2506ICT Computer Networks and Security Assignment II

Title: Creating a Proxy Cache Server
Objective: To get a deeper insight into client server communications through implementation of a HTTP Proxy Cache Server

Deadline: Midnight of Thursday 20th Oct 2016
Total Mark: 100

Why This Assignment

This assignment is designed to help you better understand the dynamics of client/server communications and give you an experience of network programming, a skill that you might find it very useful in your future career as more and more applications are now networked-based. The language of the choice for this assignment is **Java**. Here are few reasons as why Java is a very popular programming language:

1. Practicality

James Gosling has described Java as a "blue collar" programming language. It was designed to allow developers to get their job done with the minimum of fuss, whilst still enabling developers to pick up someone else's (or even their own) code at a later date and understand what it's supposed to do.

2. Backwards compatibility

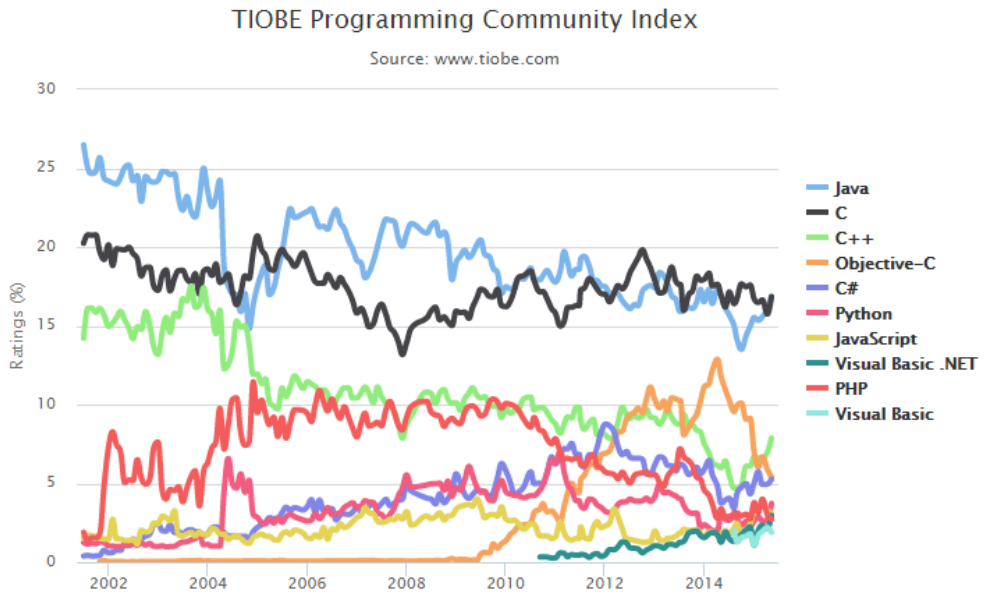
Sun and subsequently Oracle have made huge efforts to ensure that code written for one version of Java will continue to run unchanged on newer versions. There's nothing worse than taking code that works and having to change it to make it work on a newer version of the platform. That's just wasted time.

3. Scalability/Performance/Reliability

With over twenty years and thousands of man-years of development, Java is a rock-solid platform that performs on a level that can match or even exceed that of native code (thanks to some of the optimisations made by the JVM using dynamic rather than static code analysis). When it comes to scalability, just look at some of the large enterprises using Java: Twitter (who moved off Ruby-on-Rails to the JVM because RoR wouldn't scale), Spotify, Facebook, Salesforce, eBay and, of course, Oracle. Hadoop, Cassandra and Spark, the basis of most big data projects, are either written in Java or Scala or run on the JVM.

4. Popularity and Job Market Opportunities

TIOBE programming community index is a measure of popularity of programming languages, created and maintained by the TIOBE Company based in Eindhoven, the Netherlands. Looking at the TIOBE graph there is a significant upswing in Java popularity since October 2014, which is shortly after the launch of JDK 8. JDK 8 was a big change for developers. Suddenly Java developers could do things in a more functional way without having to learn a whole new language like Scala.



Also reports like this indicate that the job market is constantly looking for talent with Java skills.

Prerequisite:

As mentioned at beginning of the course, the prerequisite of this course is that you have at least some familiarity with the concept of programming. However, as a part of this assignment, support will be provided to you to getting started. **However, this requires you to start as early as you can and not leave the assignment until close to the deadline.** Also additional support may be provided on the Yammer page of the course, so I strongly encourage you to sign up to the Yammer group (By accepting the invitation email that I have sent you at the beginning of the semester), if you have not done so yet.

