ASSIGNMENT REPORT 2: MULTIPROCESS IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

Çilem Emre
cilememre10@gmail.com

Thursday 4th June, 2020

Abstract

The basis of learning the operating system begins with learning the processes. In this assignment, I produced parent and child processes and looked at their pid. I downloaded some files with the function using child process from a given url list. I used the uuid library for this. I avoid the orphan process with the waitpid command of the os library and checked which files are duplicate using the hashlib library. With this assignment, we learned new process types and made many comparisons. This assignment was a ladder to grasp the processes correctly.

1 Introduction

This task was very important because we saw many types of processes, compared them and learned how to use them. We grasped the working logic of multiprocesses and used different libraries. Processes form the basis of the operating system and this application has contributed a lot to my understanding of processes.

2 Assignments

The steps I did in this assignment were briefly to first create a new parent and child process and print their pid. Then I downloaded the files in the url provided with the child process I created. I tried to avoid the Orphan process situation and I wrote a function that found duplicate among the files I downloaded and did it using the multiprocess technique. I clearly explained how I handle these steps below.

2.1 Assignment: Python script with processes

What I used in this project:
CPU Features: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
Core and Thread Count: 4
OS: Linux Ubuntu 18.04.1 LTS
Programming Language: Pyhton 3.6

Problems

2.1.1. Creating a new child process with syscall and print its PID

In this part, I created a parent and child process using the ”os” module. I used the ”fork” command. When the fork command was called, two copies were created from the same program. Since I know that pid is parent process when it is 1, child process is 0 so I found the child process with the if else condition and finally I printed its pid.

```
def child_process(url):

        pid = os.fork()

        if pid > 0:
                print("Parent process and pid is : ", os.getpid())
                #avoid the orphan process
                os.waitpid(pid, 0)

        else:
                print("Child process and pid is : ", os.getpid())
```

2.1.2. With the child process, downloading the files via the given URL list

There was a list of urls given to us. I wrote a function that downloads files using this url list argument. I used the ”os”, ”uuid” and ”requests” modules for this function. After checking the child process, I called the function here. It downloaded each url. Continuing under the child process, I made this process with the child process. Then I created a new list of downloaded files and printed them out.

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

```
else:
        print("Child process and pid is : ", os.getpid())

        #with the child process, download the files
        for i in range(5):
                download_file(url[i],"file{}".format(i))

        download_files=os.listdir()
        download_files.remove("final.py")
        print("Download files: ",download_files)
```

2.1.3. Avoiding the orphan process situation

Orphan process is the situation when the parent finishes before child. For this part, I wrote os.waitpid(pid, 0) in the if block where there is a parent process. When I write this, parent process waits for child proces so I don’t have the orphan process situation.

2.1.4. Controlling the duplicate files within the downloaded files

In this section, firstly, I wrote a function that finds the duplicate files. This function checks each document in the downloaded file and adds it with the ”insert” command if it’s the second time from the same document. In this part, I used the multiprocess technique with the pool object in the if part. Each element in the file was checked with the md5 function I found on the internet and loaded into the a list. I used the hashlib library for the md5 function. The elements registered in the a list were checked with the multiprocess technique and the duplicate ones were added to a new list2 list. I recently removed the non-unique ones and printed these duplicate files on the screen. I added the code scripts below.

```
def find_duplicates(elem,a):

        print("find_duplicate processes pid:", os.getpid())

        j=0
        elements=[]

        for i in range(5):
                #it checks the duplicate elements
                if a[i] == elem:
                        elements.insert(j,i)
                        j +=1

                        if j > 1:
                                return elements

#I found this function on the internet
def control_md5(fname):
    hash_md5 = hashlib.md5()
    with open(fname, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()
```

```
a=[]


with Pool(5) as p:
        a = p.map(control_md5,download_files)


#find duplicate elements with multiprocessing
with Pool(5) as p:
        duplicate = p.starmap(find_duplicates,( [a[0], a],[a[1], a],
                [a[2], a], [a[3], a], [a[4], a]))


dup_list=list(duplicate)
list2=[]

#remove the not unique elements and None in dup_list
for i in dup_list:
        if i==None:
                dup_list.remove(None)
        elif i not in list2:
                list2.append(i)

print("Duplicate files: ",list2)
```

Notes

First, I printed the CPU and load avg list on the machine I used like this:

```
print("There are %d CPUs on this machine" % multiprocessing.cpu_count())
print("Load avg:", os.getloadavg())
```

I added a line of code into the function where I downloaded the files like this:

```
print("download_file processes pid:",os.getpid())
```

This line of code proved that I had downloaded my file with the child process. Their pid were equal.

In addition, I added a line of code into to my function that finds duplicate files. This proved that I used the multiprocess technique while doing this. Their pid are different from each other.

```
print("find_duplicate processes pid:", os.getpid())
```

Initially, I wrote a command that found start time and added a line of code that shows how many seconds all of these operations.

```
print ("Time taken =",time.time() - start_time,"seconds")
```

3 Results

I got this output showing parent and child pids. It shows a list of downloaded files, the pids of the processes in the functions, duplicate files list and taken time.

```
There are 4 CPUs on this machine
Load avg: (2.9, 1.53, 1.19)
Parent process and pid is :  7845
Child process and pid is :  7849
download_file processes pid: 7849
download_file processes pid: 7849
download_file processes pid: 7849
download_file processes pid: 7849
download_file processes pid: 7849
Download files:  ['file2', 'file1', 'file0', 'file4', 'file3']
find_duplicate processes pid: 7865
find_duplicate processes pid: 7866
find_duplicate processes pid: 7867
find_duplicate processes pid: 7865
find_duplicate processes pid: 7866
Duplicate files:  [[0, 3], [2, 4]]
Time taken = 4.370737314224243 seconds
```

4 Conclusion

I learned a lot with this task. One of the first things I learned; in multiprocessing, multiple processes or jobs can be run and managed by the CPU or a single program. Python's multiprocessing library has a number of powerful process spawning features which completely side-step issues.

There are two possibilities for the return value after the fork () function is called. It will be either equal to or greater than 0. Negative return only means that the function is incorrect. If it is positive, the return value for the child process is 0, while the return value for the parent process is 1. When you do a fork(), it copies everything in memory. That includes any globals you've set in imported Python modules.

I used the multiprocess technique with the pool object because pool is most useful for large amounts of processes where each process can execute quickly, while process is most useful for a small number of processes where each process would take a longer time to execute. To use the Pool class, we have to create a separate function that takes a list item as an argument like we did when using process.

With this task, I learned what the orphan process is and how it is avoided. Every running process job ends on its own. However, when the parent process creates a child process, if it is not desired to end before the child process, that is, if we want to avoid the orphan process, the child process is expected to terminate first using the waitpid () system call.

When I ran the codes, I noticed that the pids were written in a different and different order. There's no guarantee that the first process to be created will be the first to start or complete. As a result, multiprocessed code usually executes in a different order each time it is run, even if each result is always the same.

In addition, I learned the contents of different libraries such as hashlib and uuid.

This homework was very useful and made me understand how processes work. We cannot grasp the operating system without learning the processes so it was a great chance to do such a task to learn the processes and operating system in the technology age.

5 My github username

cilememre24