Objectives
This is the third assignment for the CMPUT 379 Operating Systems course at the University of Alberta. The goal is to create a simple chat program using sockets that allow communication from many clients to a single server. Clients may reside on the same or different hosts. In the meantime, a second goal is to become familiar with using socket operations including socket(), connect(), bind(), accept(), and listen(). Many of the chat features are the same as assignment two, however, this assignment also gives us the opportunity to learn how to ping clients via keepalive messages and display activity reports in a periodic way.

Design Overview
- There are two main loops: the client loop and the server loop
- The client loop processes commands from the stdin and the socket that connects it with the server if it is in a connected state
- I used a N flag to keep track of whether the the client loop should poll the socket
- N was incremented when the client connected and decremented when disconnected
- The occasions where the client could disconnect include:
    - The server responds with error due to duplicate username
    - The server responds with error due to reaching the client limit
    - The chat user voluntarily disconnects via close
- Since there is a connect() function for sockets but no corresponding disconnect() function, the workaround in my solution was to close the socket and recreate it
- Within the client loop, there are two distinguishing tasks
    - One is to process input from stdin and redirect it to the server, then wait a little while for the server to respond
    - The other task is to process commands from the server
        - On normal occasions, this simply means echoing the servers command line to stdout
        - On error conditions, the socket is destroyed and recreated
- Polling uses a timeout of 0, so the client is able to check the keep alive time against the KAL_interval every iteration to see if it should send a keep alive message to the server
- The chat server has a main listening socket that accepts connections from clients
- Initially, it listened to only nclient sockets. However, I had issues with this approach whereby even though it had seemed that I limited the number of clients that could connect, clients were still able to connect to the server. I later realized that this was because the second argument of listen() was merely advice and not strict by any means. So the workaround was to accept unlimited connections, but when I accepted the socket, I would check if the client limit was reached. If so, an error message would be sent back to the client for it to disconnect.
- The variable N is used to keep track of the number of clients. It is incremented when a client successfully connects and N <= nclient and decrements when a client disconnects.
- When a client disconnected, to preserve polling only the logged in chat users, I would have to shift all clients after the disconnected clients “one to the left” in all the resource containers belonging to the clients.
- The resource containers include
    - newsock[nclient+1] is an array of socket descriptors
    - sfpin[nclient+1] is an array of corresponding file descriptors for the sockets
    - usernames[nclient+1] is an array of usernames
    - recipients[nclient+1][nclient+1] is a list of list of recipients, where each client has a list of recipients
    - keepalive[nclient+1] is the last time the client sent a keep alive message to the server
    - lastcmd[nclient+1] is the last time the client sent a actual command (not keep alive message) to the server. (used to keep track of connected, but idle clients)

- The server also polls stdin to check for message to exit.
- On exit, the server sends an error message to all clients telling them to disconnect, waits for a while, and then closes all sockets and terminates.
- The server sets a alarm that goes off every 15 seconds. There is a signal handler for that alarm to uses the loingjmp function to go back to a point in the server loop, which handles resetting of the alarm and displaying the activity report
- While we display the activity report, we also look for disconnected clients by checking the time against the last keep alive time. If so, we display a message that they have been disconnected. Otherwise, we display a message of the clients last command time.

Project Status
All of the features highlighted in the assignment description are implemented. There is a makefile and a project report. Every feature is well documented.
Several difficulties were encountered along the way including:

- Which of read, recv, fgets with FILE *, etc. to use
- Which of write, send, puts with FILE *, etc. to use
- I eventually chose the one that provided the correct output e.g. didn't block, had the full message, flushed to the stdout, etc.
- How to get the client prompt to display correctly
- How to "disconnect" the client socket without closing it - I never found a solution so had to resolve to closing and recreating it
- Why listen was not working; a client could connect even though I set a limit on listen(). After hours of scouring the web and rereading the assignment description, I found that it was only "advice" and not a strict boundary that the second argument of listen() was providing. So I decided to allow the server to accept that clients connection but upon accepting it, check if the client limit is reached and if so, tell the client to disconnect and then close the socket.
- Using errno to detect if the server was alive
- How to implement keep alive messages
- How to implement periodic activity reporting
- Eventually settled with using a combination of alarm() and setjmp and loingjmp however I had the issue where it worked on my MAC OSX build but not my ubuntu virtual machine
- How to check for disconnected clients using keep alive messages
- How to handle polling of logged in users only; this was difficult because poll() function depends on a size variable. I resolved it by shifting logged in users to the left of the arrays.
- There were many bugs, many hours of testing and thinking to get the above issue to work appropriately. Something as simple as forgetting to set one of the arrays resulted in hours of debugging.
- Countless combinations of testing to make sure all of the features outlined in the assignment description are in place.
- Testing on my MAC OSX system as well as my ubuntu virtual machine. Sometimes the discrepancies are many more hours of debugging.
- Many more that I forget to mention

Testing and Results

- I tested using one client, one server and making sure that all of the chat commands were functional. The activity report was displaying the correct information. When I ctrl-c'ed, the activity report would display that the client has disconnected and report the last keep alive time. I was able to exit the server using the exit command.
- I tested opening the server with a client limit of 0 and made sure that the client got an error message back saying the client limit was reached when it tried to connect.
- Made sure the client didn't crash when it tried to use commands other than open and exit when it wasn't in a connected state

- Tried with multiple clients and one server. Made sure messages on one client that had others as recipients were able to send messages to them. Made sure that when one client disconnected, the other clients were still up and the correct output was being displayed in the activity report.

Acknowledgements

- APUE book for error handling
- Lecture notes on sockets
- Stack overflow for questions related to socket functions, alarm, etc.
- The linux sys call manual
- Assignment 3 project description