CS 342 Operating Systems

Spring 2023

Project 1: Processes, IPC, and Threads

Group Members

Esad İsmail Tök – 21801679

Elifnur Alsaç - 21601098

Introduction:

- The time measurements that are investigated in this report are obtained by the functions of ***<time.h>*** header file. After completing all two programs, We have used ***clock()*** function to find the start and end times of the applications and then calculated the elapsed time during the application in seconds using

  double duration = ((double)end - start)/CLOCKS_PER_SEC.

- By obtaining the elapsed times of the executions in both ***proctopk*** and ***threadtopk*** programs, we are able plot the changes on the specific cases:
  - N (number of input files) is increasing where K (number of most frequent words) stays the same in the ***proctopk*** program.
  - N (number of input files) is increasing where K (number of most frequent words) stays the same in the ***threadtopk*** program.
  - K (number of most frequent words) is increasing where N (number of input files) stays the same in the ***proctopk*** program.
  - K (number of most frequent words) is increasing where N (number of input files) stays the same in the ***threadtopk*** program.

- In this report we examine the above cases and provide plot figures to compare the running times of the programs when we use processes to implement Top K Frequent Word problem vs. when we use threads to implement Top K Frequent Word problem.

N (number of input files) Increases, K (number of most frequent words) Stays the Same:

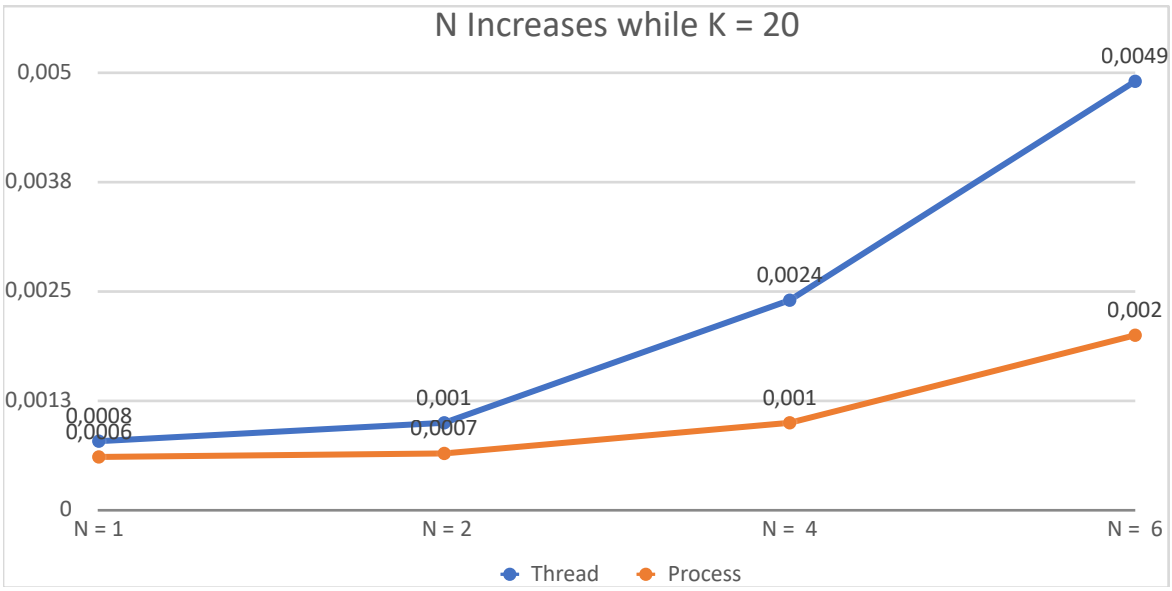

Figure 1: Plots of proctopk and threadtopk programs while K is same and N increases

- In this phase of the experiment, the behaviours of the programs ***proctopk*** and ***threadtopk*** is measured and plotted at the same graph. The X-axis of the graph represents the number N which is the number of input files that are passed as arguments. The Y-axis of the program represents the execution time of the programs that is measured in seconds. Orange colored line indicates the program ***proctopk*** and blue colored line indicates the program ***threadtopk***.

