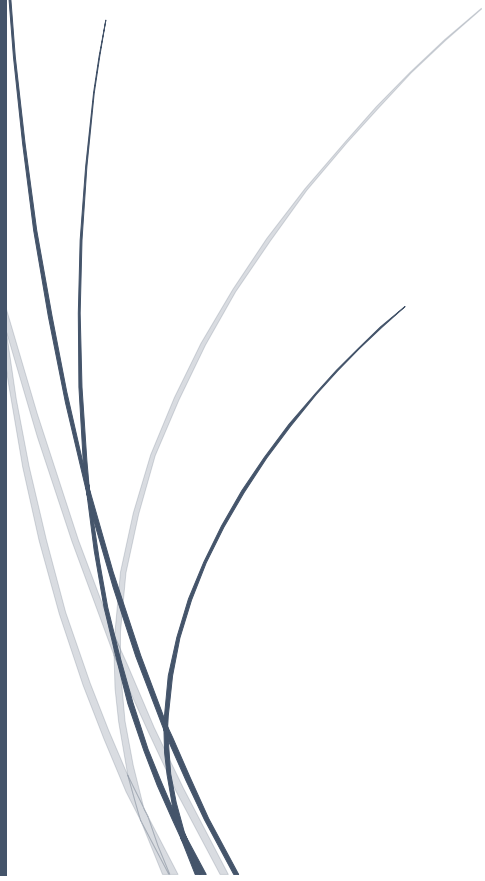
OPERATING SYSTEMS

THREADS



Description

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*.

Threads are not independent of one other like process as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Major Functions

My Thread application supports the following:

1. Matrix Multiplication

   I implemented two variations of this algorithm:

   a. The computation of each element of the output matrix happens in a thread.
   b. The computation of each row of the output matrix happens in a thread.

   For both variations, I've computed the elapsed time for each of them, and display the output of the multiplication.

2. Merge Sort

   Merge sort is an O (n log n) comparison-based sorting algorithm. It is a divide and conquer algorithm. Conceptually, a merge sort works as follows:

   a. If the list is of length 0 or 1, then it is already sorted. Otherwise:
   b. Divide the unsorted list into two sub-lists of about half the size.
   c. Sort each sub-list recursively by re-applying the merge sort.
   d. Merge the two sub-lists back into one sorted list.

   So, I've implemented it using Pthreads.

   Each time the list is divided; two threads are created to do merge-sort on each half separately.

   This step is repeated recursively until each sub-list has only one element.

   When the program finishes, it should print out the sorted array.

Organization of Code

The code is divided into 2 main partitions:

Threads_MatrixMultiplication

The following part of the code is following the main functions:

1. ReadMatrices(): Where I'm reading the 2 matrices with their dimension from input.txt file. Once read, I saved them into 2 global variables for future use.
2. ComputeVersionA(): where implementing the matrix multiplication using threads as described into part (a).
   Each created thread is responsible for calculating an element in the result matrix. That's why we are creating threads inside the second loop [when retrieving an element in the matrix].
   We create a new threat using pthread_create function to compute a matrix element, and after finishing all computations we join all these threads created by their IDs by pthread_join function.
3. ComputeVersionB(): where implementing the matrix multiplication using threads as described into part (b).
   Each created thread is responsible for calculating a row of elements in the result matrix. That's why we are creating threads inside the first loop [when iterating on each row in 2 matrices].
   We create a new threat using pthread_create function to compute a matrix element, and after finishing all computations we join all these threads created by their IDs by pthread_join function.
4. StoreResult(): where taking our calculated result matrix and saving it in output.txt file along with its taken time to do the operation.

