ASSIGNMENT REPORT 1: PROCESS AND THREAD IMPLEMENTATION

Abstract

The aim of this lab course is to understand and implement process and thread creation in Linux environment. Python programming language and Visual Studio Code IDE is chosen for this assignment. The Python version used is 3.7.3 64-bit.

The process creation in Linux environment is revised and Python support for this action researched. Besides, the multi processing techniques used in Python are investigated. Requirements of the assignment are fulfilled using *fork()* and *process()* functions in Python. In order to achieve a safe application, orphan process formation avoided using *wait()* system call in parent process.

1 Introduction

The subject of this report is to give a brief overview of the lab assignment. This lab is performed to have a clear understanding of sub processes, their creation, lifetime, risks and avoiding these risks. In addition, multi processing solutions used for implementation of some tasks. These operations have more importance than before as hardware power and number of cores in processors rise highly during the last years. The usage of this technological improvements can boost the performance of the applications, if the necessary programming concepts are followed.

2 Assignments

At the beginning of the assignment, the requirements needed to be determined. When all the tasks were examined, the requirements and constraints can be clearly noted. After choosing the programming language and environment, I needed to study the implementation of the required tasks in chosen language while keeping the constraints in mind.

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

In Linux environment creating a sub-process is different than creating it in Windows environment. Therefore, this constraint lead me to focus on Python process creation in Linux. The *fork()* function from 'os' module can be used for this purpose in Linux environment, meaning does not work in Windows operating systems. This system call duplicates current process in memory while returning 0 to child process and PID of child to parent process. It can be determined what actions

to be performed in parent or in child process by comparing the result with '0' value. The sub process created with the code below and process ID is printed to the console.

```
# Create a child process
child_process = os.fork()

    # run child process
    if child_process == 0:
        print("Child Process ID: " + str(os.getpid()))

    # run main process
    else:
        print("Main Process ID: " + str(os.getpid()))
```

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

In previous task, the child process was created and the access to process can be conditioned with the result of function. Thus, this task is aimed to create a function that is called in appropriate condition. There are two functions created for this task; *child_func()* and *download_file(url, file_name=None)*. Child function calls download_file function with constant URL parameters and associated file names. If no file name mentioned, the function generates a UUID and uses it as a file name for the downloaded files. I added file names manually in my implementation to have an easier reading of the results, especially for the next tasks.

The URL addresses that were used are;

- 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/Muğla_Sıtkı_Koçman_Üniversitesi_logo.png
- https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Hawai'i.jpg/1024px-Hawai'i.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'i.jpg/1024px-Hawai'i.jpg

The implementation in Python;

```
def download_file(url, file_name=None):
    r = requests.get(url, allow_redirects=True)
    if file_name:
        file = file_name
    else:
        file = str(uuid4())
    open(file, 'wb').write(r.content)


def child_func():
    download_file(URL1, "url1.jpeg")
```

```
download_file(URL2, "url2.png")
download_file(URL3, "url3.jpg")
download_file(URL4, "url4.jpeg")
download_file(URL5, "url5.jpg")
```

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

When a parent process finishes executing before its children, the child processes become orphan processes. When this happens, the child is adopted by the *init* process which has the PID of '1'. In some rare cases orphan processes are intentionally used by developers. Especially for the long-running jobs that do not need user attention. Except these cases, orphan processes need to be avoided in systems as they consume memory and CPU resources without user knowledge.

In order to prevent a child process become an orphan process, *wait()* system call function is used in Python language. This function waits for the children processes to exit and collects information from them. Likewise *fork()* process, *wait()* also exists in 'os' module. This function returns the child's PID and information about the child process. As it can be seen below, the *wait()* function is added into the main process condition. [1]

```
child_process = os.fork()

# run child process
if child_process == 0:
    print("Child Process ID: " + str(os.getpid()))
    child_func()

# run main process
else:
    print("Main Process ID: " + str(os.getpid()))
    # wait for child process
    os.wait()
```

2.4 Control duplicate files within the downloaded files of your python code. You should do it by using multi processing techniques.

This task is written using the multi processor module in Python. A new process created that calls the *check_files(lock)*. The lock parameter assigns necessary resources to the process and released before the function ends. The function first checks the Python script directory for items. Than check the items if they are file or directory. Finally calculate the sha256 hash for these files except the Python code file and add them to a dictionary.

Next step was comparing the dictionary values which are hashes with each other. A copy of the dictionary and two loops where one is inside other one was used for this aim.

```
def check_files(lock):
    lock.acquire()
    # generate dictionary for file and hash
    file_hash_list = {}
    # select files and folders in current folder
    for item in os.listdir():
        # check if item is file or not
```

```
        if os.path.isfile(os.path.join(".", item)):
            # check if the file is running script or downloaded files
            if item != Path(__file__).name:
                # open file
                with open(item, "rb") as f:
                    bytes = f.read()
                    # generate hash for file
                    hash = sha256(bytes).hexdigest()
                    # add file and hash to dictionary
                    file_hash_list[item] = hash

    # find duplicate hashes in dictionary and print
    new_dict = file_hash_list.copy()
    for key, value in file_hash_list.items():
        exist = False
        for k, v in new_dict.items():
            if key != k and value == v:
                print("Have same hash:", key, k, value)
                exist = True
                break
        if exist:
            new_dict.pop(k)
            new_dict.pop(key)
    lock.release()
```

The function call is done from the parent process after *wait()* function. This ensures that the child process finished and downloaded files are ready to be checked.

```
    lock = Lock()
    p1 = Process(target=check_files(lock))
    p1.start()
```

3 Results

The application successfully outputs the results of tasks according to the requirements and constraints. As it is shown at Figure 1, both functions for downloading files are running in a sub-process.

The output of the application can be seen at Figure 2. The process IDs of both main and child processes are printed to the console. Following these information, the files which have same hash values are printed. In this lab, it was noticed that the URL5, URL3 couple and URL4, URL1 couple have the same hashes, meaning that they are same files.

4 Conclusion

There are two ways of child process creation with Python, one is with *fork* function and the other one is with *subprocess* function. Subprocess function is cross-platform but it is mainly used for executing an arbitrary program where you fully write the executing command. Fork function only works in Unix systems but the child process executes the script from the exact line where it

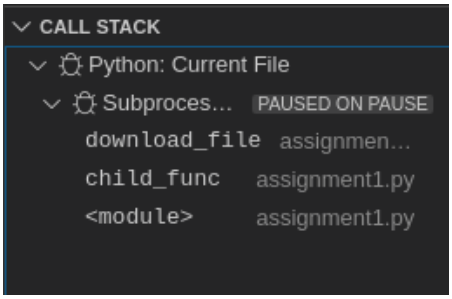

Figure 1: Visual Studio Code IDE - Call Stack window. Child process is paused. Thus, the functions running in child process can be seen here.

Figure 2: Terminal window. The output of the application is shown.

is called and returns the parent process, thus there are two processes running in same script. I chose the fork function as the whole program would be in one script file and after the ending of the child process, checking of files can be directly started.

Although I used sha256, there are also other algorithms like md5 for hash creation. Those algorithms may be faster and decrease the execution time of the application. Comparison tests can be actualized in future labs.

References

[1] Lex Toumbourou. What are zombie processes?