ASSIGNMENT REPORT 2: IMPLEMENTATION OF FORK AND MULTIPROCESSING SYNCHRONIZATION

CENG2034, OPERATING SYSTEMS

Ümit Kadiroğlu
umitkadiroglu@posta.mu.edu.tr

Sunday 7th June, 2020

Abstract

The purpose of this assignment is to get used to with child and parent processes, avoiding the orphan process situation with os.wait(), and also dealing with some duplicate checking jobs with hashlib MD5 and multiprocessing. In addition requests and uuid libraries was used. Also an interesting result was encountered in the end

1 Introduction

This assignment aims to practice the using of **fork()** system call, to know what fork() creates, to find how the orphan process situation can be prevented and the methods of detecting the duplicate files. This homework was done by using Linux Mint 19.3 Tricia, Python 3.6.9 and **os**, **requests**, **uuid**, **hashlib**, **multiprocessing** libraries.

Github: **https://github.com/umitkadiroglu.**

2 Tasks

There were 4 tasks in total. The fourth task was a bit more challenging than the others so it contains some subsections.

2.1 Task 1 (”Child Process”)

I started this task by importing **os** library. After creating the process, I put a condition to detect the child process and then used **getpid()** to to get its PID.

```python
import os

child = os.fork()

if child == 0:
    print("PID of child process: ", os.getpid())
```

Figure 1: Creating a child process

2.2 Task 2 (”Downloading the files”)

In this task, I imported **requests** and **uuid** libraries for the function called ”download file” whose job is to download files. I made an array called ”url” for URL's to download. Used the function under the child process and gave names to the files. Also added **os.exit()** for other tasks to not have problems.

```python
import os, requests, uuid,

url = ["http://wiki.netseclab.mu.edu.tr/images/thumb/f/f7/MSKU-BlockchainResearchGroup.jpeg/300px-MSKU-
BlockchainResearchGroup.jpeg","https://upload.wikimedia.org/wikipedia/tr/9/98/
Mu%C4%9Fla_S%C4%B1tk%C4%B1_Ko%C3%A7man_%C3%9Cniversitesi_logo.png","https://upload.wikimedia.org/
wikipedia/commons/thumb/c/c3/Hawai%27i.jpg/1024px-Hawai%27i.jpg","http://wiki.netseclab.mu.edu.tr/images/
thumb/f/f7/MSKU-BlockchainResearchGroup.jpeg/300px-MSKU-BlockchainResearchGroup.jpeg","https://
upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Hawai%27i.jpg/1024px-Hawai%27i.jpg"]

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)

child = os.fork()

if child == 0:
    print("PID of child process: ", os.getpid())
    download_file(url[0],"file01")
    download_file(url[1],"file02")
    download_file(url[2],"file03")
    download_file(url[3],"file04")
    download_file(url[4],"file05")
    os._exit(0)
```

Figure 2: Downloading files in child process

2.3 Task 3 (”Orphan Process”)

This time the problem was avoiding the orphan process situation. I added **wait()** for this, after the child but before the parent because I wanted my parent process to wait child to complete.

```
if child == 0:
    print("PID of child process: ", os.getpid())
    download_file(url[0],"file01")
    download_file(url[1],"file02")
    download_file(url[2],"file03")
    download_file(url[3],"file04")
    download_file(url[4],"file05")
    os._exit(0)

os.wait()
```

Figure 3: Using wait() for avoiding orphan process situation

2.4 Task 4.1 (”First Attempts”)

After I decided to use **hashlib -md5** for this task, I started by importing hashlib and multiprocessing libraries. Firstly, I created an array called ”files” to keep file names and created an empty array for hash codes. The mechanism of the ”getFileHash” function was getting a file as input, get its hash code and assign it to x, checking whether the checksum array contains this hash code and then append this hash to the checksum array.

```
files = ["file01", "file02", "file03", "file04", "file05"]
checksum = []
def getFileHash(file):
    x = hashlib.md5(open(file,'rb').read()).hexdigest()
    checksum.append(x)
    if x in checksum:
        print("File", file, "has a duplicate.")
    checksum.append(x)

with Pool(5) as p:
    p.map(getFileHash, ["file01", "file02", "file03", "file04", "file05"])

"""
getFileHash(files[0])
getFileHash(files[1])
getFileHash(files[2])
getFileHash(files[3])
getFileHash(files[4])
"""
```

Figure 4: The failed function

This made sense on paper but the result was not. It was working without multiprocessing. I assume that the reason it didn’t work with multiprocessing was the **2 cores** I have working with their own memories and what is in one’s array is not in the other’s. So, there were things that had to change.

2.5 Task 4.2 (”Change of plan”)

This time the checksum array was not empty. I filled that array with hash codes before comparing. The mechanism of the function changed to get its hash code to x, then append it to the array. This way, if there is not a duplicate, there will be 2 same hash codes; but if there is a duplicate, there will be 3. So I put a condition to check the number of this and printed positive if it is bigger than 2.

```
files = ["file01", "file02", "file03", "file04", "file05"]

checksum = [hashlib.md5(open("file01",'rb').read()).hexdigest(),
hashlib.md5(open("file02",'rb').read()).hexdigest(), hashlib.md5(open("file03",'rb').read()).hexdigest(),
hashlib.md5(open("file04",'rb').read()).hexdigest(), hashlib.md5(open("file05",'rb').read()).hexdigest()]




def getFileHash(file):
    x = hashlib.md5(open(file,'rb').read()).hexdigest()
    checksum.append(x)
    if checksum.count(x) > 2:
        print("File", file, "has a duplicate.")


with Pool(5) as p:
    p.map(getFileHash, ["file01", "file02", "file03", "file04", "file05"])
```

Figure 5: Changing the array and the function

3 Results

The results were quite surprising. Executing with multiprocessing had **increased the time by approximately 0.2 seconds.**

```
linuxmint@linuxmint:~/final$ time ./final.py
PID of child process:  5411
File file01 has a duplicate.
File file04 has a duplicate.
File file05 has a duplicate.
File file03 has a duplicate.

real    0m2,482s
user    0m0,292s
sys     0m0,077s
```

Figure 6: Output with multiprocessing

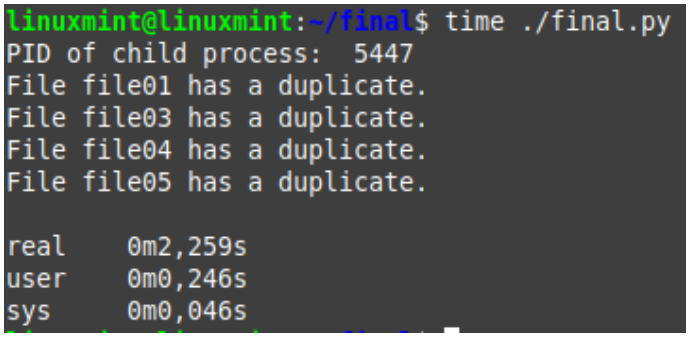

Figure 7: Output without multiprocessing

4 Conclusion

As a result, I learned that multiprocessing does not always speed-up the jobs, as seen at figure 6 and 7. The expected result was different than this one. Orphan process situation is dangerous and can be solved with os.wait() as seen at figure 3. The important thing is the place of wait(). I put it just after the child and before the parent process so it waits child process to complete and then executes the parent. Also, the hashlib library is quite useful and MD5 is calculated based on file contents.