REPORT

Operating System Principles

- *Ashmit Sachan (S3873827)*

This report is about load balancing. It explains all the measure that were taken during this assignment to show how load balancing can be used to ensure that all processes or threads finish execution at approximately the same time for any given task.

You need to have intermediary level of understanding of C++ to understand the code that was submitted with the report.

- ASHMIT SACHAN [S3873827]
- 14/ 09/ 2022
- https://github.com/ashmit-sachan/OSP_Assignment1_S3873827

NOTE: Due to some personal problems I was not able to submit this assignment on time. I tried to do as good as I could as per my understanding of the assignment. I managed to complete this assignment in 72 hours, hence there is not history of commits. I ensure you that all the work is original and I have not copied or taken reference of code from any resource except the documentation.

Thank you.

SUMMARY:

★ PURPOSE:

Use S-LBAN (Static Load Balancing Technique) to find ways to measure, adjust and optimally deliver the result of a computation task.

This report shows how S-LBAN can be used to divide a huge problem into smaller chunks and optimise them in such a way that each chunk of the problem takes approximately the same time, that is, using resources in a balanced way to ensure a compilation of the correct solution to the problem.

★ METHODS:

All methods were used as described in the Assignment Specification. From task 1 till 5.

★ RESULTS:

Described in eash section. Refer to the index table on the next page for details.

★ CONCLUSIONS:

The methods used to resolve the problem resulted in a huge improvement in the time taken by a common traditional approach to the given problem. The utilisation of resources was made in a much better way as compared to a bruteforce approach to the solution of the problem.

TASK 1:

- The combination of coreutil tools that were used to achieve this taks were “***tr***”, “***sort***”,“***shuf***” and “***awk***”:

  - tr is used to filter the words, any words that have any characters that are not alphanumeric are removed.

  - awk is used to check the length of the words. If word length is less than 3 or greater than 15, the word is not considered and is removed.

  - sort is used to sort the filtered file that is generated after all the operations to clean the dirty file are done.

  - shuf is used to shuffle the sorted file that is generated after all operations that were done to clean the file.

- The source of “Task1filter” is in “Utils.cpp” under the method named “TaskFilter”

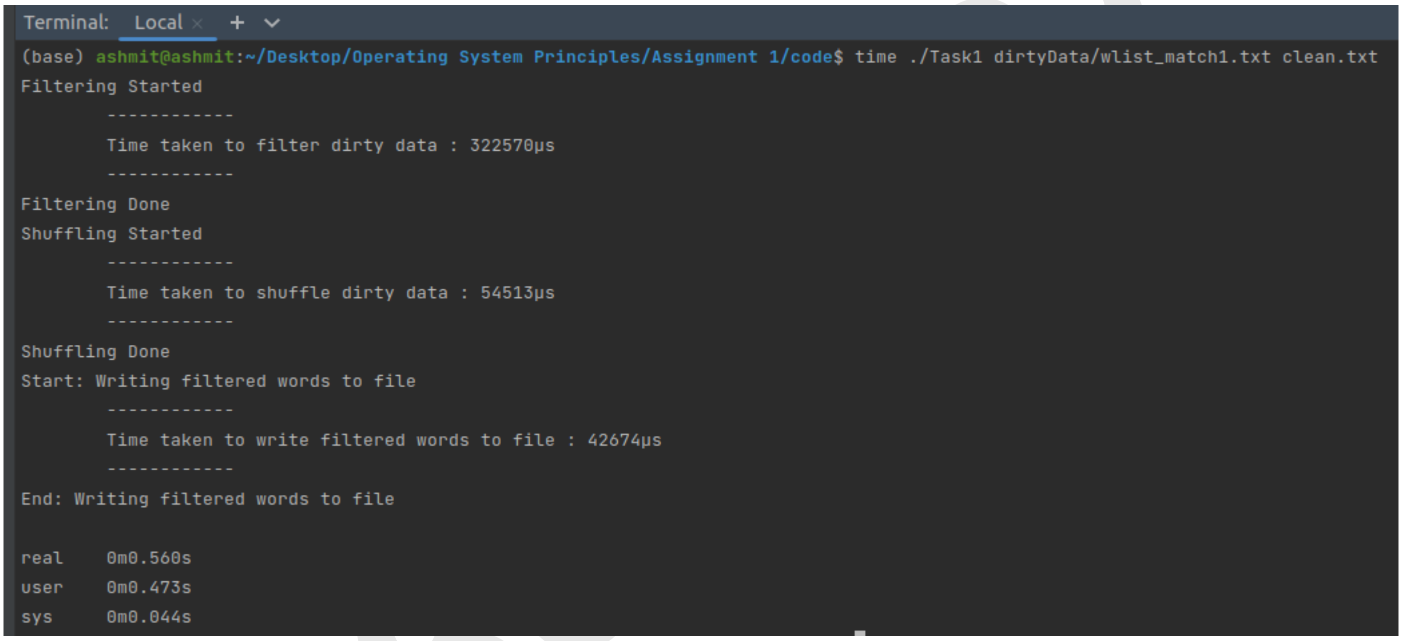
- The number of words of length 3 to 15 inclusive are found to be 1435083 in number as of clening the “wlist_match1.txt” file that is given under dirtyData folder

