G52APR Coursework 1 (2014/15)

HTTP Server using Java.

Overview

In this coursework you will implement a HTTP/1.0 server in Java. Initially this may seem like a daunting task, but by breaking the server down into smaller, related, but broadly independent parts (to be detailed later in this document) a functioning HTTP/1.0 server should be attainable.

In brief, your program should open a listen socket that HTTP/1.0 compatible clients (e.g. a web browser such as Mozilla Firefox) may connect to in order to request pages and/or files. Ultimately you should be able to use a web browser to connect to your HTTP/1.0 server in the same way that you would use it to connect to any web server.

HTTP/1.0 is defined in RFC 1945, available from http://tools.ietf.org/pdf/rfc1945.pdf. It is important that you read this document thoroughly. Although it is long, it is written in fairly nontechnical language. It is recommended that you print this document, bind it in some manner, and scrawl notes all over it: RFC 1945 will be your reference manual for the coursework, and thus it is essential that you become well acquainted with its content.

Note: The latest version of HTTP is HTTP/1.1, defined in RFC 2616. HTTP/1.1 is a significant extension of HTTP/1.0 (RFC 2616 is approximately 3 times the length of RFC 1945) and is beyond the scope of what could be reasonably expected as a second year coursework. In contrast, HTTP/0.9, which is also defined in RFC 1945, is trivial. RFC 1945 states that HTTP/1.0 servers must also implement HTTP/0.9, and therefore your server must too.

To aid with the design of your system, we have broken the coursework into component parts, each of which can be completed (more or less) independently, and then glued together to form the final system. Deadlines for these parts are as stated on the Moodle web-site for this module.

Part 1 (30%) – Client, Request Handler and Network – This part – Due 24/10/14

This part of the coursework is made of three sections.

1) The client is a simple HTTP 1.0 client which will be used to test your application. This will (probably) use apache's HTTPClient libraries.

2) The Request Handler Converts the HTTP/1.0 and HTTP/0.9 requests coming from the network connection into method calls, retrieves any required data from the File Server, and generates the responses to send back to the client.

3) The network section of your server is responsible for opening a network connection to listen for incoming connections. This should support more than one concurrent connection so will need to use threading.

Part 2 (30%) – File Server & HTML parsing for the client – Coming soon – Due 14/11/14

The file server is responsible for getting and serving files which will be stored in a directory on the FileStore. This part of the program must respond to a method called from the RequestHandler (part 1) and return values as appropriate. Your fileserver will implement the IServe interface.

Part 3 (30%) – Logger – Coming soon – Due 28/11/14

Largely unrelated to the HTTP/1.0 protocol (and only touched upon extremely lightly in RFC 1945) logging of requests and response headers is implemented by a vast majority of HTTP servers. Your server must be capable of logging data on requests and their associated responses, to a text-based log file. However, your work should be easily extensible if, for future work, the logger should be changed to, say, a database implementation.

Part 4 (10%) – Cache – Coming soon – Due 5/12/14

The cache, also unrelated to the HTTP/1.0 protocol is responsible for reducing the number of file reads required of the FileStore. It may cache the responses, along with the last-modified date of the files, allowing the response to be retrieved from the cache rather than the FileStore if the file has not changed. This is not the same as a client side cache, such as those implemented in modern browsers, as this is a cache on the server side.

