CSE 232 Section B, Computer Networks; Programming Assignment 2: TCP based Web application Lab

a. Due Saturday, Sep 28, 2024, midnight
b. To be done in groups of 2
c. You may program client/server programs in C++ or Python
d. The submission will consist of 3 parts:
    1. 2 to 4 page document describing the system you have designed (including all assumptions you have made),
    2. the code(s) as a separate file, and
    3. 5 to 8 slides that you will use to present your work before me and TAs. This must include screenshots at the client end verifying that your client program works as required.

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

Part A.

You will FIRST develop a web server that handles one HTTP request at a time. Your web server should:

1. accept and parse the HTTP request,
2. get the requested file from the server's file system,
3. create an HTTP response message consisting of the requested file preceded by header lines, and
4. send the response directly to the client.

(If the requested file is not present in the server, the server should send an HTTP status message "404 Not Found" 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, 6789, after the colon. You need to replace this port number with whatever port you have used in the server code. It should be one that is NOT reserved for current applications. 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. You may wish to try this out with port 80 as well.

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

Skeleton Python Code for the Web Server

```
#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
```

Part 2.

Currently, the web server handles only one HTTP request at a time. In order for it to handle multiple requests simultaneously, implement a multi-threaded 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. This will happen as and when HTTP requests are received.

Part 3.

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. (Does this client work with server implemented in Part 1 or Part 2 or both?)

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`