ASSIGNMENT REPORT 1: PROCESS AND THREAD IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

Murat Gun
muratgun@posta.mu.edu.tr
github.com/desxz/ceng_2034_2020_final

Sunday 7th June, 2020

Abstract

The technology is developing, the needs have increased. In line with these needs, computer hardware has also developed and took the form we use today. The development of processor and operating system architectures respond to the needs of the age.

1 Introduction

The aim of this report is to learn how to make our applications more efficient by using multi-thread, multi-process, and also to follow the stable and synchronous operation of a running process.

2 Assignments

In this project, multi-processing and multi-threading usage were made using Python. At the same time, an ancestor and child process were created and their synchronous operation was provided.

2.1 What is used in the project

2.1.1 Which kernel and system version used in the project

System Version: Ubuntu 20.04 (Virtual Box)
Kernel Version: Linux 5.4

2.1.2 Which language and version used in the project

Python 3.8
PyCharm Community Edition 2020.1.1

2.1.3 Libraries

os, this module provides a portable way of using operating system dependent functionality.
hashlib, this module implements a common interface to many different secure hash and message digest algorithms.
time, this module provides various time-related functions.
threading, this module constructs higher-level threading interfaces on top of the lower level thread module.
requests, the requests library is the defacto standard for making HTTP requests in Python.
signal, this module provides mechanisms to use signal handlers in Python.

2.2 Problems and Solves

2.2.1 Create a new child process with syscall and print its PID.

I used the **.fork()** function in the **os** library to create a child process. With this function, if the child whose pid (process id) is equal to 0 is not the process, it is the parent process. I used the **.getpid()** function in the **os** library to write the process id of the child process.

2.2.2 With the child process, download the files via the given URL list.

I used the **requests** library to download the pictures in the given urls. I made the download process in the content named **download-file()**. To do the download process with the child process, I wrote the **download-file()** function under the **os.getpid()** function under the child process block, so that the child process made the download process.

2.2.3 How can you avoid the orphan process situation with syscall?

I wrote the **os.wait()** command under the parent process block for the parent process to wait for the child process, this command keeps the process open and kept waiting for it to be closed at any time. After the child process is over, I closed the parent process and ended the program using the command **os.kill(pid (parent process), signal.SIGTERM)**. In this way, I solved the problem of orphan process.

2.2.4 Control duplicate files within the downloaded files of your python code.

At this stage, I used the names of the pictures I downloaded as numbers such as 0,1,2,3,4 for convenience. I first checked the downloaded images with the **findDuplicates()** function. After getting the correct results in these controls, I wrote this function using multithreading and multiprocessing. While writing multithreading is very easy, it took me a long time to write using multiprocessing. This is because I could not write and compare the hash of the pictures in an array because of the synchronous operation in the multiprocessing function. When all the processes were written at the same time and tried to check at the same time, the result returned an empty array. For this reason, I split the multiprocessing part into 2 parts and first put the hash of the pictures into an array, and I did this under the **takegethash()** function. Then, I checked the hash in the array I wrote using the **duplicateFinder()** function in accordance with multiprocessing. I wrote the multi Process **findDuplicates()** function to use this function as multiprocess.

3 Results

	TEST-1	TEST-2	TEST-3
1 Process	5.841s	5.182s	5.646s
2 Processes	4.252s	4.596s	3.445s
3 Processes	3.233s	3.112s	2.986s
4 Processes	3.278s	3.038s	3.154s
5 Processes	3.060s	3.200s	3.118s
6 Processes	3.435s	3.188s	3.601s

5 Threads	3.152s	4.448s	3.210s

Multiprocess and Multithread Run Times

```
ubuntu@ubuntu-VirtualBox:~/PycharmProjects/finalos$ time python3 ceng_2034_2020_
final.py
Successfully created the directory PythonPNGs/
Child process id:  4770
Directory Changed as /home/ubuntu/PycharmProjects/finalos/PythonPNGs/
PythonPNGs folder contains 5 files.
[('4', '3dcaee2bca739460bd30bb257785b107'), ('2', '3dcaee2bca739460bd30bb257785b
107'), ('3', 'c8ac40dc6b37096d61c34c9a50a794b5'), ('0', 'c8ac40dc6b37096d61c34c9
a50a794b5')]
Image 2 and Image 4 are duplicated.
Image 0 and Image 3 are duplicated.
Terminated

real    0m3,366s
user    0m0,340s
sys     0m0,073s
```

Multiprocess Using

```
ubuntu@ubuntu-VirtualBox:~/PycharmProjects/finalos$ time python3 ceng_2034_2020_
final.py
Creation of the directory PythonPNGs/ failed
Child process id:  5875
Directory Changed as /home/ubuntu/PycharmProjects/finalos/PythonPNGs/
Image 3 is duplicate of Image 0
Image 4 is duplicate of Image 2
Terminated

real    0m3,469s
user    0m0,331s
sys     0m0,081s
```

Multithread Using

```
/bin/python3.8 /home/ubuntu/PycharmProjects/finalos/ceng_2034_2020_final.py
Creation of the directory PythonPNGs/ failed
Child process id:  4354
Directory Changed as /home/ubuntu/PycharmProjects/finalos/PythonPNGs/
PythonPNGs folder contains 5 files.
[('4', '3dcaee2bca739460bd30bb257785b107'), ('2', '3dcaee2bca739460bd30bb257785b107'), ('3', 'c8ac40dc6b37896d61c34c9a58a794b5'), ('0', 'c8ac40dc6b37896d61c34c9a58a794b5')]
Image 2 and Image 4 are duplicated.
Image 0 and Image 3 are duplicated.

Process finished with exit code 143 (interrupted by signal 15: SIGTERM)
```

PyCharm Output

4 Conclusion

As a result, multi-thread and multi-process are different. They can be used effectively in Unix-based operating systems. They have advantages over each other in different transactions. The purpose of using multiprocessing in this project is that it is 1 process that compares each picture with other pictures, providing optimum working time. When we do the same with threading, we witness a more unstable working schedule. In addition, the parent process running on a child process should not stop before the child process is finished. This prevents the program from running stable.

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.