CS 3251: Computer Networking I (Section A and B) **Spring 2025**

Programming Assignment 2

Handed Out: January 14^{th}*,* 2025 *Due: 11:59pm, February* 11^{th}*,* 2025

1 Objective

- Understand the creation of sockets.
- Understand that application protocols are often simple plain text messages with special meaning.
- Understand how to parse simple text commands.

2 Introduction

In this assignment, you will create a **chat room** on a single computer where you and your (imaginary) friends can chat with each other. The following steps will be required to achieve this:

1. Create a server program that runs on a specific port passed via the command line.
2. Create a client program that can join this server.
3. The client needs a display name and a passcode to enter the chat room. (Assume all clients use the same passcode but a different display name). The passcode will be restricted to a maximum of 5 alpha-numeric characters only. Anything over 5 letters can be treated as invalid. You do not have to handle input over 20 characters long. The display name is a maximum of 8 characters long. We will not test for these corner cases.
4. The job of the server is to accept connections from clients, get their display name and passcode (in plaintext), verify that the passcode is correct, and then allow clients into the chat room.
5. When any client sends a message, the display name is shown before the message, followed by a colon (:), and the message is delivered to all other current clients.
6. Clients can type any text message, or can type one of the following shortcut codes to display specific text to everyone:
   (a) Type `:)` to display [Feeling Joyful].

(b) Type `:(` to display [Feeling Unhappy].

(c) Type `:mytime` to display the current time.

(d) Type `:+1hr` to display the current time + 1 hour.

Note: For `:mytime` and `+1hr`, format time as "Year Month Day Time Weekday". For example, 2025 Jan 10 08:23:14 Fri .

(e) Type `:dm ⟨receiver_username⟩ ⟨message⟩` to send a message to a particular client. This message will not be broadcasted to all connected clients, instead it will only be sent to the receiving client chosen by the user.

(f) Type `:Exit` to close your connection and terminate the client.

(g) [*Fun part – not graded*] backslash: `\` overrides the next word until a space or newline. For example, `\:mytime` will print :mytime instead of the actual time.

3 What will you learn?

- Basic socket programming to create a client-server application.
- How do multiple clients connect to a single server?
- How to keep multiple persistent TCP connections alive?
- Text parsing to develop a custom application layer protocol (This is, in spirit, very similar to how HTTP works, for example).

4 Which programming language to use?

You are required to use **Python** for this assignment. Python is generally supported on local systems. Thus, no container setup is required to run the project files.

However, we will be using Ubuntu 22.04 for testing your code. Therefore, you can choose to set up a similar virtual machine to test your code too. One method is Ubuntu on VirtualBox: `https://brb.nci.nih.gov/seqtools/installUbuntu.html`. Note that this step is completely optional; you can test the code on most systems natively.

5 Instructions/Expected Outputs

The autograder expects a very specific output from your programs. Please make sure that you follow all the conventions that we set and don't add any extra spaces/newlines in your output. As a rule of thumb, there should be no empty lines in your program's output and spaces should be added after a username.

(For example, ⟨username⟩: message instead of ⟨username⟩:message)
Note: There also should not be any output from your own program on your client (such as debugging prints).

5.1 Connection Establishment and Password Checking – Single Client

Note: The passcode will be restricted to a maximum of 5 letters.

You will create two programs: a client and a server. Each program takes the following CLI parameters: the client takes the server's IP address and listening port, the username, and the password (all clients should use the same password). The server takes its listening port and the password.

If the password is correct the client should print "Connected to ⟨hostname⟩ on port ⟨port⟩", otherwise, it should receive a failure message "Incorrect passcode". Whenever a new client successfully joins the chatroom, all other clients should receive a message indicating the username of the new user that has just joined the room (see below).

Note: For readability `$` signifies the start of a command and `#` signifies the start of an output. These are not meant to be included in the commands or outputs.

