ASSIGNMENT REPORT 1: PROCESS AND THREAD IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

Mehmet Cihan Sakman
mehmetcihansakman@posta.mu.edu.tr
github username: ctyp55

Friday 1st May, 2020

1 Introduction

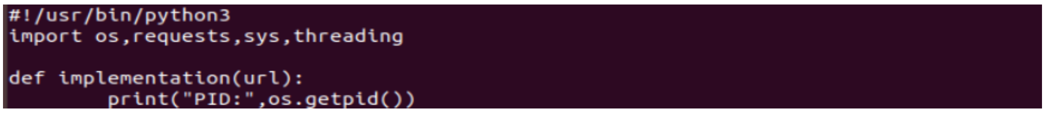
The purpose of this project is that understand how threads are process. This lab is crucial to see what are the benefits of using threads and how threads are processing.

2 Assignments

Following subsections explains how different tasks solved step by step.

2.1 Process ID of Threads (PID of Threads)

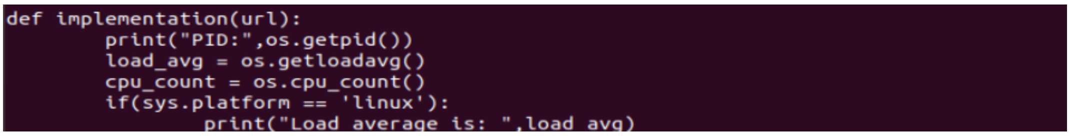
Using os.getpid() command from os library we get the process ID of each process to see whether different threads are using same process ID or not.



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

def implementation(url):
        print("PID:",os.getpid())
```

2.2 If the running OS is linux; print loadavg

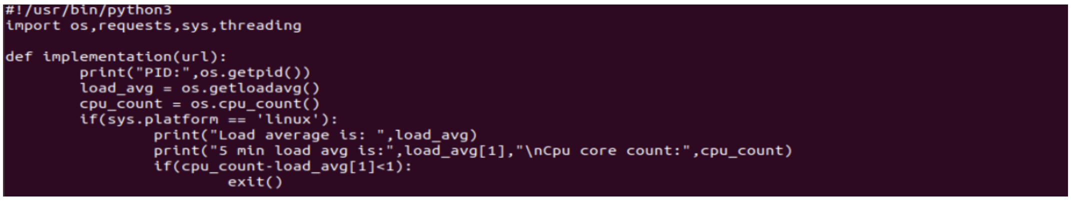
To ensure that the user is using Linux it checked by following codes in photo below. Using sys.platform command we get the information about which OS is using by user. If the user using linux we print the loadavg information with using os.getloadavg() command.



```
def implementation(url):
        print("PID:",os.getpid())
        load_avg = os.getloadavg()
        cpu_count = os.cpu_count()
        if(sys.platform == 'linux'):
                print("Load average is: ",load_avg)
```

2.3 Is there any waiting process?

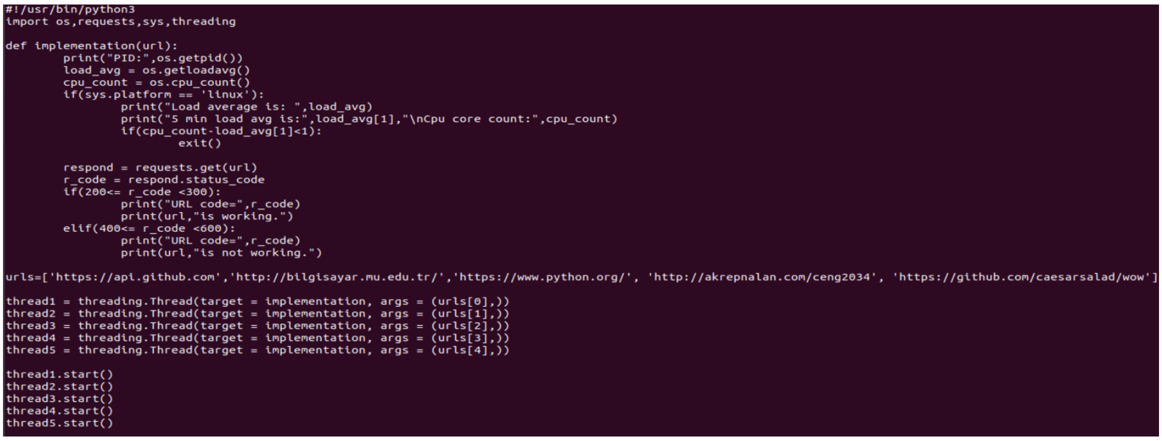
At this stage we checked if there any other process which is waiting for running. If there is another process to wait for run we just exit the script. To understand that there is an waiting process or not we checked our load averages for 5 minutes and CPU core count. To figure it out we get the difference between CPU core count and load averages for 5 minutes, if it is less than 1 we just exit the script. To get CPU core count we used os.cpucount() command and to get load average for 5 minutes we just get the second element of os.getloadavg(). You can see the codes in below photo.



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

def implementation(url):
        print("PID:",os.getpid())
        load_avg = os.getloadavg()
        cpu_count = os.cpu_count()
        if(sys.platform == 'linux'):
                print("Load average is: ",load_avg)
                print("5 min load avg is:",load_avg[1],"\nCpu core count:",cpu_count)
                if(cpu_count-load_avg[1]<1):
                        exit()
```

