Lab 1
Concurrent Programming

Name : Suraj B Thite

A description of two parallelization strategies

I have built the C++ application to parallelly sort the file implementing parallel fork-join based merge sort algorithm and lock based bucket sort algorithm. It stores the sorted list in output file passed as argument in the command line.

The usage for this application with options available in the command line are as follows :

```
mysort [--name] [source.txt] [-o out.txt] [-t NUM THREADS] [--alg=<fjmerge,fjquick,lkbucket>]
```

Strategy to parallelize merge sort :

1. Divide the data into equal sets depending number of threads to be executed in parallel.
2. Calculate the boundaries (indices) for which a thread will operate upon and store it in a structure stored passed as argument to the threads.
3. Create a thread to merge sort the data passed as argument to it in its respective subarray locations.
4. Join all the threads and merge the sub-arrays with the main array resulting in the sorted list.

Strategy to parallelize Bucket sort :

1. Set the bucket count equal to number of threads passed as arguments and determine the maximum value in the dataset to calculate the range of each bucket.
2. Depending upon the number of buckets , divide the input elements in equal halves and store it in an argument structure to be handled by each thread in execution.
3. A thread with bucket function handler and pass the structure argument to it for execution.
4. The handler assigns the elements in their sub-array to an appropriate bucket and then its joined in the main thread thus merging all the sub-array to the main array.

Code Organization :

The below tree describes the brief organization of the code for the Lab1 in SRC directory.

Lab1

|----main.cpp - Main Application File consisting sequential flow of operations.

- main()
- file_to_array()
- array_to_file()
- *fj_merge()
- get_bucket_range()
- *bucketSort()

|----main.h - header File consisting of function declarations and data structures implemented.

|----sorts.cpp - Source File consisting of functions for Merge sort and Quick sort on input data.

- mergesort()
- quicksort()
- partition()
- swap()
- merge ()

|----Makefile - Make file to build the project and application binary mysort.

Files Description:

1. **main.cpp :** The main application source file which consist of sequential flow of instruction to parse input from the user, fetch the data from the file, sort the data depending upon the arguments passed and writeback to the output file. In case of –name argument is passed, the Author name i.e. Suraj Thite is printed and the rest of the process is skipped. The getopt_long() function is used to parse the arguments from the user.
   The pthread are created depending on the arguments passed corresponding to -t flag. This later divides the data set and calculates the parameters passed to each thread. The merge sort is then executed adhering to fork-join methodology or bucket sort implementing locks depending on the --alg flag.
2. **sorts.c :** The source application for merge and quick sort algorithms along with associated member functions. The function takes the array as input arguments and sorts it.
3. **Makefile :** The make file to build the source binaries using gnu C compiler running on Linux Platform.

Compilation Instructions :

Navigate to the working directory of the project and open a terminal.

Run make to compile the project and generate mysort binary file.

Run make clean to clean the application binary from the project folder.

Execution Instructions :

Usage :

mysort [--name] [source.txt] [-o out.txt] [-t NUM THREADS] [--alg=<fj_merge , lkbucket>] as per lab1 requirements.

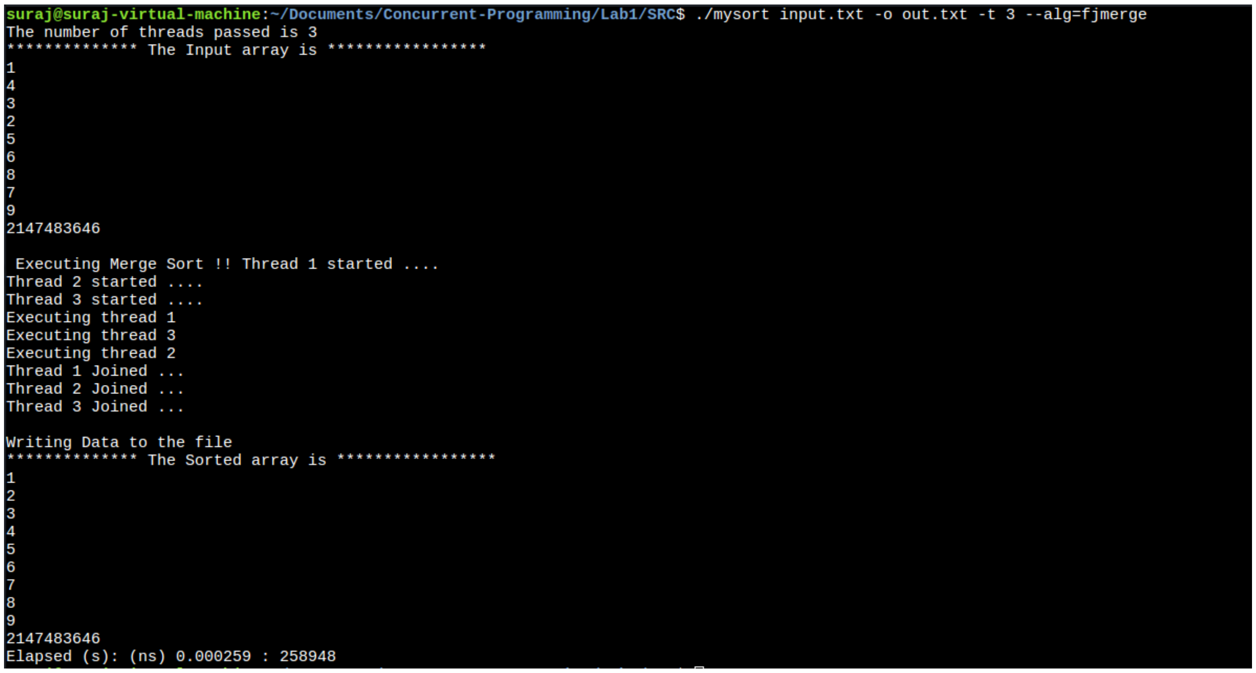
If -- name is passed as argument, the program is terminated.

