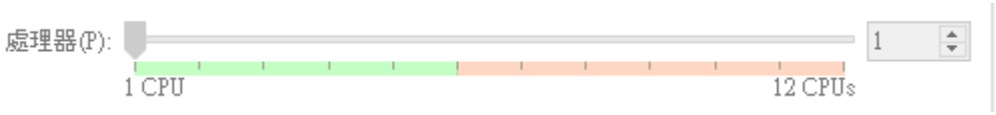
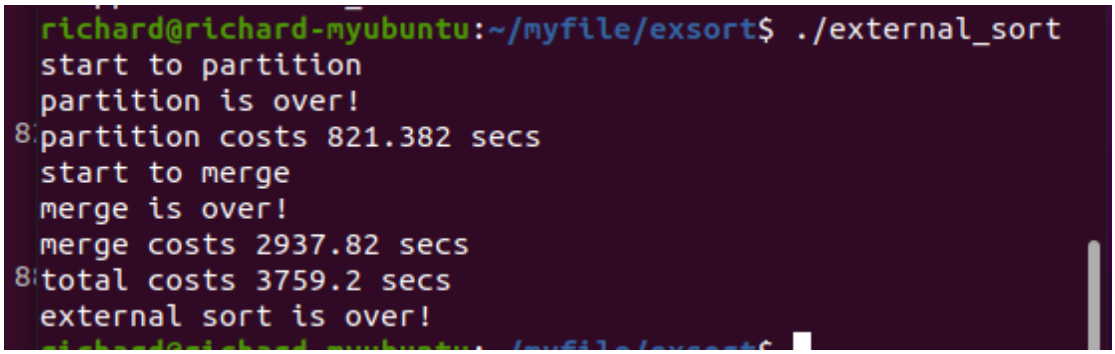
學號:E94074029
姓名:江羿賢
系級:資訊系 111 級

開發環境:
OS: Ubuntu 20.04(在虛擬機環境下)
CPU: Intel® Core™ i7-8750H CPU @2.20GHz (分配給虛擬機 1CPU)
Memory: 4GB(實際上能夠使用的只有 3GB 再多一些)
Programming Language: C++ (gcc version 9.3.0)
備註:所使用的 cpu 設定如下圖(使用 1CPU)



程式執行時間: (跑 5.6GB 的 input.txt)
3759.2 秒



程式開發與使用說明:
程式開發:
在要排序大檔案的過程中，我所使用的是 merge sort 的一個方法，先把 input.txt 讀取，並把其拆成數個 partition，而每個 partition 的部分中，因為之後考慮之後要用到多隻程式同時運行的時候，所以將每一個 partition 拆成大約 1GB 多來使得在一定數量的程式運行的時候能夠成功運行，而每個 partition 將會被拆出成 partition1,partition2….視 input 來決定數量，並把他存在與其程式碼相同的目錄底下，再最後把那些 partition 合併起來成 output.txt 輸出來。以上是算是比較大概的一個流程，再來可以看到比較一些細微的部分，在對那一些檔案去做處理的時候，必須使用一個動態配置的指標去指著，不然會不知道讀到哪裡去，而也要去記下來檔案有多大並且去算出要分成幾個檔案再來去做配置，然後當每個 partiton 寫滿了之後，要跳到下一個 partition 繼續寫，並要在 input 都寫入 partition 之後要再確認一下是不是剛好寫完了，如果不是的話要記得把最後一個寫不滿的那個檔案也要去 sort，而如果是剛好寫完的話也要記得

把多算的 partition 的數量還原回來，而在這個過程中有一個值得注意的地方是，如果將每個 partition 分的太小的話，而 input 又具有一定的大小的時候，將會產生很多個 partition 檔，因此也需要 new 出很多個指標去指著，有可能會產生不夠使用的問題，將會導致 error，所以每個 partition 檔案不能夠太小以免造成問題，而 partition 完了之後要進入到的就是 merge 的一個環節了，這邊所選擇利用 priority_queue 來當作主要的一個結構，並且利用 compare 的 function，來讓其像是 minheap，並且利用其資料結構來使得我方便的來去做 merge 成一個 output，利用其資料結構的內建 function 來達到，然後可以利用.size()的這個函式來判斷出這個裡面要不要繼續做下去，並且在最後的部分記得要把之前 new 出的把其 delete 回去，還給 memory。而在這支程式裡面也使用了<ctime>來做為計算時間的工具，並將其分成去紀錄到 partition 以及 merge 分別花了多少的時間，而也記錄下總共花了多少時間。

使用說明:

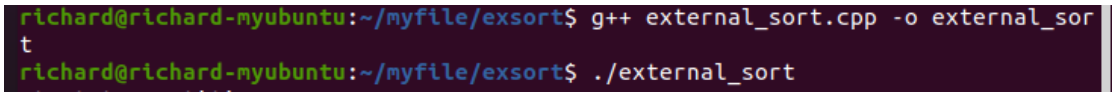
打開終端機

使用以下指令編譯

g++ external_sort.cpp -o external_sort

使用以下指令執行

./external_sort

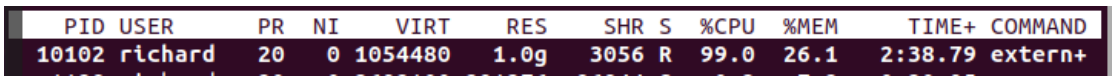


或者如上圖所示

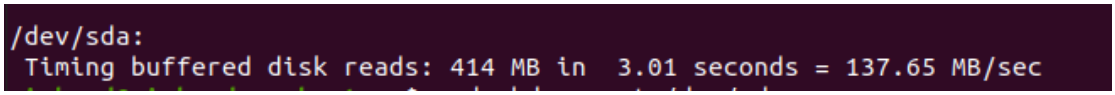
效能分析報告:

先來觀察當一支程式運行時，所使用的 CPU 以及 memory 以及 disk I/O

在 partition 的過程中，CPU 的運行大約都在 99%,而 memory 都約在 26%左右(如下圖)

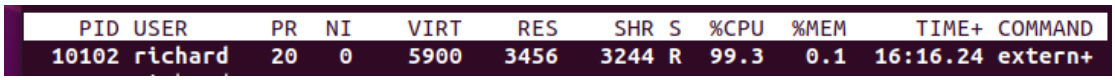


而 disk I/O 的量大約在 138MB/sec(如下圖)



表示了 CPU 都是用滿的，而 memory 在我的虛擬機設定的是 4GB，而 4GB*26%大約是 1GB，也跟我在程式中所要 partition 每個檔案的大小相近，而最終 partition 的部分耗時 821.382 秒。

而在 merge 的部分，CPU 的運行大約都在 99%,而 memory 約在 0.1%(如下圖)



而 disk I/O 的量大約在 85MB/sec(如下圖)

```
/dev/sda:
 Timing buffered disk reads: 256 MB in  3.00 seconds =  85.29 MB/sec
```

這邊的運作在 CPU 上也是吃的很滿，而 memory 在這邊的使用量是相當低的，但是在 disk 的 I/O 處理上也是很可觀，而 merge 的部分耗時 2937.82 秒。
總共耗時了 3759.2 秒(如下圖)

```
richard@richard-myubuntu:~/myfile/exsort$ ./external_sort
start to partition
partition is over!
partition costs 821.382 secs
start to merge
merge is over!
merge costs 2937.82 secs
total costs 3759.2 secs
external sort is over!
```

再來觀察當兩支程式同時運行時，所使用的 CPU 以及 memory
在 partition 的過程中，兩支程式的 CPU 的運行大約都在 49.5%,memory 都約在 26%(如下圖)

```
ID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
73 richard   20   0 1054480   1.0g   3240 R  49.5  26.1   2:25.19 external_sort1
74 richard   20   0 1054480   1.0g   3100 R  49.5  26.1   2:22.53 external_sort2
```

而 disk I/O 的量大約在 136MB/sec(如下圖)

```
/dev/sda:
 Timing buffered disk reads: 408 MB in  3.01 seconds = 135.76 MB/sec
```

在這邊也是每一支程式大概就是一人一半的 CPU 用量，CPU 用量也都是吃滿的，而 memory 的用量也都大概是 4GB*26%大約是 1GB，也就是跟我 partition 的檔案大小相近，而最終 partition 的部分分別花了 839.135 秒跟 838.959 秒。

而在 merge 時，CPU 的運行大約都在 49.8%,而 memory 約在 0.1%(如下圖)

```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 10673 richard   20   0    5900   2052   1840 R  49.8   0.1  29:58.21 external_sort1
 10674 richard   20   0    5900   2072   1864 R  49.8   0.1  29:58.19 external_sort2
```

而 disk I/O 的量大約在 82MB/sec(如下圖)

```
/dev/sda:
 Timing buffered disk reads: 246 MB in  3.01 seconds =  81.72 MB/sec
```

兩支程式在 CPU 的使用上也幾乎都是全部的各一半，也把 CPU 都吃滿了，而 memory 也都只用了 0.1%，而 disk I/O 的量也跟只跑一支程式的情況差不多，而在 merge 的部分分別花了 2988.66 秒以及 3065.5 秒。
總共耗時了 3827.8 秒以及 3904.45 秒(如下圖)

```
richard@richard-myubuntu:~/myfile/exsort1$ ./external_sort1
start to partition
partition is over!
partition costs 839.135 secs
start to merge
merge is over!
merge costs 2988.66 secs
total costs 3827.8 secs
external sort is over!
```

```
richard@richard-myubuntu:~/myfile/exsort2$ ./external_sort2
start to partition
partition is over!
partition costs 838.959 secs
start to merge
merge is over!
merge costs 3065.5 secs
total costs 3904.45 secs
external sort is over!
```

