CS421

Programming Assignment 1

29.03.2023

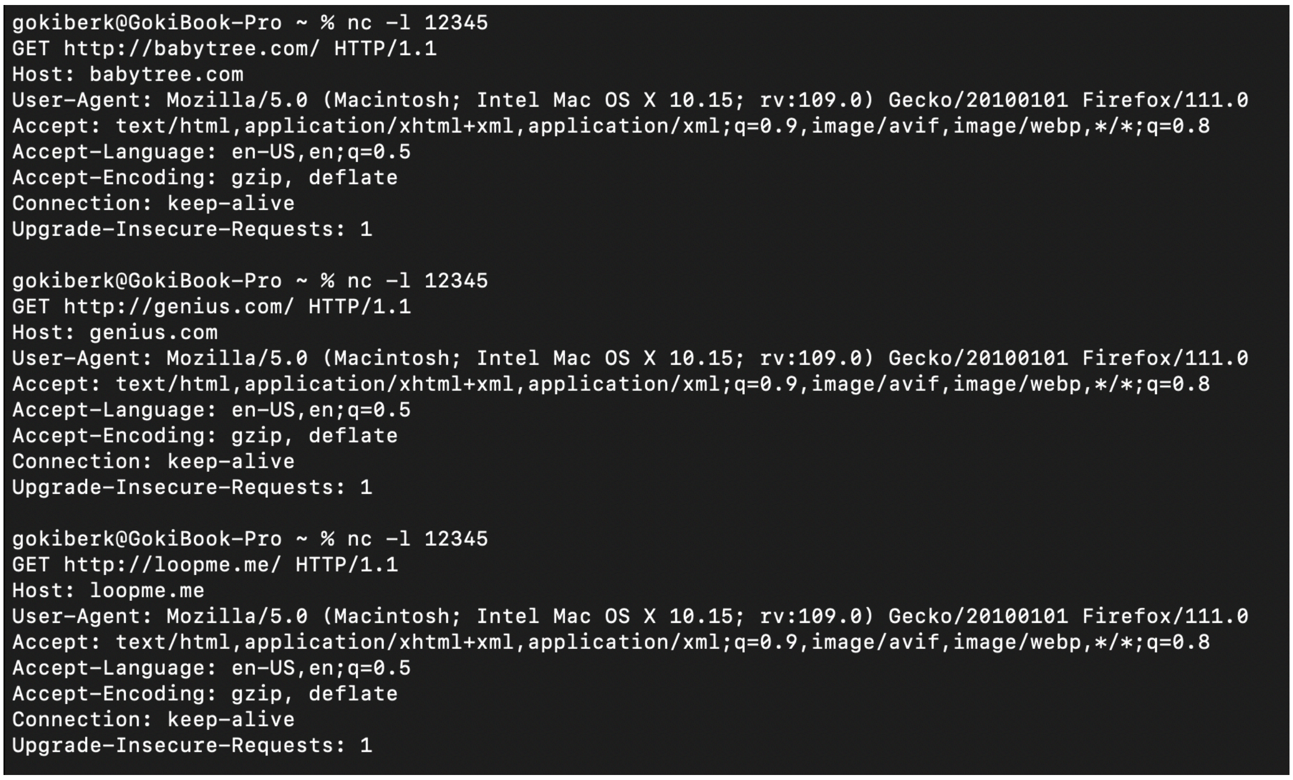
Gökberk Keskinkılıç

21801666

Part 1)

In this part, babytree.com, genius.com and loopme.me are chosen as target webpages.

The results are seen in Fig. 1.

