CSci 4131: Internet Programming (Spring 2012)
Assignment 1: HTTP Server in Java

Due Date: February 12, 2012

This assignment must be done <u>individually</u>

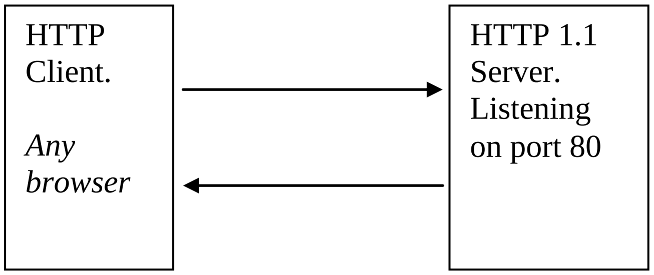
Total Points: 100

Objective:

The objective of this assignment is to learn HTTP protocol (HTTP1.1) and build a tiny HTTP server. In this assignment you would learn how to program, using Java and TCP sockets, the functionalities of an HTTP client and server. You will need to go through RFC 2616 for HTTP 1.1 protocol details.

Introduction:

When a web client (such as Internet Explorer) connects to a web server (such as www.google.com) the interaction between them happens through Hyper Text Transfer Protocol (HTTP).



The working of the basic HTTP server is described below in brief :

1. An HTTP client connects to the HTTP server and makes a HTTP request to request a web resource.
2. The HTTP Sever parses the request header fields. The request can be of type GET, POST or HEAD. For a GET requests, the server identifies the requested resource (for example an HTML file) and checks if the resource exists.
3. The HTTP server then generates an appropriate HTTP response message. If the requested resource is found, the HTTP server includes successful (2xx) status code in the response headers along with other metadata such as the Content Type and Content Length and sends the resource data as the message body. If the requested resource is not found, then the HTTP Server sends response with status code 404.

Problem Statement:

In this assignment you will modify a small HTTP Server which will implement the HTTP protocol. The sample HTTP server program that you will modify is provided on the examples webpage on the class website. Go to Examples->Java-HTTP-Client-Server->HTTPServer.java. Please refer to the Readme file to understand how to run this program.

This HTTPServer program implements a multi-threaded HTTPServer which handles HTTP GET requests from clients. It supports different content types. You will modify this HTTP Server to implement following functionalities

1. Implement conditional GET requests
2. Implement HEAD requests
3. Include response payload messages for various error conditions

4. Implement dynamic response generation by executing a Java program and using chunked transfer encoding

The detail requirements for implementing above functionalities are described below. You will need to refer to HTTP1.1 documentation (RFC 2616) for detail specifications. The specific sections of this document that you need to refer to are mentioned in the description below.

1. **Conditional GET request: (20 points)**

   The given HTTP Server on the Examples webpage handles only basic GET requests. Your HTTP Server will additionally need to handle conditional GET requests also. The client requests would contain additional headers for specifying conditional requests. Specifically you will implement 'If-Modified-Since' (Refer section 14.25) and 'If-Unmodified-Since' (Refer section 14.28) request types. The HTTP server will need to parse these header fields and send the response appropriately.

   Your program should work appropriately under following conditions.
   (5 points for each test case)
   1. If the request type is "If-Modified-Since" then your server should send appropriate response under following conditions
      a. If the requested resource is modified since the specified date, then the requested entity should be sent in the response message.
      b. If the requested resource is not modified since the specified date, then appropriate response with the required status code should be sent.
   2. If the request type is "If-Unmodified-Since" then your server should send appropriate response under following conditions
      a. If the requested resource is not modified since the specified date, then the requested entity should be sent.
      b. If the requested resource is modified since the specified date, then appropriate response with the required status code should be sent.

2. **HEAD request: (10 points)**
   Along with the GET request, your HTTP server will also support HEAD requests. You will include the same metadata (HTTP headers) as included in the response to a GET request. Please refer to section 9.4 of RFC 2616 for details of HEAD requests.
   Specifically, you will include following metadata: Content Length, Content Type, and Modification Time.

3. **Response payload for error conditions: (20 points)**
   The sample HTTP server program given to you only handles resource not found (HTTP 404) error condition, and it only sends the corresponding status code in the response header without any error message in the response body. You will need to handle additional error conditions, as specified below, and also include an appropriate error message (in plain text) in the response body, specific to the error condition.

