28th December 2022

Operating Systems

Assignment

3

Submitted by:

Syed Ali Raza

200901028

Submitted to:

Ma’am Asia Aman

Code

```
#include <iostream>
#include <thread>
#include <vector>

using namespace std;

void mergeSort(vector<int>& arr, int start, int end); //function declaration

int main()
{
    cout << "Enter the number of elements in the array: ";
    int no_of_elements;
    cin >> no_of_elements;

    cout << "Enter the elements: ";
    vector<int> arr(no_of_elements);
    for (int i = 0; i < no_of_elements; i++)
    {
        cin >> arr[i];
    }

    thread t1(mergeSort, ref(arr), 0, arr.size() - 1);
    t1.join();

    cout << "Sorted Array (by Merge Sort): ";
    for (int i = 0; i < no_of_elements; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}


vector<int> merge(vector<int> left, vector<int> right)
{
    vector<int> sorted_array;
    while (left.size() > 0 && right.size() > 0)
    {
        if (left[0] < right[0])
        {
            sorted_array.push_back(left[0]);
            left.erase(left.begin());
        }
        else
        {
            sorted_array.push_back(right[0]);
            right.erase(right.begin());
        }
    }

    while (left.size() > 0)
    {
        sorted_array.push_back(left[0]);
        left.erase(left.begin());
```

```
    }

    while (right.size() > 0)
    {
        sorted_array.push_back(right[0]);
        right.erase(right.begin());
    }

    return sorted_array;
}

void mergeSort(vector<int>& arr, int start, int end)
{
    if (start < end)
    {
        int mid = (start + end) / 2;
        vector<int> left(arr.begin() + start, arr.begin() + mid + 1);
        vector<int> right(arr.begin() + mid + 1, arr.begin() + end + 1);

        thread t1(mergeSort, ref(left), 0, left.size() - 1); // One thread is
assigned the left portion
        thread t2(mergeSort, ref(right), 0, right.size() - 1); // One thread is
assigned the right portion

        t1.join(); //Two cores = Two threads
        t2.join();

        arr = merge(left, right);
    }
}
```

Documentation

This is a C++ implementation of the merge sort algorithm, using the **thread** library to perform the sorting in parallel fashion.

- **mergeSort():**

The **mergeSort** function takes in a reference to a vector of integers (**arr**) and the start and end indices of the portion of the array to be sorted. It first checks if the start index is less than the end index, which indicates that the portion of the array being sorted has more than one element and thus can be further divided. If this condition is true, the function divides the array into two halves by finding the midpoint. It then creates two new vectors, left and right, containing the elements in the left and right halves of the original array, respectively.

- **Use of Multithreading**

Next, the function creates two threads, t1 and t2, and assigns them the tasks of sorting the left and right halves of the array using the **mergeSort** function, respectively. It then waits for both threads to finish using the join function. Once both threads have finished, the function merges the left and right halves using the merge function and assigns the result back to the original array.