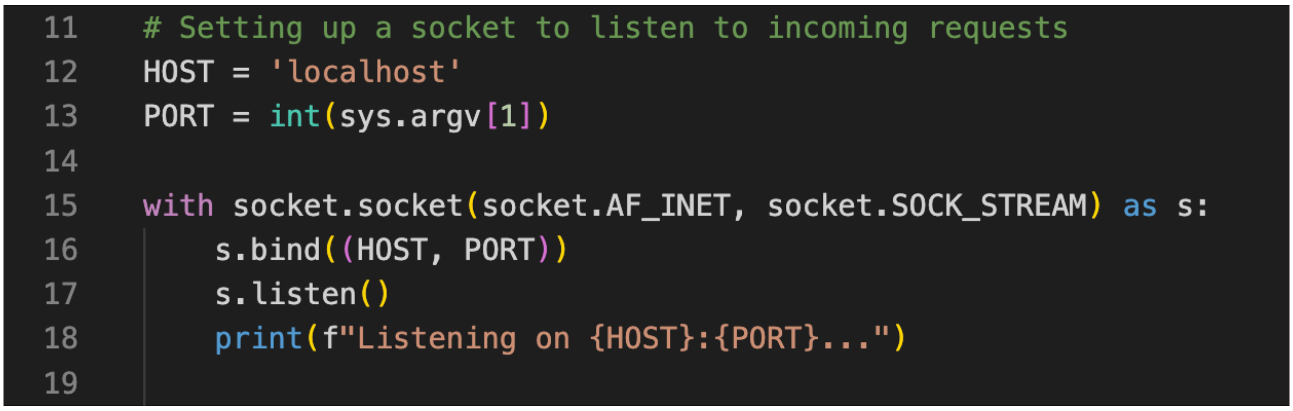


```
gokiberk@GokiBook-Pro ~ % nc -l 12345
GET http://babytree.com/ HTTP/1.1
Host: babytree.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

gokiberk@GokiBook-Pro ~ % nc -l 12345
GET http://genius.com/ HTTP/1.1
Host: genius.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

gokiberk@GokiBook-Pro ~ % nc -l 12345
GET http://loopme.me/ HTTP/1.1
Host: loopme.me
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Fig. 1. Output of the terminal for 3 different HTTP websites

Part 2)

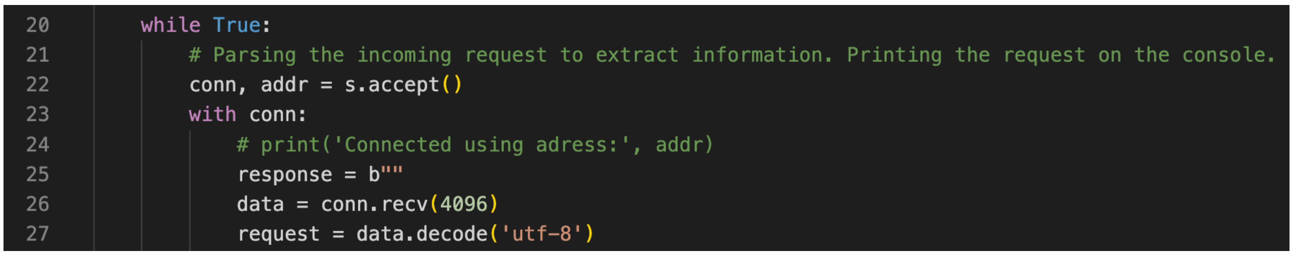
In this code snippet, port number is read from the terminal command and assigned to a variable, Python's default sys library is required to read user input. Python's socket library is used to perform socket binding. Socket binding is done once for runtime.



```python
# Setting up a socket to listen to incoming requests
HOST = 'localhost'
PORT = int(sys.argv[1])

