COEN 346

Lab Assignment #1 - Merge-Sort using multithreading

Jose Ricardo Monegro Quezada - 40087821

Karl Noory - 40059592

Yevhen Haydar - 40024141

Introduction:

In this laboratory assignment, recursive multithreading was explored by implementing a merge-sort algorithm which is within itself also recursive. It is required to sort a sequence of numbers taken from a text file (input), and write to a separate file (output) on every occasion that a thread is started and finished. It is possible to implement multithreading as part of merge-sort if the sort function itself is executed by a thread.

Discussions:

1. File I/O

To run the program, the user must call the executable with the path to the input.txt file as a CLI parameter, these parameters are stored in the argv[] vector along with the count of parameters in the argc integer. The program will then attempt to open the file using functions from the <fstream> C++ library, iterate over every line and store the integer value in a vector called integerVector using a while loop. If an error occurred while opening the file, an error message will be thrown to the user and the program will terminate, this is handled by an if statement that evaluates the returned value from the is_open() function. If it is successful, the program will set the output.txt file path to the same path as the input.txt using std::string manipulation tools. The logic behind determining the output.txt directory is designed to work on both Windows and Linux systems by looking for both forward slashes and backslashes. The following section will detail how we take the integerVector and pass it to the sort_main(...) function.

2. Merge Sort

Multithreading was implemented using the <thread> library in C++. Merge sort is initiated when sort_main(...) function is called and given the array (integerVector treated as an array) to be sorted. Within sort_main(...), we verify that the array is not empty, and proceed to create a parent thread that will execute the sort(...) function. When initiating a thread using std::thread, we need to specify the function that the thread will be executing, and the parameters if any are required to be passed to that function.

Merge-sort being a recursive algorithm, we inherently need to implement recursive multithreading, as within each sort(...) function call, we will call the sort(...) function twice, on two sub-arrays. Everytime, the sort(...) function is called by initiating a thread.

Everytime a thread is created, we will use thread_name.join() to wait for that thread's completion. This is important when implementing merge-sort because the outcome of each thread is essential for the thread within which it was created. Once both sub-arrays are sorted, they are merged together by a merge(...) function. This function is responsible for putting the numbers in the increasing order.

Conclusion:

In conclusion, multithreading is especially helpful in operations where there is either computing hardware left to be utilized, or where asynchronous tasks during a program's operation are needed. In this particular scenario however, while they might increase performance if the number of threads is limited to the amount CPU cores available, merge sort has the potential of easily launching more threads than what the system can efficiently handle.