Distributed Programming I (03MQPOV)

Laboratory exercise n.3

Exercise 3.1 (concurrent TCP server)

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

Develop a concurrent TCP 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 same protocol used in Lab exercise 2.3. The server must create processes on demand (a new process for each new TCP connection).

When developing this new server, use the same directory structure already used for the client and server of Lab exercise 2.3 (folder `lab2.3`). Copy the files named `test2.sh` and `makezip.sh` (provided in the `lab3.1` folder of the zip archive where you have found the material for Lab 3) to folder `lab2.3` and, inside `lab2.3/source`, create a new directory named `server2` (it will be a sibling of `server1` and `client1`). Write the code of your new server inside this directory (you can use the libraries you already have placed in the `source` folder).

Your code of server2 will be considered valid if it can be compiled with the following command, issued from the `source` folder:

```
gcc -std=gnu99 -o server server2/*.c *.c -Iserver2 -lpthread -lm
```

By using the client developed in Lab exercise 2.3 (client1), try to run more clients concurrently and verify that each one can establish a connection and transfer files.

Repeat the same tests you made for the server of Lab exercise 2.3: in particular, try to connect your client and server, and the reference client with your server for testing interoperability.

When you have finished testing your new server, you can run the acceptance tests by running the following command from the `lab2.3` folder:

```
./test2.sh
```

The script will tell you if the mandatory tests have been passed or not.

In order to submit your solution, run the command

```
./makezip.sh
```

from the same folder. This script will create a zip file named `lab_2.3_3.1.zip` with your solutions. Finally, go to the portal https://pad.polito.it:8080/enginframe/index.html and submit this zip file (you will receive the credentials for logging into the portal in the next days).

Optional

Modify the server so that it serves no more than 3 clients concurrently.

Then, try to activate four clients at the same time and verify that just three of them are able to get the service, i.e., receive the file. You may notice that clients connect to the server anyway using the TCP 3-way handshake even if the server has not yet called the accept() function. This is a standard behaviour of the Linux kernel to improve the server response time. However, until the server has not called accept() it cannot use the connection, i.e., commands are not received. In case the server does not call accept() for a while, the connection opened by the kernel will be automatically closed.

Try to forcefully terminate a client (by pressing CTRL+C in its window) and verify that the (parent) server is still alive and able to reply to other clients.
This optional part is NOT to be submitted.

Exercise 3.2 (TCP client with multiplexing)

Modify the client of Lab exercise 2.3 to handle an interactive user interface that accepts the following commands:

- **GET file** (requests to perform the GET of the specified file)
- **Q** (requests to close the connection with the server with the QUIT command after that an eventual file transfer is terminated)
- **A** (requests to forcefully terminate the server connection, even by interrupting active file transfers)

The user interface must be always active in order to allow "read-ahead": this means that it must process inputs even when a file transfer is ongoing.

Exercise 3.3 (TCP server with pre-forking)

Create another version of the solution of exercise 3.1 in which the processes accepting file transfer requests from clients are created as the server is created (pre-forking), and remain waiting until a client connects to them. If a client closes the connection, the process handling that client must move on to serve a new waiting client, if present, or wait for new clients.
Make configurable the number of child processes that are launched as the server starts through a command-line parameter, with a maximum of 10 children.
Verify the proper functioning of the server by launching it with 2 children and connecting 3 clients to it, each one performing a file request. Close one of the three clients and verify that the client waiting for a connection is immediately served.
Verify that the server is able to also handle the case in which a connected client crashes (for example, if closed through the kill command): the server must be able to detect this condition and start serving the next waiting client.
Finally, modify your server so that if a connection with a client remains inactive for 2 minutes the server closes it, so that the process that was waiting for the client input becomes ready to manage another client.

Exercise 3.4 (XDR-based file transfer)

Create another version of 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 the Lab material.