NOVA SCHOOL OF
SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

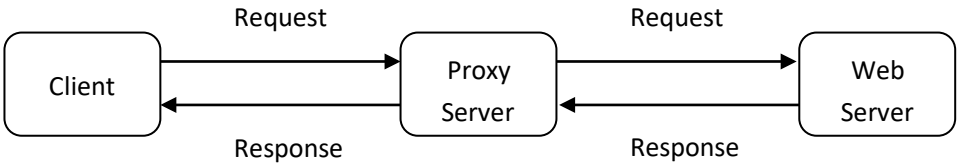
HTTP Proxy and Cache Server

(TPC4 – Redes de Computadores 2024/2025)

1. Introduction

This assignment is about how web proxy servers work. [1]

Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the **proxy server**. In other words, the client requests the objects via the proxy server. The proxy server will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.



A typical **proxy server can include a cache**. The proxy will cache the web pages each time the client makes a particular request for the first time. The basic functionality of caching works as follows.

When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache, without contacting the server. If the object is not cached, the proxy retrieves the object from the server, returns it to the client and caches a copy for future requests. In practice, the proxy server must verify that the cached responses are still valid and that they are the correct responses to the client's requests. You can read more about caching and how it is handled in HTTP in RFC 2068..

The assignment has a mandatory part: a very simplified proxy server which only handles GET-requests, but is able to handle all kinds of objects - not just HTML pages, but also images. Two versions of this proxy server are requested: one sequencial and another concurrent. There is also an optional part: adding to the base server the support of caching,

Please note that lots of web sites only accept secure connections through HTTPS. Our proxy does not support HTTPS. Please test your proxy with sites that reply to requests in HTTP. **The site http.//vps726303.ovh.net/rc2425 is available for testing.**

[1] This assignment is based on an exercise proposed by Kurose & Ross in the site that supports their book "Computer Networking: a Top-Down Approach"
https://gaia.cs.umass.edu/kurose_ross/programming/Python_code_only/

2. Proxy base version

When using a proxy, this server intercepts the request and makes some modifications to the original request. The main modification that is mandatory is in the first line of the request. The client sends a complete URL and the request sent by the proxy only contains the path to the file. For example, the browser sends

`GET http://www.example.com/index.html HTTP/1.1`

And the first line of the request sent to *www.example.com* will be

`GET /index.html HTTP/1.1`

In the following there is the incomplete Python code for a proxy without cache.

```
from socket import *
import sys
def proxyServer( portNo ):

   tcpSerSock = socket(AF_INET, SOCK_STREAM)
   tcpSerSock.bind(("0.0.0.0", portNo))
   tcpSerSock.listen(5)

   while True:
       print('Ready to serve')
       try:
            tcpCliSock, addr = tcpSerSock.accept()
       except KeyboardInterrupt:
           print("proxy exiting!")
           break
       print('Received a connection from:', addr)
       # Fill in start.
       # Obtain the request from client (should be GET) and separate it in
.      # method, URL, header lines. If there is some
       # formatting error reply with error message 400 Bad Request


       # Fill end.
       print('Connecting to original destination')
       # Fill in start.
       # Create a socket and connect to port 80 of the host in the URL
       # If connect fails prepare reply with error message 404 Not found


       # Fill in end.
       print('Connected to original destination')
       # Fill in start.
       # Receive reply from server. Separate it in status (numeric
       # and textual, header lines and body (in the case of POST)


       # Fill in end.
       print('received reply from http server')
       # Fill in start.
       # forward server's reply to client


       # Fill in end.
       print('reply forwarded to client')
       # Close the client socket
       tcpCliSock.close()

   # Proxy server is terminated by control C.
   tcpSerSock.close()
```

```
if __name__ == "__main__":
    # python proxyServer.py serverTCPPort
    if len(sys.argv)!=2:
             print("Usage: python3 proxyServer.py  proxyTCPPort")
             sys.exit(1)
    else:
             proxyServer(int(sys.argv[1]))
             sys.exit(0)
```

