ASSIGNMENT REPORT : CENG 2034 OPERATING SYSTEMS MAKEUP EXAM

BARIŞ DİLEK
barisdilek@posta.mu.edu.tr
github.com/barisdilek48/ceng-2034-2020-makeup

Saturday 20th June, 2020

Abstract

In this study, we have seen the variations of the revealed codes according to the operating systems and to prove their differences. In this study, we have seen how operating systems manage processes or memory. Linux commands and what can be done with these commands, also we use python programming language and we learned that we can manage with the data structures of python.

1 Introduction

The purpose of this laboratory is to see how operating systems manage processes and memories in the language of python. In this lab, we learned the differences and aim to create better quality codes.

2 Assignments

In this experiment, although we learned some of the Linux commands, we also used them.Also in this project, multiprocessing usage were made using Python.

2.0.1

Language and Application version in this project Python 3.8
PyCharm Community Edition 2020.1.1

2.0.2 Libraries

hashlib, is a library that contains various popular encryption algorithms.And allows us to encrypt our data in different algorithms.
time, module provides various time-related functions.
threading, module constructs higher-level threading interfaces on top of the lower level thread module.
os, module provides a portable way of using operating system dependent functionality.
urllib.request, module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world — basic and digest authentication, redirections, cookies and more.

2.1 solutions

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

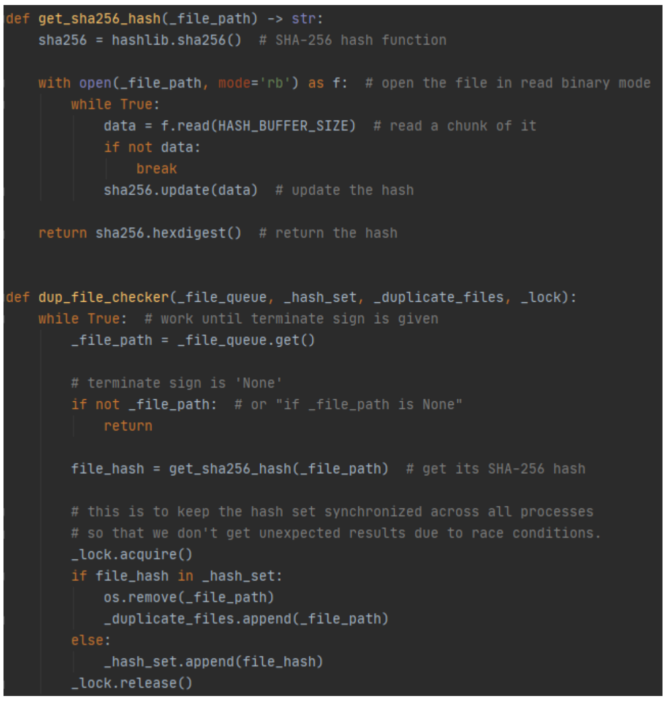
I used the **.fork()** function in the **os** library to create a child process. With this function, if the child whose pid (process id) is equal to 0 is not the process, it is the parent process.
I used the **.getpid()** function in the **os** library to write the process id of the child process.

2.1.2 Download the files via the given URL list.

I used the **urllib.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 given URL list

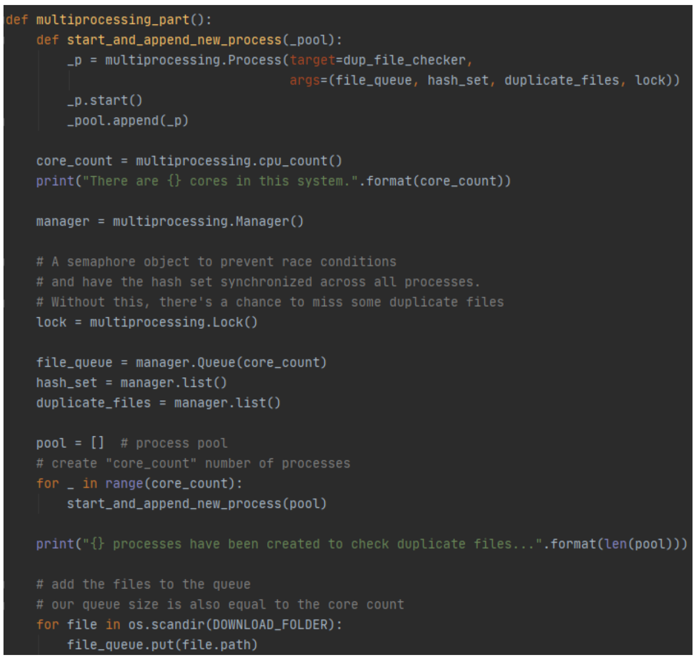
2.1.3 Multiprocessing Part

Check the system and learn the number of CPU cores. Create n processes if there are n cores.
Use processes for the correct task!
Control duplicate files within the downloaded files of your python code. You should do it by using multi processing techniques. (Hint: you can use hashlib -md5/sha256- in python to check file checksum)
The main process should check the other created processes and if takes more than 30 seconds, kill those processes (by sending signal from the main process)
The main process should check If the other processes didn't end successfully, then it should try again that process's job.

