# Assignment #1

**Clemens Lo**
A00863045
COMP 8005 January 18, 2016

# TABLE OF CONTENTS

# Overview

This project served to answer the question of which is better, multiprocessing multithreading by measuring the performance and efficiency of each mechanism.

# Design Work

## Mathematical & I/O Tasks

In both the process and threaded versions of the application, they will be calculating

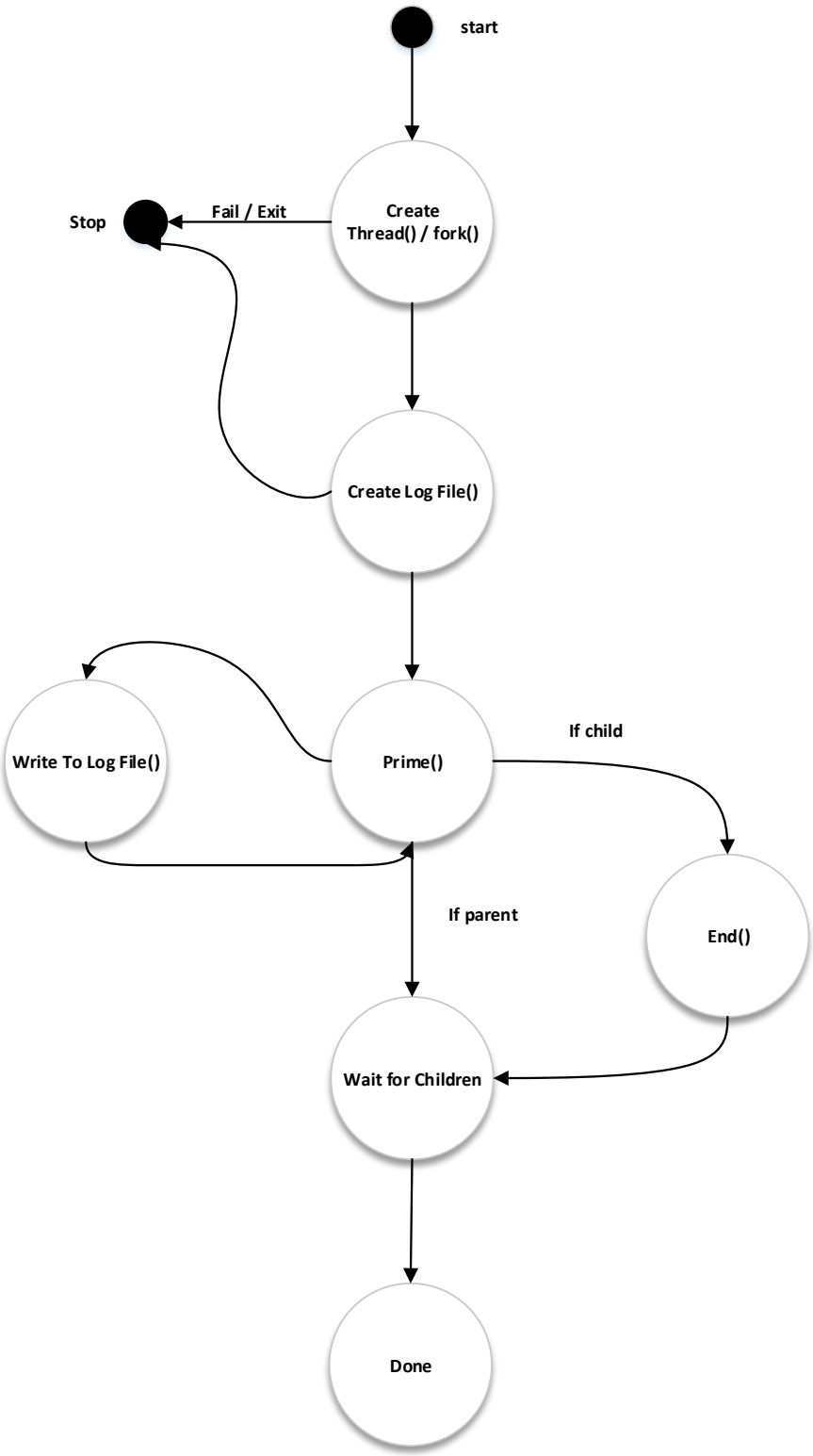
and producing lists of prime numbers.

