Step One: Determine Requirements

1. 200 OK Requirements: a. use GET method b. the requested file exists on the server c. the protocol is "HTTP/1.1"

Message part: the request line e.g. GET /test.html HTTP/1.1\r\n command + url + version

Test HTTP request: GET /test.html HTTP/1.1\r\n Host: localhost:8088\r\n

2. 304 Not Modified Requirements: a. there is "If-Modified-Since" message in the header file b. the requested file exists on the server c. compare the time in the header file and the last modified time from the file if the time from the header >= file last modified time, then it is not modified

Message part: in the header file, the line starts with "If-Modified-Since", and with a time

Test HTTP request: GET /test.html HTTP/1.1\r\n Host: localhost:8088\r\n If-Modified-Since: Tue, 14 Oct 2024 10:00:00 GMT\r\n\r\n

3. 400 Bad request Requirements: a. malformed request syntax (the number of elements in the request line is not 3) b. invalid request message framing c. the protocol is not "HTTP/1.1"

Message part: the request line e.g. GET /test.html HTTP/1.1\r\n command + url + version

Test HTTP request: GET /test.html HTTP/1.0\r\n Host: localhost:8088\r\n

4. 404 Not Found Requirements: the requested file does not exist on the server

Message part: the url in the request line

Test HTTP request: GET /notExist.html HTTP/1.1\r\n Host: localhost:8088\r\n

5. 501 Not Implemented Requirements: the method is not GET

Message part: the first element in the request line (command)

Test HTTP request: POST /test.html HTTP/1.1\r\n Host: localhost:8088\r\n

Step Two: Build Your Minimal Web Server & Test

a) The combination of HTTPfunctions.py and TCPServer.py make up our server Our server can only respond to GET requests, all other requests will be answered as 501 Not Implement.

b) Using the link in a web browers works, while the server is running of course.

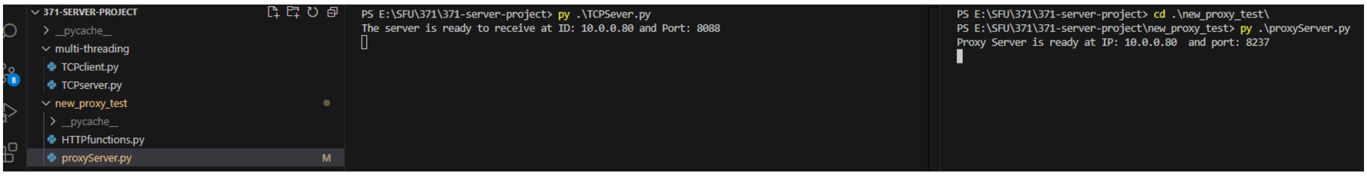
c) It the provided servertests.sh are curl commands that test our server functionallity you can run the test on lunix by running the command: `bash servertests.sh [IP_address_of_server] [Port] [url]` the bash scripts output into files needed for the functionallity they test. We have submited what those file when we ran the script.

Step Three: Performance

a) A proxy server needs to be able to receive HTTP requests and redirect them to the correct destination get the response from the correct server and return that response to the original client.

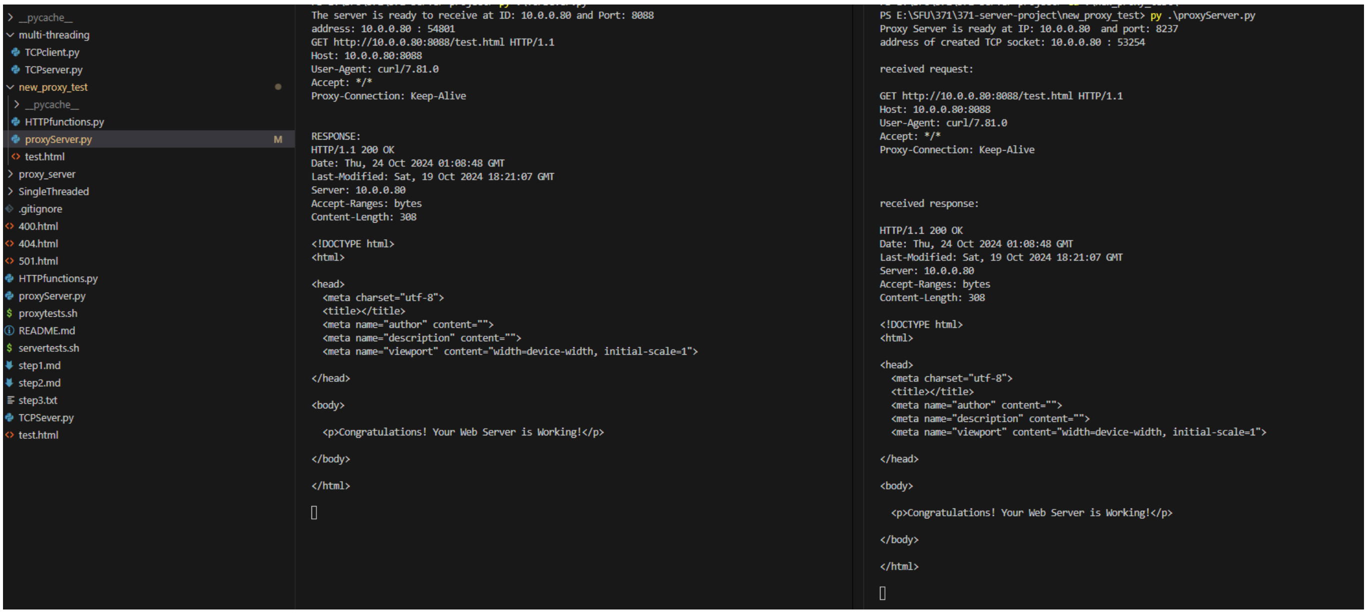
If the cache of the proxy server does not have the data which the client requested, The proxy server will fetch the hostname and port number from the client's request and create a socket connection with the web server, then send the request from the client to the web server. Then the web server will handle the request and send response and requested data back to proxy server, then the proxy server will send the response and requested data back to client. If the reponse from the web server is 200 OK, then the proxy server will cache the data. If other clients request for the data that is in the cache, the proxy server will not send requests to the web server, instead, it will find the data in the cache and handle client's request, then send back response and requested data back to the server

