ASSIGNMENT REPORT 1: PROCESS AND THREAD IMPLEMENTATION

CENG2034, OPERATING SYSTEMS

H.Barış Afşar
hudayibarisafsar@mu.edu.tr
github.com/barisavsar

Tuesday 23rd June, 2020

Abstract

Multithreading is the ability to run concurrent tasks within the same process. This way of concurrent programming allows the the threads to share state, and execute from the same memory pools. Multiprocessing also know as process forking, is a way of running multiple tasks at the same time. This is different to multithreading as we are duplicating the whole process, duplicating the memory and resource requirements

1 Introduction

The purpose of this laboratory is to download the files from the list of given URLs with a multithread structure. Checking the system and learning the number of CPU cores and the Main process, other processes created and if Over 30 seconds, kill these processes.

2 Assignments

Multithread and mulprocessing structures are used in this laboratory with python commands..

2.1 Library

Hashlib module, included in The Python Standard library is a module containing an interface to the most popular hashing algorithms.
Urllib.request is a Python module for fetching URLs (Uniform Resource Locators). It offers a very simple interface, in the form of the urlopen function. This is capable of fetching URLs using a variety of different protocols
Threading in Python is simple. It allows you to manage concurrent threads doing work at the same time. The library is called “threading“, you create “Thread” objects, and they run target functions for you. You can start potentially hundreds of threads that will operate in parallel.
The Python **time** module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.
The **OS** module in Python provides a way of using operating system dependent functionality.

The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.

3 Project

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

.fork () function in the os library to create a subprocess. With this function, A child whose pid (process id) equals 0 is the main process, not the process. .getpid () function in the os library to write the process ID of the child process.

```
def child_process():
    print("\nThe ID of the child process: ", os.getpid())

    threads = []
    file_paths = []  # file path list to remove files if the
    parent dies before the child
    for URL in URLs:
        thread = Thread(target=download_file, args=(URL,
        file_paths))  # create thread
        thread.start()  # start it
        threads.append(thread)  # add it to the threads list

    for thread in threads:  # wait until each thread is
    completed
        thread.join()

    # If the parent process ID is 1, it means that the
    parent process is dead.
    if os.getppid() == 1:
        print("It looks like the top process is somehow dead,
         so we delete the files we just downloaded.")
        for _path in file_paths:
            os.remove(_path)
        os._exit(-1)

    os._exit(0)
```

Figure 1: Child Process

3.0.2 Download the files via the given URL list.

urllib.requests requests a library to download images from a specific url.

```python
def download_file(_url: str, _file_paths: list, file_name=None):
    try:
        # We are not allowed to use any third party addiction.
        so I use built-in "urllib"
        r = urllib.request.urlopen(_url)  # request the file

        if not os.path.exists(DOWNLOAD_FOLDER):  # otherwise
        create directory
            os.makedirs(DOWNLOAD_FOLDER)

        # auxiliary variables to get file format and full local
        path
        file_format = _url.split('.')[-1]
        local_path = DOWNLOAD_FOLDER + (file_name if file_name
        else str(uuid.uuid4())) + '.' + file_format

        status_code = r.getcode()
        if status_code == 200:  # if the request was successful
            with open(local_path, 'wb') as f:
                f.write(r.read())  #write your content to a file
        else:
            raise Exception("Failed ({}): {}".format(status_code,
             _url))
    except Exception as e:
        print(e)
    else:
        print("Successful ({}): {}".format(status_code, _url))
        _file_paths.append(local_path)
```

Figure 2: Dowload url

3.0.3 Multiprocessing

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 a sign of termination is given
        _file_path = _file_queue.get()

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

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

        # This is to keep the hash set synchronized across
        all processes
```

Figure 3: Multiprocessing

```
            # So we cannot 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()


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()
    # Without this, there's a chance to miss some duplicate
    files
    lock = multiprocessing.Lock()
```

Figure 4: Multiprocessing

```
for file in os.scandir(DOWNLOAD_FOLDER):
    file_queue.put(file.path)

# "core_count" adds "No" number
# to tell each process to be terminated after more files
remain
for _ in range(core_count):
    file_queue.put(None)

while pool:  # until all transactions are done
    p = pool.pop()

    p.join(30)  # timeout after 30 seconds

    # timed out but not yet terminated

    if p.exitcode is None:
        p.close()  # close() is a safer option

    elif p.exitcode > 0: # did not succeed
        print("Rebuilding {} because it didn't
        succeed...".format(p.name))
        start_and_append_new_process(pool)

    else:
        pass

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

Figure 5: Multiprocessing

```
import hashlib
import os
import time
import sys
import uuid
import multiprocessing
import urllib.request  #built-in library for downloading files
from threading import Thread

REQUIRED_MEMORY_MB = 30  # minimum required amount of memory
in Megabytes
HASH_BUFFER_SIZE = 30000  # buffer size in bytes to use
MOVIE_FILE_PATH = None  # If the movie file has a path, do
'None'
```

Figure 6: memory hash buffer and film files

3.0.4 Check available memory of the system.
Estimate what should be the minimum required memory.
Include the routine that will wait until the system frees that memory

3.0.5 Output

```
ID of the parent process:  1
The identity of the current process:  147

The ID of the child process:  159
Successful (200): https://upload.wikimedia.org/wikipedia/tr/9/9
8/Mu%C4%9Fla_S%C4%B1tk%C4%B1_Ko%C3%A7man_%C3%9Cniversitesi_logo
.png
Successful (200): https://upload.wikimedia.org/wikipedia/common
s/thumb/c/c3/Hawai%27i.jpg/1024px-Hawai%27i.jpg
Successful (200): https://upload.wikimedia.org/wikipedia/common
s/thumb/c/c3/Hawai%27i.jpg/1024px-Hawai%27i.jpg
Successful (200): http://wiki.netseclab.mu.edu.tr/images/thumb/
f/f7/MSKU-BlockchainResearchGroup.jpeg/300px-MSKU-BlockchainRes
earchGroup.jpeg
Successful (200): http://wiki.netseclab.mu.edu.tr/images/thumb/
f/f7/MSKU-BlockchainResearchGroup.jpeg/300px-MSKU-BlockchainRes
earchGroup.jpeg
Child process has exited with status 0.

There are 4 cores in this system.
4 Actions created to check for duplicate files ...
5 duplicate files have been deleted.

exiting the program....
```

Figure 7: output

4 RESULT