In both case, processes/threads will work on two tasks; a mathematical computation, finding all the prime number over a chosen and I/O activity, writing all the prime numbers to a newly created .txt file.

Both programs allow user to specify the range of prime computation and the number of threads/processes the program will create to complete the tasks. Each thread/process will work on the exact same tasks.

The reason I chose calculating prime numbers was because this task could easily be subdivided into chunks for each process/thread to handle. Each thread/process will be assigned to the exact same amount of work and have same amount of resources.

# Design Diagrams



# Pseudocode

## Process

```
Main(number of processes/threads, range of prime number computations ) {
        open log file
        strat timer
        for loop( i to number of processes) {
                if fork() is children
                break
        run the Prime(PrimeRange, processNum)
        if its a parent
        while no children left
                timer end
                calculate the elapsed time
                write elapsed time to log file
        return
}
```

## Thread

```
Main(number of processes/threads, range of prime number computations ) {
        open log file
        strat timer
        for loop( i to number of processes) {
                create thread with Prime()
        }

        for loop( j to number of processes) {
                joing thread
```

```
        }
        timer end
        calculate the elapsed time
        write elapsed time to log file
        return
}
```

## Mathematical Computation

```
Prime(PrimeRange, processNum) {
        Open process log file
        (loop from 0 - number of iterations){
        If it is prime number
              Write the prime number and current time to log file

        Print finish message
}
```

## Testing

Show below are example of how to execute both multiprocessing and multithreading programs with 5 workers and 100000 computation range.

The first argument is the number of processes/threads to be created and second argument is the number of range of computation to calculated.

```
[root@localhost Threads]# ./Thread 5 100000
FINISHED THREAD NUMBER: 1
FINISHED THREAD NUMBER: 3
FINISHED THREAD NUMBER: 0
FINISHED THREAD NUMBER: 4
FINISHED THREAD NUMBER: 2
3.135770
```

```
[root@localhost Processes]# ./Processes 5 100000
FINISHED PROCESS NUMBER: 2
FINISHED PROCESS NUMBER: 1
FINISHED PROCESS NUMBER: 4
FINISHED PROCESS NUMBER: 3
FINISHED PROCESS NUMBER: 5
3.154168
```

Testing was done using various different process/thread counts as well as varying computation range for each process/thread.

The testing sample were taken from the result for each combination base on 5, 25, 50 processes/threads and computation range of 100, 100000, 200000.

