Assignment - Instant Messenger

Networks and Systems - Networks

Dr. Anne Reinarz

Hand-out: October 26, 2020

This assignment is to be completed and handed in via DUO. All code should be written in Python 3.5 (or above) and must be stored in files named server.py and client.py. You must use the "socket" library directly (i.e. not via any other Python module) for network communication.

Assessment tasks:

Part I: Implement a simple instant messenger (Total: 45 marks)

The first task is to implement a client-server system, which implements an instant messenger using TCP, allowing users to chat with each other. The instant messenger will consist of a client and a server program. You should be able to invoke server and client as follows:

```
python server.py [port]
python client.py [username] [hostname] [port]
```

For example, `python client.py duck 127.0.0.1 12000` would cause the client to connect to `127.0.0.1` (the local system) on port number `12000` with the username "duck".
The server and client should implement the following functionality (**30 marks**):

- When a client connects, display a simple welcome message from the server. On the server, print where the connection is coming from (IP address and port).
- Allow multiple clients to connect.
- On the client, provide an input prompt allowing the client to send messages.
- Clients should be able to send multiple messages. These messages should be displayed to all currently connected clients.
- Use the username passed to `client.py` to print "[username] has joined" or "[username] has left" on all connected clients whenever a client connects or disconnects from the server (even when leaving due to connection loss).
- Also display the the username in front of each message, so users can tell who is saying what.

- One of the connected clients disconnecting should not cause the server to crash (see also: error handling).

Hints:

- Be sure to build up the functionality of your application step by step, for example, do not attempt to write the entire server before starting the client.
- If you're unsure of how to start have a look at the lab assignments 2 and 3.
- I suggest using a dictionary to store all current connections. Ensure that when a client drops out, that connection is removed from the dictionary.
- You may find it useful to look at non-blocking sockets and `select()`, to ensure that all messages can be received. E.g. (note the format of the if statement):

```
r,w,e = select.select([client.connection],[],[client.connection])
if client.connection in r:
    [...]
```

- When implementing Part I, it is ok to send all messages directly as fixed length byte arrays over the sockets. We will look at a protocol in Part II.

Logging: 5 marks

Produce a log file called `server.log` when the server is run. This log should contain information about all clients connecting, disconnecting, as well as any messages sent.
Sample output showing at least two clients connecting, chatting and disconnecting should be included in the submission.

Error/Exception Handling: 10 marks

Any errors that can occur on either client or server side should be handled appropriately. Especially watch out for the following:

- In case of a crash, attempt to close remaining connections and print an error message to the log file.
- The server should not allow any client to transmit messages without having first set a username.
- If the server is not available or port/hostname are wrong, an error message detailing the issue should be printed.
- Clients crashing or losing connection should not impact the server or other clients.

Part II: Application Protocol (Total: 40 marks)

In order to extend our instant messenger with further features we need to define a proper protocol between client and server. Protocols define the format and order of messages sent and received among network entities, and actions taken on message transmission and receipt. Your next task is to define and implement (**20 marks**) a protocol on top of your implementation from Part 1.
This protocol should allow the following communication between server and client:

- this message should be sent to everyone
- this message should be sent only to one specified user
- users are joining/leaving or a user has renamed themselves
- the client requests a list of all current users

Further, there should be an interface making it possible for the user to:

- choose a new name
- list all other current users
- "whisper", i.e. send a message only to a specified user
- help, i.e. show all available commands. You may also want to show the available commands to the user at startup automatically.
- leave the chat

Error Handling: 10 marks

Handle all protocol errors appropriately. This includes unknown messages, whispering to a non-existing user, etc.

Documentation: 10 marks

Document the protocol between client and server in a file called `protocol.txt`. This needs to contain the following information:

- All types of messages that can be sent and any input they require.
- What the response should be (if any).
- Which messages can be sent by the server and which by the client?
- 2-5 sentences explaining your design choices

Code Quality

The code should be short, easy to read and understand. In particular, this means that:

- Program code is commented where necessary for understanding. Excessive comments should be avoided.
- Programs are appropriately structured (e.g., correct and consistent indentation style).
- An appropriate naming convention is used for variables, methods/functions and classes throughout the project. See `https://www.python.org/dev/peps/pep-0008/`.
- Appropriate use is made of external libraries and there is no evidence of redundant code, e.g. importing unused Python modules, functions that are never called, etc.
- Correct/appropriate use of conditional statement and iterative statements.
- The code is not over-engineered, i.e. it does not introduce unnecessary data structures or include overly-generic functions.

Submission details:

- Hand out date: Week 4 (Monday 26th October)
- Hand in date: Term 2 (Monday 11th February)
- Submission mode: via DUO

Important submission requirements:

Compress all your files in a single .zip file. Name your zip file "bannerID.zip" (Make sure to replace 'bannerID' with the anonymous banner ID given to you by the university). The .zip file should contain the following:

- The server and client source code in files named server.py and client.py.
- A sample server log file (server.log) corresponding to an example series of client-server interactions.
- A text file fully documenting the protocol you have defined.

Collaboration policy

You can discuss your work with anyone, but you must avoid collusion and plagiarism. Your work will be assessed for collusion and plagiarism through plagiarism detection tools.

Feedback sheet

Criterion	Marks	Comment
Part I: Client/Server		
Sever and client programs are implemented. Code realises all the functional requirements.	(/30)	
Log files	(/5)	
Error handling.	(/10)	
Part II: Protocol		
Protocol implementation	(/25)	
Error Handling	(/10)	
Documentation	(/5)	
Other requirements		
Code quality	(/15)	
Total	(/100)	