Http Server

[Ugeopgave 2] *

Mirza Hasanbasic

ABSTRACT

In this assignment we will be looking at how to implement a simple HTTP server, that will have a subset of the entire HTTP protocol. The HTTP server will still be able to communicate with client and can get multiple connections because of thread implementation. There will be performance and validation discussion about the HTTP server as well as the limitations and testing of the server.

1. THE SERVER

The pseudo coding part was made in collaboration with Mads and Mathias

The server is written in Python and to run it you have to type `./Server.py` and let the console window be. Now open a new console window and try to connect to the server. The default port is 5000 if this port doesn't work on your computer you may have to change it in the code on line 61.

You can use my `HTTPClient` to test the server or use telnet or use your own implantation.

The libraries and frameworks I use are threading, socketServer, os.path, unittest and email.utils when we use the function formatdata. I also have imported serverResponse, parseRequest and returnFile into the `Server.py` file.

The way the server work, is that it inherit from the ThreadingMixInto create a threaded connection behavior. This class defines the daemon_threads attribute that tells the server whether it should wait for a thread termination or not. The value is False per default, which means that Python will not exit until the threads that are created by the class have exited. We have set the value to True so for every request a thread is spawned and when the request ends the thread is killed.

It is important to note that the mix in class first as it overrides a method that is defined it TCPServer.

So this means that the server is supporting multiple connections from clients that request some files or something else from the server.

*

DIKU '16 El Diku, Copenhagen

DOI: 10.475/133_7

To be able to follow the HTTP standard then the server has a help function to parse the request. Here I have my `parseReqToDict`. This function will check if the request line is as follows: `Method SP Request-URI SP HTTP-Version CRLF`. So first it checks for the length of the request line and if it is different than 3 then an error is returned. If the request line is right, then if the requested method is supported if not then it throws a 400 error after that it checks for the separators. The way we split the data follows the `JSON` standard for an pleasant readability. Furthermore if the request is `HTTP/1.1` then it is required to give a `HOST` if there is non then error 400 is return

For now the only method that is supported is `GET` and the headers that are supported are `host` and `connection` These headers are the simplest and the most necessary headers to support.

The reason I support the Host request-header is that it specifies the host and port number of the resource that is requested. It is also required that a client must include a Host header field in all HTTP/1.1 request messages and all internet based HTTP/1.1 servers must respond with a 400 (Bad request) status code to any HTTP/1.1 request messages which lacks the Host header field.

The other header i.e. the connection allows the sender to specify options that are desired for that particular connection. Also HTTP/1.1 must parse the connection header field before a message is forwarded and connection options are signaled by the presence of a connection token in the connection header field. The HTTP/1.1 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. It is also notable that HTTP/1.1 apllication that do not support persistent connection must include the "close" connection option in every message. The connection also protects from a mistaken forwarding.

For the response part of the server, we need to have a correct response. For this I use the `serverResponse`. I have a response template that tells how a response will be as output. First I check if the given method a client request is supported by the server if not then I return 405 as in method not allowed. Hereafter I check for the filecontent i.e. the `abs_path` if you only look for the localhost then it will return a list of the files in the client folder. When the filecontent has passed the `200 OK` is return back to the client. If everything fails then the client will get a 400 error as bad request.

1.1 Code structure

I use 4 files. The 4 files are `Server.py`, `serverResponse.py`,

`parseReqToDict.py` and `returnFile.py`. The only that handles the server i.e. creates it and the needed threads is the `Server.py` file and this file uses the other 3 as helper files to be sure that the parsed request is correct, that crafted response follows the HTTP RFC.

1.2 Performance and validation

The performance of the server is fine as the validation of the hash checksum seems to match the zip files I tested it on. Since the sums are MD5 I used `md5sum filename` and the md5sums can be seen in the appendix as well as the place where I downloaded the files from.

