

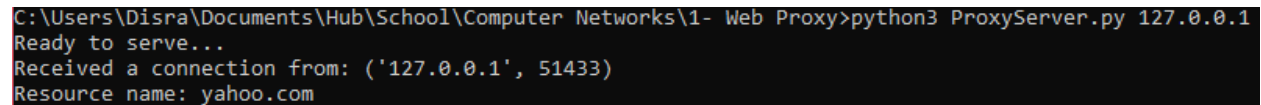
## Assignment 1 – Web Proxy

### Problem

For this assignment we were to create a simple web proxy server that would receive a request then verify the existence of the page in cache, then take the appropriate steps to create the cache and send the page to the client browser.

### Task 1 – Extract requested webpage

1a. When running the script, the server will begin listening for a connection. By entering the code <http://127.0.0.1:8888/yahoo.com> into the browser, the script received the connection and printed out the requested resource, shown on image 1.



```
C:\Users\Disra\Documents\Hub\School\Computer Networks\1- Web Proxy>python3 ProxyServer.py 127.0.0.1
Ready to serve...
Received a connection from: ('127.0.0.1', 51433)
Resource name: yahoo.com
```

Image 1

### Task 2 - Sending GET request to the Web Server and returning the response back to client

In the logic of the script, after it fails to open a requested cache file inside the try block, the program will cause an exception where the connection to the page will happen. This exception block is shown in image 2, line 69.

2a. From lines 75 to 84 in image 2, I created a new socket that uses port 80 to connect to the variable 'host', which was the URL derived from the initial GET request. This request is in the form of 'www.hostname.com'.

2b. After connecting, the script then sends 'get\_req' to the new connection using the sendall() method and encoding the string. Here 'get\_req' has the form of 'GET URL HTTP/1.0\n\n'.

Note: This is the section I found the most confusing. The makefile method opens the file for reading then proceeds to attempt to write to it. I attempted to use the line as it, as well as changing it to 'w' or 'b' mode but kept running into issues. As I thought about it, I didn't see a reason to write the GET request to a file then immediately read from it if we were going to store the returned page into cache. Doing so would create more files in the cache. The way my code was designed, the GET request can always be recreated from the extracted URL. In the end I found it easier to store that GET request in the variable 'get\_req' and use it that way. Later on, when writing the returned page from yahoo into a cache file, I needed to use the 'wb' mode to avoid formatting issues when writing to and reading from the cache file. But I had to use Python's built in file object as opposed to socket.makefile() as the latter does not support the mixed modes 'rb' and 'wb'. In retrospect, I could have easily used this method to store the GET request as a file but for the reason described above of being able to recreate the GET request

and finding it more space-efficient to avoid storing them, I justified keeping the GET request as a variable instead. I hope that is OK.

In this code section, I implemented another try-except section to catch errors. On line 99, I created an IOError exception to inform the user of an error related to writing to file when creating a cache file. After that, in line 101, I set a general exception that would catch an error that was not an IOError (as that one is checked for first) and is instead related to the rest of the code in that block which is all related to the connection to the page. In effect, the last exception will catch any errors when connecting to the page.

**2c.** In image 2, line 83, the script will receive the requested page and store it in the 'res\_message' (stands for resource message) variable. After closing the socket and checking 'res\_message' for an empty response, the script sends the requested page back to the browser client on line 92 using the corresponding socket and the sendall() method. It will then proceed to write the 'res\_message' to a cache file. The last two steps were coded in that order to prioritize the response to the client before creating the cache. If for whatever reason the script managed to send the response back to the client yet failed to write to file, the worst that would happen would be that the next time the script received the same request, it would not be able to use the cache and would attempt to recreate it once more.

Assuming the cache file was previously created, if the script received the same request to the yahoo page, the program would run the code in the try block on line 54, shown on image 3, opening the requested cache file without throwing an exception and finally sending that information back to the client, shown on line 65, image 3.

An overview of the whole process, the script creating a cache the first time then sending that cache the next time it is requested, is shown on image 4.

```

68 # Error handling for file not found in cache, need to talk to origin server
69 except IOError:
70     print('Resource not in cache')
71     if fileExist == False:
72         #FILL IN HERE...
73         try: ##Creating Request to resource##
74
75             #Connecting
76             res_sock = socket(AF_INET, SOCK_STREAM)
77             res_sock.connect((host, 80))
78
79             #Send request
80             print('Fetching page..')
81             res_sock.sendall(get_req.encode())
82
83             res_message = res_sock.recv(bufferSize)
84             res_sock.close()
85
86             if res_message == '':
87                 raise Exception('Error: No data received')
88                 break
89
90             #Sending requested page to client
91             print('Sending page to client..')
92             tcpCliSock.sendall(res_message)
93
94             #Open file for writing or create if doesn't exist
95             print('Writing page to cache..')
96             with open(filetouse, 'wb') as f:
97                 f.write(res_message)
98
99         except IOError:
100             print('Failed to write to file')
101     except:
102         print("Illegal request")

```

Image 2

```

54 try:
55     # Check whether the file exist in the cache
56     ## FILL IN HERE...
57     #Attempt to open file
58     with open(filetouse, 'rb') as f:
59         cached_page = f.read()
60
61     fileExist = True
62     # ProxyServer finds a cache hit and generates a response message
63     ## FILL IN HERE.
64     print('Sending cached page..')
65     tcpCliSock.sendall(cached_page)
66

```

Image 3

```
C:\Users\Disra\Documents\Hub\School\Computer Networks\1- Web Proxy>python3 ProxyServer.py 127.0.0.1
Ready to serve...
Received a connection from: ('127.0.0.1', 52860)
Resource name: yahoo.com
Resource not in cache
Fetching page..
Sending page to client..
Writing page to cache..
Ready to serve...
Received a connection from: ('127.0.0.1', 52873)
Resource name: yahoo.com
Sending cached page..
Ready to serve...
```

*Image 4*

## **Conclusion**

Creating this server was an interesting process. One of the biggest challenges was understanding how these messages were formatted and how and when they were sent by each party. I had to proceed by doing research to understand part of the process then experimenting on my findings to see what output I could observe and replicate. This research approach applied to the socket API as it was my first time working with it. I found by keeping exception handling in mind, I could consider various types of errors that may occur as I learned and In the end, research, testing and a lot of experimentation was what allowed me to complete the assignment and understand how the entire processed worked.