Task

Your task is to develop a small web proxy server which is able to cache web pages.

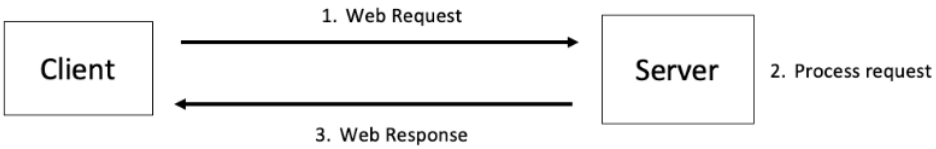
It is a very simple proxy server which only understands simple HTTP/1.1 GET-requests but is able to handle all kinds of objects - not just HTML pages, but also images.

Introduction

In this practical, you will learn how web proxy servers work and one of their basic functionalities, **caching**. Generally, when a client (e.g. your browser) makes a web request the following occurs:

1. The client sends a request to the web server
2. The web server then processes the request
3. The web server sends back a response message to the requesting client

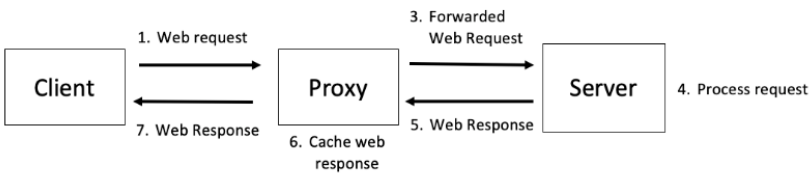
And let's say that the entire transaction takes 500 ms.



In order to improve the performance, we create a proxy server between the client and the web server. The proxy server will act as a middle-man for the web transactions. Requesting a web request will now occur in the following steps:

1. The client sends a request to the proxy server
2. Skip to step 7 If the proxy server has cached the response
3. Forward the request to the web server
4. The web server processes the request
5. The web server sends a response back to the proxy server
6. The proxy server caches the response
7. The proxy server returns the cached response to the client

On the first request, the transaction may be a fraction longer than the previous example. However, subsequent requests will be significantly faster due to reduced network latency and server load (sometimes less than 10 ms).



The mechanism for caching can be as simple as storing a copy of a resource on the proxy server's file system.

The standard you are using, HTTP1.1, in your programming assignment is described in detail in RFCs (request for comments)
[see: https://en.wikipedia.org/wiki/Request_for_Comments] (Links to an external site.)

HTTP1.1 RFC we use is 2616 [https://tools.ietf.org/html/rfc2616Links to an external site.]

Please read **Section 5.1.2** in RFC 2616.

You can read more about caching and how it is handled in HTTP in RFC 2616 in **Section 13**.

Now we want you to aim for the correct protocol/proxy behavior (see the marking rubric at the end of the practical description for what we will look for).

Steps for approaching the practical

Step 1

Understand the HTTP/1.1 requests/responses of the proxy. Your proxy MUST be able to handle GET requests from a client. Your proxy may handle other request types (such as POST) but this is not required.

You need to know the following:

1. What HTTP request will the browser send to the proxy?
2. What will HTTP response look like?
3. In what ways will the response look different if it comes from the proxy than if it comes from the origin server (i.e. the server where the original page is stored?). You will not be able to test this yet, but what do you think would happen?

Step 2

Understand the socket connections:

1. How will the client know to talk to the proxy?
2. What host and port number will the proxy be on?
3. The proxy may need to connect to the origin server (if the web object is not in the cache), what host and port number will the proxy connect to?
4. Where will the proxy get the web object information?

Step 3: Checkpoint

Make sure you have the answers to steps 1 & 2 before you go any further.

You can't code it if you don't know what should happen.

Ask questions on the discussion forum if you are unsure about the above.

Step 4: Python Code

Download the skeleton code Proxy-skeleton(2025).py Download Proxy-skeleton(2025).py. Look at the interactions you identified in steps 1 & 2 and see where they would occur in the code.

Review the python code presented in the sockets lecture. You'll find details of the socket calls in the Python Socket library
documentation https://docs.python.org/3.13/library/socket.htmlLinks to an external site.

Links to an external site.***You won't just be able to copy the lecture code***, but it shows you the key steps; creating sockets, connecting sockets, sending data on sockets and receiving data on sockets.

Your task is to make the correct socket calls and supply the correct arguments for those calls.

You will **only** need to fill in the code between the comment lines.

```
# ~~~~ INSERT CODE ~~~~
...
# ~~~~ END CODE INSERT ~~~~
```

The comments above the comments lines give you hints to the code to insert.

