Project OS1

Due Date: September 24

Purpose

This assignment lets you get practice using threads yay! and (perhaps) the Linux operating system yay, yay!.

Problem

Sorting has always been a big problem in computer science. In Data Structures, students look at many sorting algorithms and compare their performance. Many of the algorithms break the list into parts, and sort the smaller portions. Naturally, sorting a small list is faster than a large one. Now that we have threads, we can do something similar: break the list into parts, and feed the parts to separate threads that do the sorting. This is your mission.

Input

The only user input to the program is the name of a text file, which contains a list of integers to sort. The first line of the file will contain N, the number of values in the file. This is followed by N lines, where each line contains one integer value. The program should produce an error message and stop if the input file is not found. You may assume that there will be N values in the file.

Output

The output is straightforward: the program should create three or five (see below) text files. The first two (four) should be the sorted sublists created by each thread. These should be called `sorted1.txt` and `sorted2.txt` (and `sorted3.txt, sorted4.txt`, if needed). The last file should be the final sorted list, called `finalAnswer.txt`.

Specifics

- The program must be written in standard ANSI C++ with the POSIX thread library. You may do your development work on any platform, but I will test using Linux.
- You will need a dynamic integer array so that it can be created with the proper size, N.
- Here's the biggie: you should sort two sublists if $N \leq 20000$, and four sublists otherwise. Each sublist should be sorted by a separate thread.
- Each thread should do as much of the work as possible; that is, the calling function should do as little of the "set up" as it needs to get the threads working.
- NO global variables should be used (in the traditional sense; of course threads use the same address space). Drat! This means that the data must be passed to a function/method.
- The list should NEVER be copied, except perhaps when the final, merged list is created (think pass-by-reference!).
- The main function will compute the final sorted list by combining all of the sorted sublists. This can be done in linear time ($O(N)$); look back to your Data Structures notes if you don't remember how.
- The sorting function that you use to sort each sublist is up to you. I suggest using a very simple sort routine, such as insertion sort or bubble sort.

- Remember to comment and to use good style:
  - At the top of your program, you should have a comment that includes:
    - your name
    - the program name
    - the purpose or description of the program
    - what *(exactly)* is input
    - what *(exactly)* is output
  - Use good (mnemonic) variable/constant identifiers
  - Comment all variables and constants
  - Comment all functions, including pass-by-value and pass-by-reference parameters

Notes

This is not a very long program, but threads can mess with your head. I suggest that you first write a program that reads the list of numbers and sorts them, so that you know the program is reading the list correctly and that your sort is functioning properly. Only then try to add in the code for the threads.

You can earn up to 90 points if the program works with two threads; a working program with four threads can earn you the full 100 points.

Submit your source code by emailing it to me at mgousie@wheatoncollege.edu before 11:59:59 on the due date.

Submit a printed version of your source code in class on September 25^{th}.

Well, first of all, you're beginning with an illogical premise
and proceeding perfectly logically to an illogical conclusion,
which is a dangerous thing to do.
– Donald H. Rumsfeld, 11/1/2001