Assignment 1, datanet 2015

Sebastian Ostenfeldt Jensen
University of Copenhagen

1. ABSTRACT

We are to develop a http client command line tool that can download a file from a given url. This is done by low level socket programing using the python language and following the http protocol version 1.1 with the requirement that the implemented protocol are standard compliant.

2. INTRODUCTION

A socket is a door between the application level and the network. When an application that is acting as a client writes to a socket the socket then take care of putting the data on the network. When an application is a server the socket take care of listen to incoming data and present it to the application. A on top of the socket different communication protocols can be implemented. This could be ftp p2p or other internet protocols. In the case we will be looking at we are only concerned with the http protocol. The http protocol is defined in HTTP RFC (RFC 2616). In the HTTP RFC is described what is required for an implementation to be standard compliant. For this implementation we don't need to focus on all the requirements for the protocol but only the ones that concern a get request. E.g. set the http version at the first line and the header need to include the host line and the "Connection: Close"as we do not use a persistent connection.

3. DESIGN

The application is a commandline tool that takes 2 arguments. First argument is the URL and second argument is the file where the received data is written to. If only the URL is given the output is written to stdout. When the program starts it write the host and path obtained from the url to stdout. Then it writes the http headers to stdout. E.g. given the following url

```
http://www.ku.dk/diku
```

prints

```
Host: www.ku.dk
path: diku
```

```
<received header lines>
```

As we are only concerned with http the part of the url describing it is http can be omitted. So

```
www.diku.dk
```

is also a valid url.

4. IMPLEMENTATION

The application is implemented using python's socket module and sys module.

First a check on number of arguments is done and a var is set to indicate if output should be written to stdout or a file

Then three if statements splits the url in to the different parts we need for the http request. To illustrate the url has been split the host and path are written to stdout.

A socket object is created with the function socket from the socket module.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

First argument to socket, $socket.AF_INET$ indicate to use IPv4. Second argument $socket.SOCK_STREAM$ indicate to use the tcp protocol.

Then the socket is connected to the host on port 80 with the connect method.

```
s.connect((host, 80))
```

The get request and the headers are send

```
s.send( "GET /" + path + " HTTP/1.1\r\n")
s.send( "User-Agent: wget (linux-gnu)\r\n")
s.send( "Accept: */*\r\n")
s.send( "Host: " + host + "\r\n")
s.send( "Connection: Close\r\n")
s.send( "\r\n")
```

The data is then received in chunks of 4096b and stored in the variable data2 and the socket is closed

```
data = s.recv(4096)
data2 = data
while data != '':
        data = s.recv(4096)
        data2 = data2 + data
s.close()
```

The received data is split into the headers and the data and the headers are written to stdout.

```
received = data2.split("\r\n\r\n", 1)
print "\n\nReceived http headers:\n"
print received[0]
```

The data is then written to file if a filename was given as argument else it is written to stdout.

```
if not writeToFile:
        print received[1]
else:
        out = open(of, "w")
        out.write(received[1])
        out.close()
```

The full code can be found in the file httpClient.py

4.1 limitations

The application only implements a small part of the html protocol. The application is only supposed to use the get method therefore the post request method is not considered. Some other limitation is listed here.

No support for an alternative port. An alternative port could be used to connect to a http server, but this application does not impement it. To implement it the port should be extracted from the url. E.g if ku.dk where listening on port 8080 the url should be

```
ku.dk:8008
```

The port should then be extracted from the url and be given as the argument when the socket is connected.

Response messages are not considered. E.g. When receiving a response 303 redirect message the application should redirect to the new location. To make the application aware of redirects a response message with code 303 the response should result in a new request to where the resource has moved to. The new address is parsed in the response header line “Location”

5. RESULT

The test of the application has been done on the following setup

```
Computter:
Thinkpad t520 i7 quardcore, 4 Gb ram

Internet connection:
200Mbit fiber shared with 120 other appartments.
computer connected wireless using IEEE 802.11n
```

To test the application a shell script which has been written which Download 2 files from the debian mirror. a 20Mb file and a 70kb file. The speed is meassured and compared with the speed of wget. Then the md5sum of the files are compared with the md5sum given by the debian mirror. See fig 1. for the output of the test.

When downloading the 20 Mb file sometime wget is faster and sometime the python implementation is faster. The reason it is different is because of the exact load on the network when the file is downloaded.

With the small file wget is in nearly all cases faster. The reason for this difference is not because of the network part but the way the program is implemented. E.g. the python application prints more stuff to the screen and some part could be implemented in a more effective way.

```
Test: Downloading a debian file of 20Mb and 70kb with python socket and with with wget
Then the time is compared and the md5sum of the python downloaded file is compared with the
md5sum provided from the debian page

sebastian's http client
host: ftp.dk.debian.org
path: debian/dists/oldstable/Contents-amd64.gz


Received http headers:

HTTP/1.1 200 OK
Date: Tue, 28 Apr 2015 17:20:35 GMT
Server: Apache/2.2.14 (Ubuntu)
Last-Modified: Sat, 10 Jan 2015 10:46:24 GMT
ETag: "33f8002-1485535-50c49fc06c000"
Accept-Ranges: bytes
Content-Length: 21517621
Connection: close
Content-Type: application/x-gzip


starting download with wget
sebastian's http client
host: ftp.dk.debian.org
path: debian/dists/oldstable/contrib/Contents-amd64.gz


Received http headers:

HTTP/1.1 200 OK
Date: Tue, 28 Apr 2015 17:22:23 GMT
Server: Apache/2.2.14 (Ubuntu)
Last-Modified: Sat, 10 Jan 2015 10:41:51 GMT
ETag: "33ee003-11ecc-50c49ebc119c0"
Accept-Ranges: bytes
Content-Length: 73420
Connection: close
Content-Type: application/x-gzip


starting download with wget

20Mb file resaults
time python    time wget
60930ms            47121ms
md5sum of the file downloade with python and md5sum of the file stated by the debian site
98386b9a165648ebeb96e21cb1c64600
98386b9a165648ebeb96e21cb1c64600

70kb file resaults
time python    time wget
1478ms             872ms
md5sum of the file downloade with python and md5sum of the file stated by the debian site
ffa8c115370b047d1d238985f0e2505a
ffa8c115370b047d1d238985f0e2505a
```

Fig. 1. Output of the shell script testing.sh which can be found in the suplied source.