Introduction to Web Science

Assignment 4

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: November 23, 2016, 10:00 a.m.
Tutorial on: November 25, 2016, 12:00 p.m.

In this assignment we cover two topics: 1) **HTTP** & 2) **Web Content**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: Abdullah Elkindy, Stela Nebiaj, Fiorela Ciroku

1 Implementing a simplified HTTP GET Request (15 Points)

The goal of this exercise is to review the hyptertext transfer protocol and gain a better understanding of how it works.

Your task is to use the python programming language to create an HTTP client (http-client.py) that takes a URL as a command line argument and is able to download an arbitrary file from the World Wide Web and store it on your hard drive (in the same directory as your python code is running). The program should also print out the complete HTTP header of the response and store the header in a seperated file.

Your programm should only use the socket library so that you can open a TCP socket and and sys library to do command line parsing. You can either use urlparse lib or your code from assignment 3 in order to process the url which should be retrieved.

Your programm should be able to sucessfully download at least the following files:

1. http://west.uni-koblenz.de/en/studying/courses/ws1617/introduction-to-web-science
2. http://west.uni-koblenz.de/sites/default/files/styles/personen_bild/public/_IMG0076-Bearbeitet_03.jpg

Use of libraries like `httplib`, `urllib`, etc are not allowed in this task.

1.1 Hints:

There will be quite some challenges in order to finish the task

- Your program only has to be able to process HTTP-responses with status 200 OK.
- Make sure you receive the full response from your TCP socket. (create a function handling this task)
- Separated the HTTP header from the body (again create a function to do this)
- If a binary file is requested make sure it is not stored in a corrupted way

1.2 Example

```
1: python httpclient.py http://west.uni-koblenz.de/index.php
2:
3: HTTP/1.1 200 OK
4: Date: Wed, 16 Nov 2016 13:19:19 GMT
5: Server: Apache/2.4.7 (Ubuntu)
6: X-Powered-By: PHP/5.5.9-1ubuntu4.20
7: X-Drupal-Cache: HIT
8: Etag: "1479302344-0"
9: Content-Language: de
```

```
10: X-Frame-Options: SAMEORIGIN
11: X-UA-Compatible: IE=edge,chrome=1
12: X-Generator: Drupal 7 (http://drupal.org)
13: Link: <http://west.uni-koblenz.de/de>; rel="canonical",<http://west.uni-koblenz.de
14: Cache-Control: public, max-age=0
15: Last-Modified: Wed, 16 Nov 2016 13:19:04 GMT
16: Expires: Sun, 19 Nov 1978 05:00:00 GMT
17: Vary: Cookie,Accept-Encoding
18: Connection: close
19: Content-Type: text/html; charset=utf-8
```

The header will be printed and stored in index.php.header. The retrieved html document will be stored in index.php

