Project Report

MULTI THREADED WEB CRAWLER

Operating System

Group Members:

1. Afnan Ansari 11378
2. Muhammad Saad 11365
3. Muhammad Nomir 11330

Submitted To:

Miss Fizza

Date: December 30, 2021

ACKNOWLEDGEMENT:

“This Report Have Been Prepared Through The Help Of Online Consultation Of Different Webpages For Datasheets & Also In This Report We Explain Our Project Thoroughly. This Project Report Have Been Created Within The Given Time Spectrum. All Of The Given Requirement Are In This Report With Correct Format.”

ABSTRACT:

In this whole report you will see all our research and hard-work put into this report. We created a project of multi-threaded web crawler and explains its functionalities. In this report we tell you about how multi-threaded web crawler works and also show different uses for a multi-threaded web crawler. In this report you will also be shown that how different libraires are used to ease the method of creating a multi-threaded web crawler.

Web Crawler:

A web crawler also known as a web spider is a program or automated script which browses the World Wide Web in a methodical, automated manner. This process is called Web crawling or spidering.

Many legitimate sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, that will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating HTML code.

Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses or other information.

Importance Of A Web Crawler:

Thanks to digital revolution, the total amount of data on the web has increased. In 2013, IBM stated that 90% of the world's data had been created in the previous 2 years alone, and we continue to double the rate of data production every 2 years. Yet, almost 90% of data is unstructured, and web crawling is crucial to index all these unstructured data for search engines to provide relevant results. According to Google data, interest in the web crawler topic has decreased since 2004. Yet, at the same time period, interest in web scraping has outpaced the interest in web crawling.

Increasing interest in analytics and data-driven decision making are the main drivers for companies to invest in scraping.

Crawling done by search engines is no longer a topic of increasing interest since they have done this since the early 2000s.

Search engine industry is a mature industry dominated by Google and Baidu, so few companies need to build crawlers.

How Does A Web Crawler Work:

Web crawlers start their crawling process by downloading the website’s robot.txt file. The file includes sitemaps that list the URLs that the search engine can crawl. Once web crawlers start crawling a page, they discover new pages via links. These crawlers add newly discovered URLs to the crawl queue so that they can be crawled later. Thanks to these techniques, web crawlers can index every single page that is connected to others.

Examples Of A Web Crawler:

All the search engines need to have a web crawler some common examples are:

- Amazonbot is an Amazon web crawler for web content identification and backlink discovery.
- Baiduspider for Baidu
- Bingbot for Bing search engine by Microsoft
- DuckDuckBot for DuckDuckGo
- Exabot for French search engine Exalead
- Googlebot for Google
- Yahoo! Slurp for Yahoo
- Yandex Bot for Yandex

WORKING:

Explanation:

Our Code Can Be Broken Into Five Parts.

1.Menu/First User Choice
2.Base1
3.Base2
4.Threading Class
5.Main Work Of Multi-Threaded Web Crawler

1.Menu/First User Choice:

The first part is the simplest it will serve as an opening message to the user and give the user the choice to either work with the multi-threaded web crawler using a CSS selector or not and depending upon the user choice the program will move on.

CODE:

```
def Home():
    print("\t\t\t\t\t\t==========================")
    print("\t\t\t\t\t\tMulti Threaded Web Crawler")
    print("\t\t\t\t\t\t==========================")
    print("Do You Want To Use Your Own Css Selector(Y/N) : ")
    choice=input("\nPlease Enter Your Choice : ")
    print("")
    if(choice=='Y' or choice=='y'):
        Base_2()
    elif(choice=='N' or choice=='n'):
        Base_1()
```

2.Base1:

