Aldo Tali

Section 1

Operating Systems

CS342 Project 1 Report

Fall 2018-2019

Aldo Tali

21500097

22.10.2018

Purpose of report:

The following is a report that tries to answer the third part of the Operating Systems Project 01 Fall 2018 in Bilkent University. The project itself consists of creating two projects, one for testing the use of multiple processes and the other the use of multiple threads. As such, in order to test all these, several experiments were conducted and in this report the experiments are named as “runs”. A run is a program execution. For each of the test cases six different runs are given that show the execution time of the project. At the end each case is accompanied with its mean runtime and the standard deviation through different runs.

Parta)

This part contains two sections of experiments. The first one is the general case where the input size is kept constant but as the processors and the threads number increases these files also increase, meaning that each process/thread deals with a constant input size. The second part however keeps the input constant throughout the experiments, meaning that even if the application is run with 2 processors then the files given as input would be of size (size/2) each which amount to total size being fixed. We will see how both these approaches are useful in getting our insights.

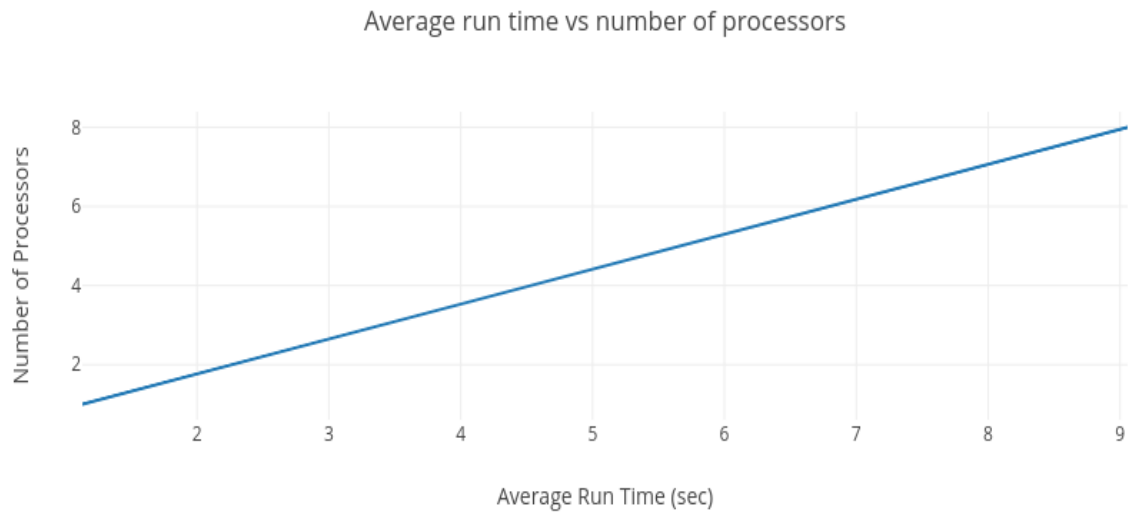
Note: For the first approach the following experiments were conducted with files containing 6551986 numbers each. The number is arbitrary, it just seemed enough based on the experiments previously made by myself to see results that run in more than a second. Also the given times are in seconds and the number of bins is 50. The latter is also arbitrary since I believed it to be enough for the distribution of the numbers in the histogram. Also the numbers given are in between the range 1-1000000 and are randomly generated through an online generator tool. Any of the values assumed above could be changed to the relevant other values that the user needs. These however were used in my report.

Note that running 1 processor means that 1 of the files described above was given as input whereas 2 means that 2 of the files are given as input since each processor takes exactly one file. This is the same for 4 and 8 meaning 4 files and 8 files respectively.