You should complete the code above. For developing the code, you can use Python libraries. In the following, same relevant ones are mentioned:

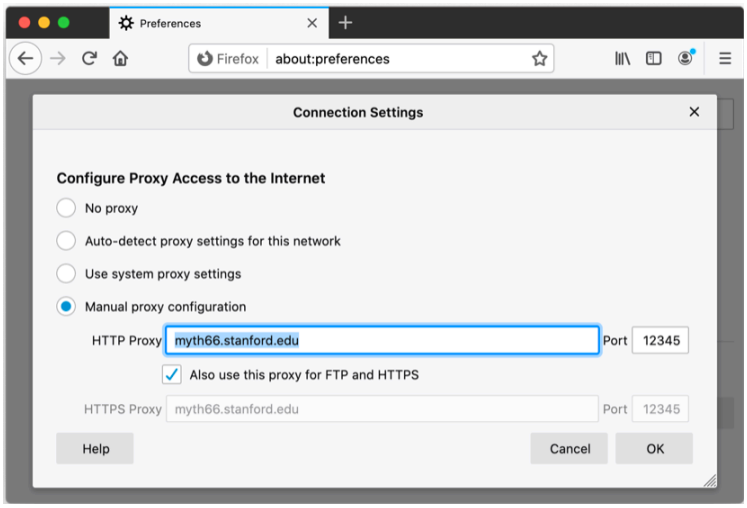
- *socket* https://docs.python.org/3/library/socket.html
- *requests* https://requests.readthedocs.io/en/latest/
- *http.client* https://docs.python.org/3/library/http.client.html
- *urllib* https://docs.python.org/3.13/library/urllib.html
- *urllib.parse* https://docs.python.org/3.13/library/urllib.request.html#module-urllib.request

For introductory information, please consult the free-access book **Python for Everybody: Exploring Data Using Python 3** of Dr. Charles R. Severance, available at https://www.py4e.com/book.php , chapters 12 and 13.

3. Testing the proxy server using a browser

Many browsers have the possibility to define a proxy in its “Settings” menu. In the following, see how to define a proxy server in Firefox.

- **Mac:** Click Firefox -> Preferences. Then scroll to the bottom to "Network Settings", click on "Settings..." and activate a manual proxy as shown in the following screenshot.
- **PC:** Click Tools -> Options. Then scroll to the bottom to "Network Settings", click on "Settings..." and activate a manual proxy as shown in the following screenshot.



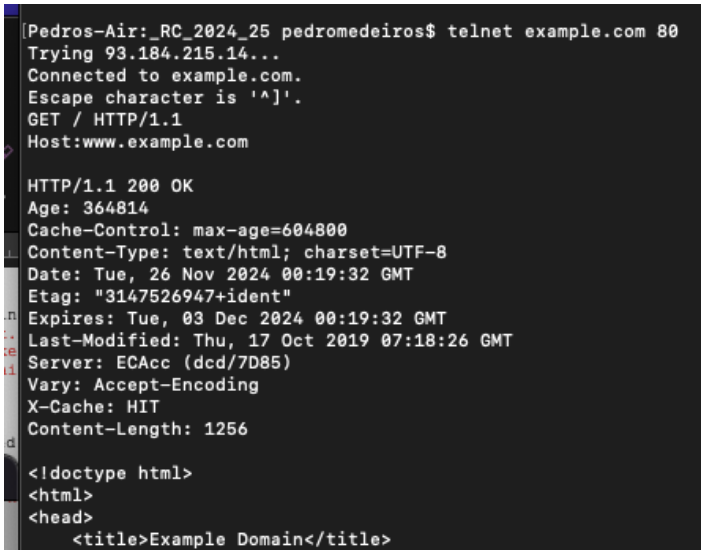
In our case, the proxy will be **localhost:X** where X is the port defined in the command line that invokes the proxy server.

