作業系統概論 HW1 Report

0716032 林佑鑫

I. Additional packages (With Python 3.8.6)

```
import time
import asyncio
import hashlib
import requests
from lxml.html import fromstring
from concurrent.futures import ProcessPoolExecutor, ThreadPoolExecutor, as_completed
```

time: 計算 task 總共運算時間

asyncio: 用於 coroutine 的 library

hashlib: 用來算出 string 的 sha256 值

requests: 從 input 的網址取得網頁的各種資料

fromstring: 用來解析 Task2 網頁的資料，並取出標題

ProcessPoolExecutor: 這個 module 可以直接開 n 個 threading 並收集結果

ThreadPoolExecutor: 這個 module 可以直接開 n 個 processes 並收集結果

as_completed: 在做上面兩種 Executor 時用來判斷 thread , process 是否已經完成

II. The effect of the number of threads on performance:

Task 1:

1 thread:

第一次：`186.06 seconds`

第二次：`194.22 seconds`

第三次：`184.98 seconds`

平均：188.42 seconds

2 threads:

第一次：`184.84 seconds`

第二次：`185.32 seconds`

第三次：`184.56 seconds`

平均：184.91 seconds

4 threads:

第一次：`186.94 seconds`

第二次：`208.94 seconds`

第三次：`186.77 seconds`

平均：194.22 seconds

100 threads:

第一次：`201.87 seconds`

第二次：`189.85 seconds`

第三次：`192.82 seconds`

平均：194.85 seconds

結論：在四種不同 thread 數量下，Task 1 執行 100 筆資料所需要的時間都大致相同，可以說明 thread 數量對 Task 1 來說影響不大。

可能原因：因為每個 thread 裡面都要跑過 string 每種可能的的情況，每個 thread 做的事情都一樣，且都是從頭開始，所以通常是第一個開始的 thread 會最先完成，所以不論 thread 數量多寡都不會影響效率。

Task 2:

1 thread:

第一次：`105.62 seconds`

第二次：`95.88 seconds`

第三次：`73.75 seconds`

平均：91.75 seconds

2 threads:

第一次：`90.22 seconds`

第二次：`108.05 seconds`

第三次：`31.44 seconds`

平均：76.56 seconds

4 threads:

第一次：`74.74 seconds`

第二次：`56.03 seconds`

第三次：79.99 seconds

平均：70.25 seconds

100 threads:

第一次：40.47 seconds

第二次：71.86 seconds

第三次：49.71 seconds

平均：54.01 seconds

結論：在四種不同 thread 數量下，Task 2 執行 100 筆資料所需要的時間依 thread 數量增加而減少，可以說明 thread 數量對 Task 2 有影響，且 thread 越多執行速度越快。

可能原因：因為每筆資料進來後會分配一個 thread 去找他的 title，所以 thread 越多，代表分工越多，找的速度自然就快。

III. The effect of the number of processes on performance:

Task 1:

1 process:

第一次：218.70 seconds

第二次：220.98 seconds

第三次：211.76 seconds

平均：217.15 seconds

2 processes:

第一次：217.37 seconds

第二次：203.74 seconds

第三次：206.46 seconds

平均：209.19 seconds

4 processes:

第一次：222.29 seconds

第二次：217.55 seconds

第三次：245.48 seconds

平均：228.44 seconds

100 processes:

第一次：1169.44 seconds

開 100 processes 對我的電腦來說太耗資源，所以速度很慢，只做一次。

結論：在四種不同 process 數量下，除了 100 processes 外執行 Task1 的 100 筆資料所需要的時間都大致相同，可以說明 process 數量對 Task 1 來說影響不大。

可能原因：和 thread 原因大致相同，因為每個 process 裡面都要跑過 string 每種可能的的情況，每個 process 做的事情都一樣，且都是從頭開始，所以通常是第一個開始的 process 會最先完成，所以在不考慮電腦效能的情況下，不論 process 數量多寡都不會影響效率。

Task 2:

1 process:

第一次：49.84 seconds

第二次：89.95 seconds

第三次：66.21 seconds

平均：68.67 seconds

2 processes:

第一次：67.53 seconds

第二次：114.35 seconds

第三次：109.24 seconds

平均：97.04 seconds

4 processes:

第一次：133.72 seconds

第二次：164.28 seconds

第三次：157.94 seconds

平均：151.98 seconds

100 processes:

第一次：973.17 seconds

一樣時間太久了，所以只做一次

結論：在四種不同 processes 數量下，執行 100 筆資料所需要的時間隨著 processes 數量增加，可以說明 processes 數量對 Task 2 有影響，且 processes 越多執行速度越慢。

可能原因：會因為 processes 增加造成效能而被拉低，雖然有平行分工，但是每個 process 所需要時間增加，結果反倒造成總運算時間增加，才會造成 processes 數量越多所需時間越長的結果。

IV. The performance comparison of multithreading, multiprocess, and coroutine

Task 1:

Single thread:

前面得到的平均：188.42 seconds

100 threads:

前面得到的平均：194.85 seconds

100 processes:

前面得到的結果：1169.44 seconds

coroutine:

第一次：`190.55 seconds`

第二次：`185.52 seconds`

第三次：`187.00 seconds`

平均：199.21 seconds

Task 2:

Single thread:

前面得到的平均：91.75 seconds

100 threads:

前面得到的平均：54.01 seconds

100 processes:

前面得到的結果：973.17 seconds

coroutine:

第一次：`10.75 seconds`

第二次：`1.89 seconds`

第三次：`6.92 seconds`

平均：6.52 seconds

結論：

Task 1：除了 100 processes 會被電腦效能卡住而拉低很多以外，其他方式做起來效果差不多，因為每個 process 做的事情都一樣，且都是從頭開始，所以通常是第一個開始的 process 會最先完成，所以 thread 的數量並不會影響運算時間。

Task 2：一樣先排除 100 processes 的情況，thread 數量增加後運算時間有明顯下降，因為在平行分工的情況下，每個 thread 都去找網頁的 title，分工完成，比單用一個 thread 來得更快速，而 coroutine 又更快速，因為他只要碰到 request 的等待時間都會去做別的事，完全不浪費等待的時間，而其中第二次的結果我猜是因為有快取所以才這麼快。