404410016 資工三 宋慶澔

# 作業系統 作業二 報告

## 題目敘述

You are asked to implement a simple sorting program with multiple threading.

- Create several threads
- Let multiple threads sort the data

程式名稱：my_sort
使用方法: my_sort rand_seed data_size (例如：my_sort 15 10000)

將 rand_seed 餵給 ANSI C 的 srand()
使用 ANSI C 的 rand()產生 data_size 個數字，將這些數字由小到大排序，
加總排序後的數字間距，例如「2, 4, 9, 18, 26」，加總後為 2+5+9+8=24，
並將結果（輸出）印出至 stdout，範例的結果應印出 24。

## Q1 如何利用平行化提高運算速度

平行化的概念就是把一份工作分割成數個部份並交給不同的人做藉此達到提昇效率的目的。假如打掃一間房間需要完成掃地和擦窗戶，兩個人一起(一人掃地、一人擦窗戶)通常會比一個人(自己掃地、擦窗戶)還要來的有效率。但有一個很重要的前提是，這份工作必須是「可以被分工」的，這樣多工才會有效益。舉例來說，如果今天打掃房間要完成的工作是掃地和拖地，而拖地前要先完成掃地，如此一來，就算一人負責掃地、一人負責拖地，也不會比較有效率，因為拖地的人要等前面的人掃完地之後才能開始工作，這樣就是屬於不可被分工的工作(工作間有前後關聯)的例子。

而題目中的排序以及計算間距剛好都是屬於可以分工的範疇，例如排序陣列，可以由兩個人一人排一半，最後在合併結果；計算間隔也可以一人排一半，最後在加總結果，在核心數夠多且不考慮CPU壅塞的情況下，K線程的執行時間理想上應為單線程的K分之一左右，**本次作業我將以雙線程(K=2)來實做。**

## Q2 如何確保多個 thread 間不會有 race condition

只要不同時對同一地方進行寫入就不會產生race condition。這次作業中排序與計算間距的部份都是可以分工的(有點類似分治Divide and Conquer的概念)，上面這些過程都不會有race condtion的情況發生，唯獨在合併thread的計算結果的時候有可能會有多人同時寫入同一變數的情況發生，面對這樣的狀況，我們要讓寫入全域(共用)變數的次數越少越好(越少次lock and unlock)，所以在計算間距的函式中，會有一個local 變數計算 local 的統計結果，最後在一次寫入共用記憶體之中(只進入critical section一次)，在寫入之前把 critical section lock住，寫完之後 unlock，如此一來可以確保正確性，又不會因為一直頻繁的上鎖、解鎖導致整體執行效能低落。

在統計數字間距的實做上，我是採用mutex lock，為了讓lock跟unlock以及進入critical section的overhead越少越好，在thread function中會有一個local變數先紀錄著loal端的結果，在返回前在一次把統計結果寫完全域(共用)變數之中(如下圖所示)

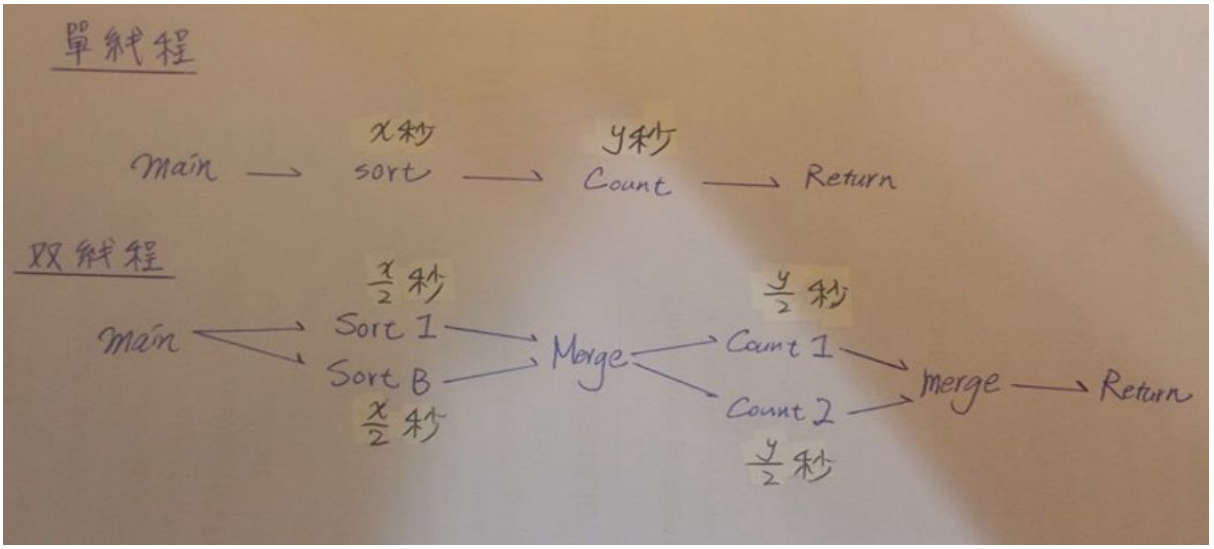
*下圖的 p -> ret 是一個指向共享記憶體的指標*

```
// pthread_diff 的呼叫函式
void pthread_diff(void *ps)
{
  int idx_i, local = 0;
  ds_pthread *p = (ds_pthread *) ps; // 先強轉一波

  for(idx_i = p->l + 1; idx_i <= p->r; idx_i++)
    local += (p->arr)[idx_i] - (p->arr)[idx_i - 1];

  pthread_mutex_lock(p->mutex);
  *(p -> ret) += local; // critical section
  pthread_mutex_unlock(p->mutex);
}
```

