# 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. You may use any html files with some embedded objects such as jpg files.

To see the result, run your server program and open a web browser. The address to browse for is `http:/<serverIP>:<port>/<filename>`; for example 'http://127.0.0.1:6789/HelloWorld.html' if you run server on the same PC as web browser. '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. 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.

We provided you two html files (https://canvas.eee.uci.edu/files/2242734/download) which will help you to test your web server. 'HTTP-wireshark-file1.html' has just one file with no embedded object. 'HTTP-wireshark-file4.html' has two embedded objects (jpg, gif).

Also test your code by trying to get a file that is not present at the server. You should get a "404 Not Found" message.

We will test your code by using an auto grader. It checks your code with html files like the second provided example.

## (1) Web Server – Fill in the Skeleton Code

```
#import socket module
from socket import *
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:], "rb")
		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:i+1])
		connectionSocket.send(b'\r\n\r\n')
		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()
```

## (2) Multi-Threaded Server

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.

## (3) HTTP Client

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
```