**Operating System Principles – Assignment 1**

***Matthew Yamen***

***S3902219***

**Performance Measurement**

I used C++ to time the code, as it is more accurate than timing the execution of how long it takes the entire command

temp

Task 2-4: Clean up temp intermediary files and fifos using unlink.

For Tasks 2-4, the merge function seems to be the bottleneck, as it slows down everything else.

**Task 1**

Using the C API regexes was MUCH faster than the C++ standard libraries std::regex. This cut down time drastically – from an average of 5-6 seconds to 1-2 seconds.

**Task 2**

Instead of reading data from original file, I am going to use the same task\_filter function that will be used in task 3 and was used I task 1, so as to keep the algorithms as similar as possible, to eliminate bias that might affect the validity of the results.

**Task 3**

Did not wait for the 13 FIFO files to be open for writing by map3 to perform the reduction step, as this is an unnecessary bottleneck, and will only slow down the code. It makes much more sense for the files to write and read in any order, and when one fifo has both a read and write thread attached, then it begins and that thread continues.

**Task 4**

Optimisation 1: Put threads that aren't writing to sleep.

This optimisation is useless, because when a thread is waiting for I/O in THIS CASE (i.e., in the case of the named pipes, it is waiting for the other end to be opened), it is put to sleep and does not use any CPU.

Optimisation 2: Prioritise threads with bigger lists to process.

Result: Likely made it slower on my laptop. This is not a good form of optimisation, because all the threads have a nice value of 0 by default anyway, and since I do not have superuser privileges on the teaching servers, I cannot use negative nice values to increase the priority. This is a problem because, we can only add positive nice values, which means some threads will be prioritised lower than other threads that belong to totally different processes (since all processes/threads default to 0 nice value and PR 20), and so, our program is actually getting less CPU time than if we had left the nice values alone (to default to their minimum). This would only be effective if we could use negative nice values, so that we can prioritise the heavier threads WITHOUT the worry of other unrelated threads having a higher priority over our ones.

**Task 5**