OS HW1 0711529 陳冠儒

python 版本：3.8.5
input data：助教提供之範例

我們先來討論一下 Multi-threading 跟 Multi-processing 的差異：
Process 就好比是一間工廠，而 thread 則是員工，如果電腦擁有多個處理核心，就代表系統可以同時調用的員工數量增加，此時可以有兩種策略。

1. Multi-processing：同一時間內為各家工廠都分配一個員工去作事，Multi-processing 平行執行。跟 single process 處理比起來，其優點在於可以在**相同的時間內完成較多的工作**。
2. Multi-threading：在同一時間內把所有員工都派到同一家工廠去工作，Multi-threading 平行執行。相較於 single threading 處理方式，它**有機會讓相同的工作在比較短的時間內完成**。

Task 的設計：

- Task1：Proof of Work
  - 使用套件：hashlib
  - 方式：利用遍歷從最小一直試到最大，直到找到結果就 return
- Task2：Get Url Header
  - 使用套件：BeatutifulSoup 的 bs4、requests
  - 方式：requests.get(url)得到 html 後，再用 bs4 去得到 head.title.text

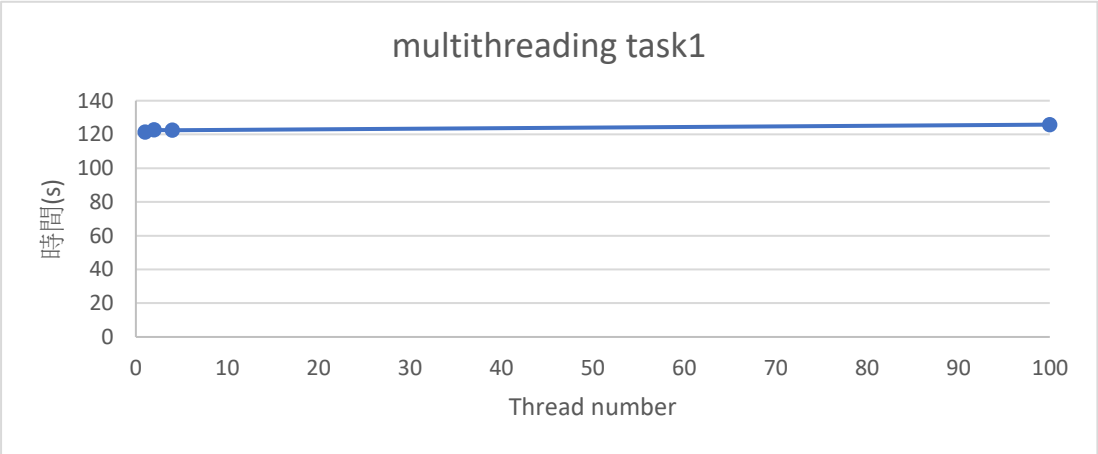
一、執行緒數量對效能的影響

- Multi-threading 的設計
  - 使用套件：threading、queue
  - 方式：將所有 task 都先放到 queue 中，再把他們 get 出來分給各個 thread 去執行

- 執行結果
  - Task 1

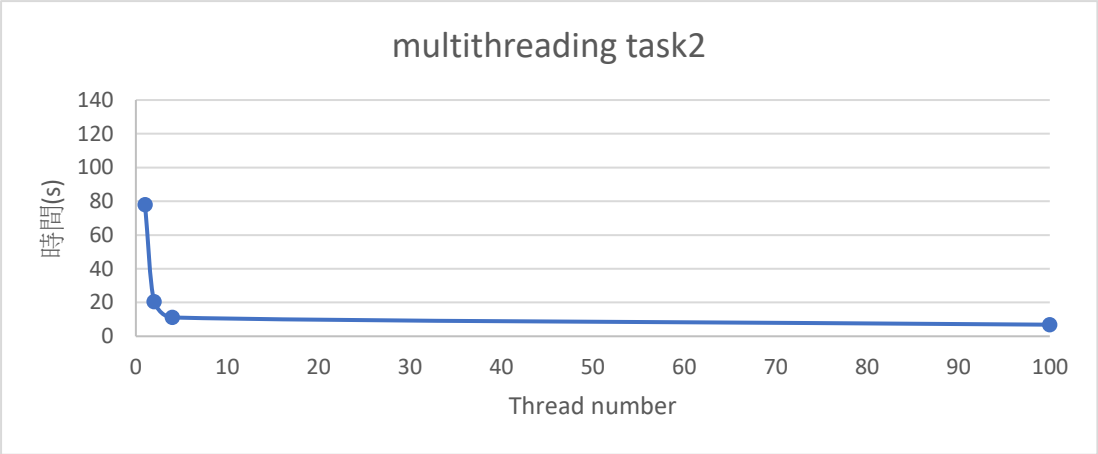
threads number	1	2	4	100
time1(s)	122.24437	124.57429	122.61275	125.53242
time2(s)	121.93974	121.60334	122.87616	125.94289
time3(s)	120.19224	122.06673	122.02587	125.99407