  i.e: clean file generated from “./dirtyData/wlist_match1.txt” has ***1435083*** lines or words, one word in each line.

- There’s a struct called “***TimeIt***” that is responsible for printing the time taken by various operations to standard output. This has been used throughout the code to find the time taken. Underlying this struct is the ***“standard chronos” library***. This measured time difference between two time points in ***microseconds***.

- The filtration process includes removing all characters that are not alphabets and then checking for the length constraint. The word is then added to a ***set*** in order to ***remove duplicates*** automatically. The set is then converted to a ***vector*** and then shuffled and retuned.

PERFORMANCE DATA: (Task 1)



```
(base) ashmit@ashmit:~/Desktop/Operating System Principles/Assignment 1/code$ time ./Task1 dirtyData/wlist_match1.txt clean.txt
Filtering Started
        ------------
        Time taken to filter dirty data : 322570μs
        ------------
Filtering Done
Shuffling Started
        ------------
        Time taken to shuffle dirty data : 54513μs
        ------------
Shuffling Done
Start: Writing filtered words to file
        ------------
        Time taken to write filtered words to file : 42674μs
        ------------
End: Writing filtered words to file

real    0m0.560s
user    0m0.473s
sys     0m0.044s
```

1. ***Filtering*** : 0.32 seconds
2. ***Shuffling*** : 0.054 seconds
3. ***Writing to File*** : 0.042 seconds

4. Overall this task completes in 0.56 seconds

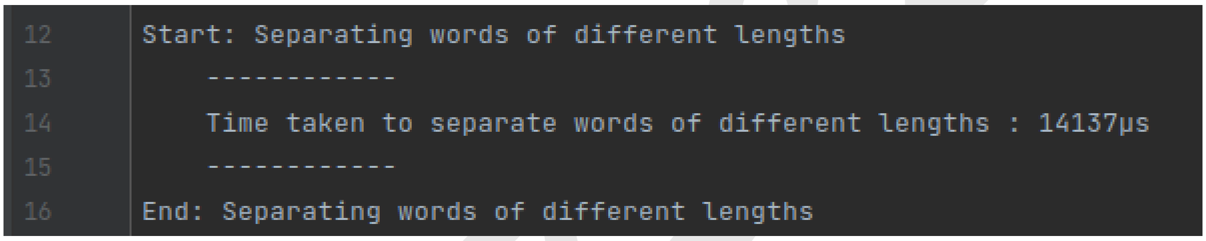
TASK 2:

- This task takes name of dirty file cleans the dirty file, and then creates multiple processes to deal with each word length.

  The ***map2*** method creates ***13 child processes***, one to handle each length of words, then sorts them according to the 3rd letter and writes to a file.

