Distributed Programming

Test on Network Programming of July 21, 2015   Time: 2 hours

The exam material is located in the folder "`exam_dp_jul2015`" within your home directory. For your convenience, the folder already contains a skeleton of the C files you have to write (in the "`source`" subfolder). It is **HIGHLY RECOMMENDED** that you start writing your code by filling these files **without moving them** and that you read carefully and completely this text before starting your work!
The exam consists of writing another version of the client and server for the lab exercise 2.3 (file transfer), using a slightly different protocol. In order to pass the exam, writing a server that behaves according to the specifications given in part 1 is mandatory.

Part 1 (mandatory to pass the exam, max 8 points)

Write a server (server1) that works according to the protocol specified in Lab exercise 2.3, but using a different format for messages, based on XDR. The XDR specifications that have to be used are in the `types.x` file, available in the `source` folder of the exam material. The messages from client to server (GET and QUIT) are XDR messages specified by the XDR type named `call_msg`, while the messages from server to client are composed of two parts. The first part is an XDR message specified by the XDR type named `response_msg`. The second part, which is present only if the first part is OK, has exactly the same format used in Lab exercise 2.3, i.e. a 32 bit unsigned integer in network byte order that specifies the number of bytes of the file to be transferred, followed by the bytes of the file.

The C file(s) of the server program must be written in the directory `$ROOT/source/server1`, where <u>$ROOT is the exam directory</u> (exam_dp_jul2015). If you have library files (e.g. the ones by Stevens) that you want to re-use for the next parts, you can put them in the directory `$ROOT/source` (remember that if you want to include some of these files in your sources you have to specify the path "`../source`"). Code generation from `types.x` must be done as specified in Further Instructions (common for all parts).
In order to test your server you can run the command

```
./test.sh
```

from the `$ROOT` directory. Note that this test program also runs other tests (for the next parts). It will indicate if at least the mandatory test has been passed.
Note that in order to pass the exam it is enough to implement a server1 that responds to a GET message by sending a correctly encoded response message.

Part 2 (max 6 points)

Write a client (named `client`) that behaves as the one developed in Lab exercise 2.3, but with the following differences:

1. The new client must use the XDR-based message formats specified in Part 1;
2. The new client must receive and use the following command line arguments: the first argument is the IPv4 address of the server, the second argument is the port number of the server, the third argument is the name of a file to be downloaded from the server.

3. After having finished the download of the file specified as third command line argument, the new client must send the QUIT command to the server and terminate.

The C file(s) of the new client program must be written under the directory `$ROOT/source/client`, where $ROOT is the exam directory (exam_dp_jul2015).

Part 3 (max 2 points)

Write a new version of the server developed in Part 1 (server2) that can serve at least 3 clients concurrently (for server1 there was no requirement about concurrency).
If server1 already had this capability, you can just copy server1 to server2.
The C file(s) of the server program must be written under the directory `$ROOT/source/server2`, where $ROOT is the exam directory (exam_dp_jul2015). The test command indicated for Part 1 will also try to test server2.

Further Instructions (common for all parts)

Your solutions will be considered valid **if and only if** they can be compiled by the following commands issued from the `source` folder (the skeletons that have been provided already compile with these commands; do not move them, just fill them!):

```
rpcgen -h types.x -o types.h

rpcgen -c types.x -o types.c

gcc -o socket_server1 server1/*.c *.c -Iserver1 -lpthread -lm

gcc -o socket_client client/*.c *.c -Iclient -lpthread -lm

gcc -o socket_server2 server2/*.c *.c -Iserver2 -lpthread -lm
```

Note that all the files that are necessary to compile your programs must be included in the source directory (e.g. it is possible to use files from the book by Stevens, but these files need to be included by you).
For your convenience, the test program also checks that your solutions can be compiled with the above commands.
All the produced source files (`server1`, `server2`, `client` and common files) must be included in a single zip archive created with the following bash command (run from the `exam_dp_jul2015` directory):

```
zip  socket.zip  source/client/*.[ch]  source/server[12]/*.[ch]  source/*.[ch]
```

The zip file with your solution must be left where it has been created by the zip command at the end of the test.

Important: Check that the zip file has been created correctly by extracting it to an empty directory, checking its contents, and checking that the compilation commands are executed with success (or that the test program works).

Warning: the last 10 minutes of the test MUST be used to prepare the zip archive and to check it (and fix any problems). If you fail to produce a valid zip file in the last 10 minutes your exam will be considered failed!

The evaluation of your work will be based on the provided tests but also other aspects of your program (e.g. robustness) will be evaluated. Then, passing all tests does not necessarily imply getting the highest score. When developing your code pay attention to making a good program, not just making a program that passes the tests provided here.