The Base1 will run if the user doesn't chooses CSS selector option or rather more specifically selects the option 'N' or 'n' because now the user won't be using a CSS selector the program will jump to Base1 where the user will be asked to enter the URL of the website to crawl and after that the number of threads which will be used to crawl the URL after that user will be given the option to select a CSS tag in which the program will search into, the user can enter their own CSS tag or they can also select from the given choices after that another option will be given to the user which will ask the user to select the HREF/link inside the tag or the string/title between the tags. After all of these things the program will start to run it will make a Queue of crawling links, initialize a thread lock which will stop other threads from doing the same work as the current thread and also not access any other given instruction while a thread is working, after that we will put the user URL into the crawling link queue that we just created and create a set of visited links in which all the visited links through the user URL will be put it will come into use in the future. After that a list of threads and a list of error links will be created which will also come into work in the future, then a small library of time comes into use which will be useful for us in telling us how much time our program takes exactly. After this a loop will run till the range of the threads user entered to create, so the amount of threads will be created inside the loop our threading class will be initialized in which all our necessary information is passed. After this loop another loop is there to join our threads because threads tend to work without order and complete without order so this loop and join will allow us to work with one thread at a time with good thread management. After that simple printing is used to show the URL, threads used, total time, total visited pages and total links with errors, also the program will run again if the user chooses to run again the user will be given the choice to enter another URL or exit the program. In this function filing will also be done in one file it will only save the entered URL(s) and in the other file it will save all the information regarding the crawling of the URL(s) and it will also show these records in the program.

CODE:

```
def Base_1():
    f=open("Base1_Links.txt","r")
    print(f.read())
    f=open("Base1_Links.txt","a")
    url=input("Please Enter URL To Crawl : ")
    f.write("\n"+url+"\n")
    f.close
    threads=input("\nPlease Enter The Number Of Threads : ")
    print("\nPlease Select A Css Tag :
\n\n1.'a'\t\t2.'div'\n\n3.'p'\t\t4.'h'\n\nYou Can Also Enter Your Own
Css Tag")
    choice=input("\nPlease Enter Your Choice : ")
    if(choice=='1'):
        css_tag='a'
    elif(choice=='2'):
        css_tag='div'
    elif(choice=='3'):
        css_tag='p'
    elif(choice=='4'):
        css_tag='h'
    else:
        css_tag=choice
    print("\nPlease Select If You Want Links Inside The Selected Tag
OR The Text Between The Selected Tags : \n\n1.Links Inside The
Selected Tag(href)\n2.The Text Between The Selected Tags(Title)")
    choice=input("\nPlease Enter Your Choice : ")
    if(choice=='1'):
        tag_info='href'
    elif(choice=='2'):
        tag_info='title'
    print("")
    css_selector=None
    crawling_links=queue.Queue()
    url_lock=threading.Lock()
    crawling_links.put(url)
    visited_links=set()
    crawling_threads=[]
    error_links=[]
    start_time=time.time()
    for i in range(int(threads)):

crawler=Crawler(url=url,css_tag=css_tag,tag_info=tag_info,css_selector
=css_selector,crawling_links=crawling_links,visited_links=visited_link
s,error_links=error_links,url_lock=url_lock)
        crawler.start()
        crawling_threads.append(crawler)
    for crawler in crawling_threads:
        crawler.join()
    end_time=time.time()-start_time
    visited=len(visited_links)
    errors=len(error_links)

print("\n=============================================================
==========================================================\n")
    print(f"URL : {url}")
    print(f"Total Threads : {threads}")
    print(f"Total Time Taken : {end_time}")
    print(f"Total Number Of Visited Pages : {visited}")
    print(f"Total Number Of Links With Errors : {errors}\n")

print("==============================================================
========================================================\n")
    f=open("Base1_Links_With_Info.txt","r")
    print(f.read())
    f=open("Base1_Links_With_Info.txt","a")
    f.write("\nURL : "+url+"\nThreads : "+threads+"\nTime :
"+str(end_time)+"\nVisited Pages : "+str(visited)+"\nLinks With Errors
: "+str(errors)+"\n")
    f.close

print("==============================================================
========================================================\n")
    print("Do You Want To Crawl Another Webpage(Y/N) : ")
    choice=input("\nPlease Enter Your Choice : ")
    print("")
    if(choice=='Y' or choice=='y'):
        Home()
    elif(choice=='N' or choice=='n'):
        exit
```

3.Base2:

The Base2 will work the same way and on the same principles of Base1 the only difference here would be that the two options of CSS tag selection and HREF/link & string/title selection will not be given as the user would be entering their own CSS selector and after that giving the option or asking for a CSS tag would be unnecessary and it will only go through the HREF/link of the selector nothing else, other than this everything will work the same as Base1.