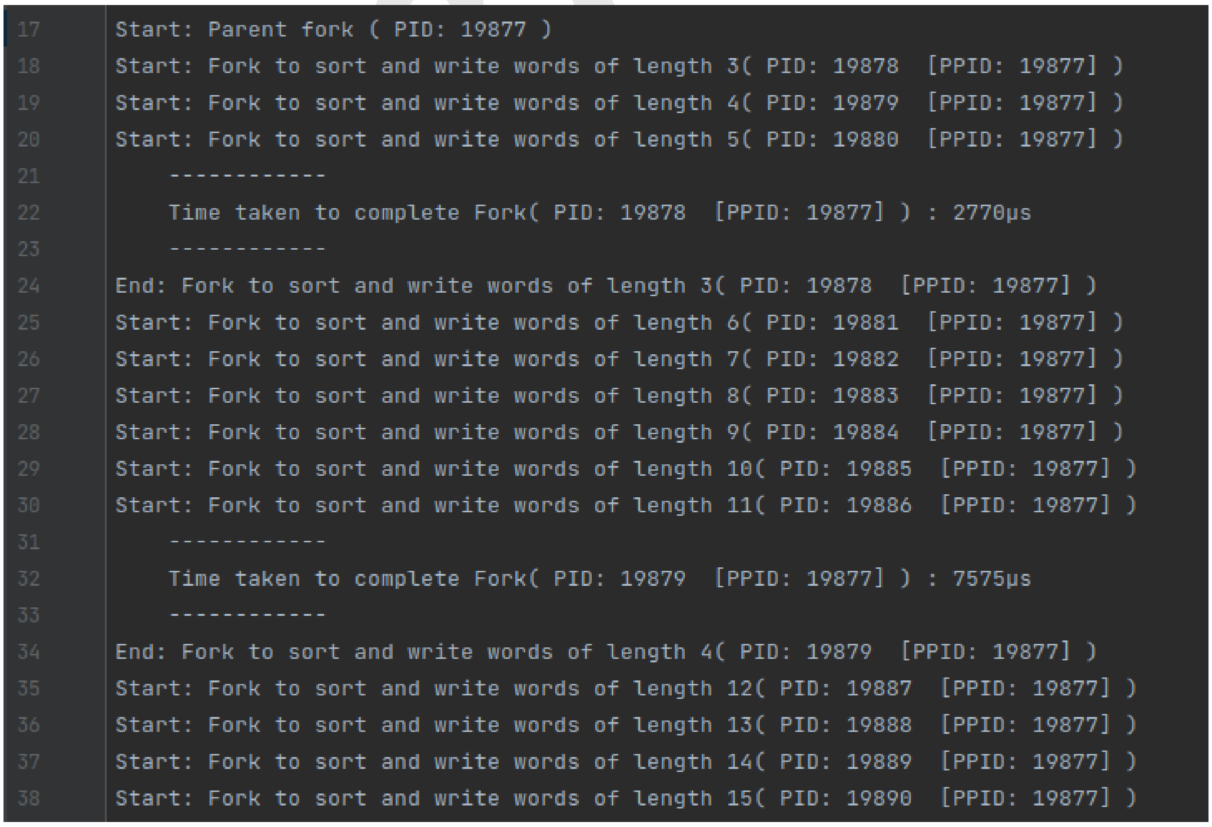
PERFORMANCE DATA: (Task 2)

1. ***Separate words of different lengths:*** 0.014 seconds



```
Start: Separating words of different lengths
    ------------
    Time taken to separate words of different lengths : 14137µs
    ------------
End: Separating words of different lengths
```