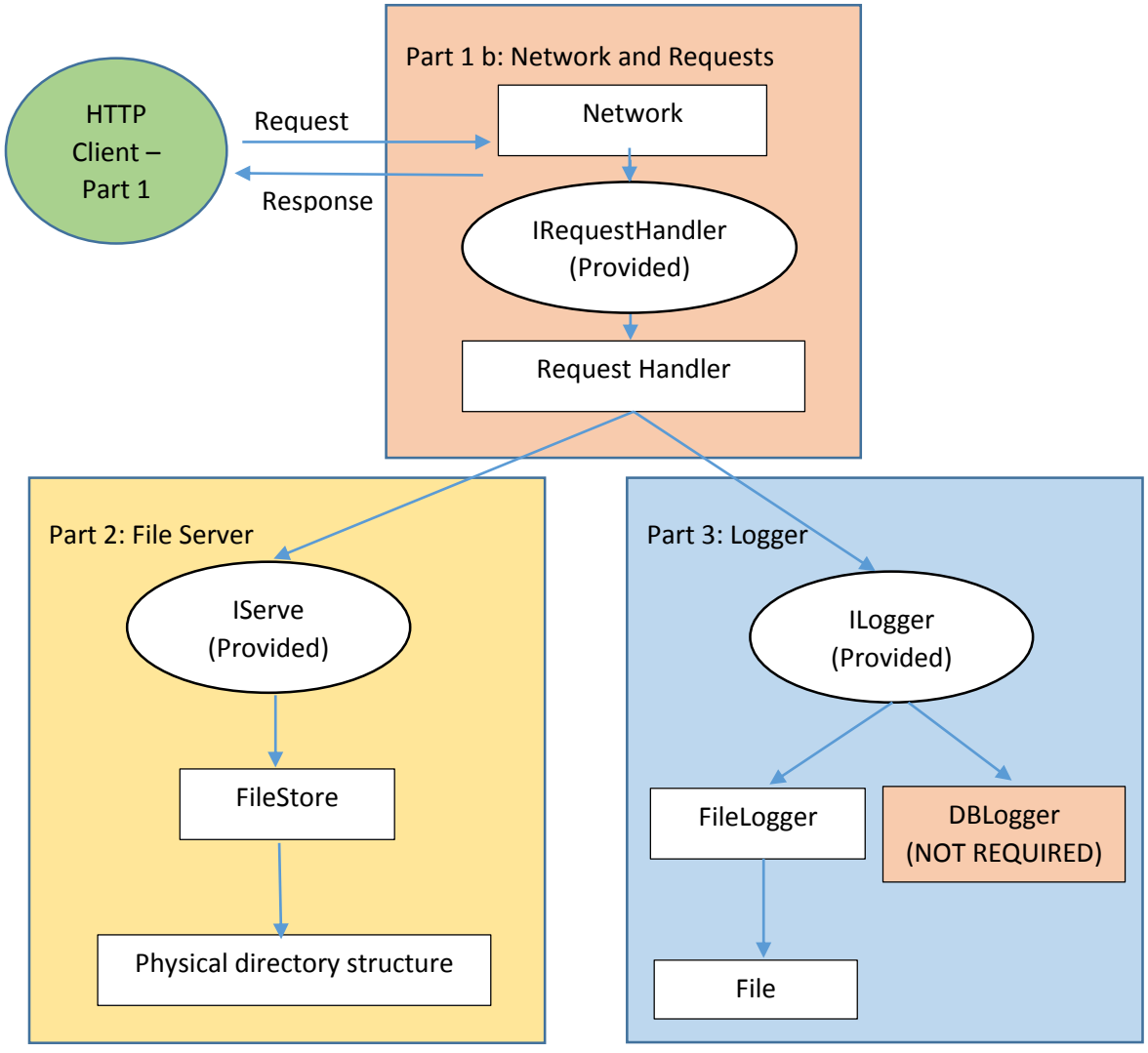

Diagram 1 – An overview of how parts 1, 2 and 3 work together. (Note that the cache is not shown – you need to decide where that is best to go.)

Each active network connection will have a Request Handler object whose methods it will call to pass data containing the HTTP request received from the client. However, the interface between the RequestHandler and the File Server, and the interface between the Request Handler and the Logger will need to be more formal, using normal Java interfaces. This will allow you to develop the Request Handler independently of both the File Server and the Logger. Once you have worked out what methods each interface will need to support you can create dummy, minimal implementations of both interfaces which you can use to test your Request Handler. In effect, we are using the Strategy and Observer patterns here. We could vary how your server records its logs just by creating a new implementation of the Logger interface (which say could log to a laser printer, or MySQL DB).

Suggested Program Structure

Diagram 1 illustrates how it is recommended that the components of your system interact. The rectangular boxes represent the absolute minimum requirement for classes: you may find that breaking some or all of these into more classes is sensible. Remember a sensible structure to your code will attract better marks. IRequestHandler, IServe and ILogger are Java interfaces. These have been defined for you, and you should not change these (available from Moodle). The methods in these interfaces will be used for testing, and so you must ensure they are implemented as expected and described in the interface.

Part 1 - Client, Request Handler and Network

These three sections are to be handed in simultaneously, though you should develop and test them independently before you then test them in combination. The client and server parts should be submitted separately (see the TMA submission site). It is recommended that you create the client first as it will allow you to test your network and request handler (in fact the main purpose of the client is for testing).

Expected development environment.

