COMP8005: Assignment #1

Thread vs Processes Load Test

Kevin Lo, A00952922
1-23-2021

Table of Contents

Introduction

In this assignment, I investigate the performance and efficiency of each implementation of using either processes or threads to compute the prime factorization of many numbers.

I created my two programs in Python to perform the prime factorization using threads or processes because the syntax for using either threads or processes are very similar.

The code used to perform prime factorization as the load was found on stackoverflow because the prime factorization function was provided in C.

The task that each thread or process will do is, retrieve the index number, perform prime factorization on the number and output the result to console and to a log file.

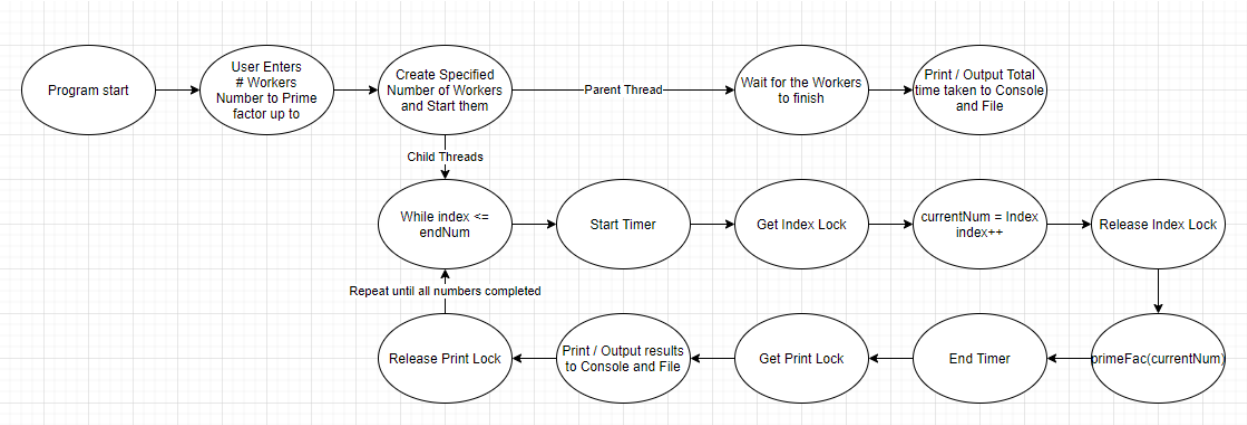
There are locks when accessing the index number and console/log output to solve race conditions that will occur when there are many threads/processes trying to access the same resources.

Threads Pseudocode

```
MAIN THREAD
get number of threads to use (eg. 4) workers
get number from user (eg. 10) endnum
index = [2]
threads = []
measure start time
for n in range(workers)
        t = thread(target=workerfunc, args=(index,endnum))
        t.start()
        threads.append(t)
for t in threads:
        t.join()
measure end time
print total time taken to console
write to log file

WORKER THREAD
workerfunc(index, endnum)
        while(index <= endnum)
                measure start time
                lockindex{
                        number = index
                        index++
                }
                primefac(number)
                measure end time
                lockprint{
                        print prime factorization/timetaken to console
                        write to log file
                }
```

