CS3510: Operating Systems - I

Coding Assignment 2 - Report

Soumi Chakraborty

ES19BTECH11017

1 Overview

This is a multithreaded program that calculates various statistical values for a list of numbers. A series of numbers is passed to the program and it then creates three separate threads. One thread will determine the average of the numbers, the second will determine the minimum value and the third will determine the maximum value.

For analytical purposes, the submission also includes another source code file to calculate the time taken by the code while executing.

2 Threads and Multithreading

A thread is a path of execution within a process. Threads belonging to the same process share memory space. An operating system that supports multithreading allows multiple threads to execute concurrently.

3 Implementation

3.1 Headers

The program uses the following header files:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
```

- **stdio.h**: contains the definitions of *printf()*, *fprintf()*, *fscanf()*, etc.
- **stdlib.h**: contains the definition of *rand()*
- **pthread.h**: contains the definitions of all the thread related functions and datatypes, i.e., *pthread_t, pthread_attr_t, pthread_attr_init(), pthread_create(), pthread_join()*
- **time.h**: contains the definitions of the functions and constants used to calculate the time taken by the multithreaded program.
  NOTE: this header file has only been included in the analysis source code file.

3.2 Structures and Global Variables

The program has the following global variables:

```
double average;
int minimum, maximum;
```

These variables are global so that all the threads (including the parent) has access to them throughout the scope of the program.
The program used for calculating the time taken by the multithreading process has an additional global array:

```
int numbers[6000010];
```

This array is required to be global primarily due to the large input files; main() is not capable of storing arrays with more than around 2 million integers.

The program includes the following struct in order to pass more than one parameter (namely the size and a pointer to the number array) to the functions in *pthread_create()*.

```c
struct input
{
    int size;
    int *array;
};
typedef struct input Input;
```

3.3 Functions

The functions *Avg()*, *Min()*, and *Max()*, are the functions that each separate thread works on, and are passed to *pthread_create()*.

main()

In the main source code file, the main function contains the parent thread. All the file handling operations are also dealt with in this function.

The following lines declare the thread identifiers and their attributes:

```c
pthread_t thread1, thread2, thread3;
pthread_attr_t attr1, attr2, attr3;
```

And the following lines create new threads, and then join them once they are done executing:

```c
pthread_create(&thread1, &attr1, Avg, (void *)&input);
pthread_join(thread1, NULL);

pthread_create(&thread2, &attr2, Min, (void *)&input);
pthread_join(thread2, NULL);

