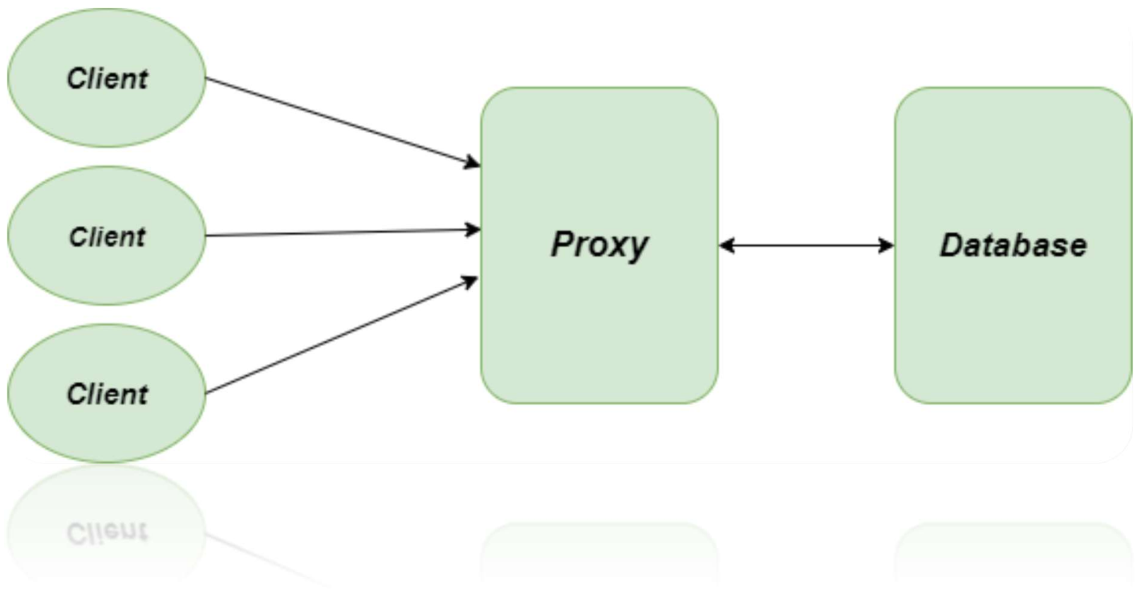
Multithreaded Proxy Server……!

Objective: The objective of this project is to create a multi threaded server to which a client can send his URL request and get access to the HTML page. In this we can access server as virtual clients through localhost.

Block Diagram:



Requirements:

Python: Python is an interpreted, high-level and general-purpose programming language. Here python is used to create sockets as the process is simple as it has inbuilt socket functionality.

Sockets: Sockets allow communication between two different processes on the same or different machines. A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client.

Firefox: Firefox browser is been used to act as a virtual host where it can send requests to the server. Any browser works but we used firefox for our compatibility.

Write permission: Write permission is the most important thing required in this project and the folder that contains the project file must be write enabled. Because without write permission you cannot access the system's ip-adressess or create the ports.

Major Task:

Setting up a proxy in the firefox web browser is the thing that is necessary as it should be accessed to the server as a client.

settings->advance proxy ip/localhost port 8080.

Handling multiple requests at a same time through different clients is another major task as it involves multithread at the server with a new thread with the same port for a client's request for the server.

```
# allowing up to 10 client connections
        self.server_socket.listen(10)
```

Storing the logs in the log file is been done in a text file “log.txt” where all the requests of the clients were stored specifically for the conformation that the requests of the clints were received. Generally a proxy server shouldn’t be storing the logs of users but here we store the requests to make sure and confirm that the connection is made properly and the requests are made properly.

```
logger_file_name = "log.txt"
```

Specifications:

Sockets : It helps us to connect a client to a server. Client is message sender and receiver and server is just a listener that works on data sent by client. Here multiple clients connect to the server of the same port using the sockets opened by the server.

