Distributed Programming I

A.Y. 2012-2013, Laboratory exercise n.3

Exercise 3.1 (concurrent TCP server)

Develop a TCP server (listening to the port number specified as first command-line parameter) that is able to accept file transfer requests from clients and sends the requested file.
The server must implement the same protocol used in exercise 2.3 and it must be of concurrent type, by activating a maximum of three contemporary childs.
By using the client developed in exercise 2.4, try to activate four clients at the same time and verify that just three of them are able to perform the connection.
Try to forcefully terminate a client (by pressing CTRL+C in its window) and verify that the (father) Server is still alive and able to reply to other clients.

Exercise 3.2 (TCP client with multiplexing)

Modify the client of 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 ingoing.

Exercise 3.3 (TCP server with pre-forking)

Modify the exercise 3.1 so that processes accepting file transfer requests from clients are created as the server is created (pre-forking), and remains 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.
Make configurable the number of children 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 client to it, everyone 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 (as example, if closed through the kill command): the server must be able to detect this condition and make itself available for an eventual next waiting client.