**OPERATING SYSTEM**

**A picture containing graphical user interface

Description automatically generated**

**ASSIGNMENT: 03**

|  |  |
| --- | --- |
| **Name** | M. ZAIN PARACHA |
| **Registration Number** | 200901076 |
| **Batch & Section** | CS (01)-A |
| **Assignment** | OPERATING SYSTEM |
| **Submitted to** | MA’AM ASIA AMAN |
| **Date** | 25th Dec, 2022 |

Q. This assignment is for testing the understanding of scheduling algorithms you have learnt so far and improve the performance of operating system with some advancement in already available algorithms. The learning outcome of this assignment is that you are able to identify core functions of operating system and how to enhance the performance with the integration of sorting algorithm.

You have learned merge sort in data structures which sorts an array in nlogn time, it is a divide and conquer technique. We can enhance the performance of merge sort using multithreading. First of all, you have to check the processor cores of your system, let’s suppose your system processor has 4 cores. Now you have to create 4 threads and divide the array among these threads and sort them using merge sort. You have to take size of array and array elements from user.

For this question you have to submit three things multithreaded merge sort c/c++ code, screenshot of available cores in your system and also the mac address screenshot of your system. No need to implement merge sort from scratch you can use merge sort code from internet but provide the link of source in the code.

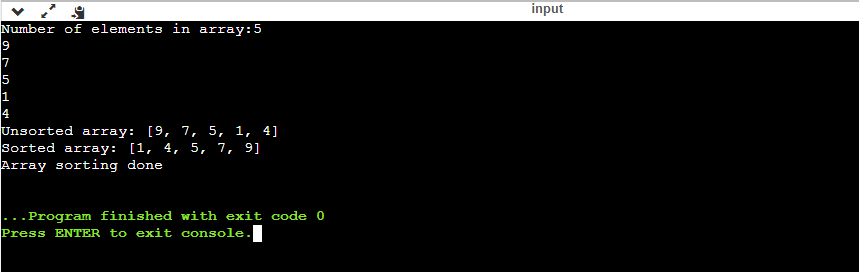
**MULTITHREADED MERGE SORT:**

Multithreaded merge sort is very similar to a regular merge sort, however, when recursively calling the merge sort each half of the list, set up your algorithm to call each merge in a new Thread. You can then wait until both threads are finished, and then merge the two lists together, and return.

**CODE:**

|  |
| --- |
| #include<iostream>  #include<thread>  using namespace std;  void Merge(int \*a,int low,int high,int mid)  {  int i,j,k,temp[high-low+1];  i = low;  k = 0;  j = mid + 1;  while (i <= mid && j <= high)  {  if (a[i] < a[j])  {  temp[k] = a[i];  k++;  i++;  }  else  {  temp[k] = a[j];  k++;  j++;  }  }  while (i <= mid)  {  temp[k] = a[i];  k++;  i++;  }  while (j <= high)  {  temp[k] = a[j];  k++;  j++;  }  for (i = low; i <= high; i++)  {  a[i] = temp[i-low];  }  }  void MergeSort(int \*a,int low,int high)  {  int mid,b;  if (low < high)  {  mid=(low+high)/2;  MergeSort(a,low,mid);  b=mid+1;  MergeSort(a,b,high);  Merge(a,low,high,mid);  }  }  int main()  {  int n,i,x,y;  cout<<"\nEnter number of elements to be sort: ";  cin>>n;  int arr[n];  for(i = 0; i < n; i++)  {  cout<<"Enter element "<<i+1<<": ";  cin>>arr[i];  }  x=(n/2)-1;  y=n/2;  thread t(MergeSort,arr,0,x); //Passing first half of array  thread t2(MergeSort,arr,y,n-1); //Passing second half of array  t.join();  t2.join();  cout<<"\nSorted Data ";  for (i = 0; i < n; i++)  {  cout<<"->"<<arr[i];  }  cout<<"Array sorted."<<endl;  return 0;  } |

**OUTPUT OF CODE:**



**EXPLANATION:**

It checks if there is one element in the list then returns the element. Else, Divide the data recursively into two halves until it can’t be divided further. At last, merge the smaller lists into new lists in sorted order.

**Threads** are the lightweight process which is responsible for executing the part of a task. Threads share common resources to execute the task concurrently. It is an implementation of multitasking where we can run multiple threads on a single processor to execute the tasks concurrently. It subdivides specific operations within a single application into individual threads. Each of the threads can run in parallel.

**OUTPUT OF CORE:**

Table

Description automatically generated with medium confidence

**OUTPUT FOR MAC ADDRESS:**

Text

Description automatically generated

Text

Description automatically generated

**GitHub:**