Network Programming for Engineers (ECE 5650)

Project 2

Team Members Names: Mostafa AlNaimi, Aya AbouZaid

Date: 3/18/2020
Due Date: 3/6/2020

Source Code(s):

```
'''
Name: Aya AbouZaid and Mostafa AlNaimi
Project 2
ECE 5650
Date : 03/17/2020
Due Date : 06/06/2020
Does your program meet all requirements? If not, explain the problem.
Yes

Does the program run correctly all the time? If not, explain the problem.
Yes

Did you adequately test the program? If not, specify.
Yes

Is the program well documented?
Yes

'''
import os
import _thread #Include Low-level threading API modules
import threading #Higher level threading modules
import time #Include time modules
from os import path #Include OS Library
from socket import * #Include Python's Socket library
from html.parser import HTMLParser #Include Python's HTML Parser library
```

```
'''
 - Global Variable section
'''
reply = "" #identify variable reply
data = ""; #identify variable data
web_path = ""; #identify variable web path
web_link = "";   #identify variable web link
global_arr=["","","",""]; #global array to store web server name and other important information.
thread_counter = 0; #Global counter to keep track of the number of threads
Parsercounter_gb = [0,1,0,0,0,0,0,0,0,0]; #Global array to keep track of parser feed.

'''
- Defining Header Data
'''
header = "Accept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: Keep-alive\r\n\r\n"
serverPort = 80 # connect to HTTP

"""
Class that defines all the threads function calls will be used in the application
"""
class webThread (threading.Thread): #Thread that handles websites requests
  def __init__(self, threadID, name, counter):
    threading.Thread.__init__(self) #Initialize the thread.
    self.threadID = threadID #Save Thread ID.
    self.name = name #Specify Thread name if exists
    self.counter = counter #Assign thread counter
  def run(self):
    print ("Starting " + self.name) #Running the thread
    # Get lock to synchronize threads
    threadLock1 = threading.Lock()
    threadLock1.acquire()
    # Call web_downloader() to download web pages.
    web_downloader(self.counter)
    # Free lock to release next thread
    threadLock1.release()
```

```
class userThread (threading.Thread): #Thread that handles Tab and user requests.
  def __init__(self, threadID, name, counter):
    threading.Thread.__init__(self) #Initialize the thread.
    self.threadID = threadID #Save Thread ID.
    self.name = name #Specify Thread name if exists
    self.counter = counter #Assign thread counter
  def run(self):
    print ("Starting Web Downloader\n") #Running the thread
    # Get lock to synchronize threads
    threadLock = threading.Lock()
    threadLock.acquire()
    # Call webtab_thread() to ask the user to open up new Tabs
    webtab_thread()
    # Free lock to release next thread
    threadLock.release()

"""
- Function that handles Receive images from web server
"""

def recev_images(imgSocket, img_file):

   imageMessage = imgSocket.recv(1024) #receive the result
   img_temp = str(imageMessage); #convert imageMessage to string
   if (img_temp.find("200 OK") < 0 and img_temp.find("301") < 0): #if statement to check
if the image does exist
      img_file = open ("error_log.html", 'wb'); #open new html
      img_file.write(imageMessage); #write the imageMessage in the img_file
      img_file.close(); #close the file
      error_code = img_temp.find("HTTP/1.1") #error check
      imgSocket.close(); #close Socket connection
      return;


   if (img_temp.find("Content-Length:") == -1): #if statement that checks for the content
length
      img_file = open ("error_log.html", 'wb'); #open a new file
      img_file.write(imageMessage); #write the imageMessage in the img_file
      img_file.close(); #close the file
      print("Can't find Content-Length") #print
      error_code = img_temp.find("HTTP/1.1") #error check
```

```
        imgSocket.close(); #close socket connection
        return;


    img_content_len = img_temp.find("Content-Length:"); #else statement if the content length exist
    x = img_temp[img_content_len+16:] #store the img_temp in variable x
    tmp4 = x.split('\\r\\n')# split x at the end of the line
    img_data_len = int(tmp4[0]) #store the number of the content length starting from index 0 to the end of the line

    img_headers_rsp = imageMessage.split(b'\r\n\r\n')[0] #split the imageMessage
    img_file.write(imageMessage[len(img_headers_rsp)+4:]); #write the image into the image file
    img_temp3 = str((imageMessage[len(img_headers_rsp)+4:])) #file image message to a string and store it in image temp3

    while (len(img_temp3) < img_data_len * 3): #while loop to keep runing if the long the image temp3 is less than the  content length *3
        imageMessage = imgSocket.recv(img_data_len) #receive the result from the user
        if (len(imageMessage) < 0): break # if the image message length less than 0 stop and break out of the loop
        img_temp3 += str(imageMessage) #adding image temp3 to the string of the image message
        img_file.write(imageMessage); #write the image message in image file
        if (img_temp3[-6:] == "b''b'''"): break; #Break if no data is received.

    img_file.close(); #close the file
    return;

"""
- Class that handles parsed HTML data including images
"""

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs): #define a function to handle the tag start
        hostname = global_arr[0]
        if tag == 'img': #if statement to check if the tag is equal to img
            for a in attrs: #for loop
                if a[0] == 'src': #if statement to check if the first index of a equal to src
                    self.imageUrl = a[1] #if yes store the 2nd index in self image url
```

```
            data = "GET /" + "/" + self.imageUrl + " HTTP/1.1\r\nHOST: " + hostname
+ "\r\n" + header #Set up Data (GET command + formatted header)
            imgSocket = socket(AF_INET, SOCK_STREAM) #create TCP socket for
server, remote port 80.
            imgSocket.connect((hostname, serverPort)) #build a connection with the
server using the name and the port.

            path_count = self.imageUrl.count("/") #set the path count to the number of /
            file_exten = self.imageUrl.split("/") #split the self image URL after /
            img_path = os.getcwd() #set the image path to value of the returns current
working directory of a process
            i = 0; #define a i variable
            final_path = "" #define final path variable
            while i < path_count: #while loop
               final_path += file_exten[i] + "/"; #set the final path result
               i = i + 1; #increment i
            if not os.access(img_path + "/" + final_path, os.W_OK): #if not statement to
test for access to path.
               os.makedirs(img_path + "/" + final_path) #recursive directory creation
function
            img_file = open (img_path + "\\" + self.imageUrl, 'wb'); #open an img file
            imgSocket.settimeout(10) #set timeout equal to 10 sec
            imgSocket.send(data.encode()) #send the decoded data to the socket
            recev_images(imgSocket, img_file) #receive images by calling the imgSocket
and img_file

      elif tag == 'div':
         for a in attrs:
            if a[0] == 'style': #Look for Background image file.
            self.imageUrl = a[1] #if yes store the 2nd index in self image url
            self.imageUrl = self.imageUrl.split("url")[1] #Check when you find the url of
the background image.
            self.imageUrl = self.imageUrl.split("(")[1] #Split unnecessarily parenthesis or
single qoute.
            self.imageUrl = self.imageUrl.split(")")[0] #Split unnecessarily parenthesis or
single qoute.
            if (self.imageUrl.find("'") > 0): #Split unnecessarily parenthesis or single
qoute.
               self.imageUrl = self.imageUrl.split("'")[1]

            if (self.imageUrl.find("\\") > 0): #Split unnecessarily parenthesis or single
```

