

# 作業系統 OS HW1 multiprocess & multithread

## I.開發環境

資訊三乙 10827234 彭桂綺

開發環境:Windows 11 (21H2)

開發工具:visual studio code [1.66.2(2022.04.11)]

使用語言:C/C++

## II.實作方法及流程

首先我會選擇在 windows 以 c/c++ 實作此次作業是因為我最習慣的環境及語言就是如此,如果遇到 bug 也熟悉如何 debug。

再來分析問題及拆解成小問題,能發現要解決以下問題:

bubbleSort、mergeSort、multiprocess 及 multithread。

- **bubbleSort**

我有改良 bubblesort,在迴圈判斷時加入 flag,在未排序的資料中比對,如果都沒有進行交換位置,則將 flag 設為 false,代表資料已排序好,提早結束排序,以加快 bubbleSort。

- **mergeSort**

我使用 K-way merge,依序比對每個陣列的第一個數,將最小的取出並放進結果陣列中,直到所有數字都被拿完。因為所有陣列都先經過 bubble sort,所以可以保證排在前面的都是最小的數字,因此此方法可行。

- **multithread**

要能實現 multithread ,需要使用 `<thread>` 標頭檔中的 `thread` 函數,將要執行的函數透過此函數執行,也可加入 `refQ` 達到 call by reference 的效果,接下來在下方加入 `join` 函數來等待 子 thread 執行結束才繼續執行接下來的程式碼,以確保後續執行正確。以上函數都有跑迴圈,以 create 多個 thread 。

- **multiprocess**

要實現 multiprocess 需要呼叫 `CreateProcess` 函數。並丟入適當參數來呼叫已經編譯好的可執行檔(需置於同目錄下),因為在 windows 上以 c 實作 multiprocess 不容易,所以細節部分會在下面做說明。以上函數都有跑迴圈,以 create 多個 process。

除了第一個方法直接呼叫 bubble sort 外,其餘方法都是先正確讀入檔案&K,再依照各方法進行 bubble sort,最後執行一次 k-way merge sort,完成。

### III. 特殊機制考量與設計

- **c/c++ 在 windows 處理 multiprocess**

在 windows 無法使用 Linux 上能使用的 `fork()` 函數, 因次需要用到 `<windows.h>` 中的 `CreateProcess` 這個函數來達成 multiprocess。知道要使用這個函數之後就要來解決 multiprocess 帶來的限制。

```
HANDLE * handles = new HANDLE[k] ;  
HANDLE * threads = new HANDLE[k] ;  
STARTUPINFO si;  
PROCESS_INFORMATION pi;
```

以上為必要的參數, `si` 為決定新程序的主窗體如何顯示的 `STARTUPINFO` 結構體、`pi` 接收新程序的識別資訊的 `PROCESS_INFORMATION` 結構體, 兩個參數都需要透過陣列存起來, 以便在跑迴圈時能確保拿到對應的參數。

```
WaitForMultipleObjectsExpand( handles, (DWORD)k); // multiprocess  
// 等待所有子process 結束
```

除此之外也需要利用此函數接收所有的 `pi`, 父 process 才能等到所有子 process 執行結束才繼續執行下面的程式碼。\*\* multiprocess 天生限制一次只能確保最多 64 個 process 在一瞬間只有一個 process 占用 CPU。 \*\*

```
CloseHandle( handles[i] );  
CloseHandle( threads[i] );
```

接下來需要關閉子 process 的 handle, 防止子程序還未執行完, 提前結束。

```
if(CreateProcess("bubblesort.exe", cmdline, NULL, NULL, false, 0, NULL, NULL, &siALL.at(i), &piALL.at(i)))
```

以上為 `createprocess` 所需參數, 第一個參數為呼叫可執行檔的路徑及名稱, 第二個參數為傳入執行檔的參數。

```
int main( int argc, char ** argv ){
```

傳入可執行檔的參數由 `argv` 取用。 `argc` 顯示傳入的參數數量。  
以上即為實作 multiprocess 的機制。

- **multithread 互斥機制**

在 multithread 中 被呼叫的 function 內要加入以下函數, 建立互斥鎖, 確保一次只有一個 thread 進入 Critical Section。在進入 CS 前上鎖, 出 CS 後會自動解鎖。

```
lock_guard<mutex> lock(g_mutex); // crtical section
```