b) If you run this curl command: `curl -i -x 10.0.0.80:8237 10.0.0.80:8088/test.html` When the proxy server does not have the requested file cached this will be the result



```
PS E:\SFU\371\371-server-project> py .\TCPSever.py
The server is ready to receive at ID: 10.0.0.80 and Port: 8088
```

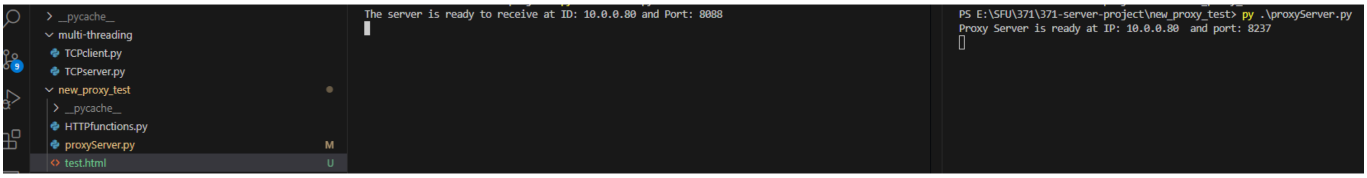
```
PS E:\SFU\371\371-server-project> cd .\new_proxy_test\
PS E:\SFU\371\371-server-project\new_proxy_test> py .\proxyServer.py
Proxy Server is ready at IP: 10.0.0.80  and port: 8237
```



```
The server is ready to receive at ID: 10.0.0.80 and Port: 8088
address: 10.0.0.80 : 54801
GET http://10.0.0.80:8088/test.html HTTP/1.1
Host: 10.0.0.80:8088
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive


RESPONSE:
HTTP/1.1 200 OK
Date: Thu, 24 Oct 2024 01:08:48 GMT
Last-Modified: Sat, 19 Oct 2024 18:21:07 GMT
Server: 10.0.0.80
Accept-Ranges: bytes
Content-Length: 308

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

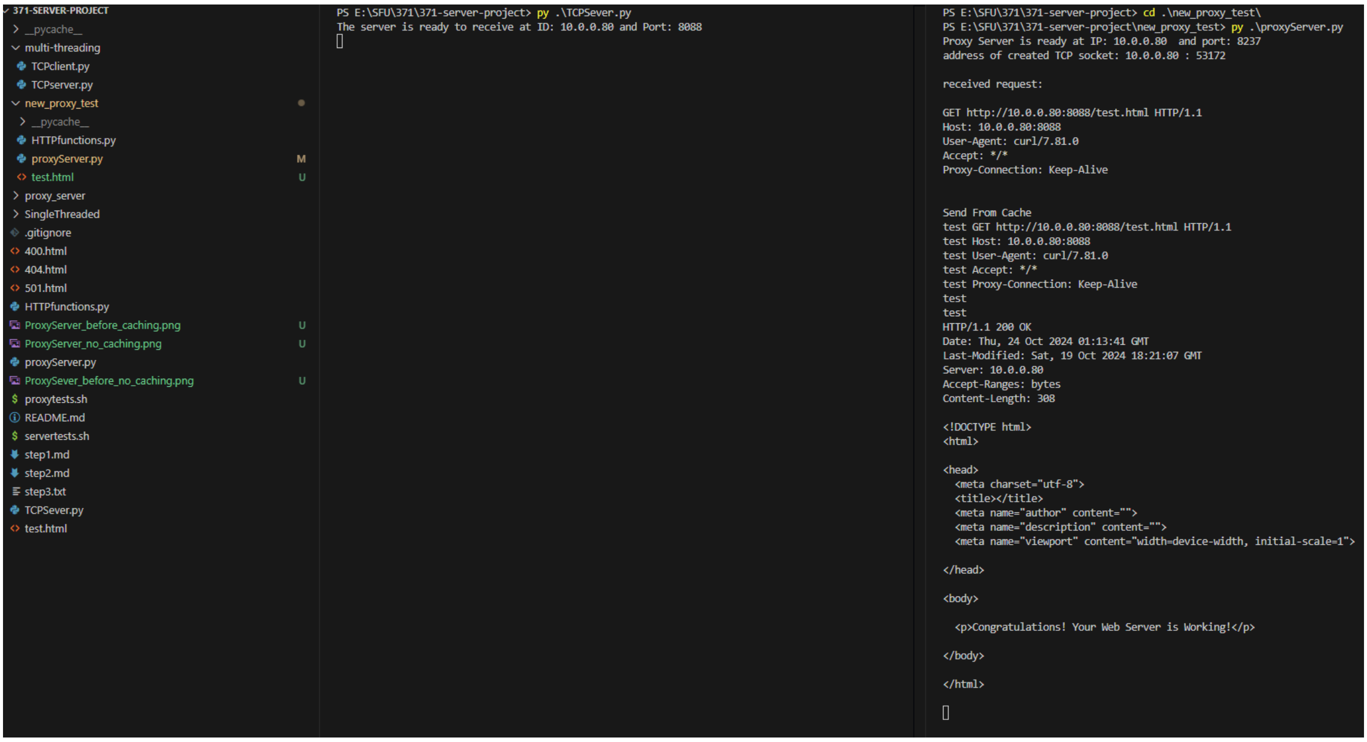
```
PS E:\SFU\371\371-server-project\new_proxy_test> py .\proxyServer.py
Proxy Server is ready at IP: 10.0.0.80  and port: 8237
address of created TCP socket: 10.0.0.80 : 53254

