**CS6456 (F2015) Operating Systems**

**Project 4**

**Title: Concurrent Mergesort**

**Due: October 21, 2015 (11:00 a.m.)**

**Points: 10**

Write a program in C/C++ to sort a sequence of integers in nondecreasing order using the mergesort algorithm. The following shows an example of sorting 8 integers in nondecreasing order using mergesort: 15 8 5 34 24 91 17 68

15 8 5 34 24 91 17 68

8 15 5 34 24 91 17 68

5 8 15 34 17 24 68 91

5 8 15 17 24 34 68 91

Your program should create multiple threads in each round to sort subsets of integers. For example, in round 1 of the example shown above, 4 threads are created to sort 4 subsets, each of which contains 2 integers. All the threads created in round 1 will be synchronized in such a way that when a thread finishes sorting its subset, it waits until all other threads are done sorting before round 2 is started. This process is iterated until the entire sequence of integers is sorted in nondecreasing order.

The integers to be sorted should be stored in a file named **indata.txt** and separated by exactly one space. You may assume that the number of integers stored in the file is power of two (e.g. 8, 16, 32, etc.). You may also assume that the total number of integers stored in the file is no more than 4096. Once the entire sequence is sorted, print all the integers to the screen and terminate your program. The sorted sequence should be printed in such a way that integers are separated by exactly one space (e.g. 5 8 15 17 24 34 68 91).

The best way to handle synchronization among threads for each round of sorting is to use a **barrier** of which its main job is to make a finished thread wait until all other threads are done. The barrier can be implemented in such a way that it is shared by all rounds of sorting via passing a value to it indicating the number of threads concurrently running for a particular round. When a thread finishes its sorting, it issues a call to the barrier and the barrier checks to see if the thread needs to wait (What the barrier does is to decrement the value passed to it. If the value is nonzero, then the thread is blocked on the **condition variable**. Otherwise, the barrier will release all threads from the condition variable via invoking the **cond.broadcast** function). Make sure you use a **semaphore** or **mutex** to protect the barrier from being accessed by multiple threads at the same time.

In your written evaluation (at least one page long), be sure to discuss your implementation, specifically: (1) how you implemented the barrier; (2) how you tested your barrier; (3) whether or not your main thread participates in comparisons; and (4) how you organized (in memory) the intermediate results.

**Miscellaneous**

(1) The project does NOT allow team work.

(2) You must use a single barrier for all rounds of sorting.

(3) You must NOT use the pthread\_join() function to wait for all of the threads to finish for each round of sorting except the last round. You must use pthread synchronization functions to implement the barrier.

(4) You must use a makefile to compile your program.

(5) You must use the Debian VM for this project.

(6) Your implementation must be in C/C++ and use the POSIX APIs for threads and synchronization functions (e.g. mutex, semaphores, condition variables, etc.)

(7) You must use the GNU C/C++ development tools (e.g. gcc, g++).

(8) Turn your project in via **collab** in a **tar** file consisting of (i) all headers you have created; (ii) source code file for project 4; (iii) a make file; and (iv) a write-up showing your design and implementation of the project. The **tar** file should be named as follows: **p4**, followed by the first letter of your first name, followed by your last name, and followed by the file extension (i.e. **tar**). For example, the tar file turned in by **John Smith** should be named **p4jsmith.tar**.

(9) Submit also a hardcopy of the **tar** file in class including the source code file, the header files you have created, and the write-up.

(10) Your program should be successfully compiled. Programs failing to compile will not get any points.

(11) You should use the following cases to test your program: (i) all integers are distinct; (ii) all integers are identical; (iii) all integers are already sorted in nondecreasing order; and (iv) all integers are already sorted in ascending order. You should test your program with the number of integers up to 4096 (you may use a random number generator to generate and simultaneously store 4096 integers in the input file). Make sure you change the base memory size in virtual box to 2046MB (~ 2GB) when you test your program with 4096 integers. The setting can be found under "System" after you launch the virtual box application.