2. ***Fork, Child Processes and measure of time:***

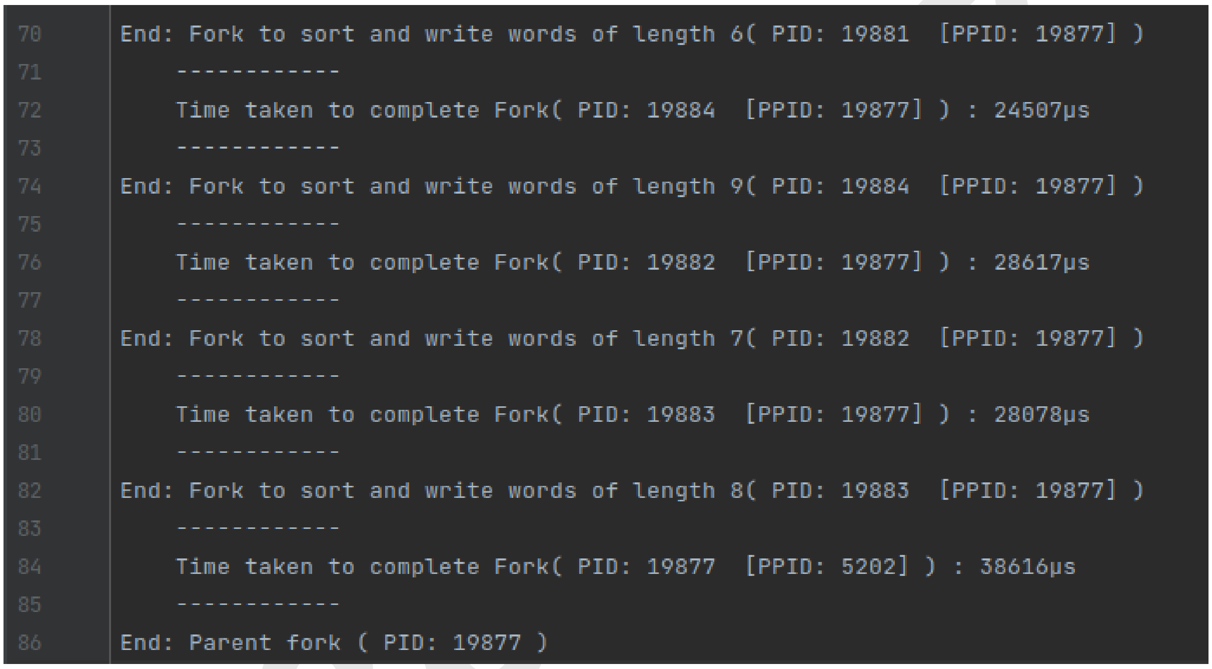


```
Start: Parent fork ( PID: 19877 )
Start: Fork to sort and write words of length 3( PID: 19878  [PPID: 19877] )
Start: Fork to sort and write words of length 4( PID: 19879  [PPID: 19877] )
Start: Fork to sort and write words of length 5( PID: 19880  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19878  [PPID: 19877] ) : 2770µs
    ------------
End: Fork to sort and write words of length 3( PID: 19878  [PPID: 19877] )
Start: Fork to sort and write words of length 6( PID: 19881  [PPID: 19877] )
Start: Fork to sort and write words of length 7( PID: 19882  [PPID: 19877] )
Start: Fork to sort and write words of length 8( PID: 19883  [PPID: 19877] )
Start: Fork to sort and write words of length 9( PID: 19884  [PPID: 19877] )
Start: Fork to sort and write words of length 10( PID: 19885  [PPID: 19877] )
Start: Fork to sort and write words of length 11( PID: 19886  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19879  [PPID: 19877] ) : 7575µs
    ------------
End: Fork to sort and write words of length 4( PID: 19879  [PPID: 19877] )
Start: Fork to sort and write words of length 12( PID: 19887  [PPID: 19877] )
Start: Fork to sort and write words of length 13( PID: 19888  [PPID: 19877] )
Start: Fork to sort and write words of length 14( PID: 19889  [PPID: 19877] )
Start: Fork to sort and write words of length 15( PID: 19890  [PPID: 19877] )
```

It can be seen when the parent fork started. A ***loop*** was used to ***make multiple child processes***.

Also two of these were completed and killed even before all of the child processes were made. This happened because there is a ***very few 3 and 4 letter words***. We know that by matching the ***process Ids (PIDs)*** of the ***Child Processes*** in Start and End.

