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

試想一個陣列裡有一千萬個數字要排序，若是把陣列切一半，分別丟到兩個線程去，排序完成之後會得到「兩個有序陣列」，在用一線性時間把這兩個有序陣列合併在一起，完成排序，在完成排序之後，一樣在透過兩個線程去計算數字的間距、輸出答案。在順利的情況下，雙線程所需要的時間應該為單線程的一半(核心數足夠多)。而多線程(n=k)的概念與雙線程(n=2)的概念一樣，本次作業將用雙線程實做。

Q2 如何確保多個 thread 間不會有 race condition

只要不要同時對同一個地方進行「寫入」，就不會有race condition。在排序的部份，因為QuickSort 本身就是分治(Divide&Conquer)的一種演算法，所以透過多工來執行是完全沒問題的。比較需要注意的是在計算、加總數字間距的時候，因為是修改同一個變數，所以可能產生race condition，面對這樣的狀況，我們要讓寫入變數的次數越少越好（越少次 lock、unlock），所以在計算間距的函式中，會先有一個 local 變數計算 local 的結果，最後在一次寫入共用記憶體之中，在寫入之前把 critical section lock住，寫完之後 unlock，如此一來可以有正確性，又不會因為一直頻繁的上鎖、解鎖導致整體執行效能低落。

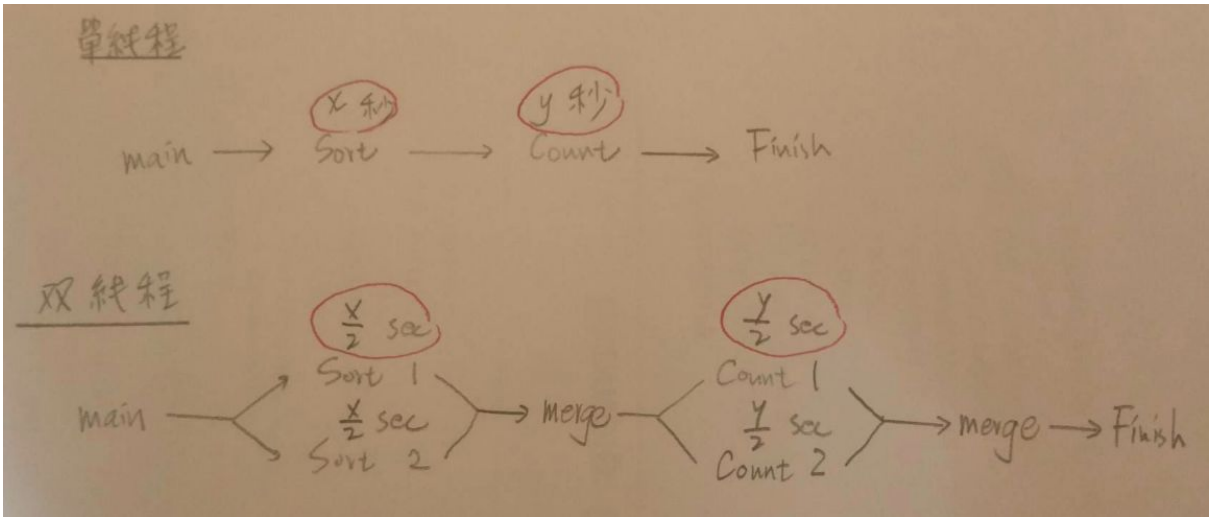
下圖的 p -> ret 是一個指向共享記憶體的指標

```
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、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x/2秒、y/2秒。藉此來達到提昇效率的目的。

在排序的部份，使用QuickSort，時間複雜度平均為O(nlogn)，以下是該演算法的虛擬碼：

```
Pseudo Code

void quickSort(A, start, end)
{
   if(start < end)
   {
     pivot <- A[end];
     for(pidx = i = start; pidx < end and i < end; i++)
     {
       if(A[i] <= pivot)
       {
         SWAP(A[i], A[pidx]);
         pidx++;
       }
     }
     SWAP(A[pidx], A[end]);
     quickSort(A, start, pidx - 1);
     quickSort(A, pidx + 1, end);
   }
```

而在merge的部份，時間複雜度為O(N)，只用兩個指標從頭跑到尾一次，即可完成merge。

```c
void merge(int *arr, int l, int m, int r)
{
  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];
}
```

執行結果（排序1000萬個整數）

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>
```