CODE:

```
def Base_2():
    f=open("Base2_Links.txt","r")
    print(f.read())
    f=open("Base2_Links.txt","a")
    url=input("\nPlease Enter URL To Crawl : ")
    f.write("\n"+url+"\n")
    f.close
    threads=input("\nPlease Enter The Number Of Threads : ")
    css_selector=input("\nPlease Enter Your Css Selector : ")
    print("")
    css_tag=None
    tag_info=None
    crawling_links=queue.Queue()
    url_lock=threading.Lock()
    crawling_links.put(url)
    visited_links=set()
    crawling_threads=[]
    error_links=[]
    start_time=time.time()
    for i in range(int(threads)):

crawler=Crawler(url=url,css_tag=css_tag,tag_info=tag_info,css_selector
=css_selector,crawling_links=crawling_links,visited_links=visited_link
s,error_links=error_links,url_lock=url_lock)
        crawler.start()
        crawling_threads.append(crawler)
    for crawler in crawling_threads:
        crawler.join()
    end_time=time.time()-start_time
    visited=len(visited_links)
    errors=len(error_links)

print("\n=============================================================
===========================================================\n")
    print(f"URL : {url}")
    print(f"Total Threads : {threads}")
    print(f"Total Time Taken : {end_time}")
    print(f"Total Number Of Visited Pages : {visited}")
    print(f"Total Number Of Links With Errors : {errors}\n")

print("===============================================================
=========================================================\n")
    f=open("Base2_Links_With_Info.txt","r")
    print(f.read())
    f=open("Base2_Links_With_Info.txt","a")
    f.write("\nURL : "+url+"\nThreads : "+threads+"\nTime :
"+str(end_time)+"\nVisited Pages : "+str(visited)+"\nLinks With Errors
: "+str(errors)+"\n")
    f.close

print("===============================================================
=========================================================\n")
    print("Do You Want To Crawl Another Webpage(Y/N) : ")
    choice=input("\nPlease Enter Your Choice : ")
    print("")
    if(choice=='Y' or choice=='y'):
        Home()
    elif(choice=='N' or choice=='n'):
        exit
```

4.Threading Class:

In the threading class there are two methods the first is of __init__ which initializes all the passed/required parameters as self so that they can be used through out the class without calling them each time.

CODE:

```
class Crawler(threading.Thread):
    def 
__init__(self,url,css_tag,tag_info,css_selector,crawling_links,visited
_links,error_links,url_lock):
        threading.Thread.__init__(self)
        self.url=url
        self.css_tag=css_tag
        self.tag_info=tag_info
        self.css_selector=css_selector
        self.crawling_links=crawling_links
        self.visited_links=visited_links
        self.error_links=error_links
        self.url_lock=url_lock
```

5.Main Method:

This is the main method where all the crawling and all the main work of a multi-threaded web crawler will be done. In this firstly SSL is initialized we create a SSL context so that our script can crawl the https sties with SSL handshake. when creating a default SSL context and making an handshake we verify the hostname with the certificate but our objective is to crawl the webpage so we will not be checking the validity of the certificate, so the hostname is put False. We are not verifying the certificate and any certificate is accepted in this mode, so the certificate is none. Now an infinite while loop is created so that all the links can be gone through. Now a global lock on our queue is created so that no two threads can access the queue at the same time, the link variable will get the link from the queue and lock will be released, then a few if conditions are given the first one is that if the link variable gets noting then the loop will break and if the links is already crawled then also the loop will be broken then a try condition is given in which we join our given URL with the link that we got from crawling the URL because when we crawl a website and get links the links never come in proper form instead they are in a continued form such as
search?q=abc
And not
https://www.google.com/search?q=abc

So, to join and make it a proper link we use join.

