CMPT 371 Mini-Project: Web & Web Proxy Server

Rana Hoshyarsadeghi
Alvin Tsang

GitHub Repo: https://github.com/Alvicorn/CMPT_371_Mini-Project.git

Notes

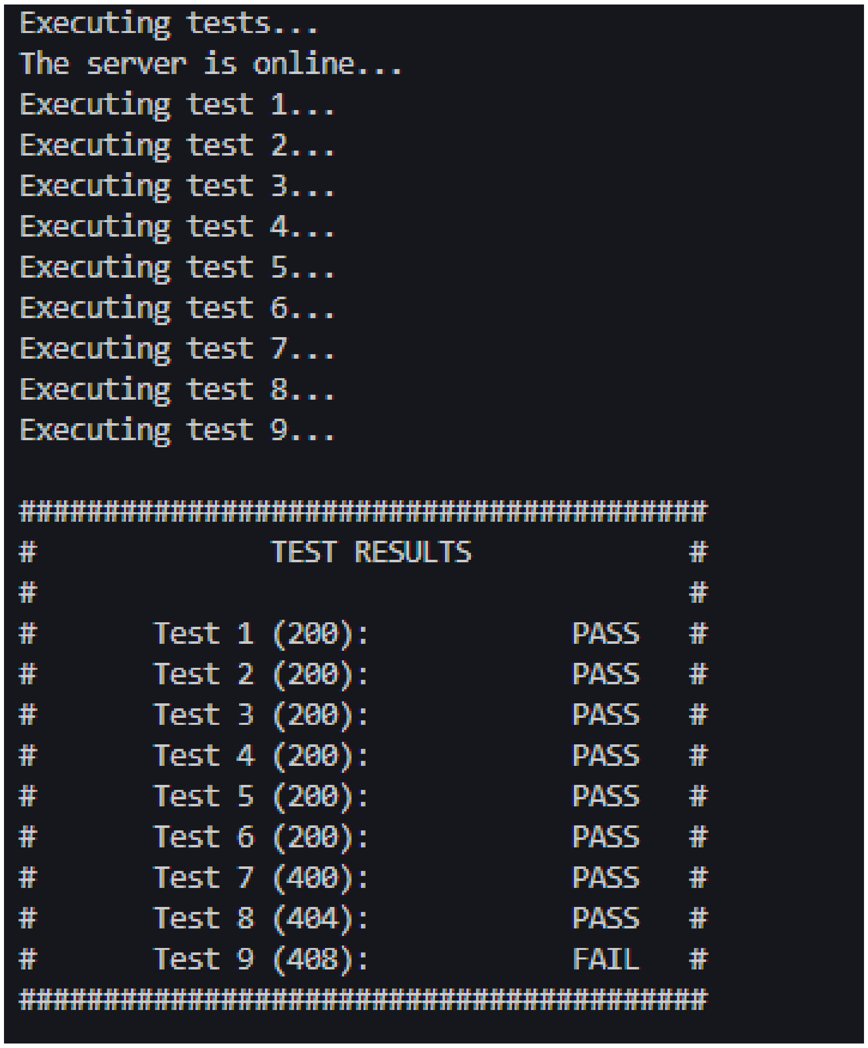
- All servers and the proxy uses the user's IP address as the host
- Server runs on port 12000
- Proxy runs on port 12001
- Multi-threaded server runs on port 12001
- Developed with Python 3.9

Part 1: Server Implementation

The server is a basic TCP web server that uses HTTP to handle and respond to client requests. The following HTTP codes were implemented. The server handles each request one at a time.

Code	Message
200	OK
400	Bad request
404	Not Found

The web server was tested in the browser by sending requests as well as a test script (TestServer.py). Below are the outputs from the tests and a short description of each test



```
Executing tests...
The server is online...
Executing test 1...
Executing test 2...
Executing test 3...
Executing test 4...
Executing test 5...
Executing test 6...
Executing test 7...
Executing test 8...
Executing test 9...

##########################################
#               TEST RESULTS             #
#                                        #
#       Test 1 (200):           PASS     #
#       Test 2 (200):           PASS     #
#       Test 3 (200):           PASS     #
#       Test 4 (200):           PASS     #
#       Test 5 (200):           PASS     #
#       Test 6 (200):           PASS     #
#       Test 7 (400):           PASS     #
#       Test 8 (404):           PASS     #
#       Test 9 (408):           FAIL     #
##########################################
```

