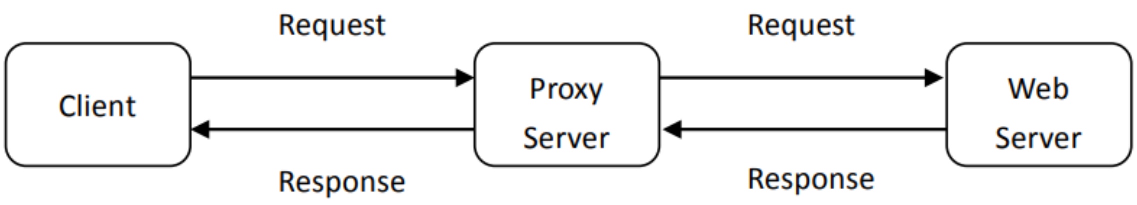
Multi-Threaded HTTP Web Proxy

Overview

In this assignment, you will develop a Web proxy. When your proxy receives an HTTP request for an object from a client, it generates a new HTTP request for the same object and sends it to the origin server. When the proxy receives the corresponding HTTP response with the object from the origin server, it creates a new HTTP response, including the object, and sends it to the client. This proxy will be multi-threaded, so that it will be able to handle multiple requests at the same time.



Test Environment

The test environment of the web proxy server is:

- Python version: 3.8 or above
- HTTP version: HTTP/1.1
- Curl version: 7.55.1 or above
- Encoding: utf-8
- HTTP Method: GET HEAD POST

Note that you are **not allowed** to use the `requests` library in python in the assignment.

Running the Proxy Server

Run the proxy server program using your command prompt and specify the IP address and port number of the proxy server. Note that the default IP address is `127.0.0.1` and the default port number is `8080`.

There are four ways to run the proxy server:

- Method 1:

```
python web_proxy.py
```

In this case, the proxy server is running on `127.0.0.1:8080`

- Method 2:

```
python web_proxy.py IP_ADDRESS
```

In this case, the proxy server is running on `IP_ADDRESS:8080`

- Method 3:

```
python web_proxy.py PORT_NUMBER
```

In this case, the proxy server is running on `127.0.0.1:PORT_NUMBER`

- Method 4:

```
python web_proxy.py IP_ADDRESS PORT_NUMBER
```

In this case, the proxy server is running on `IP_ADDRESS:PORT_NUMBER`

You can request a web page using `curl` by with `-x (--proxy)` to specify the IP address and port number of the proxy server. e.g.

```
curl -x 127.0.0.1:8080 http://www.time.org/zones/GMT.php
```

Tasks

1. Proxy GET and HEAD Request (60 points)

The simple proxy server should support HTTP GET and HTTP HEAD method. When your proxy receives an HTTP request from a client, it should parse the header of request and connect to the origin server using socket.connect(address). After getting the response message from the origin server, the proxy server should send it to the client.

You can use http://www.example.com/ for self-test, which is a website relies on HTTP.

Proxy GET Request

The client will display the body of the response when using GET method.

Run the web proxy server in one command prompt:

```
python web_proxy.py
```

Using curl to initiate a request in another command prompt in the same computer:

```
curl -x 127.0.0.1:8080 http://www.example.com/
```

You will see the corresponding response data if the proxy server works well:

```
<!doctype html>
<html>
<head>
......
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use
this
    domain in literature without prior coordination or asking for permission.
</p>
```

```
    <p><a href="https://www.iana.org/domains/example">More information...</a>
</p>
</div>
</body>
</html>
```

Proxy HEAD Request

The client will receive the header of the response when using HEAD method.

Run the web proxy server in one command prompt:

```
python web_proxy.py
```

Using curl to initiate a request in another command prompt in the same computer:

```
curl -x 127.0.0.1:8080  http://www.baidu.com/ --head
```

You will see the corresponding response header if the proxy server works well:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Connection: keep-alive
Content-Length: 277
Content-Type: text/html
......
```

2. Proxy Post Request (10 points)

Add support for POST, by including the request body sent in the POST-request.

You can use http://ecosimulation.com/testpost.html for post self-test.

Run the web proxy server in one command prompt:

```
python web_proxy.py
```

Using curl to initiate a request in another command prompt in the same computer:

```
curl -x 127.0.0.1:8080 http://ecosimulation.com/cgi-bin/testpost.cgi -X POST -d
"firstname=san&lastname=zhang"
```

You will see the following response if the proxy server works well:

```
hello!
key:lastname, value:zhang
key:firstname, value:san
goodbye!
```

3. Support Multithreading (5 points)

This proxy should support multiple users. Using asyncio or the threading library are both acceptable.

Since you will get a response soon after you make a request, you could use `curl` with `--limit-rate <speed>` to specify the maximum transfer rate you want curl to use. e.g.

Run the web proxy server in one command prompt:

```
python web_proxy.py
```

Using curl to initiate requests in another three command prompts in the same computer:

```
curl -x 127.0.0.1:8080 http://www.baidu.com/ --limit-rate 64B   1B
curl -x 127.0.0.1:8080 http://www.baidu.com/ --limit-rate 64B
curl -x 127.0.0.1:8080 http://www.baidu.com/ --limit-rate 64B
```

Your proxy server should be able to handle this case.

4. Caching (25 points)

Basic (10 points)

A typical proxy server 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. Add the simple caching functionality described above. Your implementation, however, will need to be able to write responses to the disk and fetch them from the disk when you get a cache hit. When there is a **cache hit**, the web proxy server needs to **print a specific prompt message**: `Read From Cache`. It is recommended to store the cache file in the same directory as the file `web_proxy.py`.

Advance (15 points)

If there is a cache hit, the web proxy server should modify the original request header from the client by adding a new filed `If-Modified-Since` whose value is the time when the request was cached in **GMT time format**. After sending this message to the server, the server will determine whether the requested resource has been modified after the time you sent it. If it is modified, it will return the status code `200` OK and include the latest resource. In the case, you also need to update the cache. If it returns `304` Not Modified, It means that there is no modification, and no extra content will follow. At this time, you must read the content from the cache file. Note that `If-Modified-Since` can only be used in `GET` or `HEAD` requests. (10 points)

You also need to pay attention to the concurrency problem when dealing with multithreading. You can refer to the Readers–writers problem. **Simple requirements can get full marks**, there is no need to achieve optimal solutions. (5 points)

There are some websites to help you test the advance cache:

```
http://www.w3.org/
http://www.example.com/
```

You may refer to this website for more information how-to-test-for-if-modified-since-http-header-support

What to Submit

Your complete proxy server code should be in a python file named `web_proxy.py` (100 points). You also need to write a simple report named `report.pdf` (20 points) which describes your `design ideas` (10 points) and `how you test your code` (10 points) with corresponding screenshots. You should compress the source code and report in a folder named `SID.zip` (e.g. `11810000.zip`). You need to submit the `.zip` file in sakai.