1.3 Limitations

The only limitation I could find and test was that the server timeouts when it gets a request from two or more clients that request a file larger than 512MB. The error I got was from the socketServer framework and no matter if I made the BUFFER_SIZE bigger it wouldn't work with a bigger file than 512MB for two or more clients that made a request to the server

The only method that the server can handle is the GET method and it only supports two headers i.e. host and connection

1.4 Testing

The way I tested the server was that in one console I initiated the server. Then I opened two more consoles and tried to download a file from the server simultaneously. This seemed to work fine, both my HTTPClients recieved a Status code 200 OK: Connection to 127.0.0.1 successful and that the response was written to the filename I have given. I chose to download a 512MB file as explained in the subsection Performance and validation. Hereafter I checked the checksum (md5sum) for both the files and they matched. This was done in the console window.

I have some unittesting in the file `parseReqToDict` to test whether it parses the request correctly and is it seems that unittest returns that everything is as expected.

APPENDIX

A. UNITTESTING

```
exampleGetRequestBAD = (
    "SET / HTTP/1.1 {0}"
    "Host: dr.dk{0}"
    "Connection: close{0}{0}"
    "poop".format(CRLF))


class TestStringMethod(unittest.TestCase):
    def testParseRequest(self):
        self.assertEqual(parseRequestIntoDict(exampleGetRequestBAD),
                         {'Body': 'poop',
                          'Headers': {
                              'Host': ' dr.dk',
                              'Connection': ' close'
                          },
                          'abs_path': '/',
                          'Protocol': 'HTTP/1.1',
                          'Method': 'SET'}
                         )
        self.assertEqual(parseRequestIntoDict(exampleGetRequestBAD),
                         {'Body': '',
                          'Headers': {
```

odtestParseRequest

1 test failed - 0ms

toDict TestStringMeth 0ms
quest 0ms

```
/usr/bin/python2.7 /home/mirza/pycharm-2016.1.3/helpers/pycharm/utrunner.py /home/mirza/github/datanet/Server/parseRequestIntoDict.py::TestStringMethod::testParseRequest true
Testing started at 11:45 PM ...
{'Body': 'poop', 'Headers': {'Host': ' dr.dk', 'Connection': ' close'}, 'abs_path': '/', 'Protocol': 'HTTP/1.1', 'Method': 'SET'}
{'Body': 'poop', 'Headers': {'Host': ' dr.dk', 'Connection': ' close'}, 'abs_path': '/', 'Protocol': 'HTTP/1.1', 'Method': 'SET'}

Failure
Traceback (most recent call last):
  File "/home/mirza/github/datanet/Server/parseRequestIntoDict.py", line 91, in testParseRequest
    'Method': 'SET'}
AssertionError: {'Body': 'poop', 'Headers': {'Host': ' dr.dk', 'Connection': ' close'}, 'abs_pat [truncated]... != {'Body': '', 'Headers': {'Host': ' dr.dk', 'Connection': ' clo
- {'Body': 'poop',
?           ----

+ {'Body': '',
     'Headers': {'Connection': ' close', 'Host': ' dr.dk'},
     'Method': 'SET',
     'Protocol': 'HTTP/1.1',
     'abs_path': '/'}


Process finished with exit code 0
```

B. SERVER WITH MULTIPLE CONNECTIONS

C. MD5 SUMS FOR THE FILES

http://www.thinkbroadband.com/download/
5b563100babfef2f2ec9ab2d55e97fd1 100MB.zip
3aa55f03c298b83cd7708e90d289afbd 10MB.zip
286e80b3b7420263038ab06d76774043 1GB.zip
f00be31bb73fe783209497d80aa1de89 1MB.zip
3389a0b30e05ef6613ccbdae5d9ec0bd 200MB.zip
9017804333c820e3b4249130fc989e00 20MB.zip
528972766cd55c26a570829775afd2a8 2MB.zip
2699c63cb6699b2272f78989b09e88b1 50MB.zip
dfe6504e0e8283357a3443234b266246 512MB.zip
b3215c06647bc550406a9c8ccc378756 5MB.zip