Distributed Programming I

A.Y. 2011-2012, Laboratory exercise n.1

Exercise 1.1 (test server)

In Linux environment, compile the provided test server: it will be listening for input to the specified port number, and it is able to calculate the sum of two input numbers.
Use the command:

gcc -o server_test -DTRACE server_test.c errlib.c sockwrap.c

To run it:

./server_test format port

where **format** is an option that specifies the data format that needs to be used to communicate with the client:

- **-a** to represent data in ASCII format
- **-x** to represent data in XDR

and **port** is the port number on which the server will be listening to (as ex. 1500).

Test the server by using the -a option and the **telnet** command as client application: it takes as input parameters the server address and the port number.

Note: to interrupt the connection inside the telnet application press CTRL + ALT GR +] then Q, followed by RETURN.

Note 2: to simplify things it is possible to use the functions provided by sockwrap.c (Socket, Connect, etc) that are an exact copy of the library functions but that also automatically checks the function return values in order to signal and report errors.

Exercise 1.2 (connection)

Write a client able to connect to a TCP server to the address and port number specified as first and second command-line parameters, respectively, that then terminates by properly closing the communication channel and by signaling if it has managed to perform the connection or not.

Exercise 1.3 (ASCII data)

Develop a client able to connect to a TCP server to the address and port number specified as first and second command-line parameters, respectively, that then sends two unsigned integer numbers (represented in pure binary over 16 bits) read from the standard input, and then receives the answer (sum, or error) from the server. As server it is possible to use the one compiled in the exercise 1.1
The sum is an 16-bit unsigned integer, but the server also handles the overflow case, by signaling it with a specific error (read the example).
The client sends to the server the two numbers by using a decimal notation in numeric ASCII characters. Numbers must be separated by a single space and transmission must be terminated by CR-LF. The server returns the result (a single number) by expressing it through ASCII characters, without spaces and terminated by CR-LF or a single error message (ASCII characters) also terminated by CR-LF.

Examples (every element represents a single byte sent in ASCII code):

(client -> server) 1 2 3 4 5 3 CR LF

(server -> client) 1 2 3 4 8 CR LF

or, in case of error:

(server -> client) o v e r f l o w CR LF

or:

(server -> client) i n c o r r e c t o p e r a n d s CR LF

Exercise 1.4 (client-server UDP base)

Write a client able to send to an UDP server (to the address and the port number specified by first and second parameter of the command line) a datagram containing name (max 31 characters) specified as third parameter on the command line, and that then awaits any reply datagram from the server. The client terminates by showing the content (ASCII text) of the received datagram or by signaling that it didn't receive any reply from the server in a certain timespan.

Develop an UDP server (listening to the port specified as first parameter on the command line) that replies to any received datagram by answering with a datagram containing the same name provided by the received packet.

Try to perform datagram exchanges between client and server with the following configurations:

- client sends a datagram to the same port the server is listening to
- client sends a datagram to a port in which the server is not listening
- client sends a datagram to an unreachable address (es. 10.0.0.1)

Try to connect your client to the server developed by another group and vice versa.