Assignment 2: HTTP Client and HTTP Server using Java Sockets

The goal of this assignment is to learn socket programming and become familiar with the basics of distributed programming. In addition, this assignment will help you to understand the Hypertext Transfer Protocol (HTTP), which is the most widely used Application Layer protocols on the Internet.

Assignment Description

This assignment consists of two parts. In the first part, you will implement a HTTP client to retrieve web pages from a HTTP server. And, in the second part of the assignment, you will build a HTTP server to host web pages.

HTTP Client

In the first part, you will build a HTTP client that communicates with a HTTP server to serve web pages to clients. The HTTP client program should support both HTTP version 1.0 and HTTP version 1.1. Your client should accept the following arguments in the command line as follows:

CommandLine$ java HTTPClient HTTPCommand URI Port HTTPversion

In the above usage specification, the HTTPClient denotes the name of your executable. The HTTPCommand refers to HEAD, GET, PUT or POST. You can use any URI's such as http://www.example.com or http://www.google.com to test your client implementation. And, the default port number of 80 should be used to connect with the HTTPserver. Finally, the last argument should specify the HTTPversion (either 1.0 or 1.1).

After each request, your client should display the response received from the HTTP server on the terminal and also store the response in a local file with an appropriate extension. When you retrieve a webpage from a HTTP server, you should scan the HTML file and check for embedded objects such as images. If you find an embedded object in the HTML file, you should use the GET command to retrieve those objects as well. In order to reduce the complexity of your client, we do not expect you to retrieve all types of embedded objects. During the demonstration, you should at least retrieve image files from a HTTP server. The retrieved images should be stored locally.

For PUT and POST commands, your client should read a string from an interactive command prompt. These two commands will be tested with your HTTP server program as well as a third party server that we provide.

In addition, your client should support the if-modified-since HTTP header for both HTTP version 1.0 and 1.1. The RFC standard of HTTP allows three different date formats. You are allowed to choose any format for your implementation.

HTTP Server

The HTTP server program should host a simple web page on your local machine. This server program should be multi-threaded to support multiple clients at the same time. The server should support the following operations in both HTTP 1.0 and 1.1: HEAD, GET, PUT and POST.

During the demonstration, a HTTP Client will retrieve the web page that you previously retrieved using your HTTP Client program. If your HTTP Client program retrieved the web page and the associated embedded objects correctly, then any 3rd party client such as Firefox or Chrome should be able to successfully render it.

In the case of PUT and POST commands, your server should store the data received from clients in a text file. For the PUT command, the user input should be stored in a new file on the server. While for the POST command, the user input should be appended to an existing file on the server.

Your server should support the if-modified-since HTTP header for both HTTP version 1.0 and 1.1. The RFC standard of HTTP allows three different date formats. You are allowed to choose any format for your implementation.

Your HTTP server should support both versions of HTTP protocol, similar to your HTTP client. More specifically, for HTTP/1.1, your server should use persistent connections. In addition, note that the host header field is mandatory for HTTP version 1.1.

You should at least support the following status codes on your server:

- 200 OK
- 404 Not Found
- 500 Server Error
- 304 Not Modified

Along with the status codes, the server should send the date, content type and content length headers to the client.

General Guidelines

- You are **not allowed to use HTTPURLConnection** package for this assignment. You should only use the sockets package and basic IO libraries to complete the assignment. If in doubt, ask your lab supervisor.
- You should **document your code**. During the evaluation, we will go through your code and may ask you to explain how a certain method works in your code. **Both students must be able to explain all of the code**.
- You should know the difference between HTTP 1.0 and HTTP 1.1. During the demonstration, you should explain how you have handled these differences in your code.
- Telnet can be used to test and understand the HTTP commands. You can use telnet as a debugging tool during your implementation.

Practical Information

- You should either work alone or in groups of two. You should email your group details i.e. names and student numbers to gowrisankar.ramachandran@cs.kuleuven.be on or before 27-02-2015. The subject of this email should be [CN:Assignment2].
- You are strongly encouraged to use the computers in the lab. No support will be provided for debugging your own laptop setup.
- You have 3 weeks to complete the assignment. In the 4th week, you should demonstrate your assignment to the teaching assistants. You will be marked entirely based on your performance in the demonstration.
- For students working in groups, both the students should be prepared to demonstrate the assignment. If you cannot explain your code, we will assume that you did not write it.
- The fourth session is only meant for marking your assignment. Therefore, you should **be ready to demonstrate your code at the start of the 4th practical session**. You will only be marked in your assigned session.

Marking Specifications

The following specifications will be used during the marking session.

HTTP Client Marking (worth 4 out of 9 marks)

Mark	Expected Functionality
0(E)	Client is not functional or sufficiently demonstrated.
1(D)	Client works only with certain web pages or only supports HTTP 1.0.
2(C)	Client works with all the web pages supporting HTTP 1.0 and 1.1, but not storing the web page and images correctly.
3(B)	Client works correctly with HTTP 1.0 and HTTP 1.1.
4(A)	As (B) with elegant and documented code.

HTTP Server Marking (worth 5 out of 9 marks)

Mark	Expected Functionality
0(E)	Server is not functional or sufficiently demonstrated.
1(D)	Server works, but only supports one version of HTTP or handles only one client.
2(C)	Server supports HTTP 1.0 and 1.1, but has threading problems.

3(B)	As above with the correct use of threading and proper support for status codes and headers.
4(A)	As above, server successfully serves the web page retrieved using your HTTP client program.
5(A+)	As above with elegant and documented code.

References:

1. HTTP made really easy [Strongly recommended].
   Link: http://www.jmarshall.com/easy/http/
2. The HTTP Specification: HTTP 1.0 (RFC 1945).
3. The HTTP Specification: HTTP 1.1 (RFC 2616).
4. The definition of embedded objects or MIME types (RFC 1521).
5. RFC Date Format specification (RFC 2616).