Bilkent University

Department of Computer Engineering

CS342 Project 1 Report

Section 2

Processes, IPC, and Threads

Group Members:

Efe Beydoğan 21901548

Emir Melih Erdem 21903100

Introduction

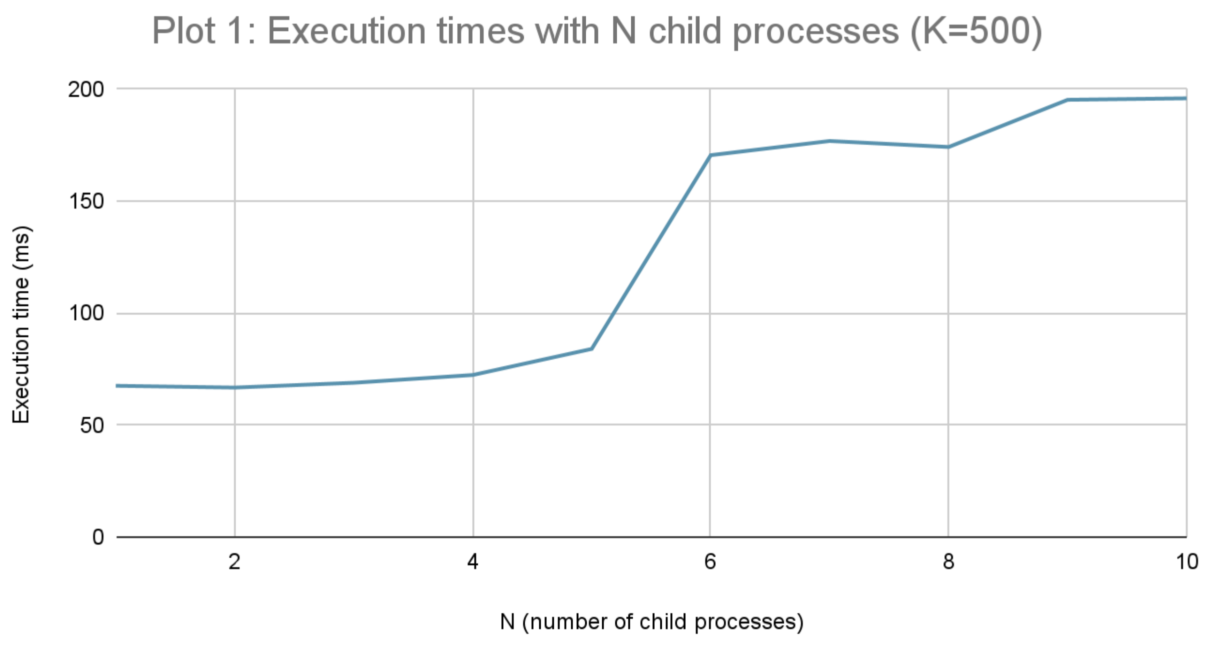
To measure the performance of our project, we have conducted four experiments. In the first part, we experimented with processes by first fixing K and then N to certain values and measuring the time it takes to run the program. In the second part, we have done the same experiments with the same values using threads.

In order to make sure we were reflecting realistic values in our report, we ran each test five times and took the average of the duration values we received.

Experiments for Part A

1) Execution time for various values of N (number of child processes), K is fixed to 500

N	1	2	3	4	5	6	7	8	9	10
Time (ms)	67.56	66.76	68.90	72.37	84.02	170.55	176.88	174.20	195.28	195.94

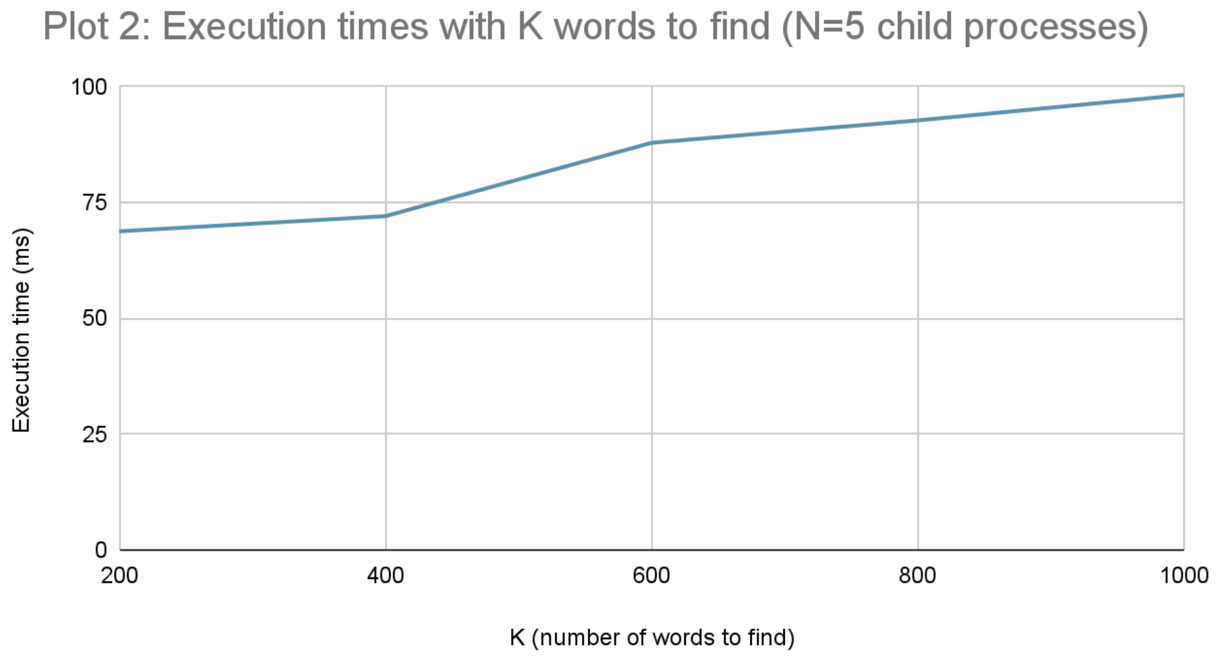


As can be observed from Plot 1, as the number of child processes increases, the time it takes for the program to run also gradually increases. This is because we use quicksort to sort the words to select the K most frequent ones, and the time complexity is, therefore, N*K*log(N*K). K is fixed to 500 in this case, but N increases and so the program generally takes longer to execute. Further, the overhead of creating and managing additional processes and the context switches between them also contribute to longer runtimes.

As we increase N from 5 to 6, there is a much sharper increase in the runtime of the program. When N = 5, the program runs for 84.02 ms, whereas when N = 6, the runtime increases to 170.55 ms, almost twice the previous runtime. This is due to the fact that the computer we measured the runtimes on has a 6-core CPU. When one core is being used by the parent, 5 child processes can occupy the remaining cores to run in parallel. So at most 6 processes can be run in parallel in the CPU, and when the number of child processes becomes 6, the runtime doubles as now there are 7 processes running together, and they cannot all be run in parallel.

2) Execution time for various values of K (number of words to find), N is fixed to 5

K	200	400	600	800	1000
Time (ms)	68.79	72.06	87.90	92.73	98.24

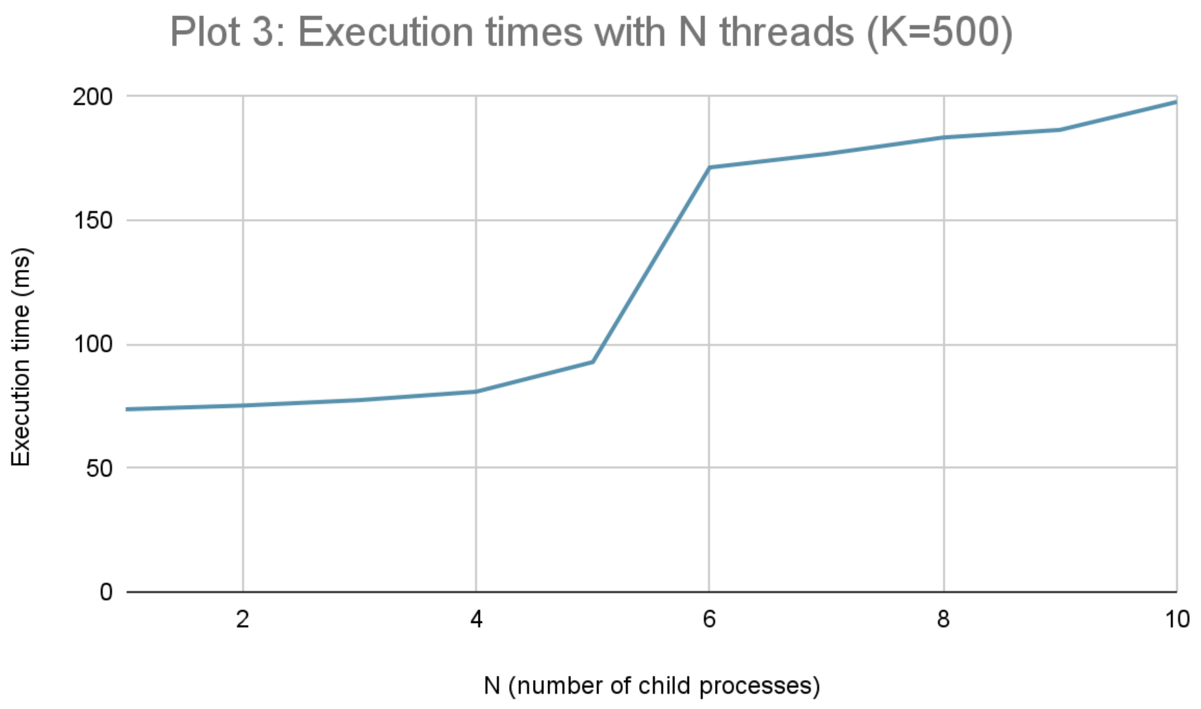


Plot 2 suggests that as the number of words to find increases, execution times increase again. Likewise, due to the time complexity of quicksort (NK*log(NK)), when N is fixed to 5, an increase in K results in longer running times for sorting the common words brought together from the child processes.

