CS 125
Web Proxy Server

1 Introduction

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 proxy server that can cache web pages. This basic server will only understand simple GET requests, but will be able to handle all kinds of objects—not just HTML pages, but also images, binaries, etc.

In normal Web operation, when the client makes a request it is sent to the Web server, which then processes the request and sends back a response to the client. To improve performance, we can insert 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. In other words, the client requests the objects via the proxy server, and the proxy will forward the client's request to the "real" Web server. The Web server will then generate a response and deliver it to the proxy, which in turn sends it to the client.

2 Code

Skeleton Python code for the proxy server is available on the class Web site. Your job is to complete the code. The places you need to fill in are marked with `# FILL IN HERE`. Each place may require one or more lines of code.

Two notes:

1. To ensure that the distributed code compiles, there are a few places where `# FILL IN HERE`' is preceded by an empty string in an assignment statement. You need to remove that empty string.

2. For the same reason, there are two "pass" statements. "Pass" is a no-op in Python; you can simply remove that line and replace it with your own code.

3 A Note About Docopt

The supplied code uses the Python `docopt` package, which is installed on Knuth, Wilkes, and the lab Macs. If you try to run the code on your own machine, you may need to install docopt first. (It's also possible to modify the code so that it doesn't depend on docopt, but since docopt is unquestionably the most amazing Python package ever written, the preferred option is to submit to the Overlord.)

4 Running the Proxy Server

Run the proxy server program from the command prompt, giving it an IP address to bind to and a port number. For the IP address, you can use "localhost" if you will be running your Web browser on the same physical machine; otherwise give the IP address of the machine you're running Python on (presumably Wilkes, so you can just use "wilkes.cs.hmc.edu"). For the port number, use your UID as given by the "id" command.

4.1 Early Debugging

To get started on your debugging, you should avoid going through a Web browser; browsers do a lot of complicated things and do them fast, which will just confuse you. Instead, use the `telnet` program (which is wonderful for network testing). If you ran your proxy on the IP address `localhost` and port 12345, type:

```
telnet localhost 12345
```

to connect to the proxy. Then type `GET http://mallet.cs.hmc.edu/index.html HTTP/1.0` and hit Enter twice. That should cause your proxy to attempt to fetch the index page from mallet.cs.hmc.edu, a page we have chosen because it is VERY simple. Only after you have gotten your proxy to work with Mallet should you try to connect with a Web browser.

4.2 Configuring your Browser

Next, you need to configure your Web browser to use the proxy server you wrote.

Explore your browser settings until you find "proxies" (see below), then enter the IP address of the machine your proxy is running on. For Wilkes, you can use its true IP address (134.173.42.167) or wilkes.cs.hmc.edu. Also enter the port you are using for your proxy. **Don't forget to undo the proxy setting after you are done testing, or your browser will stop working!**

4.2.1 Specific Browsers

With Firefox, choose Edit/Preferences, then Advanced. Click the Network tab and then, under "Connection", click Settings and choose manual proxy configuration.

In Safari, choose Safari/Preferences, then Advanced, and click "Change Settings" for Proxies. Check the box for "Web Proxy (HTTP)" and fill in the IP address in the big box and the port in the small one. (Note: you can't do this on the lab Macs, but it will work on your personal machine.)

In Internet Explorer 8, choose Tools/Internet Options, the Connections, and click LAN Settings. Check 'Proxy server,' and click Advanced. Just set your HTTP proxy, because that's all your code is going to be able to handle.

In Chrome, click on the customize button (the three horizontal lines at the top right of the window) and select Settings. Click "Show advanced settings" and then, under "Network", "Change Proxy Settings". From that point you're on your own because our version of Chrome decided to be snippy about the computer we were using. Gee, thanks, Google. Everybody else manages to make it work.

Note: Because of the CS department's firewall, you won't be able to access your proxy except in the labs. If you're working from your room or from Pomona, you can work around that problem by running Firefox directly on Wilkes.

5 Required Extensions

Once you have your proxy server running, you will need to extend it as follows:

1. Caching: A typical proxy server will cache a Web page the first time a client requests it. The basic idea is as follows: when the proxy gets a request, it checks whether the requested object is cached, and if so, 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 a full implementation, the proxy server must verify that 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.

   For our purposes, life is simpler. Add the simple caching functionality described above. You do not need to implement any replacement or validation policies. However, your implementation will need to save Web pages responses to an on-disk cache and fetch them later when you get a cache hit. To make that work, you will need to implement an 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.

2. Rewriting: One of the less savory behaviors of proxies is modifying Web pages on the fly. Usually this is done by ISPs to insert advertising, although there are more malicious applications as well. Modify your proxy so that it changes Web pages in some fashion; the details are up to you (but we encourage you to be both creative and evil!). Note that you'll need to be careful to make sure your changes are appropriate to the file type: you shouldn't do textual rewriting on JPEG images, and likewise when the server sends an image you shouldn't replace it with text (unless you also change the Content-Type, but that turns out to be harder than it sounds because sometimes the browser will be expecting an image and even if you inform it that you're sending text, it will get confused).

6 Optional Extensions

You may also wish to improve your proxy in a number of ways:

1. The skeleton code does almost 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. Real proxy servers need to be multi-threaded, because browsers request multiple pages in parallel. Use the Python threading library to make the server multi-threaded: when a request is received, create a new thread that will handle itself and then terminate. You will need to use mutual exclusion to ensure that only one thread at a time tests the cache for a file's presence.

3. The simple proxy server supports only the HTTP GET method. Add support for POST, by including the request body sent in the POST-request.

4. As given, the server only supports HTTP/1.0. If you extend it to handle HTTP/1.1, it will be more efficient.

5. As given, the proxy swallows an entire Web page before passing it to the client. That's problematic if the page is large, both because it introduces delays and because the server can potentially eat up a lot of memory. Modify your server to read small bites at a time (8K, or 8192 bytes, is a good number) and pass them on to the client right away.

7 Submission

Submit your final code as a single file using cs125submit. You should also plan to demonstrate your proxy to us in person.