UNIVERSIDADE DA CORUÑA

Computer Science Department

COMPUTER NETWORKS LAB
Practice Work 1: Development of a Web Server

Objectives

This practice work consists in the development of a multi-thread Web server, able to process multiple simultaneous service requests in parallel. The ultimate goal is building a Web server capable of interacting with a customer to navigate through a Web site. The implementation will be divided in two iterations.

The Web server will use the HTTP protocol version 1.0 (defined in RFC 1945). This means independent HTTP requests will be generated for each component of a Web page. The server will be able to process multiple requests in parallel, using multiple threads.

In the main thread, the server is listening on a fixed port. When a TCP connection request is received, the new connection is established through another port and the request is resolved in a separate thread.

We recommend following a two-step approach to implement the practice. In a first stage, the multi-thread server will only display on screen the contents of the received HTTP requests. In the second step, the server should also be able to generate the adequate response to the request. To check that the server works as expected, it is recommended using the ‘nc’ command to connect to the server port, and using the command line to send requests to the server.

The server should be started in a non-reserved port (i.e. port number should be greater than 1024) and the client will indicate the requested file. For instance, the client could request an URL like http://localhost:1111/index.html. The web server will load the file from the local disk and will return it to the client in the right format. If any error happens (e.g. the file cannot be found), the server should answer with the right HTTP code and generate a HTML page for each type of error.

The “iteration 1” (5 points) of this practice work needs to include:

- A multithread web server
- The server will implement the GET and HEAD HTTP methods
- The server will support the following file formats: HTML, plain text, GIF and PNG

Moreover, the “iteration 2” (5 points) consists of:

- “If-Modified-Since” option for the GET method (1 point). This task can be tested by simply reloading the page in the browser. When that is done, the browser adds the “If-Modified-Since” header to the HTTP request. The server should send the page only when there has been changes in the file since the last time the client obtained it. To check that this works as expected, it is recommended to use the tools for developers provided by both Firefox and Chrome. Alternatively, you can also use the “Tamper Data” plugin for Firefox.

- Implement two log files, for accesses and errors (0.5 points). The ‘access’ log will only log the requests that have been answered with a response with status code 2xx or 3xx. Requests answered with a 4xx status code will be stored in the ‘errors’ log.
    - The ‘access’ log file should have the following format for each request:
        - Request Line.
        - Client IP address.
        - Date and time when the request was received: [day/month/year hour:minute:second time_zone]
        - Status code sent in the response to the client.
        - Size (in bytes) of the file sent to the client.
    - The ‘errors’ log file should have the following format for each request:
        - Request line.
        - Client IP address.
        - Date and time when the request was received: [day/month/year hour:minute:second time_zone]
        - Error message.
- Basic configuration of the Web server (1.5 points): the Web server will have a **configuration file** allowing to indicate, at least, the following parameters (please use the java.util.Properties class for the development of this option):
    - PORT parameter: Port number where the server will listen for requests.
    - DIRECTORY_INDEX parameter: Name of the file that will be returned by the server when a folder is requested by the client (e,g, http://localhost:1111/). The program will search for this file only in the folder or subfolder that has been requested (in the root folder of the web server, considering the previous example).
    - DIRECTORY parameter: Root directory of the Web server, where the Web pages are located.
    - ALLOW directive: This directive applies to the requests that try to access a directory, instead of a file. It works as follows:
        - If the default file name (specified in the DIRECTORY_INDEX parameter) exists in the requested directory, the default file will be returned.

        - If the default file name (specified in the DIRECTORY_INDEX parameter) does not exist in the requested directory and the ALLOW directive is enabled, then the server should dynamically create an HTML page displaying the list of all files/subdirectories in it. The name of each file/subdirectory should be shown in a link. Clicking on the link of a file should open the file. Clicking on the link of a subdirectory should show the directory content (files and subdirectories), applying, in this latter case, the ALLOW directive recursively.
        - If the default file name does not exist in the requested directory and the ALLOW directive is disabled, then the server should return a 403 (Access Forbidden) error response.

