HOW TO RUN:

NOTE: the results shown are based on my virtual machine running ubuntu-20.04.5-live-server-amd64 OS with a base mem of 4GB and 1 CPU assigned processor.

Compile the programs with the provided make file with “make”

TASK 1:

After compilation, use ./Task1Filter <dirty File> <clean File>

Where dirty file is the file path to be cleaned and clean file is the path to save the cleaned data.

The resultant file is the clean file that will be used by the other programs.

TASK 2:

Run this with ./Task2 <clean File>

Map2() will generate files of lengths 3 – 15 like “T2words<word Length>.txt”

And the sorted files are saved as “T2words<word Length>sorted.txt”

The merge sorted file is saved as “T2mergedFile.txt”

TASK 3:

Run this with ./Task3 <clean File>

Map2() will generate files of lengths and saved as “T3words<word Length>.txt”

The sorted files are saved as “T3words<word Length>sorted.txt”

The merge sorted file is saved as “T3mergedFile.txt”

TASK 4:

This runs I the same way as task 3

Run this with ./Task4 <clean File>

Task 1:

- my program filters the data and any words containing special characters and or numbers like (“ , ‘ 1) are ignored and only word containing English alphabets (case ignored) are accepted.

- used the source file "wlist_match1.txt".

- using grep to search for patterns and only list words of lenght 3 to 15 using options -o for only matched words

  and -w for whole words with the passed pattern '\w\{3,15\}'.

- used sort to get rid of duplicate words with the -u option for to remove the duplicates but sorted.

- used shuf to shuffle the sorted file and redirected to the output text file.

- the command Used is below

```
grep -o -w '\w\{3,15\}' $1 | sort -u | shuf > $2
```

    $1 = input dirty file

    $2 = output clean file

Word count

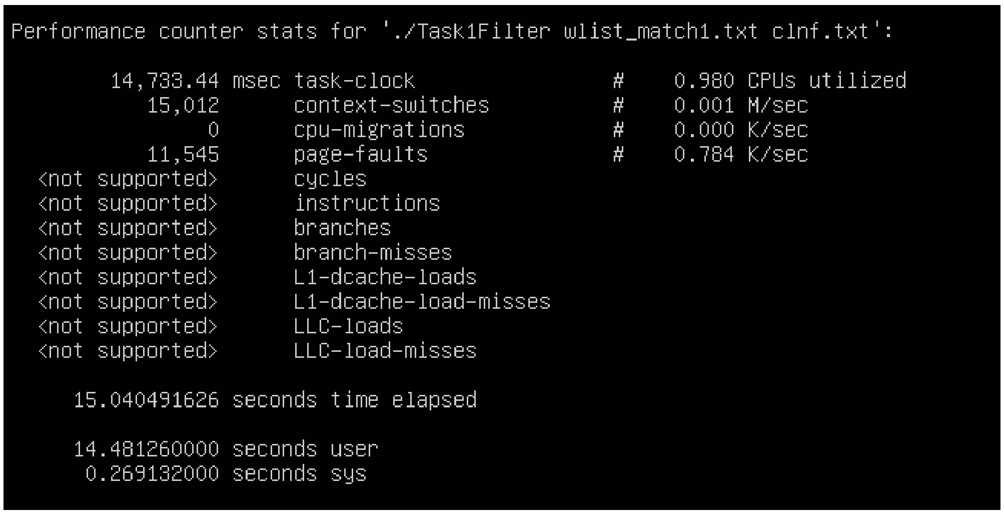
  - originl (dirty file) word count : 1516999 words

