# CS 4348/5348 Operating Systems -- Project 3

The goal of this project is to let you get familiar with thread programming. You need to use Pthread to create threads and create and use semaphores to synchronize the threads. You are to use these tools to achieve parallel sorting. Your program should be written in C++.

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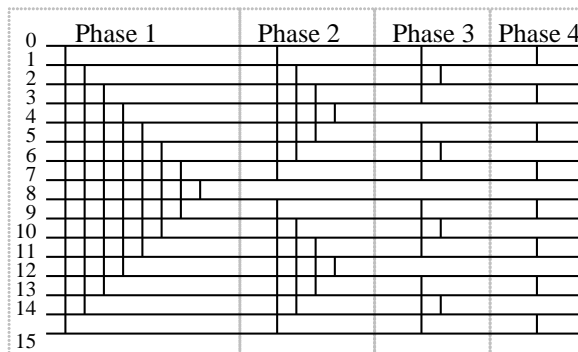
As shown in the figure, there are 4 phases in each stage. Each phase performs 8 compare-exchange operations (which can be done in parallel). The data pairs to be operated are:

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;
}
```

The pseudo code for thread t ($0 <= t < N/2$) is given in the following. Note that, N is always 2^x for some positive number x.

```
repeat for log N stages
    for p = 1 to log N (log N phases) do
    {    num_groups = 2^(p-1);
         group_size = N / num_groups;
         g = t / (group_size / 2);
         gindex = t % (group_size / 2);
         group_start = g * group_size;
         group_end = (g + 1) * group_size – 1;
         data1 = group_start + gindex;
         data2 = group_end – gindex;
         compare-exchange the array items data1 and data2;
         loop for (data1+data2) iterations doing nothing (just to introduce different computation times);
    }
```

Each thread performs one compare-exchange operation in each phase. The pairs of data to be sorted in each phase are given in the figure. 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, but the number of semaphores you use will not impact your grade.

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. 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.

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 sorted list of integers after the list is completely sorted. In the observation mode, you need to print the list of integers being sorted after each sorting phase.

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.