接下來觀察 3 支程式同時運行，所使用的 CPU 以及 memory
在 partition 的過程中，三支程式的 CPU 的運行大約都在 32~33%,memory 都約在 26%(如下圖)

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1636 richard   20   0 1054480   1.0g   1128 R  33.4  26.0   2:50.59 external_sort2
 1635 richard   20   0 1054480   1.0g    988 R  32.5  26.0   2:53.96 external_sort1
 1637 richard   20   0 1054480   1.0g   1012 R  32.1  26.0   2:48.32 external_sort3
```

而 disk I/O 的量大約在 135MB/sec(如下圖)

```
/dev/sda:
 Timing buffered disk reads:  406 MB in  3.02 seconds = 134.59 MB/sec
```

在這邊的情形每一支程式大概就是每支程式約 1/3 的 CPU 用量，CPU 用量也都是吃滿的，而 memory 的用量也都大概是 4GB*26%大約是 1GB，跟程式 partition 的檔案大小相近，而最後 partition 的部分分別花了 814.303 秒跟 814.396 秒跟 816.668 秒。

而在 merge 時，CPU 的運行大約都在 32~33%,而 memory 幾乎沒有在使用(如下圖)

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1635 richard   20   0    5900   1612   1404 R  33.2   0.0  20:56.66 external_sort1
 1637 richard   20   0    5900   1704   1492 R  33.2   0.0  20:47.21 external_sort3
 1636 richard   20   0    5900   1780   1572 R  32.9   0.0  20:52.15 external_sort2
```

而 disk I/O 的量大約在 80MB/sec(如下圖)

```
/dev/sda:
 Timing buffered disk reads:  242 MB in  3.03 seconds =  79.95 MB/sec
```

三支程式在 CPU 的使用上也幾乎都是全部的各 1/3，也把 CPU 都吃滿了，而

memory 也幾乎沒有用到，而 disk I/O 的量也跟只跑一支跟兩支程式的情況差不多，而在 merge 的部分分別花了 2977.33 秒跟 2986.83 秒跟 2972.68 秒。
總共耗時了 3791.63 秒跟 3801.22 秒跟 3789.35 秒(如下圖)

```
richard@richard-myubuntu:~/myfile/exsort1$ ./external_sort1
start to partition
partition is over!
partition costs 814.303 secs
start to merge
merge is over!
merge costs 2977.33 secs
total costs 3791.63 secs
external sort is over!
```

```
richard@richard-myubuntu:~/myfile/exsort2$ ./external_sort2
start to partition
partition is over!
partition costs 814.396 secs
start to merge
merge is over!
merge costs 2986.83 secs
total costs 3801.22 secs
external sort is over!
```

```
richard@richard-myubuntu:~/myfile/exsort3$ ./external_sort3
start to partition
partition is over!
partition costs 816.668 secs
start to merge
merge is over!
merge costs 2972.68 secs
total costs 3789.35 secs
external sort is over!
```

但是我認為這邊在計算時間上是有誤差的，因為我在現實世界中的時間並不是一個小時多，而大約是兩個多小時，而我認為主要的誤差在於虛擬機上有時候會有點卡住，而在程式即將運行完畢時，使用 top 指令所看到的 Time+是一個多小時，推測應該是 memory 快要不夠用所導致虛擬機的程式有時候會卡，所以 c++裡面的 clock 測不準其時間的變化，因此在接下來的討論中比較會偏向於同時跑第三支程式花了比程式所偵測到的還要更久的時間。

在同時運行 4 支程式之後
而因為每一個 partition 都是約為 1GB 多，而 RAM 只有 3GB 多而不到 4GB，所以在同時運行個一陣子之後，虛擬機就卡住不動了，有可能是 memory 已經不夠來提供了，程式也難以被提供資源來繼續去運行，而進一步再嘗試同時運行 5 支以及 6 支程式，也出現了卡住不動的結果，也一樣只能強制關閉重開，而在它們死機的速度都是差不多的，應該是 disk I/O 最多能寫入的速度是固定的，才會導致 memory 不夠用的時間也是差不多，而導致死機的時間差不多的