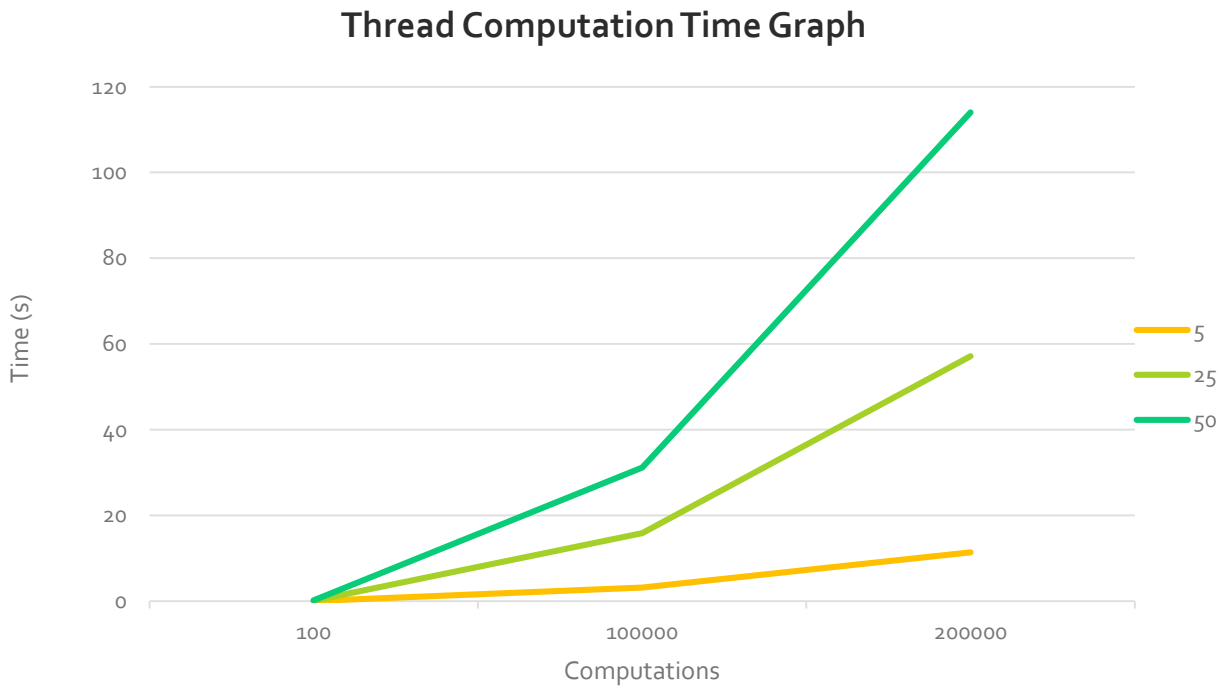
## Completion Times for Multithreading

**Table 1-1** Completion Times for Threads

| Threads | 5 Threads | 25 Threads | 50 Threads |
|---|---|---|---|
| 100 computations | 0.003618s | 0.091165s | 0.134096s |
| 100000 computations | 3.13577s | 15.826725s | 31.13211s |
| 200000 computations | 11.358658s | 57.169679s | 114.04119s |

The table above tabulates the completion times of 100 and 100000 mathematical computations (prime numbers) within 5, 25 and 50 threads.

**Graph 1-1** Computation times across 5, 25 and 50 processes



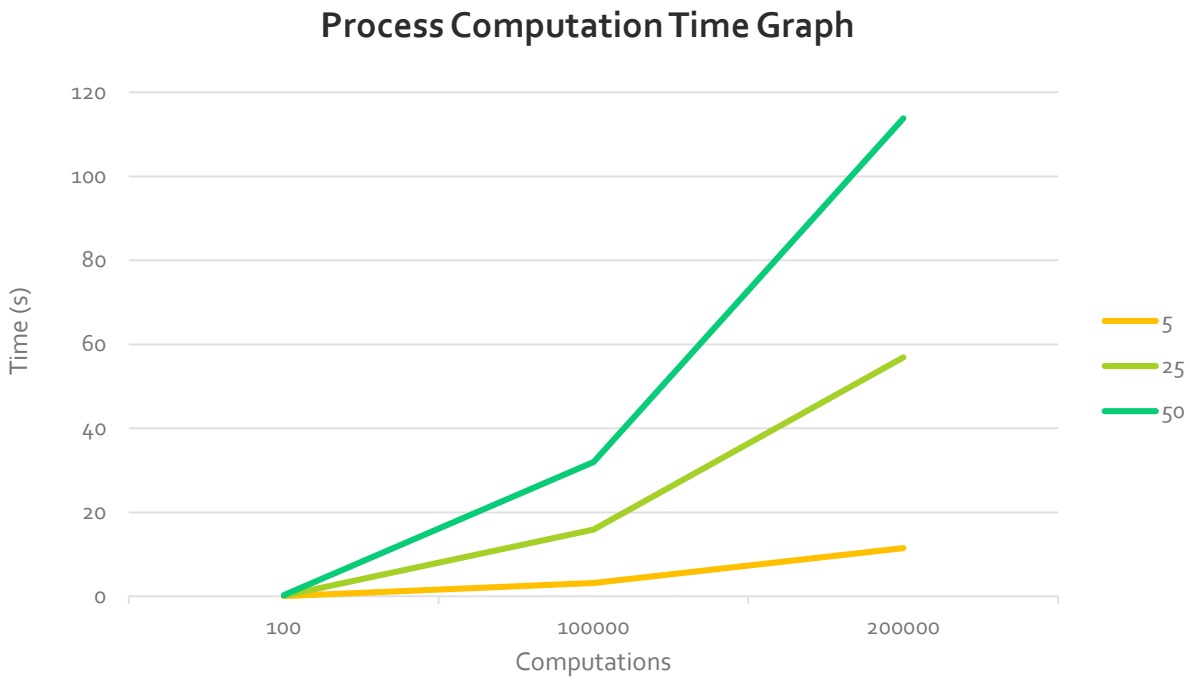
### Completion Times for Multiprocessing

