CSC 8410: Concurrent Web Proxy in Java

Due April 23, 2007

Introduction

A Web proxy is a program that acts as a middleman between a Web browser and an *end server*. Instead of contacting the end server directly to get a Web page, the browser contacts the proxy, which forwards the request on to the end server. When the end server replies to the proxy, the proxy sends the reply on to the browser.

Proxies are used for many purposes. Sometimes proxies are used in firewalls, such that the proxy is the only way for a browser inside the firewall to contact an end server outside. The proxy may do translation on the page, for instance, to make it viewable on a Web-enabled cell phone. Proxies are also used as *anonymizers*. By stripping a request of all identifying information, a proxy can make the browser anonymous to the end server. Proxies can even be used to cache Web objects, by storing a copy of, say, an image when a request for it is first made, and then serving that image in response to future requests rather than going to the end server. We recommend that you write your program in stages:

- Start by coding a sequential proxy that repeatedly waits for a request, forwards the request to the end server, and returns the result back to the browser. This part will help you understand basics about network programming and the HTTP protocol.
- Extend your proxy from the first phase to keep a log of requests in a disk file.
- Finally, upgrade your proxy to use threads so that it deals with multiple clients concurrently.

Hand Out Instructions

In your `csc8410` Unix directory, copy and unpack `proxy-handout-java.tar` from `/tmp`:

```
cp /tmp/proxy-handout-java.tar ~/csc8410
tar xvf proxy-handout-java.tar
```

This will cause a number of files to be unpacked in the directory `proxy-handout-java`:

- `ParseGET.java`: A helper class for your proxy.
- `EchoClient.java`: A complete echo client example.
- `EchoServerNoThread.java`: A complete single-threaded echo server example.
- `EchoServer.java`: A complete multi-threaded concurrent echo server example.
- `Connection.java`: Thread code for the echo server.

Testing the Echo Servers

Compile the three programs either on `tanner` or on the local machine using the `javac` compiler:

```
javac EchoServerNoThread.java
javac EchoServer.java
javac EchoClient.java
```

Test either server using `telnet` as follows. Pick a random 5-digit integer N no greater than 64000 and pass it to the server as an argument:

```
java EchoServerNoThread N
```

Using telnet. Open a second terminal window on the same machine and invoke telnet at the shell prompt, passing the machine on which the server runs and the port number N as arguments:

```
telnet localhost N
```

The telnet application opens a TCP connection to port N on the local machine. Now anything that you type into the telnet window will be sent over this connection and echoed back to you by the server. To close the connection to the server, type in ”bye”.

Using EchoClient. Open a second terminal window on the same machine and invoke EchoClient at the shell prompt, passing the machine on which the server runs and the port number N as arguments:

```
java EchoClient localhost N
```

The EchoClient application opens a TCP connection to port N on the local machine. Now anything that you type in will be sent over this connection and echoed back to you by the server. To close the connection to the server, type in ”bye”.

Repeat the test for `EchoServer`.

Part I: Implementing a Sequential Web Proxy

In this part you will implement a sequential proxy. Your proxy should open a socket and listen for a connection request. When it receives a connection request, it should accept the connection, read the HTTP request, and parse it to determine the name of the end server. It should then open a connection to the end server, send the request (altered as described in subsequent sections), receive the reply, and forward the reply (unaltered) to the browser, if the request is not blocked.

Since your proxy is a middleman between client and end server, it will have elements of both. It will act as a server to the web browser, and as a client to the end server. Thus you will get experience with both client and server programming. Use `EchoServerNoThread.java` and `EchoClient.java` as guiding examples.