qoute.

```
                self.imageUrl = self.imageUrl.split("\\")[0]

            data = "GET /" + "/" + self.imageUrl + " HTTP/1.1\r\nHOST: " + hostname + "\r\n" + header #Set up Data (GET command + formatted header)
            imgSocket = socket(AF_INET, SOCK_STREAM) #create TCP socket for server, remote port 80
            imgSocket.connect((hostname, serverPort)) #build a connection with the server using the name and the port

            path_count = self.imageUrl.count("/") #set the path count to the number of /
            file_exten = self.imageUrl.split("/") #split the self image URL after /
            img_path = os.getcwd() #set the image path to value of the returns current working directory of a process
            i = 0; #define a i variable
            final_path = "" #define final path variable
            while i < path_count: #while loop
                final_path += file_exten[i] + "/"; #set the final path result
                i = i + 1; #increment i
            if not os.access(img_path + "/" + final_path, os.W_OK): #if not statement to test for access to path.
                os.makedirs(img_path + "/" + final_path) #recursive directory creation function
            img_file = open (img_path + "\\" + self.imageUrl, 'wb'); #open an background file
            imgSocket.settimeout(10) #set timeout equal to 10 sec
            imgSocket.send(data.encode()) #send the decoded data to the socket
            recev_images(imgSocket, img_file) #receive images by calling the imgSocket and img_file

        elif tag == 'link':
            for a in attrs: #for loop
                if a[0] == 'rel' and a[1] == 'stylesheet': #if statement to check if the first, second indexes of an equal to stylesheet and rel
            if (str(attrs[1][1]) != "None"):
                self.imageUrl = attrs[1][1] #if yes store the 2nd indexes and second dimension in for CSS objects
                if (self.imageUrl != 'stylesheet'): #Make sure to point to path of the object
                    data = "GET /" + "/" + self.imageUrl + " HTTP/1.1\r\nHOST: " + hostname + "\r\n" + header #Set up Data (GET command + formatted header)
                    imgSocket = socket(AF_INET, SOCK_STREAM) #create TCP socket for
```

```
server, remote port 80
                imgSocket.connect((hostname, serverPort)) #build a connection with the
server using the name and the port

                path_count = self.imageUrl.count("/") #set the path count to the number
of /
                file_exten = self.imageUrl.split("/") #split the self image URL after /
                css_object = file_exten[-1] #Save the CSS object file which is last element
in the list
                img_path = os.getcwd() #set the image path to value of the returns current
working directory of a process
                i = 0; #define a i variable
                final_path = "" #define final path variable
                while i < path_count: #while loop
                   final_path += file_exten[i] + "/"; #set the finsl path result
                   i = i + 1; #increment i
                if (final_path.find("'b'") > 0): #Clean up junk in the data
                   final_path = final_path.split("'b'")
                   temp_clean = final_path[0]
                   final_path = temp_clean + final_path[1] #Store clenaed up path

                if not os.access(img_path + "/" + final_path, os.W_OK): #if not statement
to test for access to path.
                   os.makedirs(img_path + "/" + final_path) #recursive directory creation
function
                css_file = open (img_path + "\\" + final_path + css_object, 'wb'); #open an
CSS object file
                imgSocket.settimeout(10) #set timeout equal to 10 sec
                imgSocket.send(data.encode()) #send the decoded data to the socket
                recev_images(imgSocket, css_file) #receive images by calling the
imgSocket and css_file
   def handle_endtag(self, tag):  #define a function to handle the tag end
      return;

   def handle_data(self, data): #define a function to handle the data
      return;

"""
webtab_thread() function that will be called as the user thread.
Interactively asks the user to open up a new tab as a multithreaded application.
The user answers with yes or no. In no tabs needed anymore, it waits for all the thread to
```

```
finish up then exits.
If the users answers yes, create a thread, and calls web_downloader() to handle users requests.
"""
def webtab_thread():

    stay = 1 #Stay in the loop flag
    web_counter = 0 #internal web counters for how many tabs will be opened.
    while (stay == 1):
        web_tab = input ("Open New Tab?. Yes or No\n") #Asks the users to open up tabs.
        if (web_tab == "Yes" or web_tab == "yes"): #of the user wants a new tab.
            web_counter += 1; #Increment internal thread counter.
            thread_name = "Tab-" + str(web_counter) #Thread name + counter value.
            thread2 = webThread(web_counter, thread_name, web_counter) #Calls weThread
that calls web_downloader()
            thread2.start() #Start the thread.
            time.sleep(2)
        elif (web_tab == "No" or web_tab == "no"): #If users say no, and no threads will be
created.
            stay = 0; #Set stay in the loop flag to exit
            if(web_counter > 0):
                thread2.join() #Wait for the the threads to be done.

"""
web_downloader() function that will be called from the user thread
web_downloader() will handle the users request to access web page servers
the function handles multiple formats of the passed links from the users
It creates and establish a socket with the requested server.
Sends the request and handles the received data.
calls the HTML parser class to handle HTML object files.
"""
def web_downloader(Parsercounter):

    """
    - Handling Website links in multiple formats
    """
    web_link="" #Web server name
    file_name="" #file name of the stored HTML information.
    hostname="" #hostname of the webserver will be used to send the requests.
    web_link = input('Please Enter the link:') #asking the user for input
    if ( web_link.find("http://") >= 0 or web_link.find("https://") >= 0 ): #if statement to
```

```
look for HTTP or HTTPS
        tmp = web_link.split("/", 3); #split the link after the third / and store it in tmp
        hostname = tmp[2]; #set host name to the third indes of tmp
        file_name = hostname #Open a file to write with the same name as the requested server.
        web_path="" #The path of the objected HTML file.
        if (web_link.count("/") >= 3): #if statement that checks the number of / is = or bigger than 3
            web_path = "/" + tmp[3]; # if found set the web path

    else:
        tmp = web_link.split("/", 1); #split the web link after the first /
        hostname = tmp[0]; #set the host name to the first index of tmp
