Assignment 3: TCP Echo Server using fork()

Computer Networks (CS-UH 3012) - Spring 2022

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

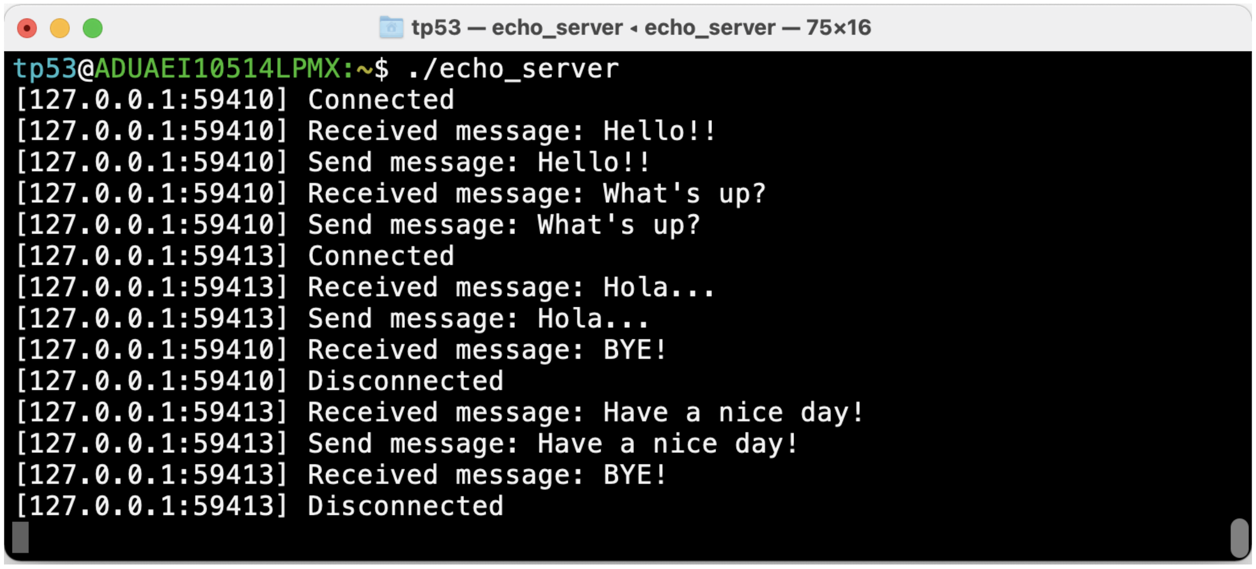
1. Any document and program code that you submit must be fully written by yourself.
2. You can discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. In other words, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit.
3. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution.
4. You are not allowed to possess solutions by someone from a different year or section, by someone from another university, or code from the Internet, etc.
5. There is never a valid reason to share your code with fellow students.
6. There is no valid reason to publish your code online in any form.
7. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g. if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi. More details can be found at: https://students.nyuad.nyu.edu/academics/registration/academic-policies/academic-integrity/

2 Assignment Goal

The goal of this assignment is to implement a TCP client and server using socket programming and the `fork()` function in C. This will give you practice in working with sockets, processes, and getting familiar with the TCP transport protocol.

3 Task Description

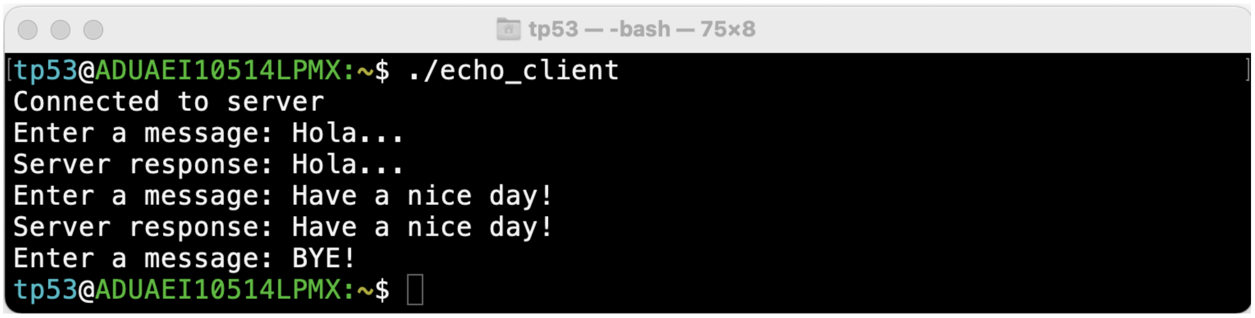
The task of this assignment is to write two C programs that exchange messages using the TCP protocol. The task consists of two implementations: a server and a client.
The TCP server program implementation opens a TCP socket (`SOCK_STREAM`), listens on port 6000 for incoming connections and after establishing a connection with a client, replies to the client with the message received from that client. The server should be able to support multiple simultaneously connected clients and hence use the `fork()` function to spawn child processes for every connected client. The server should terminate the child process when a client sends a message with the content "BYE!" or if the server receives a message of length 0 (this usually occurs when a client closes the socket or if the client has been terminated using Ctrl+C).
The server implementation should print a message for every incoming and outgoing message, including the IP address and port number of the connected client, see Figure 1. The server should also print a message when a client connects or disconnects.



