introduction

exercise 1 .

Please make sure you've completed the [setup (exercise 0)](https://docs.google.com/document/d/1aPOsAzo922GlUJXTH9d5x7F8CzWvJPL6gMoGRSumI2s/edit#) first.

In this exercise, we'll:

1. [Write a client and server that talk over TCP using a simple binary protocol](#_wjizmvz66rhi);
2. [Extend the server to support multiple clients using multithreading](#_pn0lux1rte0s);
3. [Extend the server to manage the clients' data using the filesystem](#_l7kxai7yjf9w);
4. [Expose the clients' data via a web server](#_juemjdkpq86u).

To begin, go to the [course's GitHub page](https://github.com/advanced-system-design), Find the **exercise-1** repository, clone it, install its dependencies, virtual environment and requirements, activate it, and run its tests.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **git** clone git@github.com:advanced-system-design/exercise-1.git  $ **cd** exercise-1/  $ ./scripts/install.sh  $ **source** .env/bin/activate $ **pytest** | |  |
|  |

This should fail, but it's OK — we haven't started the exercise yet.

To submit, create a new repository called **exercise-1-*<your identity number>***, and re-configure the project to point there:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **git** remote remove origin  $ **git** remote add origin \ git@github.com:advanced-system-design/exercise-1-123456789.git | |  |
|  |

When you're done with the exercise (and all the tests pass), commit your changes, upload them, and check out your repository's GitHub page to make sure you've submitted everything (e.g. [https://github.com/advanced-system-design/exercise-1-123456789](https://github.com/advanced-system-design/exercise-0-123456789)):

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **git** add .  $ **git** commit -m 'Submitting exercise 1.'  $ **git** push origin master | |  |
|  |

# question 1

During this course, we'll be writing a system that supports a [Brain Computer Interface](https://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface) — imaginary hardware (...for now) that can read minds, and upload snapshots of cognitions.

Before we start working with "real" data, let's prepare the ground. In this question, we'll write a simple client that sends a "thought", and a simple server that receives it.

First, enter the **q1/** directory.

Edit **client.py** and implement the **upload\_thought(address, user\_id, thought)** function, which receives an address (a tuple of the server's IP address and port), the user ID, and some thought (as a string) — connects to the server and sends this data using the following binary protocol, using **Little Endian** encoding:

|  |  |  |  |
| --- | --- | --- | --- |
| user ID | timestamp | thought size | thought data |
| uint64 | uint64 | uint32 | <thought size> bytes |

That is:

* 8 bytes representing the user ID, which is a unique, positive integer;
* 8 bytes representing the current timestamp, in seconds since the [Epoch](https://en.wikipedia.org/wiki/Unix_time);
* 4 bytes representing how long the data is (let's call this *n*);
* And *n* bytes of actual data.

Furthermore, implement a **command-line interface (CLI)**, so that **client.py** can be executed as a script. It should receive the address in the form **ip\_address:port**, a user ID, and a thought, uploads it, and print **done**; if an error occurs, it should print the error.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **python** client.py "127.0.0.1:5000" 1 "I'm hungry"  done | |  |
|  |

Now, edit **server.py** and implement the **run\_server(address)** function, which receives an address (a tuple of the server's IP address and port), and starts listening on it for connections.

When a connection is accepted, the function should receive its data, wait for 1 second, and print the received timestamp, user ID, and thought. If the data is incomplete, it should raise an exception.

Furthermore, implement a CLI, so that **server.py** can be executed as a script. It should receive the address in the form **ip\_address:port** and keep running until I hit **control+C**.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **python** server.py "127.0.0.1:5000"  [2019-10-25 15:12:05] user 1: I'm hungry  [2019-10-25 15:15:02] user 2: I'm sleepy  ^C | |  |
|  |

##### some hints

1. **socket.send()** doesn't guarantee all the data will be sent; consider using **socket.sendall()** instead.
2. **socket.recv()** doesn't guarantee all the data will be received; you'll have to loop and keep "accumulating" data until you're certain you've received all of it (or until the connection closes).  
   To reiterate: even something as simple as **socket.recv(4)** isn't guaranteed to return all 4 bytes, but *up to 4 bytes* — make sure to loop and accumulate it!
3. use **server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)** to avoid annoying "port is already in use" errors.
4. When running the tests, make sure your server is not running in the background, so the tests don't fail because the port is already in use.

# question 2 .

Now that our server supports a single client, it's time to extend it to support multiple clients.

First, enter the **q2/** directory, and copy over **server.py** from the previous question.

Edit **server.py** so that it spawns a new thread for every connection, and handles it there. To test it, run multiple clients at the same time:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **python** client.py "127.0.0.1:5000" 1 "I'm hungry" &  $ **python** client.py "127.0.0.1:5000" 2 "I'm sleepy" | |  |
|  |

With the server from the previous question, this would take around 2 seconds; but with our new and improved server, it should only take 1 second (even if you run 10 clients!).