        file_name = hostname #Open a file to write with the same name as the requested server.
        web_path="" #The path of the objected HTML file.
        if (web_link.count("/") > 0): #if the count of / is bigger than 0
            web_path = "/" + tmp[1];

    print ("\nGet web info for: " + hostname + web_path); #Print the website link to validate correctness


    """
    - Checking if the path is a file (Picture or a PDF) rather than an HTML Webpage
    """
    NotHtml_flag = 0;
    if (web_path.find(".jpg") >= 0 or web_path.find(".pdf") >= 0 or web_path.find(".png") >= 0):
        NotHtml_flag = 1;

    """
    - Creating Socket
    - Connect to host/web server
    - Set up Data (GET command + formatted header)
    - Send request to the web server.
    """
    clientSocket = socket(AF_INET, SOCK_STREAM) #create TCP socket for server, remote port 80
    clientSocket.connect((hostname, serverPort)) #build a connection with the server using the name and the port
```

```
clientSocket.settimeout(10) #set timeout equal to 10 sec
data = "GET /" + web_path + " HTTP/1.1\r\nHOST: " + hostname + "\r\n" + header
#use get and host to get the data
clientSocket.send(data.encode()) #encode the message and send it to the server

if (NotHtml_flag == 1): #Check if it's not a direct object file.

    path_count = web_path.count("/") #set the path count to the number of /
    file_exten = web_path.split("/") #split the self image URL after /
    img_path = os.getcwd() #set the image path to value of the returns current working
directory of a process
    i = 0; #define a i variable
    final_path = "" #define final path variable
    while i < path_count: #while loop
        final_path += file_exten[i] + "/"; #set the final path result
        i = i + 1; #increment i
    if not os.access(img_path + "/" + final_path, os.W_OK): #if not statment to test for
access to path.
        os.makedirs(img_path + "/" + final_path) #recursive directory creation function
    img_file = open (img_path + "\\" + web_path, 'wb'); #open an img file
    print ("Downloading "+file_exten[path_count]) # the object filename and path.
    recev_images(clientSocket, img_file) #receive images by calling the imgSocket and
img_file
    print ("Downloading is done."); #print a message

else: #It's a webpage not an object file.
    file = open (file_name+".html", 'wb'); #open a file to that will store the HTML page.
    modifiedMessage = clientSocket.recv(1024) #receive the result from the user
    temp2 = str(modifiedMessage); #Store the first message response to parse the
incoming data and split the header

    """
    - Check for bad responses.
    - Check if web server provide content length of data
    """

    if (temp2.find("200 OK") < 0 and temp2.find("301") < 0):  #if statement to check for
bad response
        file = open (file_name+".html", 'wb'); #open a file
        file.write(modifiedMessage); #write the modifiedMessage in the file
        file.close();#write the modifiedMessage in the file
```

```
        error_code = temp2.find("HTTP/1.1") #check for HTTP/1.1
        print ("Error code = " + temp2[error_code+8:error_code+12]) #print error
message
        clientSocket.close(); #close scoket connection
        quit() #exit the if statement

    if (temp2.find("Content-Length:") == -1): #if statement that Checks if web server
provide content length of data
        file = open (file_name+".html", 'wb'); #open a file
        file.write(modifiedMessage); #write the modifiedMessage in the file
        file.close();#write the modifiedMessage in the file
        print("Can't find Content-Length") #print a message
        error_code = temp2.find("HTTP/1.1") #error check
        print ("Error code = " + temp2[error_code+8:error_code+12]) #print error
message
        clientSocket.close(); #close scoket connection
        quit() #exit the if statement

    """
    - Get the content Length of data
    """
    content_len = temp2.find("Content-Length:"); #search for content length in temp2
    x = temp2[content_len+16:] #get the length after the content length
    tmp3 = x.split('\\r\\n')# split x till the end of the line
    data_len = int(tmp3[0]) #store the content length on data len

    """
    - Split header response information from actual content data
    """
    headers_rsp = modifiedMessage.split(b'\r\n\r\n')[0] #split the header
    file.write(modifiedMessage[len(headers_rsp)+4:]); # write the data to the file
    temp3 = str((modifiedMessage[len(headers_rsp)+4:])) #convert the data to string and
store it in temp3

    """
    - Receive all content data until it's done
    """

    while (len(temp3) < data_len * 2): #while loop
        modifiedMessage = clientSocket.recv(data_len) #receive the result from the user
        temp3 += str(modifiedMessage) #add temp3 to the string value of modifiedMessage
```

```
            file.write(modifiedMessage); #write the modifiedMessage to the file
            if temp3[-6:] == "b''b'''": break #if statement to check if the message is empty then
break
            if (temp3[-8:] == "</html>'"): break; #if it finds the end of the HTML file, break.

        parser = MyHTMLParser() #Assign parser to parse HTML files.
        if (Parsercounter == 1): #The first Thread execution.

            global_arr[0] = hostname #Store hostname in the global array so it can be used in
the HTML parser.
            parser.feed(temp3)# Feed the content data to HTML parser to get images.
            Parsercounter_gb[Parsercounter+1] = 1 #When the parser feed is done for the first
thread, sets the flag for the next thread.

        else:
            while (True):
                if (Parsercounter_gb[Parsercounter] == 1): #If not the first thread enter this
while loop until flag is set
                    global_arr[0] = hostname #Store hostname in the global array so it can be used
in the HTML parser.
                    parser.feed(temp3)# Feed the content data to HTML parser to get images.
                    Parsercounter_gb[Parsercounter+1] = 1 #Set the next thread's flag to start the
parsing.
                    break;
        print ("\nWeb information sotred in "+file_name); #print a message
        file.close(); #close the file
        clientSocket.close(); #close the socket connection

    try:
      thread_counter += 1; #increment the number of threads
      thread_name = "Thread-" + str(thread_counter) #Assign thread name + the counter
value
      thread1 = userThread(thread_counter, thread_name, thread_counter) #Call the user
thread that actively asks the users if want to open up new tabs
    except:
      print ("Error: unable to open up new tab") #Threads exception error

    thread1.start() #start user thread to interact with the user on the terminal line
thread1.join() # wait until the user thread exists
```