  - new (clean file) word count : 1371534 words

Performance

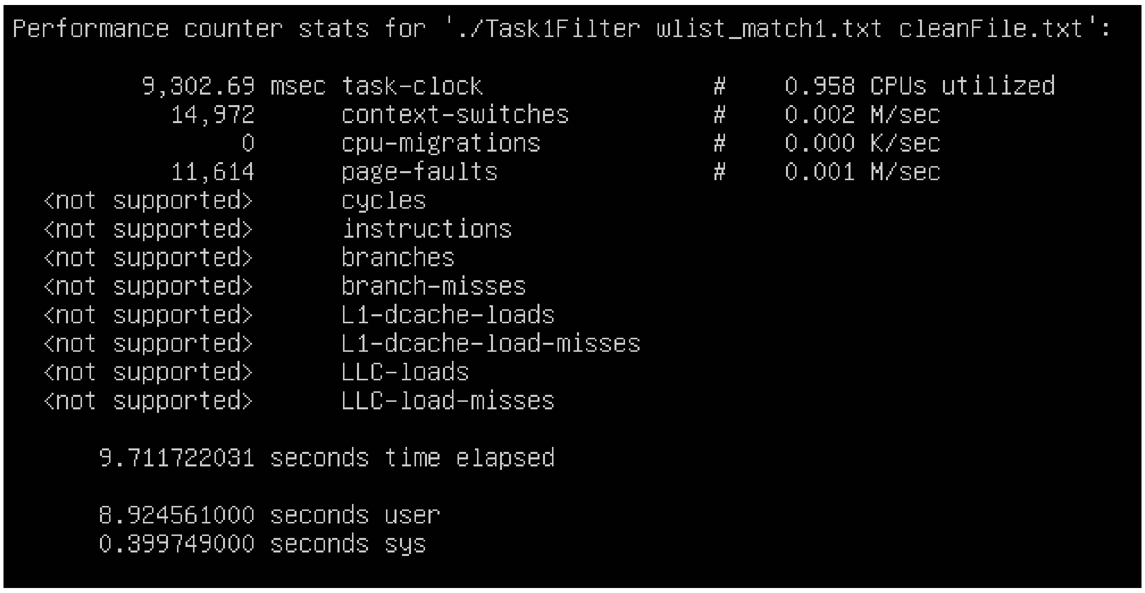
- below are performance stats for running Task1Filter for the first time. Stats show CPUs utilised by the program and total running times from both the user and the system.

-



```
Performance counter stats for './Task1Filter wlist_match1.txt clnf.txt':

       14,733.44 msec task-clock                #    0.980 CPUs utilized
          15,012      context-switches          #    0.001 M/sec
               0      cpu-migrations            #    0.000 K/sec
          11,545      page-faults               #    0.784 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses
   <not supported>      L1-dcache-loads
   <not supported>      L1-dcache-load-misses
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

    15.040491626 seconds time elapsed

    14.481260000 seconds user
     0.269132000 seconds sys
```

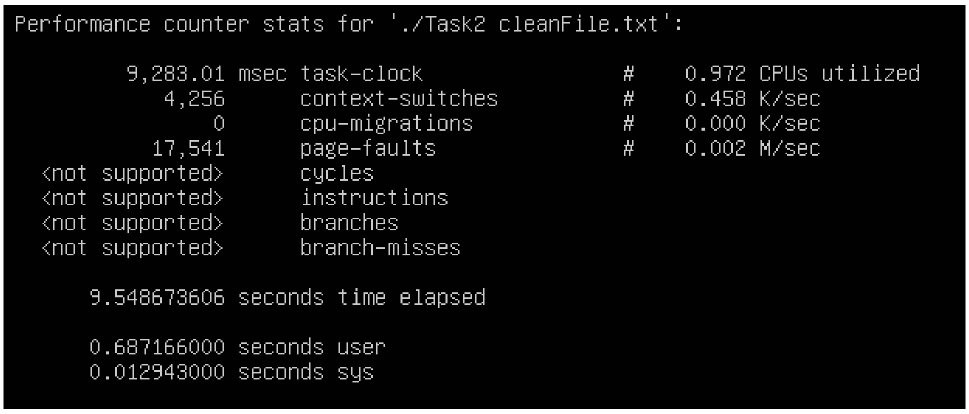
- This is the perf stats after optimising the regular expression on the grep command
- Using '^[[:lower:]]{3,15}'
- With the command ***grep -i -o -E '^[[:lower:]]{3,15}' $1 | sort -u | shuf > $2***

- 

```
Performance counter stats for './Task1Filter wlist_match1.txt cleanFile.txt':

        9,302.69 msec task-clock                #    0.958 CPUs utilized
          14,972      context-switches          #    0.002 M/sec
               0      cpu-migrations            #    0.000 K/sec
          11,614      page-faults               #    0.001 M/sec
  <not supported>      cycles
  <not supported>      instructions
  <not supported>      branches
  <not supported>      branch-misses
  <not supported>      L1-dcache-loads
  <not supported>      L1-dcache-load-misses
  <not supported>      LLC-loads
  <not supported>      LLC-load-misses

     9.711722031 seconds time elapsed

     8.924561000 seconds user
     0.399749000 seconds sys
```

- The time elapsed is less than 10 seconds which is much better than the initial command.

