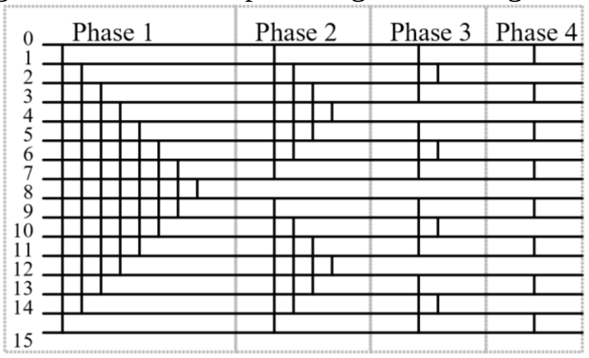
Project 4 Multithreaded Sort (Parallel Sort)

1 Overview

You will implement the balanced sorting algorithm. This algorithm requires log N stages, each of which consists of log N phases, to sort N items. For example, sorting 16 integers would require 4 stages, with 4 phases in each stage. During each phase, N/2 compare-exchange operations are made and these operations can be performed in parallel. The compare-exchange operation sorts the two items in ascending order. The following diagram illustrates the operations of the Balanced Sorting algorithm in one complete stage for sorting 16 items.



As shown in the figure, for N=16, there are 4 phases in each of the 4 stages. Each phase performs 8 compare-exchange operations (which can be done in parallel). The data pairs to be operated are:

Phase 1:
compare-exchange item 0 and item 15 compare-
exchange item 1 and item 14
. . . . . .
compare-exchange item 7 and item 8

Phase 2: compare-exchange item 0
and item 7
. . . . . .
compare-exchange item 3 and item 4
compare-exchange item 8 and item 15
. . . . . .
compare-exchange item 11 and item 12

. . . . .
.

2 Balanced Sort with Parallel Threads

Your need to write a program to implement the balanced sort algorithm. The parallel operations are performed by threads. The outline of the algorithm is as follows.

```
read the number of integers in the input list, N;
while N □ 0 do
{  read N integer numbers and store them in an array; create and
     initialize the semaphores necessary for synchronization;
     create N/2 threads to sort the array using balanced sort
     algorithm; wait for all the threads to finish; print the array of
     sorted integers;
     read the number of integers in the input list, N;
}
```

Each thread performs one compare-exchange operation in each phase. For example, thread 0 can compare-exchange data items 0 and 15 in the first phase, 0 and 7 in the second phase, 0 and 3 in the third phase, and 0 and 1 in the fourth phase. In your program, there should be no sleep() call or any similar method to introduce a delay. Also, you should create threads only once to sort one list of input data (cannot kill the threads after each phase or after each stage).

3 Synchronization

You need to synchronize the threads at the beginning or the end of each phase in order to sort correctly. This is called the barrier synchronization, i.e., all threads should cross the same barrier at the beginning (or the end) of each phase before continuing. You should only use semaphores to achieve the goal. Try to use as few semaphores as possible. You should not use any other synchronization primitives and we will definitely check your code to make sure of that.

You need to initialize your semaphores properly. However, we would like to have control over your semaphores. So you need to read in your semaphore initialization values from an input file “sema.init”. In the beginning of “sema.init” should be the number of semaphores M. Following M should be the initialization value of each semaphore you used. In case you use an array of semaphores, each array is counted as one. For example, if you use one single semaphore and an array of N semaphores, then M = 2. In this case, you need to make sure that the initial values for the semaphores in the array are all the same. Thus, you only need to put one value for an entire array in “sema.init”. In other words, your “sema.init” input file will only have M values. Note that M should be a fixed number, not dependent on N.

In your DesignDoc file, you should explain the purpose of each semaphore and what its initialization value should be (listed in the same order as those in sema.init). You also need to discuss clearly about each semaphore array you use, including the size of the array (likely to be a function of N), the purpose of the array elements, and what the initialization value should be.

4 Input and Output

The input lists of integers to be sorted are given in a file, one number in each line. In each list, the first number is the number of integers to be read in (N), and the N lines following that are all the numbers in the list to be sorted. There may be many lists in the file. The last N will be 0 to indicate the termination of the system. You can assume that there will be no errors in the input file. Also, N will always be 2 for some integer x, $x > 0$. Note: You need to supply the sema.init file and it has been defined in Section 3.

Your program should be able to print output in two modes, the observation mode and the regular mode. In the regular mode, you only need to print the initial list before the threads start and the sorted list after all threads terminate. If N >= 8, you should print 8 numbers per line; otherwise, all numbers in one line.
In the observation mode, besides the required print out of the regular mode, you also need to print out the list of integers being sorted after each phase (same printing rule as above). Also, when each thread passes the barrier (synchronization point) each time, you need to print out a message "Thread t finished stage i phase j", where t is the thread number. In both modes, after finishing sorting and printing the number list, you need to print out a line "------------------------------" to separate the sorting printouts for different number lists.

The input file name and the printing mode should be the command line input to your program (first command line input is the input file name and second is the printing mode). The print mode can be "-r" representing the regular mode or "-o" representing the observation mode.