Table 1.0 Values for No. of Processors and Run Time

Processors	Run1 (sec)	Run2 (sec)	Run3 (sec)	Run4 (sec)	Run5 (sec)	Run6 (sec)	Mean (sec)	S.Deviation
1	1.1008	1.1799	1.1372	1.1098	1.1208	1.1348	1.1306	0.025523
2	2.2756	2.2287	2.2496	2.2666	2.2581	2.2437	2.2537	0.015314
4	4.4733	4.6223	4.5030	4.5397	4.4733	4.5840	4.5326	0.055749
8	9.0353	9.0351	9.0291	9.0208	9.0710	9.1556	9.0578	0.046464

Number of Processors vs Avg. Run time.



Standard deviation on Runtime per number of processors chart.

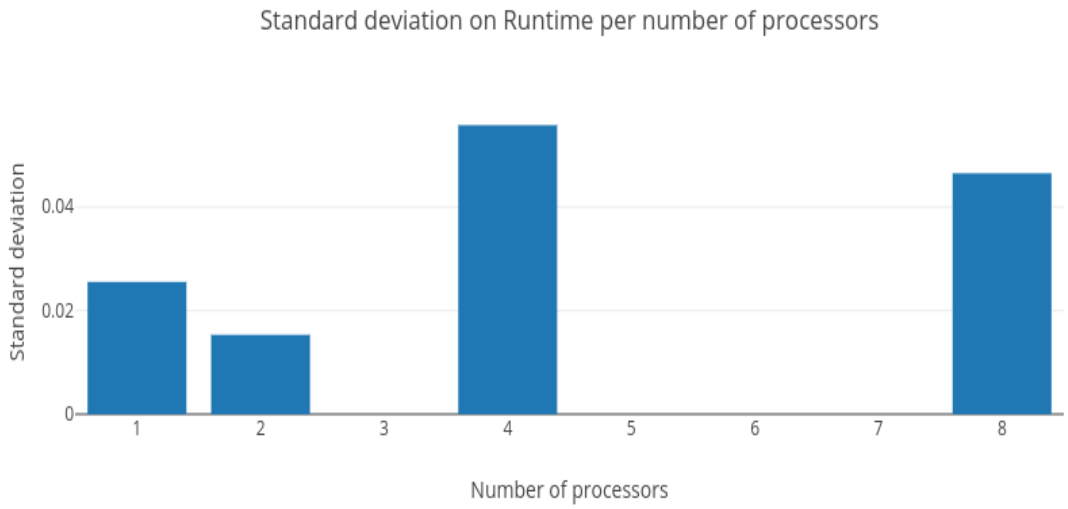
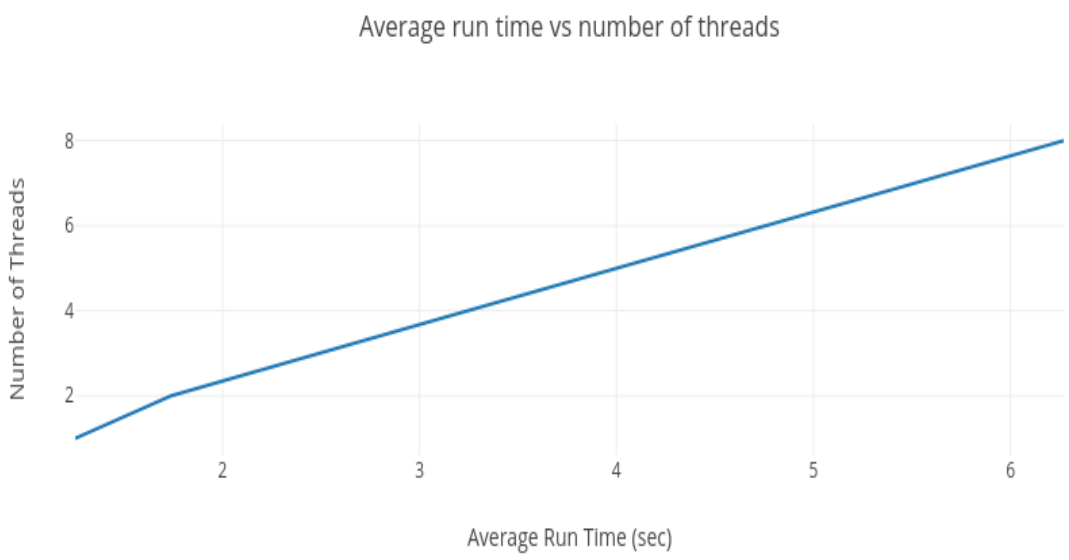


Table 2.0 Values for No. of Threads and Run Time

Threads	Run1 (sec)	Run2 (sec)	Run3 (sec)	Run4 (sec)	Run5 (sec)	Run6 (sec)	Mean (sec)	S.Deviation
1	1.1951	1.2689	1.2781	1.2641	1.2223	1.2844	1.2522	0.0323589
2	1.6810	1.7445	1.7738	1.6449	1.8083	1.7784	1.7385	0.0574649
4	3.1179	3.4505	3.2384	3.2117	3.2056	3.2260	3.2416	0.1011471
8	6.2296	6.2747	6.3369	6.1970	6.2839	6.3022	6.2707	0.0459816

Avg. Run time in sec vs number of threads



Standard deviation on Runtime per number of threads chart.

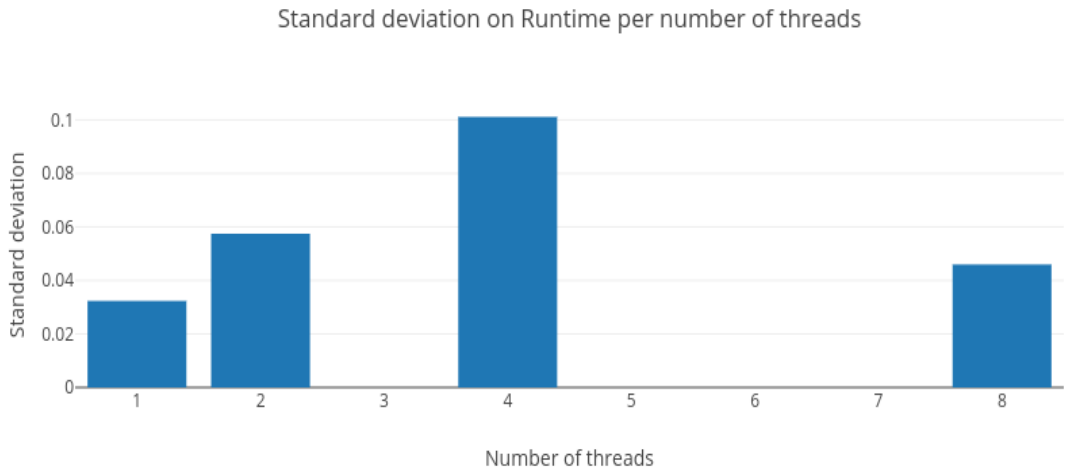


Table 3.0 Values for No. of Processors and Run Time for same file splitted into k pieces.

Processors	Run1 (sec)	Run2 (sec)	Run3 (sec)	Run4 (sec)	Run5 (sec)	Run6 (sec)	Mean (sec)	S.Deviation
1	0.6803	0.6910	0.6869	0.6828	0.6864	0.6849	0.6854	0.0033573
2	0.6856	0.7001	0.7130	0.7011	0.6985	0.6944	0.6988	0.0081885
4	0.7003	0.7032	0.7164	0.7135	0.6948	0.7022	0.7051	0.0075210
8	0.6931	0.6896	0.6937	0.6863	0.6897	0.6929	0.6909	0.0026131

