

Forum:[Hack Forums](#)

Thread: [Multithreading in Python](#)

Aexone ? - January 5, 2020, 9:39 am

Hello all,

I've coded a script that goes in a .txt file and for each line it make a request on a website and try to scrap something.

It works good but i've a lot of site to check so i want to use multithreading.

My script works like this :

```
def firstFunc():  
    with open(filename) as file:  
        for url in urls:  
            secondFunc(url) #this function make the requests, scrap and write on a file if ok
```

So i want to know how can I setup the multithreading. I tried but i got duplicate results, it means that instead of having a queue of url to pass to the different thread, they all do the same requests.

For example with 3 threads, instead of having :

```
google -> ok  
yahoo -> ok  
yandex -> ok
```

I have :

```
google -> ok  
google -> ok  
google -> ok  
yahoo -> ok  
yahoo -> ok  
yahoo ->ok
```

.....

Anyone can help me ? Thank you !

roosiinfidel09 - January 5, 2020, 10:02 am

start here.

<https://www.geeksforgeeks.org/multithreading-python-set-1/>

Invisible - January 5, 2020, 10:06 am

You can insert code with tags

```
Code def firstFunc():  
    with open(filename) as file:  
        for url in urls:  
            secondFunc(url) #this function make the requests, scrap and write on a file if ok
```

Aexone ? - January 5, 2020, 11:06 am

(Jan 5, 2020 11:02 AM)roosiinfidel09 Wrote: [\[CITE\]](#) start here.

<https://www.geeksforgeeks.org/multithreading-python-set-1/>

Same problem... i don't know why but the second thread don't continue with the other emails but redo what the first thread do

<https://prnt.sc/qjpok6>

roosiinfidel09 - January 5, 2020, 11:19 am

(Jan 5, 2020 12:06 PM)Aexone ? Wrote: [\[CITE\]](#) Same problem... i don't know why but the second thread don't continue with the other emails but redo what the first thread do

<https://prnt.sc/qjpok6>

My point was...You must be looking for your script specific solution. That is always a bad idea. Understand how multi-threading works...then try on your own...if you really can't then post the question. Also, make sure you put in words which make you appear knowledgeable about the topic. Lots of assholes here will give you -3 popularity without reading your question because you didn't include fancy words.

Nomad.Null - January 5, 2020, 8:15 pm

Sounds like the fundamental problem you're having is that you are not logically separating the list and distributing them to each thread (or more accurately, each process, since multi-threading still isn't a thing in python).

Easiest solution to your problem is to slice your list of urls equally into separate lists and pass them to each process. Better yet, you should be using a set to start with to ensure there are no duplicate urls. Using a set will automatically de-duplicate your list of urls.

Hope that helps.

Lost Soul - June 2, 2020, 4:39 am

(Jan 5, 2020 10:39 AM)Aexone ? Wrote: [\[CITE\]](#) Hello all,

I've coded a script that goes in a .txt file and for each line it make a request on a website and try to scrap something.

It works good but i've a lot of site to check so i want to use multithreading.

My script works like this :

```
def firstFunc():  
    with open(filename) as file:  
        for url in urls:  
            secondFunc(url) #this function make the requests, scrap and write on a file if ok
```

So i want to know how can I setup the multithreading. I tried but i got duplicate results, it means that instead of having a queue of url to pass to the different thread, they all do the same requests.

For example with 3 threads, instead of having :

```
google -> ok  
yahoo -> ok  
yandex -> ok
```

I have :
google -> ok
google -> ok
google -> ok
yahoo -> ok
yahoo -> ok
yahoo ->ok
.....

Anyone can help me ? Thank you !