這個結果。

在不論是幾支程式同時運行的情況當中，CPU 都是會被耗光的一個項目，而在只有一隻程式運行的情況當中，作業系統是將 CPU 幾乎全部都服務給了 external_sort 來做排序，幾乎不管再任何時刻當中 CPU 都持續的用好用滿，而在有兩隻程式同時執行的情況下，CPU 雖然被分成給兩個程式使用，但是執行並沒有慢很多，我認為主要有兩個可能的原因所造成，第一個原因是作業系統有幫我們做好類似於 pipeline 的功能，讓 CPU 不會有太多時間是在做一些沒有效率的使用，而第二個原因有可能出在於 disk I/O 在部分過程中需要有非常大量的讀取以及寫入，而在有一些部分是沒有那麼吃的，所以電腦有可能在這邊幫我們做好一些事情，讓資源分配的平衡一點，不會讓 disk I/O 有閒置的機會太久。相較於同時執行 3 支程式，同時執行三支程式相對前面兩個 case 慢了不少的一個執行時間，我認為主要的原因是因為在 CPU 跟 memory 上面有問題，因為在外部我所在的時間跟 ubuntu 虛擬機上所計算出來的時間並不是一致的，而是差了一些，原因比較高的可能是在因為 memory 已經快要到極限了所以有時候會讓整個虛擬機不太能動，而 CPU 這邊也有可能因為有 3 支程式同時運行，讓整個 pipeline 沒辦法發揮到像是同時執行 2 支程式那麼好的功效，所以會開始拖慢整個程式，而在 disk I/O 中有時候也會遇到寫入速度不夠快的問題而導致要等，可能有時候 CPU 準備計算時資料還沒好，也有可能在內部 sort 的時候，有些資料已經匯入了，但是 CPU 還在做其他支程式的 sort 導致沒有辦法發揮那麼好的效率，以上種種原因皆有可能造成執行時間變慢，而可能的原因應該是多到數不清，所以只有列出一些主要比較有可能會大幅度影響執行時間的一些原因。而接下來討論一下在我這個虛擬機環境上所遇到的一些問題分析，在這個虛擬機上因為我只有分配給它我本身電腦的 1/12 的一個計算能力，而 memory 則是給了算是滿大的 4GB，而硬碟的寫入時間理論上應該是差不多太多的，而我認為之所以沒有見到 disk I/O 明顯拖累整支程式運行的情況是因為我的 CPU 給虛擬機太少了，導致說讓 CPU 其實常常是一個滿載的一個情況，而 memory 在這個環境之中始終也是夠用的，因為在不夠用的情況應該是非常容易導致虛擬機死機，造成程式無法順利完畢執行，但是在這裡我們不討論這類的情形，但是如果不是為了要測試說同時運行幾支程式的話，應該要盡可能的開放 memory，使得程式的 partition 可以一次大一點，讓檔案數量變小減少負擔，讓內部 sort 的部分多一點交給 CPU 去執行，而在虛擬機上比較沒有出現跟我原本預期的結果原因在於說，這個的一個整體 CPU memory disk 的權重配置，不太像是一般電腦會提供的一個配置，導致會跟我預期的結果有落差，但依照比較實際的角度來看的話，我認為 disk I/O 在這一類的執行上面會拖得比較嚴重。而在硬體的優化上面的話，應該要使用更快的 CPU，更大的 memory，以及讀寫更快的 disk，而在程式的優化上面，已經做了一些有效率的資料結構來達到其目的，如 heap，以及使用效率很好的內部 sort，來使得時間

複雜度降低，這些已經在演算法有證明過說比可能 bubble sort 或者是一些僅僅使用 array 來土法煉鋼來的快，而這些在現有的演算法已經有證明過會比較快了，礙於篇幅不在這邊多作證明，這些的演算法可以幫助其減輕 CPU 的消耗。而如果要做其他的優化的話要看哪些地方可以做一個有效的 pipeline，並且如果可以在程式碼裡面塞入一些可以查看 memory 用的情況的 code，使得讓 partition 的 file 的大小不一定是要固定的也可以是動態的，只不過這個感覺比較困難去實作一些，但是概念就是要盡可能的用滿 memory 來做，而如果是一個在確定沒有其他程式會占用的 memory 的話，就讓 memory 快接近 ram 的大小應該是最棒的一個情況，但如果是有可能會有別支程式需要讓自己借過一下的話可能就要再調整一些 memory 的用量以及每個 partition 檔案的大小，而在 disk code 的優化的部分其實在那個時候不會占用到很多的 memory，所以也可以在最後一個檔案讀取完畢的時候讓其不要被清除掉，而是繼續留在 memory 當中，可以當要輪到該數值要進入 output 的時候能夠比較快的從 memory 取得，而不用再耗時的 I/O，而在 CPU 的部分的話，如果有到多核心的話可以考慮加入一些平行運算的 code，使得比較好做一個執行上的分工。希望有朝一日我有足夠的能力去讓更多程式有效率的執行讓世界變得更好，更有效率。