Assignment 5

Task Details

Implement a socket-based server. This server can accept connections from clients and echo whatever message the client sends to it. The server also needs to log the address information of each client that it accepts.

Requirement

1. Your submission should produce a program “cs392_echoserver” and a program “cs392_echoclient”.
    a. The “cs392_echoserver” takes one argument, which is the port number you want the server to use
    b. The “cs392_echoclient” takes two arguments. The first one is the IP address of the server and the second one is the port number of the server.

2. For the “cs392_echoserver” program, you need to create two source files “cs392_echoserver.c” and “cs392_log.c”, and a header file “cs392_log.h”
    a. The “cs392_echoserver.c” will implement the main logic
        i. It will need to create a socket that uses the IPv4 domain and the SOCK_STREAM type (protocol can be 0)
        ii. It will need to bind an address to the above socket. As explained above, the port number is provided by the first argument to the main function.
        iii. It need to listen to the socket after binding the address. Please restrict that at most 5 clients can be connected to this server.
        iv. It will then need to want for connections via the “accept” interface. Please make sure this interface will block until a connection request comes in.
        v. When a connection is accepted, it will need to log the address information about the client (including both IP and port number) to a log file called “cs392_socket.log”. You need to use a function called “cs392_socket_log” from “cs392_log.c” to do the logging. After the logging, the server will need to wait for message from the client.
        vi. Whenever the server receives a message from the connected client, it will need to send the message back to the client.
        vii. After sending the message back, the server will close the socket that represents the client. This basically means disconnecting the client. Yes, you are only required to receive/echo ONE message from/to a client.
        viii. After the server disconnects a client, it will need to go back and wait for a new connection by using “accept” again (after a new connection comes, you need to repeat the above echoing process).

b. The “cs392_log.c” just need to implements the function “cs392_socket_log”
    i. This function will write the address information of a client to “cs392_socket.log”.
        1. You need to write to the file in an appending manner. If the log file does not exist, you need to create a new one with the same name.
        2. You have the freedom to design the interface to pass that address information.
c. You need to declare the prototype of “cs392_socket_log” in “cs392_log.h”. You should only included the header file in “cs392_echoserver.c” (to tell the compiler the prototype of “cs392_socket_log” when it’s used in “cs392_echoserver.c”).

3. For the “cs392_echoclient” program, you need to create one source file “cs392_echoclient.c”
    a. It will need to create a socket that uses the IPv4 domain and the SOCK_STREAM type (protocol can be 0)
    b. It will need to connect to the server using the “connect” interface. As above mentioned, the IP address and the port number of the server are provided by arguments to the main function.
    c. After the connection is established, it will need to ask a message from the standard input (you can assume it has less than 1024 bytes) and then send the message to the server.
    d. After sending the message, it will need to wait for the echo from the server.
    e. After receiving the echo, it will print the message to standard output, close the socket, and exit the program.

4. The “Makefile” builds the above source code files into two programs “cs392_echoserver” and “cs392_echoclient”
    a. You are expected to include explicit rules to compile the “.c” file into object files (“.o” files)
    b. You are expected to include explicit rule to link the objects files to generate the two programs
        i. Hint: You can include indepent rules to produce multiple programs in the same Makefile.
    c. You need to include a “clean” target in your Makefile, which, if executed, can remove the object files and the two programs.
    d. DO NOT copy the Makefile from midterm and you SHOULD NOT build shared libraries.

2. Zip the .c files, .h files, and the Makefile into a cs392_ass5.zip and submit the .zip file only

How to test:

--- In one bash session, start the server program by running:

$ ./cs392_echoserver SERVERPORT
Note: Here “SERVERPORT” is a number that you can randomly pick (but make sure it’s larger than 1024).

--- In another bash session, start the client program by running:

$ ./cs392_echoclient 127.0.0.1 SERVERPORT

- When you are asked for inputs by the client, type something (readable) and see if the server echos it back.
- Note: Here “SERVERPORT” should be the same as the one provided to the server program.

Try to run the client multiple times and see if it always works

- Note: you should only start the server once, no matter how many times you start the client.

Grading Policy:

- Cannot compile or run: Will not be graded !!!
- No statement of “I pledge my honor that I have abided by the Stevens Honor System.” as comment in the beginning of your code: Will not be graded !!!
- Place the honor statement in the beginning of the “.c” files, the “.h” file, and the Makefile
- Late submissions without pre-approval will not be accepted !!!
- This assignment must be done individually. Group work will lead to 0 point !!!