Command: `$ python3 server.py -start -port <port> -passcode <passcode>`
Output: `# Server started on port <port>. Accepting connections`
Command: `$ python3 client.py -join -host <hostname> -port <port> -username <username> -passcode <passcode>`
Output (on Server): `# <username> joined the chatroom`
Output (on new Client): `# Connected to <hostname> on port <port>`

Resource: Sample code for providing command line arguments to a python application: `https://docs.python.org/3/library/argparse.html`

5.2 Connection Establishment and Password Checking - Multiple Clients

The server should be able to handle multiple clients connecting to it. This means that by running the above client command again (with a different username), the server should perform similarly. The server should also inform the already-connected clients that a new client has joined.

Command: `$ python3 client.py -join -host <hostname> -port <server's port> -username <username> -passcode <passcode>`
Output (on Server): `# <username> joined the chatroom`

Output (on new Client): `# Connected to <hostname> on port <port>`
Output (on all other clients): `# <username> joined the chatroom`

- You don't have to check for unique usernames, we will only test the code by supplying unique usernames in the test cases.
- A connected client should maintain a persistent TCP connection with the server that's only terminated when the user types :Exit .
- A client is removed only if it executes :Exit command (i.e., don't assume that a client will be forcibly terminated).

5.3 Chat Functionality

After successfully connecting to the server, clients should be able to type messages that get sent to the server when the user enters a newline. All text before the newline should be sent to the server, displayed on the server's screen, and broadcasted and displayed on the screens of all clients.

Command (on a connected client with username: ⟨username⟩): `$ Hello Room`
Output (on Server): `# <username>: Hello Room`
Output (on all other clients): `# <username>: Hello Room`

You don't have to consider messages longer than 100 characters. No need to test what happens when you exceed 100 characters, and no need to handle arbitrary long inputs. The autograder does not test such corner cases.

5.4 Chat Shortcuts

As discussed earlier, clients should be able to send shortcuts that are translated to text. The shortcut should be displayed as full text on all screens.

Command (on a connected client): `$ :)`
Output (on Server): `# <username>: [Feeling Joyful]`
Output (on all other clients): `# <username>: [Feeling Unhappy]`

For the :dm shortcut, a client should be able to send messages to a particular receiver. The server should display the sender, receiver and the entire message in its output. It is assumed that the user will always enter a 'correct' username for the receiver, i.e, a username of a connected client.

Command (on a connected client): `$ :dm <receiver_username> Hello Roommate`
Output (on Server): `# <username> to <receiver_username>: Hello Roommate`
Output (on a connected client with username: ⟨receiver_username⟩):
`# <username>: Hello Roommate`

5.5 Leaving Chatroom

As discussed earlier, clients must be able to disconnect as well. To do this, the client should be able to type `:Exit` to disconnect. All other clients should see a message that this client left the chatroom.

Command (on a connected client): `$ :Exit`
Output (on Server): `# <username> left the chatroom`
Output (on all other clients): `# <username> left the chatroom`

6 Grading Scheme

- Single server-client program sets up connection: 20 points.
- Single server, multiple clients able to connect: 20 points.
- Server receives from any client, sends to all: 20 points.
- Login and passcode implementation: 20 points.
- Text parsing for shortcut codes: 20 points (See 6. under the Introduction above).
- Compilation errors are not acceptable.

7 Programming Dos and Don'ts

- You should use **sys.stdout.flush()** after your print statements in Python to ensure that the message is printed on the terminal in a timely fashion.
- Your server should bind to **127.0.0.1** as the host.

8 What to submit?

Please implement all functionality described above in **server.py** and **client.py** and submit these files to Gradescope. Note that the submissions should be made individually.

9 Further Clarifications?

We will maintain an Ed Discussion thread (See Pinned posts) for posting any further clarifications. Please look at it for any clarifications not listed in the instructions above. Please use the thread 'Programming Assignment 2: Megathread' for posting all doubts, and questions related to PA2.