Run Number vs Run time

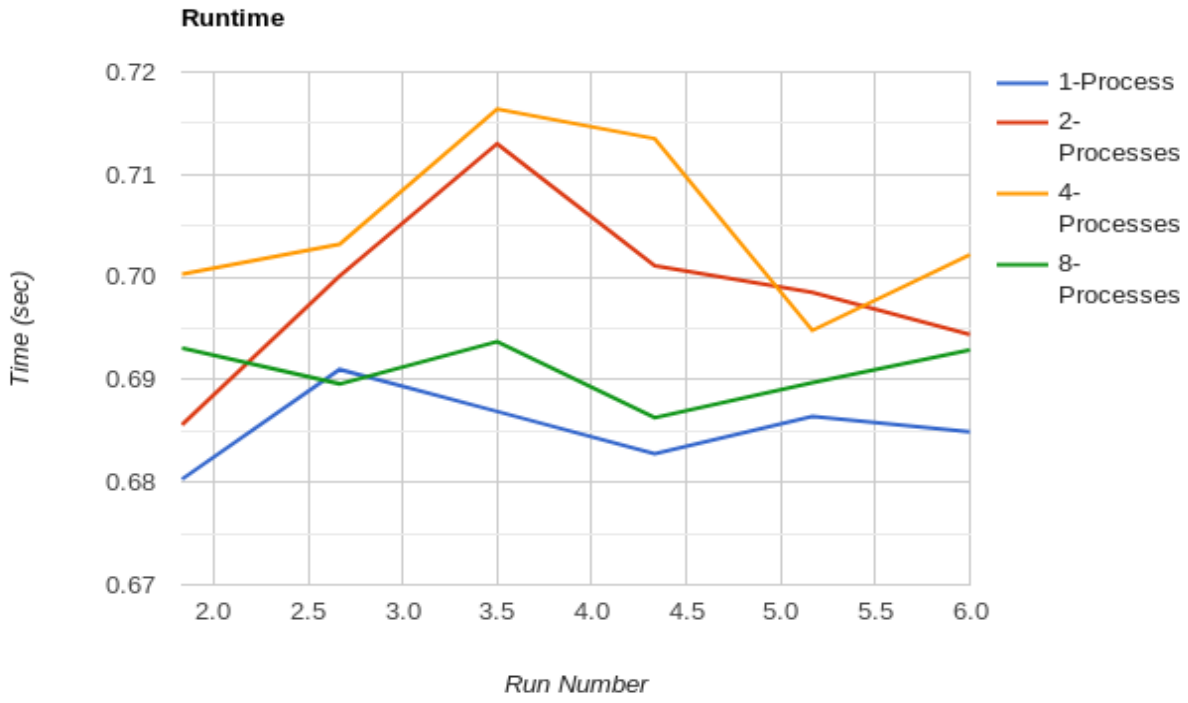
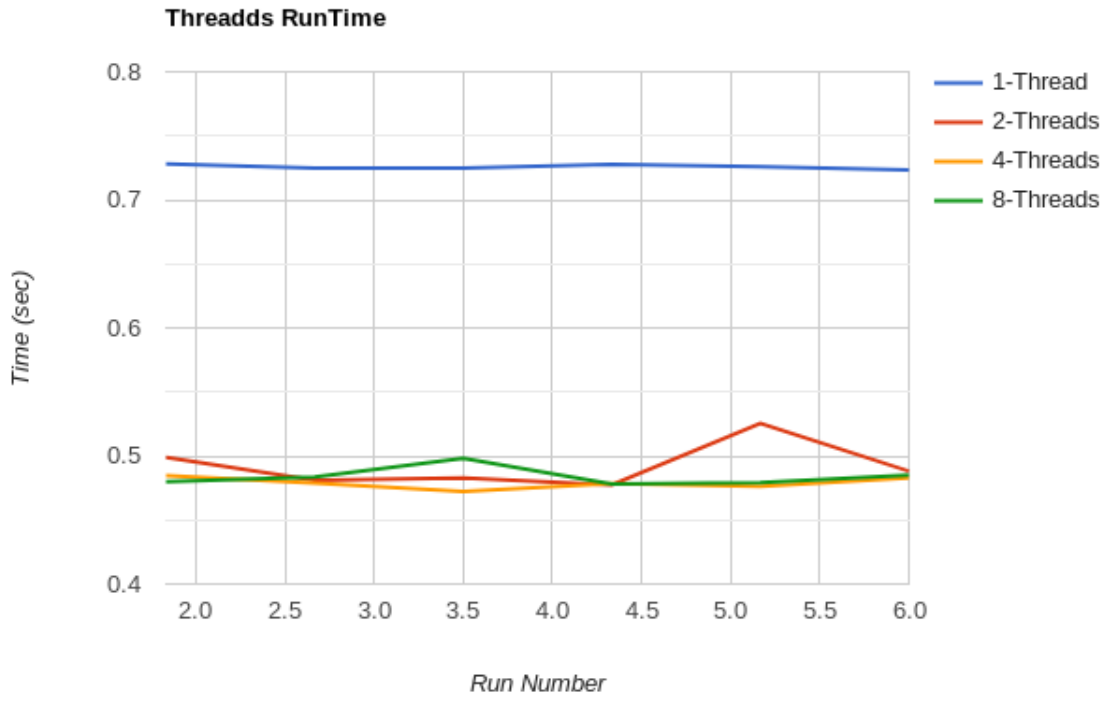


