Distributed Programming (03NQVOC)
Distributed Programming I (03MQPOV)

Laboratory exercise n.2

Exercise 2.1 (perseverant UDP client)

Modify the UDP client of exercise 1.4 so that - if it does not receive any reply from the server in 3 seconds - it re-transmits the request (up to a maximum of 5 times) then it terminates by reporting if it has received the reply or not,
Perform the same tests of exercise 1.4

Exercise 2.2 (limiting UDP server)

Modify the UDP server of exercise 1.4 so that it replies to a client only if it does not have performed more than three requests from the same IP address (since the server has been activated). The server must be able to recognize the last 10 clients that have performed a request.
Try, then, to run four times the client of exercise 1.4 against this server.
Finally, try to run two clients against this server, placed in different network nodes, alternating between them, four times for each client.

Exercise 2.3 (iterative TCP server)

Develop a TCP server (listening to the port specified as first parameter of the command line) accepting file transfer requests from clients and sending the requested file.
Develop a client that can connect to a TCP server (to the address and port number specified as first and second command-line parameters, respectively), to request files, and store them locally. File names to be requested must be provided to the client using the standard input, one per line. Every requested file must be saved locally and the client must print a message to the standard output about the performed file transfer, with file name, size and timestamp of last modification.
The protocol for file transfer works as follows: to request a file the client sends to the server the three ASCII characters “GET” followed by the ASCII space character and the ASCII characters of the file name, terminated by the ASCII carriage return (CR) and line feed (LF):

G	E	T		…filename…	CR	LF

(Note: the command includes a total of 6 characters plus the characters of the file name). The server replies by sending:

+	O	K	CR	LF	B1	B2	B3	B4	T1	T2	T3	T4	File content……

Note that this message is composed of 5 characters followed by the number of bytes of the requested file (a 32-bit unsigned integer in network byte order - bytes B1 B2 B3 B4 in the

figure), then by the timestamp of the last file modification (Unix time, i.e. number of seconds since the start of epoch, represented as a 32-bit unsigned integer in network byte order - bytes T1 T2 T3 T4 in the figure) and then by the bytes of the requested file.

To obtain the timestamp of the last file modification of the file, refer to the syscalls *stat or* f*stat.*

The client can request more files by sending many GET commands. When it intends to terminate the communication it sends:

Q	U	I	T	CR	LF

(6 characters) and then it closes the communication channel.
In case of error (e.g. illegal command, non-existing file) the server always replies with:

-	E	R	R	CR	LF

(6 characters) and then it closes the communication channel with the client.
Save the client into a directory different from the one where the server is located. Name the client directory as the client program name.

Try to connect your client with the server included in the provided lab material, and the client included in the provided lab material with your server for testing interoperability (note that the executable files are provided for both 32bit and 64bit architectures. Files with the suffix _32 are compiled for and run on 32bit Linux systems. Files without that suffix are for 64bit systems. Labinf computers are 64bit systems). If fixes are needed in your client or server, make sure that your client and server can communicate correctly with each other after the modifications. Finally you should have a client and a server that can communicate with each other and that can interoperate with the client and the server provided in the lab material.

Try the transfer of a large binary file (100MB) and check that the received copy of the file is identical to the original one (using diff) and that the implementation you developed is efficient in transferring the file in terms of transfer time.

While a connection is active try to activate a second client against the same server.
Try to activate on the same node a second instance of the server on the same port.
Try to connect the client to a non-reachable address.
Try to connect the client to an existing address but on a port the server is not listening to.
Try to de-activate the server (by pressing ^C in its window) while a client is connected.

Exercise 2.4 (standard XDR data)

Modify the TCP client developed in the first laboratory (exercise 1.3) to send the two integer numbers read from the standard input and receive the reply (sum) from the server by using the XDR standard to represent data. It is not necessary to handle errors: the server always replies with a single integer value. Use the test server compiled in the exercise 1.1 by using the -x option.