EXPERIMENT 2 – BUILDING A WEB SERVER - A HTTP CLIENT

Aim:

To develop a web server and client application using HTTP.

Software:

1. Oracle VM VirtualBox 6.1.38, Oracle Corporation
2. Ubuntu 22.04 (64-bit) Operating System
3. Microsoft Visual Studio Code, Microsoft Corporation
4. Python 3.10.7 (64-bit), Python Software Foundation

Theory:

I. HTTP:

Hyper Text Transfer Protocol, or HTTP, is a set of guidelines that servers must adhere to send all types of files, including images, text, audio, and video, across the internet (www). Clients and servers make up the internet. Consequently, if you visit the internet via a web client like Google Chrome, Mozilla, Internet Explorer, etc.

You request a web server whenever you type in the name of any website you choose to visit. Assume that when you type amazon.in, your browser—the web client—is asking the web server and frequently a large number of pages from it. It might be JSON, HTML, CSS, pictures, or video.

The fundamental interaction between you and the server is one of request and response. The HTTP protocol is being used to submit this request. A protocol is just a set of guidelines or standards that have been accepted by all users of the internet. The client-server request-response model is used here. When communicating with the server, HTTP, an application-layer protocol, often uses the Transmission Control Protocol (TCP).

II. Fundamentals of HTTP:

1. Data URLs – They are a particular class of URI that contains the resource they refer to. Although there are certain limitations, data URLs are quite practical.
2. Resource URLs – URLs of sources Although some websites the browser connects to can also access non-standard Resource URLs, those prefixed with the resource scheme are used by Firefox and Firefox browser extensions to load resources internally.
3. Multiple Input Multiple Execution (MIME) Formats – Different kinds of content can now be exchanged thanks to HTTP/1.0. This article explains how the MIME standard and the Content-Type header are used to accomplish this.

4. Choosing between URLs with and without www – This article offers advice on how to decide whether or not to use a www-prefixed name and discusses the implications of that decision.
5. Website Messages – The structure of HTTP Messages sent during requests or answers is pretty obvious. HTTP/2's binary frame and message structures encapsulate and represent HTTP/1.x messages.
6. HTTP/1.x Connection Management – The first HTTP version to offer persistent connections and pipelining was HTTP/1.1. HTTP/2 connection management - How connections are established and maintained has been extensively rethought in HTTP/2.
7. Negotiating the Connect – HTTP provides a series of headers beginning with Except for a browser to specify the format, language, or encoding it prefers. The process of this advertising, how the server should respond, and how it selects the best response are all explained in this article.
8. HTTP Server – HTTP server is an easy, static command-line program. It is robust enough for production use but also easy to use and hack for testing, local development, and learning.

III. Working:

Websites' domain names are used to access web server software, which ensures that the site's content is sent to the user who requests it. There are various parts to the software side, including at least one HTTP server. Both HTTP and URLs can be understood by the HTTP server. A web server is a piece of hardware that houses web server software and other website-related assets like HTML texts, pictures, and JavaScript files.

A web browser, such as Google Chrome or Firefox, will use HTTP to request a file that is stored on a web server. The HTTP server will accept the request when the web server receives it. Web server software, which ensures that the content of the site is transferred to the person who requests it, can be accessed through domain names for websites.

The software side is made up of several components, including at least one HTTP server. The HTTP server can comprehend both HTTP and URLs. A web server is a piece of hardware that contains JavaScript files, HTML text, and other website-related resources. It also houses web server software.

Using HTTP, a web browser like Google Chrome or Firefox will ask for a file that is kept on a web server. The HTTP server will accept the request when the web server receives it.

There are two possible ways for a web server to reply to a client request:

1. Transfer of the file to the client connected to the requested URL.

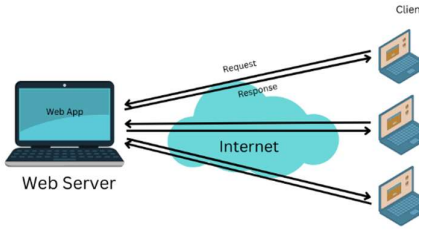

Figure 1 – Working of Web Server and Client

2. Response creation with the use of a script and database communication.

   (i) When a client requests a web page, the web server looks for the requested page and, if it is found, returns an HTTP response to the client.

