CS 385 –Operating Systems – Fall 2013
Homework Assignment 3
Searching for Order Among Chaos Using Threads

Due: **Friday November 15. Electronic copy due at 2:30, optional paper copy may be delivered to the TA.**

Overall Assignment

For this assignment you are to write a program using PTHREADS to analyze a data file looking for patterns. Each thread will analyze a different section of the file in parallel, recording their findings in a common queue of results, which a separate thread will read from and print the latest results to the screen as they arrive. The threads will use a branch-and-bound technique, discontinuing analysis as soon as it can be determined that a result will not be found that is any better than what has already been found (by any thread.)

Command Line Arguments

The command line for this program will be as follows:

orderSearcher inputDataFile nThreads [. . .]

- inputDataFile is a data file to be read and analyzed. Some potential files will be provided.
- nThreads is the number of threads to use to search for patterns in the data, not counting any support threads such as the thread to print out results as they are found. If this value is specified as zero, then the program should do the best job it can of finding patterns in the data without using threads.
- You may have additional parameters that follow the above parameters, depending on how you choose to handle some of the issues discussed below.

Input Data

The data in the file will be read in as a series of unsigned bytes (0 to 255 each), in binary.

Evaluation Criteria

The whole point of the assignment is to use threads to find blocks of data that are as "ordered" as possible, based on a number of different criteria. Note that the idea is to use **all** of the different criteria you can, not to just pick one or two off the list. There are a couple of different ways the criteria can be combined, as discussed below. Here are some potential criteria for determining order:

- The range, defined as the difference between the largest and smallest value in the data set.
- Maximum absolute value of change from one value to the next: $\max_i |X_i - X_{i+1}|$
- The sum of (absolute value of (change from one value to the next)): $\sum_i |X_i - X_{i+1}|$
- Standard deviation of the data: $\sqrt{\frac{\sum_i (X_i - \bar{X})^2}{N}}$
- Standard deviation of the change in data: $\sqrt{\frac{\sum_i (\Delta_i - \bar{\Delta})^2}{N}}$ where delta is defined as $\Delta = X_i - X_{i+1}$
- Best regression coefficient fit to a straight line, Nth order polynomial, sine curve, FFT?
- Periodicity? (Cyclic repeating patterns.)

Branch and Bound

The concept of branch and bound is to split the set of all possible solutions into smaller subsets (branching), and then eliminating entire subsets when it becomes clear that no solution within that subset can ever be better than a previous solution already found in any subset. For this application what that means is that all of the threads will share knowledge of the "best" solution found so far by any thread, and will cease analyzing any new candidates as soon as it is known that they cannot improve upon the best found so far.

For example, suppose we are considering the criteria $\sum_i |X_i - X_{i+1}|$ and we know that the best solution found so far using this criteria has a sum of 100. As soon as the sum we are calculating exceeds 100, then we can stop examining this candidate, at least with respect to this criteria.

For a more challenging example, suppose we are considering the standard deviation, $\sqrt{\frac{\sum_i (X_i - \bar{X})^2}{N}}$, and the best solution found so far is 10.0. That means that in order for this to be an improvement, the quantity under the root would have to be smaller than 100, and the sum term would have to be smaller than 100 * N.

