COMP 8005

Assignment 1 Design Document

By: Derek Wong

Instructor: Aman Abdulla

Due: 12 PM on January 25th, 2020

Table of Contents

Overview

Use multiple processes and threads to perform a system-intensive operation and then measure the performance and efficiency of each implementation.

Requirements

The application has the following constraints:

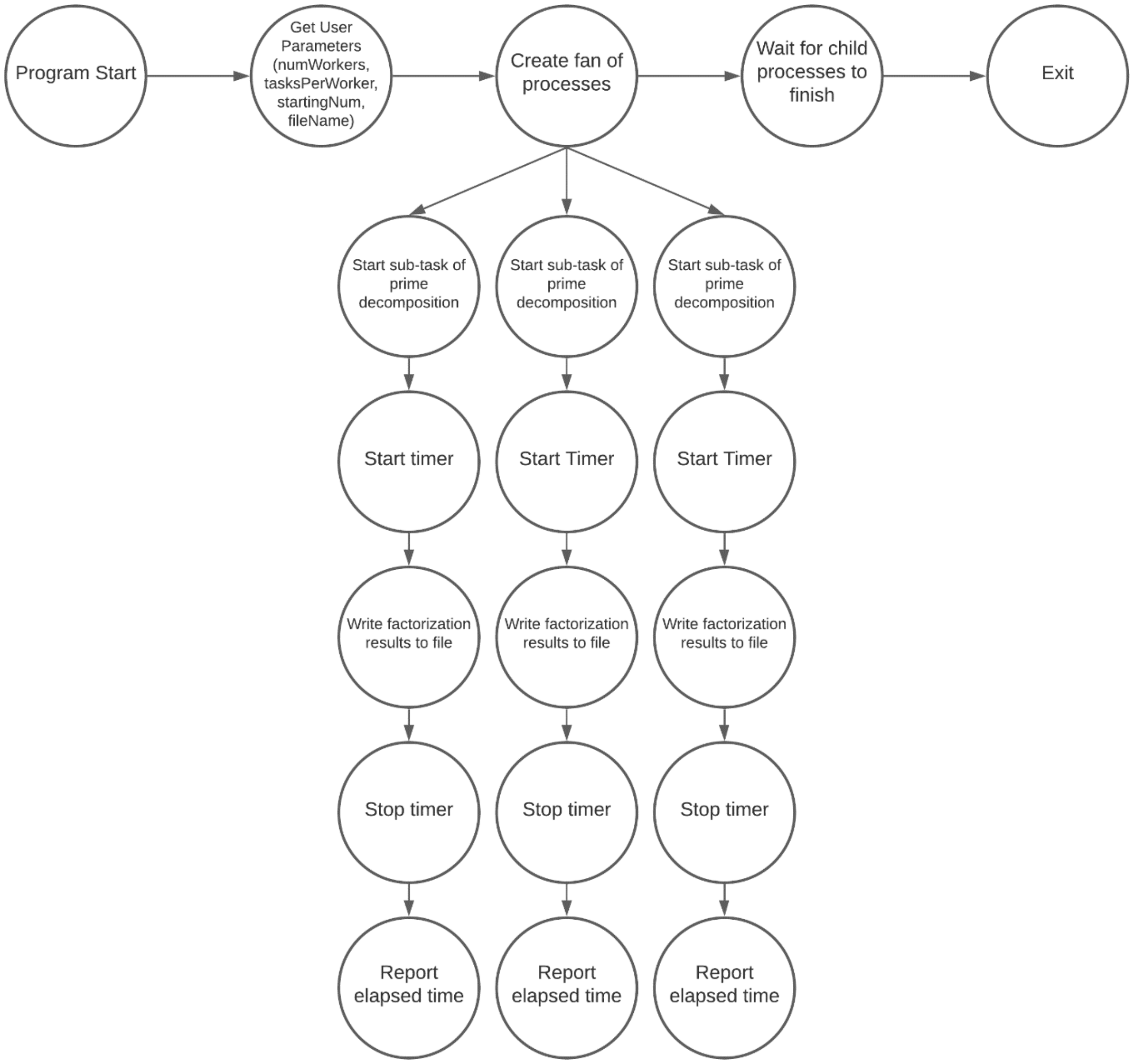
1. Any language of choice to implement the applications.
2. Each implementation must have same number of "workers", a minimum of five processes or threads.
3. Can use any other IPC constructs as necessary.

Design

- The process and thread applications will be written in C
- Prime number factorization computations written to file to perform system-intensive operations for both applications using the same algorithm and accepting the same command line arguments (number of workers, number of tasks per worker, starting number for prime factorization, filename to write to). Each will report their elapsed time taken to perform the prime number decompositions.
- Process application will fork multiple child processes.
- Thread application will use one process to spawn multiple worker threads.
- Each child/worker will have a range of prime number factorizations

Program Design

Process



Program Start
Get User Parameters (numWorkers, tasksPerWorker, startingNum, fileName)
Create fan of processes
Wait for child processes to finish
Exit
Start sub-task of prime decomposition
Start sub-task of prime decomposition
Start sub-task of prime decomposition
Start timer
Start Timer
Start Timer
Write factorization results to file
Write factorization results to file
Write factorization results to file
Stop timer
Stop timer
Stop timer
Report elapsed time
Report elapsed time
Report elapsed time