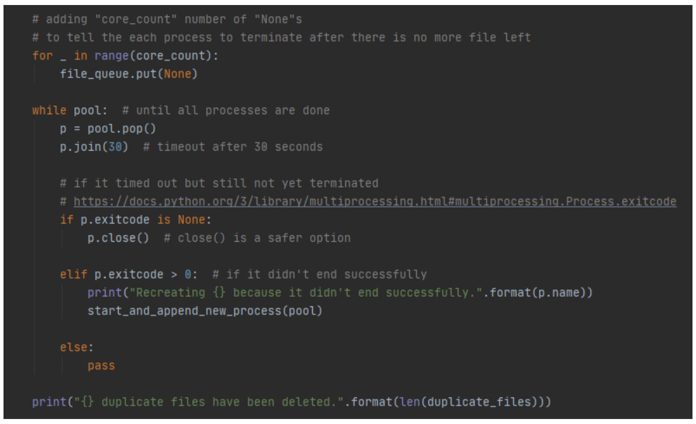


```
def get_sha256_hash(_file_path) -> str:
    sha256 = hashlib.sha256()  # SHA-256 hash function

    with open(_file_path, mode='rb') as f:  # open the file in read binary mode
        while True:
            data = f.read(HASH_BUFFER_SIZE)  # read a chunk of it
            if not data:
                break
            sha256.update(data)  # update the hash

    return sha256.hexdigest()  # return the hash


def dup_file_checker(_file_queue, _hash_set, _duplicate_files, _lock):
    while True:  # work until terminate sign is given
        _file_path = _file_queue.get()

        # terminate sign is 'None'
        if not _file_path:  # or "if _file_path is None"
            return

        file_hash = get_sha256_hash(_file_path)  # get its SHA-256 hash

        # this is to keep the hash set synchronized across all processes
        # so that we don't get unexpected results due to race conditions.
        _lock.acquire()
        if file_hash in _hash_set:
            os.remove(_file_path)
            _duplicate_files.append(_file_path)
        else:
            _hash_set.append(file_hash)
        _lock.release()
```

Figure 1: MULTIPROCESSING USAGE



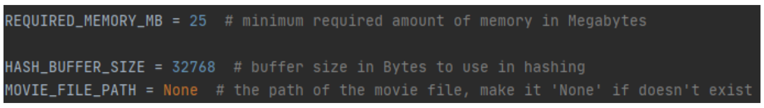
```python
def multiprocessing_part():
    def start_and_append_new_process(_pool):
        _p = multiprocessing.Process(target=dup_file_checker,
                                     args=(file_queue, hash_set, duplicate_files, lock))
        _p.start()
        _pool.append(_p)

    core_count = multiprocessing.cpu_count()
    print("There are {} cores in this system.".format(core_count))

    manager = multiprocessing.Manager()

    # A semaphore object to prevent race conditions
    # and have the hash set synchronized across all processes.
    # Without this, there's a chance to miss some duplicate files
    lock = multiprocessing.Lock()

    file_queue = manager.Queue(core_count)
    hash_set = manager.list()
    duplicate_files = manager.list()

    pool = []  # process pool
    # create "core_count" number of processes
    for _ in range(core_count):
        start_and_append_new_process(pool)

    print("{} processes have been created to check duplicate files...".format(len(pool)))

    # add the files to the queue
    # our queue size is also equal to the core count
    for file in os.scandir(DOWNLOAD_FOLDER):
        file_queue.put(file.path)
```



```python
    # adding "core_count" number of "None"s
    # to tell the each process to terminate after there is no more file left
    for _ in range(core_count):
        file_queue.put(None)

    while pool:  # until all processes are done
        p = pool.pop()
        p.join(30)  # timeout after 30 seconds

        # if it timed out but still not yet terminated
        # https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Process.exitcode
        if p.exitcode is None:
            p.close()  # close() is a safer option

        elif p.exitcode > 0:  # if it didn't end successfully
            print("Recreating {} because it didn't end successfully.".format(p.name))
            start_and_append_new_process(pool)

        else:
            pass

    print("{} duplicate files have been deleted.".format(len(duplicate_files)))
```

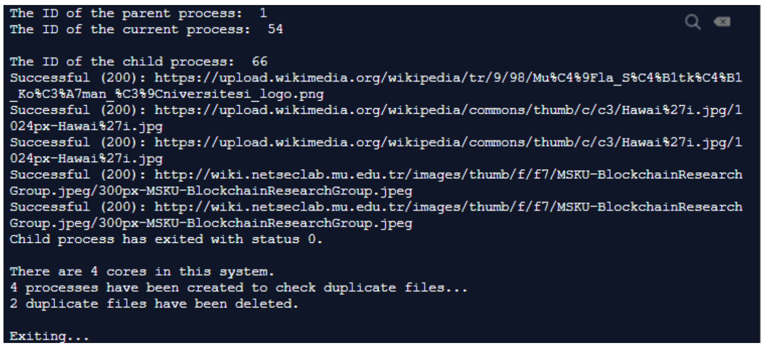
3 Results

As a result, We learned that if our operating system is Linux or Windows we can easily use multiprocessing. -We learned multiprocessing in python.Then we understood their features thoroughly. -Multi-thread and multi-process are different. They can be used effectively in Unix-based operating systems.