For the whole of this coursework we expect that you will be developing using Eclipse. This will make the coursework easier to mark as all projects will be in the same structure, but this is also useful to you as this is one of the most commonly used IDE's and it is good to know how to use it. To set up your project, please follow these instructions:

Step 1: Download the .jar library files which you will need from the Moodle page. These will be zipped together for download purposes.

Step 2: Start Eclipse. On the University systems, eclipse is located at C:\eclipse-SDK-4.2-win32-x86_64\eclipse\eclipse.exe. (Or something similar, this may be updated with a newer version)

Hint: You may want to make a shortcut to this on your desktop.

Step 3: Create a new java project (File>New>Java Project). Enter a sensible project name. Press "Next" rather than finish.

Step 4: On the next screen (Titled New Java Project), select the libraries tab. Select "Add External JARs". Select all of the libraries downloaded from Moodle. These are now added to your project. Press finish.

Step 5: You can see the package explorer on the left. All of the code you create will be in the "src" folder. Find the interface IG52APRClient and drag it into this folder. When asked, select "Copy files". This will add the interface to your project. Do the same for the class G52APRClient. It is important that you use this class to create your client. The bare bones have been outlined for you, it implements the IG52APRClient interface, but none of the methods have been created (This is for you to do).

Step 6: Any new classes you need to make, you should do so in the same package. Right click on "Default Package", and select New > Class. Do not use any package other than the default package for this exercise. (Note, you should not need to have "package" anywhere in your source code).

When you have finished developing your solution, it is expected that you will export your project from eclipse. To do this, click File > Export > General > File System. Make sure all your classes are included from the src folder, and not in any sub directories. Zip this folder to submit it. The server side of your coursework should be as a *separate* project, which will not require the external libraries. This should be set up in the same way, excluding Steps 4 and 5.

Part 1a - Client

For this section of the coursework you are to write a client which will connect to a (existing) HTTP server. You should use the class already created for you to create G52APRClient objects. These will be responsible for sending three types of HTTP/1.0 request which are HEAD, POST, and GET. Make sure to read the comments for each method on the interface carefully, and make sure to also read the HTTP/1.0 specifications.

> **Note**: To create the client, it is recommended that you use libraries from org.apache.http. These libraries are available on Moodle, **please be sure to use the ones from Moodle to ensure everyone is using the same version**. You should already have included these files in your project during part 1.1. You do not have to use these libraries, but it is strongly recommended you do so.
>
> **Hint:** You will find that using apache's DefaultHttpClient will send HTTP 1.1 requests. In order to force the client to make HTTP 1.0 requests you should use something along the lines of:

```
HttpGet getRequest = new HttpGet(url);
getRequest.setProtocolVersion(HttpVersion.HTTP_1_0);
```

You can partially test your client by sending GET and HEAD requests to http://cs.nott.ac.uk/~cah/. You will find that to connect to external sites from the University you may need to set up proxy settings, for this exercise you are not expected to do so. If you are working from home, you can also test our your POST method by using http://posttestserver.com/ , however you can also test this using your server part 1.

It may seem a daunting task to create a HTTP client, but the majority of the work is already done by the apache library. If you get stuck, the library has complete documentation here:

http://hc.apache.org/httpcomponents-client-ga/index.html

And a tutorial on creating the HTTP client here:

http://hc.apache.org/httpcomponents-client-ga/quickstart.html

Part 1b – Request Handler (The first part of the server – in a separate eclipse project)

For this section of the coursework you are to write the request handler. This will handle the HTTP/1.0 and HTTP/0.9 requests that are sent to your HTTP server over the network from the HTTP client. It must interpret the request, act upon it, and send back a valid response. Later, once the file server is completed we will be able to create full responses. For now you can return fixed Strings to report what request has been received and the action that would be taken (details shortly). Before you write the part of the program that creates the HTTP server's network socket and handles incoming connections from the various HTTP clients, you can create a test harness to enter valid HTTP request strings with fixed tests, the keyboard or otherwise. You may assume that the Network section will have already converted the commands from a sequence of octets into an array of Java bytes. You will therefore require a method of the form public byte[] processRequest(byte[] request) (this is given in the interface) that takes an array of bytes representing the request, and returns an array of bytes representing the response. This method should have nothing to do with the network and should be able to run independent of any network requirements, the network part of the coursework is responsible for returning the response, not the RequestHandler.