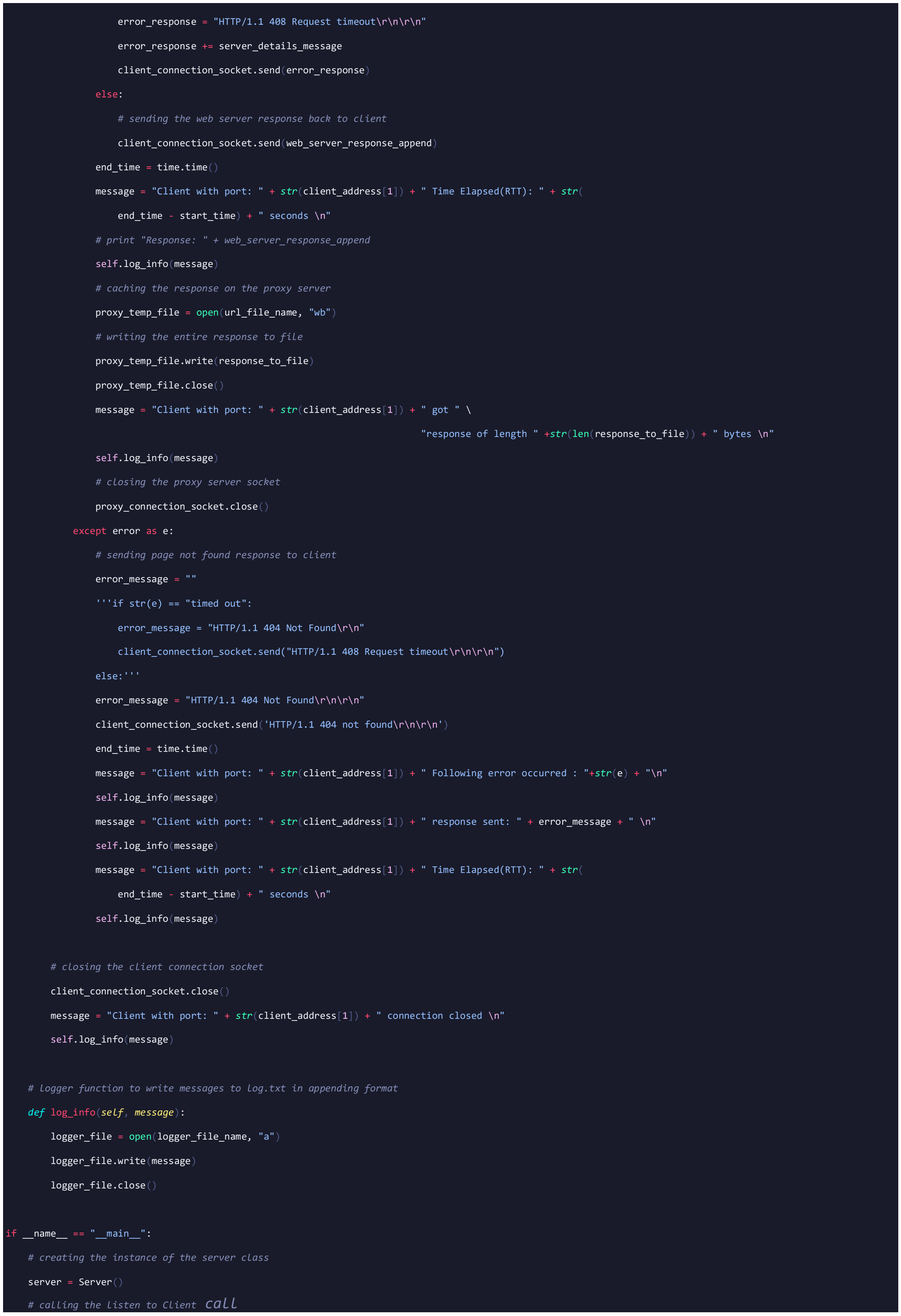
Thread handling: Multithreading in Python can be achieved by using the threading library. For invoking a thread, the caller thread creates a thread object and calls the start method on it. Once the join method is called, that initiates its execution and executes the run method of the class object.

File handling: Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file.

