# Operating System Principles
# <u>Assignment 1 Report</u>

Submitted by: Maansi Arora (s3885529)

The following is my github Repository:
https://github.com/maansiarora/OSP_Assignment1

## Task 1:

- **Task1.sh**
  Core utils used to generate the output as stated in the spec sheet are:
    - **awk**
      This one is to remove any duplicate words from the file that we are reading.
    - **sed**
      This one is basically used for editing the file, in our case to remove any duplicates and and numerical values
    - **grep**
      It is being used to keep only those words that have length greater than or equal to 3 and less than or equal to 15.
    - **shuf**
      It is being used to shuffle the file.

  More details are mentioned though comments in the .sh file itself.
  This would hence generate a file named *'shuffled_words.txt'* after all the operations told in the spec sheet.

- **Task1filter()**
  **Overview** - Through this method, we are replicating what we did in our shell file as a c++ function. We begin by separating the command arguments into input and output files. We then add the words from the file into a single list and use different functions for doing the tasks like removing duplicates using unique(). The words in the list are then added to a vector where we filter out more words on the basis of the length and characters in it. We then write it into a final file.

  The source file used for this task was *wlist_match10.txt* with the total length of the file being 59951.
  And after running Task1, the number of words in the clean file are 59225.

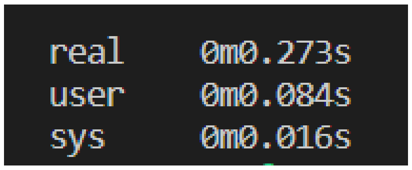
**Running the code:**
To run the task, the user first needs to execute the *'make'* command to create all the executable files.
Further to run the task one can use the following command:
'./Task1 DirtyFile.txt Cleanfile.txt'

The name of the file should be entered with the extension as well. Also, if a user wants to test out any other input files, they'll have to add that into the folder itself.

**Performance:**
To measure the performance I've used the linux time command. To use that, I wrote 'time ./Task1 DirtyFile.txt Cleanfile.txt' on the terminal.

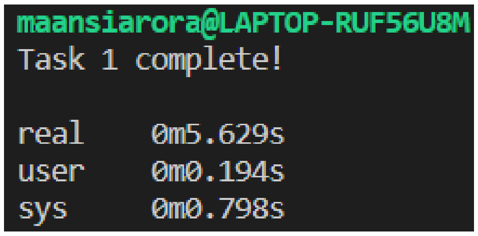
Output format:
- real: It is the time from start to the finish of the call. It is the 'wall clock' time.
- user: amount of CPU time spent in user mode
- sys: amount of CPU time spent in kernel mode

The following is the performance for the shell file:



```
real    0m0.273s
user    0m0.084s
sys     0m0.016s
```

The following is the performance for the c++ program:



```
maansiarora@LAPTOP-RUF56U8M
Task 1 complete!

real    0m5.629s
user    0m0.194s
sys     0m0.798s
```

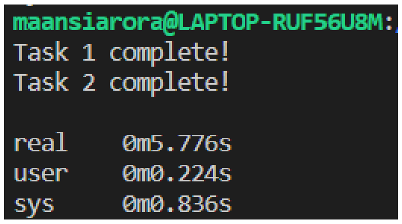
We can see that the bash script was significantly faster than the c++ program. Also, the simulation did not exceed the limit specified in the spec sheet of 10 secs.

**Task 2:**

- **map2():**
  This function is used to separate the clean file into 13 different files with words of particular lengths. We start by creating vectors of different lengths, scanning through the clean file and adding words with different lengths into their vectors. We then use a fork for sorting based on the third letter of the words. Sorting in both the child and parent occurs simultaneously.

- **reduce2():**
  This function is for merging the 13 files created into one on the basis of the lowest order word that is read from each file.

  The most recent performance record is:



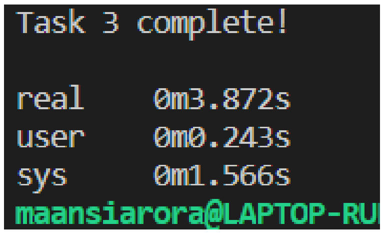
```
maansiarora@LAPTOP-RUF56U8M:
Task 1 complete!
Task 2 complete!

real    0m5.776s
user    0m0.224s
sys     0m0.836s
```

**Task 3:**

- **map3():**
  This function made 13 index arrays and threads from the global array. We then sort each thread on the basis of the third letter of each word and hence output 13 FIFO files.
- **reduce3():**
  This function further combines the 13 FIFO files created by map3() and merges them into one.

