OSP Assignment 1

Github repository: <https://github.com/ZainBangash/OSPAssignment1.git>

Task1:

For task 1, firstly I created a bash file to clean the data given in the wlist\_all folder. Initially I started with the 12th file (wlist\_match12.txt) and worked my way up to the 1st file (wlist\_match1.txt) and experimented with the different ways to sort the data. First, I wanted to remove certain characters from the words that had characters other than letter in them. For example, the word ………apple would be filtered to apple. I decided to remove all the characters that were alphabets with the exception of the apostrophe ( ‘ ). That lead to words that wouldn’t be classified as proper words. After trying different methods to make this work, eventually I came to the realization that the best way to filter it would be to remove all words that had any illegal characters, rather than just removing the illegal characters. This did not have a very major impact on the number of words left as there were still more than a million words left after this filter. A simple grep -E "^[a-zA-Z]\*$" command was used to remove any words with characters other than the characters of the alphabet. Secondly, all words whose length was either less than 3 or greater than 15 were removed using the command grep -E '^.{3,15}$'. Unfortunately, an issue arose of words that had fulfilled the criteria of the previous filters but still wouldn’t be classified as proper words. An example could be ‘aaaaand’, it can be deduced that the word intended is ‘and’ however there are too many a’s at the beginning. There were more than few occurrences of such words, and there all words with more than one occurrence of the same alphabet consecutively were removed using the command  grep -v '\(.\)\1' . Words like ‘aaand’, ‘boooth’ and ‘dooog’ were removed however as a result words such as bareroot were also removed. There were still more than a million words left to sort and do the next tasks with so this filtering approach was kept. Duplicates were also checked for by using the following command uniq -cd command. The words were also sorted on their third letter, using the following command -k1.3 clean.txt. All in all the following coreutils were used shuf, grep, sort, uniq, wc and time (to check how long each program took).

Grep was used to find patterns in the codes, such as if there were any characters apart from alphabets or if any same letter occurred consecutively. Sort was used to sort the words on the 3rd letter using the -k argument. Uniq was used to find out if the files contained any duplicates, wc was used to see the lines of each file and

For the Task1Filter.cpp file, both the clean and dirty file paths were given in as arguments, both files are opened and each line from the dirtyfile is checked, if it passes the filtering criteria then it is added to the clean file. Methods were made to check if the word only contained alphabets and if the word contained more than one character consecutively.

Limitations:

After all the filtering there were several words that can’t be said to be of the English dictionary, however as this was the best and more efficient way to do the filtering, this method was kept. It also left a large set of data to be sorted.

Outcomes:

When we run the bash file, we get the following:

Text

Description automatically generated

When we compile and run Task1 we get the following:

Text

Description automatically generated

The program opened both the wlist\_match1.txt and finalList.txt files and wrote the filtered data from the dirty file into the cleaned file, and closed the file in 1.948s. This is an improvement from the 7.440s taken by the bash file. The bash file also sorted the words, however as we will see the c++ programs achieve the same in a much faster time.

Task2:

For Task2 I had to use forks to sort the data into 13 files, a file per word length and then perform a 13 to 1 merge sort on the files. 13 forks were made and assigned a word length it had to sort. It would take the words of the word length allotted to it from the globalArray and sort it on the 3rd letter and then write the sort list to a .txt file. The reduce method would then open all 13 files, get one word from each file, sort the list, remove the first word from the sorted list, write it to the clean file, get the next word from the file that contains the same word length as the current removed word. If the file to get words from is empty then it will remove the word from the sorted list and sort the next words and the finished file will never again be opened as there is no word of the file’s word length present to trigger a getline() from that file.