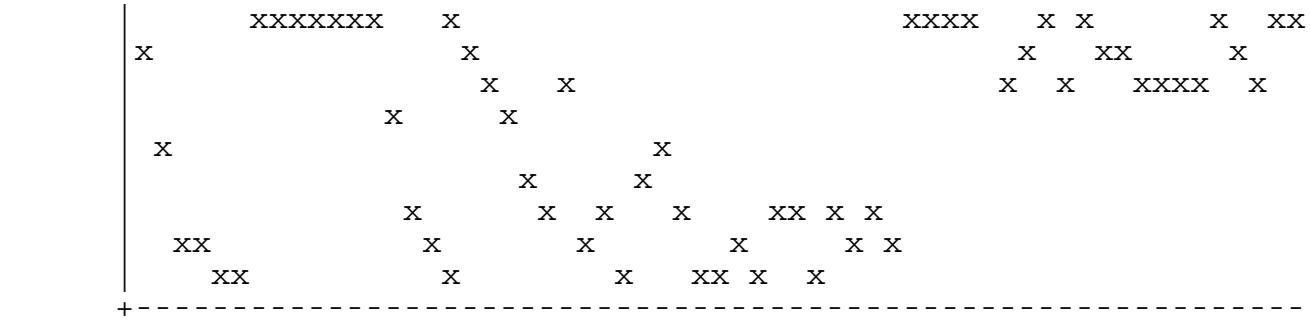
Issues to be Resolved

- **Examine effect of different scope contention, e.g. process versus system. This is required.**
- What is the best way to break up the data amongst threads?
  - Divide data into nThreads equally sized chunks?
  - Divide data into "good" sized chunks, e.g. 4K, and assign chunks to threads?
  - Divide data into "good" sized chunks and let threads pull from a common queue?
- How should multiple criteria be combined?
  - Find the "best" of each criteria independently of the others.
  - Use a priority system, where one criteria is the "most important", and the next criteria used to break ties, etc.
  - Use a weighting function, such as the sum of the criteria, the average of the criteria, the minimum of all the criteria, or some weighted composite. with different weighting values.
- How big of a string should be examined for order? 80 chars? or something else?
- What is the best way to "plot" the data (see below), particularly if there are more than 80 chars?
- How should the data in the file be interpreted?
  - unsigned char (0 to 255), short (2 bytes, 0 to 65535), or int (4 bytes, 0 to 4.29 billion)
  - float - 32 bits, positive or negative, from very small to very large, fractions possible.
  - 24-bit ints. Read three unsigned chars, (red, green, blue), multiply the first by 256^2 (65536), the second by 256, and add the three results.
- What data should be shared among threads?
  - Queue (vector) of candidate starting positions, criteria used, criteria value
  - Best results found so far
  - File handles?
- Are semaphores needed?
- Is vector< > thread-safe?
- Explore performance on multi-core SMP machines. (See http://addgadgets.com for ideas.)
- Should each thread read their own data from the file, or should all the data be loaded into memory before launching threads?
  - If each thread reads their own data, how to handle file pointers?
- How should the border between adjacent chunks of data be handled?
  - Can the "preceding" thread read ahead into the adjacent block of data?
  - Should the border sections be ignored?
  - What about the end of file?

Program Output

As each thread finds a “good” candidate section of data, they should put the parameters, (starting location, possibly length, possibly other information), into a queue, and a separate thread should read the candidates out of the queue and “plot” the results as they come in, possibly similar to the example shown here. When the program finishes it should report final statistics, such as the number of threads used, the number of candidates found, the time it took, and also report the best candidates found, with their criteria and plots.

Note: During initial development you may want to just have the "plotting" thread report the numerical results found, e.g. the constantly decreasing values of the evaluation criteria, and worry about plotting later.



Required Output

- All programs should print your name and ACCC account ID as a minimum when they first start.
- Beyond that see above.

Other Details:

- The TA must be able to compile your program (on bert and ernie) by typing "make orderSearcher". Provide a makefile if necessary. As always, you are free to develop wherever you wish, but the TA will be grading the programs based on their performance on the CS department machines.

What to Hand In:

1. Your code, **<u>including a makefile if needed, and a readme file,</u>** should be handed in electronically using Blackboard.
2. The purpose of the readme file is to make it as easy as possible for the grader to understand your program. If the readme file is too terse, then (s)he can't understand your code; If it is overly verbose, then it is extra work to read the readme file. It is up to you to provide the most effective level of documentation.
3. The readme file must specifically provide direction on how to run the program, i.e. what command line arguments are required and whether or not input needs to be redirected. It must also specifically document any optional information or command line arguments your program takes, as well as any differences between the printed and electronic version of your program.
4. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
5. **For this assignment you must also document the results of your experimentation, as well as reporting the best results found by your program.**
6. A printed copy of your program, along with any supporting documents you wish to provide, (such as hand-drawn sketches or diagrams) may be delivered to the TA, and must match exactly the electronic submissions.

7. Make sure that your **name and your ACCC account name** appear at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

- This assignment already has a lot of flexibility for exploration.