Project Report

CSE 336 Distributed Computing

Name: Mohamed Ibrahim El Ghamry ID: 15P6019

Project Title: Simple Multi-threaded Python Web Server

Date: 27 / 04 / 2018

Problem

It was requested to develop a simple Web server in Python that is capable of processing multi-requests using threads. Specifically, the Web server will

(i) create a connection socket when contacted by a client (browser);

(ii) receive the HTTP request from this connection;

(iii) parse the request to determine the specific file being requested;

(iv) get the requested file from the server's file system;

(v) create an HTTP response message consisting of the requested file preceded by header lines; and

(vi) send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, your server should return a "404 Not Found" error message.

Code

So, I started by importing everything from socket module, so we can use it for our connection. And imported the threading module, to use threads.

```
from socket import *
import threading
```

Then I've created a class called ClientThread, which has a constructor that takes the connectionSocket and the address and sets them.

```
class ClientThread(threading.Thread):
    def __init__(self, connect, address):
        threading.Thread.__init__(self)
        self.connectionSocket = connect
        self.addr = address
```

After that, the function run takes place in an infinite loop

```
    def run(self):
        while True:
```

- It tries to receive message, if message is found (file exists) it continues, if not it breaks and throws IOError that we will see later on.

```
            try:
                message = connectionSocket.recv(1024)
                if not message:
                    break
```

- Retrieves the filename, and starts reading from the file in variable outputdata

```
                filename = message.split()[1]
                print filename
                f = open(filename[1:])
                outputdata = f.read()
                print "outputdata:", outputdata
```

- Sends the correct headers

```
first_header = "HTTP/1.1 200 OK"
                header_info = {
                    "Content-Length": len(outputdata),
                    "Keep-Alive": "timeout=10, max=100",
                    "Connection": "Keep-Alive",
                    "Content-Type": "text/html"
                }

                following_header = "\r\n".join("%s:%s" % (item, header_info[item])
for item in header_info)
                print "following_header:", following_header
                connectionSocket.send("%s\r\n%s\r\n\r\n" %(first_header,
following_header))
```

- Finally, it can start sending the output data from the file it reads

```
for i in range(0, len(outputdata)):
                    connectionSocket.send(outputdata[i])
```

For the IOError part, if file does NOT exist, it does the same but it sends HTTP/1.1 404 Not Found header, and an error page is sent.

```
except IOError:
                f = open('error.html')
                outputdata = f.read()
                print "outputdata:", outputdata

                first_header = "HTTP/1.1 404 Not Found"
                header_info = {
                    "Content-Length": len(outputdata),
                    "Content-Type": "text/html"
                }
                following_header = "\r\n".join("%s:%s" % (
                    item, header_info[item]) for item in header_info)
                print "following_header:", following_header
```

```
                connectionSocket.send("%s\r\n%s\r\n\r\n" % (first_header,
following_header))
                for i in range(0, len(outputdata)):
                    connectionSocket.send(outputdata[i])
```

All of above, is implemented inside the ClientThread class, so every time a client requests, a thread take this request and perform this. For main function where the program starts, it starts by creating the server socket and socket port.

```
if __name__ == '__main__':
    serverSocket = socket(AF_INET, SOCK_STREAM)
    serverPort = 3000
```

Followed by binding socket to address and port, and places the socket into the listening state, able to send backlog outstanding connection requests

```
serverSocket.bind(('',serverPort))
serverSocket.listen(5)
```

And includes an array of threads

```
threads=[]
```

The server goes in an infinity loop, where it serves any request using the client thread class

```
while True:
        print 'Server is up and running on port %d' %serverPort
        connectionSocket, addr = serverSocket.accept()
        print "addr:\n", addr
        client_thread = ClientThread(connectionSocket, addr)
        client_thread.setDaemon(True)
        client_thread.start()
        threads.append(client_thread)
```

Lastly, it waits for all threads to join and finish, to proceed at the end by closing the connection.

```
for thread in threads:
        thread.join()
    serverSocket.close()
```

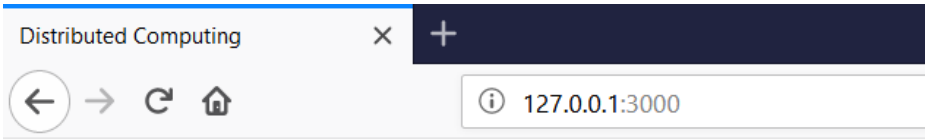
Starting the server

Server starts by the following command:

Python multithreadedServer.py

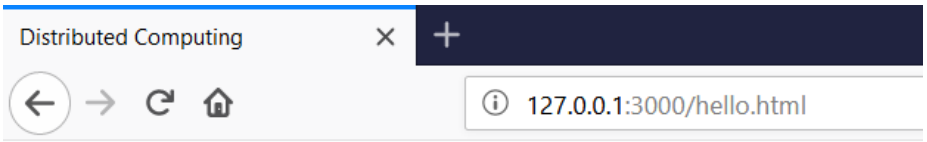
```
λ python multithreadedServer.py
Server is up and running on port 3000
```

Now it is up and running, and can serve our files on port 3000. So, we can go to: http://127.0.0.1:3000



Error: 404 Page Not Found

Hint: visit index.html page

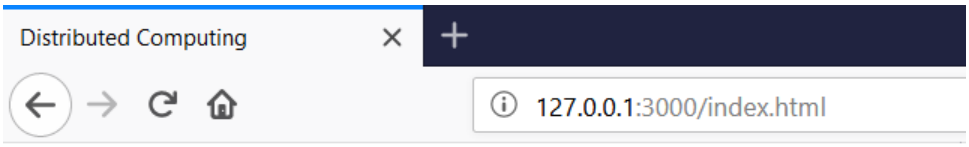


Error: 404 Page Not Found

Hint: visit index.html page

We will see a 404 page because the route to / or /hello.html is not found.

Instead we can visit an existing page like index.html at http://127.0.0.1:3000/index.html



Distributed Computing - Project 1

Voila!

We can open from multiple browses and hosts, and the server will still respond correctly, here is a screenshot of the multiple threads running to serve multiple hosts.