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    print(f"Listening on {HOST}:{PORT}...")

```

Fig. 2. Defining the host and port variables to start listening

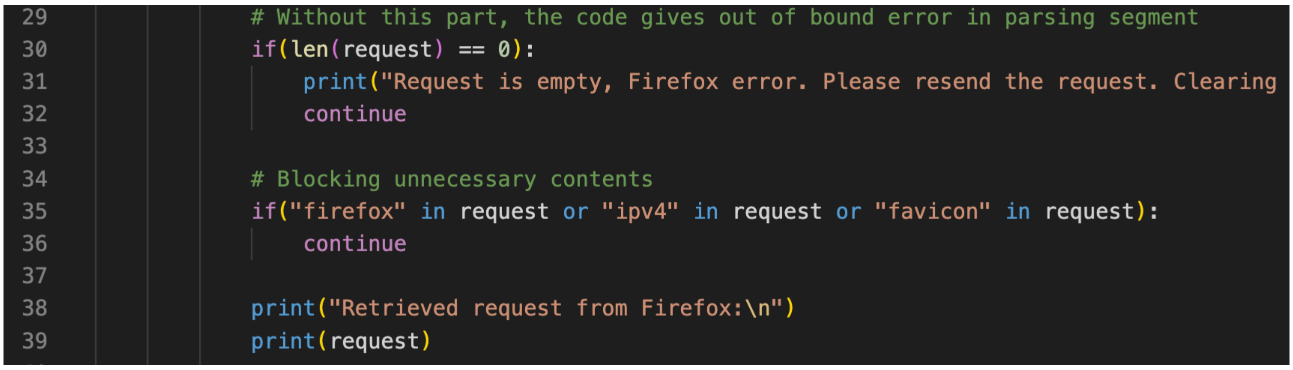
Starting with this code snippet everything is in a "while True" loop, this satisfies the condition to keep being able to connect other websites through runtime of the application. Connection is established here, every time new request is sent new connection is made. Using the connection, we receive the request that user sends to the browser.



```python
    while True:
        # Parsing the incoming request to extract information. Printing the request on the console.
        conn, addr = s.accept()
        with conn:
            # print('Connected using adress:', addr)
            response = b""
            data = conn.recv(4096)
            request = data.decode('utf-8')
```

Fig. 3. Establishing a connection

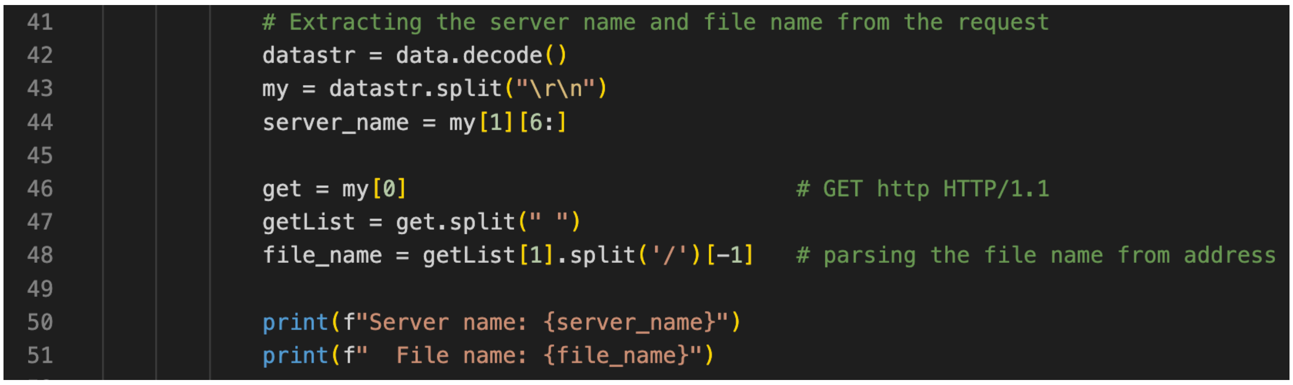
In this code snippet information in the request is analyzed. If the request is empty due to some reason, user is prompted to make another request and suggested to clear cache data of browser. If the request includes words such as firefox, ipv4 or favicon, those requests are skipped not evaluated. If the request passes those conditions, its header is printed.



```python
            # Without this part, the code gives out of bound error in parsing segment
            if(len(request) == 0):
                print("Request is empty, Firefox error. Please resend the request. Clearing
                continue

