BILKENT UNIVERSITY

CS342 - Operating Systems

Project 1 - Report

Multi-Process Application using IPC

İrem Seven

21704269

In this project, I have created two separate programs. The first application called pmc, creates multiple child processes and uses POSIX queues to communicate. The second application called tmc, does the same task as pmc but instead of creating child processes and using POSIX queues, it uses POSIX threads. The task was to taking a maximum of 5 files which includes words and generating a file that includes the words in sorted order with their occurrence numbers. I have tested both programs for different input file numbers. I have provided the same files for both of them. Below it is shown a plot for a comparison between two programs according to their execution times. Also, the tested cases are explained below the graph.

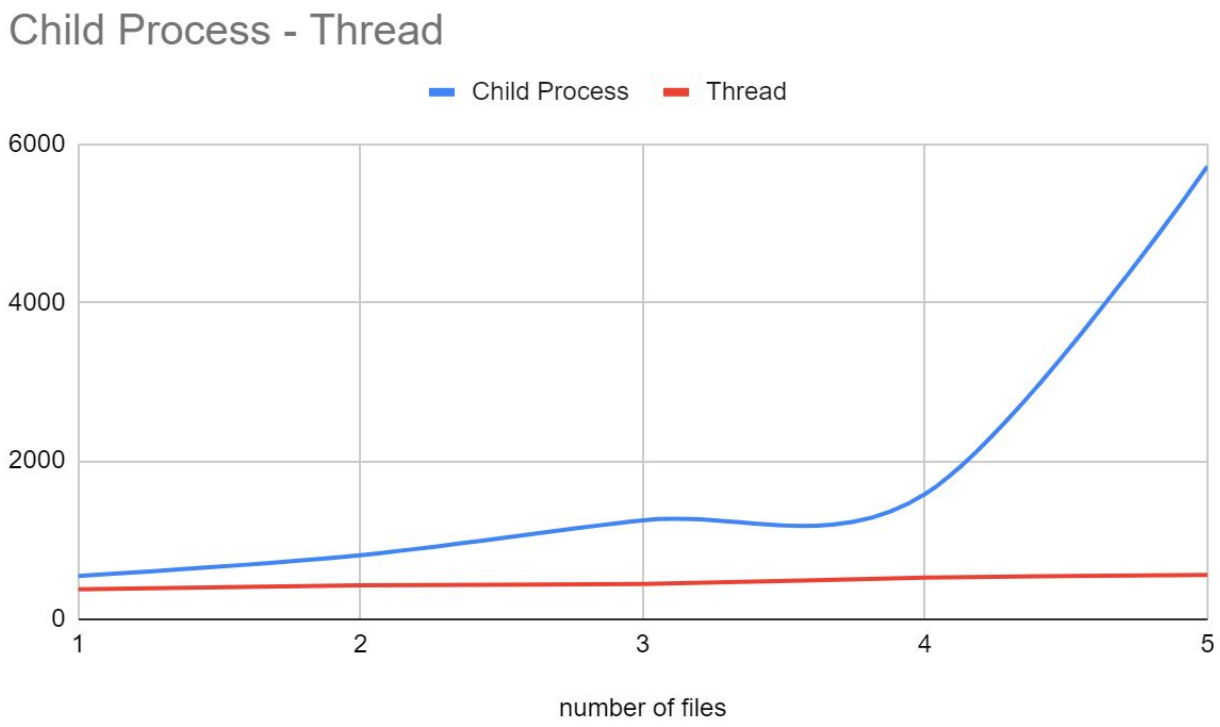

Figure 1: Plot for Comparison

I have tested for 5 different file numbers. I used the same files for both programs. I used different files for each file number. For example, I used 3 different files named words1, words2, and words3 for the number of 3 files. Each file was a different size. As can be seen from the plot, using threads was much efficient than using child processes with queues. The

difference becomes much significant as the file size increases. One of the reasons for this is the constant communication with the kernel when using child and queue. To receive and send messages we always perform system calls within the API. This situation causes the program to run slower. On the other hand, one of the reasons is my implementation preference. In pmc application, I wanted to stick with the default message queue size. That way the program can execute without any file size limit. This preference led me to deal with some delay problems. Since to implement with the default message size, which is 10, the parent process should not wait for all the children to complete the program. Thus, in my implementation, the parent process receives messages from the corresponding queues as its children have sent. However, this situation caused losing some messages due to time delays. To solve this problem I have made the parent process sleep for 0.02 ms if there are no messages in the corresponding queue. If after that time still there are no messages then the parent discards the queue knowing it's empty. This delay is not recognized when the execution happening by the user. However, it may be one of the reasons for the huge gap between them as the file number increases. Moreover, the execution time of the tmc application did not show significant differences as the file size increased. It is an indication of how efficient it is. However, using threads is less safe since the shared data is open to be changed. Below it is shown the data table for the provided plot which includes exact execution time values.

number of files	Child Process	Thread
1	548	377
2	810	428
3	1250	445
4	1581	525
5	5719	559

Figure 2: Table for Comparison

Below it is shown an example output file for sorting words in ascending order. Both programs provided the same output for the same files.

