Assignment 4: Multi-threading

Name: Fatima Tariq
ID: ft07200

Course: CS232 Operating Systems
Section: L1
Instructor: Muhammad Mobeen Movania

1 Introduction

This assignment involves creating a C program to compute the sum, minimum, and maximum values from a specified file. Initially, we'll perform these calculations using a single thread program and then a multi threaded one. Performance of both will be compared.

2 Single-threaded Approach

The program reads the dataset from a file, calculates the sum, minimum, and maximum values using a single thread, and records the processing time.

2.1 Implementation Details

The implementation is given in the figure below.

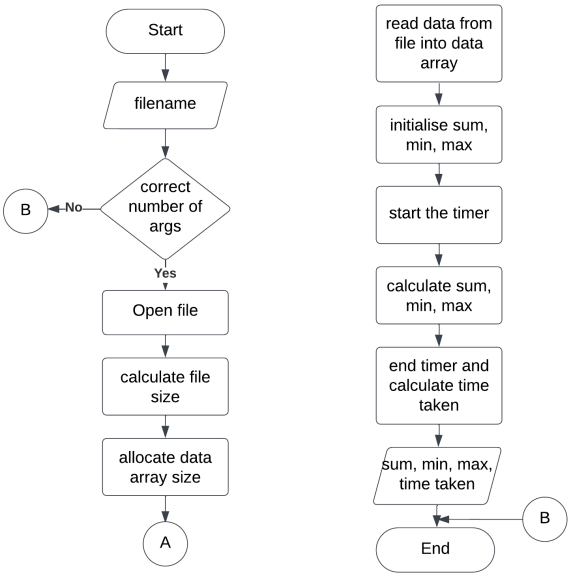

Figure 1: Single-threaded main function

3 Multi-threaded Approach

Multiple worker threads concurrently process the dataset using POSIX Threads API. Each thread handles a designated portion of the dataset, computes a partial sum, and collectively calculates the final sum. Thread-safety has been ensured by using mutex locks. The processing time for multi-threaded execution is measured.

3.1 Implementation Details

The implementation of the main function is given below. The calculate function processes a chunk of data specified by the *Thread_args* structure. The function calculates the partial sum, minimum, and maximum values within its assigned data range. To ensure thread safety, it uses a mutex (mutex) to lock critical sections where shared variables are updated.

