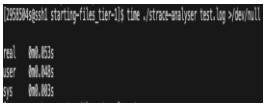
AE2 Report

Status

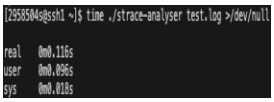
The program compiled successfully and achieved Tier 3, as it implements multi-threading.

Sequential and 2-threaded runtimes

Using the original sequential analyser, the runtime is obtained as shown below:



And using the multi-threaded implementation (with one thread), the runtime is as shown below:



Runtime with Multiple Threads

Implementing multiple threads the recorded **real** runtimes for 1, 2, 4, and 8 threads over 3 runs each and their medians are as follows:

Threads	1	2	4	8
Execution Time/Run 1	0m0.114s	0m0.128s	0m0.142s	0m0.159s
Execution Time/Run 2	0m0.124s	0m0.125s	0m0.145s	0m0.145s
Execution Time/Run 3	0m.0.121s	0m0.131s	0m0.140s	0m0.167s
Median Execution Time	0.1197s	0.1280s	0.1423s	0.1570s

Discussion

From the resulting execution times, it's evident that multithreading does not help speed up the runtime. On the contrary, as the number of threads increases so does the runtime, thus degrading the performance speed.

Firstly, in terms of task size, I have only used the provided test.log as an input, which when analysed with the original version of sequential analyser, proved to be sufficiently fast. I think the input file in itself was not big enough to warrant needing multithreading.

Secondly, looking at the program, multithreading was only implemented in the worker threads, while still maintaining a singular worker thread. This might be the cause of the speed depreciation, especially keeping in mind the fact that my program has implemented a shared mutex and WorkQueue. Thus, every time a worker/consumer calls the pop() function, it locks the queue and results in more time spent by the workers waiting instead of processing each line. This aligns with the resulting runtimes, showing that with more workers, more time is added to wait instead of processing.