Step 5: Differences in Python and C

If you are new to python, look at the code structure. Most of the code is given to you with your focus just on adding the networking code. A couple of things that are different in Python than in the C derived languages:

1. Python uses whitespace to indicate code blocks and end of lines. Note there are no brackets or braces { } around code blocks and no semi-colons ; at the end of lines. The indentation isn't just important for readability in Python, it affects how your code runs. Make sure you indent code blocks correctly.
2. Python has a tuple data structure. So functions can return more than one value, or more precisely return one tuple with multiple values, but the syntax allows the parenthesis to be left off. Tuples appear in the lecture slides in the line:

   ```
   clientSocket.connect((serverName,serverPort))
   ```

   `(serverName, serverPort)` is a tuple of two values that are passed as the argument to the `connect()` function.

   ```
   connectionSocket, addr= serverSocket.accept()
   ```

   The accept() call returns a tuple of two values, the first value is the new socket and the second is the address information. This is the same as:

   ```
   (connectionSocket, addr)= serverSocket.accept()
   ```

   The use of the `( )` around the tuple is optional.

Step 6

Start with getting the proxy working when a cached file is requested. Where it will return the response itself and does not have to contact the origin server

Step 7

Once that is working, add the code to handle the case where the file is not cached by the proxy and the proxy must request it from the origin server.

Step 8

After that, try to handle redirected webpage. 301 and 302 should they be cached? Lastly, handle cache-control header max-age=<seconds> in the response from the origin server.

Step 9

In steps 6,7, and 8, make use of both telnet utility *and* a cURL to test your proxy server. You can also use **Wireshark** to capture what is being sent/received from the origin server.

Running the Proxy Server

Download the template file after you do the quiz associated with the programming assignment and save it as Proxy.py.

Run the following command in terminal to run the proxy server

$ python Proxy.py localhost 8080

You can change localhost to listen on another IP address for requests, localhost is the address for your machine.

You can change 8080 as the port to listen to.

The skeleton code without any modifications should (tested on Python 3.13 on university systems) should go into an infinite loop. You can use Ctrl+C to terminate a program.

Testing your proxy code

- When it comes time to test your code, use cUrl and telnet utilities.
- If you want to test on an image that is the same as the web submission system, you can ssh into: uss.cs.adelaide.edu.au and then run and test your code.

Let's have a look at these two tests:

Using **cURL and useful URLs for testing your proxy**

Obtaining a remote web page

```
$ curl -iS http://localhost:8080/http://http.badssl.com
HTTP/1.1 200 OK
...
```

The above command requests http://http.badssl.com/ via the Web proxy. -i prints out response with headers and -S (uppercase S) shows errors when they occur. Executing the request will result is the first line in the response from the proxy. The response you see should ***match if you were talking directly to the origin server***, like so

```
$ curl -iS http://http.badssl.com/
HTTP/1.1 200 OK
...
```

Handle page that does not exist

```
$ curl -iS http://localhost:8080/http://http.badssl.com/fakefile.html
HTTP/1.1 404 Not Found
...
```

The response for a path that doesn't exist shows the above status code. Your proxy to work correctly will also need to handle this case too.

Handle re-directed webpage 301 and 302

The links below will generate a server response that return 301 and 302 response code respectively. This is from a server set up for helping to test HTTP clients and you can explore services at the server by loading http://httpbin.org. **Note:** the use of quotation marks to avoids the & symbol from being interpreted as a command to run a process in the background.

```
$ curl -iS "http://localhost:8080/http://httpbin.org/redirect-to?url=http://http.bad
ssl.com&status_code=301"
HTTP/1.1 301 MOVED PERMANENTLY
...

$ curl -iS "http://localhost:8080/http://httpbin.org/redirect-to?url=http://http.bad
ssl.com&status_code=302"
HTTP/1.1 302 FOUND
...
```

If you want to *see the response generated* from the http://httpbin.org service for generating 301 and 302 responses, you can type (as this will bypass your proxy and send the request directly to http://httpbin.org:

```
$ curl -iS "http://httpbin.org/redirect-to?url=http://http.badssl.com&status_code=30
1"
HTTP/1.1 301 MOVED PERMANENTLY
...

$ curl -iS "http://httpbin.org/redirect-to?url=http://http.badssl.com&status_code=30
2"
HTTP/1.1 302 FOUND
...
```

Respond correctly to the cache-control header max-age=<seconds>

By using the same server as for the previous test, the links below will generate a server response with max-age set to 0 and 3600 respectively. You can test with different max-age values by changing the value at the end of the link.

```
$ curl -iS "http://localhost:8080/http://httpbin.org/cache/0"
HTTP/1.1 200 OK
...


$ curl -iS "http://localhost:8080/http://httpbin.org/cache/3600"
HTTP/1.1 200 OK
...
```

URL of an image

Here is a link to an image you can use to test on a file other than an html file.