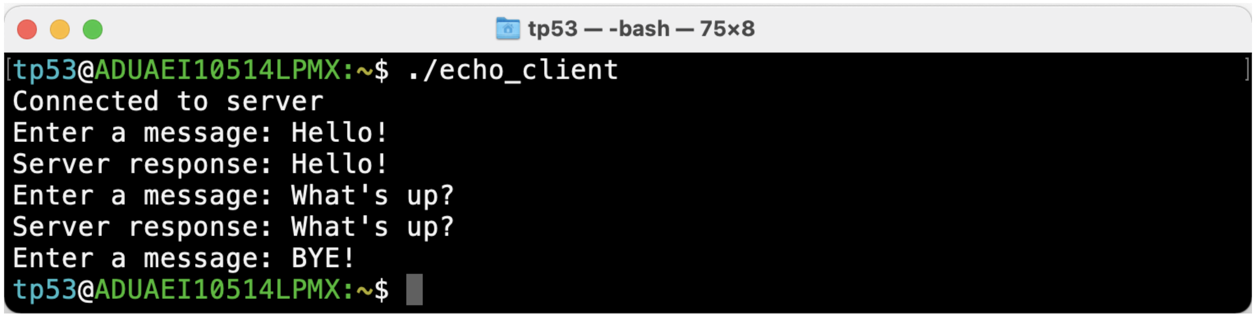
```
tp53@ADUAEI10514LPMX:~$ ./echo_server
[127.0.0.1:59410] Connected
[127.0.0.1:59410] Received message: Hello!!
[127.0.0.1:59410] Send message: Hello!!
[127.0.0.1:59410] Received message: What's up?
[127.0.0.1:59410] Send message: What's up?
[127.0.0.1:59413] Connected
[127.0.0.1:59413] Received message: Hola...
[127.0.0.1:59413] Send message: Hola...
[127.0.0.1:59410] Received message: BYE!
[127.0.0.1:59410] Disconnected
[127.0.0.1:59413] Received message: Have a nice day!
[127.0.0.1:59413] Send message: Have a nice day!
[127.0.0.1:59413] Received message: BYE!
[127.0.0.1:59413] Disconnected
```

Figure 1: Example program output of the TCP echo server with two connected clients

The TCP client program implementation opens a TCP socket, connects to the server on port 6000 and asks the user to input a message. After the message is entered and the user hits the return key, the client sends the message to the server and waits for the reply, see Figure 2. The client should terminate after the user enters the message "BYE!".



```
tp53@ADUAEI10514LPMX:~$ ./echo_client
Connected to server
Enter a message: Hola...
Server response: Hola...
Enter a message: Have a nice day!
Server response: Have a nice day!
Enter a message: BYE!
tp53@ADUAEI10514LPMX:~$ 
```



```
tp53@ADUAEI10514LPMX:~$ ./echo_client
Connected to server
Enter a message: Hello!
Server response: Hello!
Enter a message: What's up?
Server response: What's up?
Enter a message: BYE!
tp53@ADUAEI10514LPMX:~$ 
```

Figure 2: Example program output of two TCP clients connected to the TCP echo server

4 Implementation Hints

For the client to server communication, you can use the localhost IP address 127.0.0.1 which allows you to run the client(s) and server in different terminals on the same computer. When creating the `sockaddr_in srvaddr` in the client, you can use the following code snippet to set the destination IP address of the server to 127.0.0.1:

```
    srvaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

The maximum message length the client and server support is limited to 100 bytes. Also, the server is supposed to run indefinitely, hence you do not need to implement the closing of the socket when the server is terminated from the terminal using Ctrl+C. Instead, you can use the following code snippet to avoid binding issues of the socket:

```
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int));
```

When using the `fork()` function to spawn a child process, make sure the server code that handles the receive and reply is only executed on the child process (process ID == 0). Please refer to the manual of the `fork()` function.

5 Grading

Description	**Score (/4)**
Successful compiling of the programs using ONE Makefile	0.5
Successful client-server communication using TCP sockets on port 6000	1
Using fork() to handle multiple connected clients	1
Handling disconnections of the client (and terminating the child process)	0.5
Printing of events, as depicted in Figure 1 and 2	0.5
Usage of meaningful comments	0.5

6 Submission Details and Policy

Submission Deadline: The deadline of this assignment is after 6 days of its release via Brightspace. No extensions will be given.

Submission Format and System: You can directly submit your files (both C files and ONE Makefile) as a zip file on Brightspace (https://brightspace.nyu.edu/). Due to technical limitations, submissions via email are not accepted.

Late Submissions: Late submissions will be penalized by 10% per 24 hours, with a maximum of 3 days late. In case of a late submission, please upload your zip file to Brightspace and inform the TA and the professor.