Testing Procedure, including Description of Inputs:

Open up a commands line terminal on Windows (cmd). Start the application. The user will be asked if wants to open up a tab or not. Type "yes", then will be asked to provide a link to download. After providing a link, the user will be asked again if wants to open up another tab and so forth. Three websites were tested concurrently:

http://nabil.eng.wayne.edu/
http://webpages.eng.wayne.edu/~ad5781/mypage.htm
http://suzanars.eng.wayne.edu/

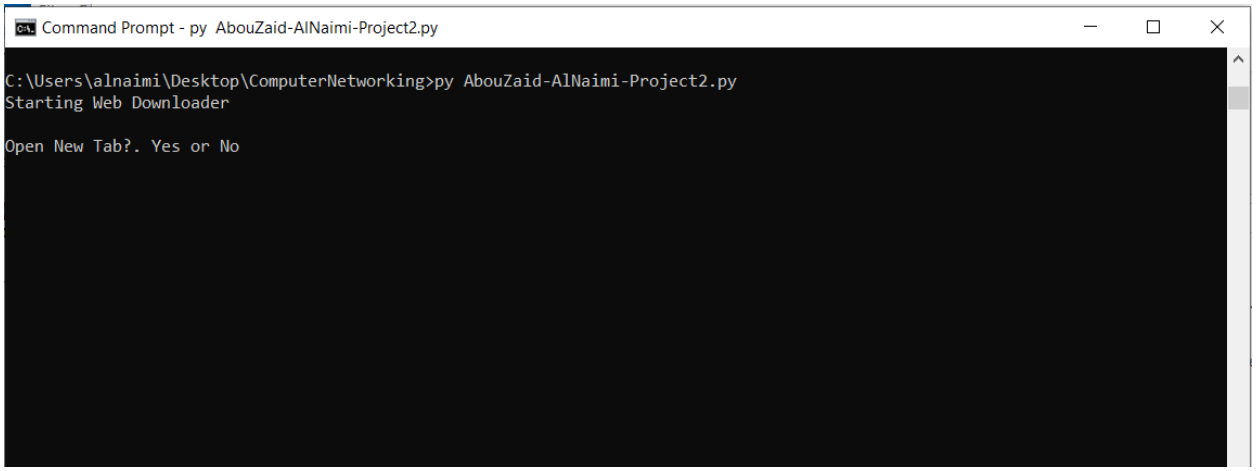
The program will download webpages concurrently with all objects downloaded with the same path on the web server.

Also, the program was tested downloading an object directly and a different format of a website:

webpages.eng.wayne.edu/~ad5781/mypage.htm
http://nabil.eng.wayne.edu/_resources/images/nabil.jpg

Screenshots and Their Explanations:

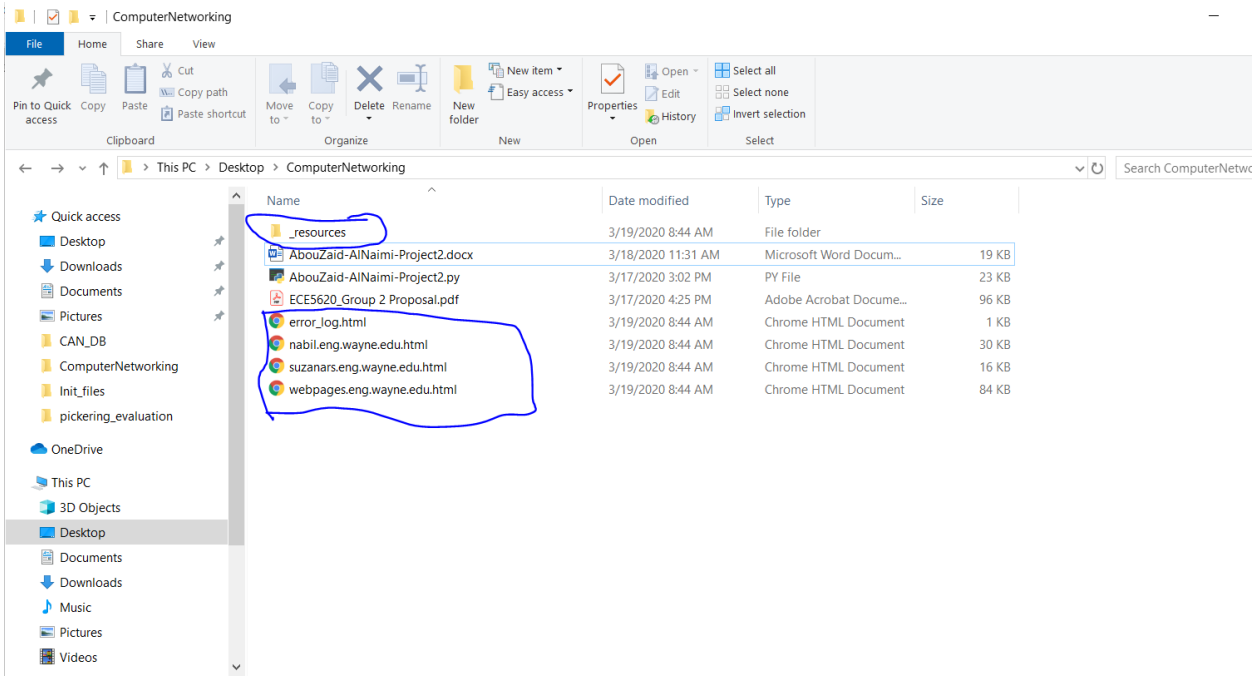
1- Program Run:



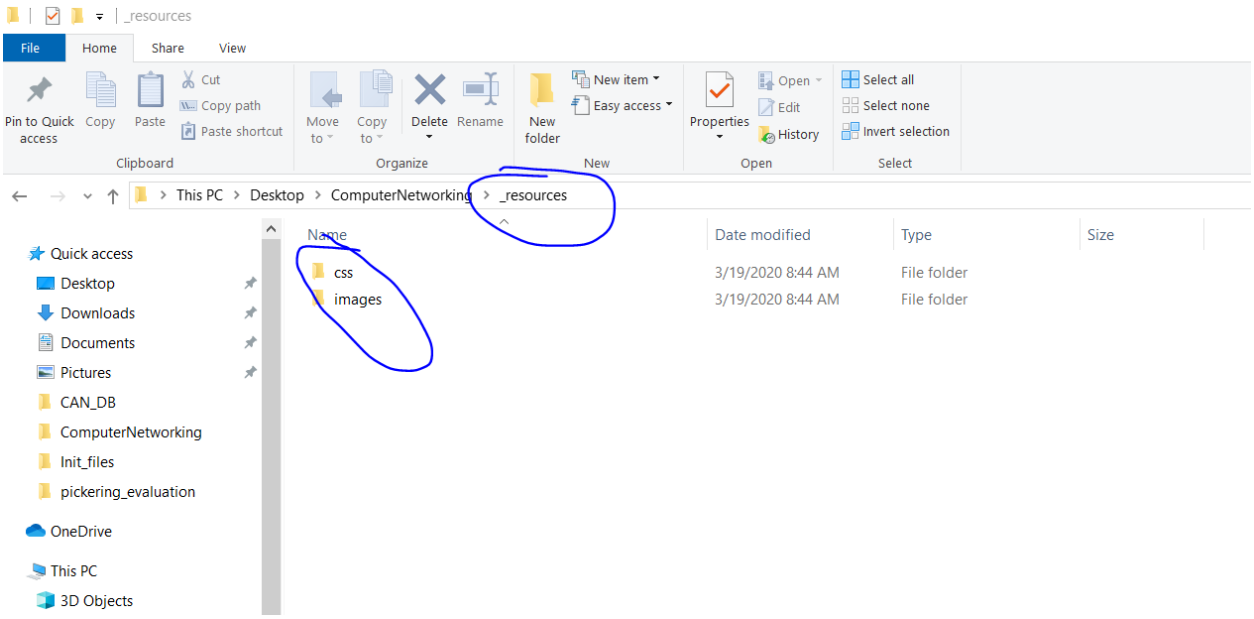
2- Testing 3 websites concurrently (Multi-threading):

```
Command Prompt

C:\Users\alnaimi\Desktop\ComputerNetworking>py AbouZaid-AlNaimi-Project2.py
Starting Web Downloader

Open New Tab?. Yes or No
yes
Starting Tab-1
Please Enter the link:http://nabil.eng.wayne.edu/

Get web info for: nabil.eng.wayne.edu/
Open New Tab?. Yes or No
yes
Starting Tab-2
Please Enter the link:http://webpages.eng.wayne.edu/~ad5781/mypage.htm

Get web info for: webpages.eng.wayne.edu/~ad5781/mypage.htm
Open New Tab?. Yes or No
yes
Starting Tab-3
Please Enter the link:http://suzanars.eng.wayne.edu/

Get web info for: suzanars.eng.wayne.edu/Open New Tab?. Yes or No

no

Web information sotred in nabil.eng.wayne.edu

Web information sotred in webpages.eng.wayne.edu

Web information sotred in suzanars.eng.wayne.edu

C:\Users\alnaimi\Desktop\ComputerNetworking>
```

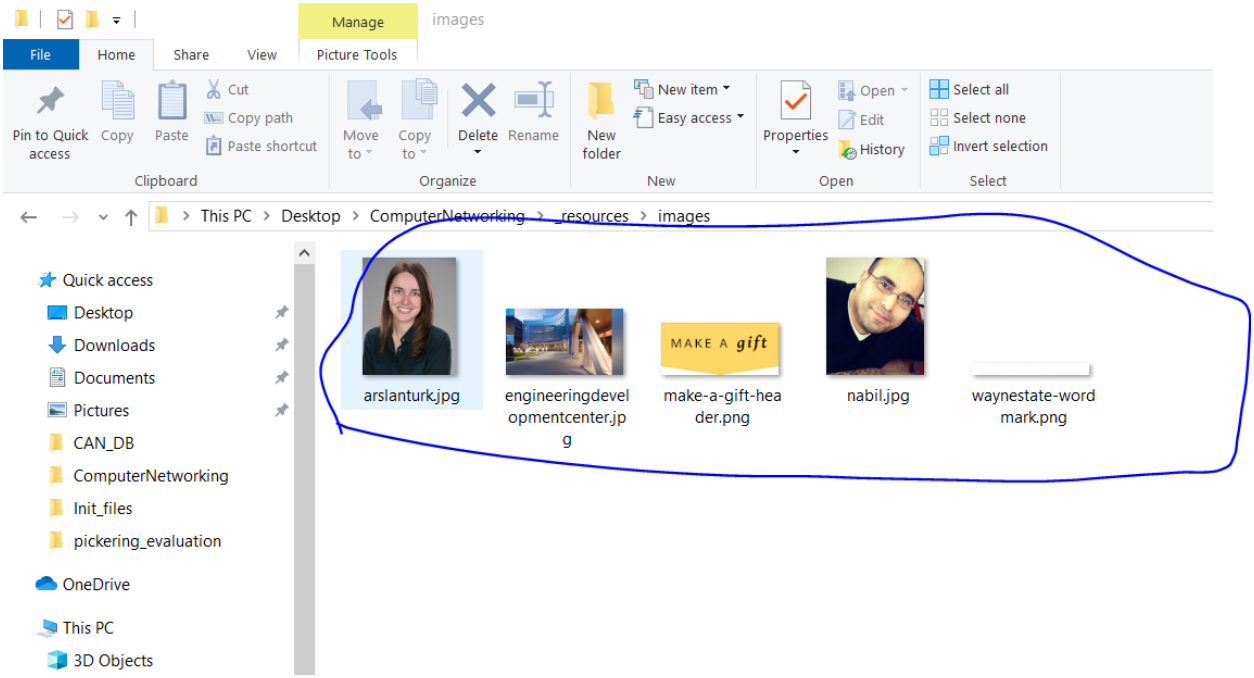
3- Generated webpages, downloaded object files:



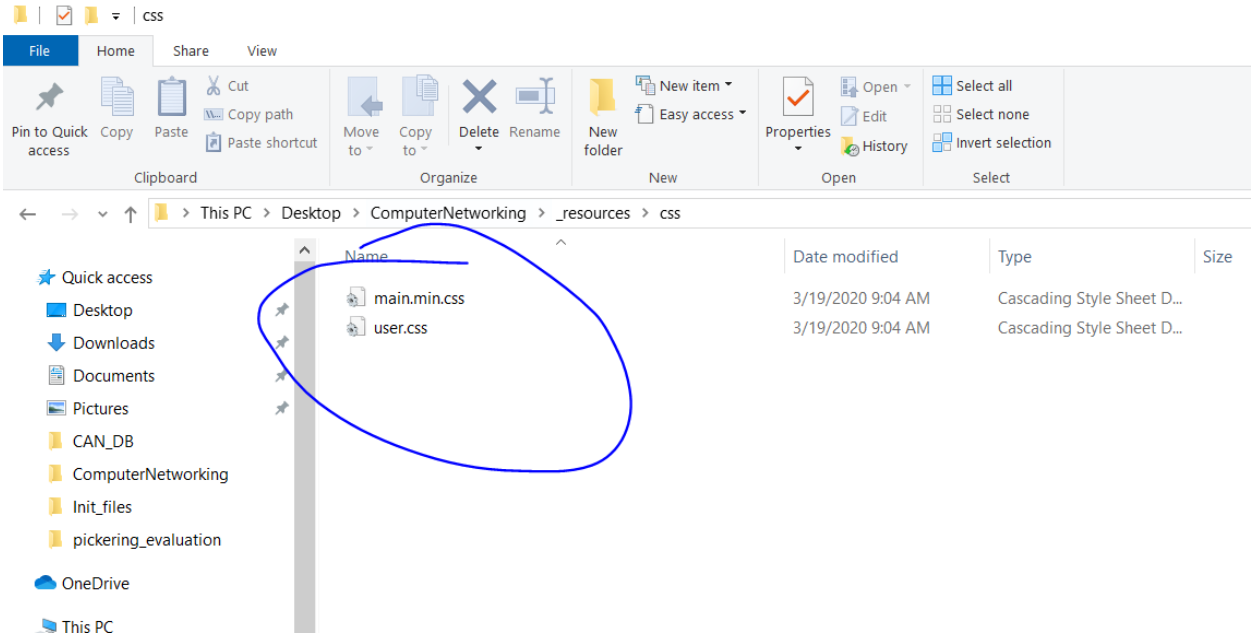
Screenshot 1 Generated web pages and folders