\

Test Number	Description
1	Create a connection with the server. Send a request for *test.html* and get a response. The expected response was 200.
2	Create a connection with the server. Send a request for *test1.html* and get a response. The expected response was 200.
3	Create a connection with the server. Send a request for *test2.html* and get a response. The expected response was 200.
4	Create a connection with the server. Send a request for *test3.html* and get a response. The expected response was 200.
5	Create a connection with the server. Send a request for *test4.html* and get a response. The expected response was 200.
6	Create a connection with the server. Send a request for *test5.html* and get a response. The expected response was 200.
7	Create a connection with the server. Send a request for *test.htm* and get a response. The expected response was 400 as the request format for the file is invalid
8	Create a connection with the server. Send a request for *notFound.html* and get a response. The expected response was 404 as the file name does not exist in the *Files* folder
9	Create a connection with the server. The expected response was 408.

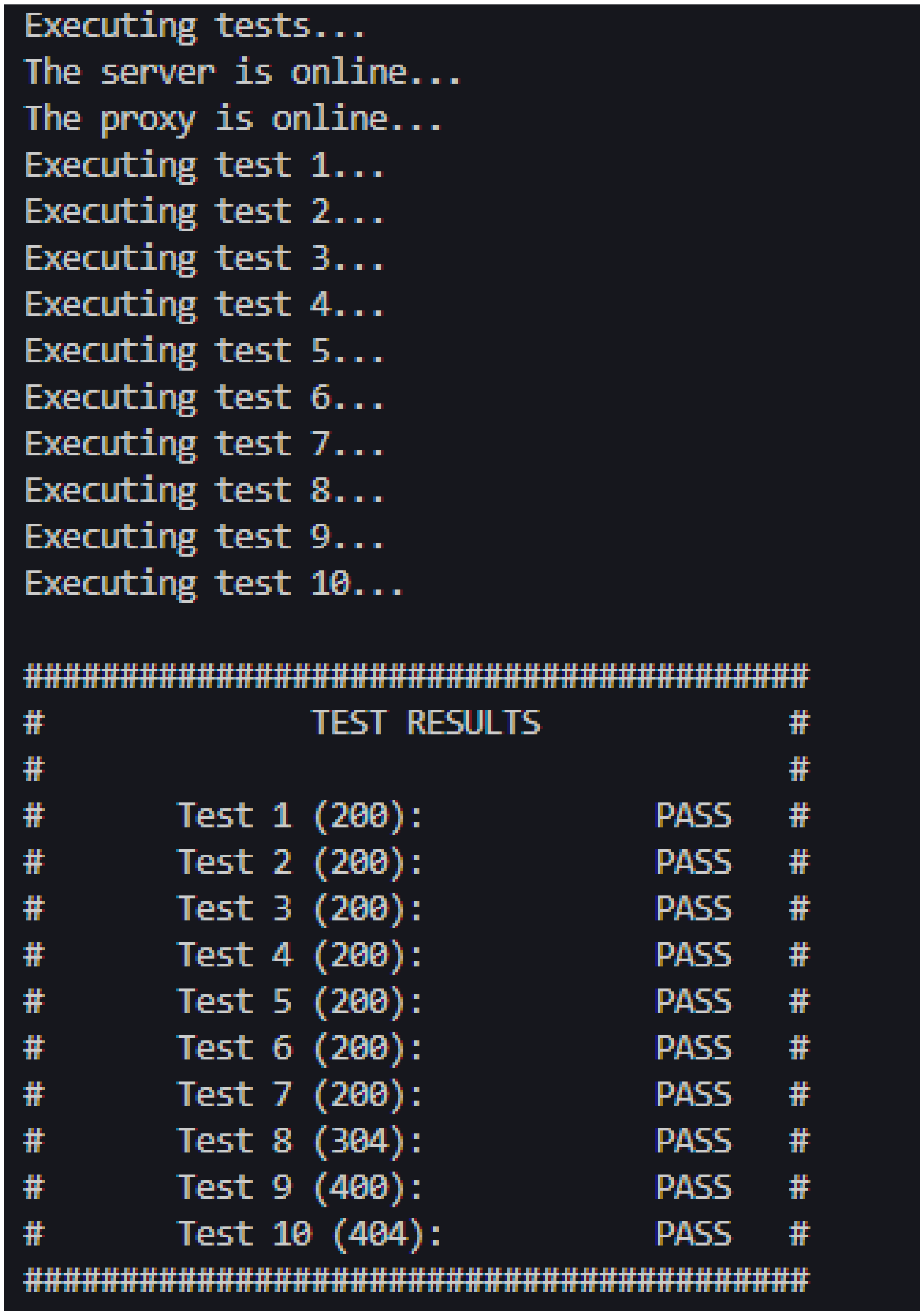
Part 2: Proxy Implementation

The proxy is a basic TCP proxy server that uses HTTP to handle and respond to client requests. The following HTTP codes were implemented. The proxy handles each request one at a time. For this project, the proxy server runs on the same machine as the web server, just on different ports. The benefits of caching were simulated by having all the cached files stored in the *Cache* folder. This cache folder is only created when necessary and will be updated by the proxy. There is also an info file name *files.json*. This file stores information such as the number of cached items, the file path to a cached item, when it was added to the cache and when it was last modified.
One would want to have a cache because handling requests from the proxy server is generally faster. The proxy server is closer to the end user compared to the web server. Therefore it is able to respond faster and reduce traffic on the web server. See below for the specifications:

- For simplicity, the cache size is set to 5
- Cached items will be stored in a folder named *Cache*
- Eviction policy: the oldest cached item is evicted to make space for the new item
- On a cache hit…
    - Check if it has been modified based on a conditional-Get
        - If modified, return the new data (200)
        - Otherwise return the original data (304)
    - Client → Proxy → Client
- On a cache miss..
    - Send a request to the web server.
        - If there is a negative response from the web server, redirect the message to the client
        - Otherwise, save the data in the cache and respond to the client
        - Apply the eviction policy when the cache is full