2.4 Threads Processing

At this stage of the project we have five different URL to check whether these URLs are valid or not. To check it we send the URLs separately by using five threads. First of all we get the URL by using requests.get() command from requests library. After that we check the response code by using statuscode() command. If the URL's response status something like 2XX this code is valid otherwise not valid. You can see the codes in below photo.



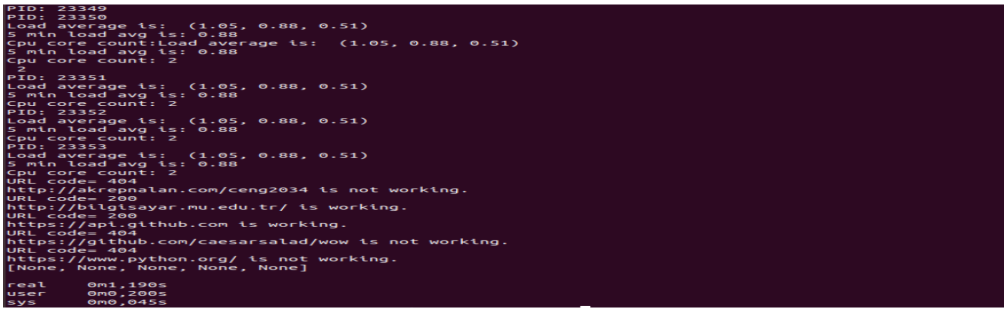
```
#!/usr/bin/python3
import os,requests,sys,threading

def implementation(url):
        print("PID:",os.getpid())
        load_avg = os.getloadavg()
        cpu_count = os.cpu_count()
        if(sys.platform == 'linux'):
                print("Load average is: ",load_avg)
                print("5 min load avg is:",load_avg[1],"\nCpu core count:",cpu_count)
                if(cpu_count-load_avg[1]<1):
                        exit()

        respond = requests.get(url)
        r_code = respond.status_code
        if(200<= r_code <300):
                print("URL code=",r_code)
                print(url,"is working.")
        elif(400<= r_code <600):
                print("URL code=",r_code)
                print(url,"is not working.")

urls=['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 = implementation, args = (urls[0],))
thread2 = threading.Thread(target = implementation, args = (urls[1],))
thread3 = threading.Thread(target = implementation, args = (urls[2],))
thread4 = threading.Thread(target = implementation, args = (urls[3],))
thread5 = threading.Thread(target = implementation, args = (urls[4],))

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

3 Results

We will handle results step by step.

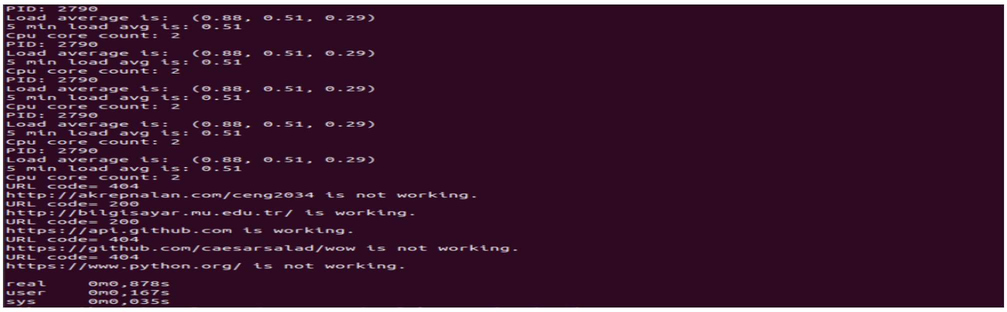
3.1 Process ID's

Result from assignment 2.1 we will compare Multithreading (Figure2) and Multiprocessing (Figure1). In Figure1 and Figure2 we can see the process ID's of these scripts.