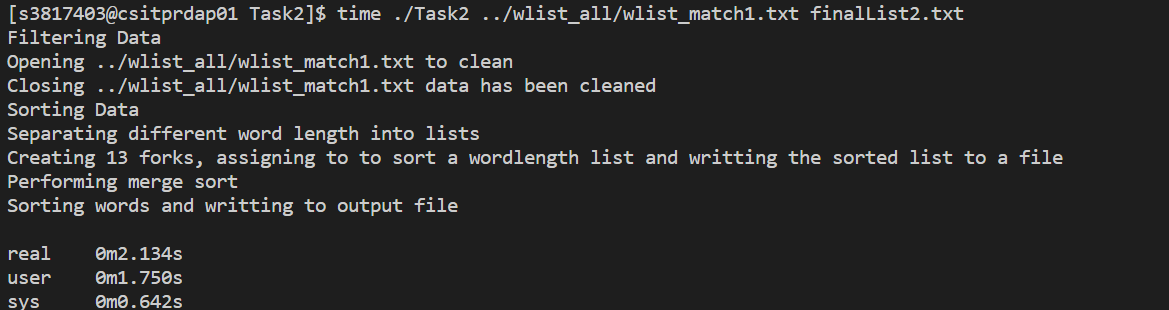
Limitations:

A limitation or error or failure of logic initially was that a file was being opened and closed every time a new word was needed to be inputted to the list, to be sorted. As there are around a million words, the whole program was taking around 15 minutes, however once the files were left open outside of the loop the program was significantly faster. Forks were only used with Map3 and not reduce as the instructions provided indicated forks were to be used for sorting the words of a certain length and outputting them to a file for its word length.

Outcomes:

As a result of sorting from the 3rd letter onwards, words that contain ‘a’ as their 1st letter rather than their 3rd letter will not be first on the list, rather words that contain ‘a’ as their 3rd letter will come before words that contain ‘a’ as the 1st but not 3rd letter.

When Task2 was compiled and ran, it produced the following results:



Task3:

2 threads were created in the main function, one thread was for map3 while another was for map3 (one for map3 and one for map4), and a further 13 were created in the map function to sort their respective word. Each thread created in map3 was for the sort function, in which each thread sorts its list of words of the allocated word length and writes to the fifo file. While reduce3 reads from the fifo file and continues with the same functionality as from reduce2.

Limitations:

Used std::sort rather than qsort, as didn’t seem to be working and I had used std::sort for the previous task. Similarly, some issues were encountered with the fifo files, as sometimes they weren’t being created (Operation not supported error) and sometimes they were created but weren’t being opened properly (bad read/write descriptor). Bad read/write descriptor seemed to be fixed when I deleted the fifo files before running Task3 again. At the end, the fifo files and the merge sort worked fine. 2 threads were created in the main function (one for map3 and one for reduce3), and a further 13 were created in the map function to sort their respective word. No threads were made in the reduce function as the specs didn’t specify. Map3 was writing into fifo files and reduce3 was reading from fifo files, if reduce3 was made to wait for map3, then an error was produced. -pthread was added to the compile command to enable it to run.

Outcomes:

Text

Description automatically generated

Task 3 can be seen to take more time than Task2, however this is due to the changes that had to be made to reduce3 to compensate for the fifo files. If txt files were used instead of fifo files, then Task3 would have taken less time than Task2. Time is lost in reduce3, for adding the words from fifo files to a vector then getting words from the vector and preforming the sort and remove to clean file. With txt files this would take a lot lesser of a time.

Task4:

The only real change to this task as compared to the previous task is in Map4. Priority is changed of every file depending on how many words are for that word length. With respect to the big 0 representation, it can be denoted as O(n log(n)). Where n is the number of words in the list and log(n) is the complexity for std::sort. The number of words in the list and the length of the word also matters. If the words of length 15 were the same as length 7, the difference of the sort function would become obvious. I have decided to change the priority of the threads based on the number of words assigned to that thread. Pthread\_setschedparam was used to change the priority. Thread in charge of words of length 7 were given a higher priority than words of length 15 or 3, that have a smaller number of words.

Limitations:

Have to run using sudo command to set priority for threads, other the error Operation not permitted is encountered.

Outcomes:

Using the command

sudo time ./Task4 ../wlist\_all/wlist\_match1.txt finalList4.txt we get the following result

Text

Description automatically generated

The task was finished in about 1.2 seconds. I ran this on my laptop rather than the server as I had to use the sudo command so that the priority of the threads could be changed. The faster threads will be scheduled later, as the slower threads will be given priority over them, resulting in a lower wait time for the faster threads.