Screenshot 2 generated subfolders

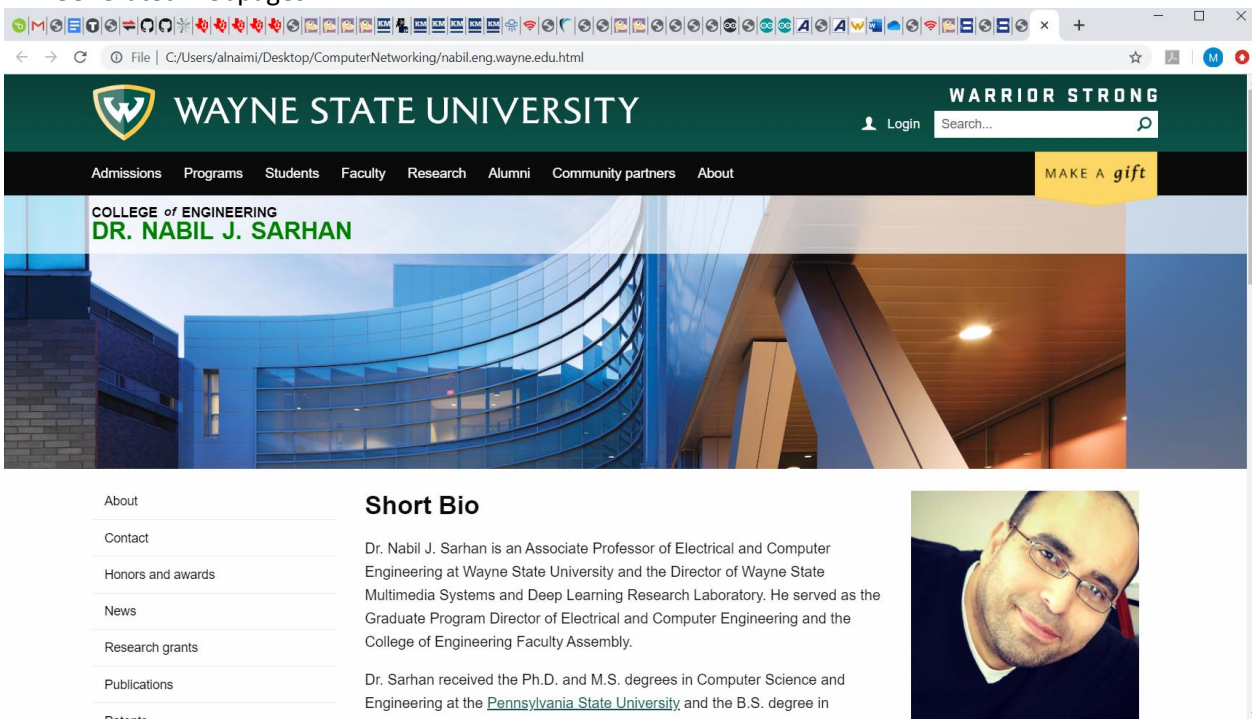


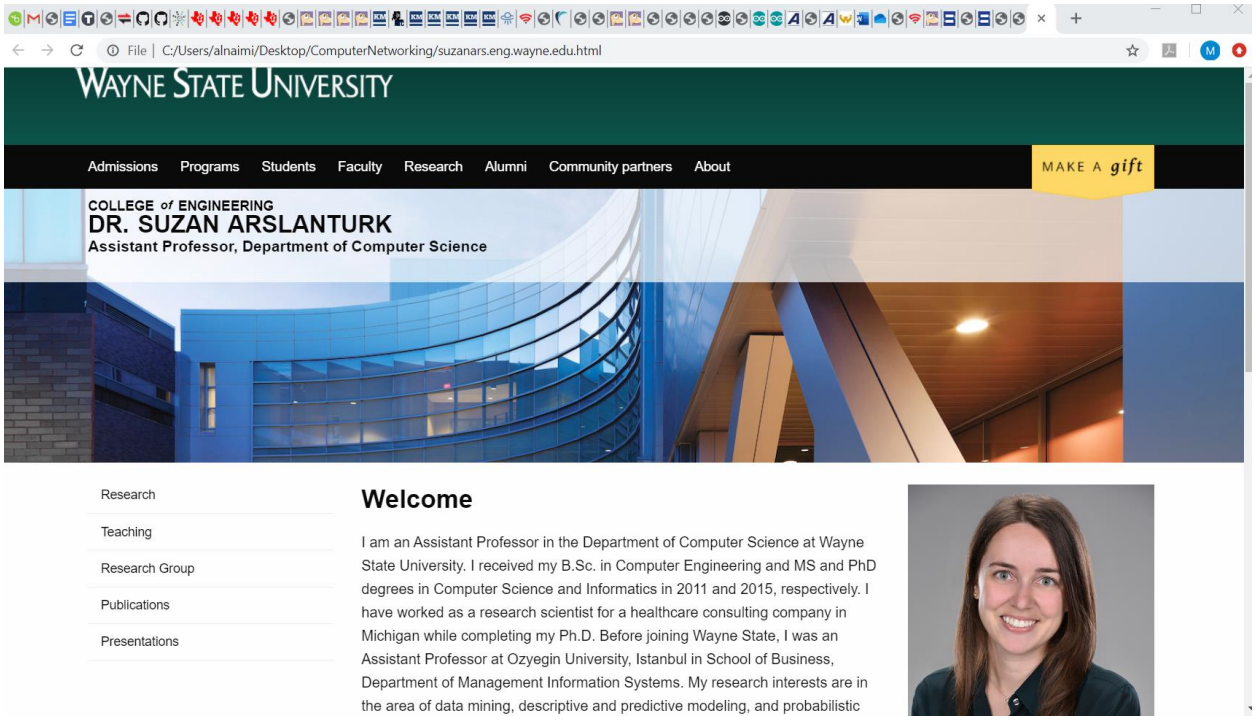
Screenshot 3 Downloaded pictures

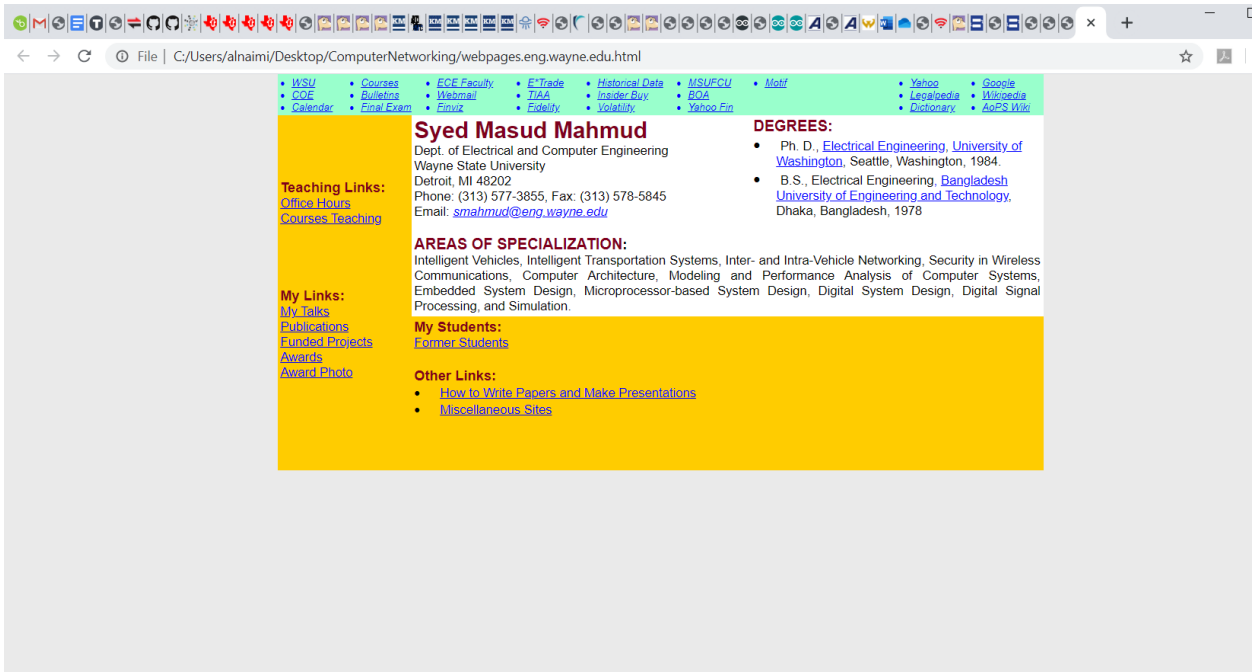


Screenshot 4 CSS objects

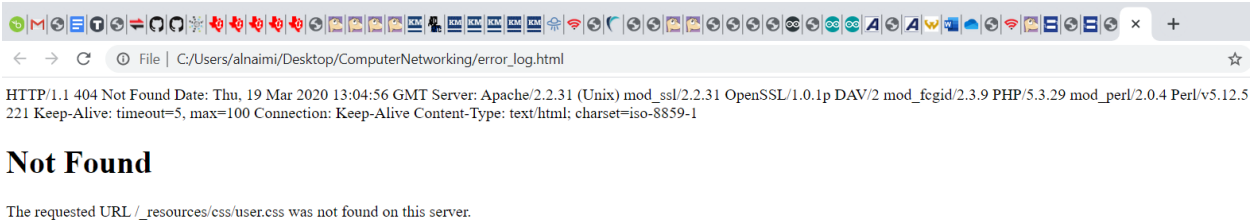
4- Generated webpages:



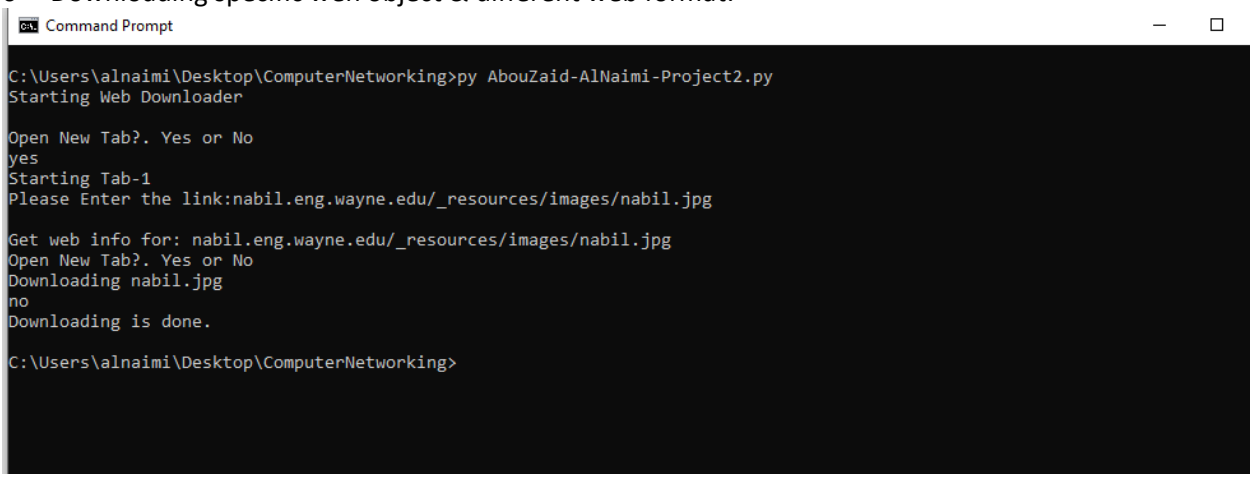


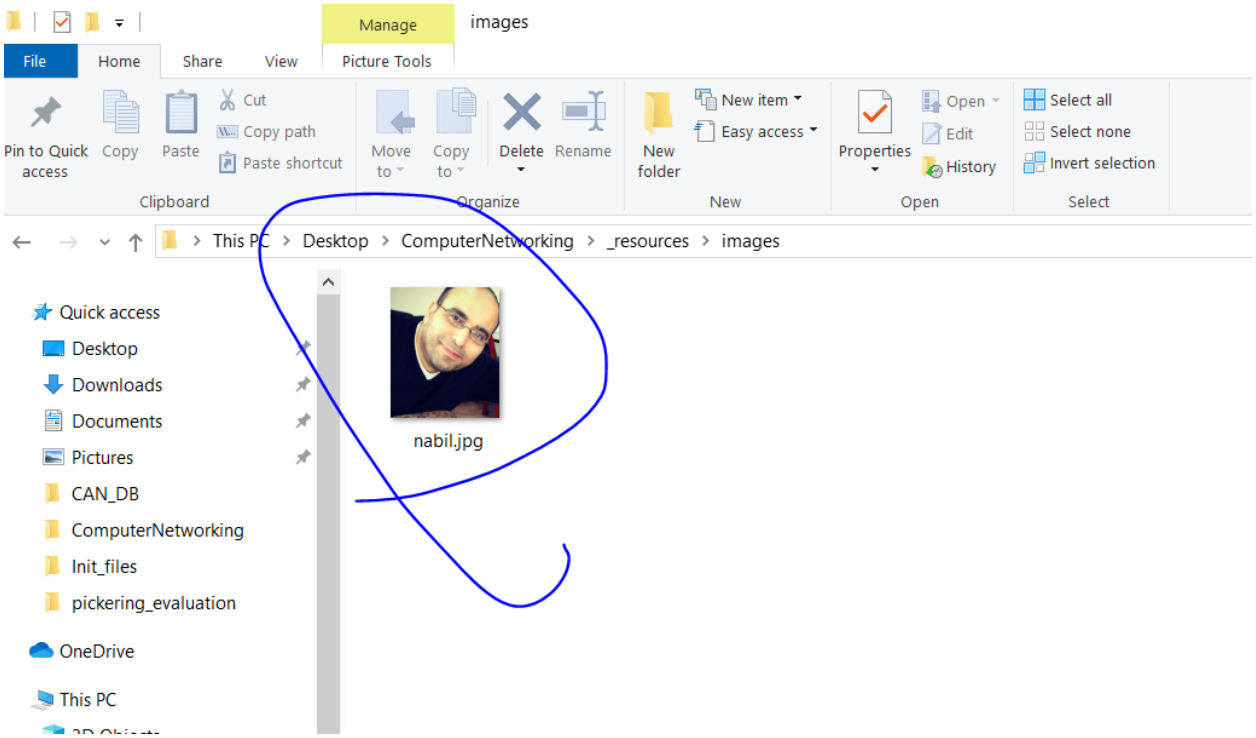


5- Generated error_log.html:



6- Downloading specific wen object & different web format:





Program Completion Status and Self-Critique:

- Does your program meet all requirements? If not, explain the problem.

  Yes.
- Does the program run correctly all the time? If not, explain the problem.
  Yes. However, secured https websites could cause problems.

- Did you adequately test the program? If not, specify.
  Yes

- Is the program well documented?

  Yes

Performance Evaluation Methodology:

The web downloader application implements a multi-threading concept where the user will be asked to open multiple **tabs** (user threads) each time the user wants to access a website page, or a web object on a web server. The user could answer ("YES" or "yes") to open a new tab (thread) and ("NO" or "no") to stop the application. The program will wait until all objects/pages are downloaded then exists.

The program is allowed to open 10 tabs (threads) concurrently. In addition, multi-threading locks are used to synchronize concurrent threads and shared accesses of all threads.

The program keeps track of all web objects, need to be downloaded and create the appropriate paths the object has on the web server. If the path has been already created from a previous object, then the program will only download the object in the created path.

Comparative Results:

Project 1	Project 2
Single websites access	Up to 10 Multi-tabs (Threads)
Accepts different URL format	Accepts different URL format
Only webs pages access	Web pages and web objects
One main application	Main application & 1 active user thread
Only handles images	Handles images, pdf, background and CSS
Store web objects according to their server path	Store web objects according to their server path
Could get extra bytes of data at the end of stream, handling data not precisely.	Added checks to handle data received precisely by checking end of HTML file and empty bytes.

Analysis of the Results:

Program highlights:

1- 10 tabs could be opened.
2- The program accepts multiple URL format including **http** or without.
3- You can access a web page directly, or specific object on the server.
4- The objects within a web page will be downloaded with respect to the same path on the server.
5- The program stores webpages locally with the same server hostname.
6- If the program can't access an object on the server, it will generate an error log page (error_log.html) to specify what was the cause of the error.

Limitations:

1- The program can't exceed more than 10 threads due to memory allocations and considerations of overwhelming the host controller.
2- Program doesn't handle all different types of web server accesses including secured and encrypted web servers.
3- The program doesn't redirect automatically to pages that have been moved to different locations on the web server.