Source Code:

```
# proxy server with caching implementation in python

# python library for time, system and regular expression

import sys, time, re

# python library to handle threads

import threading

# python library to handle socket connection

from socket import *


# getting the port number from the user through command line

# checking if the arguments in the command line has more than one parameter

if len(sys.argv) == 2:

    # expecting the input to be of the form python server.py <port number>

    server_port = int(sys.argv[1])

else:

    # default port of the server is 8080 if the user doesn't supply any parameters

    server_port = 8080

logger_file_name = "log.txt"



class Server:


    def __init__(self):

        try:

            self.server_socket = socket(AF_INET, SOCK_STREAM)  # Create a TCP socket

            # AF_inet = IPv4 and SOCK_STREAM = TCP

            self.server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)  # Re-use the socket
        except error as e:
            print 'Unable to create/re-use the socket. Error: %s' % e
            message = 'Unable to create/re-use the socket. Error: %s' % e
            self.log_info(message)
        # bind the socket to a public/local host, and a port
        self.server_socket.bind(('', server_port))
        # allowing up to 10 client connections
        self.server_socket.listen(10)
        message = "Host Name: Localhost and Host address: 127.0.0.1 and Host port: " + str(server_port) + "\n"
        self.log_info(message)
        print "Server is ready to listen for clients"

    def listen_to_client(self):
        """ waiting for client to connect over tcp to the proxy server"""
        while True:
            # accepting client connection
            client_connection_socket, client_address = self.server_socket.accept()
            # printing the relevant client details on the server side
            client_details_log = "******************** Client Details:- ********************\n"
            client_details_log += "Client host name: "+str(client_address[0]) + "\nClient port number: "+str(client_address[1]) + "\n"
            client_socket_details = getaddrinfo(str(client_address[0]), client_address[1])
            client_details_log += "Socket family: "+str(client_socket_details[0][0]) + "\n"
            client_details_log += "Socket type: "+str(client_socket_details[0][1]) + "\n"
            client_details_log += "Socket protocol: "+str(client_socket_details[0][2]) + "\n"
            client_details_log += "Timeout: "+str(client_connection_socket.gettimeout()) + "\n"
            client_details_log += "*********************************************************\n"
            self.log_info(client_details_log)
            # Logging
            message = "Client IP address: "+str(client_address[0])+" and Client port number: "\
                      + str(client_address[1])+ "\n"
            self.log_info(message)
            # creating a new thread for every client
            d = threading.Thread(name=str(client_address), target=self.proxy_thread,
                                 args=(client_connection_socket, client_address))
            d.setDaemon(True)
            d.start()
        self.server_socket.close()

    def proxy_thread(self, client_connection_socket, client_address):
        """ method to create a new thread for every client connected """

        # starting the timer to calculate the elapsed time
        start_time = time.time()
        # getting the client request
        client_request = client_connection_socket.recv(1024)
        # if the request is not empty request i.e it contains some data
        if client_request:
            # getting request length
            request_length = len(client_request)
            message = "Client with port: " + str(client_address[1]) + " request length is " + str(
                request_length) + " bytes \n"
            self.log_info(message)

            message = "Client with port: " + str(client_address[1]) + " generated request: " + str(client_request).splitlines()[0] + " \n"
```

```python
        self.log_info(message)


        # Parsing the request line and headers sent by the client
        # since the request will be of the form GET http://www.abc.com HTTP/1.1 extracting the http part
        resp_part = client_request.split(' ')[0]
        if resp_part == "GET":
            http_part = client_request.split(' ')[1]
            # stripping the http part to get only the URL and removing the trailing / from the request
            double_slash_pos = str(http_part).find("//")
            url_connect = ""
            url_slash_check = list()
            url_slash = str()
            # if no http part to the url
            if double_slash_pos == -1:
                url_part = http_part[1:]
                # getting the www.abc.com part
                url_connect = url_part.split('/')[0]
            else:
                # if the url ends with / removing it e.g: www.example.com/
                if http_part.split('//')[1][-1] == "/":
                    url_part = http_part.split('//')[1][:-1]
                    # getting the www.abc.com part
                    url_connect = url_part.split('/')[0]
                else:
                    url_part = http_part.split('//')[1]
                    # getting the www.abc.com part
                    url_connect = url_part.split('/')[0]


            # getting the part after the host
            url_slash_check = url_part.split('/')[1:]
            url_slash = ""
            if url_slash_check:
                for path in url_slash_check:
                    url_slash += "/"
                    url_slash += path