    - Client → Proxy → Web Server → Proxy → Client
- No changes are required on the client side as the implementation should be hidden from the client
- The following HTTP codes have been implemented

Code	Message
200	OK
400	Bad request
404	Not Found

The web server was tested in the browser by sending requests as well as a test script (TestProxy.py). Below are the outputs from the tests and a short description of each test



```
Executing tests...
The server is online...
The proxy is online...
Executing test 1...
Executing test 2...
Executing test 3...
Executing test 4...
Executing test 5...
Executing test 6...
Executing test 7...
Executing test 8...
Executing test 9...
Executing test 10...

##########################################
#              TEST RESULTS              #
#                                        #
#       Test 1 (200):           PASS     #
#       Test 2 (200):           PASS     #
#       Test 3 (200):           PASS     #
#       Test 4 (200):           PASS     #
#       Test 5 (200):           PASS     #
#       Test 6 (200):           PASS     #
#       Test 7 (200):           PASS     #
#       Test 8 (304):           PASS     #
#       Test 9 (400):           PASS     #
#       Test 10 (404):          PASS     #
##########################################
```

Test Number	Description
1	Create a connection with the proxy. Send a request for *test.html* to the server and get a response. *file.html* is stored in the cache. The expected response was 200.
2	Create a connection with the proxy. Send a request for *test1.html* to the server and get a response. *file.html1* is stored in the cache. The expected response was 200.
3	Create a connection with the server. Send a request for *test2.html* to the server and get a response. *file2.html* is stored in the cache. The expected response was 200.
4	Create a connection with the server. Send a request for *test4.html* to the server and get a response. *file4.html* is stored in the cache. The expected response was 200.
5	Create a connection with the server. Send a request for *test5.html* to the server and get a response. *test.html* is evicted and *file5.html* is stored in the cache. The expected response was 200.
6	Create a connection with the proxy. Send a request for *test5.html* and get a response from the proxy. The expected response was 200.
7	Create a connection with the proxy. Send a request for *test5.html,* check if it has been updated since 2023 and get a response from the proxy. The expected response was 200.
8	Create a connection with the proxy. Send a request for *test5.html,* check if it has been updated since 2021 and get a response from the proxy. The expected response was 304.
9	Create a connection with the proxy. Send a request for *test.htm* and get a response. The expected response was 400 as the request format for the file is invalid
10	Create a connection with the server. Send a request for