After creation of all the Child Processes



```
End: Fork to sort and write words of length 6( PID: 19881  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19884  [PPID: 19877] ) : 24507µs
    ------------
End: Fork to sort and write words of length 9( PID: 19884  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19882  [PPID: 19877] ) : 28617µs
    ------------
End: Fork to sort and write words of length 7( PID: 19882  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19883  [PPID: 19877] ) : 28078µs
    ------------
End: Fork to sort and write words of length 8( PID: 19883  [PPID: 19877] )
    ------------
    Time taken to complete Fork( PID: 19877  [PPID: 5202] ) : 38616µs
    ------------
End: Parent fork ( PID: 19877 )
```

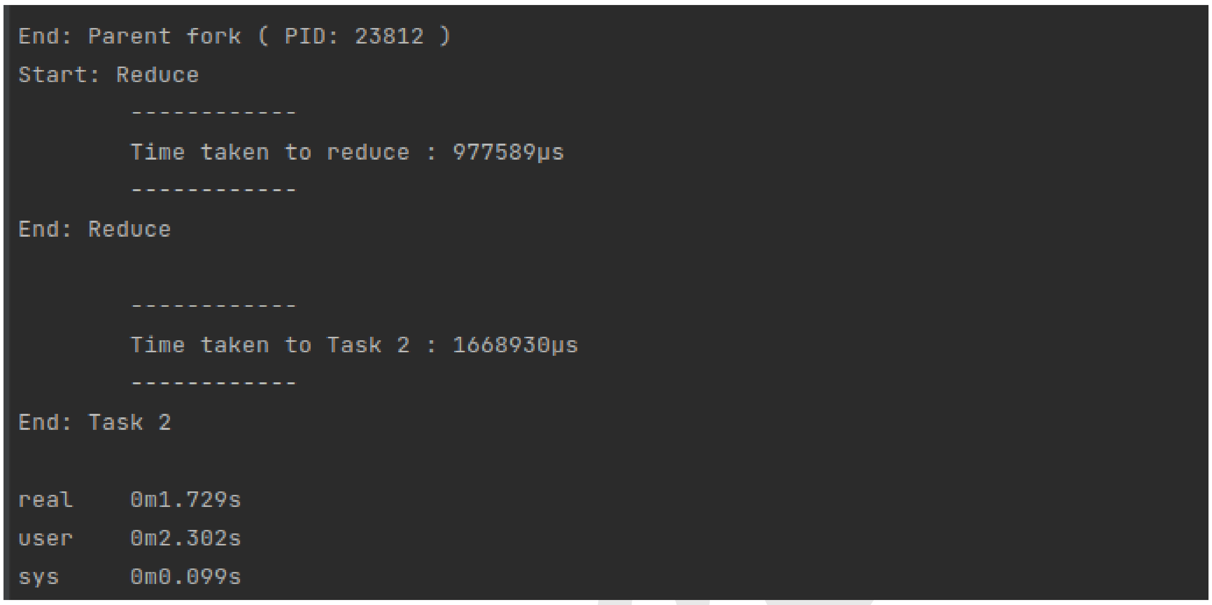
All Child processes finish according to their ***respective loads***. The parent process waits till all processes are killed and ***omits the total time taken to complete the sorting and writing to file***.

Note that the Child Processes were started in sequence but they finish when their processing finishes. In the image above we can see that the sequence of finishing was : **6 < 9 < 7 < 8.**

I have also printed the PPIDs of the processes just to prove that all processes except one are from the same parent.

3. ***Reduce and time taken:***

   Finally after all of this is done, the reduce2 method opens all sorted files and perforems a 13 -> 1 merge sort.



```
End: Parent fork ( PID: 23812 )
Start: Reduce
        ------------
        Time taken to reduce : 977589µs
        ------------
End: Reduce

        ------------
        Time taken to Task 2 : 1668930µs
        ------------
End: Task 2

real    0m1.729s
user    0m2.302s
sys     0m0.099s
```

   Reduce method took ***0.98 seconds*** to process.