pthread_create(&thread3, &attr3, Max, (void *)&input);
pthread_join(thread3, NULL);
```

Since the three functions passed to pthread_create() store their final results in global variables, we make use of these variables to store the output in a separate text file.

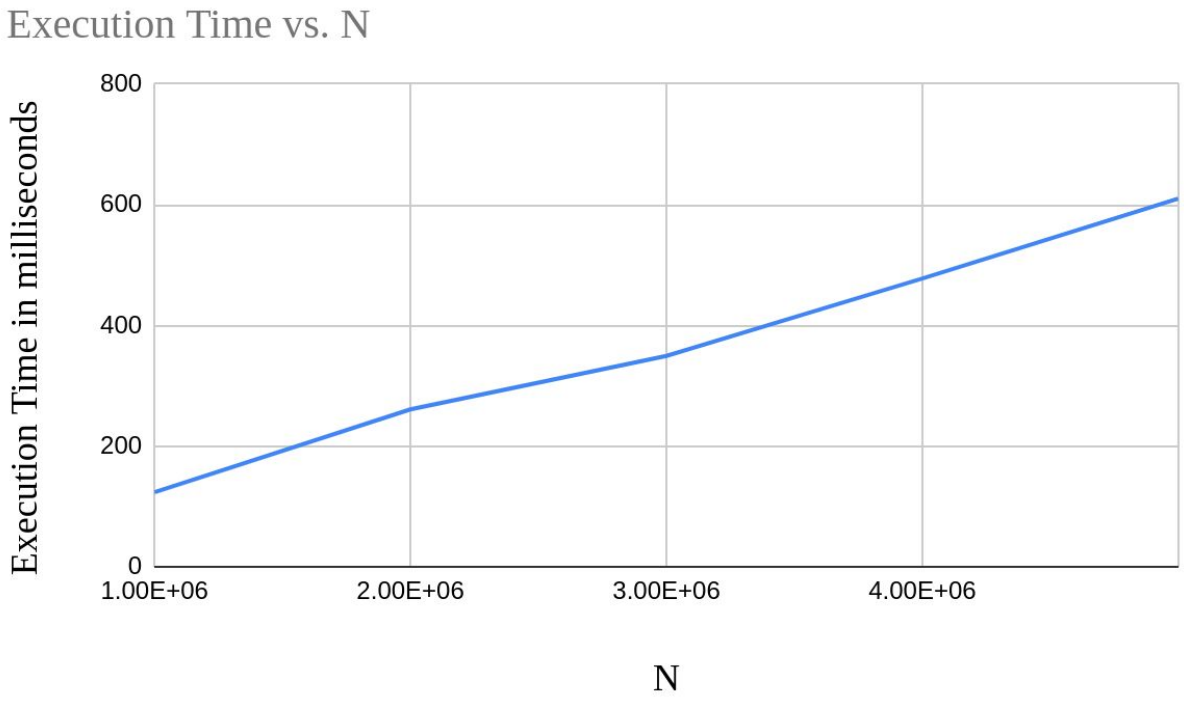
4 Analysis

To check the average running time of the multithreading program, my submission includes code which does so. The analysis file generates random numbers between 0 to N (N being the number of elements in the array) and stores them in the input file ***inp.txt***. The function which handles the multithreading is then called and executed using the contents in the input file. The elapsed time is calculated by calling the clock() function before and after the threads() function is called. Since checking the execution time of just one iteration of threads() might result in inaccurate results, the program executes threads() using the same input file for 50 iterations and finally calculates the mean execution time.

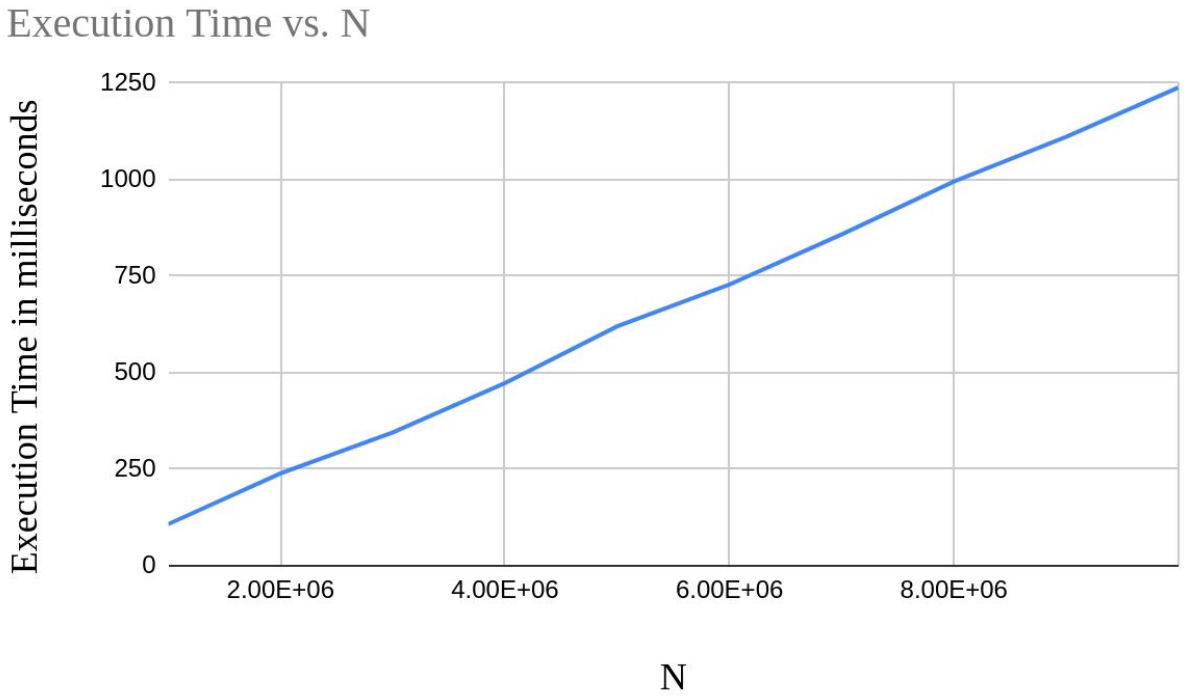
As specified in the problem statement, N ranges from 1 million to 5 million, unsurprisingly, the execution time of the program increases as the size of N increases. The following execution times are in milliseconds.

```
Elapsed time for case 1: 123.411000
Elapsed time for case 2: 260.882000
Elapsed time for case 3: 349.481000
Elapsed time for case 4: 477.789000
Elapsed time for case 5: 610.405000
```

Case 1 corresponds to 1 million numbers, and so on and so forth.



Here's another graph where N ranges from 1 million to 10 million.



Clearly, N and the execution times seem to be related to each other linearly.