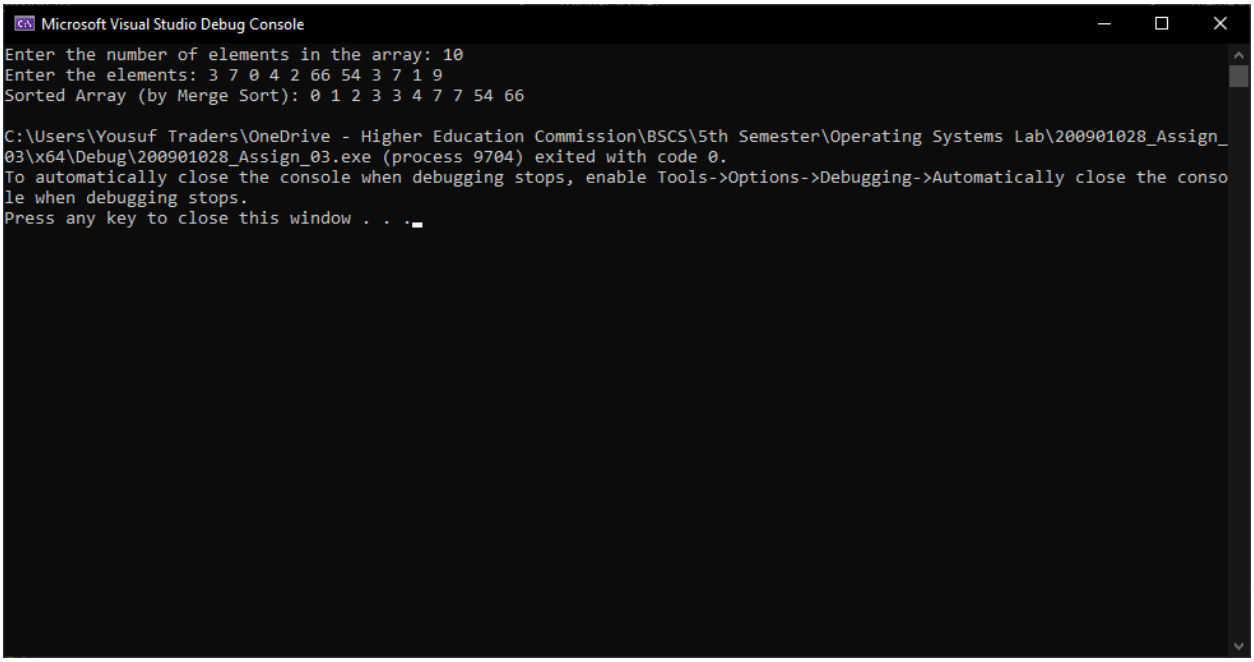
- **merge():**

The merge function takes in two vectors of integers, left and right, and returns a new vector containing the elements from both vectors in sorted order. It does this by repeatedly comparing the first elements of the left and right arrays and adding the smaller of the two to the **sorted_array** vector, until one of the arrays is empty. It then adds the remaining elements from the other array to the **sorted_array** vector.

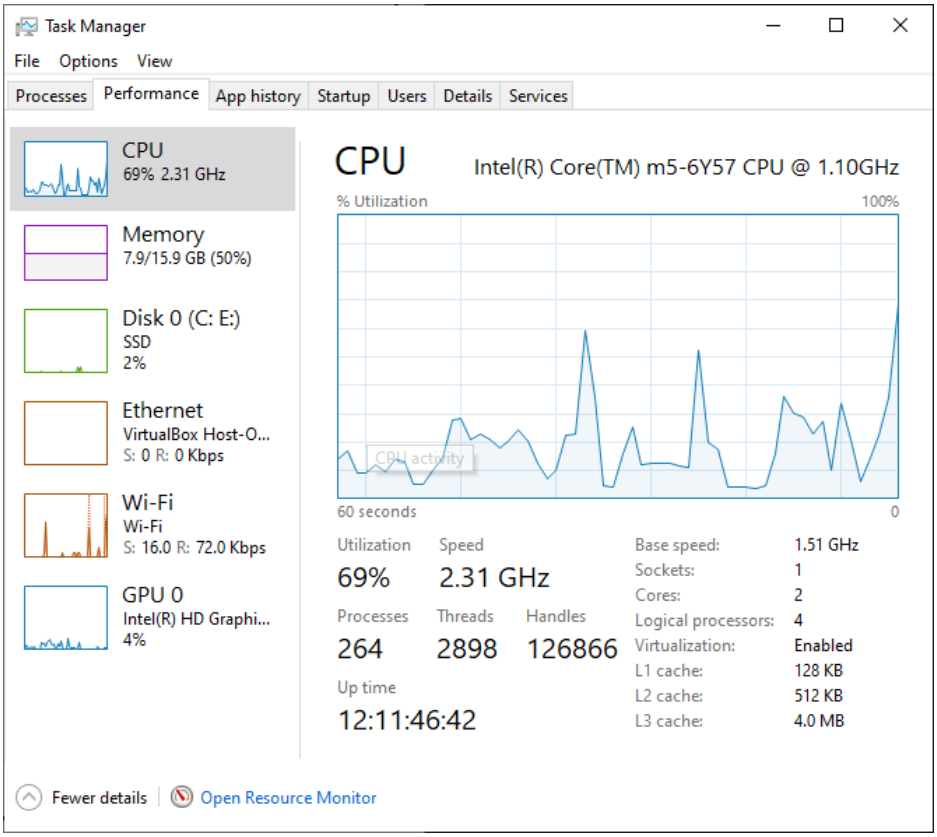
- **main()**

In the main function, the user is prompted to enter the number of elements in the array and the elements themselves. A thread, t1, is then created to sort the array using the **mergeSort** function. The main function waits for t1 to finish using the **join** function, and then prints the sorted array.

Output



No of Cores: 2



MAC Address: