Assignment 2

Problem Statement

K Laxman 2018CS50408

We need to write a server-client program that achieves the following :

- server listens on a port, server program is run with a port number as an input.
- client makes a TCP connection and server sends a file to the client.
- In addition to acting as a server, the server can also act like a proxy.

The server program is run with a port number as an input on console or as an argument.

Implementation Approach

Both Client and Server programs are implemented in **python3.6**. We are using socket programming and multi-threading to support *concurrent connections* in the server.

Client Implementation:

The Client reads the input file, take first proxy out and rend a tcp message to the that proxy. The message includes the file name and the remaining list of proxies/server.

For example, suppose the input file is given as follows:

```
filename
hostname1 12000
hostname2 11000
```

then the client will separate hostname1 and 12000 from the list and will establish a tcp connection with hostname1 on port 12000. If the server is up and accepting, then it will receive the following message from the client:

```
filename
hostname1 11000
```

After sending this message, the client will begin to receive the messages *serially* (byte by byte) for a continuous flow. If the client does not receive the data within 10 seconds, it will show a timeout error.

Server Implementation:

As I mentioned before, a server can act as a proxy as well, if and only if it gets a non zero list of proxies/servers from the client (or previous proxy).

The server socket listen with a maximum of 5 buffer connections capacity. Whenever a new client request is accepted, it creates a new thread for that pair of *clientsocket* and *address*. And it begins to search for another request. It shows that the server supports *concurrent connections* simultaneously.
Now let's analyze the working of a thread. It receives the message from the client. We split it in filename and the list of remaining proxies. Here two conditions occur:

- If the list of remaining proxies is empty, then it will work as a server. It will read the name of the desired file and look for that file. Then it will read that file byte by byte and send data continuously to the client. In this way, the data streamed continuously from the server through the proxies.
- If the list of remaining proxies is non empty, then it will work as a proxy. It will pick the top most server/proxy from the list, establish a client-server connection and send the filename and the remaining list of proxies. Then it will wait to receive the data from the next proxy/server. When the data begins to stream byte by byte, it simultaneously sends the bytes to the client socket.

This Model fulfils all the **requirements** that was mentioned in the assignment.

Testing

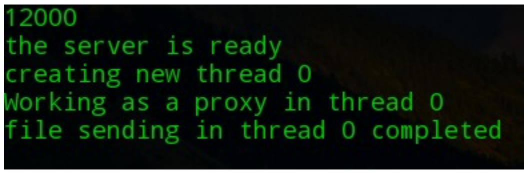
For the testing purpose, we need more than 1 publicly accessible computers. So I took two PCs from my family members and made them publicly accessible for tcp connections for port no. 12000 and 11000 using the tool *ngrok*.

```
pchost1 12000
pchost2 11000
```

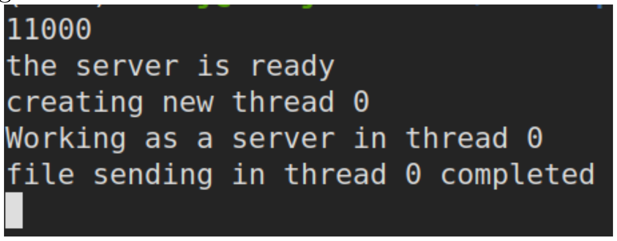
The file *myfile* was saved in pchost2.
I run the server on both pcs. Servers will keep running until we stop it.
I execute the client file in another pc with the following input file:

```
myfile
pchost1 12000
pchost2 11000
```

Both servers worked as expected and gave the following information on console:



```
12000
the server is ready
creating new thread 0
Working as a proxy in thread 0
file sending in thread 0 completed
```



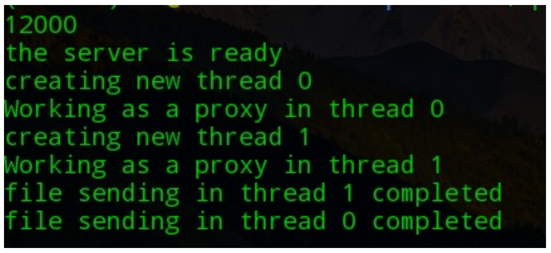
```
11000
the server is ready
creating new thread 0
Working as a server in thread 0
file sending in thread 0 completed
```

The file got downloaded in the client.

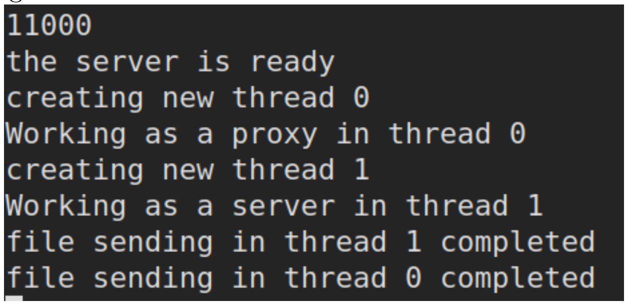
It clearly shows that in this case, the pchost1 worked as a proxy and pchost2 worked as a server. Since, no other requests were made to these servers at the same time, so single threads executed in both.
Now let's change the input file as the following:

```
myfile
pchost1 12000
pchost2 11000
pchost1 12000
pchost2 11000
```

Both servers worked as expected and gave the following information on console:



```
12000
the server is ready
creating new thread 0
Working as a proxy in thread 0
creating new thread 1
Working as a proxy in thread 1
file sending in thread 1 completed
file sending in thread 0 completed
```



```
11000
the server is ready
creating new thread 0
Working as a proxy in thread 0
creating new thread 1
Working as a server in thread 1
file sending in thread 1 completed
file sending in thread 0 completed
```

The file got downloaded in the client.
It clearly shows that in this case, two simultanously connections were made on both pchost1 and pchost2. In pchost1, both threads worked as proxies as expected. In pchost2, the first thread worked as a proxy, and the second thread worked as a server.
It shows that the server supports *concurrent connections*.

Commands to run the program:

For the client file, use

```
python client.py input.txt
```

Where input.txt is the input file in the client program.

For the server file, use

```
python server.py <port no>
```

Or

```
python server.py
```

And give the port no. in the input from console.