Lab 6: Message-oriented Communication

Many distributed systems and applications are built directly on top of the simple message-oriented model offered by the transport layer through transport-level sockets.

Conceptually, a socket is a communication endpoint to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read. A socket forms an abstraction over the actual port that is used by the local operating system for a specific transport protocol.

Let's consider two approaches of python **socket servers**:

1. Thread approach. The first one uses blocking methods to interact with socket. Because of this we had to create separate thread for each client. This approach is similar to how apache web server initially works.

2. Selector approach. The second approach is to use non blocking methods and OS selectors to interact only with active socket connections. In this case we can have a single thread which manages all clients. This approach is similar to how nginx web server works. The second approach is not bounded with server's memory and potentially can handle much more simultaneous connections. But only if we do not perform heavy work inside this single thread (or fixed amount of threads). All long running task should be proxied to application servers. That is why in high load it is usual to have nginx as frontend server (load balancer + serving static files) and several apache servers as application backends.

Now we need something with which a server can interact - **socket client**. We can use **telnet** protocol to connect to the server. Type these commands in the terminal:

1. Start the server in one terminal:

   ```
   $ python3 server_threads.py
   ```

2. Keep the above terminal open. Open another terminal and type:

   ```
   $ telnet localhost 8800
   ```

Telnet – is a client-server protocol, based on TCP transport usually on port 23. When it was developed in 1969 telnet were used in closed environments (academic institutions, or at large private and government research facilities). Where were no security requirements. Nowadays it replaced by SSH for using in public networks.

Usage:

```
$ telnet [hostname|ip] [port]
```

Assignment:

Create server and client scripts to transfer files using one of the approaches described above. Client script should accept name of file and server address+port as an arguments.

Format:

```
$ python3 your_script.py file domain-name|ip-address port-number
```

Example usage:

```
$ python3 send_file.py meme.png your_vps.ru 8800
```

```
$ python3 send_file.py video.mp4 18.33.12.49 9332
```

In addition, client script should calculate and print to console the current progress of transferring in percents. Server should run on AWS (use other VPS if you already have) and save the file with the same name as it was on the client machine.

Notes:

- Do not forget about name collisions. You can add '`_copy<number>`' to the end of the name before file extension if there are duplicates eg:

  ```
  meme.png meme_copy1.png meme_copy2.png
  ```

- If you have problems with transferring name you can save the file on the server with a random name and send the generated name back to the client. But in this case, you lose 0.5 points.

The submission should be a ***.pdf*** file with the following sections:

1) screenshot of sending progress
2) screenshot of ls -la . output
3) well formatted source code
4) link to GitHub repo or gist with source code.

References:

https://realpython.com/python-sockets/