CS 284 - Assignment 3 - Client/Server chatroom

Important Note:
You will be working in groups of two (2) for this assignment. Please choose your partner as soon as possible. One partner will submit the final project and the main header should list the names of both partners.

This assignment involves generating a simple client/server chatroom with sockets. The server should support up to 10 clients at one time. The client's behavior will be defined as the following: upon starting the client, the user will be asked to supply a hostname for the server, and a username.

Once the connection is formed, the chatroom behavior is defined as follows:
A user types a message and presses return/enter. This message is sent to the server, and echoed onto any other connected client terminals. Beyond this basic behavior, a few special commands should be defined. If a user types /exit, /quit, or /part, the client should exit. The server should then print a message that the client(identified by the username he gave) has left the room. But if the user presses control-C at the client side, instead of exiting, the client should print a nice error message asking user to type /exit, /quit, or /part to exit the chatroom.

Finally, the server should gracefully quit on shutdown, that is, if a control-C is given to the server, it should warn the clients that it will shut down in ten seconds, wait ten seconds, then shut off. When the clients receive warning that server is shutting in ten seconds, they exit just before server does.

For the connections, you'll be using the Berkeley Sockets API. Dr. Ercal has several examples on his website.

For signal catching, there are again some examples on the class website:
http://web.mst.edu/~ercal/284/SignalExamples/Samples.html

Tips:
On the server side, client connections will be represented as threads...that is, one thread should be spawned per client connection.
On the client side, there are multiple ways to implement read/write ability, but threading (two threads, or one thread and main) is an obvious way to implement read/write functionality.

TEMPLATES PROVIDED BELOW:

```
/******************************************************************************/
/*   PROGRAM: client-template for the assignment                              */
/*                                                                            */
/*   Client creates a socket to connect to Server.                            */
/*   When the communication established, Client writes data to server         */
/*   and echoes the response from Server.                                     */
/*                                                                            */
/*   gcc -o client client.c -lpthread -D_REENTRANT                            */
/*                                                                            */
/******************************************************************************/


int main()
{
  socket stuff (including gethostbyname(), bcopy(), socket())

  connect

  introduce the client to server by passing the nickname;

  read the echoed message;

  create a thread for reading messages;

  while(still input to keyboard)
  {
    write input to socket
  }

  close stuff ...
}



void* reader(void* arg)
{
    get FD
    while (read(FD) > 0)
    {
       print to screen what is read
    }
    close(FD)
}
```

```
/*****************************************************************************************************/
/*   PROGRAM: server-template for the assignment                                                     */
/*                                                                                                   */
/*   Using socket() to create an endpoint for communication. It returns socket descriptor          */
/*   Using bind() to bind/assign a name to an unnamed socket.                                        */
/*   Using listen() to listen for connections on a socket. Using accept() to accept                 */
/*   a connection on a socket.  It returns the descriptor for the accepted socket                   */
/*                                                                                                   */
/*   gcc -o server server.c -lpthread -D_REENTRANT                                                   */
/*                                                                                                   */
/*****************************************************************************************************/

int FDarray[MAX_CLIENT]   /* allocate as many file descriptors
                             as the number of clients  */
int counter; mutex m;
{
   socket
   bind

   listen(n)   /* n is the size of the queue that holds incoming requests
                  from clients that want to connect  */

   while(( someint = accept(socket)) > 0)
   {
      lock(m);
      FD[counter++] = someint;   /* first check for room here though */
      unlock(m);
      thr_create(bob, someint, ....)
   }

   close stuff ...

}

void bob(void* arg)    /* what does 'bob' do ? */
{
   get  fd;

   get the host name;
   read in client name;
   print a message about the new client;

   while ((read(FD, buf)) > 0)
   {
      lock(m)
      loop
        write message to each FD
      unlock(m)
   }

   lock(m);
   remove myself from FDarray
   unlock(m)
   close(FD)
   thr_exit()
}
```