Distributed Programming

Test on Network Programming of July 7, 2016 Time: 2 hours 10 minutes

The exam material is located in the folder "`exam_dp_jul2016`" 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 extending the client and server for the lab exercise 3.4 (file transfer with XDR). The new client and server must use the following extended XDR definitions (these definitions are already available in the file xdr_types.x):

```
enum tagtype {
            GET = 0,
            OK   = 1,
            QUIT   = 2,
            ERR   = 3,
            GETMD5 = 4,
            OKMD5 = 5
};

struct file {
      opaque contents<>;
      unsigned int last_mod_time;
};

union message switch (tagtype tag) {
         case GET:
         case GETMD5:
            string filename<256>;
         case OK:
            struct file fdata;
         case QUIT:
            void;
         case ERR:
            void;
         case OKMD5:
            opaque MD5<16>;
 };
```

Part 1 (mandatory to pass the exam, max 7 points)
Write a server (server1) that works according to the protocol specified in Lab exercise 3.4 (for your convenience the text of Lab exercise 3.4 is included in this file at the end), but with the XDR definitions given above. When server1 receives a GETMD5 message, server1 has to open the specified file and compute the MD5 digest of the file contents. If the file can be opened and read without errors, server1 responds with a OKMD5 message including the computed MD5. If instead the file cannot be opened or read, server1 responds with the ERR message. For the computation of the MD5 of a file, server1 must use the utility functions provided in source form in the file md5.c. A usage example can be found in utility.c. Read carefully the comments available in utility.c.

The server must listen to the port specified as first argument on the command line, on all the available interfaces.
The C file(s) of the server program must be written in the directory `$ROOT/source/server1`, where `$ROOT` is the exam directory (exam_dp_jul2016). 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 "`..`").
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 tests have been passed.

Part 2 (max 4 points)
Write a client (named `client`) that behaves as the one developed in Lab exercise 3.4, but using the XDR definitions given above.
The new client must receive and use the following command line arguments. The list of arguments may start with the optional "-x" (if present, the client must use the XDR-based protocol), followed (in the order specified here) by the IPv4 address or hostname of the server, the port number of the server, and the name of a file whose contents or MD5 digest has to be received from the server. When the optional argument "-x" is present as the first argument, it is possible to have a fifth optional argument, "-md5". If this additional argument is present, the client must get the MD5 digest of the specified file, while if the argument is absent, the client must get the file contents. The IP address of the server is expressed in the standard (dotted decimal) notation, while the port number is expressed as decimal number. The file received by the client must be saved in the client working directory with the same name as the original file, while the MD5 digest of the file requested by the client must be saved by the client in its working directory in a file with name equal to the original filename with suffix ".md5". For example, if the file to be requested is named "f.txt", the MD5 of the file must be saved by the client in a file named "f.txt.md5".
The MD5 digest must be written in text form, one byte after the other, without spaces, using the formatter "%02x" of the printf function, and terminated by "\n". (The text format is the same printed by the md5sum command available through the shell)
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_jul2016). The test command indicated for Part 1 will also try to test Part 2.

Part 3 (max 5 points)
Write a new version of the server developed in Part 1 (named server2) that can serve clients concurrently (for server1 there was no requirement about concurrency). Moreover, server2 must accept an additional argument on the command line. This new argument (let us call it n) must be an integer number. If n is 0, server2 must create child processes on demand, and these processes must terminate after having served the request. If instead n is greater than 0 and no larger than 10, server2 must use pre-forking and n represents the number of processes that can serve client requests.

The C file(s) of server2 must be written under the directory `$ROOT/source/server2`, where `$ROOT` is the exam directory (exam_dp_jul2016). The test command indicated for Part 1 will also try to test server2.

Further Instructions (common for all parts)
In order to pass the exam it is enough to implement a server1 that sends a correctly encoded MD5 digest to the client, or to implement a server1 and a client that can interact with each other as expected.
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 -o xdr_types.h xdr_types.x

rpcgen -c -o xdr_types.c xdr_types.x

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_jul2016` directory):

```
./makezip.sh
```

At the end of the exam, the zip file with your solution must be left where it has been created by the zip command.

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.

Exercise 3.4 (XDR-based file transfer)

Modify the client and server developed so far so that they accept an optional –x extra argument before the other arguments (i.e. as first argument). If the argument is present in the command line, a similar protocol, but based on XDR, has to be used. According to the new protocol, the messages from client to server and from server to client are all represented by the XDR `message` type defined as follows:

```
enum tagtype {
            GET = 0,
            OK   = 1,
            QUIT   = 2,
            ERR   = 3
};

struct file {
      opaque contents<>;
      unsigned int last_mod_time;
};

union message switch (tagtype tag) {
         case GET:
            string filename<256>;
         case OK:
            struct file fdata;
         case QUIT:
            void;
         case ERR:
            void;
 };
```

Cross test your client and server for interoperability by using the client and server executable files (for linux) provided with this text.