- Implementation of persistent connections (0.5 points): the Web server will not close the TCP connection immediately after sending the response to the client. In turn, it will remain open for 1 minute to answer other possible requests from the same client. If no new requests are received in one minute, the server will close the connection. With this option, the Web server is implementing the HTTP 1.1 protocol. Note: you do not need to implement the keep-alive header.
- Dynamic pages (1.5 points): the Web server must be able to dynamically generate response pages depending on the parameters received in the HTTP request. We call ‘dynamic page’ to a response whose content is automatically generated by the server using as input the request parameters. We will create a Java class for each one of these pages, and the code in this Java class will be in charge of generating the response. By convention, we use the **.do** extension to identify this kind of requests.
    - To implement this task, we provide the following elements:
        - saludo.html: example HTML form which generates a GET request with different parameters.
        - ServerUtils.java: an utility class which loads the class that will attend the HTTP request.
        - MiniServlet.java: one interface that must be implemented by all the Java classes (commonly called ‘servlets’) that generate dynamic responses to HTTP requests.
        - MiServlet.java: one example class implementing the interface.
    - The objectives are: 1) to understand the example provided, 2) to develop at least one class that implements the interface (MiniServlet.java), and 3) modify the provided HTML form (or create a new one) to generate new requests to a new dynamic page

(which will be served by the implemented class). Moreover, exceptions must be handled properly.

- The server should be able to identify which class should attend each request.
  - For instance: we can assume that the requested resource matches with the name of the class (e.g. to respond to a request to YourServlet.do, we will run the YourServlet class).

The practice should be implemented using directly the sockets functionality in Java, without using any class that implements totally or partially the features of the HTTP Protocol (as, for example, the HttpURLConnection class).

Introduction to HTTP

The HTTP (HyperText Transfer Protocol) protocol is specified in RFC 1945 (version 1.0) and RFC 2616 (version 1.1). This protocol defines how clients (typically browsers) request Web pages to Web servers, and how they perform the transfer of these pages.

The HTTP protocol is based on the TCP protocol (which provides a connection-oriented, reliable service), which ensures that every HTTP message issued by the client or the server is received at the other end without any modifications. In addition, HTTP is a stateless protocol, this is, the HTTP server does not store any information about the clients. Each request received by the server is treated independently of the requests previously received from that or other clients.

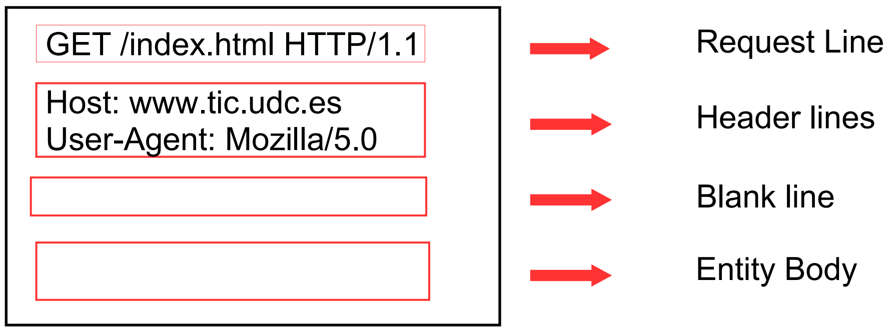
The HTTP 1.0 protocol uses non-persistent connections, meaning that it is necessary to establish a separate TCP connection for each of the components of a Web page. In summary, the procedure for requesting an URL (e.g. http://www.tic.udc.es/index.html) would be as follows

1. The HTTP client initiates the TCP connection with the server www.tic.udc.es on port 80.
2. The HTTP client requests the resource index.html
3. The HTTP server receives the request, searches for the resource, encapsulates it in the HTTP response message and sends it.
4. The server terminates the TCP connection.
5. The HTTP client receives the response and terminates the TCP connection.
6. The client extracts the file of the response message, examines the HTML file and finds references to other resources (e.g. images).
7. Return to step 1, for each of the new resources found in the previous step.

The main drawback of this approach is that it may introduce significant delays, since it is necessary to wait two times the RTT (Round Trip Time), one for the establishment of the connection and the other for the request and response of the resource. In addition, using a connection for each request involves a greater use of resources (buffers, variables, timeouts...), both on the client and on the server.

These problems are solved by means of the HTTP 1.1 protocol. This version of the protocol uses persistent connections, meaning that the server does not immediately close the established TCP connections, in anticipation of new requests. After a period of inactivity, these connections are closed.

