HW1

學號:0716049

姓名:詹凱傑

Task 1

Thread

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 單執行緒 | 1987.35 | 2449.09 | 2353.608 |
| 雙執行緒 | 1582.86 | 1534.62 | 1820.69 |
| 4個執行緒 | 1500.45 | 1514.98 | 1517.64 |
| 100個執行緒 | 1050.37 | 1169.36 | 1161.97 |

Process

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 單行程 | 1406.25 | 1134.02 | 1113.19 |
| 雙行程 | 632.105 | 528.29 | 568.70 |
| 4個行程 | 404.97 | 306.40 | 383.67 |
| 100個行程 | 189.64 | 211.12 | 188.68 |

協程

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 協程 | 910.66 | 915.89 | 902.83 |

(單位: 秒)

Task 2

Thread

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 單執行緒 | 393.14 | 343.53 | 99.64 |
| 雙執行緒 | 173.67 | 80.77 | 23.94 |
| 4個執行緒 | 86.78 | 35.06 | 17.202 |
| 100個執行緒 | 2.25 | 1.044 | 1.069 |

Process

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 單行程 | 81.67 | 131.48 | 150.51 |
| 雙行程 | 27.53 | 30.24 | 50.03 |
| 4個行程 | 13.63 | 22.02 | 21.97 |
| 100個行程 | 8.64 | 10.06 | 9.35 |

協程

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一次測試 | 第二次測試 | 第三次測試 |
| 協程 | 62.58 | 81.89 | 68.81  (單位: 秒) |

**Python版本: 3.8.1**

**用到的module:**

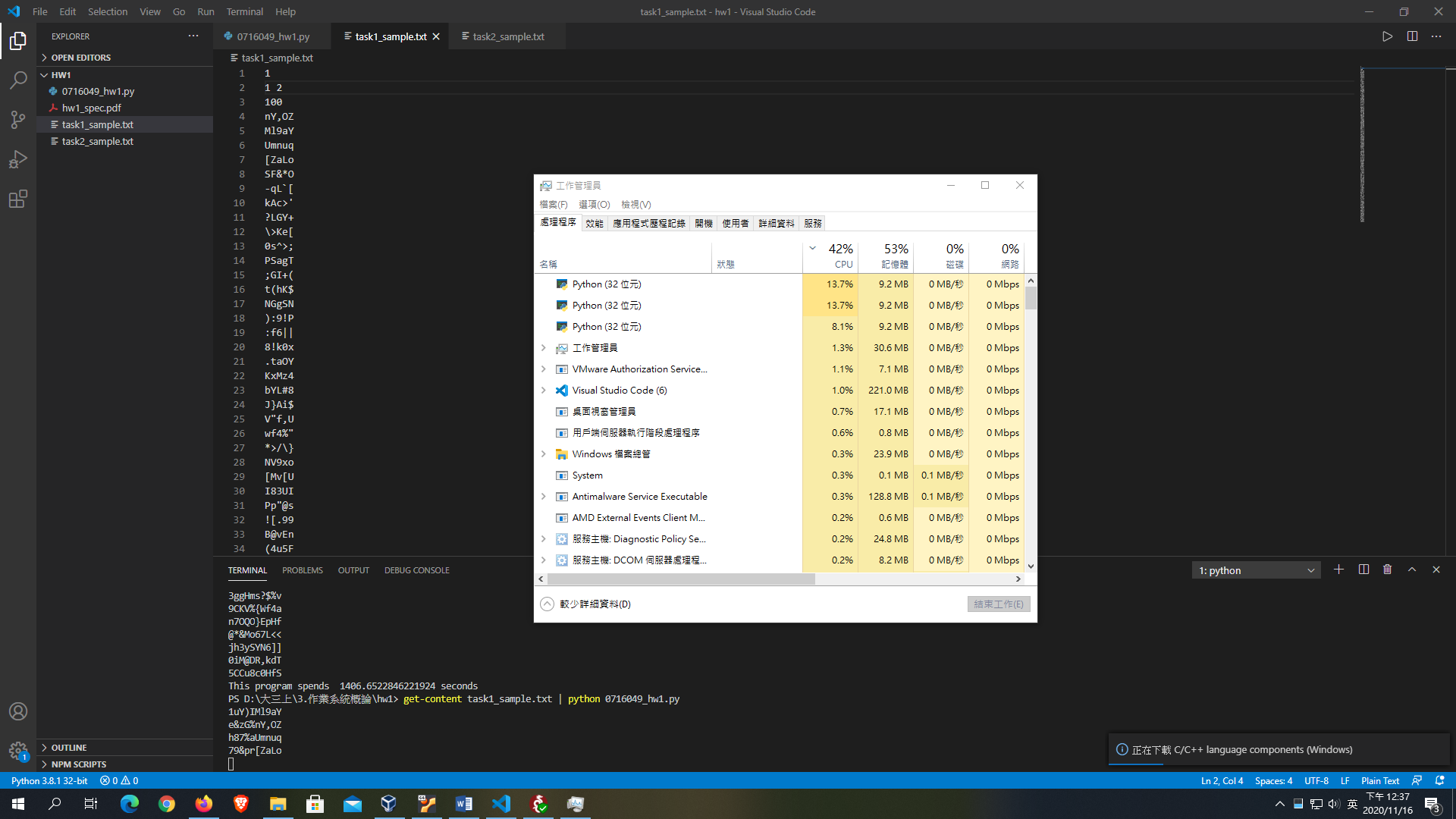
**hashlib, requests, random, time, threading, multiprocessing, asyncio**