Experiments for Part B

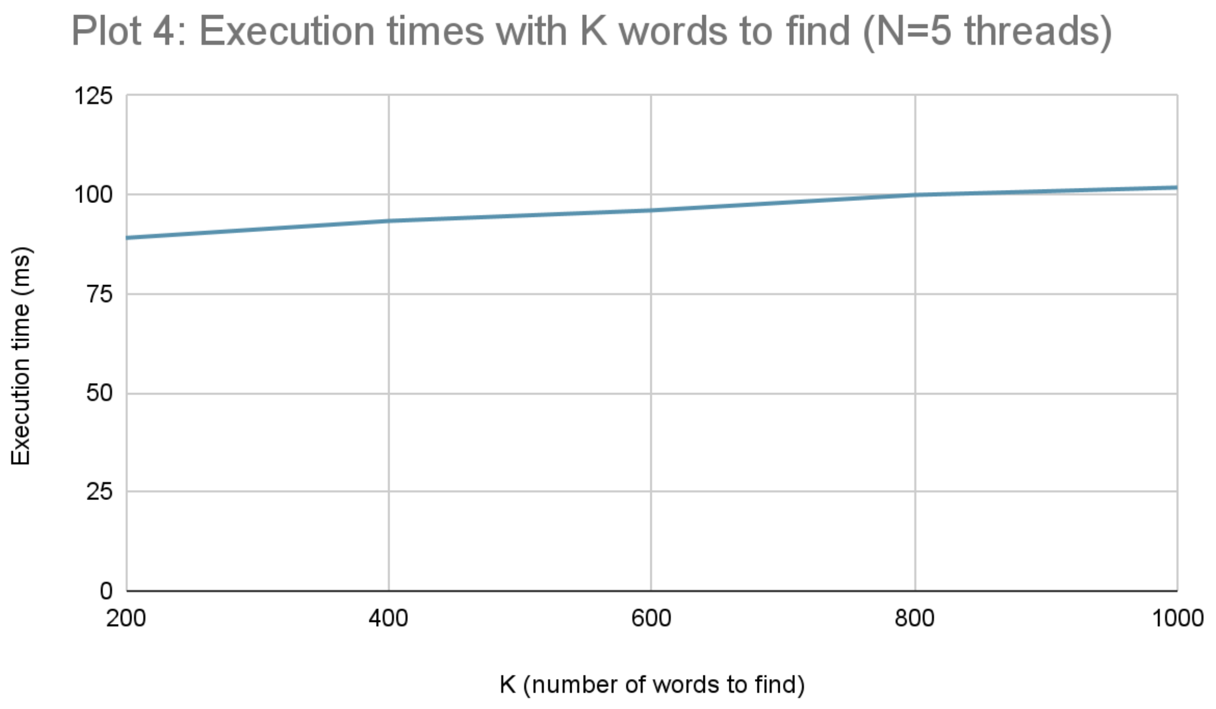
1) Execution time for various values of N (number of child processes), K is fixed to 500

N	1	2	3	4	5	6	7	8	9	10
Time (ms)	73.75	75.25	77.45	80.84	92.83	171.38	176.85	183.48	186.56	197.85



2) Execution time for various values of K (number of words to find), N is fixed to 5

K	200	400	600	800	1000
Time (ms)	89.09	93.36	96.03	99.92	101.80



As can be observed from Plots 3 and 4, the runtimes for Part B are very similar to those in Part A. As the number of child processes increases, the time it takes for the program also gradually increases, as explained in Part A, this is due to quicksort.

As we increase N from 5 to 6, there is a much sharper increase in the runtime of the program. This was explained in Part A to be due to the CPU having 6 cores.

Conclusion

The runtimes of the program for the tests in parts A and B turned out to be very similar. We actually would have expected the program for Part B to be faster than Part A, as threads share the code and the global variables of a process, hence generating less overhead compared to processes where for each child process, the address space is copied. This could be explained by the fact that we are using shared memory for interprocess communication in Part A, so our program spends significantly less time in kernel mode compared to the time it would’ve spent in kernel mode if we had used the message passing IPC mechanism. The huge overhead of switching to the kernel for IPC is overcome by using shared memory. We verified this by running our program with the “time” system call and observing that the value for “sys” is very small, almost 0, whereas the “user” value is much higher.

Also, during these experiments, very long input files are used, each with approximately 40,000 words (formed by around 220,000 characters). Reading, sorting, processing, and writing this vast data dominate the execution time. This may also explain why there are no noticeable differences in running times between parts A and B.