- **防呆機制**

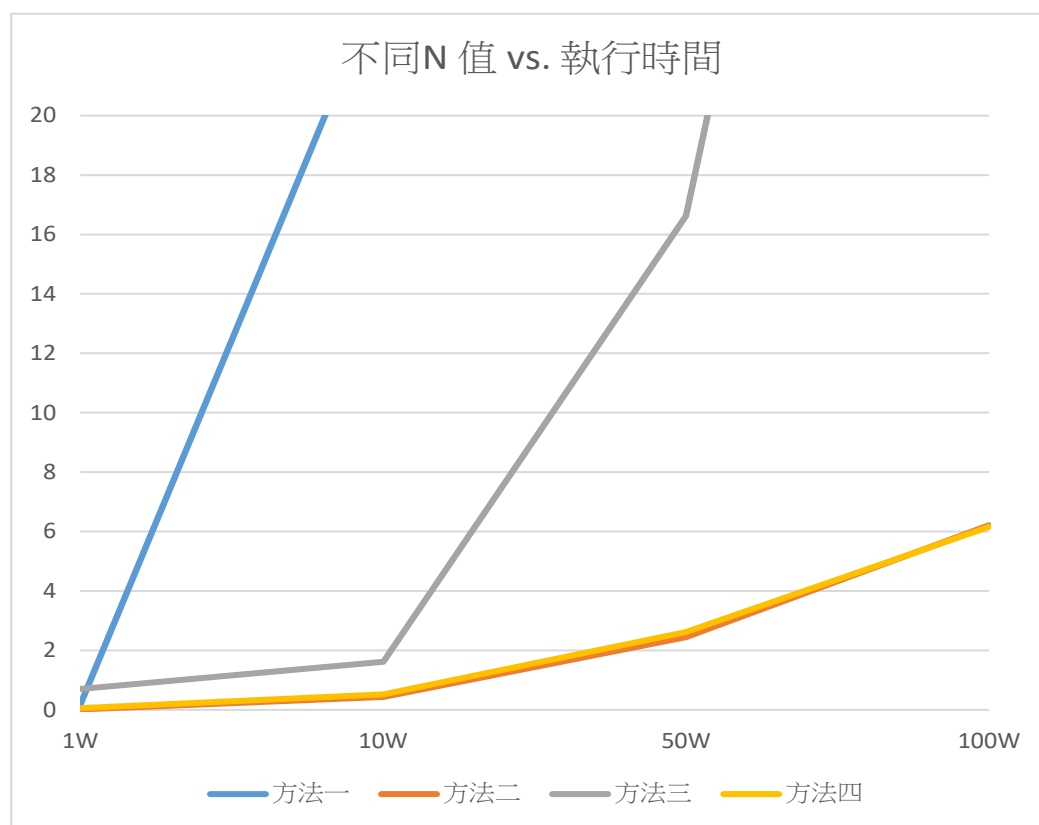
1. 檔案輸入時, 檔名要存在才繼續, 否則持續等待輸入直到正確開啟檔案, 除非使用者輸入數字 0 即跳出, 回到方法選單。

2. K 值不能 = 0 及大於資料大小。

#### IV.分析結果和原因

- [ 不同 N 值 vs. 執行時間 ] 單位: 秒

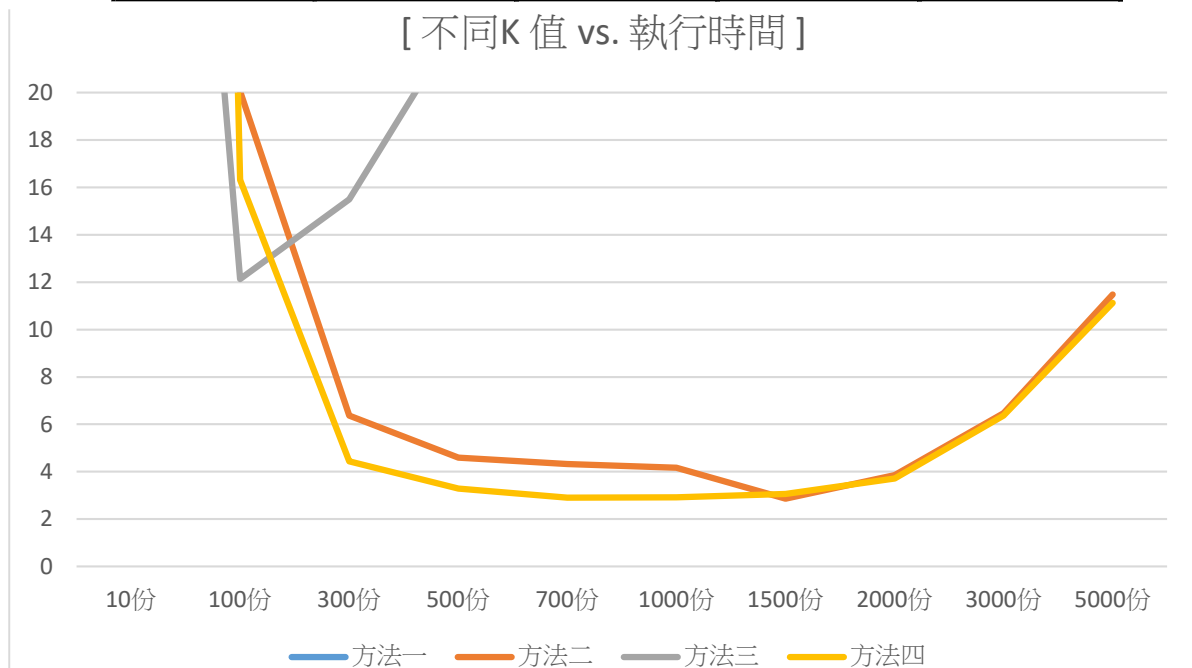
	1W	10W	50W	100W	單位	K
方法一	0.19	24.722	686.633	2670.07	秒	
方法二	0.009	0.429	2.442	6.209	秒	k=1000
方法三	0.702	1.622	16.623	63.881	秒	k=1000
方法四	0.053	0.524	2.617	6.153	秒	k=1000



