CSE 310 – Computer Networking
Fall, 2020
Programming Assignment 01
Multi-threaded Webserver and HTTP Client

Assigned: 09/14/2020
Due: 09/28/2020

For the first programming assignment I will be asking you to
build a multi-threaded web server and a simple HTTP client.

Please read the rest of this document describing Lab 1 carefully
for the details of the project and skeleton code. I have also
posted the skeleton code for the web server as a Python source
file.

You must complete both Optional Exercise 1 and Optional Exercise
2 as part of this programming assignment.

You may build your web server and HTTP client in either Python,
Java, or C.

BUT, I will only be releasing skeleton code in Python.

Submission Instructions:
1. Submit all source code.
2. Submit a README in .txt or .pdf format describing how to
   run your submission. This includes how to run and connect
   to your web server and how to run your client. Include your
   name and IDs in the README.
3. Submit screenshots of your browser interacting with your
   web server. The address bar must be clearly visible in the
   screenshots.
4. Please submit all files in a compressed directory as a zip
   file.
5. Please name your submission file:
   CSE310_PA01_LastName_FirstName.zip
6. This is an individual programming assignment. Any
   collaboration on coding will be considered a violation of
   academic honesty.

Lab 1: Web Server Lab

In this lab, you will learn the basics of socket programming for TCP connections in Python: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.

Code

Below you will find the skeleton code for the Web server. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

Running the Server

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.

Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

What to Hand in

You will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

Skeleton Python Code for the Web Server

```python
#import socket module

from socket import *

import sys # In order to terminate the program


serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepare a sever socket

#Fill in start

#Fill in end

while True:

    #Establish the connection

    print('Ready to serve...')

    connectionSocket, addr =   #Fill in start              #Fill in end

    try:

        message =   #Fill in start          #Fill in end

        filename = message.split()[1]

        f = open(filename[1:])

        outputdata = #Fill in start       #Fill in end

        #Send one HTTP header line into socket

        #Fill in start

        #Fill in end

        #Send the content of the requested file to the client

        for i in range(0, len(outputdata)):

            connectionSocket.send(outputdata[i].encode())

        connectionSocket.send("\r\n".encode())


        connectionSocket.close()

    except IOError:

        #Send response message for file not found

        #Fill in start
```

```
        #Fill in end

        #Close client socket

        #Fill in start

        #Fill in end

serverSocket.close()

sys.exit()#Terminate the program after sending the corresponding data
```

Optional Exercises

1. Currently, the web server handles only one HTTP request at a time. Implement a multithreaded server that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.

2. Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method.
    The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client.

    ```
    client.py server_host server_port filename
    ```