Sophie Schaesberg

Professor Olivier Perard

Operating Systems

3 February 2023

Process & Thread Assignment

1. **Create a Program with 5 child processes in parallel:**

import os

from multiprocessing import Process

def pureCpuIntensive(x):

sum=0

for x in range(100000000):

sum+=x

def run\_process(process\_number):

print(f"Starting process {process\_number}")

pureCpuIntensive(process\_number)

print(f"Finished process {process\_number}")

processes = []

for i in range(5):

p = Process(target=run\_process, args=(i,))

processes.append(p)

p.start()

for p in processes:

p.join()

Graphical user interface, application

Description automatically generated**Graphical user interface

Description automatically generated with low confidence**

1. Import os and multiprocessing modules
2. Define pureCpuIntensive function, which executes a CPU-intensive operation of summing up numbers in a range of 100000000
3. Define run\_process function, which takes process\_number argument and prints a starting message for the process, calls pureCpuIntensive function with the argument, and prints an end message for the process.
4. Initialize empty list processes to store Process objects.
5. Use a for loop to create 5 Process objects, each with run\_process function as target and i as argument. The i value ranges from 0 to 4
6. Start each process in the list processes using the start method.
7. Use another for loop to call the join method for each process in the processes list, which blocks the main process from terminating until the child processes finish
8. **Create a Program with 2 child processes in Tree organization:**

import time

import os

from multiprocessing import Process

def pureCpuIntensive(process\_n):

print(f'Starting process N {process\_n} process pid', os.getpid())

int\_time = time.time()

sum = 0

for x in range(100000000):

sum += x

end\_time = time.time()

print(f'Process N {process\_n} took time:', end\_time - int\_time)

def child(number):

pureCpuIntensive(number)

p2 = Process(target=pureCpuIntensive, args=(number + 1,))

p2.start()

p2.join()

def multiprocessing\_2\_tree():

print("multiprocessing\_2\_tree started")

int\_time = time.time()

int\_process\_time = time.process\_time()

p1 = Process(target=child, args=(1,))

p1.start()

p1.join()

end\_time = time.time()

end\_process\_time = time.process\_time()

ex\_time = end\_time - int\_time

ex\_cpu\_time = end\_process\_time - int\_process\_time

print("Total time of execution:", ex\_time)

print("CPU time of execution:", ex\_cpu\_time)

return [ex\_time, ex\_cpu\_time]

if \_\_name\_\_ == "\_\_main\_\_":

multiprocessing\_2\_tree()

**Graphical user interface, application

Description automatically generatedGraphical user interface, text

Description automatically generated**

1. Function pureCpuIntensive takes an argument process\_n and executes a CPU intensive calculation by initializing variable sum to 0 and doing the calculation in a for loop from 0 to 100000000, adding each iteration to the sum variable. Execution time of calculation for current process is printed
2. Function child takes an argument number. Calls pureCpuIntensive function with argument number. Creates new process p2 and calls pureCpuIntensive function with argument number + 1
3. multiprocessing\_2\_tree initializes the time of execution and CPU time, creates a new process p1 and calls child function with argument 1. Waits for process p1 to finish, calculates total execution CPU time, prints them
4. main function is executed when the code is run. Calls multiprocessing\_2\_tree function, creating 2 child processes in a tree
5. **Create a Program with 5 child Threads.**
6. import time
7. import threading
8. def pureCpuIntensive(x):
9. sum = 0
10. for i in range(100000000):
11. sum += i
12. threads = []
13. for i in range(5):
14. thread = threading.Thread(target=pureCpuIntensive, args=(i,))
15. threads.append(thread)
16. thread.start()
17. for thread in threads:
18. **A picture containing graphical user interface

    Description automatically generatedGraphical user interface, application

    Description automatically generated** thread.join()
19. Import time and threading libraries
20. pureCpuIntensive(x) calculates the sum of integers from 0 to 100000000
21. Create an empty list threads to store threads
22. For loop to create 5 threads. For each iteration a thread is created using the threading.Thread class. target argument of this class is set to function pureCpuIntensive and argument args to (i,)
23. starts created thread and appends it to list threads
24. for loop to join all threads in list threads before program terminates

**Explain if Multiprocessing or MultiThreading is faster in your Operating System.**

The execution time for multi-process (9.57) is faster than multi-threading (15.13). In my operating system MacOSX, multi-processing is faster than multi-threading, but this depends on various factors such as the number of child processes/threads, system specifications and the type of task.