MPCS 54001, Spring 2016
Project #2
Due: Wednesday, June 1, 11:59:59pm

NOTE: Complete this project in the same team of two as for Project 1.

Tasks

Extend your basic web server from project 1 to:

- support encrypted connections via HTTPS, using TLS (SSL 3.0)
- support persistent connections
- listen simultaneously on HTTP and HTTPS ports.

Hints

SSL

You will need a server certificate in order to accept HTTPS connections. The `keytool` command will create a basic one that works well with the Java libraries. Run this in your project 2 directory:

```
keytool -genkey -alias project2 -keystore server.jks
```

This generates a "JKS" (short for Java Key Store) type KeyStore, with the certificate it generates of type "SunX509" (as far as the KeyManagerFactory class is concerned). Remember the password that you choose when you create the certificate. You don't really need to fill out all the fields for your name, organization, location etc that you'll be prompted for.

The following standard Java classes will be useful:

- KeyStore
- KeyManagerFactory
- SSLContext
- SSLServerSocket

Briefly, you need to create an SSLServerSocket, rather than a ServerSocket. The classes above (and your certificate) will let you do that. Feel free to use the Internet as a resource (particularly the Oracle documentation).

Although it's normally a poor security practice, in order to aid grading, **hardcode your keystore's password in the source itself (don't rely on the TAs or me having to type in any passwords when we start your server). We should just need to have the .jks file.**

Once you've figured out how to create an SSLServerSocket rather than a ServerSocket, to test things out remember to specify "https://" rather than "http://".

Persistent Connections

“Persistent connections” means the client can request that connection remain open for transferring more files. As is standard in HTTP 1.1, connections should be considered persistent unless specified otherwise. See http://en.wikipedia.org/wiki/HTTP_persistent_connection.

The converse of this is also true: if the client sends the “Connection: close” header, the server should close the connection when it’s done.

Listening Simultaneously
To listen simultaneously for both encrypted and unencrypted connections, you’ll need to define a second flag (--sslServerPort) and specify both that and --serverPort at server startup. Bind the HTTP server socket to the --serverPort value, and bind the HTTPS server socket to the --sslServerPort value.

You actually need to be able to accept() an HTTP and HTTPS connection at the same time without blocking. The simplest way to accomplish this is by multithreading your server to have one thread handling HTTP requests, and one thread handling HTTPS requests. Think about how to structure your source code + classes in order to avoid silly code duplication. You’ll probably want to look into the Thread and Runnable classes to help you out here.

You might be thinking about using select(), but Java’s support for select() is poor when SSL sockets are in the mix (there is no SslServerSocketChannel). If I was writing this in C++ or Go I’d use select(), **but I strongly advise against going down that path when working in Java** (that way lies madness).

Deliverables

Build a tarball of your Java files, your Makefile, **and your keystore file**:

```
tar cvf cnetID1_cnetID2_project2.tar *.java Makefile server.jks
```

where cnetID1 and cnetID2 are your usernames.

Email the tarball to Kevin (kkrafthe@uchicago.edu). In the body of your email, indicate both of the members of your group.

Grading

We will compile your submissions + grade them against an automated web client I wrote.

General grading rubric is as follows:

- Compiles, starts, binds to a TCP port: **10 points**
- HTTPS GET requests return 200 + the data for all existing files in tree: **10 points**
- HTTPS requests return 301 for all paths specified as redirects in redirects.defs: **10 points**
- HTTPS requests return 404 for all non-existent paths: **5 points**
- Any POST request or other unknown op code returns a 403: **10 points**
- Server can handle multiple requests in succession without restarting (loops around and

accepts the connection again: **10 points**

- Client sending “Connection: Keep-Alive” results in a persistent connection: **15 points**
- Client sending “Connection: close” results in a non-persistent connection: **10 points**
- Accepting both HTTPS and HTTP connections simultaneously: **10 points**
- Style points, instructor discretion (efficiency, code correctness, or other considerations e.g., you didn’t just hardcode responses to the files in the tarball, but actually read the files): **10 points**

for a total of 100 points.