Figure 1 – process state diagram

Thread

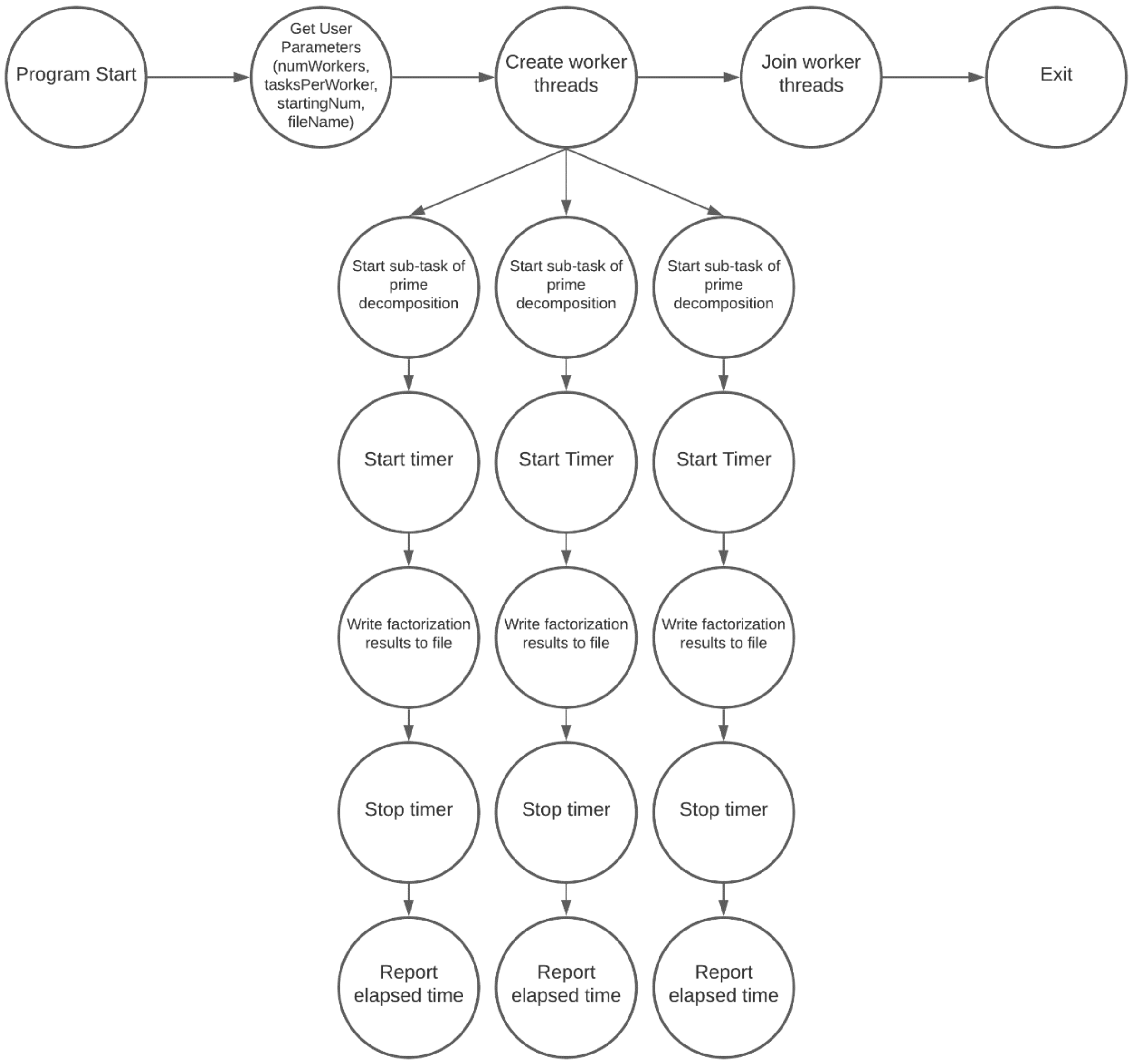

Program Start
Get User Parameters (numWorkers, tasksPerWorker, startingNum, fileName)
Create worker threads
Join worker threads
Exit
Start sub-task of prime decomposition
Start sub-task of prime decomposition
Start sub-task of prime decomposition
Start timer
Start Timer
Start Timer
Write factorization results to file
Write factorization results to file
Write factorization results to file
Stop timer
Stop timer
Stop timer
Report elapsed time
Report elapsed time
Report elapsed time

Figure 2 – thread state diagram

Pseudo Code

Process

```
Get user parameters from command line:
    Set number of processes, tasks per worker, starting number, file name

Initialize full range of prime numbers list

Create a range of numbers for each process to decompose:
    Assign the start of each range a unique index

Create fan of processes:
    For num in number of processes:
        Fork

If child process:
    For number in task range:
        Decompose(number)
        Start timer
        Write to file
        Stop timer
        Report elapsed time

If not child process:
    While there are still child processes running:
        Wait for child process to exit

Free prime number list memory
```

Thread

```
Declare ThreadInfo struct to be used in each thread task

Get user parameters from command line:
    Set number of threads, tasks per worker, starting number, file name

Initialize full range of prime numbers list

Create a range of numbers for each thread to decompose

Initialize threads list, thread info list, destination list for workers to store results

Assign tasks to workers:
    For num in number of threads:
        pthread_create(threads list[num], NULL, task, thread info list[num])

Join threads to main process

Free memory for threads list
Free memory for thread info list
Free memory for destination list

task:
    Read thread info data
    For number in task range:
        Decompose(number)
        Start timer
        Write to file
        Stop timer
        Report elapsed time
```