David Wade

Concurrent Programming

Lab 0

Write-Up

The code for this lab is organized mainly into three separate C++ files, each with its own corresponding header file. There is a file that implements the Quick Sort sorting method, another that implements the Merge Sort sorting method, and the last being the Driver file. In addition, a Makefile is included. The Quick Sort and Merge Sort code organization is pretty straight forward. The merge sort starts by continually splitting the list of data in half until each element is in its own unique list, then is merged back together through comparisons, finally returning a sorted list of the original data. Similarly, the Quick Sort method is also a divide and conquer algorithm. This algorithm essentially chooses a pivot, I chose the first index to be the pivot, and then calls a partition function to arrange all smaller elements of the pivot to the left and all bigger elements to the right. The pivot ends up in its sorted index when the partition method is finished. Idealy, for optimal run time, one would choose the first pivot to be one that is the median of the list, however, choosing either the fist, middle, or last element will work as well. Once the first pivot is partitioned, the algorithm then recursively goes through the left side, then followed by the right side elements. The hardest part of this lab for me was creating the driver file. I have never personally used getopt_long() for parsing the command line arguments, so a bit of a learning curve was required in order to create a properly qorking driver. After I figured out getopt, the rest of the driver was very straight forwar. I parse the lines

of the input file, add the elemnts into a vector, sort the vector in the algorithm of the user's choosing, and output the new sorted vector into an output file. In order to compile the program, the user simply needs to type a single make command and the entire lab will compile and create object files for each .cpp file. In order to execute the program, the user must do an executable called "mysort". The syntax for the executable is as follows "./mysort [--name] [sourcefile.txt] [-o outputfile.txt] [--alg-<merge,quick>]". If the name argument is given, then the program will display my name and terminate. Otherwise, the program will read the input source file, sort the file in the algorithm of the user's choosing, and then output it to a file named of the user's choosing. If an invalid sorting algorithm is inputted, the program will give a Usage error and remind the user of the proper syntax.