1. 執行緒對效能的影響

不管從task1還是從task2來看，增加thread的數量都能夠讓執行時間縮短，但縮短的幅度不太一樣。先從task2來看，送出request後要等到server回應。在一個thread的情況下，送出一個request後，等到server回應後在送下一個request，這樣等待的時間就會是100 個request的時間相加，算是浪費的很多時間在等待，所以如果能用多個thread，在前一個thread送出request然後在等待時，下一個thread就先送出request，這樣可以節省等待的時間，所以等待的時間就會少很多。再來，我有觀察到比較有趣的地方，就是我的結果會越跑越快，例如:第三次的4個thread時就比第一次測4個thread快很多，由於我的測資是用助教給的檔案，且三次都是同一組測資，所以可能是經過幾次request後有一些cache產生，才會讓速度越來越快。

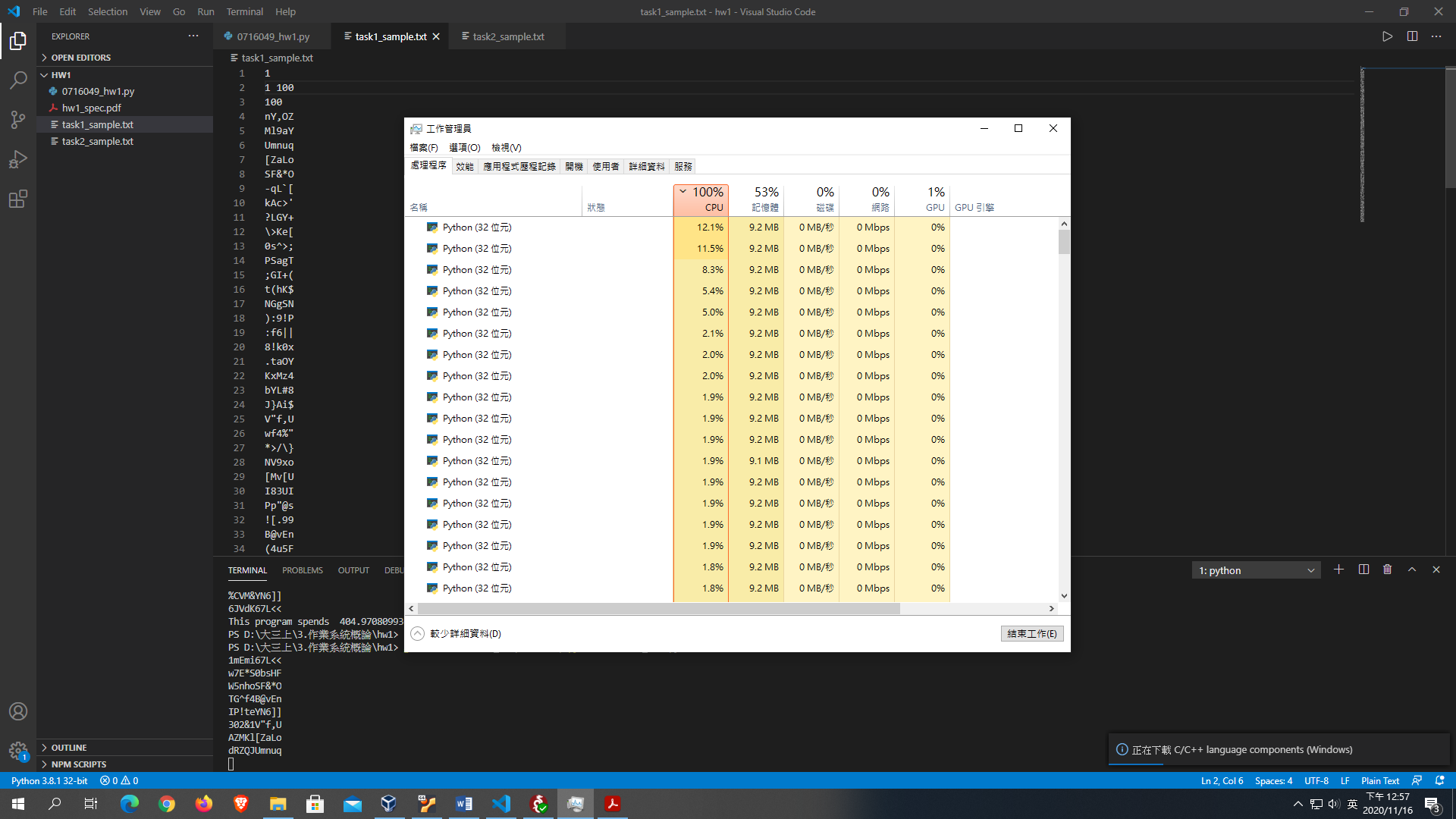
再來會到第一個task，第一個task跟第二的task的性質不一樣，第一個task要cpu大量的運算來得到答案，所以在多個thread的情況下，得到答案的時間會變短，但減少的比例更第二的task比起來就沒有快很多。而我推測可能的原因，是因為不管是多少個thread，要完成task 1 的運算量都是差不多的，所以時間也不會快很多。而在task 1中，我也有注意到一個有趣的地方，由於我的算法是random選一個數，然後看hash完後的結果有沒有符合條件，不符合就random再選一個，直到找到答案，由於這種算法也有點運氣的成分，再加上我們只要測試100筆，看起來很多但從機率的觀點來看，每一次運算還是有很大的運氣成分，這可能是導致相同數量的thread，但每次的結果有些差異的原因。