   (ii) An HTTP response will be sent by the web server if the requested web page cannot be found, Error 404 No such page.

   (iii) If a client requests additional resources, the web server will get in touch with the application server and data store to build the HTTP response.

Source Code:

19CCE304_Expt_Code.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
"""
http.server - Defines classes for implementing HTTP servers.
HTTPServer - This class builds on the TCPServer class by storing the server
address as instance variables named server_name and server_port.
BaseHTTPRequestHandler - This class is used to handle the HTTP requests that
arrive at the server.
"""

import time # Provides various time-related functions

HOST = "192.168.109.13" # IP address of server
PORT = 5000 # Port Number

class requestHandler(BaseHTTPRequestHandler):

    # GET method is used to appends form data to the URL in name or value
pair:-
    def do_GET(self):
        self.send_response(200) # Sends the response header only
        self.send_header("Content-type", "text/html") # Adds the HTTP header
to an internal buffer which will be written to the output stream when either
end_headers() or flush_headers() is invoked.
        self.end_headers() # Adds a blank line (indicating the end of the HTTP
headers in the response) to the headers buffer and calls flush_headers().

        self.wfile.write(bytes("<html><body><h1>HTTP_client_server</h1></body>
</html>", "utf-8")) # Contains the output strean for writing a response back
to the client.

    # POST is a method that is supported by HTTP and depicts that a web server
accepts the data included in the body of the message:-
    def do_POST(self):
        self.send_response(200)
        self.send_header("Content-type", "application/json")
```

```
        self.end_headers()

        date = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time()))
# Convert a tuple or struct_time representing a time as returned by gmtime()
or localtime() to a string as specified by the format argument.
        self.wfile.write(bytes('{"time": "'+ date +'"}', "utf-8"))

server = HTTPServer((HOST, PORT), requestHandler)
print("The Server is Running...")

server.serve_forever() # Handle requests until an explicit shutdown() request.
server.server_close() # Clean up the server. May be overridden.