            # Blocking unnecessary contents
            if("firefox" in request or "ipv4" in request or "favicon" in request):
                continue

            print("Retrieved request from Firefox:\n")
            print(request)
```

Fig. 4. Evaluating the request

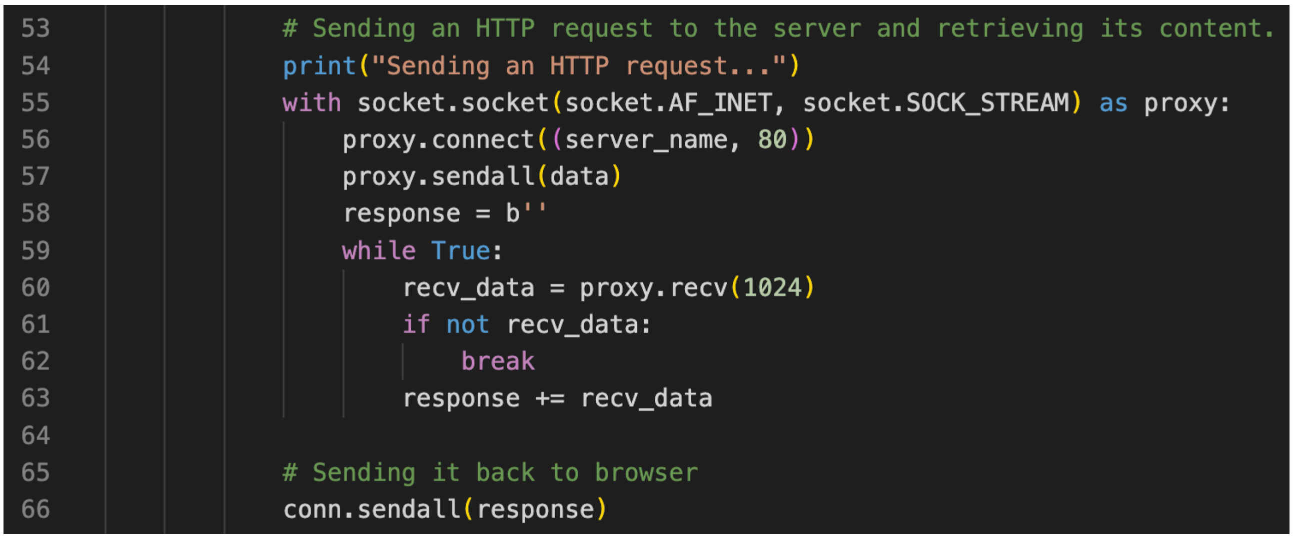
Here parsing is implemented for server name and file name. From the GET request line of header text, those required fields are extracted and printed on console.



```python
# Extracting the server name and file name from the request
datastr = data.decode()
my = datastr.split("\r\n")
server_name = my[1][6:]

get = my[0]                             # GET http HTTP/1.1
getList = get.split(" ")
file_name = getList[1].split('/')[-1]   # parsing the file name from address

print(f"Server name: {server_name}")
print(f"  File name: {file_name}")
```

Fig. 5. Parsing the request

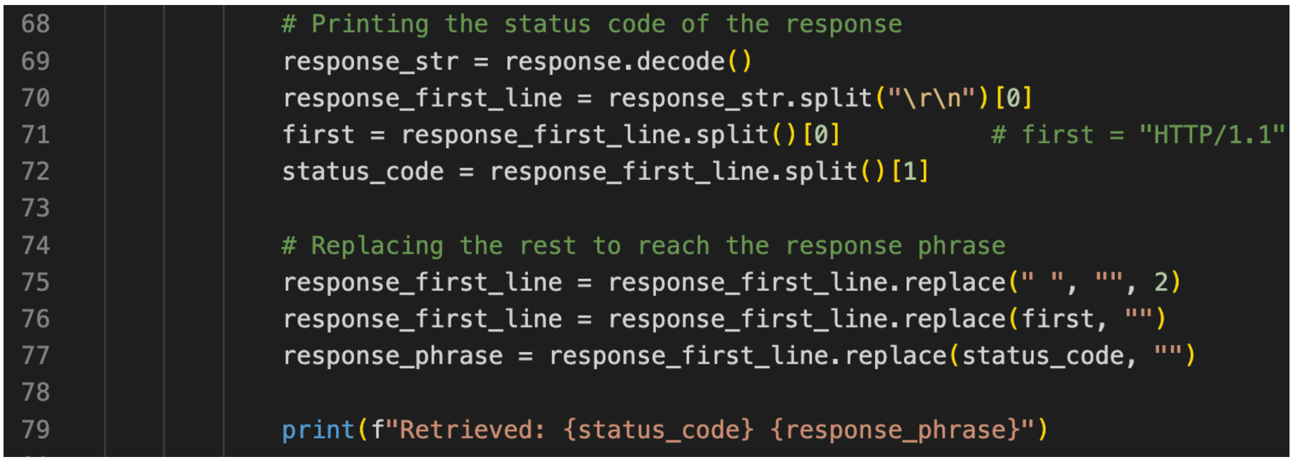
This is where proxy does most of its job. First an HTTP request is sent to server from proxy. Details of this request comes from the user request to browser. Then proxy receives the response from server. As it might be a large file, it makes sure that all of it is received by running in a “while True” loop. Then this response data is retransferred to browser so user can visualize it from the browser.



