Distributed Programming

Test on Network Programming of September 14, 2016   Time: 2 hours 10 minutes

The exam material is located in the folder "`exam_dp_sep2016`" 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 developing a different version of the client and server developed for lab exercise 1.4 (client-server UDP base, the text of the lab is available at the end of this text for your convenience). The new client and server must use the following request-response protocol for communication:

- The **request** (sent by the client) is a UDP datagram that contains a sequence of ASCII characters, to be interpreted as a filename, for which the client wants to get some information (whether the file with that filename is available in the server and what is its size in bytes). This sequence of characters has no termination character (its end is the end of the datagram), and its maximum length is 256 characters. Only alphanumeric characters, the underscore and the dot are admitted as part of the filename.
- The **response** (sent by the server upon receiving a request) is a UDP datagram that contains the following sequence of fields (exactly in this order):
  - a 4-bytes integer code, represented in network byte order, with the following possible values:
    - 0 in case of error in the request (request too long or including non-admitted characters)
    - 1 in case the requested file does not exist in the server working directory
    - 2 in case the requested file exists in the server working directory
  - a copy of the sequence of ASCII characters sent by the client in the request (limited to the first 256 characters in case of request that exceeds this limit)
  - another 4-bytes integer, represented in network byte order, which has to be included only if the first 4-bytes integer of the response is 2 and whose value is the length of the file with the requested filename in bytes.

  Note that the second field (copy of the filename) has no delimiter. Its length can be obtained by difference (length of the datagram minus 4 or minus 8, according to whether the third field exists or not).
- Clients are supposed to silently discard response messages that have erroneous encodings or that do not match the filename that the client requested.

Part 1 (mandatory to pass the exam, max 7 points)
Write a server (server1) that works according to the protocol specified above. The server must listen to the port specified as first (and only) argument on the command line, on all the available interfaces.
Each time the server receives a request from a client, before sending the response, it must output (to the standard output) a line of text with the following format:

    <prefix> <client-ip> <client-port> <response-code> <filename>

where:

- <prefix> is the fixed 7-character ASCII string "REQUEST"
- <client-ip> is the IP address of the client that made the reques,t in dotted decimal
- <client-port> is the client port number, written as a decimal number
- <response-code> is the response code that the server includes in the response, written as a decimal number
- <filename> is the file name requested by the client (a sequence of ASCII characters), to be omitted in case of bad request (i.e. one that produces response code 0)

Each pair of adjacent fields in this text line must be separated by one space and the line must be terminated as usual by \n.
Important: these lines of text should be the only outputs written by server1 to the standard output.
Important: flush the output as soon as the output has been produced.
Suggestion: the file size can be obtained by calling the stat() function. See man for details.
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_sep2016). 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 5 points)
Write a client (named `client`) that behaves as the one developed in Lab exercise 2.1 (you can find a copy at the end of this text for your convenience), but using the protocol specified above. Differently from Lab exercise 2.1, this client must try a single re-transmission before giving up, while the timeout period is 3 seconds, as in Lab exercise 2.1. In case the client receives a response that is not correctly formatted or that does not match the filename that the client requested, the client must silently discard it, and wait for another 3 seconds.
The client must receive and use the following command line arguments (exactly in the order specified here): the IPv4 address or hostname of the server, the port number of the server, and the filename to be sent to the server. The IP address of the server is expressed in the standard (dotted decimal) notation, while the port number is expressed as decimal number. The client must write to the standard output a line of text with the following format:

<prefix> <response-code> <size-of-file>

where:

- <prefix> is the fixed 8-character ASCII string "RESPONSE"
- <response-code> is the response code received as the first field of the response or -1 in case no response has been received. This number is expressed in decimal notation.
- <size-of-file> is the size of the file in bytes (absent if it was not received from the server)

Each pair of adjacent fields in this text line must be separated by one space and the line must be terminated as usual by \n.

Important: these lines of text should be the only outputs written by the client to the standard output.
Important: flush the output as soon as the output has been produced.

The C file(s) of the 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 4 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) using pre-forking. The server must receive the number of processes that can serve client requests concurrently as the second argument on the command line (after the port number).
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 response 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!):

```
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_sep2016` 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 1.4 (client-server UDP base)

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

Develop a 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 the server is NOT listening to
- client sends a datagram to an unreachable address (es. 10.0.0.1)

In the provided lab material you can find the executable version of a client and of a server that behave as expected (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).

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, in order to test interoperability of your code. 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.

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