4. In total Task 2 was finished in ***about 1.73 seconds.***

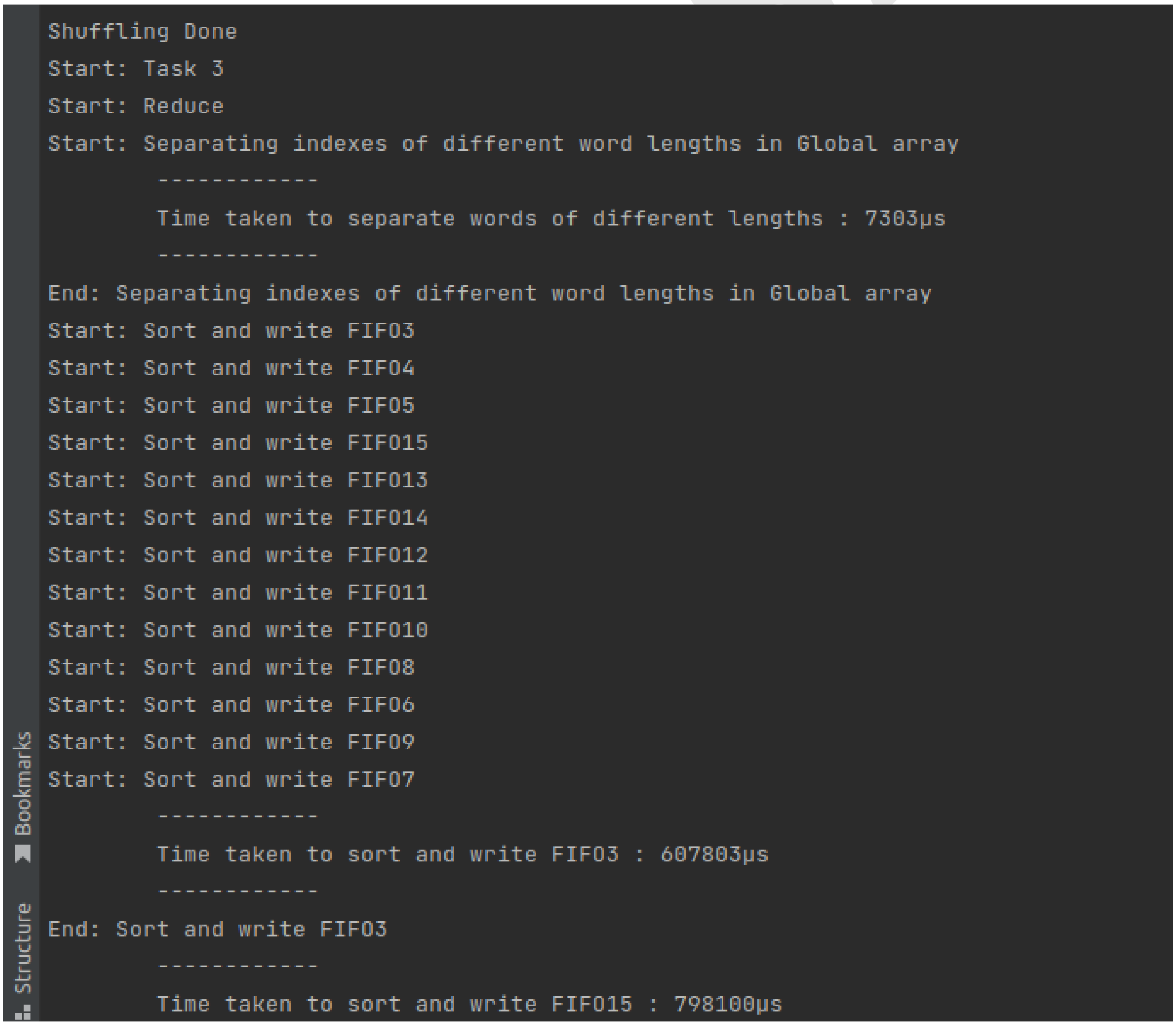
TASK 3:

- This task takes name of dirty file cleans the dirty file, and then creates multiple threads to deal with each word length. We will also see the use of FIFO files in this task.

  The ***map3*** method creates ***13 threads in the current process***, one to handle each length of words, then sorts them according to the 3rd letter onwards and writes to a FIFO file.