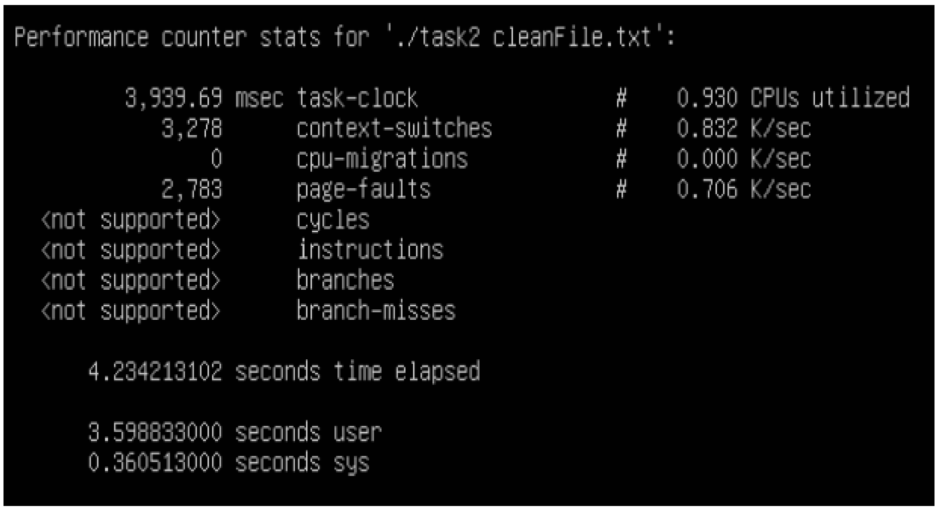
Task 2

- Fork() creates a new child process which shares the exact copy of the address space but different and share the same memory segment.
- As of the above statement, when fork() is called, 2^n of child processes are created when n = the times fork() is called. i.e., calling 3 folks() would mean 8 processes created.
- In map2(), popen() was used to execute awk command to split the clean file into individual word lengths from 3 – 15. Fork() was used to create child process to sort the created files by the 3rd character of each word for every word length 3 – 15.
  - Running map2() produces the following stats:
  - 

```
Performance counter stats for './Task2 cleanFile.txt':

        9,283.01 msec task-clock                #    0.972 CPUs utilized
           4,256      context-switches          #    0.458 K/sec
               0      cpu-migrations            #    0.000 K/sec
          17,541      page-faults               #    0.002 M/sec
  <not supported>      cycles
  <not supported>      instructions
  <not supported>      branches
  <not supported>      branch-misses

     9.548673606 seconds time elapsed

     0.687166000 seconds user
     0.012943000 seconds sys
```

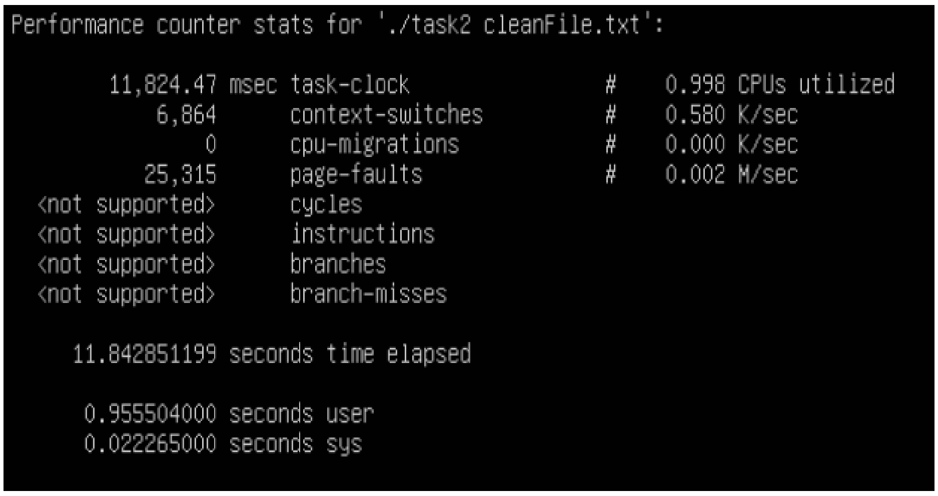
- Running Task2
  - Use: ./Task2 <clean file>

