Concordia University
Engineering and
Computer Science

LABORATORY ASSIGNMENT #1

Merge-sort using multithreading

COEN 346

Team members:

Ahmed Enani (26721281)
Josh Lafleur (40189389)
Eden Bouskila (40170349)

We certify that this submission is our original work and meets the Faculty's Expectations of Originality

Name: Ahmed Enani	ID No: 26721281
Signature:	**Date:** 06/10/2022
Name: Josh Lafleur	ID No: 40189389
Signature: Joshua Lafleur	**Date:** 06/10/2022
Name: Eden Bouskila	ID No: 40170349
Signature:	**Date:** 06/10/2022

1. HIGH-LEVEL DESCRIPTION

1.1 Structures

There are 3 application specific data structures used in the code, not including any standard library types.

1. **int_mutex_S**: Integer value with associated lock. This is to be used to prevent race conditions between threads when retrieving an ID value.
2. **thread_data_S**: Contains the thread ID as well as an integer vector [in out] which holds the elements being given to the thread to be sorted and returns the sorted elements from the thread.
3. **output_S**: Contains the output stream with an associated lock. This is to facilitate locking of the output and allow for a cleaner implementation of multiple output streams whether testing or running in production.

1.2 Methods

There are 2 application specific methods used in the code, as well as the utilization of 6 pthread library calls.

A. Application specific
   a. **main**: This function is responsible for reading in the values from the Input.txt file and creating the top level thread.
   b. **sort_child**: This function is the implemented merge-sort function. It recursively creates child threads until the children threads are given 1 or 2 elements in its input array, which it will then sort. Upon completion of the sort, the higher level threads merge the returned sorted vectors.

B. pthread library
   a. **pthread_create**: Creates a thread.
   b. **pthread_join**: Joins to a thread until it has finished execution.
   c. **pthread_mutex_init:** Initializes a mutex.
   d. **pthread_mutex_destroy**: Destroys a mutex.
   e. **pthread_mutex_lock**: Locks a mutex.
   f. **pthread_mutex_unlock**: Unlocks a mutex.

1.3 Threads

All threads are functions of void* sort_child(void*). The sort_child function is a recursive threading function.

1.4 Program Flow

The main function reads from the Input.txt file and loads these values into an integer vector. The main function then calls the top level thread which is a thread of void* sort_child(void*). The sort_child function then verifies that the input vector of the thread_data_S has a size greater than 2. If this is the case, the function then breaks the input vector into 2 smaller vectors and calls itself recursively until it has 2 or less elements in the thread_data_S vector. Once the sort_child function has 1 or 2 elements, it sorts them and returns the sorted vector through the thread_data_S data structure to the parent thread. When the parent thread receives the sorted children data, it merges both vectors to again return it to its parent.

Each of the main and sort_child functions access the output stream and thread id once the mutex of each is free. This prevents race conditions on each of the data sources and allows for a clean and consistent program execution.

2. CONCLUSION

In conclusion, multi-threading greatly increases the throughput capabilities that a program can handle. By allowing multiple logical cores to attack a CPU-bound problem, it allows for larger amounts of computational resources to be given to a problem.

For our application, we had a lot of issues as 2 of the 3 members were on Windows and the third was on Linux. This presented development road-blocks as Windows no longer supports POSIX functionality. Due to this, we had tried to implement an OS independent framework through compiler defines and pre-compiler MACRO expansions without any luck, and due to frustrations this was given up on.

Outside of the issues faced by conflicting Operating Systems, we had a very happy process of implementing multi-threading support due to the depth of knowledge of a team member which allowed the entire team to work through the problem in a step-by-step fashion going from conceptual implementation all the way through to a working product. This allowed all team members to evaluate their own initial implementation, and assess both the positive and negative consequences of those design decisions.

3. CONTRIBUTIONS

	Tasks	
Team Members	Code Implementation	Writing Report
Josh Lafleur	Multithreading capabilities and mutex implementation	Accuracy of document
Eden Bouskila	Program structure and merge-sort implementation	Transferred information
Ahmed Enani	Program structure and merge-sort implementation	Formatting and organization