CS332: TCP Chat Server

For this assignment, you need to write the chat server for which you wrote a client last week.

The basic outline of the server code is this:

- Your code reads options from the command line (see below).
- Your code opens a socket, makes it `REUSEADDR` (look it up!), binds it to a port (default 12345), and begins listening on it.
- Add this main socket to a list of sockets from which to read. I'll refer to that variable as `socks`.
- In a forever loop:
    - Do a `select()` on `socks`
    - If the socket that is ready to read from is the main one, accept the new connection, and send the string "Thank you for connecting" on the socket. Add the new client socket to your `socks` list.
    - Else (the socket is an already connected client socket):
        - Call `recv(1024)` on the socket (to read up to 1024 bytes), reading in the message from the socket.
        - If the result is bad, remove that socket from `socks`.
        - Else:
            - Send the message you read from the socket on all the other client sockets.

You may implement this in any language you want, but the outline I've given you above is based on my python implementation.

You must implement these command-line arguments:

Name(s)	Option	Meaning	Default value (if option not given)
-v, --verbose		Turn on verbose printing to help debugging	False
-p, --port	Port number	The TCP port the server is listening on.	12345

When verbose is turned on, your client should print out enough information to help you figure out what your program is doing. E.g., my client prints out a message whenever it gets a new connection from a client, whenever that client connection goes away, whenever it resends a message to all clients, etc.

Implementation Notes

Your code should handle errors gracefully. I.e., if a system call returns an error, you should print out a useful message and exit gracefully. The user should not see a core dump or a stack trace printed out. You must handle the case where the server goes down when your client is still connected.

You will find the first lines of the talk_server.py file in `/home/cs/332/sp2016/chat/` folder.

You exit your talk_server by doing `Ctrl-C` in the terminal window in which it is running.

There are a **ton** of implementations of this online. For your own learning experience, do not copy and paste code from them. If you have to look up how to use select(), e.g., you'll probably see an implementation of the client or server… Try to implement as much as you can without copying code from online. **You must cite every place where you find code that you copy into your code.**

Those Doing the Class for Honors

The server must recognize and respond to the following special message that clients can send:

- `/who`: return a string of info about who all is connected. The string looks like this:

  ```
  <n> clients connected.
  <name> connected from <IP address>, <port>.
  … <previous line repeated for each client> …
  ```

  To do this, you'll have to keep track of info about the names the clients chose for themselves, and where the clients are connected from. I do this in a dictionary that maps the socket to a (name, IPaddress, port) tuple. The IP address and port you get from the result of the accept() call. Initially, the name will be unknown, so I just use the string "`<unknown>`". The name you will have to get from a message sent by the client. Recall that each client prepends the following to the beginning of each message:

  ```
  <name> says: <text of message>.
  ```

  When you receive a message from a client, parse that message, extracting the <name> and update your dictionary with the name.

Grading Rubric: 20 pts total

- 15 points for correctness.
    - Operates correctly: 8
    - Handles command-line options correctly: 4
    - Handles error conditions correctly: 3
- 5 points for clean, well-indented, well-documented code.

Don't forget to put your name, date, etc., in a comment at the top of your file(s).
Submit by copying to `/home/cs/332/current/<yourid>/chat/`.