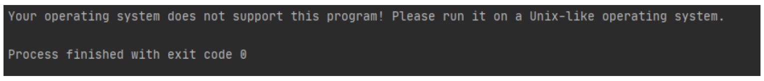
```
REQUIRED_MEMORY_MB = 25  # minimum required amount of memory in Megabytes

HASH_BUFFER_SIZE = 32768  # buffer size in Bytes to use in hashing
MOVIE_FILE_PATH = None  # the path of the movie file, make it 'None' if doesn't exist
```

Figure 2: specifying the needed memory hash buffer and film files



```
The ID of the parent process:  1
The ID of the current process:  54

The ID of the child process:  66
Successful (200): https://upload.wikimedia.org/wikipedia/tr/9/98/Mu%C4%9Fla_S%C4%B1tk%C4%B1
_Ko%C3%A7man_%C3%9Cniversitesi_logo.png
Successful (200): https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Hawai%27i.jpg/1
024px-Hawai%27i.jpg
Successful (200): https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Hawai%27i.jpg/1
024px-Hawai%27i.jpg
Successful (200): http://wiki.netseclab.mu.edu.tr/images/thumb/f/f7/MSKU-BlockchainResearch
Group.jpeg/300px-MSKU-BlockchainResearchGroup.jpeg
Successful (200): http://wiki.netseclab.mu.edu.tr/images/thumb/f/f7/MSKU-BlockchainResearch
Group.jpeg/300px-MSKU-BlockchainResearchGroup.jpeg
Child process has exited with status 0.

There are 4 cores in this system.
4 processes have been created to check duplicate files...
2 duplicate files have been deleted.

Exiting...
```

Figure 3: Online Linux python compiler output

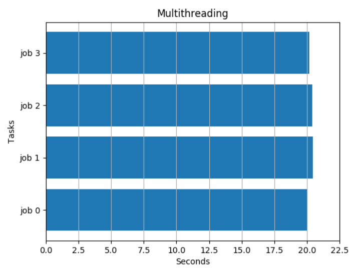


```
Your operating system does not support this program! Please run it on a Unix-like operating system.

Process finished with exit code 0
```

Figure 4: When we run the code on Windows OS

4 Conclusion

As a conclusion, the differences of the python language according to the operating systems and the outputs according to the operating systems were examined and the results were observed.



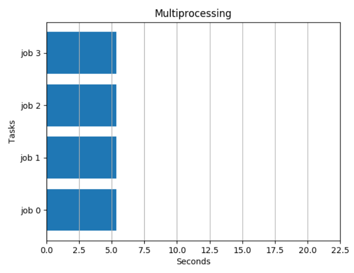
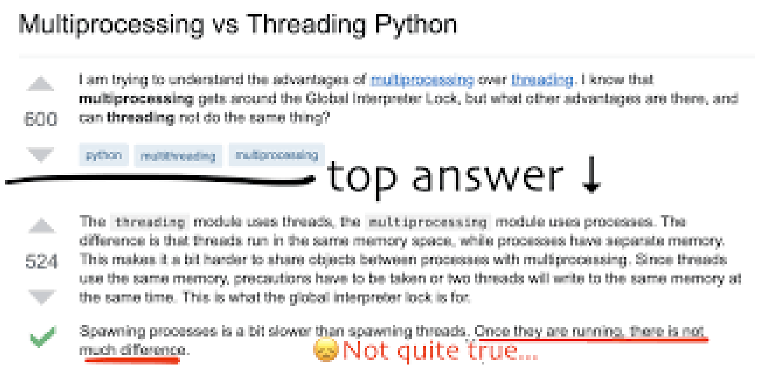


Figure 5: Other issues we learned multiprocessing and multithreading in python.In the picture above, we have seen a brief definition of the two terms.



Multiprocessing vs Threading Python

600

I am trying to understand the advantages of multiprocessing over threading. I know that **multiprocessing** gets around the Global Interpreter Lock, but what other advantages are there, and can **threading** not do the same thing?

python multithreading multiprocessing

top answer ↓

524

The `threading` module uses threads, the `multiprocessing` module uses processes. The difference is that threads run in the same memory space, while processes have separate memory. This makes it a bit harder to share objects between processes with multiprocessing. Since threads use the same memory, precautions have to be taken or two threads will write to the same memory at the same time. This is what the global interpreter lock is for.

Spawning processes is a bit slower than spawning threads. Once they are running, there is not much difference.

Not quite true...

Figure 6: We observe the difference between multithreading and multiprocessing. Multithreading takes 20 seconds, while multiprocessing performs faster processing.. So now that we are convinced that they're not the same, we would like to know why.We have seen and understood the difference from the pictures above.