

ASSIGNMENT REPORT: PROCESS AND THREAD IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

Ayhan YÜKSEK
ayhanyuksek@posta.mu.edu.tr

Sunday 3rd May, 2020

1 Introduction

In this assignment, we created a python script. The task of this script was to retrieve addresses from the address list sequentially and check if they were working. First of all, we ran this script we created normally and then ran it using the multithreading method. The purpose of these trials was to see in which case the script would run and complete faster. Thanks to this assignment, we will grasp in what situations the multithreading method is advantageous and what these advantages are .

2 Assignments

In this assignment, 4 different tasks were requested from us.These tasks are:

1-) Write a function in python script that checks if internet addresses are working.

– *Firstly, I import python request library to write this function. And using this library, I wrote the following code:*

```
#!/usr/bin/python3
import os, threading,requests

def url_arr(aList):

    r = requests.get(aList, auth=("user","pass"))
    r.status_code
    print("Status of url: ",r.status_code)
    if(r.status_code == 200):
        print("url is working.")
    else:
        print("url is not working.")
```

2-) Print PID of python script

– *In this task, we are asked for the pid (process id-number) of our script. We can retrieve this data using the python's "OS" library.And with os.getpid()*

```
import os, threading,requests
print("PID of script is: ",os.getpid())
```

3-) Print loadavg of functions.

–We can think of these values as CPU counters that show the process load. We can retrieve this data using the python's "OS" library.

```
avg = os.getloadavg()
print("loadavg is: ",avg)
```

4-) Take and print "5 min. Loadavg "value and the number of cpu cores and print. And compare the two values.

–In this task, we take the 5 min. loadavg value and compare it with the number of cores of the CPU. If the difference between them exceeds 1, the script ends. because it is undesirable to exceed 1. In this task, we can implement it with a code like the one below:

```
def url_arr(aList):

    avg = os.getloadavg()
    print("loadavg is: ",avg)

    cpu= os.cpu_count()
    print("5 min loadavg value and cpu core count are: ",avg[1],"and",cpu )
    if((cpu - avg[1])<1):
        exit()
```

3 Results

```
arr = ["https://api.github.com",
" http://bilgisayar.mu.edu.tr/",
"https://www.python.org/",
"http://akrepnalan.com/ceng2034",
"https://github.com/caesarsalad-wow"]

url_arr(arr[0])
url_arr(arr[1])
url_arr(arr[2])
url_arr(arr[3])
url_arr(arr[4])

arr = ["https://api.github.com",
" http://bilgisayar.mu.edu.tr/",
"https://www.python.org/",
"http://akrepnalan.com/ceng2034",
"https://github.com/caesarsalad-wow"]

thread1 = threading.Thread(target=url_arr, args=(arr[0],))
thread2 = threading.Thread(target=url_arr, args=(arr[1],))
thread3 = threading.Thread(target=url_arr, args=(arr[2],))
thread4 = threading.Thread(target=url_arr, args=(arr[3],))
thread5 = threading.Thread(target=url_arr, args=(arr[4],))

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
```

Figure 1: process implementation

Figure 2: Multithreaded implementation

We can run our function in the script in two ways. The first is to run our function once per element of our array, as we see in figure 1. The second is to create a thread for each element of our list and start our threads.

Although it may seem costly in two transactions at first glance, we can summarize the difference between them as follows.

– In Figure 1 will not switch to another transaction until one transaction is finished. For example, arr[1] will not work before the function ends for arr[0].

– But in figure 2, since we are doing multithreaded, one process can run in another process when it starts running. This is the advantage of multi-threaded. For example, when our function starts working for arr[0], while function is waiting for a response from arr[0], arr[1] starts working.

In order to better understand the difference between them, we need to examine the output of both situations. The important thing in the output is the time required for the script to finish. We can print the output time with the 'time' command.

```
PID of script is: 1849
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 403
url is not working.
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 200
url is working.
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 404
url is not working.
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 404
url is not working.
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 404
url is not working.
loadavg is: (0.87, 0.5, 0.22)
5 min loadavg value and cpu core count are: 0.5 and 2
Status of url: 404
url is not working.
real 0m2,184s
user 0m0,211s
sys 0m0,014s
```

```
loadavg is: (0.29, 0.24, 0.11)
5 min loadavg value and cpu core count are: 0.24 and 2
Status of url: 404
url is not working.
Status of url: 200
url is working.
Status of url: 404
url is not working.
Status of url: 403
url is not working.
Status of url: 404
url is not working.
Status of url: 404
url is not working.
real 0m0,638s
user 0m0,192s
sys 0m0,030s
```

Figure 4: Multithreaded implementation

Figure 3: process implementation

As you can see in the diagrams above, real time in figure 3 is 2.184s and real time in figure 4 is 0.638s.

4 Conclusion

As a result, we have a better understanding of the advantages of multithreaded processing while performing the tasks given in this assignment. When we run our function as in figure 1, we found that the function works for the first element of the list and goes to the second process after it is finished. This can be extremely costly. Likewise, thanks to the multithreaded application in figure 2, we were able to run our function faster. So why? This is due to the feature of the thread. According to this feature, when first thread starts working, while first thread is waiting for response from function, another thread starts working.

In addition, the fact that multithreaded works faster than normal showed us that our script is not a cpu bound code. If more than one thread uses a common variable, access control of this variable is provided by Global Interpreter Lock. GIL causes only one to work at any given time, no matter how many threads there are. So it might not make sense to make multithreaded in cpu bound codes.

GITHUB ACCOUNT:

<https://github.com/ayhanyuksek>