   Your HTTP server should handle following error conditions: 403, 404, 405, and 406. It should send appropriate error responses as specified below.
   (5 points for each test case)
   a. If the requested resource does not have appropriate permissions (i.e. it is not world-readable), 403 error response should be sent.
   b. If the requested resource is not found, 404 error response should be sent.

c. If the request is a POST request, then server should send 405 – method not allowed response.
d. If the request contains accept headers, and the content characteristics of the requested resource are not acceptable according to the accept headers, for example accept headers specified only html files and the requested entity is an image file, then 406 error response should be sent.

Rediraction Response: **(10 points)**

Additionally you will also send redirection responses for certain URLs. If the request is for a resource named "csci4131", then the client should be redirected to the following location: http://www-users.cselabs.umn.edu/classes/Spring-2012/csci4131/. For example if the request is for URL is http://comp1.cs.umn.edu:5555/csci4131 it should be redirected to the above URL. The browser should be automatically redirected to the new URL. For this, your server should send an appropriate response message with required headers. You would have to include appropriate location headers in the response. The redirection response would include "Permanently moved" status code. Please refer to section 10.3 of RFC 2616 for details.

4. **Dynamic response generation with chunked transfer encoding:** **(40 points)**

To implement this functionality, your HTTP Server would run a Java program which would generate output dynamically to be sent to the client. For this, the client would request a Java class file (`.class`) as the requested resource in the HTTP request. Your HTTP server would check if the requested resource is a `class` file. If the specified resource is a `class` file then the HTTP Server would execute the corresponding Java program as a new process.
Use the Runtime.exec() method to execute a new process. The HTTP Server would read the output generated by the Java program. It would then return the output to the client as the HTTP response. The Java program to be executed would be given to you and hence you will not need to implement a separate Java program for this purpose. Basically, this program would read a JPEG image file and output the raw image data which would be read by your HTTP Server. This program and the image file are available on the examples webpage: Examples->Java-HTTP-Client-Server->ImageGen.java and input.jpeg

The response message would be sent to the client as a series of chunks using the 'chunked transfer encoding' (Refer section 3.6 for details of chunked transfer encoding). Note that the HTTP response headers would be generated by the HTTP Server and not the Java program. Since the program would generate a JPEG image, you would need to include the Content Type header field with value as "image/jpeg" in the response. You would also include appropriate headers for chunked transfer encoding. In this case, you will not include "content-length" header field. You will need to include "transfer-encoding" field as specified in section 14.41.

For this part, your HTTP server should satisfy following requirements.
1. The specified Java program should be executed and its output should be read by the server. (10 points)
2. The response should be sent in series of chunks with chunked transfer encoding (20 points)
3. Correct response should be generated, i.e. the image should be appropriately displayed in the browser. (10 points)

Testing guidelines:
To understand how to run the HTTPServer refer to the Readme file on the examples webpage. You should pick a port number above 5000 to run the HTTP server. You can test the HTTP sever using a browser as follow. Suppose you are running the server on a machine named "silver.cs.umn.edu" and on port number 5555, then your URL would be http://silver.cs.umn.edu:5555/<resource-path>.
If you run the HTTP server on any of the CSE lab machines, then you would not be able to connect to your server from any machine outside the CSE domain due to firewall. Therefore, you should run both the client (or browser) and server on CSE machines only. Alternatively, you can run the HTTP server and client on your home machine.

Useful Java Classes:
Here is a list of Java classes and packages that may be useful for this Assignment.

- Socket
- ServerSocket
- Runtime
- java.net and java.io packages
- Check the following site http://java.sun.com/j2se/1.4.2/docs/api/ for additional information on these classes.
- Lecture Notes #3 are also useful. Specifically, check the following example programs:
    a. HTTPget program
    b. EchoServer program
    c. HTTPServer
- The above programs are also available on the Examples page of the Class web-page.

Things to submit:
Submit a tar file with the name: assignment1.tar

The tar file (assignment1.tar) should contain:
1. HTTPServer.java
2. Any other Java files that you might be using
3. A Readme file including the instructions to be followed to execute your HTTP server.

Please include the following the information in your Readme file:
1. Name
2. Student ID
3. Known issues with your program, if any.

Documentation and Comments:
Please comment your code sufficiently. We reserve the rights to deduct up to 10% of the allocated points if there are NO proper comments in the code or if the Readme file is not provided.