Table 4.0 Values for No. of Threads and Run Time for same file splitted into k pieces.

Threads	Run1 (sec)	Run2 (sec)	Run3 (sec)	Run4 (sec)	Run5 (sec)	Run6 (sec)	Mean (sec)	S.Deviation
1	0.7281	0.7250	0.7249	0.7277	0.7259	0.7232	0.7258	0.0016892
2	0.4988	0.4811	0.4827	0.4774	0.5254	0.4881	0.4923	0.0163049
4	0.4846	0.4789	0.4723	0.4785	0.4764	0.4829	0.4789	0.0040508
8	0.4799	0.4836	0.4980	0.4781	0.4793	0.4851	0.484	0.0067216

Run Number vs Run time



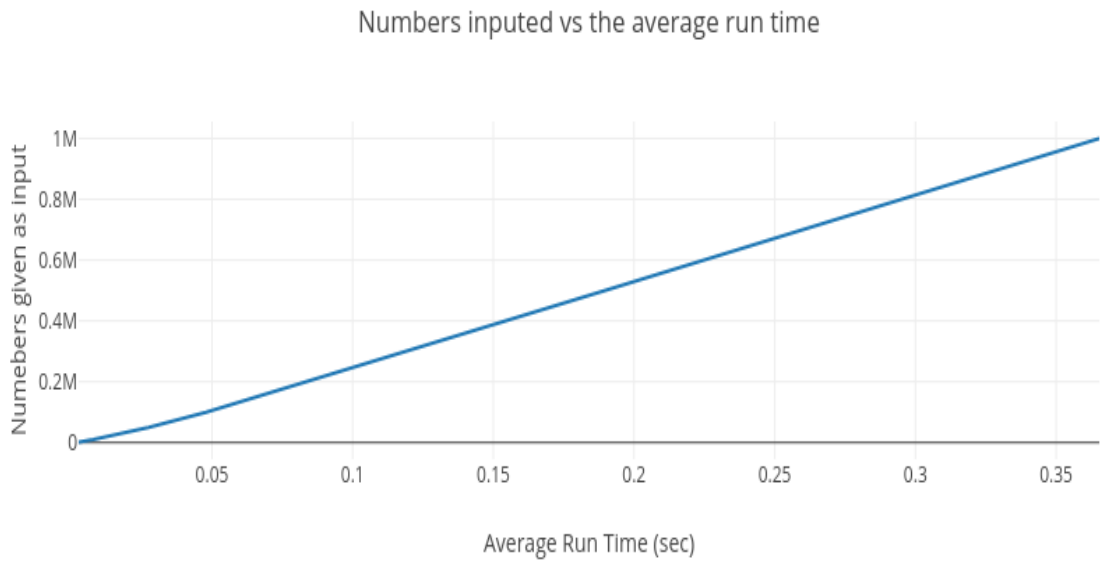
Partb)