HTTP Requests to Servers

HTTP requests are of many types, but we will only concern ourselves with the GET request in this assignment. Suppose that you type the URL `http://www.google.com/index.html` into your Web browser. The Web browser will open a TCP connection to `www.google.com` on port 80 and send the request (followed by a number of lines not shown here):

```
GET /index.html HTTP/1.0\r\n
```

The `\r\n` sequence represents the ASCII character 13 (`'\r'`) followed by ASCII character 10 (`'\n'`). In response, the web server will locate the file called `/index.html` and transmit its contents to the web client. Try out the following:

1. Telnet to your favorite Web server, for example

   ```
   telnet www.google.com 80
   ```

   The telnet application opens a TCP connection to port 80 (default HTTP port) on `www.google.com`. Anything that you type in next will be sent over this connection.

2. Type in an HTTP GET request:

   ```
   GET /index.html HTTP/1.0
   ```

   and hit Enter TWICE. In this way you send this minimal (but complete) GET request to the HTTP server. It is possible that you won't see your GET request on the screen as you type it in. If you make a typing error, you will get a "`Bad Request`" response from the server (in that case, start over).

3. Look at the response send by the HTTP server! It should be identical to the text you see when selecting "View/Source" on the top bar menu of your browser.

HTTP Requests to Proxies

Web browsers can be configured to use a proxy server (see the Hints section) instead of sending requests directly to Web servers. In this case, when you try to load a page such as

```
http://www.google.com/index.html
```

the browser contacts the designated proxy server and asks it for the page as follows:

```
GET http://www.google.com/index.html HTTP/1.0
```

Notice the difference: instead of asking just for the resource `/index.html`, the web browser asks for the full URL `http://www.google.com/index.html`. This is to give the proxy enough information to fetch the file `/index.html` from `www.yahoo.com` on behalf of the web browser and deliver it as a result, or do whatever it wants to do.

To fetch the file, the proxy extracts the file name from the GET line and forwards the request

```
GET /index.html HTTP/1.0\r\n
```

along with subsequent lines, as they came from the browser; the last line is always empty (`\r\n`).

Port Numbers

You proxy should listen for its connection requests on the port number passed in on the command line:

```
bash$ java ProxyServer 12345
```

A unique port number has been assigned to you below. This is to ensure that your port number is not currently being used by other students or system services (see `/etc/services` for a list of the port numbers reserved by system services).

Name	Port Number
Agya Adueni	10001
Bi, Xiaoyan	10002
Bouton, Kevin	10003
Bulava, Jonathan	10004
Gapinski, Bogdan	10005
Giordano, Russell	10006
Gladstone, Andrew	10007
Grande, Jacquelyn	10008
Kuchibhotla, SuryaKamalKiran	10009
Malladi, Venkata	10010
Mizas, Timothy	10011
Obusek, Andrew	10012
Paudel, Prabin	10014
Price, Donald	10015
Repole, Paolo	10016
Tao, Tao	10017
Ward, Christopher	10018
Zhu, Hui	10019

PART II: Logging Requests

Your proxy should keep track of all requests in a log file named `proxy.log`. Each log file entry should be a line of the form:

```
Date: browserIP URL size
```

where `browserIP` is the IP address of the browser, `URL` is the URL asked for, `size` is the size in bytes of the object that was returned. For instance:

```
Sun 26 Feb 2007 02:51:02 EST: 128.2.111.38 http://www.xxx.com/ 34056
```

Note that `size` is essentially the number of bytes received from the end server, from the time the connection is opened to the time it is closed. Only requests that are met by a response from an end server should be logged.

Part III: Implementing Concurrency with Threads

Real proxies do not process requests sequentially. They deal with multiple requests concurrently. Once you have a working sequential logging proxy, you should alter it to handle multiple requests concurrently. The simplest approach is to create a new thread to deal with each new connection request that arrives:

1. Accept a TCP connection from a real HTTP client (running in a Web browser)
2. Spawn a new thread to handle the connection
3. Go to Step 1

Use `EchoServer.java` as a guiding example. With this approach, it is possible for multiple peer threads to access the log file concurrently. Thus, you will need to use a semaphore to synchronize access to the file such that only one peer thread can modify it at a time. If you do not synchronize the threads, the log file might be corrupted. For instance, one line in the file might begin in the middle of another.

