Networks Sub-module Assignment

(2021-2022)

1. Assignment introduction

This assignment includes two parts. Part 1 is to code an Instant Messenger in Python 3.5 (or above). Part 2 is to analyse a wireless transmission scenario and design your protocol. For detailed descriptions and tasks, please refer to the individual parts.

You should hand in your completed assignments via Blackboard Ultra. The deadline for your submissions is **2pm Thursday February 10th, 2022**. Each of you should include the following four files in your submission: the programmed client file (client.py), the programmed server file (server.py), the log file (server.log) and a PDF file answering the tasks in Part 2.

Please note that all submissions will be subject to plagiarism and collusion checks.

2. Assignment descriptions and tasks

Part 1: Implement an instant messenger (45 marks)

Note that all codes 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 communications.

You are required to implement a client-server system, which implements an instant messenger **using TCP** allowing users to chat with each other. This 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 John 127.0.0.1 12000` would enable the client using the username "`John`" to connect to `127.0.0.1` (i.e., the local system) via port number `12000`.

Please complete the following functions required for the server or clients in your programs.

1) The server and clients should implement the following functionality (30 marks):

Need to enable the user to be able to exit via user-commands

- 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 number).
- 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 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.

Does this include push updates with messages?

2) Logging (5 marks)

Fix log order

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.

3) 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.

Need To Test

- 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.

Need To Test

- Clients crashing or losing connection should not impact the server or other clients.

Process Killed, Process Ctrl+C, Network Comms Drop

****

Hints for Part 1 A:

1. 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.
2. If you are unsure of how to start, I suggest you to have a look at the lab assignments 2 & 3.
3. I also suggest using a dictionary to store all current connections. Ensure that when a client drops out, that connection is removed from the dictionary.
4. 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: [...]
```

5. It is ok to send all messages directly as fixed length byte arrays over the sockets.

****

Part 2: Analyse a simple wireless network (40 marks)

Refer to the following wireless network scenario.

Assume a wireless network consists of an access point (denoted as X) and two wireless nodes (denoted as A and B respectively). Assume that wireless nodes **A and B cannot hear each other's transmissions**, but **they can hear X** (i.e. X can hear A and B). Suppose

- At time *0 μs*, X is sending a packet to some other node and it completes sending this packet at time *100 μs*.
- At time *30 μs*, a packet becomes available for transmission at A. A needs *200 μs* to send this packet.
- At time *70 μs*, a packet becomes available for transmission at B. B needs *100 μs* to send its packet.
- Let the value of the backoff timer for A be *40 μs* and the value of the backoff timer for B be *60 μs*.

1) Sketch the above described topology to include wireless nodes X, A, B, and their coverage. You may use tools such as Word, Paint, Visio, etc. to complete this topology.

(5 marks)

2) Analyse the above transmission situation and describe the transmission procedure. In your description, clearly state which node starts to sense and transmit at what time and how long it would take. Start your description from *0 us* in the network.

(15 marks: 3 marks for the access point, 6 marks for each of the wireless nodes)

3) Design a protocol to address an issue that potentially happen during the above transmissions. In your protocol description, you will use one paragraph to state the issue that you want to address and your idea to address this issue. You will then present the steps of your protocol operations. Note, in each step, you should include all types of messages and who should send these messages.

(20 marks: 10 marks for the description of issue and protocol idea, 10 marks for protocol procedure description.)

3. How to submit?

You will submit four files:

- server.py,
- client.py,
- server.log
- a .pdf file. In this file, you present your analysis and design for Part 2. A template for the .pdf file is in the "Assignment" folder of the Networks sub-module on Blackboard Ultra.

Please compress all files into a single .zip file, and name your .zip file as "YourNameYourBannerID.pdf" (make sure to replace "YourName" with your own name and "YourBannerID" with the ID given to you by the University. For example, `JohnDavidson000856789.pdf`). Please upload your compressed file to Blackboard Ultra for submission.

The submission deadline is **2pm Thursday February 10th 2022**.

4. Collaboration policy

You may discuss your work with anyone, but you must do your work yourself and comply with the University rules regarding plagiarism and collusion (https://www.dur.ac.uk/learningandteaching.handbook/6/2/4/1/). Your submissions will be assessed for collusion and plagiarism.

5. Feedback Sheet

Criterion	Marks	Comments
Part 1 Instant messenger (45)		
Sever and client programs are implemented. Code realises all the functional requirements	/30	
Log file.	/5	
Error handling.	/10	
Part 2 Analyse a simple wireless network (40)		
Topology.	/5	
Transmission procedure description.	/15	
Protocol design.	/20	
Others (15)		
Code quality.	/8	
Presentation and readability (for the .PDF file).	/7	

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.

Presentation and readability:

Content should be concise yet clearly describing the full idea and protocol. In particular, this means that:

- Words should highly focus on describing the protocol idea(s) and steps. Too much background content or other irrelevant content will lose points.
- Give concise explanations/justifications of your idea/protocol.
- Avoid very long sentences. Consider using different ways of presenting your ideas (say tables, figures, etc.) as appropriate.
- Font(s) should be legible, particularly for those in a figure.
- Format and fonts should be consistent throughout the file.