**CSci 230 (Fall 2019)**

Programming Assignment 5

(Due in 2 weeks from the posting date)

(Hand-in should include the code files and the output of the programs)

1. Multi-thread programming of multiple producers and consumers.

Write a multithreading C program to shuffle the contents of 5 files.

There are 5 pre-existing files whose contents are consecutive integers. The contents of the 5 files look like:

Input File 1: 10001, 10002, 10003, ….

Input File 2: 20001, 20002, 20003, ….

Input File 3: 30001, 30002, 30003, ….

Input File 4: 40001, 40002, 40003, ….

Input File 5: 50001, 50002, 50003, ….

All files consist of the same number of integers. You decide the number of integers.

There are 5 producers and 5 consumers. The 5 producer and the 5 consumers share only ONE circular queue. The number of the integers in the files has to be at least twice the size of the shared circular queue.

1. Each producer is associated with a file, with file *k* being associated with producer *k* (1 <= *k* <= 5). Each producer reads integers from its associated file and write the integers to the shared circular queue.
2. Each consumer reads integers from the shared circular queue and write the integers to its own output file.

Your program has to achieve the following effects.

* 1. All output files consist of the same number of integers as the number of elements in the input files.
  2. The integers included in an output file have to be consecutively ordered except the leading digit. For example, the sequence 30001, 50002, 20003, 30004, …. satisfies this requirement. Alternatively speaking, the contents of all output files should be identical if we remove the leading digit.

This effect is called shuffling the integers at every position in the sequence. An example of the possible contents of the output files can be:

Output File 1: 10001, 30002, 50003, ….

Output File 2: 20001, 20002, 20003, ….

Output File 3: 30001, 10002, 40003, ….

Output File 4: 40001, 50002, 10003, ….

Output File 5: 50001, 40002, 30003, ….

Your program must create one producer process and one consumer process. The producer process has to create 5 producer threads within the producer process. The consumer process has to create 5 consumer threads within the consumer process. Each producer thread functions as producer to taken actions as described in i). Each consumer thread functions as consumer to taken actions as described in ii). Each consumer is not allowed to perform any sorting on the integers read from the shared circular queue. Each time, a consumer can only read an integer from the shared circular queue and immediately write this integer to its own output file. The suffices of the integers have to be naturally ordered by your code without performing any kind of sorting.

All processes have to terminate normally upon the completion of the shuffling, as well as that all membership threads are appropriately terminated before the termination of a process. Your program has to ensure that the queue operations only happen at the head and the tail. Namely, searching into the circular queue is not allowed.

Hint: You need to implement barrier in your code in order to maintain the consecutive orders of the suffices of the integers in the output files.

1. Multi-thread programming of parallelized merge sorting.

Write a multithreading C program to perform parallelized merge sorting of a sequence of integers. Link to the tutorial of merge sorting is here:

<https://www.bogotobogo.com/Algorithms/mergesort.php#mergesortC>

Your program has to satisfy the following requirements:

1. Generating a random sequence of integers. For demonstration purposes, this sequence has to be sufficiently different from a naturally ordered sequence. You also need to make the length of the sequence as part of the command-line argument.
2. Only the subroutine of merging two subsequences into a new ordered sequence is allowed. You cannot make use of other sorting methods as the subroutine. Namely, your code has to implement the merge sort method in the parallelized fashion.
3. No recursion can be used.
4. Multiple threads have to be created to perform the merge sort. Each thread takes the same actions as follows:
   1. If the length of the input sequence to a thread is 1, then this thread terminates with a return value of this sequence with a single integer.
   2. If the length of the input sequence to a thread is bigger than 1, then split the sequence into 2 subsequences of roughly the same lengths (only differ by 1 at most). Then, each of the 2 subsequences is fed to a separate child thread. After that, this thread waits for its child threads to terminate. Upon the termination of both its child threads, this thread merges the two ordered subsequences into a single ordered sequence. After merging, this thread terminates with the returning value of the ordered sequence resulted from merging.
5. Each thread has to produce some output to demonstrate its operations:
   1. The input sequence to the thread.
   2. The 2 subsequences after splitting (if the length of the input sequence is bigger than 1).
   3. The 2 subsequences returned from the child threads (if child threads have been created.)
   4. The single sequence after merging.