```python
# Sending an HTTP request to the server and retrieving its content.
print("Sending an HTTP request...")
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as proxy:
    proxy.connect((server_name, 80))
    proxy.sendall(data)
    response = b''
    while True:
        recv_data = proxy.recv(1024)
        if not recv_data:
            break
        response += recv_data

# Sending it back to browser
conn.sendall(response)
```

Fig. 6. Sending the request from to server, receiving the data and sending to browser.

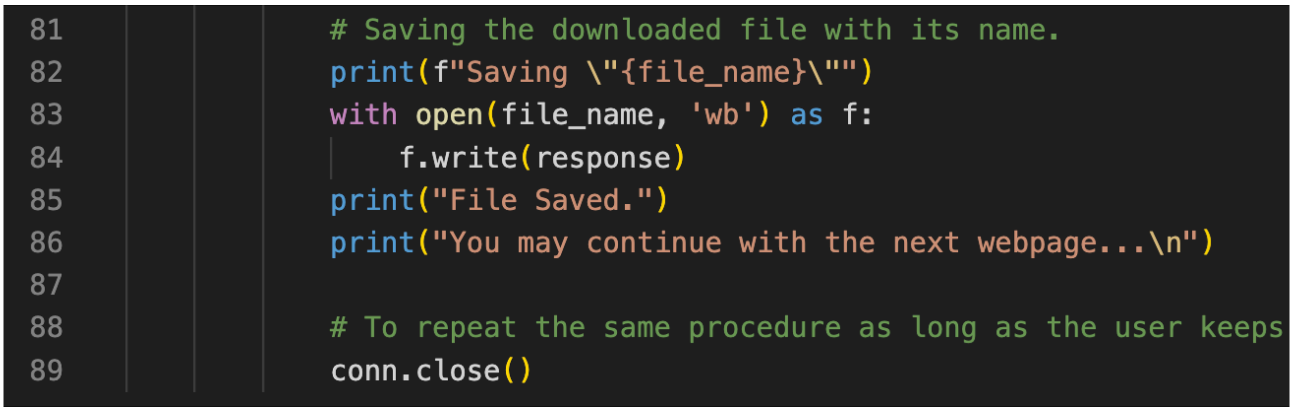
Parsing the response message is implemented in this code segment. Using string manipulations, status code and response phrase are extracted from the response.



```python
            # Printing the status code of the response
            response_str = response.decode()
            response_first_line = response_str.split("\r\n")[0]
            first = response_first_line.split()[0]          # first = "HTTP/1.1"
            status_code = response_first_line.split()[1]

            # Replacing the rest to reach the response phrase
            response_first_line = response_first_line.replace(" ", "", 2)
            response_first_line = response_first_line.replace(first, "")
            response_phrase = response_first_line.replace(status_code, "")

            print(f"Retrieved: {status_code} {response_phrase}")
```

Fig. 7. Parsing the response

Response is saved in this code segment. New text document is named using the file name that was extracted in Fig. 5. After file is saved user is prompted with a message indicating that the program is ready to act as a proxy for another webpage that user browses in browser.



```python
            # Saving the downloaded file with its name.
            print(f"Saving \"{file_name}\"")
            with open(file_name, 'wb') as f:
                f.write(response)
            print("File Saved.")
            print("You may continue with the next webpage...\n")

            # To repeat the same procedure as long as the user keeps
            conn.close()
```

Fig. 8. Saving the response as a text file