received request:

GET http://10.0.0.80:8088/test.html HTTP/1.1
Host: 10.0.0.80:8088
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive



received response:

HTTP/1.1 200 OK
Date: Thu, 24 Oct 2024 01:08:48 GMT
Last-Modified: Sat, 19 Oct 2024 18:21:07 GMT
Server: 10.0.0.80
Accept-Ranges: bytes
Content-Length: 308

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

When the proxy server does have the requested file cached this will be the result



```
The server is ready to receive at ID: 10.0.0.80 and Port: 8088
```

```
PS E:\SFU\371\371-server-project\new_proxy_test> py .\proxyServer.py
Proxy Server is ready at IP: 10.0.0.80  and port: 8237
```



```
371-SERVER-PROJECT
  > __pycache__
  v multi-threading
    TCPclient.py
    TCPserver.py
  v new_proxy_test
    > __pycache__
    HTTPfunctions.py
    proxyServer.py                        M
    test.html                             U
  > proxy_server
  > SingleThreaded
  .gitignore
  400.html
  404.html
  501.html
  HTTPfunctions.py
  ProxyServer_before_caching.png          U
  ProxyServer_no_caching.png              U
  proxyServer.py
  ProxySever_before_no_caching.png        U
  proxytests.sh
  README.md
  servertests.sh
  step1.md
  step2.md
  step3.txt
  TCPSever.py
  test.html
```

```
PS E:\SFU\371\371-server-project> py .\TCPSever.py
The server is ready to receive at ID: 10.0.0.80 and Port: 8088
```

```
PS E:\SFU\371\371-server-project> cd .\new_proxy_test\
PS E:\SFU\371\371-server-project\new_proxy_test> py .\proxyServer.py
Proxy Server is ready at IP: 10.0.0.80  and port: 8237
address of created TCP socket: 10.0.0.80 : 53172

received request:

GET http://10.0.0.80:8088/test.html HTTP/1.1
Host: 10.0.0.80:8088
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive


Send From Cache
test GET http://10.0.0.80:8088/test.html HTTP/1.1
test Host: 10.0.0.80:8088
test User-Agent: curl/7.81.0
test Accept: */*
test Proxy-Connection: Keep-Alive
test
test
HTTP/1.1 200 OK
Date: Thu, 24 Oct 2024 01:13:41 GMT
Last-Modified: Sat, 19 Oct 2024 18:21:07 GMT
Server: 10.0.0.80
Accept-Ranges: bytes
Content-Length: 308

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="author" content="">
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

  <p>Congratulations! Your Web Server is Working!</p>

</body>

</html>
```

c) We initially created the server single threaded and then implement multi-threaded. But quickly changed it beacuse the server needs to be able to respond to multiple clients at a time. When you accept a connecting you make a thread to handle the connection when that specific connection ends you close the connection and the thread. Being multithreaded alows for better performance because at some points your server might be waiting for the client, during that time it can service other clients but only if it is multithreaded. It also allows the server to be almost continuouly accepting new connections from clients. Our server has a max backlog of clients to be connected of 5.