CS 281: Assignment 1 (100 points)

Due: September 16, 11:59 PM US Central Time
Updated: January 14, 2015

Objective:

The objective of this assignment is to become familiar with the UNIX-style process management concepts and APIs, namely fork, exec, and wait. During this exercise you will use another command line utility called sort.

Useful Man Pages:

- man 2 exec
- man 2 fork
- man 2 wait
- man 1 sort

Overview: Your program will sort an input file containing a list of numbers, one per line, and output a file that contains those numbers in sorted order. In Linux this sorting can be done using the sort utility. To help you with this assignment, we have provided a number of helpful files. These files include:

- `mergesort.cxx`: This file contains the `main` function, which processes command line arguments and invokes the sorting functionality that you will implement. We do not envision any changes are necessary to this file. The command line options have already been included in the file.
- `utility.hxx` and `utility.txx`: This contains utility functions for file I/O. It is not expected that you will make any changes to this file for this assignment.
- `sort_strategy.hxx`: This file contains an abstract base class for the `Sort_Strategy` class. We have provided `Insertion_Sort` as an example. An empty definition for two strategies, `gnusort1` and `gnusortn`, has also been declared.
- `sort_strategy.txx`: This file contains the implementation of the `Insertion_Sort` class and a factory method `get_strategy`. It is expected that you will add the implementation for `gnusort1` and `gnusortn`. Review the `get_strategy` function that constructs an instance of the sort strategy.

Your program should implement two strategies: one called gnusort1 and one called gnusortn.

Step 1: Use the GNU Sort utility to sort the input and implement the gnusort1 strategy

In the first step of this assignment, you will use the GNU 'sort' utility to sort the content of the array you are provided. The array you are provided should be read in as an input file. You will output the array you are given to a temporary file (please see `utility.hxx`). You will then spawn a child process:

- In the child, you will construct appropriate command line arguments to invoke the GNU sort utility so that it reads from the previously created temporary file and sends its output to a second temporary file.
- In the parent, you will wait for the child to complete. If the child was successful (i.e., returns 0), you will load the sorted array and return. Note that the `get_strategy` function is called in the main function and produces the output file from the sorted data. If the child was not successful, you will throw an instance of `std::exception`, containing the error encountered by the child process.

Step 2: Spawn multiple children to sort the input and implement the gnusortn strategy

In this step, you will use the additional command line argument '–num-parallel' (n), which will be an integer in the range 2-8, that specifies how many subprocesses should be used to sort the input. Based on the value of n, you will split the input into n separate pieces and spawn n instances of the gnusort1 utility to sort these subproblems (use the gnusort1 strategy), then spawn an $(n+1)^{th}$ instance of GNU sort to merge the results.
Note: The GNU sort utility can also be used to merge the input of multiple files together, e.g. `sort test3.data test1.data test2.data --numeric-sort --merge -o test.data`. See the man page for details.
The following steps should be used to complete this part of the assignment:

1. Write the input data to to n temporary files, dividing it equally.
2. Spawn n children, one to sort each of these temporary files. Each child should use the executable gnusort1 strategy developed as part 1 of this assignment.
3. Wait for the children to complete, and if they were successful, spawn another child process to merge the output of the previous step. This merging can be done using the standard GNU sort utility with the merge feature as described above.
4. Wait for this child to complete and then read in the result.

Evaluation

Your assignment will be graded according to the following criteria:

- **30 points**: Completion and correct execution of "Step 1" functionality.
- **30 points**: Completion and correct execution of "Step 2" functionality.
- **20 points**: Correct and insightful programming style.
- **10 points**: Consistent formatting and indenting style.
- **10 points**: Correct and useful commenting. For guidelines, please see
  http://www.dre.vanderbilt.edu/s̃chmidt/cs251/commenting.html

Step One is due by **Thursday, January 22** for code review. We encourage you to attempt to have step two completed at that time so it can benefit from review. **Step Two** is due by **Thursday, January 29**.
Solutions received after the due date will not be graded. Solutions that do not compile will not be graded. Please remember to both commit your code to your repository and push it to the server.
All work must be your own. Collaboration or sharing code is not permitted.

1.