**Project 1**

80 points for 325, 100 points for 425

(20 extra credit points for 325 students doing the 425 portion)

EECS325/425.

Due: 11:59pm on October 30, 2015.

Enterprise networks often employ a Web proxy as an intermediary between the employee browsers and the rest of the Internet. The proxy can serve multiple roles: block inappropriate websites, monitor employee web surfing, screen incoming pages for malware, and (as mentioned in class) provide a shared cache for Web objects. In this project, you will develop a simplified Web proxy that accepts HTTP requests from browsers, generates the corresponding HTTP requests for the same objects to the origin servers and forwards the responses to the browsers. Some instructions:

1. While in reality a proxy usually runs on a separate host, you can develop it on your own machine. To test your proxy, you can run both the browser and the proxy on the same machine (e.g., your laptop). Once everything seems to work, you must deploy your proxy on a departmental machine. We will be testing your proxy on the departmental machine, so make sure it works there.
2. The machine is a linux server named [eecslinab1.engineering.cwru.edu](http://eecslinab1.engineering.cwru.edu/); you should have an account on it already (use your Case network ID and password, should be activated on October 1). You will need to pick a port on which your proxy will listen for browser requests. Since everyone is sharing the same machine, we need to avoid port collisions. Please refer to the 325 and 425 student lists, find your order number N, and then pick port 5000+N if you are in the 325 list, and 5500+N if you are in the 425 list.
3. Please use Java for this project (to make grading feasible). Please use TCP sockets directly, do not use high-level methods and classes such as URLConnection. (You do not need to use sockets explicitly for DNS resolutions – built-in Java mechanisms are fine).
4. (This is a frequent source of bugs!) As the first step, find out if your proxy needs to change the request it receives from the browser in any way before forwarding it to the server. You can do this by learning from the Internet or by just writing, as the first step, a fragment of the proxy that reads browser requests and does nothing else, then configuring your browser to use your proxy and send a trial request. Compare (using wireshark) the requests the browser sends when configured to use your proxy with the requests it sends when it is configured to send them to origin servers directly.
5. You do not need to cache Web responses (this is a non-caching proxy that would presumably be used not for performance improvement but other reasons mentioned above). You can assume you only need to support one browser at a time. You can also assume the browser does NOT use pipelining but please allow for the possibility of it using persistent connections (this is actually the default behavior of most browsers). Further, keep in mind that browsers often open several parallel connections – your proxy must be able to handle that.
6. Please put all necessary files into your home\_directory/project1. We will start your proxy server by first compiling it by invoking “javac proxyd.java” in the “home\_directory/project1/src” directory and then executing the command “java proxyd –port 50025” (for student # 25) from the same directory. Make sure your program has the above name and accepts the above argument, (and also compiles!).

**425 only:**

1. To save on DNS resolutions, implement internal DNS caching within the proxy: save your DNS resolutions for future use and before you need one, check if you have the required resolution in the cache. For simplicity, please ignore TTL – just reuse each resolution for the default of up to 30 seconds.

Tips:

* Keep in mind that the browsers often send requests to the proxy over the same TCP connection, no matter which sites they are accessing. The proxy, obviously, will need to open different connections to different sites. But also keep in mind that the browsers often open parallel connections to your proxy.
* Since you will be using the standard socket API, your reads are blocking: each read call will only return if the socket has the requested number of bytes or if the socket has been closed. This will require a multi-threaded implementation any time you might have to read from one socket while possibly blocking on a read from another socket.
* Continuing the above thought, in particular, I recommend to have two threads for each connection from the client: one for reading requests from the client and writing requests to the server, and the other for reading responses from the server and writing them to the client (since we assume the client does not use pipelining we need to deal with only one server at a time for a given client connection, always closing the connection to the current server before contacting the new server). An alternative would have been to have one thread, which takes one request from the client, forwards it to the server, reads the response from the server and writes the response back to the client, and then looks for the next request from the client.  To do this in one thread, and to avoid blocking execution by calling read() when there are no bytes to read, the proxy must be able to tell when the client request is ended (not too hard) and when the server response is ended (harder).   Using two threads avoids this complication.
* Alternatively, you can get away with a single thread per connection if you force the server to not use persistent connection. To this end, when your proxy forwards a request from the client to the origin server, make sure your proxy looks for a "Connection" header field in the request and replaces "Connection: keep-alive" with "Connection: close" or, if there is no "Connection" header field in the request, insert "Connection: close".  This way, you force the server to always close the connection after processing the request, which simplifies things significantly. In particular, you can now detect the end of response by reading “EOF” from the socket.
* Do not use Char[] or String buffers when shuffling bytes between the server and browser sides. Use byte buffers. Remember, the server may be sending binary data (e.g., jpg files). In fact, it might be easier to read from sockets one byte at a time. Furthermore, remember that when you write something to the socket, this data does not necessarily go out to the wire immediately: the kernel can buffer it to assemble a larger chunk. This often creates a “hanging” effect, where your proxy sends the last portion of the response to the client but the client never receives it until your proxy closes the socket (which you may want to keep open to support persistent connections). The same effect may occur when you write a request to the server. The easiest solution, since we do not worry about performance, is to always flush your sockets after writing each byte (by calling flush() on the socket).
* To test your proxy, configure your browser to use it (on Safari open the preferences dialog, go to “advanced” tab, then to “Proxies: Change settings”, then check the “Web proxy” button and enter the proxy’s IP address and port number). Make sure you do not check any other buttons (in particular, leave “SSL Proxy” unchecked), otherwise your browser will try to use your proxy for HTTPS connections and your proxy will not be able to handle them.

Deliverables: Well-commented (within source code, no separate class documentation needed) code + README with (a) which port you are using (b) instructions to operate the proxy (c) which browser you used to test it with and (d) which web sites you tested on.

We will be testing your proxy by running it on [eecslinab1.engineering.cwru.edu](http://eecslinab1.engineering.cwru.edu/) and configuring the browser to send its requests to this proxy, and then accessing some web sites.  We will also read your code.

We will use the following grading instructions:

* Proxy handles a sequence of client requests (does not get stuck after the first one) 20
* Proxy transfers text data 20
* Proxy transfers binary data (e.g., images) 10
* Proxy converts absolute URLs to relative URLs 5
* Proxy can load a complex web site, e.g., [cnn.com](http://cnn.com/) 15
* Code is logical and understandable, with inlined comments 10
* (425) Implementation of DNS cache 20