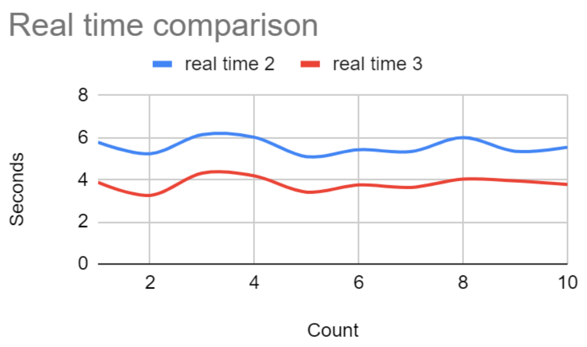
  The most recent performance record is:

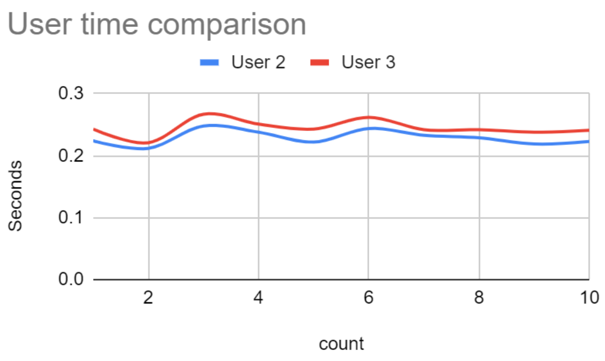


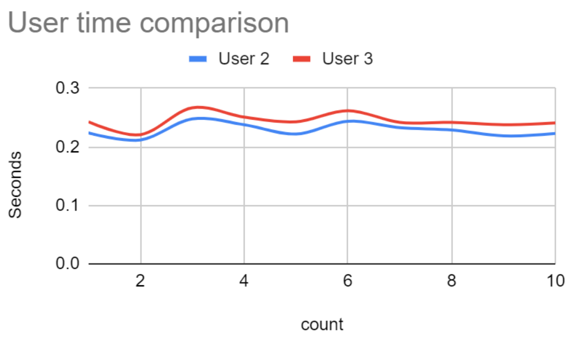
```
Task 3 complete!

real    0m3.872s
user    0m0.243s
sys     0m1.566s
maansiarora@LAPTOP-RU
```

**Comparing Task 2 and Task 3 performance:**
Now, the keep track of the performance in terms of time, I recorded the final outcomes several times which is presented as below:

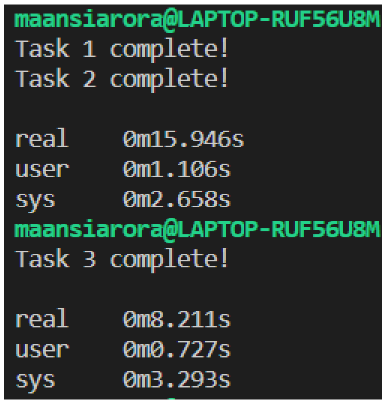






We can observe that task 3 performs better in terms of real time but otherwise for user and sys, the time taken by task 2 is lower even though there is not much difference between them.

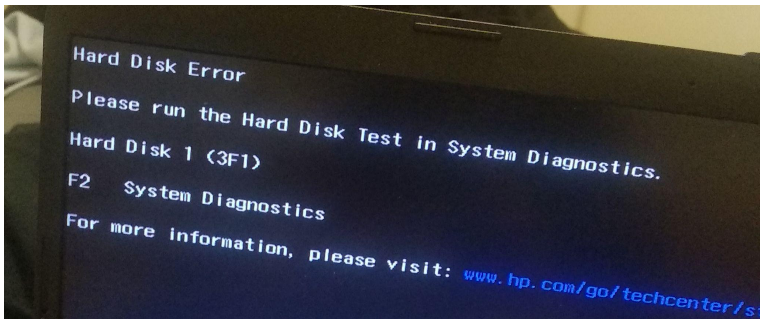
Side note: I haven't included the outlier results like the one shown below because my laptop's hard disk just crashed while I was finalizing my assignment and writing down the results. And because of this the performance of my laptop instantly dropped.



```
maansiarora@LAPTOP-RUF56U8M
Task 1 complete!
Task 2 complete!

real    0m15.946s
user    0m1.106s
sys     0m2.658s
maansiarora@LAPTOP-RUF56U8M
Task 3 complete!

real    0m8.211s
user    0m0.727s
sys     0m3.293s
```



```
Hard Disk Error
Please run the Hard Disk Test in System Diagnostics.
Hard Disk 1 (3F1)
F2   System Diagnostics
For more information, please visit: www.hp.com/go/techcenter/s
```

## Analyzing the performance results:

Seeing the graphs, we can see that task 3 is faster than task 2. Looking at both the approaches, task 2 used processes and task 3 used threads.
The results make sense because **inter-thread communication** is simpler than that between different processes. Also **context switching** is much faster in threads than between processes because it requires very little memory copying which makes it easier for the OS to stop one thread and start another one.
Considering both these factors of inter-process communication and context switching between processes, we can validate that more time taken by task 2 is fair.

## Limitations:

- I'm using c++ instead of c.
- I wasn't able to create and open fifo files itself. So writing within them was not being executed so I went forward with using normal files.
- I wasn't able to figure out how to use qsort() in c++. Not sure if it's possible or not, so I went ahead with using sort() itself.
- Was trying to test using gprof for profiling but wasn't working out at the last moment and the gmon.out was showing all 0 performance values.