> **Hint:** to convert from a Java String into an array of ASCII-encoded bytes you can use the getBytes method on the String class, e.g.
> String myString = "Hello, there.";
> byte[] myBytes = myString.getBytes("US-ASCII");
>
> **Hint 2**: To convert from an array of bytes to a String:
> String myString = new String(myBytes);

At this stage you will not have your file server or logger set up, but it is still important to test that your request handler is calling the correct methods. Your request handler should have a method for each of GET, HEAD and POST. As these methods are not yet implemented, they should respond with the correct selection from the following :

```
HTTP/1.0 501 GET Not Implemented
HTTP/1.0 501 HEAD Not Implemented
HTTP/1.0 501 POST Not Implemented
```

In your RequestHandler, instead of handling this in one method, you should make sure that each request is going to a different method to handle it – this will make the work easier for future parts.

.

Part 1c - Network

The network part of this coursework is how your server will interact with the client. This part of the coursework needs to accept multiple connections and receive requests from the client. Your requests are to be passed to the Request Handler. The response from your request handler should then be sent back over the connection. For this part of the coursework you will need to use threading and sockets to make sure that your server can handle multiple connections. Your program should handle any network problems gracefully. The first argument passed into your server's Main() should be the port – this is extremely important. Although usually web servers will use port 80, due

to permissions on the university machine you can only open certain ports, for testing you may use 4444**, but ensure that it is passed in as an argument to the main method and is not hard coded into your server.**

You will need to start this part by looking at the lecture slides on sockets and the oracle documentation may also be a very useful resource:

http://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

Testing

Please supply us with any evidence you feel appropriate to show how you have tested your code. This might be a short commentary on how you tested it, the output of test results, an indication of where in your source code to see the testing or whatever you feel necessary and useful Please restrict this to a maximum of 500 words. You should include two files, one file named clientTest.txt in the client project and one named serverTest.txt in the server project. Please only submit text files with no formatting, ie do NOT submit pdf, Word documents or anything else other than raw text. Maybe use telnet to test your server network – or you can even use a browser at this stage. Telnet is an excellent way to also test other servers, to see how they respond.

Marks

Each part of the coursework carries a weighting. All of the sections of part 1 will add up to 30% of your final G52APR grade.

Assessment Criteria:

The main parts of the assessment criteria are:

- Correctness (40%) – Does your work do what it should? This will be tested by us, with a series of JUnit tests. You should test this yourself before submitting. An example JUnit class, with one test is available on Moodle.
- Readability, extensibility and code style (30%) - As with Coursework 0, several rules will be used to check for bad practises such as methods with a very high complexity that should be broken down, if statements that don't have braces, bad variable naming and so on. You will be allowed 1 pre-submission test, this will highlight all of these problems, giving you the chance to fix them before it counts towards your final grade. Use this submission wisely.
- Testing documents (15%) – Your testing document should include details about how you tested your work. E.g, testing the client with an invalid URL or testing the server with a malformed request. Consider the use of JUnit testing.
- Overall structure (15%) – The overall structure of your work, e.g sensible use of classes with clear responsibilities. This will be manually assessed.

Assessment Mechanism:

You will get **1** pre-test submission and **1** main submission, for each of the parts (client and server). These are separate areas in the TMA system. The pre-submission area will run a subset of the tests which the main (the real) submission area will run, therefore you should make sure to submit to the pre-test area first! Try not to waste this submission, ensure that you are submitting the correct files to the correct area. If the results you receive for the part 1 pre-test are unexpected, ask in labs – this gives us a chance to discuss problems in your work and how to improve it. If you feel TMA has in some way malfunctioned, let us know, we will check the server logs and if necessary, will reset your submission to allow you to resubmit your pre-tests. Accidentally submitting your work to the main test area instead of the pre-test area will fully test your program and start the manual marking

process – be very careful not to make this mistake as you will miss out on any feedback available before your work has been marked. Failing to submit to the main area will mean that your work is not marked.

Your main submission is final. As soon as you submit the main submission, it will be partially automatically marked and partially manually marked. We do not have the resources to manually assess the work multiple times, so this is the final submission. Whilst manually marking, we will also look over any automated tests to ensure the results are fair. If for any reason you have problems with the main submission please let us know ASAP.

We are using this system to allow you to get feedback whilst working, and to give you a chance to learn from the feedback, fix the work, and submit the final version. Although this seems a little complex, the intention is to help the learning process and improve the quality of submissions.