ECE 362 – Embedded Operating Systems

Assignment 6

Complete problem 1 before starting problem 2!

I recommend you regularly run the following command. Especially before logging off.

```
ps -ef | grep $USER
```

This will help you determine if runaway processes. If you do, enter the command kill -9 pid, where pid is the process you want to terminate.

1. Create a program that creates 4 or 8 threads where each thread prints out its thread Id. Your program should accept a parameter:

   `-t <num>` Number of threads. The value should be either 1, 4 or 8.

2. Develop a multithreaded program to sort an array of 64,000 integers. Your program should accept two parameters:

   `-t <num>` Number of threads. The value should be either 1, 4 or 8.

   `-s <num>` Random seed. Use `srand()` initialize randomization with the seed.

   - Use the sort algorithm discussed in class
     - Break the array into equally sized partitions
     - Sort the partitions separately in different threads
     - Merge the results back together. This is a multistep process multiple threads merge partitions and then those partitions are merged in the next step.
     - Write a simple routine that checks the final array is in sorted order. If the array is sorted correctly print "Sort complete." Otherwise print an error message.

   - Each of the worker threads should use a bubble sort algorithm (intentionally slow).

```
for (x = 0 ; x < ( n - 1 ); x++)
   {for (y = 0 ; y < n - x - 1; y++)
      {if (array[y] > array[y+1])
          {temp  = array[y];
          array[y]   = array[y+1];
           array[y+1] = temp;
        }
      }
```

   - Use an array size of 64,000 elements that you randomly initialize using `rand()`

   - You may find it interesting to try different sized arrays and vary the number of threads. During development you will want to use a small array, perhaps start with 2 threads, and printout your array results to confirm the sort and merge worked. The final version of your program should only print out the simple message generated by your simple checker.

   - Along with your code, turn in a comparison of how much time the multithread program takes versus running the same problem using a 1 threaded bubblesort algorithm. To determine how much time is required, you might use the unix time program (or "timer" from the first programming assignment).