CS342
Operating Systems
Spring 2022

Project #1
Processes, IPC, and Thread

Group Members:

Giray Akyol, 21902298, Section 1

Muhammed Can Küçükaslan, 21901779, Section 1

Procedure

First we have created sample input files with 10000, 50000, 100000, 250000, 500000, 750000, 1000000, 3000000, 5000000, and 10000000 lines (integers). Then we have executed the server with 10 files of sames size (i.e. 10 files each having 10000 lines, files each having 50000 lines, and so on).,

Time measurement is starts before the creation of child processes or threads and ends with the termination of all the child processes or threads –in corresponding programs. Thus, the measured time does not include the time between the start of server program and the client which would've seriously affected the measurements.

The client's request is fixed to 1000 consecutive intervals of size 1000, where first interval starts from 0. Thus the intervals are [0, 1000), [1000, 2000) … [999000,1000000).

We made measurements for four times. 2 times using the fork (child process) and two times using the threads.

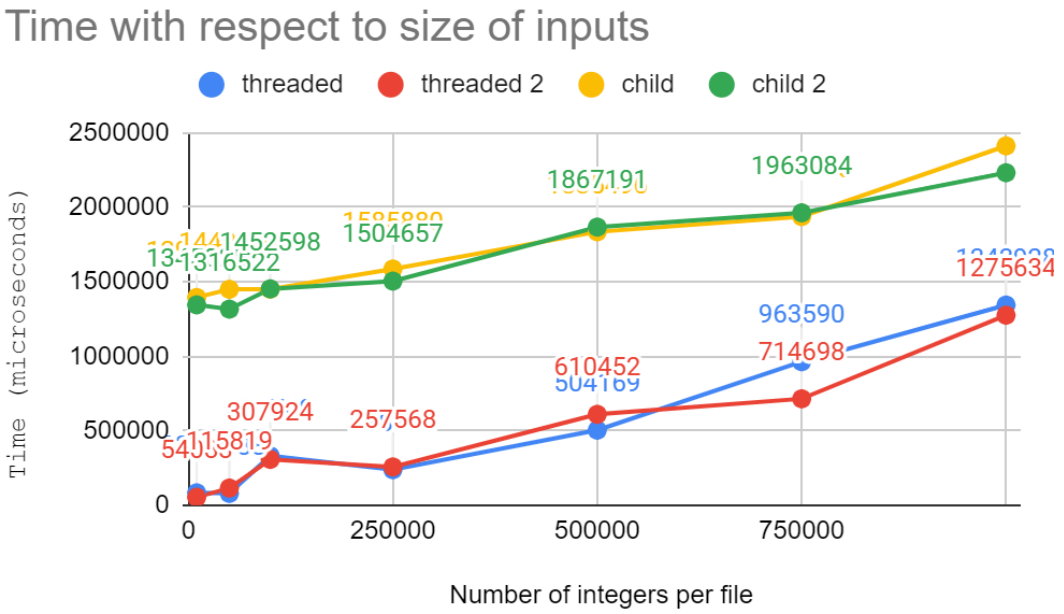
The program is tested using a computer with 8-core AMD Ryzen 5 3500U processor and 7 GB RAM. The compiler used is "gcc (`Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0`".

Data, Result and Conclusion

Threads and processes read file and calculate the intervals at a similar efficiency and speed because they do almost the same operations. However the big difference between threads and processes is the speed of information exchange between main program (of server) and child processes or threads which it creates and communicates with. Our hypothesis is that because of the message queue bottleneck of the process they will be slower, especially because our implementation for threads does not use locks. As an example let's say both the child process and thread calculated the interval array in 10ms while the work of thread is done here, the process will need to send the results to the server while other child processes are also trying to do the same thing, this will be particularly evident with small file sizes with high interval counts. According to our hypothesis, threads will run faster than processes but as the file size increases the effect will diminish because the amount of time spent to process file increases while the time spent communication between main program (of server) and child processes or threads stays same (since in both cases same amount of messages sent). Lastly threads can also schedule better since the kernel does not have to copy & restore process state.

First of all, in the first graph we clearly see that threads run faster than processes. Also we see that with a small number of integers per file (10k & 100k)

threads are about 11 times faster but with a higher number of integers per file. It is clear from the first graph that the program execution is much more fast in the threaded version for the input files of 10000-1000000 lines.



In the second graph, however, it isn't clear if any of the preferences has a significant advantage over the other. Because In this case the input files have the size of 1000000-10000000 and the amount of time spent on the file processing is much more than the time spent to the message queue.
Both of these results support our hypothesis.