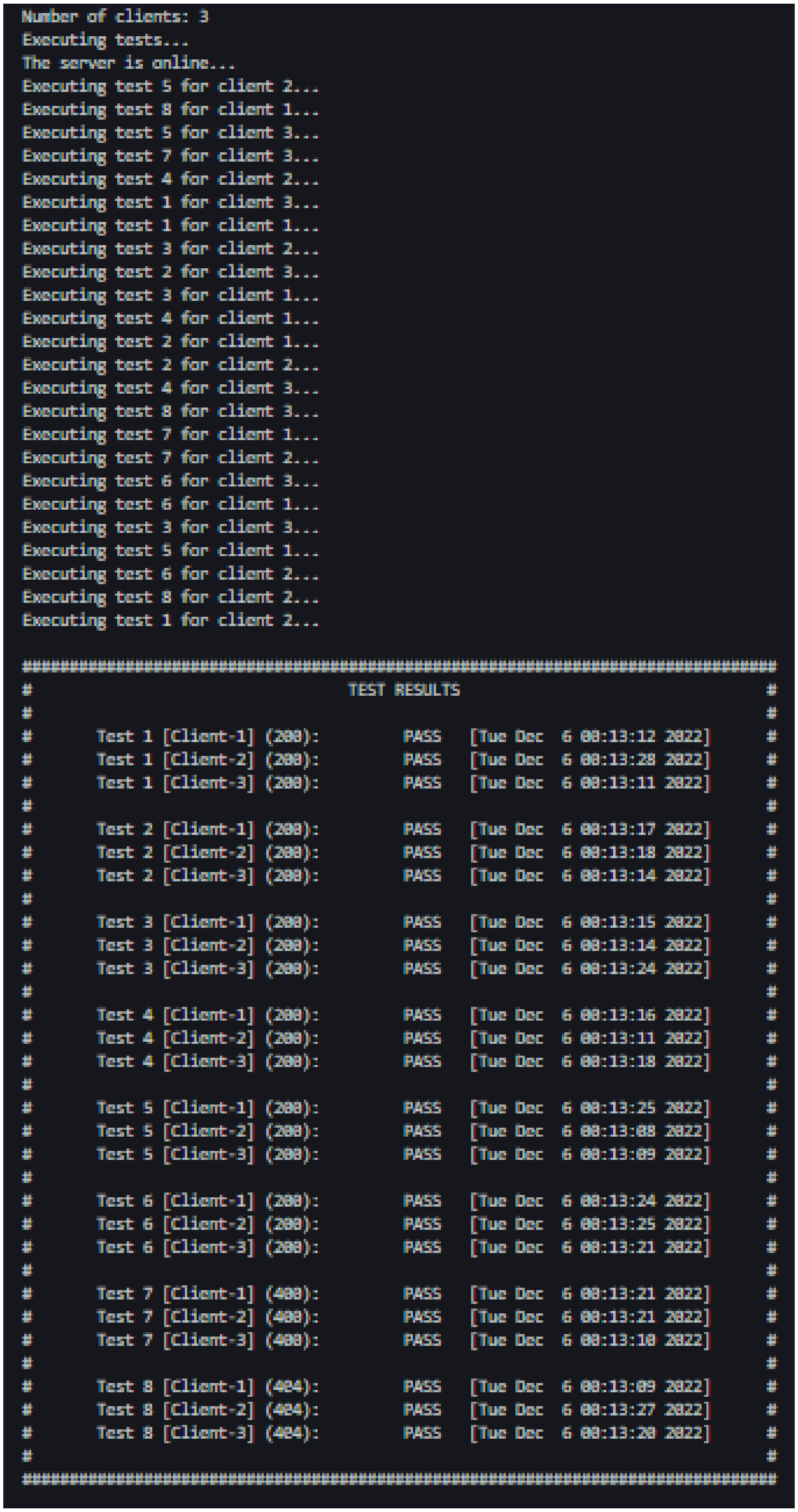
	notFound.html and get a response. The expected response was 404 as the file name does not exist in the *Files* folder

Part 3: Multi-Threaded Server

The server is a TCP web server that uses HTTP to handle and respond to client requests. The following HTTP codes were implemented. The server is multi-threaded and can handle multiple client requests. The main thread is listening at a fixed port number (12002). With each request, it is handled by a child thread.

