ASSIGNMENT REPORT 2: MULTI PROCESSING IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

Furkan Baldır
furkanbaldir13@gmail.com

Monday 1st June, 2020

Abstract

Today, this is the world of computers. Almost everything connected with computers. That's why humanity needs people who can speak the language of computers. These people are programmers. However, if a programmers want to be a good programmer, they need to understand how operating system works. They need to understand creating child process in the operating system. Understanding the multi processing is important to understand the computer.

Github Page

`https://github.com/furkanbaldir/ceng_2034_2020_final`

1 Introduction

In this homework, our goal is learning and distinguishing how parent and child processes work. Also, the other goal is understanding how multiprocessing works.

2 Assignments

2.1 What I used in project

CPU Features

Model Name: AMD A8-7410 APU with AMD Radeon R5 Graphics
Core Count: 4
Thread Count: 4

Operating System

Ubuntu 20.04 LTS Budgie Environment

Kernel Version

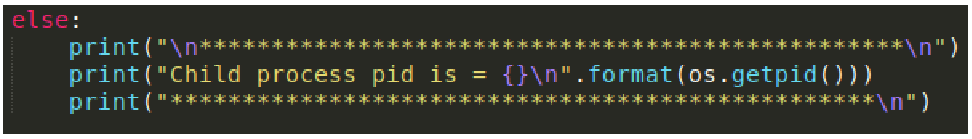
Linux 5.4.0-31-generic x86_64

Programming Language

Python 3.8.2 (Imported modules: os, uuid, requests, multiprocessing, hashlib, time)

2.2 Problems

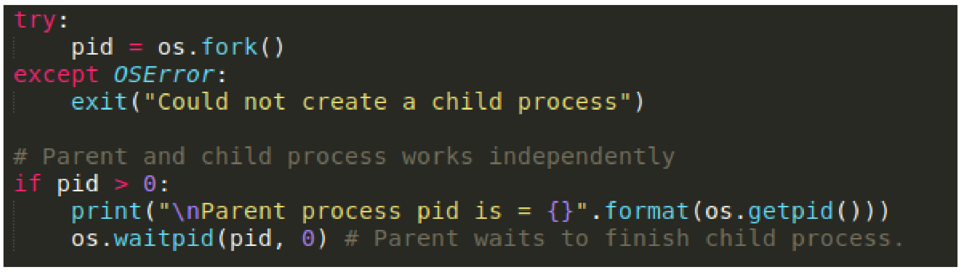
2.2.1 Create and Print Child PID



```
else:
    print("\n**************************************************\n")
    print("Child process pid is = {}\n".format(os.getpid()))
    print("**************************************************\n")
```

To solve this problem; I used *os* module. Firstly, I create child process with $os.fork()$. After that when return value is 0, $os.getpid()$ will return child PID.

2.2.2 Avoiding the Orphan Process Problem

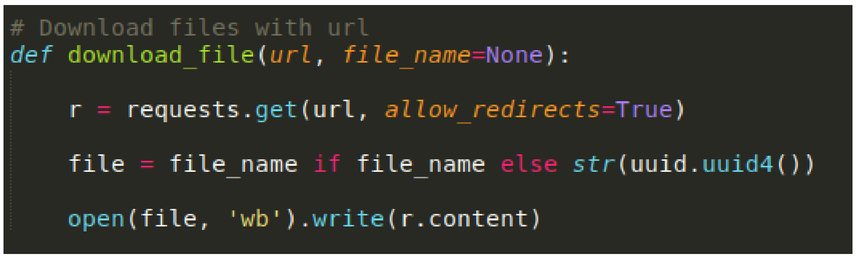


```
try:
    pid = os.fork()
except OSError:
    exit("Could not create a child process")

# Parent and child process works independently
if pid > 0:
    print("\nParent process pid is = {}".format(os.getpid()))
    os.waitpid(pid, 0) # Parent waits to finish child process.
```

To solve this problem; I used if-else statement to compare $os.fork()$ return value. If this value greter than 0, it is parent process; else, it is child process. That's why when value greater than 0, I used $os.waitpid(pid, 0)$ to wait parent process.

