COSC370 - Socket Programming - Web Server

In this programming assignment, 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 servers 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
On a separate page 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 code ... """`. 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., 131.118.33.196). From another host, open a browser and provide the corresponding URL. For example:

```
http://131.118.33.196:6789/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. Note that I might ask for a demo of your program. Also, you're allowed to work with one classmate; if so, your team only needs to turn in one solution on MyClass.

Skeleton Python Code for the Web Server

```
#import socket module
from socket import *
import sys # In order to terminate the program

"""
 Fill in code to create a TCP server socket, assign a port number,
 bind the socket to server address and server port,
 and listen to at most one connection at a time
"""

# Server should be up and running and listening to the incoming connections
while True:
     print('Ready to serve...')
     """
      Fill in code to set up a new connection from the client
     """
    try:
      """
        Fill in code to receive a request message from the client.
      """
        # Extract the path (which is the second part of HTTP header)
        # of the requested object from the message.
        # assuming message holds the data from the previous line(s) of code
        filename = message.split()[1]

        # Because the extracted path of the HTTP request includes
        # a character '\', we read the path from the second character
        f = open(filename[1:])

        # Store the entire content of the requested file in a temporary buffer
        outputdata = f.read()

        """
         Fill in code to send the the HTTP response header line
         ("HTTP/1.1 200 OK\r\n\r\n")  to the connection socket.
        """

        #Send the content of the requested file to the client
        #Assuming connectionSocket has been created above
        for i in range(0, len(outputdata)):
                connectionSocket.send(outputdata[i].encode())
        connectionSocket.send("\r\n".encode())
        connectionSocket.close()

        except IOError:
            """
              Fill in code to send response message for file not found
              (use "HTTP/1.1 404 Not Found\r\n\r\n"),
              and send  404 Not Found as HTML body.
              Fill in code to close client connection socket.
            """


"""
 Fill in code to close server socket.
"""
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 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.

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
   ```