Assignment 2:

Distributed Programming with Sockets

Miguel Morales Expósito (mml212, SN: 2618012)

1. Environment

The programs have been run in a virtual machine with Ubuntu 18.04.1 LTS. Source files have been compiled using gcc 7.3.0.

To compile the program execute the command *make* in the *A2* folder.

2. Key-value store

Client execution: ./client <server address> <port> [put <key> <value>] [get <key>]

Server execution: ./serv(1|2|3|4) <port>

Serv1

Client: The program makes a string with the format of the protocol specified on the assignment based on the parameters passed to the program. Instead of number 103 for get I attach a 'g' in the string because it is computed the same way as 103. I do the same for the code 112, I attach the letter 'p'.

The client creates a sockets that connect to the server, sends the string, waits for a reply if there was any "get" query, closes the sockets and finishes.

The function *handle_reply* gets the string reply from the server and prints new line if received 'n' (code 110) or the value related to the specified key if 'f' is received (code 102).

To write the bytes on the sockets, the function *writen* is used [1].

Server: It creates a socket, bind it and listen from it. When it accepts a new connection, it reads the bytes from that socket and process the message from the client with the function *handle_message.*

The structure decided to keep the key-value pairs is an array of structs pair. Struct pair contain an array of char for the key and an array of char for the value. There is a variable size that holds the current size of the array. Every time a "put" command is received, the pair is inserted in position size in the array and after that size is incremented by 1.

This server is iterative so it accepts a connection from one client, reads from it sockets, process the message, close the socket and wait for accepting a new connection.

Serv2

Client: The implementation of the client is the same as in serv1.

Server: This server differs from serv1 when receiving a connection. When a new connection is accepted, the parents forks and it's the child who process the request. Then the parent closes the socket.

Since the children are the one accessing the dictionary, in this version of the server, the array of pairs is declared as shared memory between the parent and the children. After a child dies the memory is detached from it.

In order to avoid race conditions, the pair struct is upgraded in this version and it also contains a semaphore that is locked down when a child is updating one of the values from the dictionary. There is also a semaphore that is locked down every time that a new pair is added. This is done because there can be race conditions when reading and incrementing the value of *size* which contains the length of the array.

Serv3

Client: The implementation of the client is the same as in serv1.

Server: This server uses preforking. The parent make as many children as indicated by the second parameter passed to it. Then, every children is ready to accept new connections. In order to increase performance, a mutex lock is added when accepting connections in every child.

Serv4

Client: The implementation of the client is the same as in serv1.

Server. This server uses threading. The main process creates a thread every time that a new connection is accepted. The thread executes the function *treat_request* that reads the message from the client, process it, sends a reply to the client and closes the socket.

3. A talk program

The chatting program connects to users one in server mode and another in client mode with sockets. Each user creates a child that is continuously reading one byte from the socket. The parent every time that the user inputs a character sends it to the other user through the socket. In order to be able to send a character without pressing return carriage, the program calls system ("/bin/stty raw").

The counterpart of using system ("/bin/stty raw") is that new lines are not sent as new lines.

The implementation of both users disconnecting when pressing control + C could not be done.

Bibliography

[1] *Andrew M. Rudoff, Bill Fenner, W. Richard Steven.* UNIX Network Programming: Sockets Introduction.