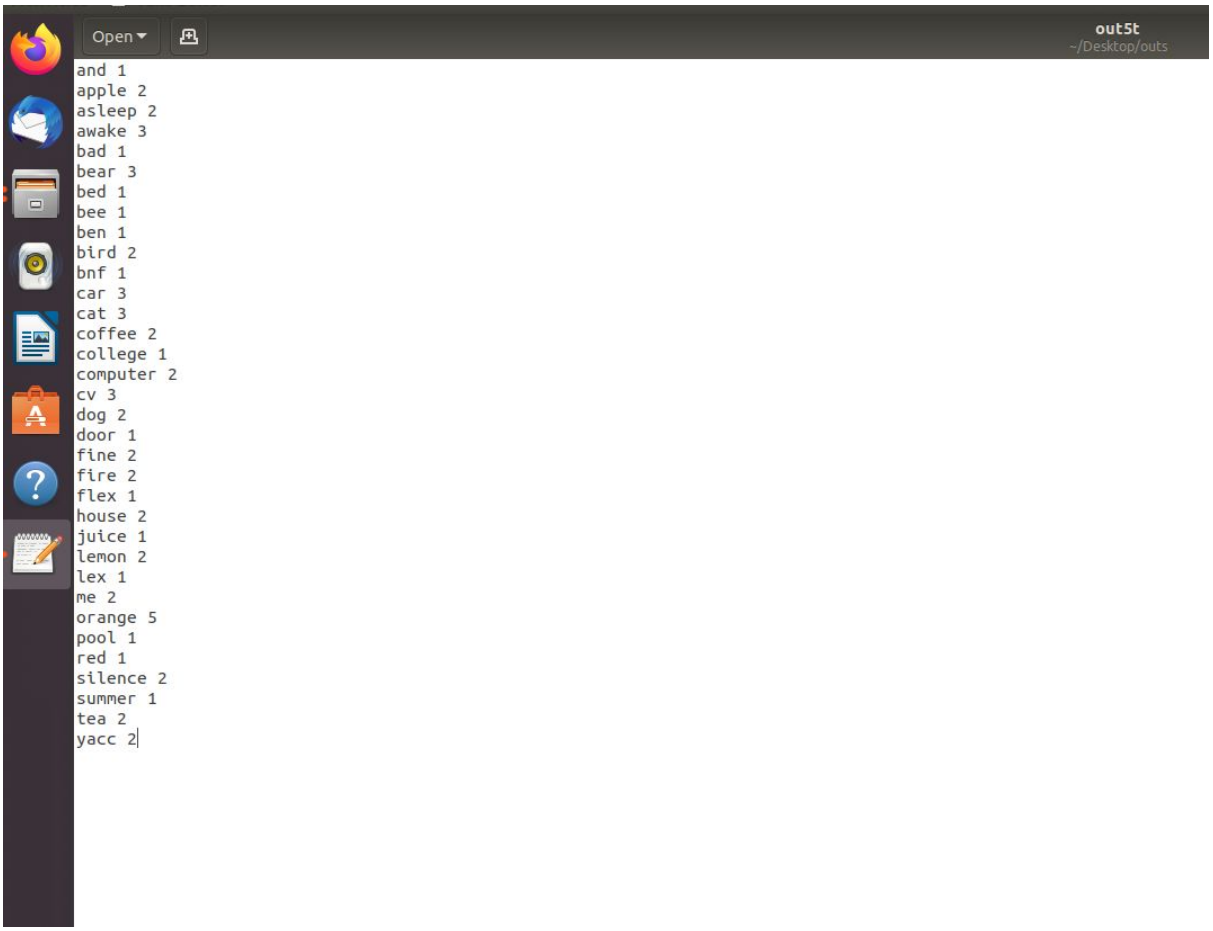

Figure 3: Example Output File

Test Environment:

Intel Core i7 7700HQ (Kaby Lake)

Linux on Virtual Box