Bilkent University

Department of Computer Engineering

2022 - 2023 Fall Semester

CS 342 Operating Systems

Project #1- Processes, IPC, and Threads

Authors:

Efe Erkan	21902248
Recep Uysal	21803637

Instructor: İbrahim Körpeoğlu

Implementation

In this project, we implemented two console applications that find word frequencies in a given input data set, most generally regular text files. These two applications do exactly the same task, however, the first application was implemented by using POSIX message queues and concurrent processes, the second application was implemented by using concurrent threads. In other words, in the first app, the parent process creates child processes for file processing and in the second app, file processing is done in concurrent threads.

The first application's name is *pword* and POSIX message queues were used for IPC (Interprocess Communication). Parent process creates child processes for each given file and these child processes send a group of words via message queue at a given size in bytes. While child processes are sending messages to the message queue, the parent process collects these messages and it inserts each word in the message into a binary search tree. A special binary search tree was implemented for this project and this tree contains nodes which consist of a string and a count number associated with the string. While inserting a string, the binary search tree also calculates the number of appearances (frequency) of the string. The reasons for using a binary search tree are not knowing the total word count in the files and efficiency provided for this data structure.

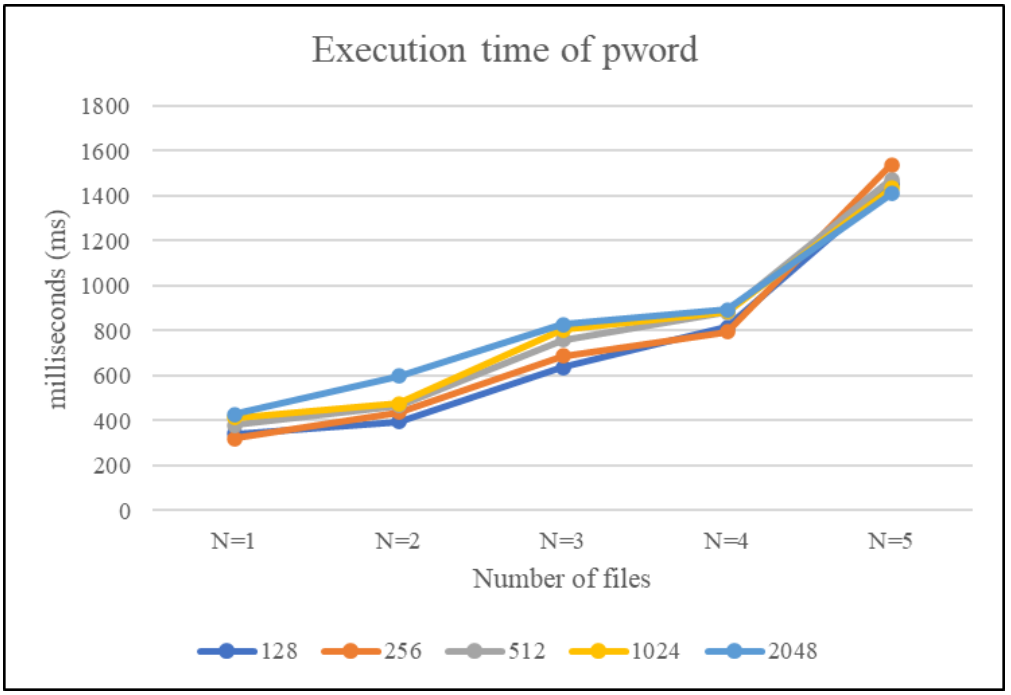
In the second application *tword*, concurrent threads were used instead of processes by using PThreads in POSIX. Due to the fact that threads share global variables, each thread uses a global binary search tree for inserting words one by one. In other words, each thread inserts strings into a single binary search tree and the tree calculates word frequencies like in the first application.

Experiments

In this part, several experiments were performed in order to observe the performance of *pword* in various file numbers and message sizes and the performance of *tword* in various file numbers. In *pword*, message sizes of 128, 256, 512, 1024, 2048 bytes and file numbers of 1, 2, 3, 4, and 5 are used for testing purposes. In *tword*, file numbers 1, 2, 3, 4, and 5 is used for testing purposes. In order to do this, gettimeofday() function in *sys/time* library was used for measuring the execution time of each case.

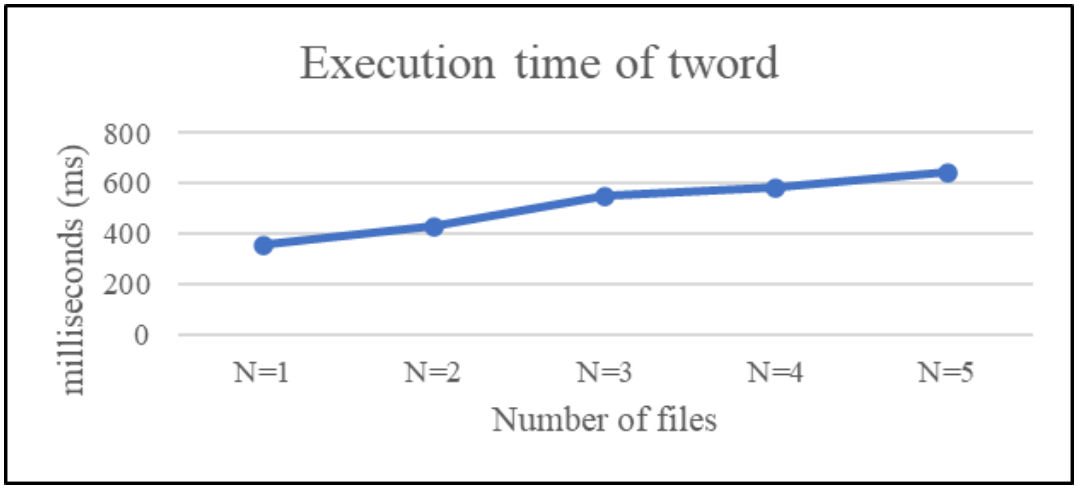
Execution time of pword in milliseconds

Message size (bytes)	N=1	N=2	N=3	N=4	N=5
128	342	394	636	815	1452
256	322	437	688	796	1538
512	380	467	758	883	1472
1024	414	477	805	886	1435
2048	429	598	829	893	1411



Execution time of tword in milliseconds

N=1	N=2	N=3	N=4	N=5
356	430	549	584	645



Evaluation

Looking at the *pword* program and if we focus on the number of files while fixing the message size, it can be said that the execution time of the program rises approximately linearly. The reasons for that can be the concurrency of the child processes. If the program was designed in a way that the files are processed sequentially, we would expect the execution time to rise more steeply. For example, the execution time of when N equals five would be five times of the execution time of when N equals one (not exactly but similar). As a result, when concurrency is introduced to the program, the rise of the execution time is slower and the main rise of the execution time can be the context switch latency and the number of files to be processed.

If we focus on the message sizes while fixing the number of files, it can be said that the message size has no significant effect on the execution time. The main reason for this can be inserting the words one by one independently with the number of words in a message. When the message size is bigger, the frequency of the processes' access to the message queue becomes lower and the number of words in a message increases. When the message size is lower, the frequency of the processes' access to the message queue becomes higher and the number of words in a message decreases. Overall, the total number of words to be inserted

into the binary search tree does not change and the execution time is similar among different message sizes.

Looking at the *tword* program, the execution time rises in a slow fashion similar to the *pword* program because of the concurrency. If we compare the execution times of tword and word, it can be said that *tword*'s execution time is smaller than *pword*'s execution time. The reason for this situation can be the usage and execution overhead of message queue in *pword* and the ease of creating threads compared to child processes.