average time(s)	121.45878	122.74812	122.50493	125.82313



➢ Task 2

threads number	1	2	4	100
time1(s)	91.484702	13.94796	11.97931	6.51481
time2(s)	87.942572	26.50696	10.80837	6.4377
time3(s)	54.606038	20.80738	10.64665	6.579089
average time(s)	78.011104	20.42077	11.14478	6.510533



● 結果說明

Multi-thread 的優點是

1. 在一部分 blocked 時，可以切換到另一部份繼續跑
2. 同一個 process 下的 thread 可以共享資源
3. 他進行 context switch 的速度遠快於 process

但我們可以發現在 task1 下他的速度沒有下降，因為 task1 是 CPU bound，需要進行大量的計算，並且沒有任何 I/O burst，所以此時的 thread 並沒有任何作用，甚至可能會因為要 create thread 而使得花費的時間更久，以上面工廠的例子來說明，就是一個工作已經使得一間工廠的運作達到了最

大，故此時即使有再多的員工也無法增快速度。

在 task2 下，在 requests.get(url)時會被 blocked 住，所以此時可以發揮 thread 的效用，進行 concurrent 的運行，使得花費時間大幅的下降。

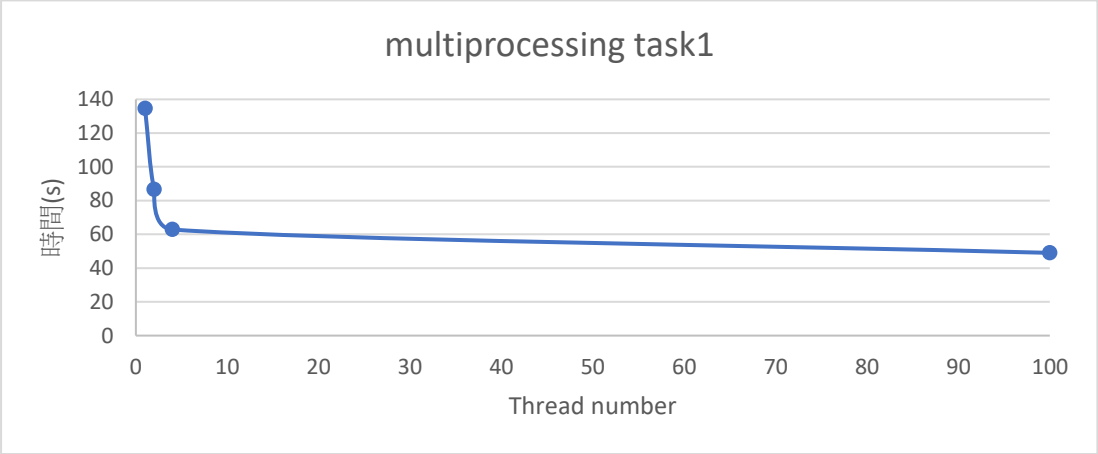
二、行程數對效能的影響

- Multi-Process 設計
  - 使用套件：multiprocessing 的 Pool
  - 方式：利用 processing pool 將 task map 到各個 process 去執行。

