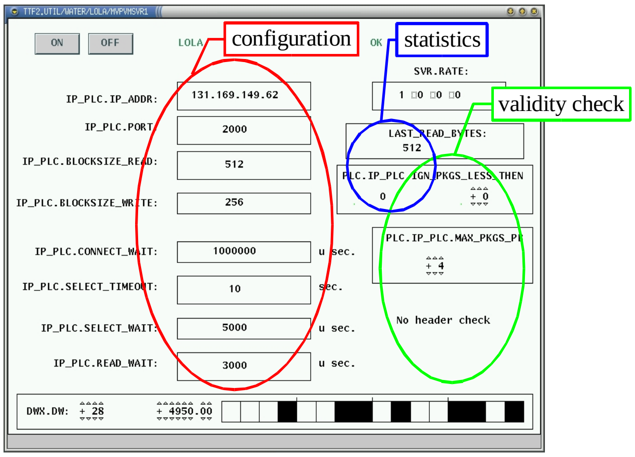
NETWORK CONFIGURATION, PLC COMMISSIONING

At first the PLC IP address, the port, and the block size for read and write data has to be defined. The timeouts and wait times have to be defined:

- Set the timeout 5 times longer than the PLC send cycle.
- The connect wait time times longer than the PLC send cycle.
- The select wait depends on the block size to read. A good starting value is 1 ms per 100 byte.

The following checks can be switched on:

- With or without header check.
- Ignore TCP/IP packages less then N bytes.
- Maximum TCP/IP packages per data block.



```
TTF2.UTIL/WATER/LOLA/MVPVMSVR1

IP_PLC.IP_ADDR:           131.169.149.62
IP_PLC.PORT:              2000
IP_PLC.BLOCKSIZE_READ:    512
IP_PLC.BLOCKSIZE_WRITE:   256
IP_PLC.CONNECT_WAIT:      1000000    u sec.
IP_PLC.SELECT_TIMEOUT:    10         sec.
IP_PLC.SELECT_WAIT:       5000       u sec.
IP_PLC.READ_WAIT:         3000       u sec.

SVR.RATE:  1 0 0 0
LAST_READ_BYTES: 512
PLC.IP_PLC_IGN_PKGS_LESS_THEN  0   + 0
PLC.IP_PLC.MAX_PKGS_PE  + 4
No header check

DWX.DW:  + 28   + 4950.00
```

Figure 4: DOOCS PLC configuration.

SCRIPT BASED SERVER GENERATION

Mainly all DOOCS PLC servers are generated with the automated server generation script. The script is able to generate the source for a complete DOOCS server from a definition file. All PLC servers have mainly the same structure but very different data to handle. The script extracts all needed variables from the definition file. It makes some general checks e.g. it checks that all needed template- and data function files are available. It creates the source and header files, the makefile and a first configuration file for the new server. The files and definitions for Debian package creation where also produced.

The main part of the definition file in figure 5 contains the constructors. The D_plcnet is the class for the communication with the PLC. The D_cmd class is used for commands to this PLC. The next lines are constructors for float, integer and bit representation.

```
D_plcnet      plcnet_("PLC.IP_PLC",this,0)
D__cmd        cTyH_("THY.ON thyratron on",plcnet_0,this, 8, 0, 5, 1)
D_vph_plcnet  tHV_("THY.FIL_VOLT voltage",plcnet_0, this, 1, 0)
D_int_plcnet  tSt_("THY.STATUS status",plcnet_0, this, 1, 5)
D_bit_plcnet  tSR_("THY.READY thyratron ready",plcnet_0, this, 1, 5, 0)
```

Figure 5: Part of the definitiion file.

The automated server generation script relieves the programmer of writing again and again the same structures for device servers and prevents mistyping. With the assistance of the automated server generation it is easy to handle a bunch of servers which differs only by their names and some simple properties. Therefore it is only necessary to develop a simple definition file.

PLC SUBSYSTEMS CONNECTED TO DOOCS

There are water cooling systems for the gun and for klystrons. Complete klystron subsystems build by industry. Valve controlers and other vacuum components build at DESY and interlock systems e.g BIS (beam inhibit system) and the laser interlock.

ACKNOLEDGEMENTS

We like to thank all the colleagues who have contributed to the development and commissioning of the TTF VUV-FEL PLCs connected to DOOCS via TCP/IP. In particular we wish to thank Olaf Krebs and Slava Korobov for their helpful suggestions. We also thank Arthur Agababyan for his help on thread programming in DOOCS servers.

REFERENCES

[1] DESY Deutsches Elektronen Synchrotron
http://www.desy.de/

[2] DOOCS Distributed Object Oriented Control System
http://doocs.desy.de/

[3] Siemens AG
http://www2.automation.siemens.com

[4] Zotteljedis Tipps zur Socket-Programmierung
http://www.zotteljedi.de/doc/socket-tipps/index.html