Now a header is used it is so because some servers will only allow the connection if it comes from a verified browser and in this case we will be using a Firefox header.

After that we will be opening the URL using a SSL handshake. After that the BeautifulSoup(BS4) library is used which will return the source code/HTML representation of the webpage.

After that the condition will run according to the selection of Base1 or Base2 if the user chose Base1 then the CSS tag which the user chose will be selected and the loop will run until the BS4 library finds all the chosen CSS tags in the HTML page and if the CSS tag has the HREF or a string between the tags, depends upon the user choice, then the link will be put into the crawling links queue after the whole page is completed the crawled URL will go into the visited links list and if an exception occurred during crawling it will go into the error links list and crawling will keep going on until all the URL(s) in the crawling links queue is crawled and after that it will be completed and show us the result.

If the user chose Base2 instead of Base1 then all the things would be the same aside from one thing that the loop where CSS tag was selected the given CSS selector would be selected and the crawler will crawl through the webpage according to that CSS selector.

CODE:

```
def Main(self):
        my_ssl=ssl.create_default_context()
        my_ssl.check_hostname=False
        my_ssl.verify_mode=ssl.CERT_NONE
        while True:
            self.url_lock.acquire()
            link=self.crawling_links.get()
            self.url_lock.release()
            if link is None:
                break
            if link in self.visited_links:
                print(f"The URL {link} Is Visited")
                break
            try:
                link=urljoin(self.url,link)
                req=Request(link, headers={'User-
Agent':'Mozilla/5.0'})
                response=urlopen(req,context=my_ssl)
                print(f"The URL {response.geturl()} Crawled With
Status {response.getcode()}")
                soup=BeautifulSoup(response.read(),"html.parser")
                if self.css_selector is None:
                    for tag in soup.find_all(self.css_tag):
                        if (tag.get("href") not in
self.visited_links):
                            self.crawling_links.put(tag.get("href"))
                        else:
                            href=urljoin(self.url,tag.get("href"))
                            title=tag.string
                            if(self.tag_info=='href'):
                                print(f"\nThe URL {href} Is Visited")
                            elif(self.tag_info=='title'):
                                print(f"\nThe URL {title} Is Visited")
                else:
                    all_links=soup.select(self.css_selector)
                    for tag in all_links:
                        if (tag.get("href") not in
self.visited_links):
                            self.crawling_links.put(tag.get("href"))
                        else:
                            href=urljoin(self.url,tag.get("href"))
                            print(f"\nThe URL {href} Is Visited")
                print(f"\n{link} Is Added To The Crawled List")
                self.visited_links.add(link)
            except URLError as e:
                print(f"\nThe Given URL Threw Error : {e.reason} ")
                self.error_links.append(link)
            finally:
                self.crawling_links.task_done()
```

Libraries Used:

1.from urllib.request import Request, urlopen, URLError, urljoin
2.import time
3.import threading
4.import queue
5.from bs4 import BeautifulSoup
6.import ssl

1.This library is used so that we can join our URL with the crawled links, find links with errors, cover ourselves as a certified browser so that more URL(s) can be crawled.

2.This library is used to get time, record time, see time and other places where time would matter.

3.This library is used so that we can create threads and make the threads work on different sub processes of a process.

4.This library is used so a queue can be created and we use queue quite often in our code to keep our links that are to be crawled in a good and organized manner.

5.This library is one of the most important libraries to make a web crawler as it makes our work very easy to crawl a website however we want and find whatever we want.

6.This library is also one of the most important libraires as it allows us to go to many different websites which we would not be able to go as a simple web crawler as many websites deny a web crawler to work this allows us to disguise our program as a certified browser and help us crawl the website.