The HTTP protocol follows a simple request-response model. The format of a request can be seen in the following figure. The only required fields are the request line and the blank line.



The request-line specifies the type of request, and consists of three fields:

- Method
- URL: requested resource.
- Version of the HTTP protocol used by the browser.

The main methods defined in the HTTP protocol are:

- GET: requests a resource from the server.
- HEAD: requests metainformation about a resource from the server. The HTTP response will be similar to GET responses, but the server will not include the entity body in the response (the entity body is empty).
- POST: allows including data in the entity body of the request.
- PUT: allows clients to upload a file to a certain path (HTTP 1.1 only).
- DELETE: allows clients to remove a file from the server (HTTP 1.1 only).

In the header lines, a typical browser may include multiple options. The most relevant are:

- Host: Specifies the server containing the requested resource.
- User-Agent: Specifies the type of client (e.g. type of browser) that is making the request.
- If-Modified-Since: used jointly with the GET method. The server should not return the requested resource if it has not been modified since the specified date.

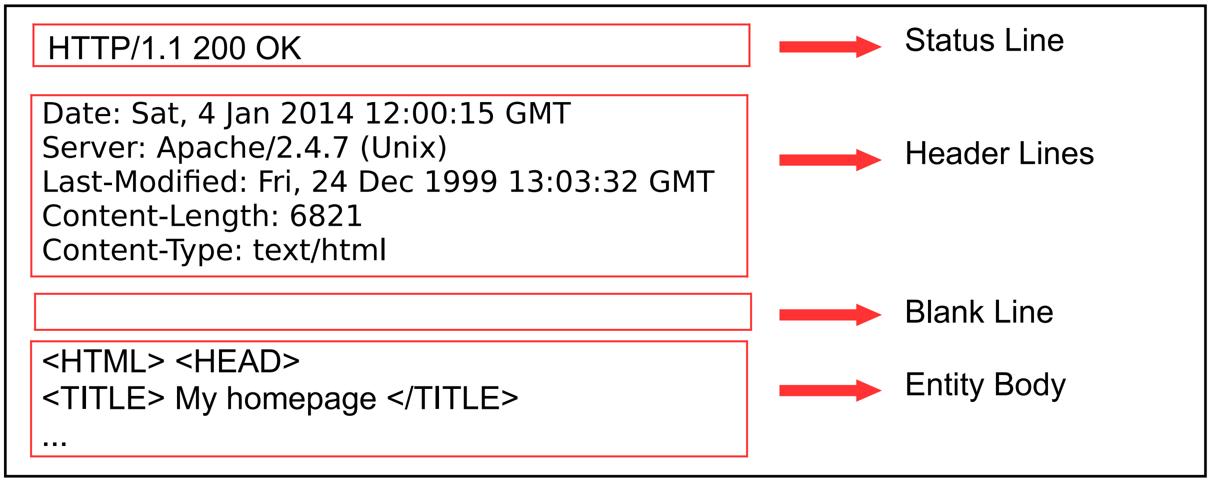
The "If-Modified-Since" option is used by the implementations of the caches on the client. The request format would be as follows:

```
GET /images/udc.gif HTTP/1.1
User-Agent: Mozilla/5.0
If-Modified-Since: Fri, 10 Jan 2014 13:03:32 GMT
```

If the resource has not been modified since the specified date, the server should respond as follows:

```
HTTP/1.1 304 Not Modified
Date: Wed, 5 Feb 2014 20:30:43 GMT
Server: Apache/2.4.7 (Unix)
```

The format of HTTP responses can be seen in the following figure.


HTTP/1.1 200 OK
Date: Sat, 4 Jan 2014 12:00:15 GMT
Server: Apache/2.4.7 (Unix)
Last-Modified: Fri, 24 Dec 1999 13:03:32 GMT
Content-Length: 6821
Content-Type: text/html
<HTML> <HEAD>
<TITLE> My homepage </TITLE>
...
Status Line
Header Lines
Blank Line
Entity Body

The status line includes three fields:

- Version: version used by the Web server.

