Assignment 2 CPSC3600 50 points

UDPchat app

In this *individual* assignment you will implement a command line UDPchat application. It will allow one client to communicate with the server at a time. You will have one UDP server and one UDP client. The starter code from the textbook is sufficient for you to complete this assignment - instead of one UDP client and one UDP server exchanging one message, you will exchange several messages that will appear on both client's and server's screens. You have completed Activity4 in class, so you already have the starter files and you know how to run them. Please review the chapter in the TCP sockets programming textbook for detailed explanation of the code.

What to submit

You will submit a tarball without any subdirectories that contains two files: a file named *udpchat.c* and a simple *makefile* with *make* and *make clean*. Your C code will contain the code to run both the server and the client.

Implementation/Execution

1. Open a terminal on two different SoC machines, one will be your server and the other will be your client. Your server should be able to communicate with several clients one after the other.

2. Find the IP address of the server.

3. The same executable named *udpchat* will run in both terminals. The difference between the client and the server is the number of command line arguments. To run the client you will pass an IP address of the server and some large port number, such as 88888 of 55555, etc. To run the server you will only pass the (same) port number. So, based on the number of CLA you can either call a client function, or a server function. There is more than one correct way of doing this assignment, and solutions will vary.

4. Chat messages will appear on both screens, so both participants, who we assume are separated geographically, can see them on their corresponding screens. "Connection and termination" messages appear on the server's screen only.

5. When conversation is over, the client will send a goodbye message to the server. Server will send its own goodbuy to that user, and print the message that conversation is over. Since this is a connectionless protocol, there is no need to establish or tear down the connection. Please see chapter for details.

Sample output:

[client terminal]
$$ udpchat 130.124.23.45 55555
user: Hello friend, how are you?
server: Very well, thank you!
user: Do you like ice-cream?
server: Strawberry, with gummy bears on top, and you?
user: I love chocolate mint.
server: Sounds good.
user: Goodbye!
server: Goodbye!

[server terminal]
$$ udpchat 55555
Talking to client 130.124.23.41 (your IP will be different)
    user: Hello server, how are you?
    server: Very well, thank you!
    user: Do you like ice-cream?
    server: Strawberry with gummy bears on top, and you?
    user: I love chocolate mint.
    server: Sounds good.
    user: Goodbye!
    server: Goodbye!
Conversation with 130.124.23.41 ended

At this point conversation is over. Notice that "Goodbye!" will be on the line by itself, or can be appended to the cleint's message. If another client starts conversation with the server, server will print the second conversation below the first one. You can them terminate your server by Ctrl^C.

[server terminal]
$$ udpchat 55555
Talking to client 130.124.23.41 (your IP will be different)
    user: Hello server, how are you?
    server: Very well, thank you!
    user: Do you like ice-cream?
    server: Strawberry with gummy bears on top, and you?
    user: I love chocolate mint.
    server: Sounds good.
    user: Goodbye!
    server: Goodbye!
Conversation with 130.124.23.41 ended

Talking to client 130.124.23.42 (IP will be different)
    user: Hey, what is your favorite tv show?
    server: CSI, and yours? !
    user: NCIS. I love their characters. Goodbye!
    server: Goodbye!
Conversation with 130.124.23.42 ended

Tips and Hints:
1. Only add a few lines of code at a time and compile.
2. Exchange one message first, and then build on that.
3. Use commenting out and print statements as debugging tools.
4. There is more than one correct solution, details were not provided, so you can design the program the way you like it. Output has to match the example shown above.
5. Program must run on the SoC machines.

As usually, this is an individual assignment and you can only consult your TAs and your teacher.