ECE 2220: System Programming Concepts **Fall 2017**
Machine Problem 8 **Due: 11:59 pm, Thursday, December 7**

In this lab, each student is to write a **single** program called **`lab8.c`** which implements a chat program. The student specifically explores:

- Utilizing the ncurses library to draw character-based graphics.
- Exploring network programming through use of sockets.
- Learning about event-based programs that wait for and respond to input from the keyboard and the network.

The objective of the program is to extend the provided template **`chat.c`** to add character-based graphics to a two-person chat program. The provided chat program, **`chat.c`**, opens a network socket on a specific port to a remote machine. When an interrupt is received from the keyboard, the program collects the line of text and sends it to a socket and thus to the remote machine. When an interrupt is received from the socket, the program collects the text from the remote machine and displays it to the screen.

You are to devise a scheme to display each new line of text, and scroll the remaining text. Use the lower half of the screen to display the text that has been typed on the keyboard. The space on the top of the screen displays the text received from the remote host. As each new line of text is received, the previous lines should scroll up to fill the available space in the terminal.

The provided program, **`chat_curses.c`**, provides a very rough initial design for the character-based graphics. The program sets up the curses system and puts a dividing line into the middle of the screen. The keyboard input is line-buffered, and an interrupt is given to your program only when the user types the Enter key. Add similar capabilities to the **`chat.c`** program. While a line of text typed by the local user is collected, the program must **not** block or delay displaying text from remote user. Your design must have some type of information line with a format that makes it stand out visually from the text. The information line must at a minimum contain the host name of the remote host.

When the user types **`quit`**, the program should send "quit" as the final message to the remote host and then end the program.

Notes

1. **Examples of all the techniques needed for this machine problem are found in the lecture notes for Chapter 8.** See the **`hello.c`** series of files. Remember to include the **`-lncurses`** flag with **`gcc`** to link in the library. If the library is not available use the Ubuntu Software Center to add the package **`libncurses5-dev`**

2. Terminal problems: While working with the ncurses library if the output to the screen no longer works correctly due to a bug, try typing the Unix command **`stty sane`** in the window to reset to the default values. Make sure your program ends with a call to **`endwin()`** to reset the terminal display.

3. Network problems: Due to firewalls and NAT servers, the network packets may not be delivered across the internet. Two simple approaches can be used to quickly test your code. First, you can use the hostname **`localhost`** to have the OS simply take packets sent to the socket and return them back to the same socket. Your program will operate correctly even in this situation. Simply use "**`./lab8 localhost`**" to test using the full network software. To test with a network connection,

log into two of the CES **apolloXX.ces.clemson.edu** machines. Since all the **apolloXX** machines are on the same local area network, all packets are delivered with very high probability.

4. Host problems: To use sockets, your program must bind to a specific port number. Only one program on a host can bind to a port at a time. Thus, if another program has already bound to the port number, your program will get a binding error upon startup. This might happen, for example, if another student from our class is logged into the same host and has already run the chat program. One simple solution is to pick a new port number. Just change the **MYPORT** number in the code to a nearby number (e.g., any number within the range plus or minus approximately 100). Note: to chat with a program on a remote host both programs must use the same port number.

5. Extensions: you are encouraged to modify the design of the chat display. For example, you could display the text from the remote user with a justification to the left side of the screen, and the local text with a right justification. Your code must still scroll prior lines, and the status line must be displayed somewhere. You could update the status line with additional details, such as the user name of the remote connection, the time, the size of the last message, the time the last message was received, etc. You could add the ability to scroll back lines of text that have rolled off the top of the display. Use the arrow keys or a keyword in the same way "quit" is used to end the program. You could add the ability to save all the text in a chat session to a file. You could add color using functions available in the ncurses library. You could extend the design to allow chatting among more than two users (e.g., see the **selectserver.c** multi-person chat server on Beej's Guide to Network Programming web site).

6. Submission requirements: The code you submit must compile using the `-Wall` flag and **no** compiler errors or warnings should be printed. To receive credit for this assignment your code must correctly establish a chat connection and display some lines of text as in the provided ncurses example. Code that does not compile or fails to pass the minimum tests will not be accepted or graded.

7. Submit your file **lab8.c** to the ECE assign server. You submit by email to ece_assign@clemson.edu. Use as subject header ECE222-1,#8. When you submit to the assign server, verify that you get an automatically generated confirmation email within a few minutes.

See the ECE 222 Programming Guide for additional requirements that apply to all programming assignments.

Work must be completed by each individual student. See the course syllabus for additional policies.