```
PID:  23349
PID:  23350
Load average is:   (1.05, 0.88, 0.51)
5 min load avg is: 0.88
Cpu core count:Load average is:   (1.05, 0.88, 0.51)
5 min load avg is: 0.88
Cpu core count: 2
 2
PID:  23351
Load average is:   (1.05, 0.88, 0.51)
5 min load avg is: 0.88
Cpu core count: 2
PID:  23352
Load average is:   (1.05, 0.88, 0.51)
5 min load avg is: 0.88
Cpu core count: 2
PID:  23353
Load average is:   (1.05, 0.88, 0.51)
5 min load avg is: 0.88
Cpu core count: 2
URL code= 404
http://akrepnalan.com/ceng2034 is not working.
URL code= 200
http://bilgisayar.mu.edu.tr/ is working.
URL code= 200
https://api.github.com is working.
URL code= 404
https://github.com/caesarsalad/wow is not working.
URL code= 404
https://www.python.org/ is not working.
[None, None, None, None, None]

real	0m1,190s
user	0m0,200s
sys	0m0,045s
```

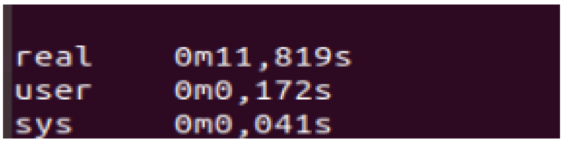
Figure 1: Multiprocessing



```
PID:  2790
Load average is:   (0.88, 0.51, 0.29)
5 min load avg is: 0.51
Cpu core count: 2
PID:  2790
Load average is:   (0.88, 0.51, 0.29)
5 min load avg is: 0.51
Cpu core count: 2
PID:  2790
Load average is:   (0.88, 0.51, 0.29)
5 min load avg is: 0.51
Cpu core count: 2
PID:  2790
Load average is:   (0.88, 0.51, 0.29)
5 min load avg is: 0.51
Cpu core count: 2
PID:  2790
Load average is:   (0.88, 0.51, 0.29)
5 min load avg is: 0.51
Cpu core count: 2
URL code= 404
http://akrepnalan.com/ceng2034 is not working.
URL code= 200
http://bilgisayar.mu.edu.tr/ is working.
URL code= 200
https://api.github.com is working.
URL code= 404
https://github.com/caesarsalad/wow is not working.
URL code= 404
https://www.python.org/ is not working.

real	0m0,878s
user	0m0,167s
sys	0m0,035s
```

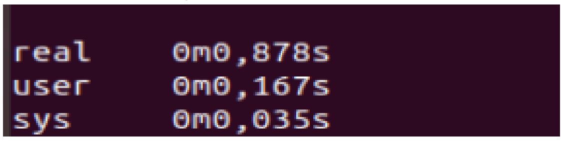
Figure 2: Multithreading

3.2 Time Saving



```
real	0m11,819s
user	0m0,172s
sys	0m0,041s
```

Figure 3: Normal Time



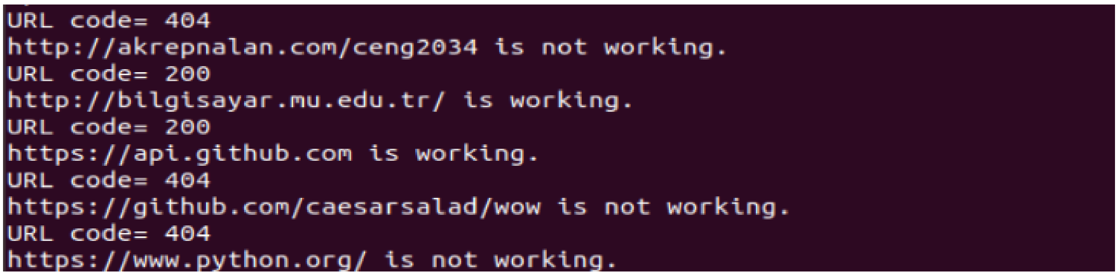
```
real	0m0,878s
user	0m0,167s
sys	0m0,035s
```

Figure 4: Multithread Time

In Figure3: Normal process time is 11.8second, and in Figure4: Multithread processing time is 0.8second. This project clearly showed us that Multithread processing is much faster than normal processing.

3.3 URL Response Code

Another result from assignment 2.4 showing us that we can check any website whether it is available or not with checking their response code. If the response code like 2XX it means website is available otherwise it is not available. We can clearly see that in Figure5.



```
URL code= 404
http://akrepnalan.com/ceng2034 is not working.
URL code= 200
http://bilgisayar.mu.edu.tr/ is working.
URL code= 200
https://api.github.com is working.
URL code= 404
https://github.com/caesarsalad/wow is not working.
URL code= 404
https://www.python.org/ is not working.
```

Figure 5: Response Codes for URLs

4 Conclusion

The purpose of the having this project was showing the positive effect of using threads. As we discuss before in Results section it is obvious that the Multithreading process is much faster than Normal process(without threading) in this project. If we check assignment 2.4 while we getting URL we are waiting for response from target site. At this time our threads starting to get another URL's response and it goes like that and we save our running time for this project.

Result from Results3.1 showing that in Multithreading, threads are using same memory therefore their process ID are same. On the other hand in Multiprocessing each core uses their own memory and therefore their process ID are different.