OS Project1 說明文件

資訊三乙 10927256 姜美羚

一、開發環境:

作業系統 : win10
CPU : intel i7-13700
IDE : Visual Studio Code
開發語言: python 3.9

二、實作方法&流程:

方法一: 將讀入的 N 筆資料存進 list 中，並直接進行 Bubble Sort

方法二: 先將讀入的 N 筆資料存進 list 中，並將此 N 筆資料分割成 K 份，再建立一個新的 Process，並使用 start()來啟動 Process，將事前被分割成 K 份的資料，先各別進行 Bubble Sort 之後，再進行 Merge，完成排序。結束之後，使用 join()將 Process 關閉。由於對整支程式而言，main 是一個父 Process，而後來建立的 Process 是子 Process，兩個 Process 之間資料不共享，彼此獨立，因此需要 Shared memory。而在本支程式中，所使用的方法是 Manager().list()，來將父 Process 的 list 共享給子 Process。

方法三: 先將讀入的 N 筆資料存進 list 中，並將此 N 筆資料分割成 K 份，再建立 K 個新的 Process，並使用 start()來啟動這 K 個 Process，將事前被分割成 K 份的資料，用這 K 個 Process 各別進行 Bubble Sort。結束之後，使用 join()將這 K 個 Process 關閉。再建立總共 K-1 個 Process 進行 Merge，完成排序。和方法二一樣，main 和這些 process 存在父子之間的關係，因此也需要考慮 Shared memory 的問題。

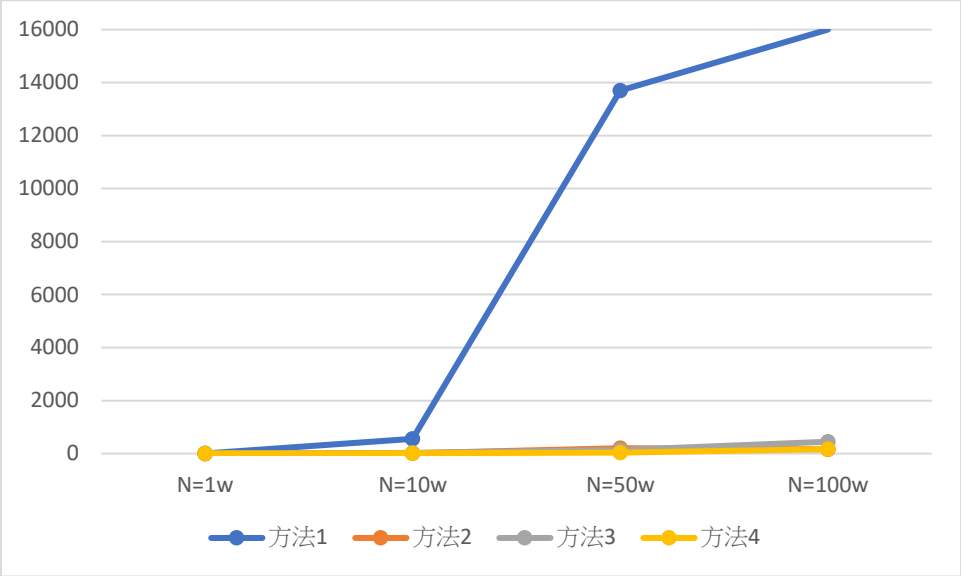
方法四: 先將讀入的 N 筆資料存進 list 中，並將此 N 筆資料分割成 K 份，再建立 K 個 Thread，並使用 start()來啟動這 K 個 Thread，將事前被分割成 K 份的資料，用這 K 個 Thread 各別進行 Bubble Sort。結束之後，使用 join()將這 K 個 Thread 關閉。再建立總共 K-1 個 Thread 進行 Merge，完成排序。

三、**探討結果和原因:**

(一)相同 K 值，不同資料筆數(N)的執行時間: (單位: 秒)

K=1000	N=1w	N=10w	N=50w	N=100w
方法 1	3.6379	552.7866	13701.5758	超過 10 小時
方法 2	0.2857	8.4511	203.2326	156.1779
方法 3	14.5988	21.0695	131.9833	447.3265
方法 4	0.2077	8.1396	37.0561	166.0999

表一、相同 K 值(K=1000)，但不同資料筆數(N)