Outline for Part I

1. Read a request from the client and parse the first line. If it is not a GET request, ignore it (close the connection). Let us use this example (first line of a GET request):

   ```
   GET http://www.villanova.edu:80/index.html HTTP/1.0\r\n
   ```

   Parse this line to determine the hostname (`www.villanova.edu`), the port number (default `80`, if none specified) and the pathname of the desired HTML object (`/index.html`). This line tells your proxy to contact the server running on the host `www.villanova.edu` on port `80`.

2. Establish a TCP connection to the Web server determined in Step 1 (`www.villanova.edu`).

3. Forward the GET request and the following HTTP header lines (using only the pathname instead of the full URL) to the Web server.

   ```
   GET /index.html HTTP/1.0 \r\n
   Accept: */*  \r\n
   Accept-Language: en-us \r\n
   If-Modified-Since: Mon, 17 Mar 2007 13:55:31 GMT \r\n
   If-None-Match: "1763b-235a-3d85e2d3" \r\n
   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n
   Host: www.csc.villanova.edu \r\n
   Proxy-Connection: Keep-Alive \r\n
   \r\n
   ```

   Make sure that the header ends with a blank line, to signal the end of the request.

4. Read everything sent back by the Web server and send it back to the client. Since binary files (such as images) may get sent to the client, use a loop similar to the one below (instead of `readLine`):

```
InputStream netIn;          /* to read from the Web server */
OutputStream clientOut;     /* to write to the Web client */

/* Initialize netIn, clientOut */

byte [] buffer = new byte[4096];
int bytes_read;
while ((bytes_read = netIn.read(buffer)) != -1)
{
    clientOut.write(buffer, 0, bytes_read);
    clientOut.flush();
}
```

5. Close the connections to the server and client and exit.

Evaluation

You will be evaluated on the basis of a demo to your instructor. Evaluation criteria include:

- Basic proxy functionaliy. Your sequential proxy should correctly accept connections, forward the requests to the end server, and pass the response back to the browser, making a log entry for each request. Your program should be able to proxy browser requests to the following Web sites and correctly log the requests:
  - `http://www.yahoo.com`
  - `http://www.aol.com`
- Handling concurrent requests.
- Style. Your code should begin with a comment block that describes in a general way how your proxy works. Furthermore, each function should have a comment block describing what that function does. Furthermore, your threads should run detached, and your code should not have any memory leaks.

Hints

- The best way to get going on your proxy is to start with the basic echo server from your handout and then gradually add functionality that turns the server into a proxy.
- Initially, you should debug your proxy using telnet as the client.
- Later, test your proxy with a real browser. Explore the browser settings until you find “proxies”, then enter the host and port where you’re running yours. With Netscape, choose Edit, then Preferences, then Advanced, then Proxies, then Manual Proxy Configuration. In Internet Explorer, choose Tools, then Options, then Connections, then LAN Settings. Check ’Use proxy server,’ and click Advanced. Just set your HTTP proxy, because that’s all your code is going to be able to handle.

- Since the focus of this assignment is on concurrency and network communication, we provide you with the helper routine `ParseGET`, which extracts the hostname, path, and port components from a URL, and reformats the first GET line for the end server.
- You may find it useful to assign each thread a small unique integer ID (such as the current request number) and then pass this ID as one of the arguments to the thread routine. If you display this ID in each of your debugging output statements, then you can accurately track the activity of each thread.
- Since the log file is being written to by multiple threads, you must protect it with mutual exclusion semaphores whenever you write to it.

Handin Instructions

- Remove any extraneous print statements.
- Make sure that you have included your identifying information in your proxy code.
- Hand in a printout of your code and a README file that lists any known debugs or deviations from the specifications.
- Schedule a demo time with me.