CSE5344 – Project 1 (Fall 2013)

Multithreaded Web Proxy Server

Objectives

- Gain exposure to socket programming
- Understand basic functionalities of a web proxy server
- Explore basic structures of HTTP GET and POST messages

Due: Nov 3, 2013, 11:59pm

Project Specification

In this project, you will implement a web proxy server. Your implementation should be able to perform the following functionalities:

- Receiving HTTP requests (GET and POST method) from a browser and forwarding them to the origin server.
- Sending corresponding HTTP responses receiving from the origin server to the client.
- Handling multiple requests at the same time (multithreaded proxy server).
- Handling errors when a client requests an object which is not available.
- Caching web pages each time the client makes a particular request for the first time and sending the cached web pages to the client when a cache hit occurs. You do not need to implement any replacement or validation of the cached web pages.

Please refer to socket programming assignment 4 of the textbook (Reference 3) for more detail on the functionalities of a web proxy server and the skeleton code in Python.

References

1. http://www.w3.org/Protocols/rfc2616/rfc2616.html: HTTP/1.1 specification
2. http://heather.cs.ucdavis.edu/~matloff/Python/PLN/FastLanePython.pdf: Introduction to Python
3. J. Kurose and K. Ross, "*Computer Networking: A Top-Down Approach*," 6th edition, chapter 2, section 2.2, 2.7: HTTP message format, Web caching, and Socket programming.

Notes

- An example syntax for running the proxy server:

  proxy_server_code_name [port_number]

The optional argument 'port_number' is the port on which the proxy server is listening to a connection from a client. If the port number is not entered, the default port 8080 is used.

- For testing POST method, you can go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html and upload a file, e.g., *alice.txt*, to this website.

Submission Guidelines

- You will turn in the complete proxy server code and any additional programs as needed. In case your code is not working by deadline, send it anyway and review it with the TA for partial credit.
- The code must be well documented.
- You must submit a readme file in text format which includes instructions on how to compile and run your programs. The readme file must also mention the development environment used as well as any packages that might be required for running the codes.
- All source files and other necessary items must be zipped into a single file with the naming convention <your-uta-id>_<your-name>.zip which is emailed to the class TA (*mqnguyen@mavs.uta.edu*) with the subject "CSE5344 - Project 1". Do NOT include any runnable executable (binary) program.
- Make sure your name and your UTA ID are listed in the readme file and in the comments at the beginning of your source files.

Additional Requirements/Instructions

- Complete documentation and coherent instructions for running the code are recommended, otherwise you may be asked to come and give the TA a demo if he is not able to execute your programs from the instructions provided.
- If you are using any code from some external source or book, you must mention it explicitly in the code as well as the readme file. If not mentioned, it will be considered as plagiarism and your project will not be evaluated.
- You can discuss with other classmates on steps/algorithms to implement the project. However, the source codes must be written by yourself.

Grading (100 points)

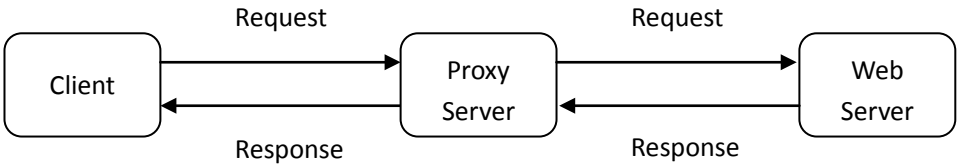
- Receive/send messages from/to a web browser successfully (20 points)
- Receive/send messages from/to web servers successfully (20 points)
- Process GET method correctly (20 points)
- Process POST method correctly (10 points)
- Handle multiple requests at the same time (multithreaded implementation) (10 points)
- Handle errors when a client requests an object that is not available (5 points)
- Cache web pages properly (10 points)
- Well-documented codes (5 points)

Socket Programming Assignment 4: HTTP Web Proxy Server

In this lab, you will learn how web proxy servers work and one of their basic functionalities – caching.

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 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 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.



Code

Below you will find the skeleton code for the client. 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.

Running the Proxy Server

Run the proxy server program using your command prompt and 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.

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 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 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

What to Hand in

You will hand in the complete proxy server code and screenshots at the client side verifying that you indeed get the web page via the proxy server.

Skeleton Python Code for the 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:
	# Strat 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)
                        fileobj.write("GET "+"http://" + filename + "
                                        HTTP/1.0\n\n")
                        # 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.
```

Optional Exercises

1. Currently the proxy server 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.
2. The simple proxy server supports only HTTP GET method. Add support for POST, by including the request body sent in the POST-request.
3. *Caching*: 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. 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. 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. You can keep this data structure in main memory; there is no need to make it persist across shutdowns.