Note: The following results were calculated by running the *phistogram* executable several times. Therefore the results are applicable for the 2 process execution of the given files. To test the run time with different sizes in here it is assumed that a size is given by the number of numbers that are inserted to the input files. As such a file of size 1000 means that there are 1000 numbers in the files given as input. The numbers are completely randomly generated and they are within the range 1-1000000 inclusive. The number of bins used was 50. These were personal choices but the experiments can be repeated on different sizes as well. Lastly the terminology explanation given in the previous part also holds here for the calculations.

Table 5.0 Values for Input Size and Run Time of a 2 process application.

Numbers given as input	Run1 (sec)	Run2 (sec)	Run3 (sec)	Run4 (sec)	Run5 (sec)	Run6 (sec)	Mean (sec)	S.Deviation
1000	0.0028	0.0031	0.0027	0.0024	0.0019	0.0029	0.00263333	0.00039016
5000	0.0047	0.0054	0.0046	0.0068	0.0059	0.0060	0.00556666	0.00076739
10000	0.0082	0.0077	0.0078	0.0078	0.0075	0.0084	0.0079	0.00030551
50000	0.0299	0.0295	0.0252	0.0283	0.0249	0.0273	0.02751666	0.00193599
100000	0.0459	0.0495	0.0439	0.0539	0.0444	0.0501	0.04795	0.00355047
500000	0.1849	0.1933	0.1885	0.1995	0.1884	0.1847	0.1898833	0.00516347
1000000	0.3484	0.3923	0.3494	0.3579	0.3614	0.3827	0.36535	0.01653348

Average Run Time vs the numbers given as input (file size) of a 2 process application.



Insights:

Regarding the part a, when dealing with a fixed input size file the assumption would be that when diving the work in different processes then a relevant speed up would be obtained. As seen from the results such a thing does not happen in practice. We can see from the plot of Table 3.0 that 1 Process application runs faster than the 2 and 4 process applications. A reason for this could be that overheads are payed due to the fact that each processor also has to write the intermediate files besides the computational work that is present. Also there is no certainty that all the code is non-blocking. To support these claims we can see that 8-Process

application which receives much less numbers as input per processor has a better performing time than the 2 and 4 processors. Also this is supported in the plot of Table 4.0. As we can see from the plot in the case of running the threads, the 1 thread application is much slower than the 2,4 or 8 thread application since there is no overhead in reading files. However it also shows that the 2,4 and 8 thread applications seem to reach a lower bound to the fact that there are no more non blocking requests that can be executed faster. If this argument was not valid then we would expect from the plots of Table 1.0 and Table 2.0 (these show how the run time fluctuates when you give each process/thread the same input size file) to show that the run time increases proportionally to the number of processes/threads. Although the latter seems valid since both graphs seems liner if we examine more closely we see that the 8 process application takes only about 6 times as much as the 1 process application, instead of 8, meaning that even in this case processes/threads improve the run time of the overall application but not on a linear scale to the increase of processes/threads.

Regarding part b, we can see from the plot of table 5.0 that with the increase of the input size there is an approximately linear increase in the run time. Although this might seem not logical at first I argue the opposite. First of all if one examines the plot carefully the inclination is not fully linear, however because of the sample size and environment it appears to be so. Nevertheless it

can be approximated to a linear function since with the increase of files more files need to be written and the overhead is also increased. In here in order not to be repetitive I would like to state that the argument stated for part a also holds here. In addition it is worth arguing that in the case that these experiments were conducted with a threaded application the graph would probably be different.

To conclude both cases would improve the overall execution if the application has non-blocking execution calls. Nevertheless the improvement is bounded by a certain limit which is determined by the hardware capabilities of the laptop.