Lab 1: Web Server and Client Programming

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 an HTTP packet. You will also learn some basics of HTTP header format.

Task 1. Web Server

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 on the server, the server should send an HTTP "404 Not Found" message back to the client.

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 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 on the server. You should get a "404 Not Found" message.

Skeleton Python Code for the Web Server

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 # To terminate the program

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepare a sever socket
#Fill in start
# Assign a port number

# Bind the socket to server address and server port

# Listen to at most 1 connection at a time

#Server should be up and running and listening to the incoming connections
#Fill in end

while True:
     print('The server is ready to receive')
```

```
	# Set up a new connection from the client by calling accept() method on
the socket
	connectionSocket, addr = #Fill in start   #Fill in end

	# If an exception occurs during the execution of try clause
	# the rest of the clause is skipped
	# If the exception type matches the word after except
	# the except clause is executed
	try:
		# Receives the request message from the client and decode it
		message = #Fill in start      #Fill in end
		# Extract the path of the requested object from the message
		# The path is the second part of HTTP header, identified by [1]
		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:])
		# Read and store the entire content of the requested file in a
temporary buffer
		outputdata = #Fill in start    #Fill in end
		# Send the HTTP response header line to the connection socket
		connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n".encode())

		# Send the content of the requested file to the connection socket
		for i in range(0, len(outputdata)):
			connectionSocket.send(outputdata[i].encode())
		connectionSocket.send("\r\n".encode())

		# Close the client connection socket
		#Fill in start
		#Fill in end

	except IOError:
			# Send HTTP response message for file not found
			connectionSocket.send("HTTP/1.1 404 Not
Found\r\n\r\n".encode())
			connectionSocket.send("<html><head></head><body><h1>404 Not
Found</h1></body></html>\r\n".encode())
			# Close the client connection socket
			#Fill in start
			#Fill in end

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

Task 2. Web Client

Instead of using a web browser as the client, for Task 2 you will develop a web client that connects to the web server in Task1, download the index.html file and display it. If you run both on the same computer, use different terminals or environments for the client and server, e.g., running the server in Visual Studio Code, and the client on the command prompt. (You can also connect to some other web server online by modifying the `serverName` and `serverPort` in the code.) If the requested file is not present in the server, the client should get an HTTP "404 Not Found" message.

Running the Client

Put an HTML file (e.g., index.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), or use `localhost` if both client and server are on the same machine. Your server is already working by testing it with a web

browser. Now you need to make the client work.

From another host or another terminal on the same host, run your client. In the client code, provide the server name and port. 'index.html' is the name of the file you placed in the server directory. The client should then display the contents of index.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, and the client should get a "404 Not Found" message.

Skeleton Python Code for the Web Client

Below you will find the skeleton code for the web client. 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.

```
from socket import *

# Server details
serverName = 'localhost'  # or the server IP address if the server is on a
different machine
serverPort = 6789  # Make sure this matches the server's port

# Create a TCP client socket
#Fill in start
#Fill in end

# Connect to the server
#Fill in start
#Fill in end

# Prepare the HTTP GET request. The .format(serverName) method call at # the
end of the string is used to insert the value of the serverName # # variable
into the placeholder {} in the Host header.
request = "GET /index.html HTTP/1.1\r\nHost: {}\r\n\r\n".format(serverName)

try:
    # Send the request to the server
    #Fill in start
    #Fill in end

    # Receive and print the server's response
    response = clientSocket.recv(1024).decode()
    print("Server response:")
    print(response)

    # Receive and print the content (if any)
    while True:
        data = #Fill in start#Fill in end
        if not data:
            break
        print(data)

except Exception as e:
    print("An error occurred:", str(e))

finally:
    # Close the client socket
    #Fill in start
    #Fill in end
```

What to Hand in

For Task 1, you will hand in the complete server code, along with the screenshots of the web browser, verifying that you actually retrieved the contents of the HTML file from the server. For Task 2, you will hand in the complete client code, along with the screenshots of your client, verifying that you actually retrieved the contents of the HTML file from the server.