- The passed file is the clean file and used to create the slit file of lengths 3 – 15 saved as “T2words<word length>.txt”
- These split files are then used by map2() to the sort by 3rd letter and then saved as “T2words<word length>sorted.txt”
- It is recommended that the split files should be put in their own directory
  - Like putting the split files in a folder called “wordsListUnsorted” and putting the sorted files in “wordsListSorted”
- The above saved files are then used by reduce() to merge sort.

- Reduce()
  - This method merge sorts all the files 3 – 15 that were created by the map2() after sorting by the 3rd letter. This creates a new file “T2mergedFile.txt” which contains all the merge sorted data.
  - Below is the perf stats for running reduce()
  - 

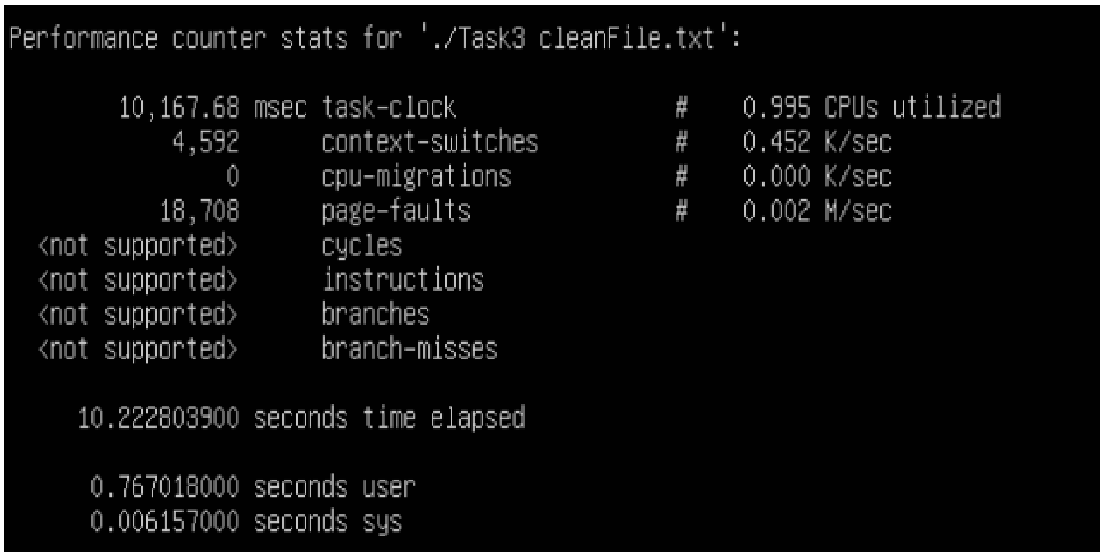
```
Performance counter stats for './task2 cleanFile.txt':

        3,939.69 msec task-clock                #    0.930 CPUs utilized
           3,278      context-switches          #    0.832 K/sec
               0      cpu-migrations            #    0.000 K/sec
           2,783      page-faults               #    0.706 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     4.234213102 seconds time elapsed

     3.598833000 seconds user
     0.360513000 seconds sys
```

  - It is expected after map in main in my program, the command line args is used by map2() method to get the clean file.
  - Running both map2() and then reduce() takes a little more than 11.8 total running time as shown below