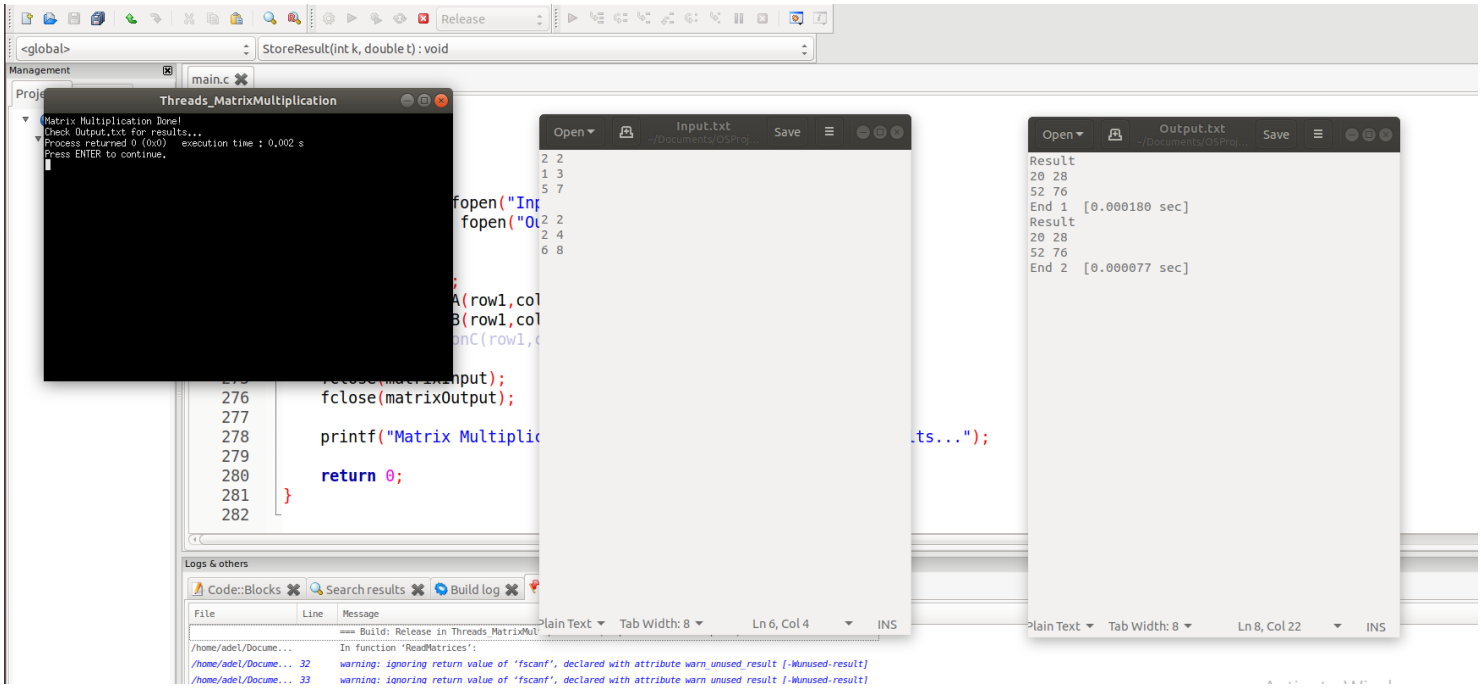
Threads_MergeSort

The following part of the code is following the main functions:

1. ReadMatrices(): Where I'm reading the array with its size from input.txt file. Once read, I saved it into a global variable for future sorting.
2. Sort(): where the starting of my sorting algorithm.
   We begin our Merge Sort algorithm in a thread containing the required parameter in info struct.
   We calculate also here the total time taking in sorting.
3. MergeSort(): where the core algorithm is being executed.
   Merge sort is a divide an conquer algorithm. Its idea is to divide the array into 2 parts and solve it recursively. After sorting each part recursively, I merge the 2 parts of the array.
   Solving each half recursively is done on a separate thread.
   This means that each time I recursively call merge sort function on a smaller part of the array , it's done on a separate thread.
   Before merging 2 sorted halves, I must merge the corresponding 2 threads that sorted each half.
4. Merge(): A sub function used in merge sort algorithm.
   This function is responsible for merging two sorted sub- parts of the array given the start and the end indices of each sub-part.
   This functionality is done by iterating in order on each half and comparing element by element in order to have a new sorted sub array combining the two halves.
5. StoreResult(): where taking my sorted array and saving it in output.txt file along with its taken time to do the operation.

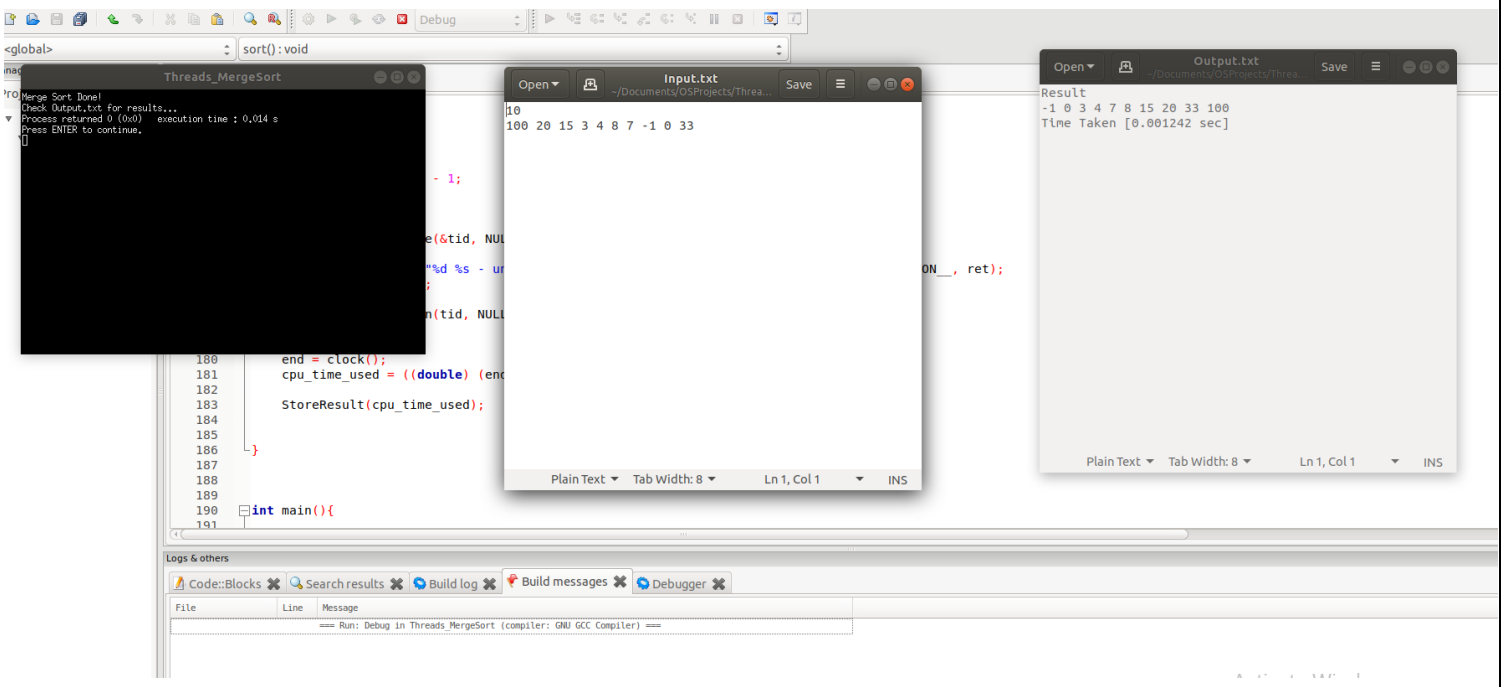
Sample Runs & Screenshots

Threads_MatrixMultiplication



Comparing the time taken by the 2 method I found that making a separate thread for each row calculation is much faster than creating a thread responsible for calculating each element in the matrix

Threads_MergeSort



Thank You