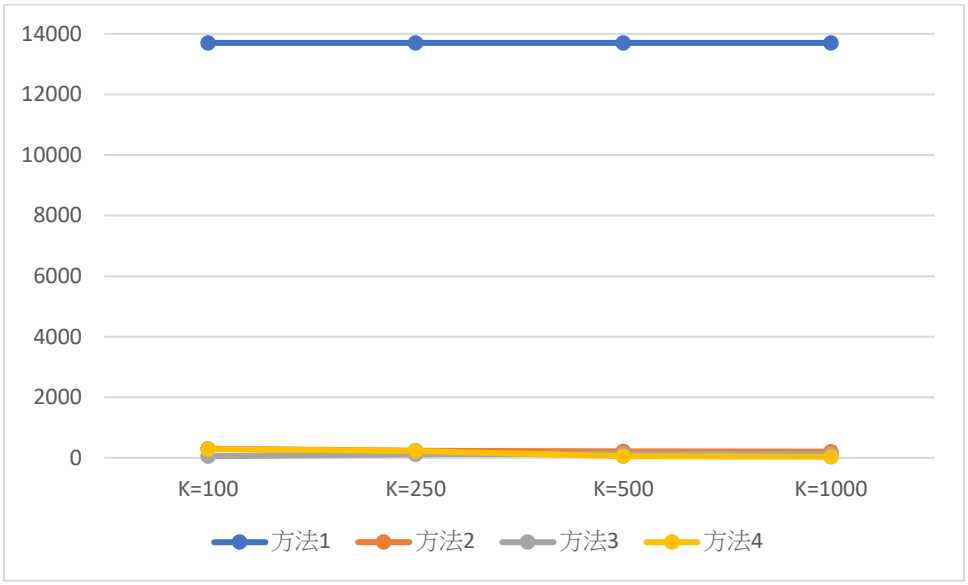
圖一、相同 K 值(K=1000)，但不同資料筆數(N)

由上表和圖可以觀察到，方法一因為只是使用單純的 Bubblesort，所以速度明顯最慢，而方法二三四並沒有明顯的差異。在 50 萬筆資料中，可以觀察到，方法四的速度比方法二三還快，推測是因為方法四因為是使用 thread，而方法二三是使用 Process，而 thread 因為是共用地址，在 context switch 的速度會比 process 快，才導致這種結果。而方法三的速度比方法二快，推測是因為 process 的數量不同所導致的結果。

(二)相同資料筆數(N)，不同 K 值的執行時間: (單位: 秒)

N=50w	K=100	K=250	K=500	K=1000
方法 1	13701.58	13701.58	13701.58	13701.58
方法 2	297.6917	228.8217	211.5991	203.2326
方法 3	52.6779	118.9893	135.6939	131.9833
方法 4	291.2883	220.4477	52.7123	37.05601

表二、相同資料筆數(N=50w)，不同 K 值



圖二、相同資料筆數(N=50w)，不同 K 值

由上圖和表可見。在資料筆數相同的情況下，當 K 值越大，所耗費的時間就會越小。推測是因為當 k 值越大時，單次 bubblesort 所要做的資料量減小，所以速度就變快了。