Discussion:

- From *Figure 1* it can be seen that the execution of both programs increases as N gets higher. It is an expected outcome since we are creating the same numbers of child processes and threads as the number of input files which is N. The more number of processes and threads, which we can generally tasks, the more CPU resources that our program will require. Also the time required to create and terminate child processes and threads together with the time required to do context switches makes both programs to run slower as the number N increases. So while we are increasing the number of tasks, our CPU runs all the tasks concurrently but not necessarily in parallel. The reason why these task cannot run in parallel is because that the Ubuntu VM that the programs run has only one core so each task need to run on this core without parallelisation.
- Since the VM has only one core dedicated and there is no parallelisation of the task execution, ***threadtopk*** cannot make use of the parallelisation benefit that would have been provided my a multicore system. Therefore it can be seen that rather than having a better performance compared to ***proctopk***, our ***threadtopk*** program has a worse performance compared to ***proctopk***. In general using threads takes less time and advantageous compared to the child process usage since creating and terminating threads takes less time than the ones in processes. For the context switches of threads the same conditions holds so that it takes less time than it does in child processes. The reason that thread takes less time is because threads share some memory portions of their parents other than stack and program counter. However, the reason why ***proctopk*** takes more time than ***threadtopk*** might be the case that the Ubuntu VM has only single core and the additional threads might be overwhelming the single CPU rather than helping.

K (number of most frequent words) Increases, N (number of input files) Stays the Same:

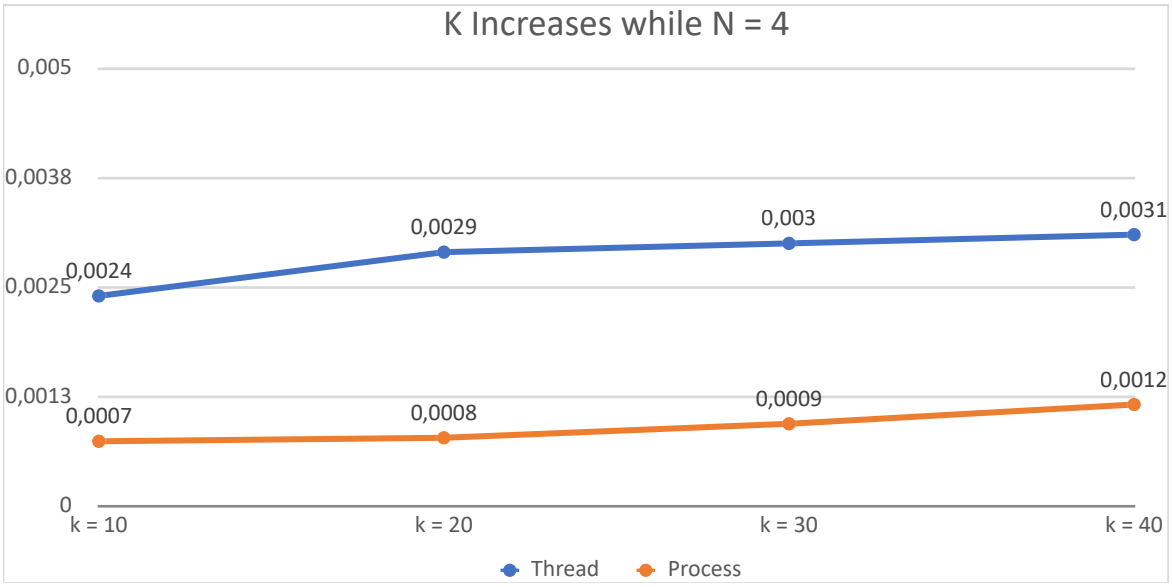

Figure 2: Plots of proctopk and threadtopk programs while N is same and K increases

- In this phase of the experiment, the behaviours of the programs ***proctopk*** and ***threadtopk*** is measured and plotted at the same graph. The X-axis of the graph represents the number K which is the number of most frequent words that we are looking in the files. The Y-axis of the program represents the execution time of the programs that is measured in seconds. Orange colored line indicates the program ***proctopk*** and blue colored line indicates the program ***threadtopk***.

Discussion:

- From *Figure 2* it can be seen that the execution times (in seconds) of the two programs ***threadtopk*** and ***proctopk*** are slightly increasing as the number K increases from 10 to 40.
- It is expected that the time it takes between the different values of K should not be much different because in the implementation of the programs, minheap data structure is used to store the the most frequent occurring words. Since the asymptotic insertion and access time into the minheap data structure takes O(logn) time, we can be sure that the increase in the number of most frequent words that we need to store should not increase that much. Therefore in both programs we see slight increases among the different K values.
- The reason why ***threadtopk*** program does not take less time than ***proctopk*** program is similar as in the case of "K (number of most frequent words) Increases, N (number of input files) Stays the Same" in the above chapter. Normally we would expect that a multithreaded program to take less time that a multiprocessor program because it better utilized the multicored system, namely the CPU's of the system. However, in this case, our Ubuntu VM has only a single core dedicated. Therefore, the 4 worker threads created in this scenario cannot make use of the multicore property. So instead of helping in terms of performance, ***threadtopk*** make things worse while there is only one core in the system. So ***proctopk*** behaves better in terms of execution time for each value of K.