**Q3 使用圖形說明你的程式碼是很有效率的，與簡述使用的演算法(如果有用到的話)或分割、合併的方式，可搭配 mysort, mymerge 等函數名稱說明。**




從上圖可看出原本單線程完成排序以及完成計算間距個別需要X秒以及Y秒，透過雙線程去分工，理想狀況下可以讓排序以及計算的時間變成原來的一半，也就是0.5X秒以及0.5Y秒，藉此來達到提昇效率的目的。

*Sort我是自己實做QuickSort，每次選用最右邊的元素當作pivot，遞迴完成排序。*
*平均時間複雜度 O(nlgn) 比c qsort 慢一點點點*

```c
// inline 的 swap 函式
__attribute__((always_inline)) inline void SWAP(int *x, int *y)
{
  int temp = *x;
  *x = *y;
  *y = temp;
}


// 自訂義的 quick sort, 每次選最右邊的當作 pivot
void quickSort(int *arr, int lbnd, int rbnd)
{
  if(lbnd >= rbnd)
    return;

  int pivot, pidx, i;
  pivot = arr[rbnd];
  for(pidx = lbnd, i = lbnd; pidx < rbnd && i < rbnd; i++)
  {
    if(arr[i] <= pivot)
    {
      SWAP(&arr[pidx], &arr[i]);
      pidx++;
    }
  }
  SWAP(&arr[pidx], &arr[rbnd]);
  quickSort(arr, lbnd, pidx - 1);
  quickSort(arr, pidx + 1, rbnd);
}
```

*至於Merge的部份，因為是兩個有序陣列，所以用兩個指標個別從頭走到尾就可以完成合併了。時間複雜度O(N)。*

```c
// 合併兩個 sorted array
void merge(int *arr, int size, int l, int m, int r)
{
  // merge 需要一塊跟 arr 一樣大的空間
  int *temp = (int *) malloc(sizeof(int) * size);
  if(temp == NULL)
  {
    fprintf(stderr, "Malloc error.\n"); // malloc 失敗
    exit(1);
  }
  int i, j, k;
  for(i = l, j = m + 1, k = l; i <= m && j <= r;)
  {
    if(arr[i] < arr[j])
      temp[k++] = arr[i++];
    else
      temp[k++] = arr[j++];
  }
  while(i <= m)
    temp[k++] = arr[i++];
  while(j <= r)
    temp[k++] = arr[j++];
  for(i = l; i <= r; i++)
    arr[i] = temp[i];

  free(temp);
}
```

## 執行結果（排序1000萬個整數）

*從real跟user的time可以看出，雙線程的分工比起單線程大約能提昇一半左右的效率。*

**localhost**

```
howard@howard-PE70-2QE:~/SystemProgramming/multi-sort$ make
gcc -pthread multi-sort-and-diff.c
howard@howard-PE70-2QE:~/SystemProgramming/multi-sort$ ./a.out 10 10000000
Total difference: 2147482866
howard@howard-PE70-2QE:~/SystemProgramming/multi-sort$ time ./a.out 10 10000000
Total difference: 2147482866

real    0m1.091s
user    0m1.976s
sys     0m0.016s
howard@howard-PE70-2QE:~/SystemProgramming/multi-sort$
```

**mcore8.cs.ccu.edu.tw**

```
sch104u@mcore8[6:37pm]~/multi-sort> gcc -pthread m
makefile                multi-sort-and-diff.c
sch104u@mcore8[6:37pm]~/multi-sort> gcc -pthread multi-sort-and-diff.c
sch104u@mcore8[6:37pm]~/multi-sort> time ./a.out 10 10000000
Total difference: 2147482866
4.152u 0.040s 0:02.37 176.7%    0+0k 0+0io 0pf+0w
sch104u@mcore8[6:37pm]~/multi-sort>
```