Student ID: F74077120

Name: 蔡孝龍 Marc Fraser Go Magante

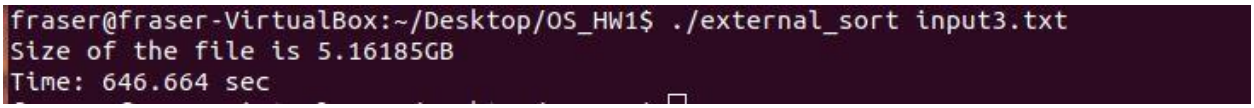
Department and Grade: Computer Science and Information Engineering (大三)

Development environment:

- OS: Ubuntu 18.04.2 LTS (VM VirtualBox)
- CPU: Intel Core i7-8750H CPU @ 2.20 x 6
- Memory: 16GB but in VirtualBox is 2.9 GB so approx. 3GB
- Programming Language (version): C++ 11

Execution time:

- 646.664 seconds approx. (5.2 GB)



Development and user guide:

1. Development

- In order to develop a program that would support large data sorting, I've used the external merge sort which the TA discussed during the homework discussion. Basically, I split the large text file into multiple smaller text files and in those smaller text files, the text files are already sorted in ascending order. The next thing I did was comparing each sorted text files by inserting each smallest values from each file into the minheap. Afterwards, I draw the minimum value from the heap then insert it to the output file then add a new element/value from the same file where it was drawn into the heap. All of this was possible through the help of the queue library. I used the data structure minheap so that I could combine all the files at once and write it into the output file. In order to create the minheap, I created two classes one serves as a minheap node and the other is to compare so that I can use the priority_queue of C++.

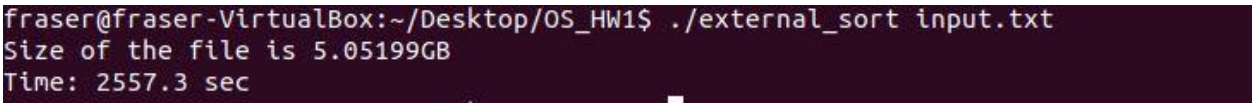
2. User guide:

- # Compile
- $ g++ -o external_sort external_sort.cpp
- # Run
- $ ./external_sort [data path]

Performance analysis report:

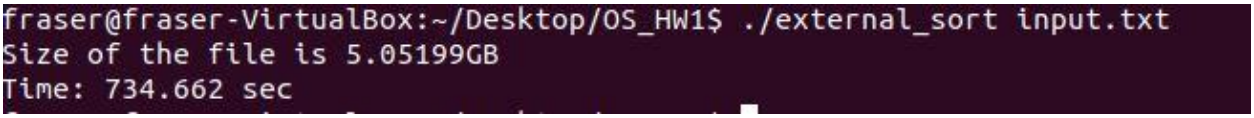
1. How great is your optimization and how did you verify it?

- My initial program was very slow with the time 2557.3 seconds for an approx. 5.1GB file.

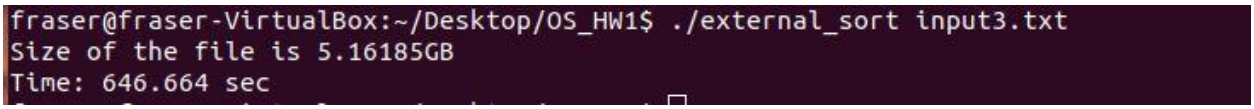


- Since the initial program will generate many temporary sorted files, I think it will take a lot of time to create, close and delete files all the time so by changing the maxreadnum, my **temporary sorted files became fewer but bigger**; also, I **changed some of the for loops structure into descending order** that only has two conditions which made my program run faster compared to having several sorted files and a for loop that has three conditions. In the actual test, the speed will decrease with more files.

- **I changed endl to \n**. The execution time of my code vastly increased. It seems that endl will force the output buffer to be cleared every time, which wastes a lot of time.

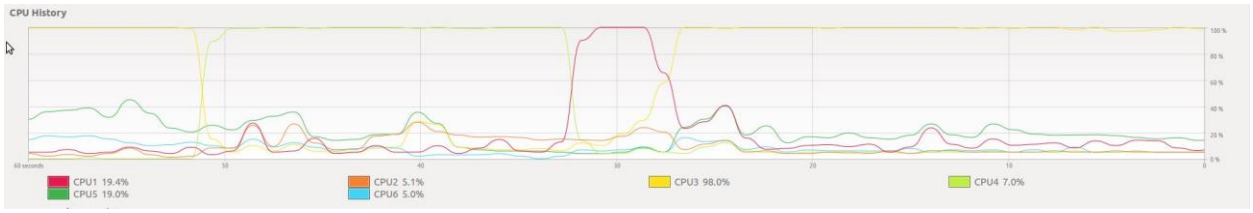


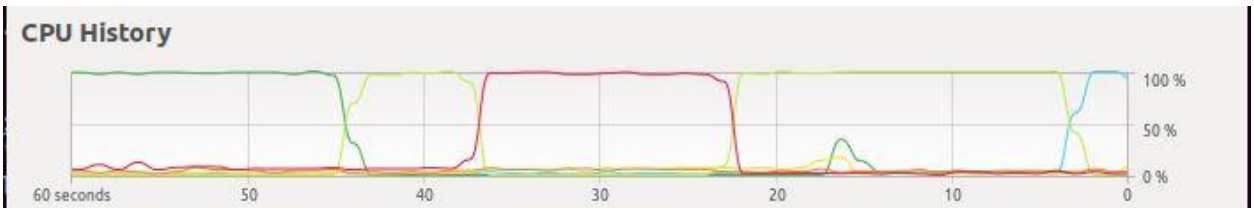
- Instead of using C++'s File open and close also printing the values to a file, I **used C's file open, close, and fprintf** for only the temp and output files. Since C is faster than C++, using these will help my program run faster.