Failure to open the input file or write sorted data to the output file or incorrect no of arguments passed will stop the further execution printing appropriate messages on terminal.

Extant Bugs :

No bugs in the code as such, the project compiles successfully and passes the test case as mentioned in the assignment requirements.

Parallel merge sort

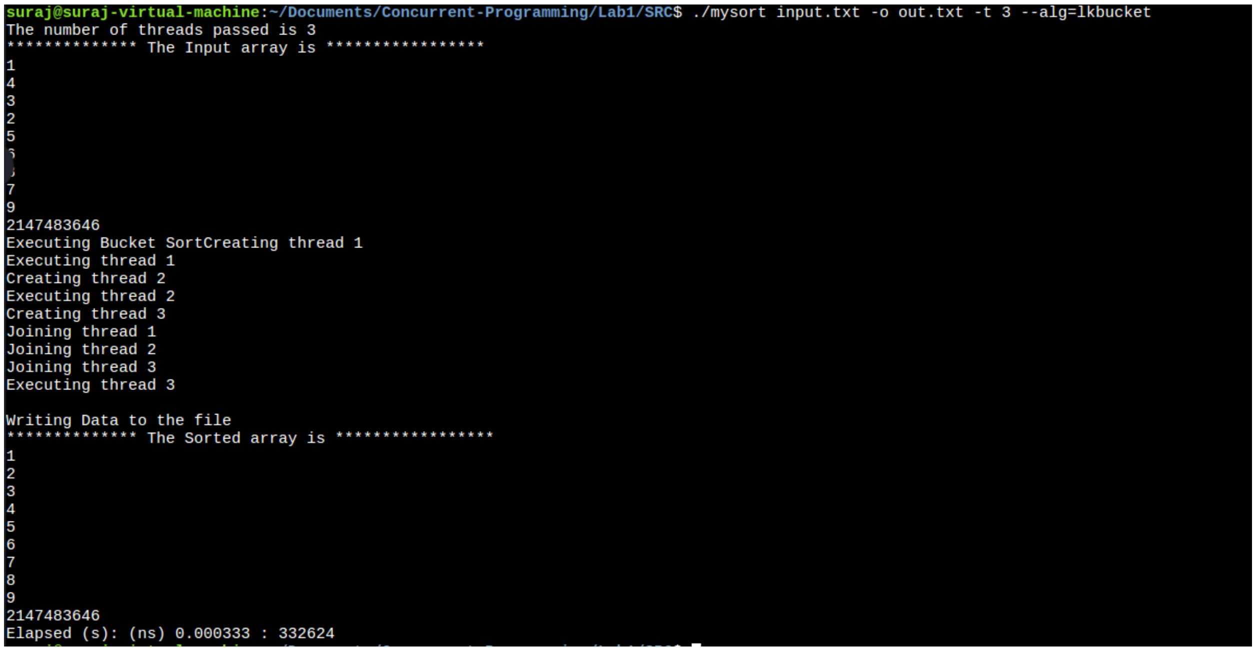


```
suraj@suraj-virtual-machine:~/Documents/Concurrent-Programming/Lab1/SRC$ ./mysort input.txt -o out.txt -t 3 --alg=fjmerge
The number of threads passed is 3
************** The Input array is ****************
1
4
3
2
5
6
8
7
9
2147483646

 Executing Merge Sort !! Thread 1 started ....
Thread 2 started ....
Thread 3 started ....
Executing thread 1
Executing thread 3
Executing thread 2
Thread 1 Joined ...
Thread 2 Joined ...
Thread 3 Joined ...

Writing Data to the file
************** The Sorted array is ****************
1
2
3
4
5
6
7
8
9
2147483646
Elapsed (s): (ns) 0.000259 : 258948
```

Parallel Bucket Sort :



```
suraj@suraj-virtual-machine:~/Documents/Concurrent-Programming/Lab1/SRC$ ./mysort input.txt -o out.txt -t 3 --alg=lkbucket
The number of threads passed is 3
************** The Input array is *****************
1
4
3
2
5
6
8
7
9
2147483646
Executing Bucket SortCreating thread 1
Executing thread 1
Creating thread 2
Executing thread 2
Creating thread 3
Joining thread 1
Joining thread 2
Joining thread 3
Executing thread 3

Writing Data to the file
************** The Sorted array is *****************
1
2
3
4
5
6
7
8
9
2147483646
Elapsed (s): (ns) 0.000333 : 332624
```

The test cases as mentioned in the lab1 assignment document passes successfully as per the above images.