PERFORMANCE DATA: (Task 3)

1. ***Map method and timing:***



```
Shuffling Done
Start: Task 3
Start: Reduce
Start: Separating indexes of different word lengths in Global array
        ------------
        Time taken to separate words of different lengths : 7303μs
        ------------
End: Separating indexes of different word lengths in Global array
Start: Sort and write FIFO3
Start: Sort and write FIFO4
Start: Sort and write FIFO5
Start: Sort and write FIFO15
Start: Sort and write FIFO13
Start: Sort and write FIFO14
Start: Sort and write FIFO12
Start: Sort and write FIFO11
Start: Sort and write FIFO10
Start: Sort and write FIFO8
Start: Sort and write FIFO6
Start: Sort and write FIFO9
Start: Sort and write FIFO7
        ------------
        Time taken to sort and write FIFO3 : 607803μs
        ------------
End: Sort and write FIFO3
        ------------
        Time taken to sort and write FIFO15 : 798100μs
```

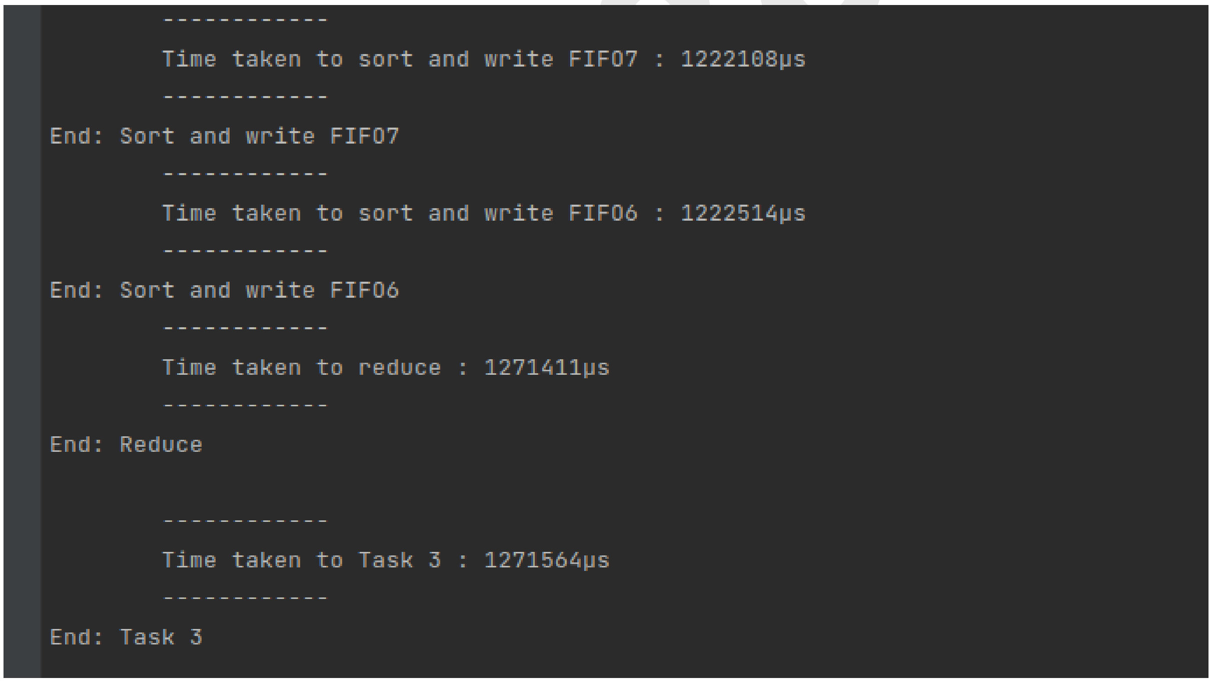
When the main method is called, both of the threads are run, hence the message **“START: Reduce”** and the thread for map also runs and is locked in and executed first.

The indexes are then put into lists as per their respective word lengths. Time taken to do this operation is 0.007 seconds.

Then we move on to creating the FIFO files and writing the sorted words into respective FIFOs.

2. *Reduce method and timing:*

As soon as the map method is executed, the lock is lighted and the execution moves on to the reduce method.



```
        ------------
        Time taken to sort and write FIFO7 : 1222108µs
        ------------
End: Sort and write FIFO7
        ------------
        Time taken to sort and write FIFO6 : 1222514µs
        ------------
End: Sort and write FIFO6
        ------------
        Time taken to reduce : 1271411µs
        ------------
End: Reduce


        ------------
        Time taken to Task 3 : 1271564µs
        ------------
End: Task 3
```

The reduce method then reads the FIFO files and performs a 13->1 merge sort.