Threads Finite State Machine

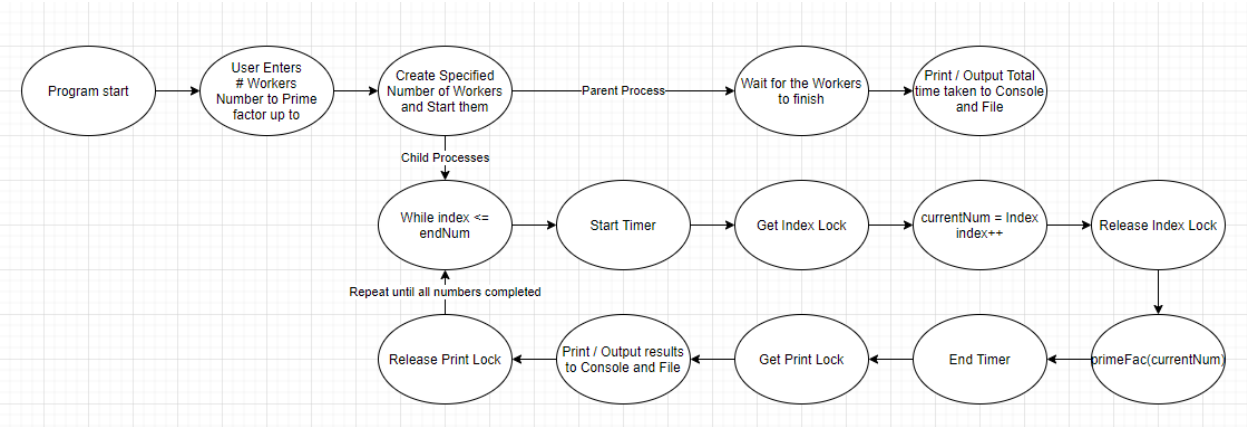


Processes Pseudocode

```
MAIN PROCESS
get number of processes to use (eg. 4) workers
get number from user (eg. 10) endnum
index = [2]
processes = []
measure start time
for n in range(workers)
        p = process(target=workerfunc, args=(index,endnum,lockindex,lockprint))
        p.start()
        proccesses.append(t)
for p in procecsses:
        p.join()
measure end time
print total time taken to console
write to log file

WORKER PROCESS
workerfunc(index, endnum)
        while(index <= endnum)
                measure start time
                lockindex{
                        number = index
                        index++
                }
                primefac(number)
                measure end time
                lockprint{
                        print prime factorization/timetaken to console
                        write to log file
                }
```

Processes Finite State Machine



Results

Testing was done on a computer running Fedora operating system running 8 workers on prime factorization up to 100000.

The results that I have found was that processes were much faster than threads by almost twice the speed. Prime factorization using processes finished the task in about 70.9 seconds while the threads took 120 seconds to complete. Both programs were using 8 workers and did the prime factorization up to the number 100000.

Looking at the info provided by htop, it seems like processes are more suited to using more CPU utilization to complete the task. This is because processes were using 95-100% of my CPU as opposed to 40-50% using threads.

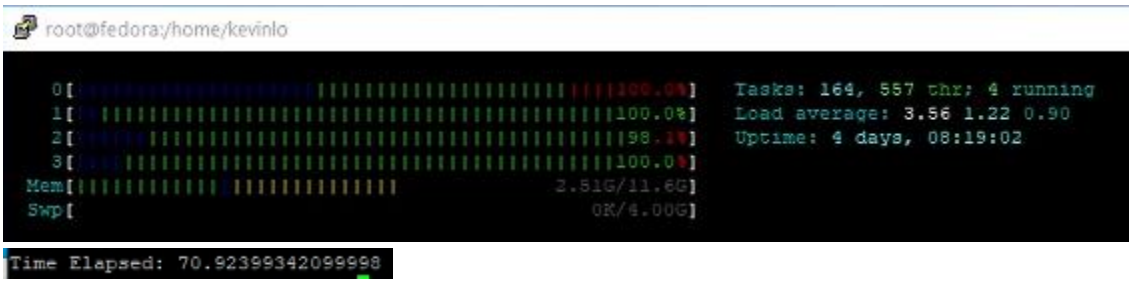


Figure 1 CPU Usage of Processes & Time Taken

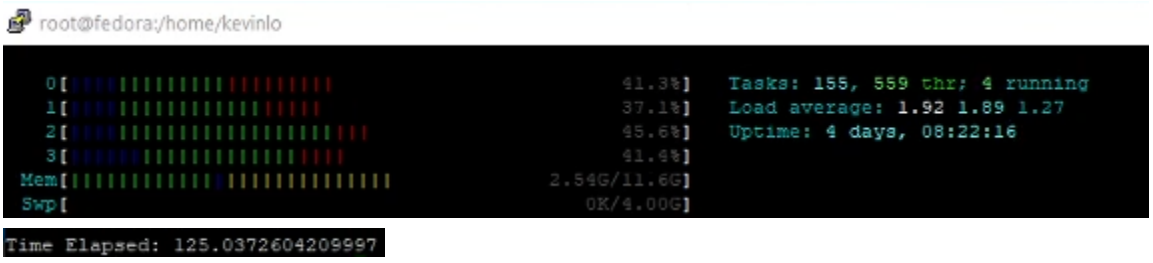


Figure 2 CPU Usage of Threads & Time Taken

Based on info I have read about differences between processes and threads, processes are more heavyweight and are able to use more system resources to complete their task, threads are more lightweight. Processes were able to fully take advantage of the CPU's power to calculate the prime factorization much faster compared to threads.

From information I found online, threads do not all run at the same time because they are given time slices to run which means threads do not actually run simultaneously. Processes can run simultaneously which is one of the other reasons why processes finished faster than threads.

Processes are great for number crunching tasks such as prime factorization which is why it can outshine threads in this task because they can fully utilize the CPU and run simultaneously unlike threads.