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.

Delta Group Members

Oana Dumitrasc
odumitrasc@uni-koblenz.de

Alisa Becker
alisabecker@uni-koblenz.de

Omar Aly
oaly@uni-koblenz.de

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 finnish 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)
- Sperated 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

```
python httpclient.py http://west.uni-koblenz.de/index.php

HTTP/1.1 200 OK
Date: Wed, 16 Nov 2016 13:19:19 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.20
X-Drupal-Cache: HIT
Etag: "1479302344-0"
Content-Language: de
```

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

HTTP Client - GET Request

```
import socket
from urllib.parse import urlparse
import time


# function to separate header from body & print header to console
# also save body in the appropriate file format according to the
# content type value of the header
def getHeaderAndBody(receivedData, fPath):
    # find the starting part of body
    startOfBody = receivedData.find(b"\r\n\r\n")
    # creating the header part
    header = receivedData[:startOfBody]
    # creating the body part
    # the constant 4 is just to escape the \r\n\r\n characters
    body = receivedData[(startOfBody + 4):]

    # if the HTTP status is 200 OK we process header & body
    # else we just print out the header
    if (str(header)).find("200 OK") >= 0:
        # creating the header dictionary
        # to know the content type
        headerDictionary = createHeaderDict(header.decode("utf-8"))
        # extracting the content type of the body
        contentType = headerDictionary["Content-Type"]
        # getting the image name by splitting on the url path
        splittedPath = fPath.split("/")
        fileName = splittedPath[-1]
        # according to the content type
        # save the body in appropriate format
        if contentType.find("image") >= 0:
            # writing to the body file
            # which will be created if it doesn't exist
            # using 'wb' to write bytes to the file and not strings
            bodyFile = open(fileName, "wb")
            bodyFile.write(body)
            # writing to the header file
            # which will be created if it doesn't exist
            headerFile = open("image_header.txt", 'w')
            headerFile.write(header.decode("utf-8"))
        elif contentType.find("text/html") >= 0:
            # if the file name doesn't have extension we add it
            # using the content type value
            content = (((contentType.split(";"))[0]).split("/"))[1]
            fileExtension = "" if fileName.find(".") >= 0 else content
            # writing to the body file
            # which will be created if it doesn't exist
            # using 'w' to write string to the file
            bodyFile = open(fileName + "." + fileExtension, 'w')
            bodyFile.write(body.decode("utf-8"))
            # writing to the header file
            # which will be created if it doesn't exist
            headerFile = open("html_header.txt", 'w')
            headerFile.write(header.decode("utf-8"))
    # showing the header file content on the console
    print(header.decode("utf-8"))


# method to create a dictionary for header content from header string
def createHeaderDict(header):
    headerDict = dict()
    headerLines = header.split("\r\n")
    for i in range(1, len(headerLines)):
        keyValue = headerLines[i].split(": ")
        headerDict[keyValue[0]] = keyValue[1]

    return headerDict


# creating function to keep socket open till we receive all data
# or connection timeout(default 3 seconds)
def recvFullResponse(socket, timeout=3.0):
    # using non blocking sockets
    socket.setblocking(0)

    # variables to hold our partial and final completed data
    # received from the server
    allData = list()
    partialData = ''

    # begin counting time for timeout
    startTime = time.time()

    # variable to determine how much data we want to receive
    bytesSize = 1024

    while True:
        # we don't need to wait if we didn't receive any further data
        # than what we already received
        if allData and time.time() - startTime > timeout:
            break
        # wait double timeout if we didn't get any data at all
        # then break with a readable message to the user
        elif time.time() - startTime > timeout * 2:
            return "Time out and No Data Recieved"
        else:
            # try catch block
            # trying to receive data if nothing received we just do
            # another loop till the server responded with some data
            try:
                # receiving data
                partialData = socket.recv(bytesSize)
                # check if the get request result status is 200 OK
                # the bytesSize variable is used to not recheck
                # for each and every next part of the data
                if bytesSize == 1024 and (str(partialData)).find("200 OK") < 0:
                    return ''.join(partialData)
                else:
                    bytesSize = 4096
                    if partialData:
                        # append new data to already received data
                        allData.append(partialData)
                        # for sure reset start time if we get data
                        # to be sure we didn't get timeout before
                        # data receiving completion
                        startTime = time.time()
                    # wait for a while before doing another loop
                    else:
                        time.sleep(0.1)

            except:
                pass

    # use join with empty byte to return data
    # in a one byte sequence, for later extracting header purposes
    return b''.join(allData)


try:
    # create TCP socket
    sckt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
except:
    print("Socket Creation Failed")

# accepting user inputs from command line
userInput = input("URL: ")
# parsing url using urlparse library
parsedURL = urlparse(userInput)
filePath = parsedURL.path
# creating server address using the server name from
# parseURL.netloc parameter and setting the port to
serverAdd = (parsedURL.netloc, 80)
# creating get request using the path extracted from the given URL
getRequest = "GET " + filePath + " HTTP/1.0\r\nHost: " + parsedURL.netloc + "\r\n

try:
    # connecting to the server
    sckt.connect(serverAdd)
except socket.error:
    print("Connection to the server failed")

try:
    # sending request to the server
    sckt.sendall(bytes(getRequest, "UTF-8"))
    # storing received data in a variable
    receivedData = recvFullResponse(sckt)
    # doing the final work
    getHeaderAndBody(receivedData, filePath)
except:
    print("Sending Message Failed")

# closing socket
sckt.close()
```

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.

Download Everything

```
import socket
from urllib.parse import urlparse
import re

# importing another python file
# this syntax it is used in order to avoid
# the problems caused by the hyphen in the name
taskOne = __import__("http-client")

def getUrls(fileName):
    # opening the file from which we want to get
    # the urls for the images
    file = open(fileName, 'r')
    fileContent = file.read()
    # find all the image tags in order to get the correct urls
    urls = re.findall('<img [^>]*src=\"([^\"]+)', fileContent)
    prsdURL = urlparse(inputUrl)
    # looping over the urls
    # if the first character is /
    # then this is a relative path
    # and we have to make it absolute
    for url in urls:
        if url[0] == '/':
            urls[urls.index(url)] = prsdURL.scheme + "://" + prsdURL.netloc + url
    return urls


# accepting user inputs from command line
inputUrl = input("URL: ")
localFileName = input("File Name: ")
imagesUrls = getUrls(localFileName)
# printing out the urls
print(imagesUrls)
for imgUrl in imagesUrls:

    try:
        # create TCP socket
        sckt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except:
        print("Socket Creation Failed")

    # parsing url using urlparse library
    parseURL = urlparse(imgUrl)
    flPath = parseURL.path
    # creating server address using the server name from
    # parseURL.netloc parameter and setting the port to
    serverAdd = (parseURL.netloc, 80)
    # creating get request using the path extracted from the given URL
    getRequest = "GET " + flPath + " HTTP/1.0\r\nHost: " + parseURL.netloc + "\r\

    try:
        # connecting to the server
        sckt.connect(serverAdd)
    except socket.error:
        print("Connection to the server failed")

    try:
        # sending request to the server
        sckt.sendall(bytes(getRequest, "UTF-8"))
        # using the methods from the first task in order to download the images
        taskOne.getHeaderAndBody(taskOne.recvFullResponse(sckt), flPath)
    except:
        print("Sending Message Failed")

    sckt.close()
```

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`.