BILKENT UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

CS 421
COMPUTER NETWORKS

PROJECT 1

Osman Batur İnce	21802609	Section 1
Burak Yiğit Uslu	21801745	Section 1

11/11/2021

GET Requests & Responses

HTTP GET requests are in the form:

```
GET {directory} HTTP/1.1
Host: {host-url}
```

It is important to note that the above message also includes a carriage return and a line feed at the end of each line, and a double carriage return and a line feed at the end of the request, which indicates the end of the message. The {host-url} specifies the host/server that is holding the desired file and the {directory} is the particular file within the host-URL we want to access. The HTTP version could change between 1.1 and 1.0, depending on whether the host supports HTTP 1.1 or the client's needs.

A real example of an HTTP GET request is the following:

```
GET ~cs421/fall21/project1/index1.txt HTTP/1.1
Host: www.cs.bilkent.edu.tr
```

An important detail about the above request is that it is in the most basic form. A HTTP GET request may also include additional fields to specify the accepted file formats, accepted languages, or other relevant details. Since these fields were not necessary for the project, we did not use them. The only other tag we used in the HTTP get requests were the range tag, which is used in the following way:

```
GET ~cs421/fall21/project1/index1.txt HTTP/1.1
Host: www.cs.bilkent.edu.tr
Range: bytes=0-500
```

In this example, 0 is the lower limit and 500 is the upper limit for the accepted range. In the response to the above request, only the bytes between 0 and 500 are returned from the desired file to the client.

Once the client makes a HTTP GET request to the host server, the host sends a response.

The response to a HTTP GET request is in the form:

```
HTTP/1.1 {status-code}
Date: {date}
Server: {server}
Last-Modified: {date-last-modified}
ETag: {etag}
Accept-Ranges: {accepted-ranges}
Content-Length: {content-length}
Content-Type: {content-type}

{data}
```

These responses contain the HTTP header, which contains information about the requested file, and the content of the file, denoted by {data} in the above example. In the HTTP header, {status-code} denotes whether the request was successful and the requested file was available. If the {status-code} is 200 OK, this means that the requested file was available, and is sent in the {data} part of the response. If the {status-code} is something other than 200 OK, this means that the requested file could not be sent back to the client. In this case, the format of the rest of the HTTP header may change, and the {data} segment is usually empty. {content-length} contains the length of the desired file, and the {content-type} contains the type of the requested file, such as text/plain. Finally, the {etag} and the {date-last-modified} are used for checking whether the file was modified since it was last requested. This is done for caching purposes.

Relevant Status Codes
200 OK: This status response code means that the request was successful, and the desired data is returned. In general, status codes in the form 2XX mean the request was successful.
206 Partial Content: This status response code is an indicator that the request was successful, and only the desired range of data is returned.
301/302 Permanent Redirect/Temporary Redirect: This status response code means that the request was redirected to somewhere else. In general, status response codes in the form 3XX mean redirection.
404 Not Found: This means that the requested page was not found. In general, status codes in the form 4XX mean that there is a client error regarding the request.

500 Internal Server Error: This means that the request made by the client could not be handled because of a problem with the server. In general, response codes in the form 5XX mean there was some sort of server error.

Sequence Of Events In An HTTP Connection

The sequence of events in an HTTP connection is as follows:

The client, or in this case the FileDownloader, initiates a TCP connection to the HTTP server, at the port number 80, which is reserved for HTTP connections. Then, the server accepts the connection and notifies the client. Following this, the client sends the HTTP request message into the TCP socket, which redirects this message to the server based on the URL in the request. After receiving this request the server sends back a response, either with the status code in the form 2XX indicating that the request was successful, or a response with a different status code, indicating the reason for failure. In non-persistent HTTP connections, the server socket is closed after this. In persistent connections, the server socket may remain open for more requests, without needing to establish another TCP connection again. Within the context of this project, however, this does not make a difference as we only request 1 txt file from each URL in the list in the index file. In the project, after receiving the requested file, the connection is closed, and the FileDownloader moves on to the next URL in the list.

Notes Regarding Our Implementation

In our implementation, we receive the response data inside a while loop and write it into a buffer, until the entire data is obtained. We do this because TCP is a stream oriented protocol and the entire response might not have arrived on our first call of the socket.recv() function in Python.

References

J.F. Kurose and K.W. Ross, Computer Networking, 5th ed., Addison Wesley, 2010.