2020_OS_Fall_HW1: Benchmark Your Computer Black Box

學號 : F14076083
姓名 : 魏湧致
系級 : 資訊 111

開發環境 :

- 使用 Virtual Machine
- OS : Ubuntu 18.04.2 LTS
- CPU : Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 2
- Memory : 3.9 GiB
- Programming Language(version) : C++ 11

程式執行時間 :

- 程式中若設定 memory 為 4GB 會產生 out-of-memory 的錯誤，故設定 3GB，而測資為 4.9 GB (49,4217,3405 bytes)的檔案
  使用 time 指令得到執行時間
  優化前 :
  real 52m14.619s
  user 23m42.647s
  sys 28m12.328s
  優化後 :
  real 18m14.153s
  user 17m0.396s
  sys 0m32.009s

程式開發與使用說明：

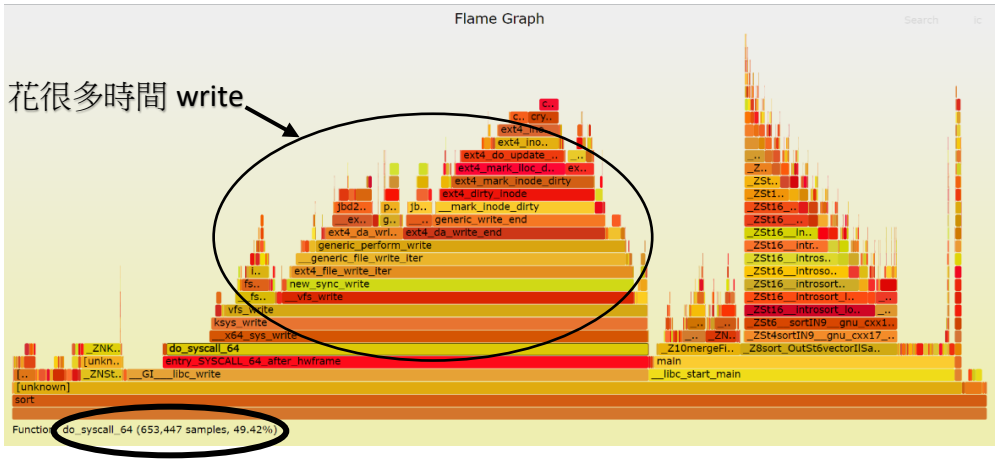
- **你是如何開發這支排序程式，以達成支援大資料排序的行為 :**
  使用 Merge Sort，先將部分的 input 讀到記憶體中排序，產生 output1，接著依序讀入其餘的 input 來排序並產生 output file，全部的 input 切割到多個排序完的檔案中後再將各個 output file 合併，這樣的方法可以對比記憶體容量還大的資料完成操作。

- **你的程式該如何使用，請詳細說明執行的步驟 :**
  Compile : g++ -o sort sort.cpp
  Run : ./sort

效能分析報告：

- **你開發的排序程式對硬體效能優化程度的說明與驗證 ：**

  未優化前的火焰圖如下，大多數的時間都花在寫檔上，do_syscall_64 佔了大部分的比例，讀檔相較起來花較少時間，因此推測主要的優化方向是更改寫檔的方式。



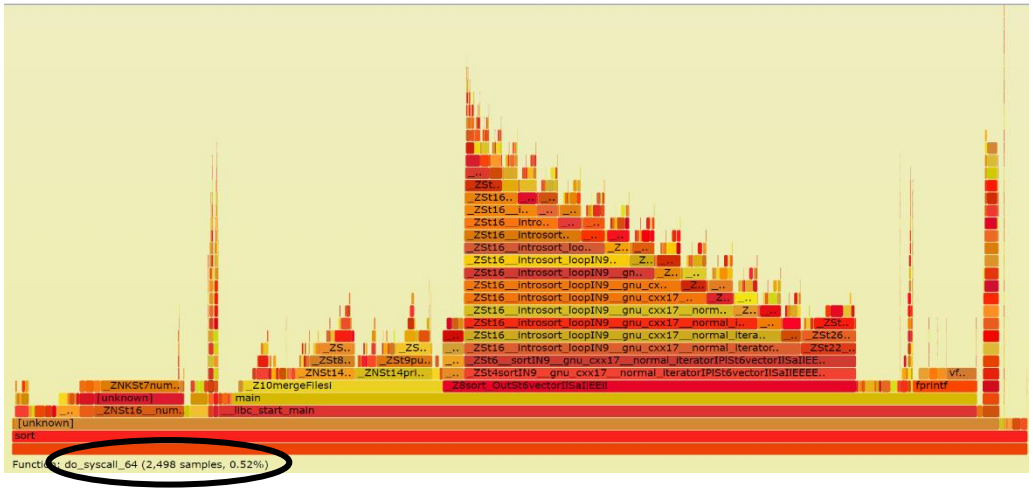
sort 完後寫入 output file 的方法優化前為 ：

```
ofstream outFile("output" + to_string(BlockNum) + ".txt");
for (int i=0; i<ArrSize; ++i)
        outFile << inputNum[i] << endl;
outFile.close();
```