            # checking if port number is provided
            client_request_port_start = str(url_part).find(":")
            # default port number
            port_number = 80
            # replacing all the non alphanumeric characters with under score
            url_file_name = re.sub('[^0-9a-zA-Z]+', '_', url_part)
            if client_request_port_start == -1:
                pass
            else:
                port_number = int(url_part.split(':')[1])
            self.find_file(url_file_name, client_connection_socket, port_number, client_address, start_time, url_connect, url_slash)
        else:
            # a call other than GET occurred
            message = "Client with port: " + str(client_address[1]) + " generated a call other than GET: " + resp_part + " \n"
            client_connection_socket.send("HTTP/1.1 405 Method Not Allowed\r\n\r\n")
            client_connection_socket.close()
            self.log_info(message)
            message = "HTTP/1.1 405 Method Not Allowed\r\n\r\n"
            self.log_info(message)
        else:
            # a blank request call was made by a client
            client_connection_socket.send("")
            client_connection_socket.close()
            message = "Client with port: " + str(client_address[1]) + " connection closed \n"
            self.log_info(message)

    def find_file(self, url_file_name, client_connection_socket, port_number, client_address, start_time, url_connect, url_slash):
        try:
            # getting the cached file for the url if it exists
            cached_file = open(url_file_name, "r")
            # reading the contents of the file
            message = "Client with port: " + str(client_address[1]) + ": Cache hit occurred" \
                                                                      "  for the request. Reading from file \n"
            self.log_info(message)
            # get proxy server details since the data is fetched from cache
            server_socket_details = getaddrinfo("localhost", port_number)
            server_details_message = "<body> Cached Server Details:- <br />"
            server_details_message += "Server host name: localhost <br /> Server port number: " + str(port_number) + " <br>"
            server_details_message += "Socket family: " + str(server_socket_details[0][0]) + "<br>"
            server_details_message += "Socket type: " + str(server_socket_details[0][1]) + "<br>"
            server_details_message += "Socket protocol: " + str(server_socket_details[0][2]) + "<br>"
            server_details_message += "Timeout: " + str(client_connection_socket.gettimeout()) + "<br> </body>"
            response_message = ""
            # print "reading data line by line and appending it"
            with open(url_file_name) as f:
                for line in f:
                    response_message += line
            # print 'finished reading the data'
            # appending the server details message to the response
            response_message += server_details_message
            # closing the file handler
            cached_file.close()
            # sending the cached data

            client_connection_socket.send(response_message)
            end_time = time.time()
            message = "Client with port: " + str(client_address[1]) + ": Response Length: " + str(len(response_message)) + " bytes\n"
            self.log_info(message)
            message = "Client with port: " + str(client_address[1]) + " Time Elapsed(RTT): "+str(end_time - start_time) + " seconds \n"
            self.log_info(message)

        except IOError as e:
            message = "Client with port: " + str(client_address[1]) + " Cache miss occurred " \
                                                                      "for the request. Hitting web server \n"
            self.log_info(message)
            """there is no cached file for the specified URL,
            so we need to fetch the URL from the proxy server and cache it
            To get the URL we need to create a socket on proxy machine"""
            proxy_connection_socket = None
            try:
                # creating the socket from the proxy server
                proxy_connection_socket = socket(AF_INET, SOCK_STREAM)
                # setting time out so that after last packet if not other packet comes socket will auto close
            # in 2 seconds
        except error as e:
            print 'Unable to create the socket. Error: %s' % e
            message = 'Unable to create the socket. Error: %s' % e
            self.log_info(message)
        try:
            proxy_connection_socket.settimeout(2)
            # connecting to the url specified by the client
            proxy_connection_socket.connect((url_connect, port_number))
            # sending GET request from client to the web server
            web_request = str()
            if url_slash:
                web_request = b"GET /" + url_slash[1:] + " HTTP/1.1\nHost: " + url_connect + "\n\n"
            else:
                web_request = b"GET / HTTP/1.1\nHost: " + url_connect + "\n\n"

            # print "GET web request: "+web_request
            proxy_connection_socket.send(web_request)
            message = "Client with port: " + str(client_address[1]) + " generated request of length " \
                                                                      "to web server "+str(len(web_request))+ " bytes \n"
            self.log_info(message)
            message = "Client with port: " + str(client_address[1]) + " generated request " \
                                                                      "to web server as: "+str(web_request) + " \n"
            self.log_info(message)
