CS214 Project 2: Basic Data Sorter - Multithreaded Analysis & ReadMe

Alan Sempruch
Sarah Khasawneh

Time Analysis for both multithreading and multiprocessing using the same tests run multiples times. Each test was run and the average run time was calculated and input into the tables below.

Project 2: Run Time Analysis

Test	Total Num Lines	Real (s)	User (s)	System(s)	Description
TestA	5044	0.203	0.119	0.011	Long Subdir: 13 nested Dir> 1 file at end
TestB>folder1	10	0.002	0.002	0.001	1 Dir> 1 File
TestB>folder2	80	0.012	0.002	0.003	1 Dir> 2 Files
TestB>folder4	215	0.031	0.020	0.004	1 Dir> 4 files
TestB>folder8	902	0.072	0.040	0.006	1 Dir> 8 fi
testdir_level_2	60,000	4.066	1.795	0.333	3 Dir> 600 files
TestC	1902	0.655	0.557	0.088	Combination of above

Project 1: Run Time Analysis

Test	Total Num Lines	Real (s)	User (s)	System(s)	Description
TestA	5044	0.153	0.088	0.019	Long Subdir> 13 nested Dir> 1 file at end
TestB>folder1	10	0.124	0.074	0.009	1 Dir> 1 File
TestB>folder2	80	0.009	0.005	0.005	1 Dir> 2 Files
TestB>folder4	215	0.015	0.006	0.003	1 Dir> 4 files
TestB>folder8	902	0.015	0.006	0.006	1 Dir> 8 files
testdir_level_2	60,000	0.665	0.009	0.024	3 Dir> 600 files
TestC	1902	0.270	0.010	0.029	Combination of above

·*Total number of tests run: 56*

- Questions:

Is the comparison between run times a fair one? Why or why not?
No it is not because threads share the heap, and the way we designed our program, we modify the heap frequently, resulting in many mutex locks needing to be placed, thus leading Project01- Multi-Processing taking lead with faster run time per test. It is also not fair because Project 2 requires the final set of data to be sorted while Project 1 does not.

What are some reasons for the discrepancies of the times or for lack of discrepancies?
The code is not optimized for multithreading. We are reusing code from previous programs and upgrading it depending on the assignment. There is also the use of a global array, in which all files are input to. As the number of files increases, the run time for running though the array increases with it.

If there are differences, is it possible to make the slower one faster? How? If there were no differences, is it possible to make one faster than the other? How?

It is possible to make the slower one faster. It would however involve changing the current data structure used, as the current one is not optimized to be modified by multiple threads simultaneously.
On the other hand, if there were no differences, the multithreading version could be made faster by ensuring that very few mutex locks are used. This would require the code to be optimized for one data structure being modified simultaneously by many threads without resulting in errors.

Is mergesort the right option for a multithreaded sorting program? Why or why not?
Mergesort is not the right option for a multithreaded sorting program because it is expensive even though it runs in O(nlogn) time. Additionally, mergesort does not take advantage of partially sorted data (which we have at the end of the program's runtime) as efficiently as insertion sort does.