Create an IRC Server and Client - IRC Client/Server connections

Submit Assignment

Due Friday by 11:59pm **Points** 80 **Submitting** a file upload

For this second IRC deliverable, you will begin setting up the actual IRC network. Specifically, you'll need to implement the message handlers responsible for processing SERVER, USER, and QUIT messages. By the end of this deliverable, you should be able to pass the SERVER, USER, and QUIT test cases. This will require you to implement several functions in IRCServer.py and IRCClient.py.

Note: If you've already submitted the complete assignment, you do not need to submit for this assignment.

You specifically need to implement the following functions in IRCServer.py:

- **process_data(self, select_key, data)**
  - Process_data is responsible for taking a received message, spliting it into its discrete components (prefix, command, params), and passing this to the appropriate message handler.
  - IRC messages can be highly variable. The only constant is that all IRC messages have a command value. The prefix is optional and params are optional. Furthermore, it is possible for an IRC command to only have non-trailing params, non-trailing params + a trailing param, or only a trailing param. Your code needs to handle all possible combinations of messages.
    - The presence of an optional prefix is indicated by a message starting with **:**
    - The presence of an optional trailing parameter is indicated by a message containing a : that is not the first character
    - Spaces separate different components of the message, EXCEPT for in the trailing parameter. All characters after the : indiciating a trailing parameter should be treated as part of the trailing parameter.
  - Once you have parsed the message into its prefix, command, and params, you should call self.message_handlers[command](select_key, prefix, command, params) in order to pass the message on to the appropriate message handler.
- **handle_server_message(select_key, prefix, command, params)**
  - SERVER messages contain information about new servers registering with the IRC network. The goal of this function is to update your adjacent_servers and server_lookuptable data structures with information about the new server, and to pass information about the new server to all other adjacent servers.
  - If a SERVER message does not contain a prefix, this means that the new server is directly adjacent to you. This also means that it is a brand new server that knows nothing about the state of the larger network. You are then responsible for sending the new server all information you know about the

  state of the server, including information about yourself, other servers, users, and channels. You communicate this information to the new server by sending it standard IRC messages.
  - When a socket originally connected to you, you created a ConnectionData object. Once you receive a SERVER message, you now know that the entity on the other side of the socket is a server. You now need to replace that ConnectionData object with a ServerDetails object that can hold information information about the server
- **handle_user_message(select_key, prefix, command, params)**
  - USER messages contain information about new users registering with the IRC network. The goal of this function is to update your adjacent_users and user_lookuptable data structures with information about the new user, and to pass information about the new user to all other adjacent servers.
  - If a USER message does not contain a prefix, this means that the new user is directly adjacent to you. You should send the user a RPL_WELCOME numeric_reply once it registers with you.
  - When a socket originally connected to you, you created a ConnectionData object. Once you receive a USER message, you now know that the entity on the other side of the socket is a user. You now need to replace that ConnectionData object with a UserDetails object that can hold information information about the server
- **handle_quit_message(select_key, prefix, command, params)**
  - QUIT messages remove a user from the IRC network. The server should remove the user from its adjacent_users (if it is an adjacent user) and users_lookuptable data structures. It should then broadcast this user's QUIT message to all other adjacent servers.
  - In a future assignment, the user will be removed from all channels it has registered with. You do not need to implement that functionality currently.

I recommend you implement the following functions. These functions encapsulate commonly used code that will make your message handlers more compact and easier to write.

- **send_message_to_server(self, name_of_server_to_send_to, message)**
  - Using the name of the server, look up the associated ServerDetails object stored in self.servers_lookuptable, and add the message to this server's write buffer
- **send_message_to_client(self, name_of_client_to_send_to, message)**
  - This method is slightly more complex that send_message_to_server. You must first determine if the user is adjacent to this server, or if it is connected to a remote server.
  - First, look up the user's UserDetails object in the users_lookuptable using the user's name
  - Next, determine if the user is adjacent. If so, add the message to the user's UserDetails write buffer
  - If not, find the first link stored in the UserDetails object and forward this message to that server, which will deliver the message if that user is adjacent to it, or forward it on if the user is remote
- **send_message_to_select_key(self, select_key, message)**
  - If you need to send a message BEFORE a server or user has been registered in the appropriate lookuptable, then you should use this method.
  - Here, you'll fetch the ConnectionData object associated with this select_key and add the message directly to that write buffer
- **broadcast_message_to_servers(self, message, ignore_server=None)**

- This function should call send_message_to_server for all *adjacent* servers, except for the server named in ignore_server.
- Ignore_server is included as a parameter because you will sometimes want to broadcast a forwarded message to all adjacent servers, EXCEPT for the adjacent server that originally gave you the message you are forwarding

----

There is not as much to do in IRCClient.py. You specifically need to implement the following functions in IRCClient.py:

- **connect_to_server(self):**
  - In connect to server, you need to first create a socket and connect to the server (server information is stored in self.serveraddr and self.serverport).
  - Once you've connected, you need to send your initial USER registration message to the server. Next, listen for a response containing the RPL_WELCOME message and call self.start_listening_to_server()
- **listen_for_server_input(self):**
  - You should call recv on the socket and pass the message to process_server_input(). Loop until self.request_terminate is set to true.
  - NOTE: You should never set request_terminate to true. That is handled for you by the testing framework
- **process_server_input(self):**
  - Split the data into individual messages, and then divide each message into its individual components.
  - Once this is done, check to see if the numeric response in the message is contained in self.response_handlers. If it is, call self.response_handlers[numeric_response](prefix, params)
  - Else, if it's not, call self.print_message_to_user(" ".join(params))
- **send_message_to_server(self, message):**
  - Encode and send the message to the server using your socket
- **quit(self, quit_message=None):**
  - Prepare a QUIT message and send it to the server

I recommend you implement the functions in the following order:

1. IRCServer.process_data()
2. IRCServer.handle_server_message()
3. IRCServer send/broadcast helpers
4. IRCClient.connect_to_server()
5. IRCClient.send_message_to_server
6. IRCServer.handler_user_message()
7. IRCClient.listen_for_server_input()
8. IRCClient.process_server_input()
9. IRCClient.quit()

10. IRCServer.handle_quit_message()

To run the test cases, run **IRCNetworkLauncher.py**. Make sure you have a folder named TestCases in the same directory as the python file, and the appropriate test cases in this folder. This will happen automatically if you download the project folder as a zip and extract it.

What to submit:

Please submit a zip file containing your IRCServer.py and IRCClient.py files.