I can help you, it is so easy. it will be ready in like 5 minutes haha. Pm me if u still interested.

vorticalbox - June 2, 2020, 5:34 am

I think async is better than threading because of the GIL and multi processing is memory intensive as each process gets the copy of the list.

here is an async script to get the HTML title from urls

PHP Code

```
<?php
import asyncio
import http3
from lxml.html import fromstring

client = http3.AsyncClient()

urls = [
    'https://google.co.uk',
    'https://youtube.co.uk',
    'https://duckduckgo.com'
]

async def get_title(url: str) -> str:
    r = await client.get(url)
    tree = fromstring(r.content)
    print(tree.findtext('.//title'))

loop = asyncio.get_event_loop()

tasks = asyncio.gather(*[get_title(url) for url in urls])
loop.run_until_complete(tasks)
```

synchronizing - June 2, 2020, 8:00 am

(Jun 2, 2020 05:34 AM)vorticalbox Wrote: [\[CITE\]](#) I think async is better than threading because of the GIL and multi processing is memory intensive as each process gets the copy of the list.

...

@OP, asyncio is definitely the better choice when it comes to scraping multiple websites. To explain further, I'm adapting an answer I wrote elsewhere unto here. @vorticalbox used http3, and I used aiohttp for sending GET requests asynchronously. It's your choice which you prefer, but both should generally work in the same manner.

You should look into asynchronous programming. Different from thread, asynchronous code runs in the same thread, but it runs inside an event loop. This event loop automatically switches context between different operations when the Python keyword `await` is present.

In other words, think of scraping websites as the following:

Code client sends request -> ... waiting for server reply ... <- server replies

Sending a request is an operation that takes a very small amount of time and consumes almost no resources. The real time consumer is waiting for the server to respond, and then processing the server's reply. If instead we do something that resembles the following:

Code client sends request -> switch operation -> ... wait ... <- server replies

client sends request -> switch operation -> ... wait ... <- server replies

client sends request -> switch operation -> ... wait ... <- server replies

...

Then we can minimize our time waiting for the server to reply, and instead already be shooting the next request over. In other words, what we can effectively do is tell Python to send the request, and then instantly switch to a different part of our code that sends another request, and then another part that sends another request, and so on. When all the requests are sent, we can come back and start interpreting the individual server replies.

There is a lot of references online on how to program asynchronously in Python (using the built-in `asyncio` module + PyPi installable `aiohttp` module), and I would suggest Googling away. Here is a code sample that will take less than 4 seconds to scrape over 100 websites (note that this scales extremely well, and 4 seconds is actually due to the print statements... without, it's actually closer to 2 seconds):

```
Code import asyncio
```

```
import aiohttp
```

```
import time
```

```
websites = ["https://www.youtube.com
```

```
https://www.facebook.com
```

```
https://www.baidu.com
```

```
https://www.yahoo.com
```

```
https://www.amazon.com
```

```
https://www.wikipedia.org
```

```
http://www.qq.com
```

```
https://www.google.co.in
```

```
https://www.twitter.com
```

```
https://www.live.com
```

```
http://www.taobao.com
```

```
https://www.bing.com
```

```
https://www.instagram.com
```

```
http://www.weibo.com
```

```
http://www.sina.com.cn
```

```
https://www.linkedin.com
```

```
http://www.yahoo.co.jp
```

```
http://www.msn.com
```

```
http://www.uol.com.br
```