由上圖可以看出,隨資料筆數的增加,方法 2 及方法 4 執行時間並沒有顯著上升,可見此兩種方法比其他兩種有效率,另外方法 4 並沒有明顯的比方法 2 快有可能是因為在執行方法 4 時,要執行的 thread 太多,可能超過 CPU 一次可以負荷的上限,所以同時能執行的 thread 數會被限制,不會到 1000 這麼多,而將資料量切成小份後 bubblesort 所需執行時間就不會太久,所以 multithread 的優勢就不會明顯,因此方法 2 及 4 的執行時間才會如此接近。另外方法 3 因為需要處理共用記憶體的問題,我選擇以讀寫檔案的方式進行,能降低實作困難,但是因為我沒有做讀寫檔案優化( eg. 以 2 進位讀寫),所以執行時間有大部分都花在 io,不能看到 multiprocessing 所帶來的快速真的很可惜,之後有時間會加強檔案的讀寫,以享受 multiprocessing 所帶來的效益。

● [ 不同 K 值 vs. 執行時間 ] 單位: 秒

100W				
	方法一	方法二	方法三	方法四
10份	2670.07	206.254	67.029	202.822
100份	2670.07	20.059	12.132	16.323
300份	2670.07	6.362	15.498	4.444
500份	2670.07	4.59	22.971	3.287
700份	2670.07	4.321	31.262	2.9
1000份	2670.07	4.175	44.884	2.918
1500份	2670.07	2.867	77.106	3.062
2000份	2670.07	3.857	102.57	3.72
3000份	2670.07	6.462	140.964	6.366
5000份	2670.07	11.482	221.223	11.127



我以 100W 筆資料來測試不同 K 值的執行時間差異,可以看出方法 1 執行時間已經超出其餘方法太多,所以不顯示。方法 2 & 4 的最佳執行時間有一個甜蜜點,大約落在 1500 份左右。方法 3 因為沒有優化檔案讀寫所以在切超過 300 份之後執行時間就會大幅增加,以我的電腦硬體來說方法 3 的最佳執行時間是在 K = 100 時。另外可以發現很酷的一件事就是方法 2 & 4 在切超過 1500 份之後執行時間不減反增,我想可能是因為硬體限制同時能執行的 thread 及 process 數量,超過限制之後就需要頻繁 context switch 而造成執行時間不減反增的狀況,但是在 1500 份以前就能明顯看出 multithread 是比較有效率的。

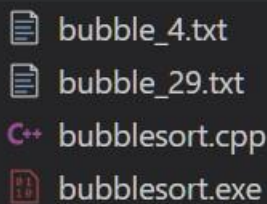
## 其他的發現

方法三在執行時會將 CPU utilization 吃滿,反觀 multithread 在執行時 cpu 的使用率大概就 3 - 5 %,當然,因為所有 thread 都是共用一個 process,所以一個 process 跟多個 process 比起來 CPU 的使用率一定有明顯落差。

## V. 撰寫程式時遇到的 bug(請截圖)及相關的解決方法

- 檔案無法刪除乾淨 (圖中 bubble\_4、buble\_29 未如期刪除)

```
inputFile.close() ;  
remove(fileName.c_str()) ;// 刪掉檔案
```



後來在 multiprocess 之後增加 close handle hThread 即可,但要確保這個 **PROCESS\_INFORMATION** 型別的 hThread 要與放進 createProcess 參數的要一致。

```
threads[i]= piALL.at(i).hThread; // 增加此行
```

```
CloseHandle( threads[i] ); // 增加此行即可
```

- multithread 無法如期執行

```
// 用 thread 執行 bubble sort function 並用 ref() 函數 達成 call by reference  
//threads.push_back(thread(threadNewBubbleSort, std::ref(allSorted.at(x))));  
} // for  
  
// 從上面的寫法拉下來獨立迴圈就可以了?????????  
for ( i = 0; i < k; i++) {  
    threads.push_back(thread(threadNewBubbleSort, std::ref(allSorted.at(i))));  
} // for
```

如圖所示,將呼叫 **thread** 的那行移到下方獨立回圈內就可以了。實際原因不明。

- createProcess 參數型別不相符

```
LPTSTR cmdLine = (LPTSTR) tempParm; // 型別轉換
```

```
LPCTSTR exeName = (LPCTSTR)temp.c_str(); // 型別轉換
```

強制型別轉換後即可。