What to Submit

You need to submit two items:

(1) The actual java source code, and

(2) A written report explaining how your program works.

Item 1: (weighting 50-85/100, depending on which level of implementation, see description below). For item (1):

(a) Your program **needs** to be able to run on Window OS, otherwise, it may not be marked.

(b) You **must** verify that your code compiles and runs as expected, and your code submission should be self-contained and **should not** depend on external libraries (-5 marks penalty).

(c) You codes **should be commented** where appropriate to ensure readability (-3 marks penalty).

Item 2: (weighting 15/100)

You should hand in a report (**maximum five A4 pages excluding cover page & appendices, in MS word format or pdf**) explaining the logic and flow of your program. Your report should be well-structured and easy to read. A person should understand what you are trying to do and what your program can do after reading your report. Also, it should:

(a) Contain **screen shot** of your program runtime (-3 marks penalty).

(b) Contain details on how to compile and run your program (-2 marks penalty)

(c) You should **indicate the level** of implementation (-2 marks penalty).

(d) You must have a cover page (-2 marks penalty).

(e) You should also include a **listing of your code in the appendix**, and **highlight** all places in the code that is written by you (-5 marks penalty).

Handing in:

You should submit your report and codes online in a single zip folder. **Your zip folder should be named as**: **2506ict_as2_firstname_lastname_studentID.zip** (replace the firstname, lastname, studentID with yours).

Late submission may incur a penalty as per the course outline. This is **individual** work and **plagiarism will be dealt with seriously**.

Note: All penalty marks specified above may be deducted on top of the overall marks you are given.

Programming Assignment: Proxy Cache

In this lab you will develop a small web proxy server which is also able to cache web pages. This 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. There will be three levels of complexity for your web proxy server, and they carry different marks. Your final mark will depend on which level of complexity you have implemented. The three levels are as follows:

Level 1: Simplest proxy server (50 marks)
This is the simplest web proxy server you would need to implement. Your web proxy server should be able to receive requests from client, forward them to the web server, read replies from web server, and return them to the client. There is actually no caching of the retrieved objects!

For this level of implementation, the required error handling in the proxy is very basic and you do not need to inform the client about errors in the request. When there are errors in the request, the proxy will simply stop processing the request and the client will eventually get a timeout.

The proxy works as follows:

1. The proxy listens for requests from clients
2. When there is a request, the proxy spawns a new thread for handling the request and creates an HttpRequest-object which contains the request.
3. The proxy sends the request to the server and reads the server's reply into an HttpResponse-object.
4. The thread sends the response back to the requesting client.

Level 2: Proxy server with caching (70 marks)
In this web proxy server, besides having the functionality of the level 1 implementation, your web proxy server is required to cache the retrieved objects. Your implementation will need to be able to write responses to the cache and fetch them from cache when you get a cache hit. For this you need to implement some internal data structure (for example HashMap or similar) in the proxy to keep track of which objects are cached. You can keep this data structure in main memory; there is no need to make it persist across shutdowns.

The proxy works as follows:

1. The proxy listens for requests from clients
2. When there is a request, the proxy spawns a new thread for handling the request and creates an HttpRequest-object which contains the request.
3. The proxy then checks if the requested object is cached, and if yes, reads the object from the cache into an HttpResponse-object, without contacting the server.
4. If the requested object is not cached, the proxy sends the request to the server and reads the server's reply into an HttpResponse-object. A copy of the response is also cached for future requests.
5. The thread sends the response back to the requesting client.

Level 3: Proxy server with caching and validation (85 marks)
In this level of complexity, besides having the caching capability as described in level 2 above, your web proxy server would also check whether the object in the cache is still fresh (i.e. whether it is still up to date). Your web proxy server would use validators to check whether the object has changed since it was last cached. In modern web servers using HTTP1.1 protocol, the two validators are: (1) *Last-Modified* header, and (2) *ETag* header.

The Last-Modified header indicates the time the object was last changed. When a cache has an object stored that includes a Last-Modified header, it can use it to ask the web server if the object has changed since the last time it was seen, with an If-Modified-Since request.

In addition to the Last-Modified header, HTTP1.1 also introduces a new kind of validator called the *ETag*. ETags are unique identifiers that are generated by the server and changed every time the object is modified. Because the server controls how the ETag is generated, web proxy caches can be sure that if the ETag matches when they make a If-None-Match request, the object really is the same.

Most modern Web servers will generate both ETag and Last-Modified headers to use as validators for static content (i.e., files). To see how these appeared in the HTTP1.1 header, you can examine the "http-ethereal-trace-1" trace using Wireshark as in the Wireshark_HTTP lab.