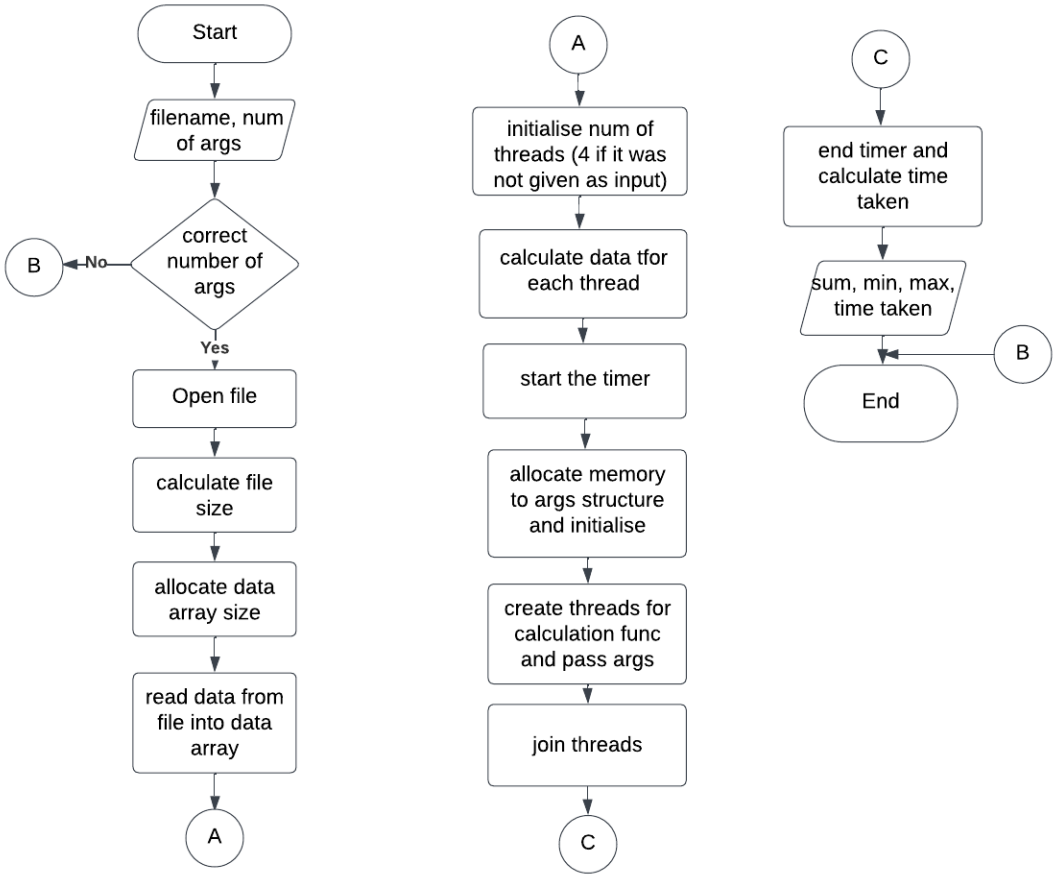

Figure 2: Multi-threaded main function

4 Timing and Performance

Both single-threaded and multi-threaded executions were timed for each data file to evaluate the impact of using multiple threads.

4.1 Timing Methodology

The time was calculated using clock() and clock_gettime() functions. The program was run 4 times for each data file and then average time was calculated as shown in table 1 and 2. This was done to ensure there are no biases due to overheads.

4.2 Timing Results

4.2.1 Single-threaded Program

File	Average Time (seconds)
data_tiny.txt	0.000019
data_small.txt	0.001980
data_medium.txt	0.019880
data_large.txt	0.422750

Table 1: Time taken for single-threaded program

4.2.2 Multi-threaded Program

File	Average Time (seconds)			
	2 Threads	4 Threads	16 Threads	32 Threads
data_tiny.txt	0.000144	0.000135	0.000841	0.000933
data_small.txt	0.001403	0.000893	0.000714	0.000811
data_medium.txt	0.013256	0.007947	0.005615	0.005596
data_large.txt	0.216613	0.092942	0.064740	0.061638

Table 2: Time taken for multi-threaded program

4.3 Performance Analysis

4.3.1 Single-threaded Program (Table 1)

- **data tiny.txt:** Very fast execution, likely due to the small size of the dataset.
- **data small.txt:** Reasonable execution time, expected for a moderate-sized dataset.
- **data medium.txt:** Noticeably longer execution time, as expected for a larger dataset.
- **data large.txt:** Significant increase in execution time, reflecting the challenge of processing a very large dataset.

4.3.2 Multi-threaded Program (Table 2)

- **data tiny.txt:** The overhead of thread creation and synchronization makes the multi-threaded approach slower for small datasets.
- **data small.txt:** Slight improvements with more threads, but not as significant due to the manageable size of the dataset.
- **data medium.txt:** Substantial speedup with additional threads, demonstrating the effectiveness of parallel processing for larger datasets.

- **data large.txt:** Significant speedup with more threads, indicating that multi-threading is beneficial for very large datasets. However, diminishing returns are observed beyond a certain number of threads, possibly due to hardware limitations.

For further comparison, a graph of performnace of different number of threads is given below for the medium and large dataset.

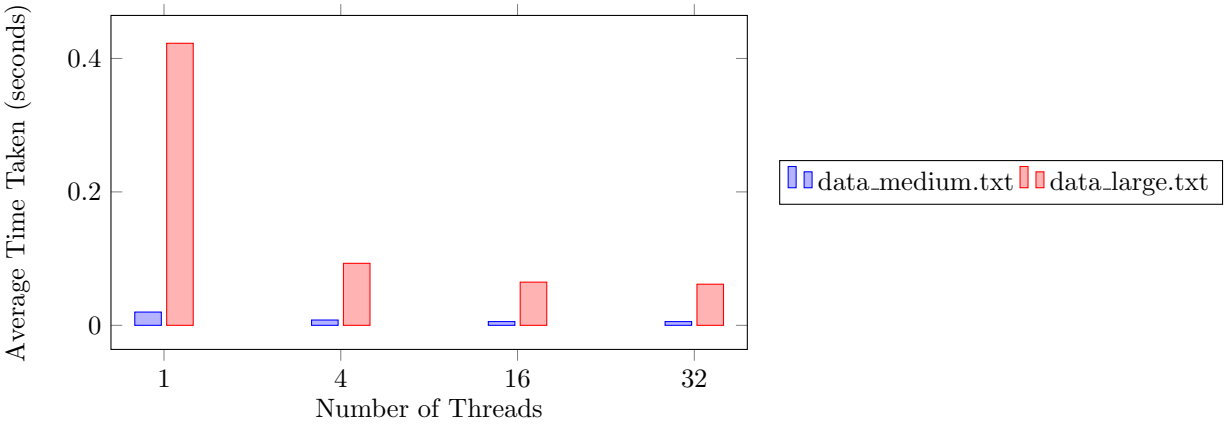

Figure 3: Time taken for different number of threads

4.3.3 Conclusion

Overall, the multi-threaded program shows substantial benefits for larger datasets, but the choice of the number of threads needs to be chosen according to the size of the dataset.