四、觀察

(1) Multiprocess： 當執行方法三的時候，Process 所新增情況(真的會開出很多 process)。

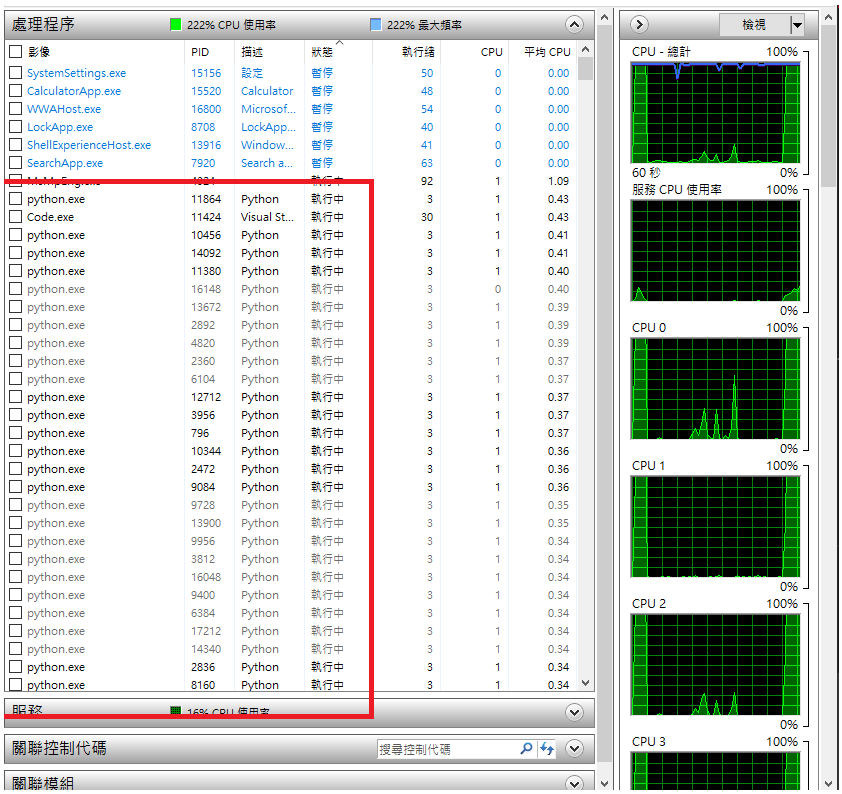


圖 1、從資源監視器觀看 process 產生的數量

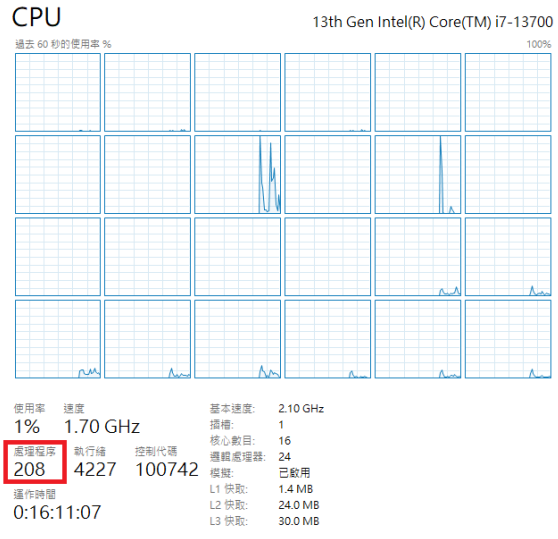


圖 2、執行任務三前

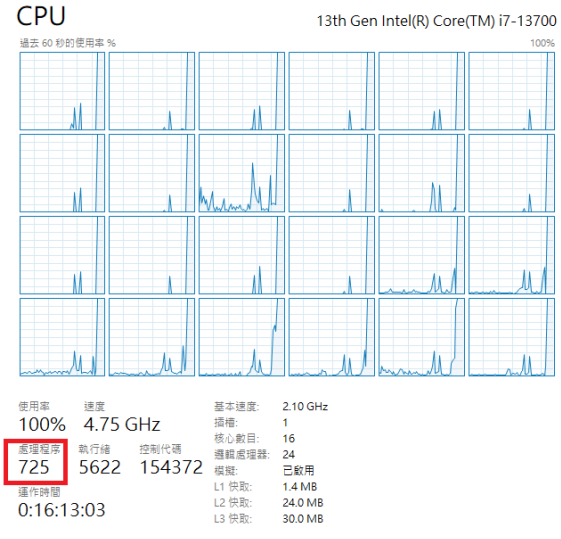


圖 3、執行任務三中

由圖 2 和圖 3 可以看出執行前後 process 數量差異，以及每個 CPU core 的工作量也加大。

(2) Multithread : 當執行方法四的時候，Thread 所新增情況

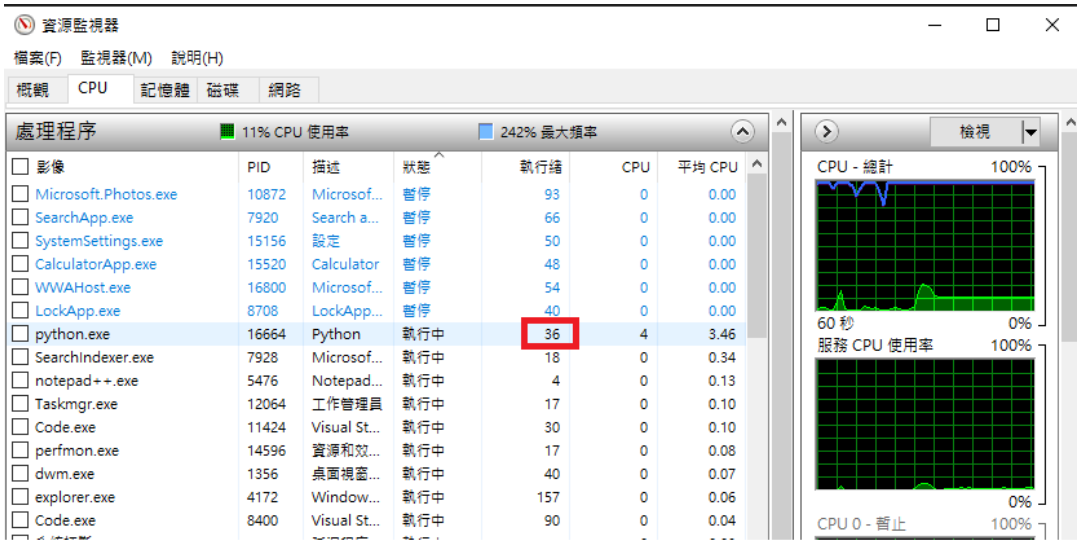


圖 4、從資源監視器觀看 Thread 的數量變化

因為 thread 是增加在同一個 process 中，所以當程式在執行時，方框中的數字會一直變動。