- http://http-textarea.badssl.com/icons/icon-red.pngLinks to an external site.

Using **telnet**

You can use all of the URLs above with telenet. However, with telnet, you need to connect to your proxy fist. This is a good tool to use to make sure you can connect and send basic requests and if you want to see all of the bytes in the response dumped to your screen. Using telenet together with cUrl will help you test your proxy.

```
[prompt]$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET http://http.badssl.com HTTP/1.1
```

Note that we used the absolute URL because telnet here is connected to you proxy and not the origin server and we need to send the full resource location to the proxy so the proxy knows where to locate the resource on the Internet.

URLs for testing (Be aware!)

When testing, be aware that for certain URLs, although you issue http:// requests, for security reasons, they might get directed to https://. It has now become common practice for most websites to redirect http:// to https://. Cosequently, you may have seen 30x like responses. This is why we provide a set of URLs to use for testing where the webservers still support HTTP requests.

Here are some variants in addition to http://http.badssl.com/ and the links at httpbin.org but feel free to share other you find on the discussion forums.

- http://http-textarea.badssl.com/Links to an external site. (just to show you that there are other pages at this web server)
- http://jsonplaceholder.typicode.com/todos/1Links to an external site.

Configuring your Browser (Optional)

You can also directly configure your web browser to use your proxy if you want without using the URI. This depends on your browser.

- In Internet Explorer, you can set the proxy in *Tools* > *Internet Options* > *Connections tab* > *LAN Settings*.
- In Netscape (and derived browsers such as Mozilla), you can set the proxy in *Tools* > *Options* > *Advanced tab* > *Network tab* > *Connection Settings*.

In both cases, you need to give the address of the proxy and the port number that you set when you ran the proxy server. You should be able to run the proxy and the browser on the same computer without any problem. With this approach, to get a web page using the proxy server, you simply provide the URL of the page you want.

For example, running http://http.badssl.com/ would be the same as running http://localhost:8888/http://http.badssl.com/

Set up like this, the browser should successfully load both the main web page and all associated files due to reasons we'll discuss in tutorials.

Submission

Your proxy server will be inside the Proxy.py file only.

Your work will need to be submitted to Gradescope through the submission portal at the bottom of this assignment page.

Gradescope will perform a static analysis of your code. Your code will be run and marked after the deadline.

Marking

- See the attached **rubric** for the functionality we will be testing for and the associated marks.
- This assignment will be marked, off-line, after the deadline. Gradescope will do some basic sanity checks for you, but you need to test your code thoroughly and, on an image, (with the same operating system, python and associated libraries, etc.) as Gradescope system (see **Testing your code** section).
- The final marks with feedback will be available on Gradescope and your course lecturer/marker will send an **announcement** to indicate that the marks are up.

Bonus Marks Questions (Optional)

Your mission, should you chose to accept it, will be rewarding!

If you have completed Prac 1 and would like some extension work. Here are three questions to tackle for 6 bonus marks (*so if you get full marks for the prac, the 6 bonus marks will be on top of that*).

Bonus mark questions (2 marks for each component that is correctly implemented)

Our current proxy doesn't check if the cached file is still ***fresh***. In practice, the proxy server must verify that the cached responses are still valid and that they are consistent with the responses the client would receive from the origin server. You can read more about caching and how it is handled in HTTP in RFC 2616.

Modify your web proxy to handle the following:

1. Check the Expires header of cached objects to determine if a new copy is needed from the origin server instead of just sending back the cached copy (2 marks)
2. Pre-fetch the associated files of the main webpage and cache them in the proxy server (DO NOT send them back to the client if the client does not request them). Look for "href=" and "src=" in the HTML. (2 marks)
3. The current proxy only handles URLs of the form hostname/file. Add the ability to handle origin server ports that are specified in the URL, i.e. hostname:portnumber/file (2 marks)

Hand in: please hand in your *Proxy.py* and *Proxy-bonus.py* for the version that contains the solution to the bonus questions. You can edit any part of the code necessary for *Proxy-bonus.py*. (Please note that you are handing in two files). However

1. At the start of the *Proxy-bonus.py* file explain which of the above you have implemented and ***how.***
2. For each implementation and any code added, please clearly document your code so that it is clear how you built your solution for each of the three extension questions above. ***No comments or no explanation implies no marks, sorry! So please document your work so we can easily understand your implementation.***