- The server program can be started correctly, and it will listen to the PORT number as specified by the first argument (+10).
    - The TA will randomly pick a PORT number to test your server program (which will be larger than 1024)

- Pass the test of one client (+10; 5 clients will be sequentially started to do the test, so 50 in total for this part)
    - The process of testing a client is:
        - Your client program will be started with “127.0.0.1” and the PORT number used by the server
        - The TA randomly inputs a message to the client
        - The TA will check if the server correctly echos the message back
    - No use of socket communication will lead to 0 points in this part

- Correct use of socket operations (+8)
    - This means using the interfaces correctly, including “socket”, “bind”, “listen”, “accept”, “connection”, “close”, “recv/read (or other message sending interfaces)”, “send/write (or other message receiving interfaces)” (+1 for correct use of one interface)

- Correctly implement the log file (+10)
    - Correctly manage the file, including creation if not exit, opening, appending, and closing (+5)
    - Correctly log the address of the client (IP + PORT) (+5)

- Correct error checking and handling (+10)
    - Please do that for memory operations, file operations, and socket operations.
    - Missing of one error checking and handling (-2), until all 10 points for this part are deducted.

- Correct Makefile (with cleanup rules) (+12). **If you copy my midterm Makefile, YOU WILL GET 0 POINT HERE, even if you change the file names!!!**
    - Explicit rules to compile source files into object files (+6)
    - Explicit rules to link object files into two programs (+4)
    - Clean up rules (2)

In this test, you will need to run the server and the client at the same time. As the server will block the current bash session, you will need two bash sessions to run them. In the following, I provide a tutorial of using TMUX to create two bash sessions in parallel:

Tutorial of using TMUX to run multiple bash sessions in parallel

Step 1: Check if you have “tmux” by running

$ tmux

If you get a notification that “tmux” is not installed, please run:
$ sudo apt-get install tmux

Step 2: Start tmux (if you succeed in step 1, skip this step) by running:

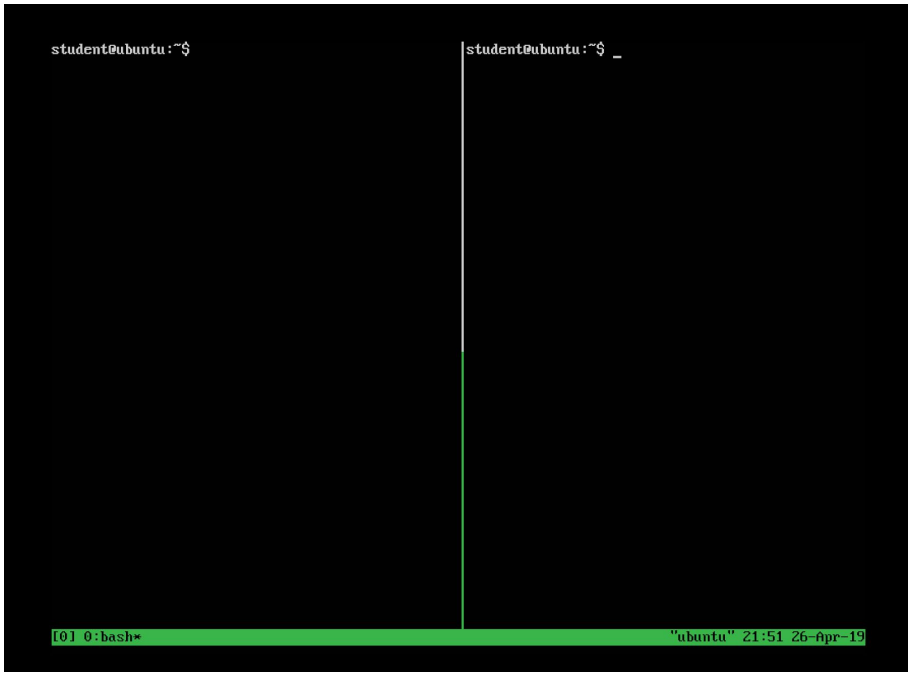
$ tmux

You will get a window like the following:

Step 3: Create another bash session by

First typing: CTRL + b (at the same time)

And then typing: %
You will see a window like this:



And you will be controlling the session that has a flashing cursor. You can run whatever command you want in that session (such as starting a socket server).

Step 4: To move to the other bash session, you can:

First type: CTRL + b (at the same time)

And then type: the left key (“<-”) or right key (“->”) (the “left” key will move you to the left session and the “right” key will move you to the right session).

Once you moved to the other session, you can also run whatever commands you want (such as starting the socket client).