Code	Message
200	OK
400	Bad request
404	Not Found

The web server was tested in the browser by sending requests as well as a test script (TestMultiTheadServer.py). Below are the outputs from the tests and a short description of each test. Note that the test script will take input from the user to replicate multiple clients communicating with the server. It is recommended that the input be kept small as the processing time increases quite rapidly. Also, note that the test executes the order and client numbers are randomized to simulate the unpredictability of multiple users.



```
Number of clients: 3
Executing tests...
The server is online...
Executing test 5 for client 2...
Executing test 8 for client 1...
Executing test 5 for client 3...
Executing test 7 for client 3...
Executing test 4 for client 2...
Executing test 1 for client 3...
Executing test 1 for client 1...
Executing test 3 for client 2...
Executing test 2 for client 3...
Executing test 3 for client 1...
Executing test 4 for client 1...
Executing test 2 for client 1...
Executing test 2 for client 2...
Executing test 4 for client 3...
Executing test 8 for client 3...
Executing test 7 for client 1...
Executing test 7 for client 2...
Executing test 6 for client 3...
Executing test 6 for client 1...
Executing test 3 for client 3...
Executing test 5 for client 1...
Executing test 6 for client 2...
Executing test 8 for client 2...
Executing test 1 for client 2...

##################################################################################
#                                  TEST RESULTS                                  #
#                                                                                #
#       Test 1 [Client-1] (200):         PASS   [Tue Dec  6 00:13:12 2022]       #
#       Test 1 [Client-2] (200):         PASS   [Tue Dec  6 00:13:28 2022]       #
#       Test 1 [Client-3] (200):         PASS   [Tue Dec  6 00:13:11 2022]       #
#                                                                                #
#       Test 2 [Client-1] (200):         PASS   [Tue Dec  6 00:13:17 2022]       #
#       Test 2 [Client-2] (200):         PASS   [Tue Dec  6 00:13:18 2022]       #
#       Test 2 [Client-3] (200):         PASS   [Tue Dec  6 00:13:14 2022]       #
#                                                                                #
#       Test 3 [Client-1] (200):         PASS   [Tue Dec  6 00:13:15 2022]       #
#       Test 3 [Client-2] (200):         PASS   [Tue Dec  6 00:13:14 2022]       #
#       Test 3 [Client-3] (200):         PASS   [Tue Dec  6 00:13:24 2022]       #
#                                                                                #
#       Test 4 [Client-1] (200):         PASS   [Tue Dec  6 00:13:16 2022]       #
#       Test 4 [Client-2] (200):         PASS   [Tue Dec  6 00:13:11 2022]       #
#       Test 4 [Client-3] (200):         PASS   [Tue Dec  6 00:13:18 2022]       #
#                                                                                #
#       Test 5 [Client-1] (200):         PASS   [Tue Dec  6 00:13:25 2022]       #
#       Test 5 [Client-2] (200):         PASS   [Tue Dec  6 00:13:08 2022]       #
#       Test 5 [Client-3] (200):         PASS   [Tue Dec  6 00:13:09 2022]       #
#                                                                                #
#       Test 6 [Client-1] (200):         PASS   [Tue Dec  6 00:13:24 2022]       #
#       Test 6 [Client-2] (200):         PASS   [Tue Dec  6 00:13:25 2022]       #
#       Test 6 [Client-3] (200):         PASS   [Tue Dec  6 00:13:21 2022]       #
#                                                                                #
#       Test 7 [Client-1] (400):         PASS   [Tue Dec  6 00:13:21 2022]       #
#       Test 7 [Client-2] (400):         PASS   [Tue Dec  6 00:13:21 2022]       #
#       Test 7 [Client-3] (400):         PASS   [Tue Dec  6 00:13:10 2022]       #
#                                                                                #
#       Test 8 [Client-1] (404):         PASS   [Tue Dec  6 00:13:09 2022]       #
#       Test 8 [Client-2] (404):         PASS   [Tue Dec  6 00:13:27 2022]       #
#       Test 8 [Client-3] (404):         PASS   [Tue Dec  6 00:13:20 2022]       #
#                                                                                #
##################################################################################
```

Test Number	Description
1	Create a connection with the server. Send a request for *test.html* and get a response. The expected response was 200.
2	Create a connection with the server. Send a request for *test1.html* and get a response. The expected response was 200.
3	Create a connection with the server. Send a request for *test2.html* and get a response. The expected response was 200.
4	Create a connection with the server. Send a request for *test3.html* and get a response. The expected response was 200.
5	Create a connection with the server. Send a request for *test4.html* and get a response. The expected response was 200.
6	Create a connection with the server. Send a request for *test5.html* and get a response. The expected response was 200.
7	Create a connection with the server. Send a request for *test.htm* and get a response. The expected response was 400 as the request format for the file is invalid
8	Create a connection with the server. Send a request for *notFound.html* and get a response. The expected response was 404 as the file name does not exist in the *Files* folder