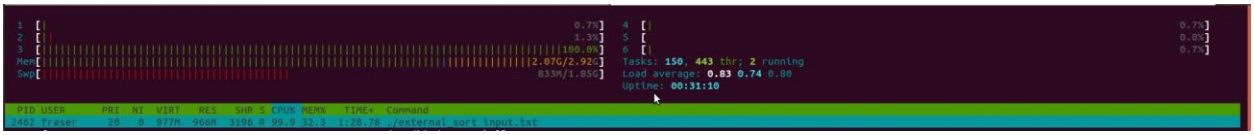


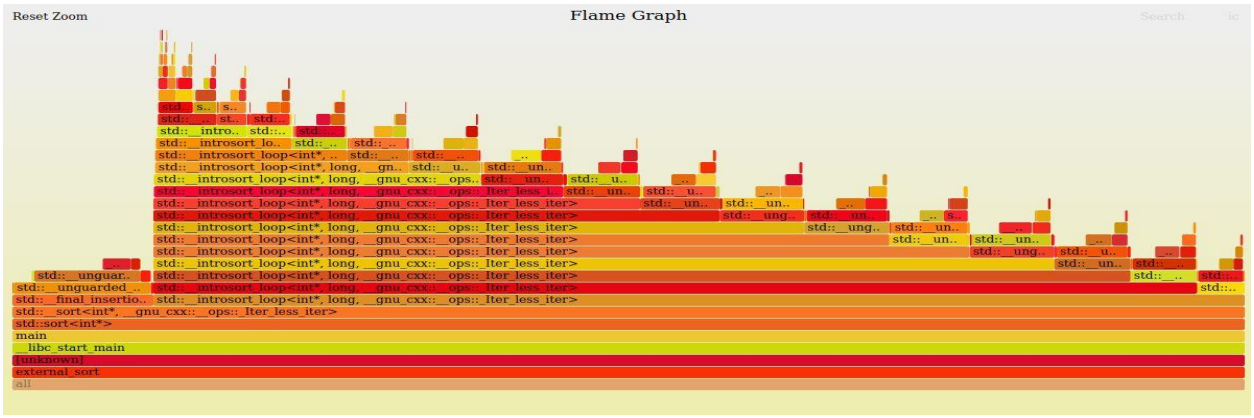
***All of which were verified through the program's execution time.**

2. What did you observe when OS managing multiple your sorting programs?

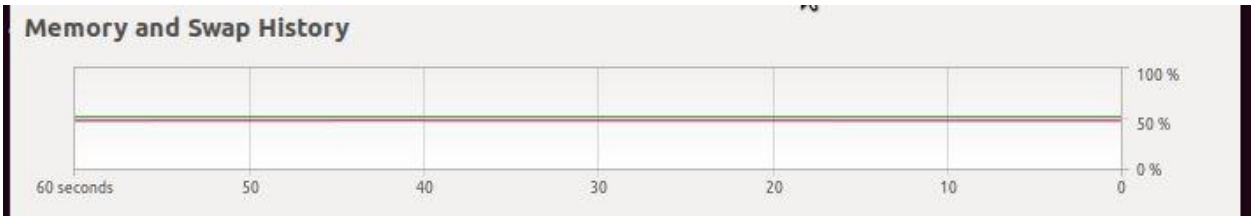








In the CPU usage, it can be found that the first half of the CPU has periodic fluctuations. The judgment is to execute the split function, split the input.txt into multiple temp files, and do the sort. After the experiment, the sort eats the most CPU resources. In the latter half, it can be found that the CPU occupancy ratio is not high. And after htop found out, it is the programs inside the system that are mostly used. We can also see in the flame graph that the process that uses the most CPU in 1 minute which has the larger the width and more CPU time is used was the external sort.



In memory usage, there seems to be no obvious change in the graph.

Under the sequencing program, I observed the performance of the system and conclude which optimization services the OS design should provide:

First is **branch predict** because when writing the file because when writing the file, I have an if statement in a for loop to determine whether to wrap up at the end. This part of the OS does branch prediction. Second is **CPU migration**. This can also be clearly seen from the system monitor (each color has high and low peaks), the context switch can make the computer operate more smoothly, and reducing CPU migration is also an important function of the OS (reducing I/O). Third is **page**. It can be found that the memory usage is always very small during the execution process. In fact, this is due to the page. The OS uses virtual memory to put many unnecessary memory back into the disk, resulting in memory usage not being written later. The data rose sharply. You can see that there are actually quite a few page fault numbers. The fourth is **system call**. It can be found that the execution time of the program is divided into usr time & sys time, because when doing file operations, you must call system call to do open, write, read and disk read and write.