將使用”<<”運算子的方法改為先將 vector copy 到 ostream_iterator 內，再寫入 output file :

```
ofstream outFile{"output" + to_string(BlockNum) + ".txt", ios::out};
copy(begin(inputNum), begin(inputNum)+ArrSize,
                                ostream_iterator<long>(outFile, "\n"));
outFile.close();
```

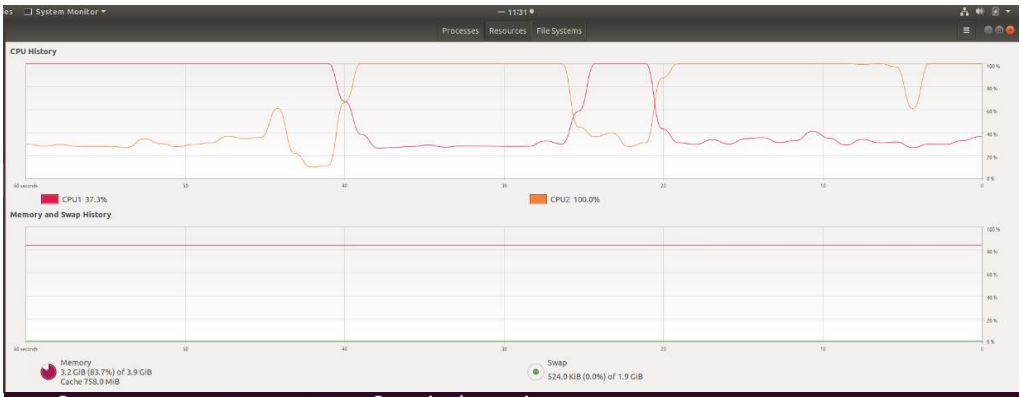
更改後的火焰圖如下，do_syscall_64 所佔用的比例大幅減少，程式不再花很多時間 write。從程式執行時間也可以觀察到，優化後由 system 所花的時間由原本的 28 分鐘大幅降低為 30 秒，而整體的執行時間約差了 3 倍。



由這些觀察可以得知寫檔是拖慢程式執行速度的關鍵，也能驗證往寫檔優化的正確性。

- **系統資源的觀察：**

CPU：如下圖，在執行程式時工作會由兩個核心輪流分擔，透過 perf stat 得到的 cpu-migrations 數據可以知道在程式執行的這段時間，作業系統為了讓 cpu 的每一個 core 工作量平衡，讓程式在這兩個 core 內移動了 280 次。




```
Performance counter stats for './sort':

   1052841.118730      task-clock (msec)         #    0.962 CPUs utilized
          10,5082      context-switches          #    0.100 K/sec
              280      cpu-migrations            #    0.000 K/sec
          78,6563      page-faults               #    0.747 K/sec
  <not supported>      cycles
  <not supported>      instructions
  <not supported>      branches
  <not supported>      branch-misses

   1093.927303867 seconds time elapsed

   1020.389340000 seconds user
     31.986396000 seconds sys
```

Memory：在設計排序程式時，若請求的記憶體空間為 4GB，執行後會跑出 out-of-memory 的錯誤或被系統 kill，因為有一些優先程度更高或更重要的程式需要資源，若將全部的記憶體都分配給這支排序程式會讓系統無法運作，因此作業系統會有一定的保護程序讓系統免於崩潰。