<https://www.google.de>
<http://www.yandex.ru>
<http://www.hao123.com>
<https://www.google.co.uk>
<https://www.reddit.com>
<https://www.ebay.com>
<https://www.google.fr>
<https://www.t.co>
<http://www.tmall.com>
<http://www.google.com.br>
<https://www.360.cn>
<http://www.sohu.com>
<https://www.amazon.co.jp>
<http://www.pinterest.com>
<https://www.netflix.com>
<http://www.google.it>
<https://www.google.ru>
<https://www.microsoft.com>
<http://www.google.es>
<https://www.wordpress.com>
<http://www.gmw.cn>
<https://www.tumblr.com>
<http://www.paypal.com>
<http://www.blogspot.com>
<http://www.imgur.com>
<https://www.stackoverflow.com>
<https://www.aliexpress.com>
<https://www.naver.com>
<http://www.ok.ru>
<https://www.apple.com>
<http://www.github.com>
<http://www.chinadaily.com.cn>
<http://www.imdb.com>
<https://www.google.co.kr>
<http://www.fc2.com>
<http://www.jd.com>
<http://www.blogger.com>
<http://www.163.com>
<http://www.google.ca>
<https://www.whatsapp.com>
<https://www.amazon.in>
<http://www.office.com>
<http://www.tianya.cn>
<http://www.google.co.id>
<http://www.youku.com>
<https://www.example.com>
<http://www.craigslist.org>
<https://www.amazon.de>
<http://www.nicovideo.jp>
<https://www.google.pl>
<http://www.soso.com>
<http://www.bilibili.com>
<http://www.dropbox.com>

http://www.xinhuanet.com
http://www.outbrain.com
http://www.pixnet.net
http://www.alibaba.com
http://www.alipay.com
http://www.chrome.com
http://www.booking.com
http://www.googleusercontent.com
http://www.google.com.au
http://www.popads.net
http://www.cntv.cn
http://www.zhihu.com
https://www.amazon.co.uk
http://www.diply.com
http://www.coccoc.com
https://www.cnn.com
http://www.bbc.co.uk
https://www.twitch.tv
https://www.wikia.com
http://www.google.co.th
http://www.go.com
https://www.google.com.ph
http://www.doubleclick.net
http://www.onet.pl
http://www.googleadservices.com
http://www.accuweather.com
http://www.googleweblight.com
http://www.answers.yahoo.com""

```
async def get(url):
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(url=url) as response:
                resp = await response.read()
                print("Successfully got url {} with response of length {}".format(url, len(resp)))
    except Exception as e:
        print("Unable to get url {} due to {}".format(url, e.__class__))
```

```
async def main(urls, amount):
    ret = await asyncio.gather(*[get(url) for url in urls])
    print("Finalized all. ret is a list of len {} outputs.".format(len(ret)))
```

```
urls = websites.split("\n")
amount = len(urls)
```

```
start = time.time()
asyncio.run(main(urls, amount))
end = time.time()
```

```
print("Took {} seconds to pull {} websites.".format(end - start, amount))
```

Outputs:

Code Successfully got url <http://www.google.com.br> with response of length 12188.
Successfully got url <http://www.google.it> with response of length 12155.
Successfully got url <https://www.t.co> with response of length 0.
Successfully got url <http://www.msn.com> with response of length 46335.
Successfully got url <http://www.chinadaily.com.cn> with response of length 122053.
Successfully got url <https://www.google.co.in> with response of length 11557.
Successfully got url <https://www.google.de> with response of length 12135.
Successfully got url <https://www.facebook.com> with response of length 115258.
Successfully got url <http://www.gmw.cn> with response of length 120866.
Successfully got url <https://www.google.co.uk> with response of length 11540.
Successfully got url <https://www.google.fr> with response of length 12189.
Successfully got url <http://www.google.es> with response of length 12163.
A bunch of other output similar to the above. Couldn't include but HF has a limit on post length.

...

Finalized all. ret is a list of len 100 outputs.

Took 3.899034023284912 seconds to pull 100 websites.

As you can see 100 websites from across the world were successfully reached (with or without https) in about 4 seconds with aiohttp on my internet connection. Keep in mind the following can slow down the program by a few ms:

print statements (yes, including the ones placed in the code above).

Reaching servers further away from your geographical location.

The example above has both instances of the above, and therefore it is arguably the least-optimized way of doing what you have asked. However, I do believe it is a great start for what you are looking for.

Lindros - June 3, 2020, 2:15 am

You can look into using a Thread Pool Executor if you want a different method. It does all of the thread managing for you and is pretty easy to use.