- 執行結果

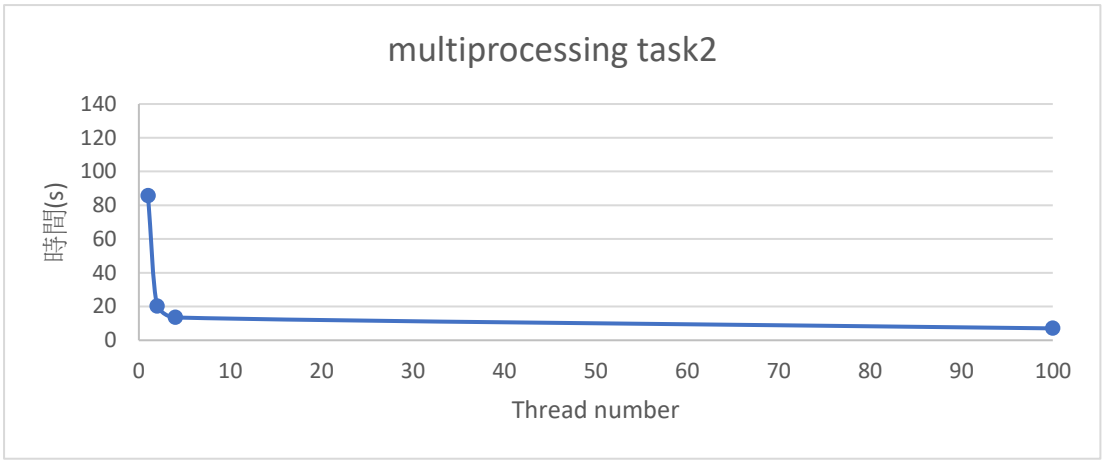
➢ Task 1

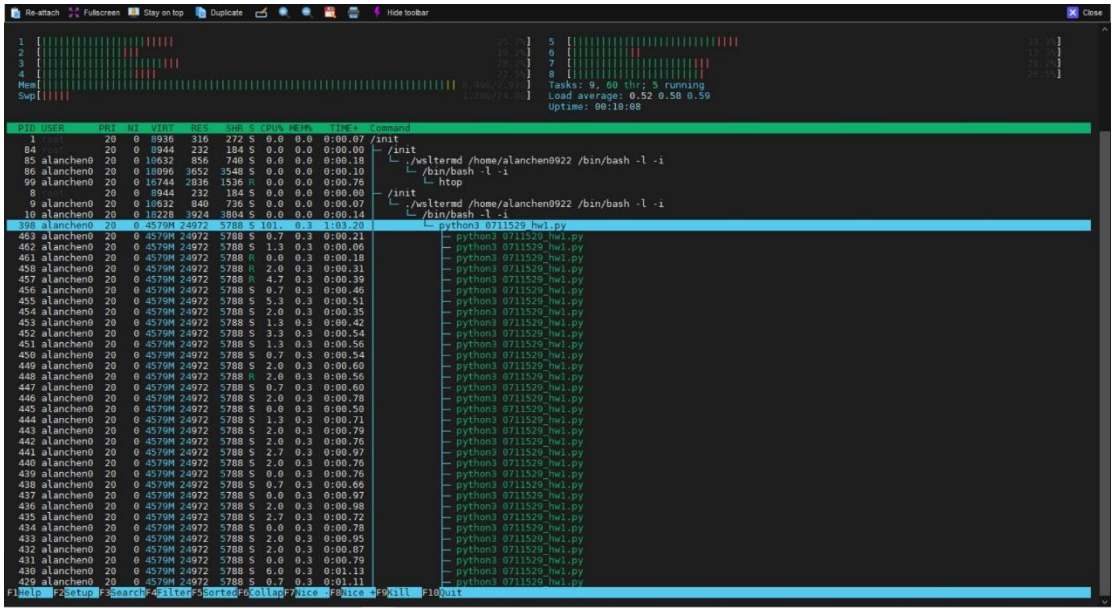
processes number	1	2	4	100
time1(s)	137.23133	85.092686	62.814119	47.427244
time2(s)	133.15478	87.785173	62.830519	49.103679
time3(s)	133.69309	86.995256	62.873352	50.542476
average time(s)	134.69306	86.624372	62.83933	49.024466