- 

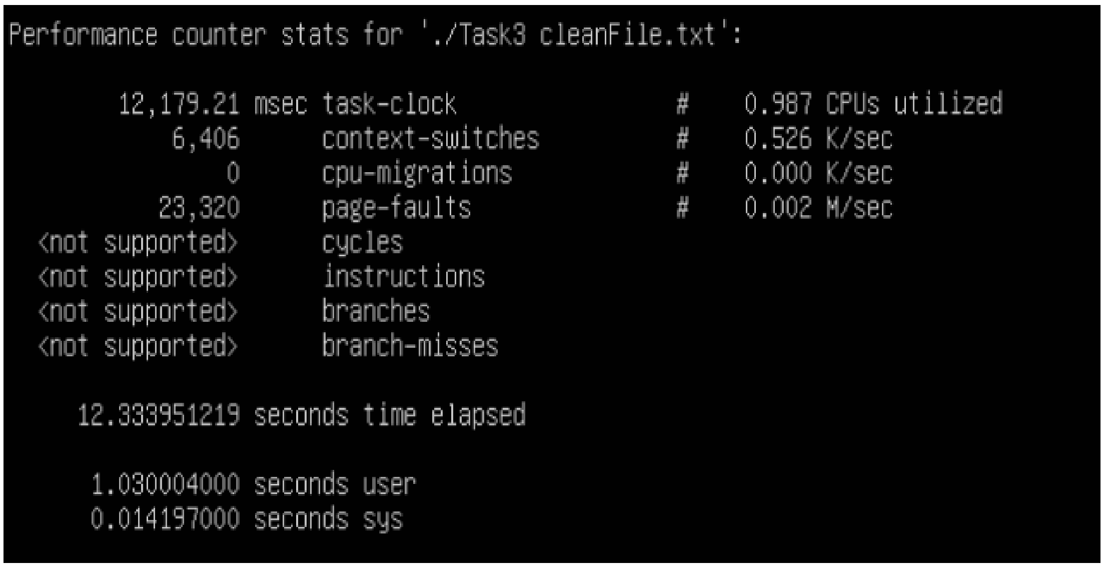
```
Performance counter stats for './task2 cleanFile.txt':

       11,824.47 msec task-clock                #    0.998 CPUs utilized
           6,864      context-switches          #    0.580 K/sec
               0      cpu-migrations            #    0.000 K/sec
          25,315      page-faults               #    0.002 M/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

    11.842851199 seconds time elapsed

     0.955504000 seconds user
     0.022265000 seconds sys
```

Task 3

- Task3 uses the file created after running the Task1 filter shell script. This file is the used by map3() and reduce using threads.
- This uses popen() in write mode to a FILE pointer in loop for each word length from 3 – 15
- Below is the result of executing map3() by itself. This takes long to run but when reduce is called after, the times are bound to increase but times also vary depending on the operating system being used.
- 

```
Performance counter stats for './Task3 cleanFile.txt':

       10,167.68 msec task-clock                #    0.995 CPUs utilized
           4,592      context-switches          #    0.452 K/sec
               0      cpu-migrations            #    0.000 K/sec
          18,708      page-faults               #    0.002 M/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

    10.222803900 seconds time elapsed

     0.767018000 seconds user
     0.006157000 seconds sys
```

- Reduce3() also uses popen to execute a cat command to read the sorted files and pipe those reading to the new file to contain the merge sorted data “T3mergedFile.txt”.
- Below are the total running times including reduce3 with perf stat

- 

```
Performance counter stats for './Task3 cleanFile.txt':

        12,179.21 msec task-clock                #    0.987 CPUs utilized
            6,406      context-switches          #    0.526 K/sec
                0      cpu-migrations            #    0.000 K/sec
           23,320      page-faults               #    0.002 M/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     12.333951219 seconds time elapsed

      1.030004000 seconds user
      0.014197000 seconds sys
```

- In comparison from the running map3() alone and running both map and reduce, the times noticeably different as it now takes a little over 12 seconds to execute both methods
- Word count for each word lengths
    - From the data file I used which from the provided site https://www.keithv.com/software/wlist/ which was the wlist_match1 source file, the most frequent word length was 7 with 211660 words of length 7.

Task 4

- Task4 works in the same way as Task3 and there aren't improvements that I can seem to make for map4() as the popen runs a command and although I am thinking of using the nice method to give words of lengths 7 more time as per my findings above, I am unable to do it due to the command executed by the popen method.
- There isn't much to do in reduce4 either. But I am certain that the threads would spend the most on words of length 7 from my finding based on the source file mentioned above.
- Map4 would have a O(n + m) time complexity (n is the split files for each word lengths and m is the merging sort part) as the read data from the file are piped and redirected into a file of length n. reduce would have just O(n) for the same reason as map4.
- Comparing longer string lengths can add up over large data sets but would only make very little difference when dealing with relatively small data files.