- **在同時運行多支你所開發的排序程式下，你對系統效能的觀察，並結論 OS 的設計要提供哪些優化服務：**

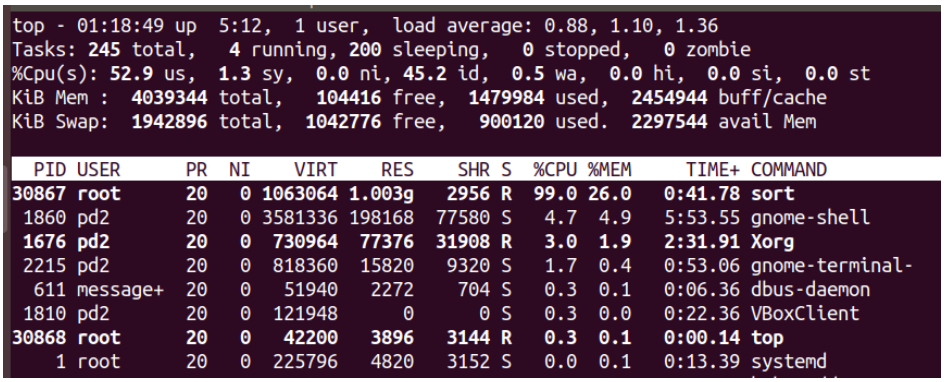
(程式設定 memory 為 1GB，測資為 2GB 的檔案)

只執行一支程式的時間與執行時用 top 指令查看的資訊：

real 7m14.611s

user 6m49.611s

sys 0m13.254s

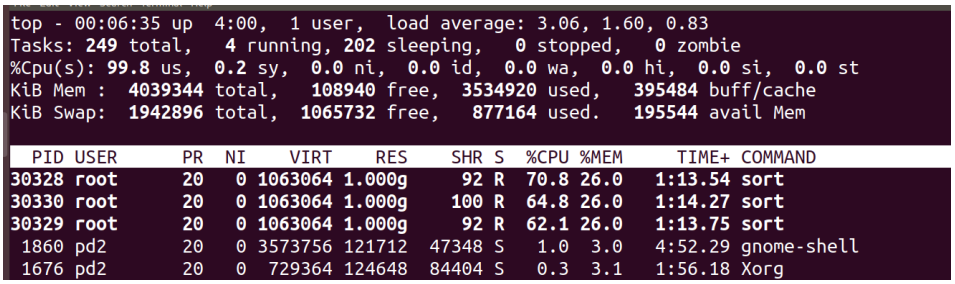


三支程式同時的執行時間與執行時用 top 指令查看的資訊：

real 11m8.837s
user 7m1.519s
sys 0m7.507s

real 11m8.838s
user 7m0.473s
sys 0m7.206s

real 11m9.195s
user 7m1.531s
sys 0m7.211s



分析：三支相同程式同時運行時，執行時間都差不多落在 11.9 分，跑單支程式的時間約為 7 分鐘，而由 top 指令可以看出在 CPU 使用率的部分，只跑一支程式時使用率約在 50%，同時跑三支程式的使用率將近 100%。推測是跑單支程式時對系統的負擔並不大，只使用部分的 CPU 就能完成工作，所以沒有將 CPU 資源用盡，留下一些資源來處理接下來可能出現的其他 task，而跑三支程式時對系統負擔較大，故使用到全部的 CPU，另外由 top 指令得到的 CPU 占用率分別為 70.8%、64.8%、62.1%，因為作業系統讓三支優先順序相同的程式輪流使用 CPU，所以執行時間才會很接近。

結論：在這次的作業中，利用排序比記憶體大的資料來觀察作業系統在這些程式中扮演的角色。

對 CPU 來說，作業系統會讓每支程式分配到一定的 CPU 資源，讓優先程度相同的程式分配到相似的使用率，盡可能的同時完成，可以優化的地方是在執行單支程式時能先分配較多的資源，讓 CPU 使用率提高，當有其他優先度較高的工作出現時再降低 CPU 使用率。

對 Memory 來說，當程式一次要求超出整體記憶體容量或剩餘記憶體容量的資源時，作業系統會出現 out of memory 的錯誤或直接 kill 程式來讓系統能夠維持運作而不會崩潰，可以優化的地方是當剩餘記憶體不夠程式執行時，不用馬上讓該程式終止，或許可以讓不是那麼重要的其他程式釋放記憶體，讓想執行程式可以繼續執行。

對 Disk 來說，反覆的讀寫是拖慢執行時間的一大關鍵，但可以在程式內解決，如將 cin/cout 換成 scanf/printf。