            # getting the web server response which is expected to be a file
            server_socket_details = getaddrinfo(url_connect, port_number)
            server_details_message = "<body> Web Server Details:- <br />"
            server_details_message += "Server host name: " + url_connect + " <br /> Server port number: " + str(port_number) + "<br />"
            server_details_message += "Socket family: " + str(server_socket_details[0][0]) + "<br />"
            server_details_message += "Socket type: " + str(server_socket_details[0][1]) + "<br />"
            server_details_message += "Socket protocol: " + str(server_socket_details[0][2]) + "<br />"
            server_details_message += "Timeout: " + str(client_connection_socket.gettimeout()) + "<br /> </body>"
            web_server_response_append = ""
            # to get server response in loop until zero data or timeout of 2 seconds is reached
            timeout_flag = False
            while True:
                try:
                    web_server_response = proxy_connection_socket.recv(4096)
                except timeout:
                    # a time out occurred on waiting for server response so break out of loop
                    if len(web_server_response_append) <= 0:
                        timeout_flag = True
                    break
                if len(web_server_response) > 0:
                    web_server_response_append += web_server_response
                else:
                    # all the data has been received so break out of the loop
                    break
            # variable to store response to file
            response_to_file = web_server_response_append
            # appending web server details to the response sent to client
            web_server_response_append += server_details_message
            if timeout_flag:
                # got bored waiting for response
```



```
                    error_response = "HTTP/1.1 408 Request timeout\r\n\r\n"
                    error_response += server_details_message
                    client_connection_socket.send(error_response)
                else:
                    # sending the web server response back to client
                    client_connection_socket.send(web_server_response_append)
                end_time = time.time()
                message = "Client with port: " + str(client_address[1]) + " Time Elapsed(RTT): " + str(
                    end_time - start_time) + " seconds \n"
                # print "Response: " + web_server_response_append
                self.log_info(message)
                # caching the response on the proxy server
                proxy_temp_file = open(url_file_name, "wb")
                # writing the entire response to file
                proxy_temp_file.write(response_to_file)
                proxy_temp_file.close()
                message = "Client with port: " + str(client_address[1]) + " got " \
                                                                          "response of length " +str(len(response_to_file)) + " bytes \n"
                self.log_info(message)
                # closing the proxy server socket
                proxy_connection_socket.close()
            except error as e:
                # sending page not found response to client
                error_message = ""
                '''if str(e) == "timed out":
                    error_message = "HTTP/1.1 404 Not Found\r\n"
                    client_connection_socket.send("HTTP/1.1 408 Request timeout\r\n\r\n")
                else:'''
                error_message = "HTTP/1.1 404 Not Found\r\n\r\n"
                client_connection_socket.send('HTTP/1.1 404 not found\r\n\r\n')
                end_time = time.time()
                message = "Client with port: " + str(client_address[1]) + " Following error occurred : "+str(e) + "\n"
                self.log_info(message)
                message = "Client with port: " + str(client_address[1]) + " response sent: " + error_message + " \n"
                self.log_info(message)
                message = "Client with port: " + str(client_address[1]) + " Time Elapsed(RTT): " + str(
                    end_time - start_time) + " seconds \n"
                self.log_info(message)

        # closing the client connection socket
        client_connection_socket.close()
        message = "Client with port: " + str(client_address[1]) + " connection closed \n"
        self.log_info(message)

    # logger function to write messages to log.txt in appending format
    def log_info(self, message):
        logger_file = open(logger_file_name, "a")
        logger_file.write(message)
        logger_file.close()

if __name__ == "__main__":
    # creating the instance of the server class
    server = Server()
    # calling the listen to Client call
    server.listen_to_client()
```

Conclusion: We have achieved a multi-threaded proxy server with the localhost: 127.0.0.1/google.com as the client.

Host is recorded as: Host Name: Localhost and Host address: 127.0.0.1 and Host port: 8080

```
******************** Client Details:- ********************
Client host name: 127.0.0.1
Client port number: 64835
Socket family: AddressFamily.AF_INET
Socket type: 0
Socket protocol: 0
Timeout: None
*****************************************************
helloClient IP address: 127.0.0.1 and Client port number: 64835
Client with port: 64835 request length is 233 bytes
Client with port: 64835 generated request: b'CONNECT getpocket.cdn.mozilla.net:443 HTTP/1.1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0\r\nProxy-Connection: keep-alive\r\nConnection: keep-alive\r\nHost: getpocket.cdn.mozilla.net:443\r\n\r\n'
```