# Client URL (cURL) is a computer software project providing a library and
command-line tool for transferring data using various network protocols.
```

Screenshots of Input and Output:

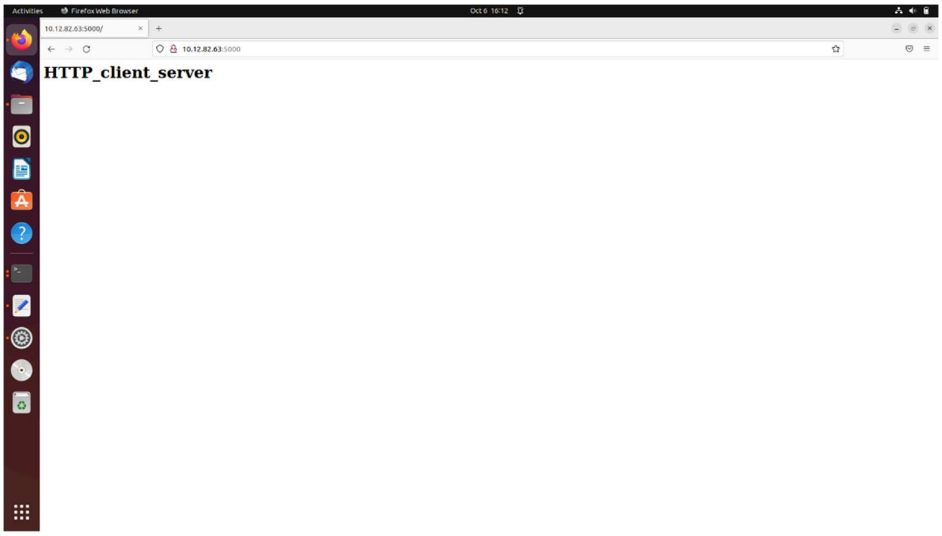

Figure 2 – HTTP Server Formed by HTML Code

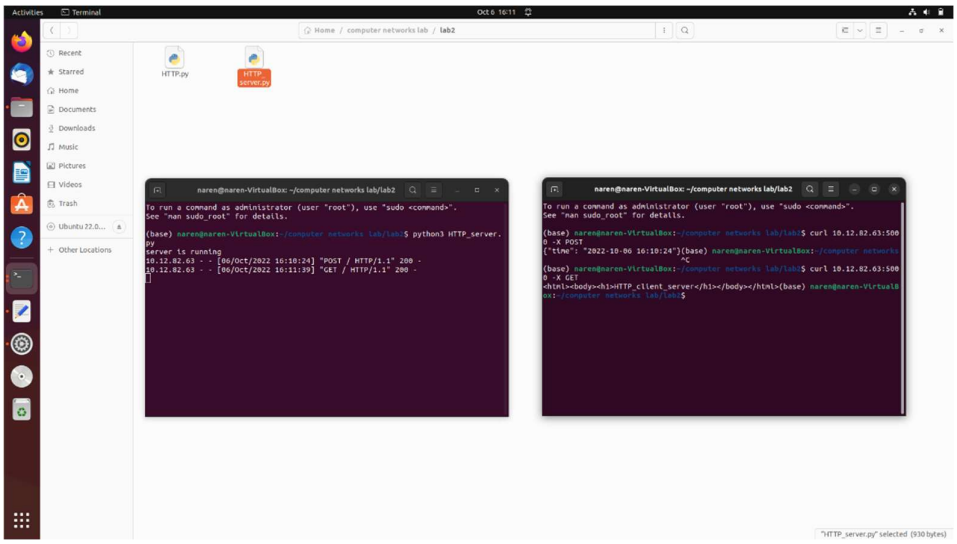

Figure 3 – Client and Server Communication

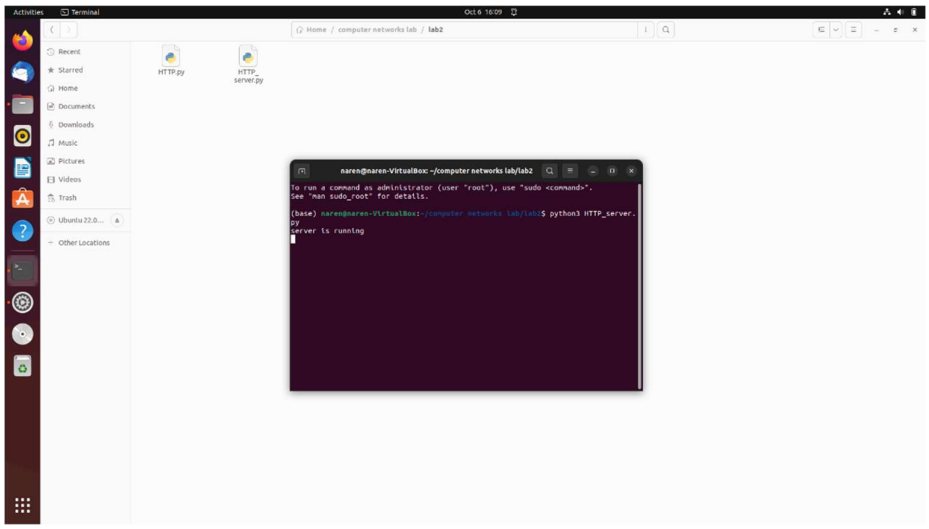

Figure 4 – Active Server

Conclusion:

Thus, the web server has been created and the required HTML script is displayed in the web browser under HTTP. Similarly, the client can access the server and get the necessary response from the server. Hence, all the simulation results were verified successfully.

References:

1. Foundations of Python network programming by Beaulne, Alexandre Goerzen, John Membrey, Peter Rhodes, Brandon 2014.
2. HTTP Server Python Documentation - http.server — HTTP servers — Python 3.10.7 documentation

Contribution:

1. Narendran S [CB.EN.U4CCE20036] – Development of Driver Code + Theory Documentation
2. Narun T [CB.EN.U4CCE20037] – Development of HTTP Server + Testing
3. Pabbathi Greeshma [CB.EN.U4CCE20040] – Development of HTTP Client + Topic Research
4. Santosh [CB.EN.U4CCE20053] – Code Debugger and Reviewer