Assignment: Anonymous Message Board
Networks and Systems – Networks

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.

Assessment tasks
You are required to implement a client-server system, which implements a simple anonymous bulletin board system using TCP. Your system should include the following:

Server Program

- Write a server program in Python. Name your program server.py.
- Once the server starts, it listens on a specified IP and port for incoming TCP/IP requests.
  - Your server must be invokable as follows:
    python server.py *serverip port*
    - E.g.: python server.py 127.0.0.1 12000
      would cause the server to listen on 127.0.0.1 (the local system) port number 12000.
- The server receives requests from clients and responds to them as described below.
- The server retrieves any requested information and returns the result to the client.
- The server must handle the following errors:
  - Unavailable/busy port: print error and exit.
  - No message boards defined: print error and exit.
  - Specified board does not exist: notify client of error.
  - Invalid message (e.g. undefined command or missing parameter): notify client of error.

Server message board state

- The server maintains the message board state using the contents of the “board” subfolder of the working directory it was executed in.
  - Each subfolder of the “board” folder represents one message board; folder names represent board titles. Spaces in board titles should be replaced with underscores (“_”). There is a single flat list of boards (i.e. there is no hierarchy of boards).
  - Each file in a subfolder of “board” represents one message; filenames represent dates at which messages were posted and message titles. Messages should be saved with filenames beginning with a timestamp of the form YYYYMMDD-HHMMSS, separated from the title with a minus sign “-”:
    - E.g.: “20191101-091100-This_is_Test_number_1”
      represents a message titled “This is Test number 1” posted at 9.11am on 1 November 2019.
    - File contents: content of the posted message.
- Before running your server program, the user will create the “board” folder, and one or more empty subfolders inside “board” folder to represent message boards.
- To post a message, the server creates a file with the appropriate name in the corresponding folder.
- To list message boards or read messages, the server examines the contents of the “board” folder in the filesystem.

Client program

- Write a client program in Python. Name your program client.py.
- The client sends its requests to a server listening on a specified host and port.
  - Your client must be invokable as follows:
    python client.py *serverip port*

        - E.g.: python client.py 127.0.0.1 12000
          would cause the client to attempt to connect to a server already listening on 127.0.0.1 (the local system) port number 12000.
- When the client starts, it should immediately attempt to connect to the server and send a GET_BOARDS command to find out what message boards exist, and display the result to the user as a numbered list (e.g. "1. Name of first board; 2. Name of second board; 3. …").
- The client then waits for user input, which must be one of:
    - Any number from the list – requests a list of 100 most recent messages in that board from the server, and displays the contents of these messages.
    - POST – posts a message, by first prompting the user for 1) the number of the board to post to; 2) the message title (a single line of text); 3) the message content (a single line of text), then sending a POST_MESSAGE message to the server.
    - QUIT – closes any open connections and terminates the client.
- After sending a command to the server, the client must inform the user that the command was successful, or if not should display an appropriate error message.
- Your code should handle situations that may occur prior, during and after client-server interaction. At a minimum, the client program should handle the following situations:
    - Server is not running/unavailable (print error and exit).
    - Server returned an error to the GET_BOARDS request (print error and exit).
    - Server returned an error in response to any other request (print error and continue waiting for input).
    - Server response to any request is not received after 10 seconds (print error and exit).

Your client and server must be able to exchange the following types of message:

Message	**Parameters**	**Example**	**Response/action**
GET_BOARDS	None	GET_BOARDS	A list of defined message board titles
GET_MESSAGES	Board title	GET_MESSAGES(test_board)	A list of the 100 most recent messages in the specified board
POST_MESSAGE	Board title, post title, message content	POST_MESSAGE(test_board, "A New Message", "This is an example message. It could be quite long.")	Create a file in the corresponding folder representing the message.

The precise formats used to serialize these messages are up to you.

Log files

- The server program should record the following items of information for each request it receives during all client-server communication in a log file named "server.log". One line should be used for each request, and the logged data should be tab delimited.
    - The IP and port of the client (e.g. "127.0.0.1:21000").
    - The date and time of the incoming client connection request (e.g., "Mon 8 Oct 11:57:27 2019"). [Exact format of this is up to implementer, provided it contains all corresponding date-time information.];
    - The type of message used in the request (e.g. "GET_BOARDS" or "GET_MESSAGES").
    - Whether the request was successful handled ("OK") or not ("Error");

Other requirements:

- The assignment requirements (e.g., program and file naming requirements, submission requirements, etc.) have been followed.
- Your system is compatible with Windows, Mac and Linux and runs correctly on any university machine.
- Your client-server system can either use one connection per request, or one connection per client. **However**, your client-server system must be capable of serving multiple clients concurrently.

Submission details:

- Submission date: Friday 6 December 2019 at 14:00
- Submission mode: via DUO
- Feedback date: Friday 24 January 2020

Important requirements:

Compress all your files in a single .zip file. Name your zip file as "*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.
- Sample client and server log files (client.log and server.log) corresponding to an example series of client-server interactions.
- A "board" folder containing the associated message board contents.

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	**Mark**	**Comment**
Sever program is implemented. Code realises all the functional requirements.	(/30)	
Client program is implemented. Code realises all the functional requirements.	(/30)	
Log files are generated automatically by the programs and contain correct information.	(/10)	
Error handling: The programs handle all errors appropriately.	(/10)	
Code quality: • Program code is sufficiently commented. • Programs are appropriately structured (e.g., correct and consistent indentation style). • Appropriate naming convention is used for variables, methods/functions and classes throughout the project. • Appropriate use of external libraries and there is no evidence of redundant code, e.g. importing unused Python modules. • Correct/appropriate use of conditional statement and iterative statements.	(/10)	
All of the 'Other requirements' of the assignment have been met.	(/10)	
Total	(/100)	