The proxy works as follows:

1. The proxy listens for requests from clients
2. When there is a request, the proxy spawns a new thread for handling the request and creates an HttpRequest-object which contains the request.
3. The proxy then checks if the requested object is cached, and if yes, check whether the object is still fresh. If the object is still fresh, the proxy reads the object directly from the cache into an HttpResponse-object. If the object is out dated, the proxy reads the server's reply into an HttpResponse-object and updates the cached copy of the response for future requests.
4. If the requested object is not cached, the proxy sends the request to the server and reads the server's reply into an HttpResponse-object. A copy of the response is also cached for future requests.
5. The thread sends the response back to the requesting client.

For implementation at level 2 & 3, an invalid request from the client should be answered with an appropriate error code, i.e. "Bad Request" (400) or "Not Implemented" (501) for valid HTTP methods other than GET. Similarly, if headers are not properly formatted for parsing, your client should also generate a type-400 message. See RFC 1945 section 9.5 - Server Error.

For implementation at level 2 & 3, when testing your code you need to empty your web browser's cache every time you access a web page so that it is always accessing your web proxy server for the page rather than using its own cached copy.

Similar to level 2 implementation, you can keep this data structure in main memory; there is no need to make it persist across shutdowns.

Note: Minimal help beyond what is given here will be provided for level 2 & 3 implementation, to gain the extra marks present in level 2 & 3 implementation, you would need to do your own research to figure out how to implement the additional functionality in these levels.

The Provided Code

The provided code is divided into three classes as follows:

- **ProxyCache** holds the start-up code for the proxy and code for handling the requests. The basic structure in this code works for level 1, for level 2 and 3 implementation, you will need to modify the flow structure of the code.
- **HttpRequest** contains the routines for parsing and processing the incoming requests from clients.
- **HttpResponse** takes care of reading the replies from servers and processing them.

Your task will be to complete the proxy so that it is able to receive requests, forward them, read replies, and return those to the clients. You will need to complete the classes **ProxyCache**, **HttpRequest**, and **HttpResponse**.

For the simplest implementation (i.e. level 1), the places in the three classes where you need to fill in code are marked for you with /* Fill in */, and each place may require one or more lines of code.

For level 1, most of the error handling in the proxy is very simple and it does not inform the client about errors in the request. When there are errors, the proxy will simply stop processing the request and the client will eventually get a timeout.

For level 2 and 3 implementation, you need to put in extra codes in the three classes at appropriate places (and possibly modify the flow of the program) to implement the added functionality. You also need to handle errors better instead of just leaving it till timeout.

Programming Hints

Most of the code you need to write relates to processing HTTP requests and responses as well as handling Java sockets.

One point worth noting is the processing of replies from the server. In an HTTP response, the headers are sent as ASCII lines, separated by CRLF character sequences. The headers are followed by an empty line and the response body, which can be binary data in the case of images, for example.

Java separates the input streams according to whether they are text-based or binary, which presents a small problem in this case. Only DataInputStreams are able to handle both text and binary data simultaneously; all other streams are either pure text (e.g., BufferedReader), or pure binary (e.g., BufferedInputStream), and mixing them on the same socket does not generally work.

The DataInputStream has a small gotcha, because it is not able to guarantee that the data it reads can be correctly converted to the correct characters on every platform (DataInputStream.readLine() function). In the case of this assignment, the conversion usually works, but the compiler will flag the DataInputStream.readLine()-method as deprecated and will refuse to compile without the -deprecation flag.

It is highly recommended that you use the DataInputStream for reading the response.

Compiling the Proxy

As explained above, the proxy uses DataInputStreams for processing the replies from servers. This is because the replies are a mixture of textual and binary data and the only input streams in Java which allow treating both at the same time are DataInputStreams. For compilation, you will need to use the -deprecation argument for the compiler as follows:

```
javac -deprecation *.java
```

Running the Proxy

Running the proxy is as follows:

```
java ProxyCache port
```

where *port* is the port number on which you want the proxy to listen for incoming connections from clients. As your web proxy server is very primitive and only handle the GET request, you should try to access only simple web pages to test your program.

Configuring Your Browser

You will also need to configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in "Internet Options" in the Connections tab under LAN Settings. You need to give the address of the proxy and the port number which you gave when you started the proxy. You can also run the proxy and browser on the same computer without any problems – in this case, the IP address of your proxy would be *localhost* or 127.0.0.1.