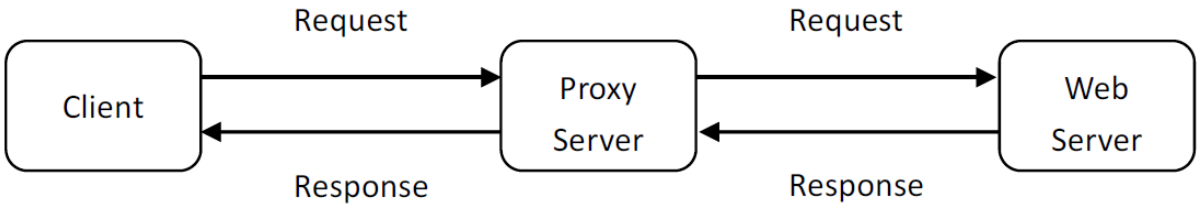
CSEE 4119, Spring 2019

PA 1: HTTP Web Proxy Due: March 4

Introduction

In this programming assignment, you will develop a (multi-threaded) web proxy server which is able to cache web pages. It is a simple proxy server which only understands simple GET requests, but is able to handle all kinds of objects - not just HTML pages, but also images.

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 cache the web pages each time the client makes a particular request for the first time. When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache. 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.



Steps:

The assignment has been divided into 4 steps:

Step 1: Simple proxy server (60 points) (+10 bonus points if you make the code better, see description of Step 1)

Step 2: Error handling (15 points)

Step 3: Cache verification and replacement (25)

Step 4: Multi-threading (optional, 10 bonus points!)

Total: 100 points (+ 20 bonus points!)

Step 1: Simple Proxy Server

1. Complete the Skeleton code

At the end of this document, you will find the skeleton code for a simple proxy server. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

Note 1: The code uses the method `socket.makefile` that you can read more about it online. You can alternatively use `.send` and `.recv` methods. You should inspect the code to lean how each line works.

Note 2: This is not the most elegant code for this Step. For example, the proxy throws away many useful headers in the client request. We give you bonus points if you come up with a better way of sending requests from proxy to server and response from proxy to client.

2. Running the Proxy Server

Run the proxy server program using your command prompt. Then request a web page from your browser. Direct the requests to the proxy server using your IP address and port number.

For e.g. http://localhost:8888/www.google.com

To use the proxy server with browser and proxy on separate computers, you will need the IP address on which your proxy server is running. In this case, while running the proxy, you will have to replace the "localhost" with the IP address of the computer where the proxy server is running. Also note the port number used. You will replace the port number used here "8888" with the port number you have used in your server code at which your proxy server is listening.

3. Configuring your Browser

You can also directly configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in Tools > Internet Options > Connections tab > LAN Settings. In 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 gave 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 e.g. http://www.google.com

At the end of step 1, you have a simple proxy server. You should verify that you indeed get the web pages via the proxy server.

Step 2: Add Error Handling

The skeleton code does no error handling. This can be a problem especially when the client requests an object which is not available, since the "404 Not found" response usually has no response body and the proxy assumes there is a body and tries to read it. Add error handling capability to proxy.

Step 3: Cache Verification and Replacement

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. Add the simple caching functionality described above using conditional GET. Your implementation will need to be able to replace the objects in the cache with their up-to-date versions. For this you need to implement some internal data structure in the proxy to keep track of which objects are cached, their last modified dates, and where they are on the disk. You can keep this data structure in main memory; there is no need to make it persist across shutdowns.

Step 4: Multi-threading (optional)

Right now, your proxy can handle one request at a time. Write a multi-threaded proxy using `threading` module, so that your proxy will be able to handle multiple requests at the same time.

What to Hand in:

Please submit a zip file using the format <UNI>. zip (e.g. jg3465.zip) to PA1 in courseworks. Your zip file must include:

- complete proxy server code (called ProxyServer.py)
- screenshots at the client side verifying that you indeed get the web page via the proxy server.
- A README text file that explains the IP address and port number for your proxy. Also explain what data structure you used for caching. If you do the bonus parts, please indicate which bonus parts and explain your approach. Also add any other instructions that we need to be able to run your code.

Notes:

- Make sure your code is well commented and readable or you will be taken 5-10 points.
- If your code doesn't compile, we will call you to have a look at it and fix it. However, it will result in a deduction of 20% of your total points. We will **use Python 2.7.x** to test your code. So make sure your code runs using these versions of Python.

- You are permitted and encouraged to help each other through Piazza. However, you may not share source code. Refrain from getting any code off the Internet. Please read the academic integrity policy in the first lecture.

A Skeleton Python Code for the Simple Proxy Server

```
from socket import *
import sys
if len(sys.argv) <= 1:
	print('Usage : "python ProxyServer.py server_ip"\n[server_ip : It is the IP
Address Of Proxy Server')
	sys.exit(2)
# Create a server socket, bind it to a port and start listening
tcpSerSock = socket(AF_INET, SOCK_STREAM)
# Fill in start.
# Fill in end.
while 1:
	# Start receiving data from the client
	print('Ready to serve...')
	tcpCliSock, addr = tcpSerSock.accept()
	print('Received a connection from:', addr)
	message = # Fill in start. # Fill in end.
	print(message)
	# Extract the filename from the given message
	print(message.split()[1])
	filename = message.split()[1].partition("/")[2]
	print(filename)
	fileExist = "false"
	filetouse = "/" + filename
	print(filetouse)
	try:
		# Check wether the file exist in the cache
		f = open(filetouse[1:], "r")
		outputdata = f.readlines()
		fileExist = "true"
		# ProxyServer finds a cache hit and generates a response message
		tcpCliSock.send("HTTP/1.0 200 OK\r\n")
		tcpCliSock.send("Content-Type:text/html\r\n")
		# Fill in start.
		# Fill in end.
			print('Read from cache')
	# Error handling for file not found in cache
	except IOError:
		if fileExist == "false":
			# Create a socket on the proxyserver
			c = # Fill in start. # Fill in end.
```

```
                hostn = filename.replace("www.","",1)
                print(hostn)
                try:
                        # Connect to the socket to port 80
                        # Fill in start.
                        # Fill in end.
                        # Create a temporary file on this socket and ask port 80
                        for the file requested by the client
                        fileobj = c.makefile('r', 0) #Instead of using send and
                        recv, we can use makefile
                        fileobj.write("GET "+"http://" + filename + "
                        HTTP/1.0\n\n") #If this line does not work, write separate
                        lines for "Get .." and "Host:.."
                        # Read the response into buffer
                        # Fill in start.
                        # Fill in end.
                        # Create a new file in the cache for the requested file.
                        # Also send the response in the buffer to client socket
                        and the corresponding file in the cache
                        tmpFile = open("./" + filename,"wb")
                        # Fill in start.
                        # Fill in end.
                except:
                        print("Illegal request")
        else:
                # HTTP response message for file not found
                # Fill in start.
                # Fill in end.
    # Close the client and the server sockets
    tcpCliSock.close()
# Fill in start.
# Fill in end.
```