In recent versions of browsers this change can be difficult to perform: If that occurs, you should resort to the alternatives mentioned in section 4.

4. Testing the proxy server using the command line

There are several alternatives for testing the web proxy. In this section, we discuss command-line tools and in the next one, how to use the browser for that purpose.

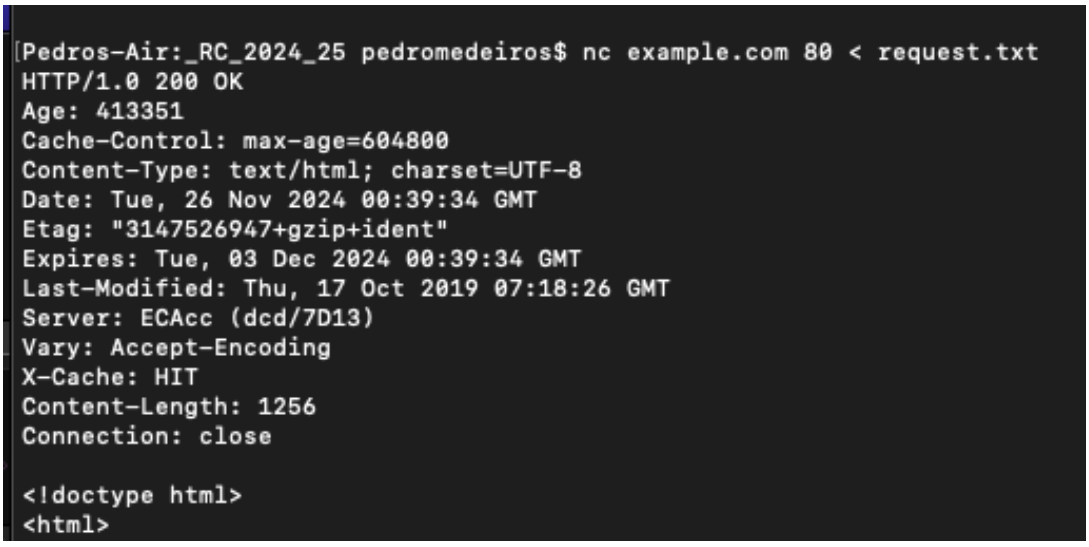
Telnet can be used to interact with a web server as demonstrated in the following picture. Don't forget to press RETURN twice to end the request.



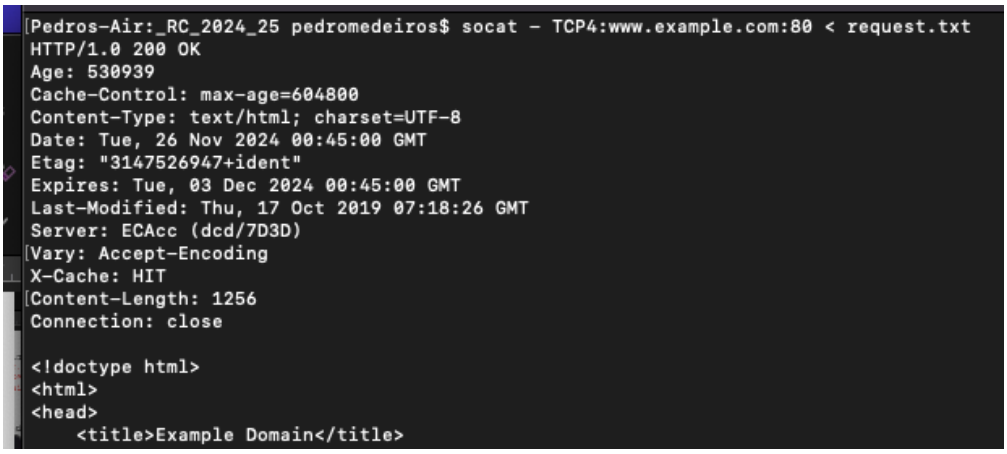
Telnet does not allow the use of redirection. Other tools (ex.: *nc*, *socat* or *curl*, illustrated below) allow it and are, thus, preferable. Let's assume that we have a file called *request.txt* with the following contents:

```
GET / HTTP/1.0
Host: www.example.com
Connection:close
(blank line, only with CR, LF)
```

