CS/CE 232/324 Operating Systems
Assignment No.4

Owais Waheed (ow07611)

December 3, 2023

1 Implementation of Single-threaded Program

1.1 Data Structures

The primary data structure used in the single-threaded program is an integer array named `data`. This array is dynamically allocated to store the dataset. The program initializes the array with a predetermined size and dynamically adjusts the size as the dataset is being read.

1.2 Algorithm Overview

The single-threaded algorithm follows a straightforward process:

1. **Initialization:** The program initializes variables, including the data array, sum, minimum (`min`), and maximum (`max`) values.
2. **File Reading**: The content of data file is read and then stored in the dynamically stored array ensuring proper handling of file operation. Also memory is managed efficiently as the algorithm dynamically adjusts the size of the `data` array as needed to accommodate the incoming dataset.
3. **Data Processing Loop:** The main processing loop iterates through the dataset, updating the running sum and identifying the minimum and maximum values encountered.
4. **Clock Measurement:** The clock is employed to measure the execution time of the algorithm using the time library functions provided in *C*.
5. **Memory Management:** Proper memory management practices are followed, ensuring that the allocated memory for the `data` array is freed at the end of the program to prevent memory leaks.

2 Implementation of Multi-threaded Program

2.1 Mutexes for Synchronization

The multithreaded implementation utilizes two mutexes for synchronization:

1. `sum_mutex` safeguards the global sum, ensuring only one thread can update it at a time to prevent race conditions.
2. `min_max_mutex` is employed to protect the global minimum and maximum values in a similar manner.

2.2 Thread-Specific Data Structure

The `struct ThreadData` structure is introduced to store data specific to each thread. It includes information such as the chunk of data the thread processes, along with its partial sum, minimum, and maximum values. This ensures that each thread works independently on its assigned portion of the data.

2.3 Global Variables for Results Accumulation

Global variables (`global_sum`, `global_min`, and `global_max`) are utilized to accumulate results from individual threads. Mutexes are employed during updates to maintain thread safety, preventing race conditions.

2.4 Thread Creation and Joining

The implementation involves the creation and joining of worker threads:

1. A loop is used to create and launch worker threads (`pthread_create`), with each thread processing a specific chunk of the dataset.

2. Another loop (`pthread_join`) is employed to wait for all threads to complete before proceeding with the final result calculation.

2.5 Thread-Safe Update of Global Sum, Min, and Max

Mutexes ensure the thread-safe update of global variables:

1. `sum_mutex` protects the partial sum calculated by each thread, allowing only one thread at a time to update the global sum.

2. `min_max_mutex` protects the partial min and max, ensuring synchronized updates to the global minimum and maximum values.

2.6 Proper Memory Management

Memory allocated for thread-specific data (`threadDataArray`) and thread IDs (`threads`) is freed after the threads have completed their execution. This ensures proper resource management and prevents memory leaks.

3 Discussion

In order to evaluate the performance of my programs, I have calculated the time taken by each single and multi thread program to find minimum and maximum value in dataset and also calculate the sum of all entries of the dataset. In single thread program the timer starts before calling these functions and in multithreaded program the timer starts with the creation of threads and it terminates at the completion of task. I have not put timer in the specific thread as indicated in assignment instructions because some I wanted to compare on the basis of completion of task on equal chunk of data. Moreover, I have documented the average time after running the program 20 times on the same parameters to get the accurate results.

Threads	Data Tiny	Data Small	Data Medium	Data Large
2	0.000131	0.007659	0.1495	1.78
4	0.000267	0.004712	0.079659	1.037
8	0.000765	0.005189	0.03786	0.558
16	0.001156	0.005087	0.0586	0.3266
32	N/A	0.009825	0.0634	0.374
64	N/A	N/A	0.06257	0.338

Table 1: Execution times for varying numbers of threads on different datasets.

Dataset	Single-threaded Time (s)	Multi-threaded Time (s)
Dataset tiny	0.000006	0.000131 (2 threads)
Dataset small	0.004812	0.004712 (4 threads)
Dataset medium	0.048945	0.03786 (8 threads)
Dataset large	0.567253	0.3266 (16threads)

Table 2: Comparison of execution times for single and multi-threaded implementations.

In above comparison we can see that there is no benefit of using multithreads when the data is small. But the multi threads take more time in the small data set since there is overhead of creating and maintaining threads and synchronization. But as the size of data set increases, we can see the efficiency of multi-threadeded program also increases. The multithreaded program took almost half of the time taken by single-threaded program for the large dataset. The optimal number of threads for multi-threaded program was found to be 16 in my device.