**Table 1-2** Completion Times for Processes

| Processes | 5 processes | 25 processes | 50 processes |
|---|---|---|---|
| 100 computations | 0.005824s | 0.156798s | 0.271226s |
| 100000 computations | 3.154168s | 15.883922s | 31.965651s |
| 200000 computations | 11.498987s | 56.845766s | 113.793601s |

The table above tabulates the completion times of 100 and 100000 mathematical computations (prime numbers) within 5, 25 and 50 threads.

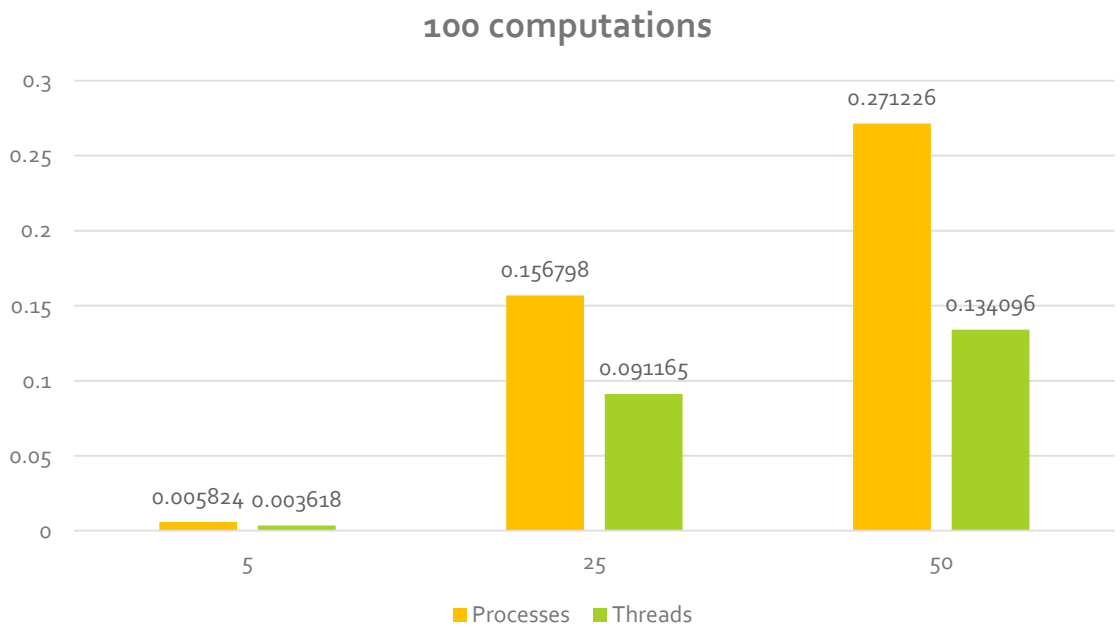
**Graph 1-2** Computation times across 5, 25 and 50 threads




