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 file transfer TCP server) → OF COURSE GIT

FOR EXEMPTION, this exercise has to be submitted by May 21, 2018, 11:59am

SERVER

Develop a TCP sequential server (listening to the port specified as the first parameter of the command line, as a decimal integer) that, after having established a TCP connection with a client, accepts file transfer requests from the client and sends the requested files back to the client, following the protocol specified below. The files available for being sent by the server are the ones accessible in the server file system from the working directory of the server.

CLIENT

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). After having established the connection, the client requests the transfer of the files whose names are specified on the command line as third and subsequent parameters, and stores them locally in its working directory. After having transferred and saved locally a file, the client must print a message to the standard output about the performed file transfer, including the file name, followed by the file size (in bytes, as a decimal number) and timestamp of last modification (as a decimal number).

PROTOCOL

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

[CLIENT]

(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………

[SERVER]

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.

TIME STAMP

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

The client can request more files using the same TCP connection, by sending many GET commands, one after the other. When it intends to terminate the communication it sends:

Q	U	I	T	CR	LF

[CLIENT]

(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

[SERVER]

(6 characters) and then it closes the connection with the client.

RUNNING RULES

When implementing your client and server, use the directory structure included in the zip file provided with this text. After having unzipped the archive, you will find a directory named `lab2.3`, which includes a `source` folder with a nested `server1` subfolder where you will write the server and a `client1` subfolder where you will write the client. Skeleton empty files (one for the client and one for the server) are already present. Simply fill these files with your programs without moving them. You can use libraries of functions (e.g. the ones provided by Stevens). The C sources of such libraries must be copied into the `source` folder (do not put them into the `client1` or `server1` subdirectories, as such directories must contain only your own code!). Also, remember that if you want to include some of these files in your sources you have to specify the "`..`" path in the include directive).
Your code will be considered valid if it can be compiled with the following commands, issued from the `source` folder:

```
gcc -std=gnu99 -o server server1/*.c *.c -Iserver1 -lpthread -lm

gcc -std=gnu99 -o client client1/*.c *.c -Iclient1 -lpthread -lm
```

The `lab2.3` folder also contains a subfolder named `tools`, which includes some testing tools, among which you can find the executable files of a reference client and server that behave according to the specified protocol and that you can use for interoperability tests. Try to connect your client with the reference server, and the reference client with your server for testing interoperability (note that the executable files are provided for both 32bit and 64bit architectures. Files with the _32 suffix 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 reference client and server.

64 BIT SOLUTION IS PREFERRED

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.

ERROR CHECK

While a connection is active try to activate a second client against the same server.

1 - Try to activate on the same node a second instance of the server on the same port.

2 - Try to connect the client to a non-reachable address.

3 - Try to connect the client to an existing address but on a port the server is not listening to.

4 - Try to stop the server (by pressing ^C in its window) while a client is connected.

TESTING SCRIPT

When you have finished testing your client and server, you can use the test script provided in the `lab2.3` folder in order to perform a final check that your client and server conform to the essential requirements and pass the mandatory tests necessary for submission. In order to run the script, just give the following command from the `lab2.3` folder

```
./test.sh
```

The script will tell you if your solution is acceptable. If not, fix the errors and retry until you pass the tests. The same acceptance tests will be executed on our server when you will submit your solution. Additional aspects of your solution will be checked after submission closing, in order to decide about your exemption and to assign you a mark.

If you want to run the acceptance tests on a 32-bit system you have first to overwrite the 64-bit version executables under `tools` with their respective 32-bit versions. For example:

```
mv server_tcp_2.3_32 server_tcp_2.3
```

You will have to possibility to submit your solution to be considered for exemption **together with the solution of another exercise that will be assigned with lab3**. Instructions about how to submit will be given with lab3.

~~**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 exercise 1.1 by using the -x option.~~