# 

# question 3

Now that our server supports multiple clients, it's time to extend it to actually retain their information.

First, enter the **q3/** directory, and copy over **server.py** from the previous question.

Edit **server.py** so that **run\_server(address, data\_dir)** also receives a data directory, in which it stores the clients' data (note that the CLI should be changed, too!).

When user **u** thinks the thought **t** at time **d**, instead of printing it, the server should write the thought to disk as **data\_dir/u/d.txt**; it's also no longer necessary to wait for 1 second before handling the client.

For example, if **data\_dir** is **/tmp/data/** and user **1** thought **I'm hungry** at time **1572005525**, the server should write **I'm hungry** to **/tmp/data/1/2019-10-25\_15-12-05.txt**.

If two (or more) thoughts of the same user occur at the same time, they should be written to the same file, in any order, separated by line breaks.

For example, if user **1** also thought **I'm sleepy** at the same time, the file **/tmp/data/1/2019-10-25\_15-12-05.txt** should contain:  
I'm hungry  
I'm sleepy

##### hints

1. I recommend using the **pathlib** module to handle files and directories.
2. If the user directory doesn't exist, you'd have to create it.
3. Beware of race conditions: if two threads are handling two thoughts that occurred to the same user at the same time, they'd be writing to the same file concurrently...

# question 4 .

Now that we store user data, we'd like to expose it via a website, so that it's easy to list all the users, and see who thought what at what time.

First, enter the **q4/** directory. In it, you'll find the **data/** directory, which is full of thoughts by different users at different times.

Edit **web.py** and implement the **run\_webserver(address, data\_dir)** function, which receives an address (a tuple of the server's IP address and port) and a data directory, and starts a web server on that address, serving that data.

As per tradition, implement a CLI, so that **web.py** can be executed as a script. It should receive the address in the form **ip\_address:port** and the data directory, and keep running until I hit **control+C**.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| $ **python** web.py "127.0.0.1:8000" data/  127.0.0.1 - - [26/Oct/2019 19:57:20] "GET / HTTP/1.1" 200 -  127.0.0.1 - - [26/Oct/2019 19:57:20] "GET /users/1 HTTP/1.1" 200 - | |  |
|  |

When you run the web server and browse its index (e.g. open a browser and go to **http://127.0.0.1:8000**), you should see a list of all the users available.

If don't know HTML, don't worry — all you have to do is make your web server return this:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **<html>**  **<head>**  **<title>**Brain Computer Interface**</title>**  **</head>**  **<body>**  **<ul>**  **<li><a** **href**="/users/1"**>**user 1**</a></li>**  **<li><a** **href**="/users/2"**>**user 2**</a></li>**  **</ul>**  **</body>**  **</html>** | |  |
|  |

Similarly, when you browse **/users/<id>**, you should see a table of that user's thoughts:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **<html>**  **<head>**  **<title>**Brain Computer Interface: User 1**</title>**  **</head>**  **<body>**  **<table>**  **<tr>**  **<td>**2019-10-25 15:12:05**</td>**  **<td>**I'm hungry**</td>**  **</tr>**  **<tr>**  **<td>**2019-10-25 15:15:02**</td>**  **<td>**I'm sleepy**</td>**  **</tr>**  **</table>**  **</body>**  **</html>** | |  |
|  |

I feel it's kinda obvious, but I'm going to say it anyway: the web pages should be created by the server **dynamically** — so when a new user is created, it should appear in the users list the next time I browse it; and when a new thought is uploaded, it should appear in that user's page. I *don't* expect you to type in all the thoughts in **data/** as HTML.

Here's how things should look, more or less:

|  |  |
| --- | --- |
|  |  |

##### hints

1. Don't write an HTTP server yourself — use the **http.server** module. There are plenty of tutorials on Google on how to write a simple web server.
2. You can store the HTML as a template with a placeholder, and format the necessary data in on-demand. For example:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| \_INDEX\_HTML = '''  <html>  <head>…</head>  <body>  <ul>  {users}  </ul>  </body>  </html>  '''  \_USER\_LINE\_HTML = '''  <li><a href="/users/{user\_id}">user {user\_id}</a></li>  ''' … users\_html = []  **for** user\_dir **in** data\_dir.iterdir():  users\_html.append(\_USER\_LINE\_HTML.format(user\_id=user\_dir.name))  index\_html = \_INDEX\_HTML.format(users='\n'.join(users\_html)) | |  |
|  |

1. When running the tests, make sure your webserver is not running in the background, so the tests don't fail because the port is already in use.