Note that the threads were started in sequence number but they finish executing according to their respective loads.

3. ***Total time:***
   The total time taken by Task 3 was 1.27 seconds.

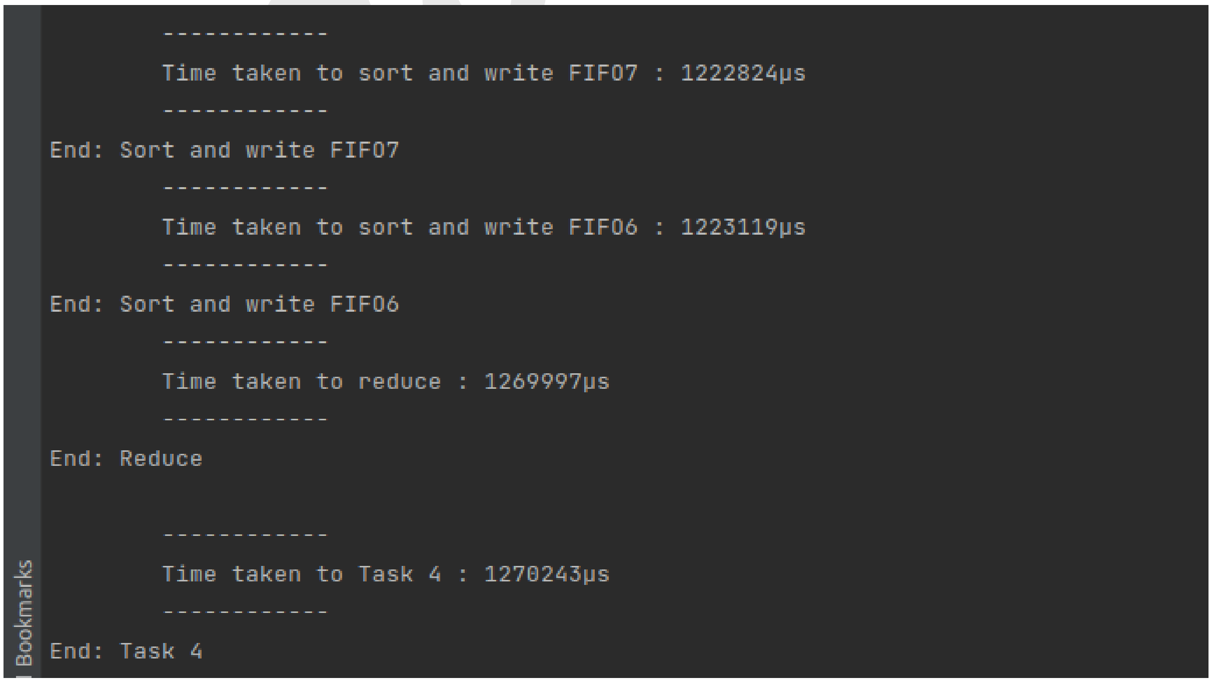
   We can see a 0.5 second decrease in time due to the use of FIFO files. Which facilitate a much faster transfer of data between two threads or processes.

TASK 4:

- This task takes name of dirty file cleans the dirty file, and then creates multiple threads to deal with each word length. We will also see the use of FIFO files in this task.

  The ***map4*** method creates ***13 threads in the current process***, then takes into account the number of words that each thread has to deal with, then sets the thread execution priority level. Then it proceeds as Task 3.

PERFORMANCE DATA: (Task 4)



```
        ------------
        Time taken to sort and write FIFO7 : 1222824µs
        ------------
End: Sort and write FIFO7
        ------------
        Time taken to sort and write FIFO6 : 1223119µs
        ------------
End: Sort and write FIFO6
        ------------
        Time taken to reduce : 1269997µs
        ------------
End: Reduce

        ------------
        Time taken to Task 4 : 1270243µs
        ------------
End: Task 4
```

Similar timing as Task 3. Sometimes better. We predicted it to be better but then found out that the reason for similar timing was the presence of a 16 core Ryzen 9 5950X CPU. The amount or resources available are in an adequate amount thus no need of thread scheduling.

Although it is noticeable that the tasks get executed in the sequence of their set priority.