2.2.3 Download Files from URL List



```
# Download files with url
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)
```

To solve this problem; I used *uuid* and *requests* module, and $download\_file()$ function. I downlaoded every URLs one by one.

2.2.4 Control Duplicate Files with Multiprocessing

```python
# Return checksum md5 values
def 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()
```

```python
# Return duplicate indexes in list
def check_duplicate(element, list_md5):

    duplicate_count = 0
    indexes = []

    for i in range(len(list_md5)):
        if list_md5[i] == element:
            duplicate_count += 1
            indexes.append(i)

    if duplicate_count > 1:

        #print(indexes)
        return indexes
```

```python
# Return new list with unique members
def unique(list1):

    # init a null list
    unique_list = []

    # traverse for all elements
    for x in list1:
        if x == None:
            list1.remove(None)
        elif x not in unique_list:
            unique_list.append(x)

    return unique_list
```

```python
# Creating a list_duplicate list with using multiprocessing
with Pool(5) as p:
    list_duplicate = p.starmap(check_duplicate,( [list_md5[0],
     list_md5],[list_md5[1], list_md5],
     [list_md5[2], list_md5], [list_md5[3],
     list_md5], [list_md5[4], list_md5]))
```

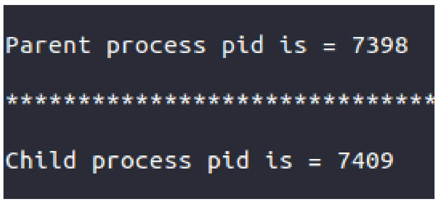
To solve this problem; I used ***os***, ***multiprocessing*** and ***hashlib*** modules. Firstly I checked checksum values with ***md5***() function then checked unique duplicate values with using ***check_duplicate***() and ***unique***() functions with using multiprocessing.

2.2.5 Checking Time Values

To solve this problem; I used ***time*** module to check finishing times. ***time.time***() function used for this.

3 Results

3.1 About Child and Parent PID



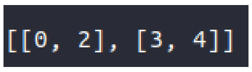
```
Parent process pid is = 7398

******************************

Child process pid is = 7409
```

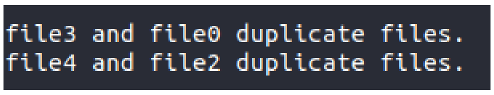
When I re-launch my application, Parent PID of the program changes. Also child PID changes and it is different number from parent.

3.2 Duplicate files



```
[[0, 2], [3, 4]]
```

I used arrays to use duplicate problem. This picture is result of which indexes are duplicates each other.

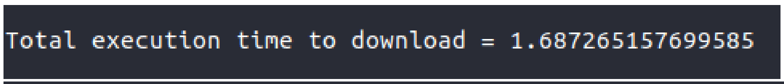


```
file3 and file0 duplicate files.
file4 and file2 duplicate files.
```

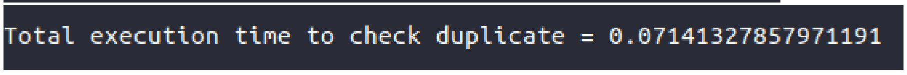
Also while I downloading files, every file have different names as filex (x is autoincrement). We can see the results with this names.

3.3 Time Results

In the right conditions, multiprocessing save a lot of time. I didn't use multiprocessing while I was downloading files, but I used multiprocessing while I was checking duplicates. These are results.



```
Total execution time to download = 1.687265157699585
```



```
Total execution time to check duplicate = 0.07141327857971191
```

Note

My CPU has 4 cores but I use 5 core in python with ***Pool***. So practically, if we use multiprocessing more than max core count of cpu, there is no error. Remaining cores are waiting to finish other processes.

4 Conclusion

To conclude this project, on GNU / Linux, every application create new processes as file. When re-launch the application, again operating system creates new process. That's why programmer should use these processes with more efficient ways. In this project, I can definitely say that, multiprocessing can save a lot of time, also some problems can solve with child processes. To make a good program, we need to understand these processes logic.