IMPLEMENTATION OF FORK AND MULTIPROCESSING SYNCHRONIZATION

CENG2034, OPERATING SYSTEMS

Ayhan YÜKSEK
ayhanyuksek@posta.mu.edu.tr

Thursday 4th June, 2020

Abstract

The purpose of this homework is to learn how to write the codes that we can write in linux or windows terminal in python scripts and use fork() method. I used the os,uuid,requests,hashlib and multiprocessing libraries to this homework. I have experienced what multitasking is used for thanks to this assignment. In addition, I experienced creating child process and getting rid of orphan status

1 Introduction

In this assignment, we created a python script. Just like midterm homework. The task of this script is that we take the addresses from our address list and download the .jpeg files in them to our computer. It also allows them to be detected if more than one of our downloaded files has landed. First of all, we ran this script we created normally and then ran it using the multiprocessing method. The purpose of these trials was to see in which case the script would run and complete faster.

2 Assignments

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

1-) Create a new child process with syscall and print its PID.

– First of all, I import python os library to write this function. And using this library I wrote the following code:

```
def parent_child(url):
    child_proc = os.fork()  print("child pid is: ",os.getpid())
```

2-) With the child process, download the files via the given URL list.

– *We did this task using python's requests library and wrote the code below:*

```
def download_file(url,file_name = None):
    r = requests.get(url, allow_redirects = True)
    file = file_name if file_name else str(uuid.uuid4())
    open(file, 'wb').write(r.content)
```

3-) How can you avoid the orphan process situation with syscall?

–*The orphan process state is when the parent ends before the child. To avoid this situation, we can use the os.wait() command:*

```
os.wait()
print("parent pid is: ",os.getpid())
```

4-) Control duplicate files within the downloaded files of your python code.

–*In this task, we need to check the content of the downloaded files. We can use -md5/sha256- for this. We used the md5 method. We got the hash codes of the files we downloaded and checked the duplitace status.*

```
def find_duplicate(name):
    hashcode=["c8ac40dc6b37096d61c34c9a50a794b5",
              "7ed4550abfccb9470f03ba3b0200a05a",
              "3dcaee2bca739460bd30bb257785b107",
              "c8ac40dc6b37096d61c34c9a50a794b5",
              "3dcaee2bca739460bd30bb257785b107"]
    stat=hashlib.md5(open(name,'rb').read()).hexdigest()
    if hashcode.count(stat)==2:
        print(name,"is duplicate")
    else:
        print(name,"is uniq")
```

3 Results

As seen in figure 1, first of all, we have done our download tasks as it is called in our Child process. Meanwhile, in figure 2, our parent process was put on hold with the command *os.wait ()*. Thus, the situation of orphan situation was avoided.

```
def parent_child(url):
    child_proc = os.fork()
    if child_proc > 0:
        os.wait()

        print("parent pid is: ",os.getpid())
        os._exit(os.EX_OK)
    else:
        print("child pid is: ",os.getpid())
        j=0

        for i in url:
            download_file(i,name[j])
            j = j + 1
```

Figure 1: Create child process

```
def parent_child(url):
    child_proc = os.fork()
    if child_proc > 0:


        print("parent pid is: ",os.getpid())

        os.wait()
```

Figure 2: Wait command

–In this section, we need to examine the fork process so that we can better understand the child process. *os.fork()* creates two processes: A parent process and a child process.This is the copy of the child process parent that occurs. In figure 3, after running our script , we have seen that separate ”PIDs” are created for child and parent.

```
child pid is:  1724
parent pid is:  1720
```

Figure 3: Child and parent process

—- *Our output showed that our fork command was working successfully.* —-

In our homework, we were asked to examine the duplicate status of the files we downloaded. For this task we used the function in figure 4

```
def find_duplicate(name):
    hashcode=["c8ac40dc6b37096d61c34c9a50a794b5",
              "7ed4550abfccb9470f03ba3b0200a05a",
              "3dcaee2bca739460bd30bb257785b107",
              "c8ac40dc6b37096d61c34c9a50a794b5",
              "3dcaee2bca739460bd30bb257785b107"]
    stat=hashlib.md5(open(name,'rb').read()).hexdigest()
    if hashcode.count(stat)==2:
        print(name,"is duplicate")
    else:
        print(name,"is uniq")
```

Figure 4: Duplicate function

– We tried 2 ways to run this function. First of all, we called in the normal way, once for each element, second we tried the multiprocessing way.

```
find_duplicate(name[0])
find_duplicate(name[1])
find_duplicate(name[2])
find_duplicate(name[3])
find_duplicate(name[4])
```

Figure 5: Normal implementation

```
with Pool(5) as p:
    print(p.map(find_duplicate, [name[0],name[1],name[2],name[3],name[4]]))
```

Figure 6: Multiprocessing implementation

– After running these methods, we checked the running times with the time command. Our expectation was to see a certain difference in duration between the two ways. But they worked very close to each other in terms of time. Because of this situation, we could not decide which way is more reasonable to use.

```
real    0m2,629s
user    0m0,260s
sys     0m0,030s
```

Figure 7: Multiprocessing time

```
real    0m2,517s
user    0m0,276s
sys     0m0,187s
```

Figure 8: Normal way time

—– *The times are very close to each other as seen in figures 7 and 8*

4 Conclusion

As a result, we tried to see how the child process is created and the advantages of using multiprocessing while performing the tasks given in this assignment. When we create a child process as in figure 1, our parent process first worked and finished. This is undesirable because our child process we have created is now out of our control. In order to solve this situation, we forced our parent process to wait for the child process.
After downloading the files, we had to work on it. We used the multiprocessing method, which we think would be advantageous for these operations. But we couldn't provide the advantage we expected with multiprocessing. In this case, the most important question to be asked is this. What could be the reason for this? The purpose of using multiprocessing is to get the jobs that require a lot of processor load by using all the possibilities of the processor. So we can do a big job faster by dividing it into multiple processes. In this case, we think that the task we want to do is a low cost task and it happens very quickly. As evidence for this, we can show the output times in Figures 7 and 8. Maybe that's why we couldn't see the difference in the way we wanted. To summarize, I think that as a result of the tests I did at the end of this assignment, large workloads are required to see the difference of multiprocess.

GITHUB ACCOUNT:

https://github.com/ayhanyuksek