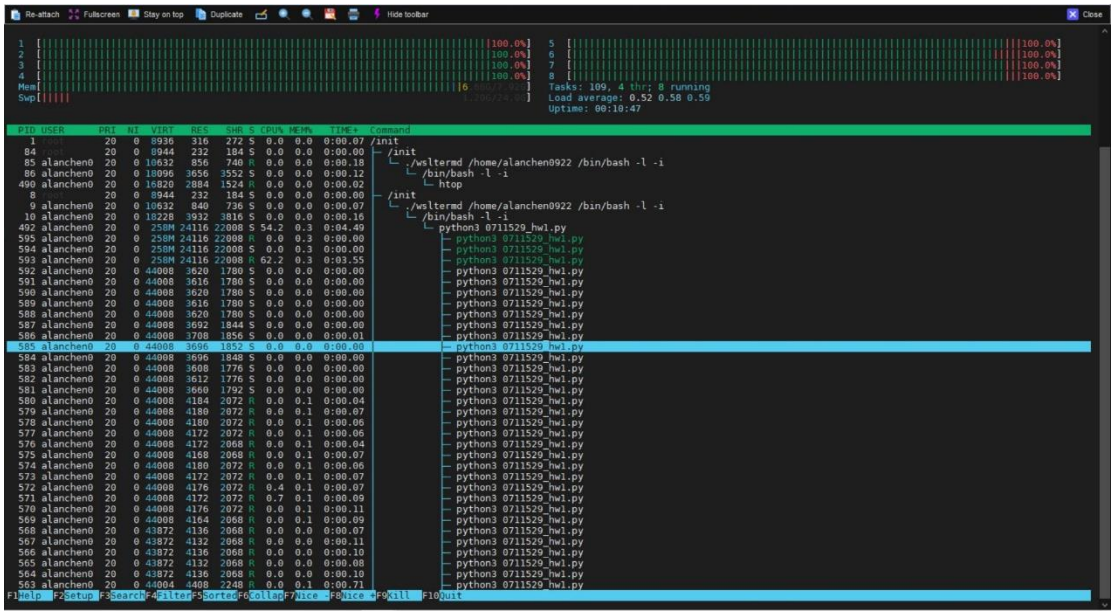
➢ Task 2

processes number	1	2	4	100
time1(s)	99.218601	21.00603	16.07411	5.701726
time2(s)	67.952653	18.74726	9.836351	5.54199
time3(s)	89.621445	20.70694	14.57304	4.120386
average time(s)	85.597566	20.15341	13.4945	5.12136733





Multi-thread Task1 100 thread



Multi-process Task1 100 process

- 結果說明

由上面兩個執行狀況可以看到 multi-process 可以使用全部的 CPU 資源，而 multi-thread 只能使用一個 process 能使用的 CPU 資源，故 multi-process 在 task1 和 task2 執行的速度都能隨著 process 的增加而減少。

三、多執行緒、多行程、協程的效能比較

- Corotine 設計
  - 使用套件：asyncio
  - 方式：使用 get_event_loop()並 create_task()將各個 task 包裝進去，最後在 requests.get 設下 await run_in_executor 讓他在那邊中斷與切換。

- 執行結果

➢ Task 1

	單執行緒	100 個執行緒	100 個行程	協程
time1	122.24437	125.53242	47.427244	124.19102
time2	121.93974	125.94289	49.103679	120.61525
time3	120.19224	125.99407	50.542476	128.11047
average time	121.45878	125.82313	49.024466	124.30558

➢ Task 2

	單執行緒	100 個執行緒	100 個行程	協程
time1	91.484702	6.51481	5.701726	6.538094
time2	87.942572	6.4377	5.54199	6.858862
time3	54.606038	6.579089	4.120386	6.601222
average time	78.011104	6.510533	5.121367	6.666059

- 結果說明

首先先來說一下 coroutine 的原理

Coroutine 是輕量化的 Thread (Light-weight Thread)，他是屬於協同式多工，程式會定時放棄已佔有的執行資源讓其他程式執行，由程式自己讓出執行資源，作業系統不會干涉。而 multi-thread 則是搶佔式多工，作業系統會根據程式的優先權去安排當下哪個程式能有資源去執行。因為 coroutine 之間的切換是在上層，不需要由 OS 來處理，故他的 context switch 負擔會較 mutli-thread 更小。

由結果可以發現，task1 的時間並沒有縮減，因為如果在 multi-thread 那裏說的，task1 是 CPU bound，並且完全沒有 I/O burst，所以 coroutine 的結果跟 multi-thread 相似沒有什麼效果，而 multi-processing 則明顯的可以縮短時間，利用 CPU 資源去縮短運算時間。

而在 task2 在 requests.get(url)的地方會有 blocked，此時 coroutine 就可以切換到其他的去執行，而不用等他，故可以減少執行的時間。而結果顯示 multi-thread 和 coroutine 的時間會比較接近，因為他們的原理比較相同，都是在同一個 process 下去進行切換，而 multi-process 的速度還是最快，因為他能夠使用的資源會是最多的，在不會有太大的 context switch overhead 下，他會是最快的。

四、Reference:

- coroutine 原理：https://ithelp.ithome.com.tw/articles/10235352?sc=iThomeR
- coroutine python：https://docs.python.org/zh-cn/3/library/asyncio-task.html