## Observations

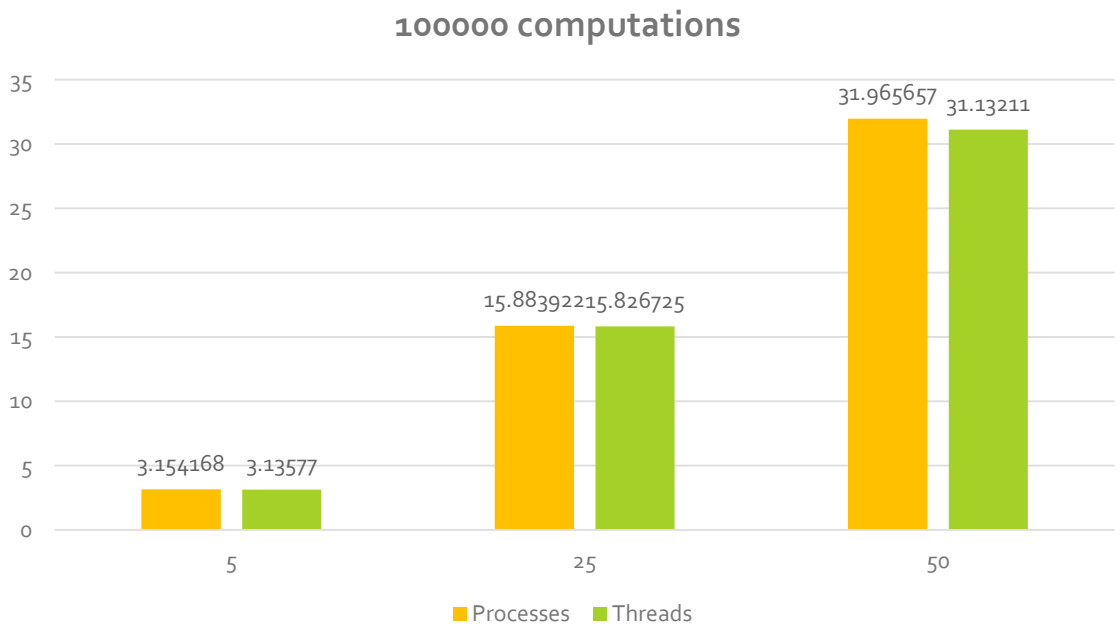
Though analyzing the data, I found that there is a very marginal difference between the performance of threads and processes. Threads are noticeably better in speed overall.

## Graph 2-1

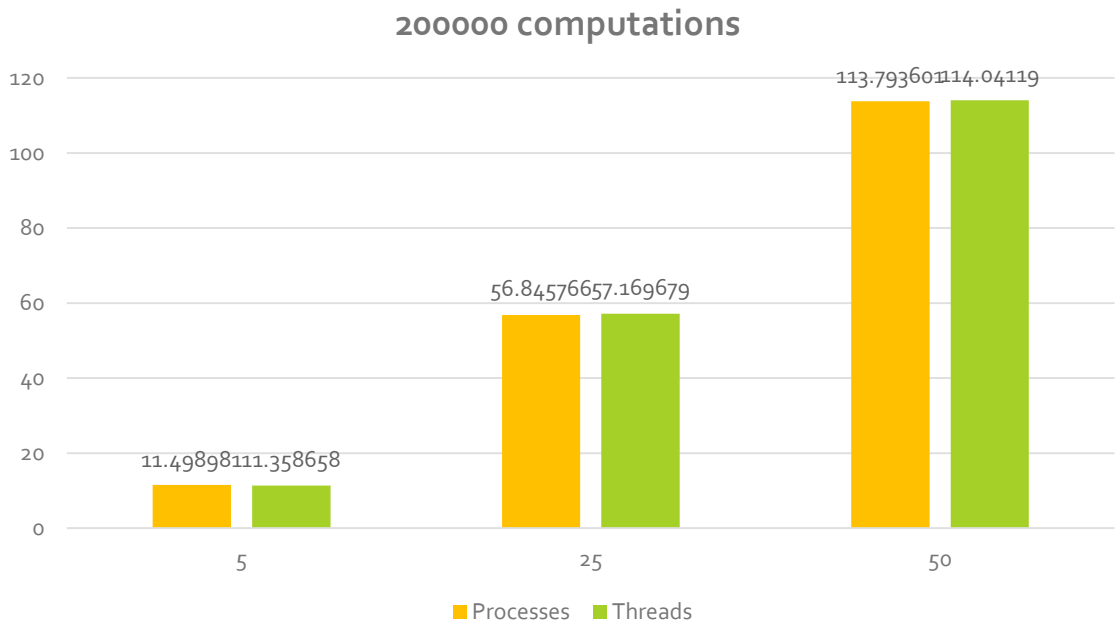



In 100 computations, there are noticeable differences in performance. The threads have shorter completion times than processes.

## Graph 2-2




## Graph 2-3




Along with the larger size of calculation, the completion times between processes and threads are near identical.

## Conclusion

Base on the data obtain from this experiment, multithreading takes advantage over multiprocessing when it comes to creation time. When the computation is small, threads have better performance than processes.

In conclusion, I would say that the performance of either multithreading or multiprocessing should not have significant effects on computing time. However, if the application will invoke continuously create new processes/process and work on small tasks, I would recommend using multithreading programing.