- Status code: numerical code representing if the response to the request is successful or whether there has been an error.
- Phrase: associated with each numerical code there is a phrase that informs about the nature of the code. All codes and phrases of the protocol can be found in RFC 1945. The most important ones are:
    - 200 OK
    - 400 Bad Request: the request uses an incorrect format, which is not understood by the server.
    - 403 Forbidden: the server understands the request, but refuses to attend it.
    - 404 Not Found: the requested resource does not exist in the server.

The server can also return many different options in the header lines of a response. Some of the most relevant are:

- Date: date and time when the response was created and sent.
- Server: specifies the type of Web server that has attended the request.
- Content-Length: indicates the number of bytes of the requested resource.
- Content-Type: specifies the type of resource contained in the entity body. This field is necessary, because the file extension does not formally specify the associated resource type. The most commonly used types are:
    - text/html: indicates that the response is in HTML format.
    - text/plain: indicates that the answer is in plain text.
    - image/gif: indicates that it is an image in gif format.
    - image/png: indicates that it is an image in png format.
    - application/octet-stream: used when the file format is not identified.
- Last-Modified: indicates the date and time when the resource was modified for the last time. If it has never been modified, it will indicate the creation date and time of the resource.

For the implementation of this practice work, the status line must be present in all generated responses and the first four header lines (Date, Server, Content-Length and Content-Type) will be sent whenever possible.

The Last-Modified header will be needed only if the If-Modified-Since option is implemented.

How to test the "iteration 1" of the practice work?

1. Use the 'nc' command to connect to the server and leave it open (this is to prove that the server is multithread).
2. Open a browser and load a HTML page containing plain text, gif and png images.
3. If the page is displayed => Multithread OK, GET OK, basic formats OK
4. Return to the terminal running the 'nc' command, and send a HEAD request (e.g. HEAD index.html HTTP/1.0).
5. Verify that it returns the correct status line and the header lines => HEAD OK.
6. Verify that the following headers are sent correctly: Date, Server, Content-Type, Content-Length and Last-Modified (only if the If-Modified-Since option is implemented) => HTTP header parameters OK.
7. Start another 'nc' command to connect to the server and send an incorrect request (e.g. GETO index.html HTTP/1.0): check that the server returns a 400 Bad Request error + an error page (with the correct header lines) => Bad Request OK
8. Request a resource that does not exist (e.g. GET indeeex.html HTTP/1.0): check the server returns a 404 Not Found error + an error page (with the correct header lines) => Not Found OK.
9. We provide some files and a Java application for validating the previous points. Therefore, check the previous points again using the httptester.jar file (*java -jar httptester.jar <host> <port> [<0-9>]*). NOTE: the files fic.png, LICENSE.txt and udc.gif must be located in the root folder of the web server.

DELIVERY INSTRUCTIONS

This practice work will be developed from February 18 to March 22, 2019. The evaluation will be based on the files uploaded by the student to her/his SVN repository:

https://svn.fic.udc.es/grao2/red/18-19/.

In this repository, there must be a directory **p1**, containing the Netbeans project with the implemented code.

The delivery of this practice will take place in two phases:

1. In the first phase, the student will develop the "iteration 1" of the practice. The deadline for this phase is March 8, 2019 at 20:00. In the repository, there must exist a review with "iteration 1" as commit message. This part will have a weight of 0.5 points in the final grade. Each student will defend her/his practice work with the teacher in her/his usual practice class during the week starting on March 11, 2019. The file httptester.jar will be used for checking that the Java code works properly.
2. In the second phase, the student will develop the "iteration 2" of the practice. The deadline for this phase is March 22, 2019 at 20:00. The student should create a revision with "iteration 2" as commit message. This part will have a weight of 0.5 points in the final grade. Each student will defend her/his practice work with the teacher in her/his usual practice class during the week starting on March 25, 2019.

The files uploaded to the repository after the deadline for each phase will be not taken into account in the evaluation. Defending the practice work after the assigned practice class can provoke that the practice work will be considered as not presented.

When defending the practice work, the student will also need to show to the teacher that the software works as expected. The student should be also able to explain any component of it.

If the practice work is delivered after the deadline or the 'p1' folder does not exist in the repository, this practice work will be considered as not presented.

The iteration 2 of the practice cannot be delivered or evaluated if the iteration 1 was not delivered in time.

During the defense of the iteration 2, httptester.jar tests will be performed again. The grade for the iteration 1 can be decreased according to the results of the tests, but never increased.