Purpose: This assignment requires an in-depth understanding of a single process concurrent connection-oriented client-server program (Chapter 13). You will build a multi-user chat server. You will use a concurrent connection-oriented single process server (TCP/select).

THE CHAT SERVER

The chat server will listen on your well-known TCP port. The server will accept up to as many client connections at a time as is allowed by the number of file descriptors. Whenever it receives a message from a client it will echo that message to all other clients currently connected (but not to the original sender). If any error occurs on a socket connected to the client, the server will close that socket and clear that bit (from the `afds`); it will not error exit.

Details:

The server will use select so that it gets both incoming messages and incoming connection requests. This is the same as Comer's code.

The server will echo the incoming messages using a for loop. This for loop should contain an if statement that carefully skips the master socket, the socket of the client that sent the message and all closed sockets.

I recommend you remove Comer's call (`echo(fd)`) and replace the `if` statement with a `for` loop.

If you try to do this assignment by modifying Comer's `echo` procedure itself, you will need modify `echo` so that it is passed references to `afds`, `rfds`, `nfds`, `fd`. This gets really messy, it's easier to put all the code into the main program.

So, if you are following my recommendation, `if(echo(fd)` gets replaced by a `read` (`recv`) followed by an `for(int fd2=0...`. Associated with the read the read there needs to be an `if` statements. If the read shows an error or an eof; there is nothing to echo to the other clients and the socket is no longer in use, so the server will close the read socket, clear the read bit from the `afds` and skip the `fd2` loop. (You may use the C `continue` statement to do this.)

Inside the `fd2` loop There are two `if` statements. The first has a fairly complex boolean condition regarding which other sockets to write to; it skips the master socket, skips the `fd`, and allows only clients found in the `afds`. If a socket meets all these conditions, it uses a to echo the message to that client. (That isit writes (`send` on the `fd2` socket.)

You should be writting exactly what you read. Writting exactly what you read is precisely what Comer's code does, except, he writes it back to the sender, you loop is writting it to every client except the sender.

You should place your second `if` statement on the `send`. If `send` returns an error (`<0`); close the send socket, clear the send socket from the `afds`; and keep looping. That is, if one client goes bad, we eliminate that client from our list, and keep looping until we have sent the message to all other valid clients.

I recommend you start with the copy of `TCPmechod.c` found in the `comer_examples` directory. It has been modified (from the version in the book) to use `send` and `recv`.

This whole code is less than 10 lines long, but it is fairly complex; watch your curly braces to be sure you get the correct nestings.

Submit: The a copy of the source code for the your server. In addition, the source code for the server must be placed in your home directory in a file named `chatd.c`.

THE CHAT CLIENT

When started, the chat client opens a TCP connection to the server. Whatever the client sees the user type will be sent to the server (i.e., the client gets a line from the keyboard, then sends that line to the server). Whatever the client receives from the server will be printed on the screen (i.e., the client receives from the socket, then prints what it has recv'd to the screen). Typing control-D (EOF from keyboard) will cause the client to shutdown the socket, recv from the socket until it gets an end of file from the server.

A chat client is provided for you as `shells/chatc`. This is an executable file; I am not supplying you the source. You will build a client of your own as the next assignment.