Project 4—Multithreaded Sorting Application

Deadline: Wednesday Oct. 30 11:59 pm.
Submissions: both electronic copy and hard copy. Screenshots: your output.

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term ***sorting threads***) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a ***merging thread*** —which merges the two sublists into a single sorted list in **ascending** order.

Because global data are shared across all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array.

This programming project will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting.

The main thread will output the sorted array once all threads have exited.

Implementation Guideline:

1. Structure "parameters"

```
typedef struct
{
   int from_index;
   int to_index;
} parameters;
```

Since a thread only takes one argument, we need define a structure to pass more than one argument to a thread. In this project, each sorting thread needs the starting index and the ending index of a sublist. The merge thread needs the starting position of each sublist. We use the same structure 'parameters'.

2. `pthread_create` and `pthread_join`

Create sorting threads. Each thread sorts a half of the array. Use the structure defined in 1 to specify which half. Wait for the two sorting threads to terminate. Then create the third thread to merge the two halves. Finally, wait for the merge thread to terminate and print out the sorted array.

3. `void *sorter(void *params)`

Implement the sorting algorithm in this function. You can choose any sorting algorithm. After sorted, the sublist will still be in the original array and same index range.

4. `void *merger(void *params)`

This is the same *merge* like you did in project1. The "merger" will merge two sorted halves into a new array called *result*.