1. 行程數量對效能的影響

多的process的情況下，趨勢也是差不多的，就是越多process，完成task的時間越短。這邊先討論task1的結果，從直覺來看，一個process分成兩個process，執行時間應該要變一半，然後變成100個 process，則執行時間會變100分之一，但實際上來看不可能變這麼快，從工作管理員來看，

2個process

(1個main+產生的2的process)



100的process

當開到100個processes時，cpu的使用率已經達到100%了，所以到越多thread，增快的比例也越來越少，一開始單個process變兩個process時，有可能真的增快兩倍，但便100個processes時，只有增快十倍。然後在測試這些數據時，由於要跑很久，所以我不太可能就放著電腦等全部的結果跑完，可能還會做一些其他的事，而我發些這似乎也會影響到task要執行的時間，畢竟系統上還有很多其他的process要一起競爭技統的資源，所以當task 1在執行時，若我在電腦上也做一些要複雜運算的操作，就會讓執行時間變更長。

在task 2的部分，第一題有說到兩個task的性質不同，task 1是CPU bound， task 2是IO bound，所以用多個process一起等待，也會縮短總共的等待時間。然後在測試multi process時，有了前面的經驗，可能會有

Cache，所以我是在不同天測試的，除了第一次測試是和multi thread測試時同一天外，另外兩次都是在不同天跑的，或許這樣測出來會更準確。

1. 多執行緒、多行程、協程的效能比較

在第一個task中，得到答案的時間排序是，100個processes最短，再來從結果來看，100個threads跟協程的結果很接近，用協程好像快一點，最後是單個thread最慢。雖然用100個processes跑的效能是最好的，但同時這個方式用到的cpu資源也是最多的，而在CPU bound的process中multi thread和協程都沒辦法達到很好的效果，不過我認為協程可以比100個threads還要快一點的原因在於協程的切換比較快，且切換的overhead也比較小。

第二個task中，屬於IO bound的task，而得到答案的時間排序是，100的threads跟100的processes都很快就求出答案，而100的threads又比100個processes快，再來依序是協程和單個thread。首先是100個threads和100個processes的比較，我認為會造成這個結果的原因是因為新產生一個process所需要的overhead比新產生一個thread大，所需的資源也比較多，所以會造成100個processes所需的時間比較久。再來是協程的部分，雖然協程不是真正的多線程，但協程之間最切換是由程式控制，不會由OS控制，所以切換的速度比較快，再加上這個task是屬於IO bound，所以協程跟單個thread比起來，就變得好有效率。

心得:

由於我之前之有用C++寫過multithread的程式，但沒有用python寫過，所以第一次用Python寫這類型的code覺得還蠻新鮮的，此外，這次的作業每個個別的運算彼此間都是獨立的，所以不用用到Lock，由於我之前寫multithread時，覺得Lock很麻煩，且因為有Lock，沒辦法真的看出multithread的優勢，跑起來都差不多，但在這次作業中，可以明確的看出使用multithread和multiprocessing的優點，這方面我覺得蠻有趣的，且有些結果也有符合上課所學的，讓我對這方面又更了解。最後，我覺得這次的作業還算是蠻有趣的，且我也學到了如何用python寫多執行緒、多核心、協程的程式，算是學到了不少。