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.

```
#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 connection socket
    #Fill in start
    #Fill in end
serverSocket.close()
sys.exit()#Terminate the program after sending the corresponding data
```

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. Also note 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.

Look at the packages

Now start wireshark and browse your webpage again. Capture the HTTP request and response messages.

1. Look in the trace and find the IP and the port number of the client socket, the server (welcoming) socket and the connection socket.
2. Point out the Ethernet header, the IP header, the TCP header and the HTTP header in the "packet bytes" view of your HTTP response message. And explain the meaning of some interesting bytes in the packet.
   NB: We will cover the IP header and Ethernet header in a later lecture.

What to Hand in

1. The complete server code.
2. Screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.
3. Screen shots of any control printouts from the server.
4. Screenshots from Wireshark, marked with references to the question.
5. Answers to the questions.
6. Any additional information and learnings as you please.