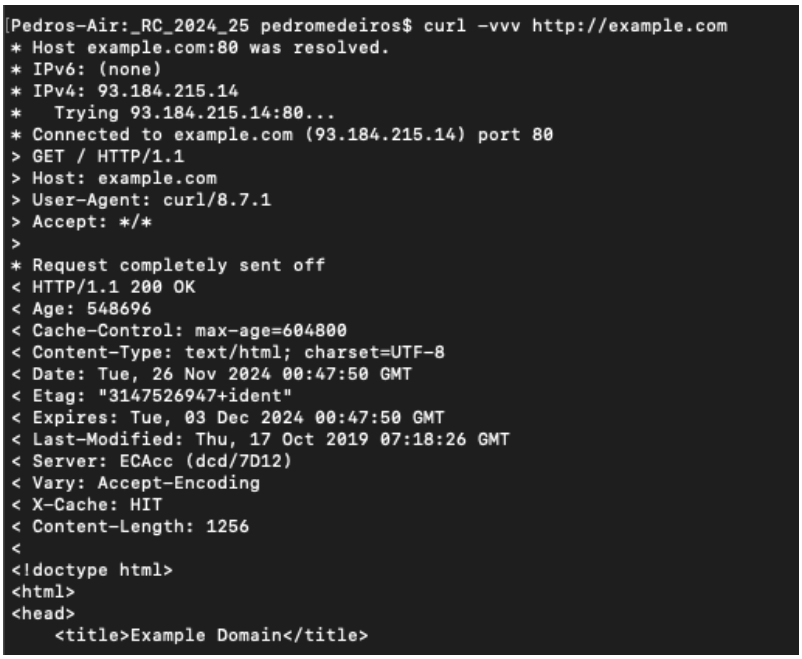
nc (ncat in some systems)



socat



curl



curl allows the specification of a proxy, which will be useful in this assignment. The syntax is as follows.

`curl --proxy “http://47.98.205.231:33332“ “http://www.example.com:80”`

In some operating systems, the installation of programs like *curl* can be difficult. There are two programs available in CLIP that can work has a (very limited) replacement of *curl*:

- *httpClient1.py* that is invoked with a command line

  python3 httpClient1.py hostname

  that performs several GET HTTP operations targeted at an HTTP server in *hostname.*

- *httpClient2.py* that is invoked with a command line

  python3 httpClient2.py proxyHostName proxyPort

  that is similar to *httpClient1*, but where GET commands can be sent to a proxy server in (*proxyHostName, proxyPort)*.

5. Multithreaded version

To accelerate our proxy server, modify the code, allowing the concurrent processing of multiple requests from clients.

6. Proxy with cache

Add the simple caching functionality described above. You do not need to implement any replacement or validation policies. Your implementation, however, will need to be able to write responses to the disk (i.e., the cache) and fetch them from the disk when you get a cache hit. For this you need to implement some internal data structure in the proxy to keep track of which objects are cached and where they are on the disk. A suggestion would be to use a distinct folder for each site.

You can keep this data structure in main memory; there is no need to make it persist across shutdowns.

Don´t forget that does not make sense to cache the contents of dinamically generated pages.

7. Delivery

The delivery should be done **before 11:00 on December 3, 2024**. The submission has two parts:

- a Google form containing the identification of the students that submit the work and questions about the functionality of the code. This should be filled by both students of the group.
- The code developed should be uploaded through Moodle. This should only be done by one of the members of the group. Please separate your delivery in 3 source Python files:
    - A sequential proxy without cache
    - A concurrent proxy without cache
    - A sequential proxy with cache. (this last one is optional)

More details will be sent later through a CLIP message.