Answer: httpclient.py

```
1: #Stela Nebiaj
2: #Fiorela Ciroku
3: #Abdullah Elkindy
4:
5:
6:
7: import socket
8: import sys
9: import os
10: from urllib.parse import urlparse
11: import requests
12: import struct
13: import time
14: import errno
15:
16: # a function to create a directory
17: def create_dir(path):
18:     try:
19:         os.makedirs(path)
20:     except OSError as exception:
21:         if exception.errno != errno.EEXIST:
22:             raise
23: #main function
24: def http_req(url):
25: #get file name
26:     url_array=url.split("/")
27:     file_name=url_array[len(url_array)-1]
28:
29:     is_image=False;
30:
31:     if "." not in file_name:
32:         file_name = file_name+".php"
33:     else:
34:         is_image = True
```

```
35:
36:
37: #checking the response if 200 we continue
38:     if requests.get(url).status_code !=200:
39:         print("ERROR: HTTP status code ",requests.get(url).status_code," is not a
40:         sys.exit()
41: #creating the download path in the local machine
42:     download_path="data/"
43:     if is_image:
44:         download_path=download_path+"images/"
45:
46:     create_dir(download_path)
47:
48:     #creating the files on the local machine
49:     o = urlparse(url)
50:     file_path_header=download_path+file_name+".header"
51:     file_header=open(file_path_header,"w")
52:     file_path_body=download_path+file_name
53:     if is_image:
54:         file_body=open(file_path_body,"wb")
55:     else:
56:         file_body=open(file_path_body,"w")
57:     #defining the socket
58:     try:
59:         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
60:     except socket.error:
61:         print ('Failed to create socket')
62:         sys.exit()
63:     print('socket created')
64:
65:     host = o.netloc
66:     port = 80
67:
68:
69:     sock.connect((host, port))
70:
71:     message="GET "+o.path+" HTTP/1.0\r\n\r\n"
72:
73:     #we send the GET message req encoded. It has to be bytes on the socket.
74:     try :
75:         sock.send(message.encode('utf-8'))
76:     except socket.error:
77:         print("send failed")
78:         sys.exit()
79:
80:
81:     print ("Message send successfully")
82:
83:     #recv_timeout is a function where the data are received. We receive all as un
```

```
84:     #and body of php files. As for images we keep them unencoded to be able to sa
85:
86:     def recv_timeout(the_socket,timeout=2):
87:         #make socket non blocking
88:         the_socket.setblocking(0)
89:
90:
91:         total_data=[];
92:         data='';
93:
94:
95:         begin=time.time()
96:         while 1:
97:             #if you got some data, then break after timeout
98:             if total_data and time.time()-begin > timeout:
99:                 break
100:
101:            #if you got no data at all, wait a little longer
102:            elif time.time()-begin > timeout*2:
103:                break
104:
105:
106:            try:
107:                data = the_socket.recv(8192)
108:
109:                if data:
110:                    total_data.append(data)
111:                    #change the beginning time for measurement
112:                    begin=time.time()
113:                else:
114:                    #sleep for sometime to indicate a gap
115:                    time.sleep(0.1)
116:            except:
117:                pass
118:
119:
120:        return b''.join(total_data)
121:
122:
123:
124:    header,body = recv_timeout(sock).split(b'\r\n\r\n',1)
125:
126:
127:    #here we decide to decode the body of php files and leave the body of images
128:    print(header.decode('utf-8'))
129:    file_header.write(header.decode('utf-8'))
130:    if is_image:
131:        file_body.write(body)
132:    else:
```

```
133:         file_body.write(body.decode('utf-8'))
134:     sock.close()
135: #for terminal invocation
136: if len(sys.argv) == 2:
137:     http_req(sys.argv[1])
138: else:
139:     print('Invalid invocation of the function: python httpclient.py url')
```

2 Download Everything (15 Points)

If you have successfully managed to solve the previous exercise you are able to download a web page from any url. Unfortionately in order to successfully render that very webpage the browser might need to download all the included images

In this exercise you should create a python file (downloadEverything.py) which takes two arguments. The first argument should be a name of a locally stored html file. The second argument is the url from which this file was downloaded.

Your program should

1. be able to find a list of urls the images that need to be downloaded for successful rendering the html file.
2. print the list of URLs to the console.
3. call the program from task 1 (or if you couldn't complete task 1 you can call wget or use any python lib to fulfill the http request) to download all the necessary images and store them on your hard drive.

To finnish the task you are allowed to use the 're' library for regualar expressions and everything that you have been allowed to use in task 1.

2.1 Hints

1. If you couldn't finnish the last task you can simulate the relevant behavior by using the program wget which is available in almost any UNIX shell.
2. Some files mentioned in the html file might use relative or absolut paths and not fully qualified urls. Those should be fixed to the correct full urls.
3. In case you run problems with constructing urls from relative or absult file paths you can always check with your web browser how the url is dereferenced.

Answer: downloadEverything.py

```
1: #Stela Nebiaj
2: #Fiorela Ciroku
3: #Abdullah Elkindy
4:
5: import sys
6: import re
7: from httpclient import http_req
8: def downloadImages(file_name,url):
9:         #We download the php file using http_req from task 1
10:        http_req(url)
11:        #read the php file into the variable data
12:        with open('data/'+file_name, 'r') as myfile:
```

```
13:             data=myfile.read().replace('\n', ' ')
14:         #use regex to get the source of each image tag and save it in images array
15:         images = re.findall(r'<img[^>]*\ssrc="(.*?)"', data)
16:         i=0
17:         #We loop on each img to clean its url and then use http_req from task 1 to
18:         while i<len(images):
19:                 if "?" in images[i]:
20:                         images[i]=images[i].split("?")[0]
21:
22:                 if "http" not in images[i]:
23:                 #Im adding the url to these tags because they're relative to the
24:                 #i discarded them from http_req use
25:                         images[i]=url+images[i]
26:                 else:
27:                         http_req(images[i])
28:                 i=i+1
29:         #last we print all urls.
30:         j=0
31:         while j<len(images):
32:                 print(images[j])
33:                 j=j+1
34:
35: if len(sys.argv) == 3:
36:     downloadImages(sys.argv[1],sys.argv[2])
37: else:
38:     print('Invalid invocation of the function: python downloadEverything.py file_
```

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment4/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.