327107 - Operating Systems

Assignment No. 03

'Multi-Threading.'

Submitted By: Wania Shafqat

Registration No: 200901063

Dept/Batch: BSCS - 01 Section: A

Submitted To: Ma'am Asia Aman

Email ID: waniashafqat02@gmail.com

Date: 28.12.2022

Merge Sort using Multi-Threading

Introduction

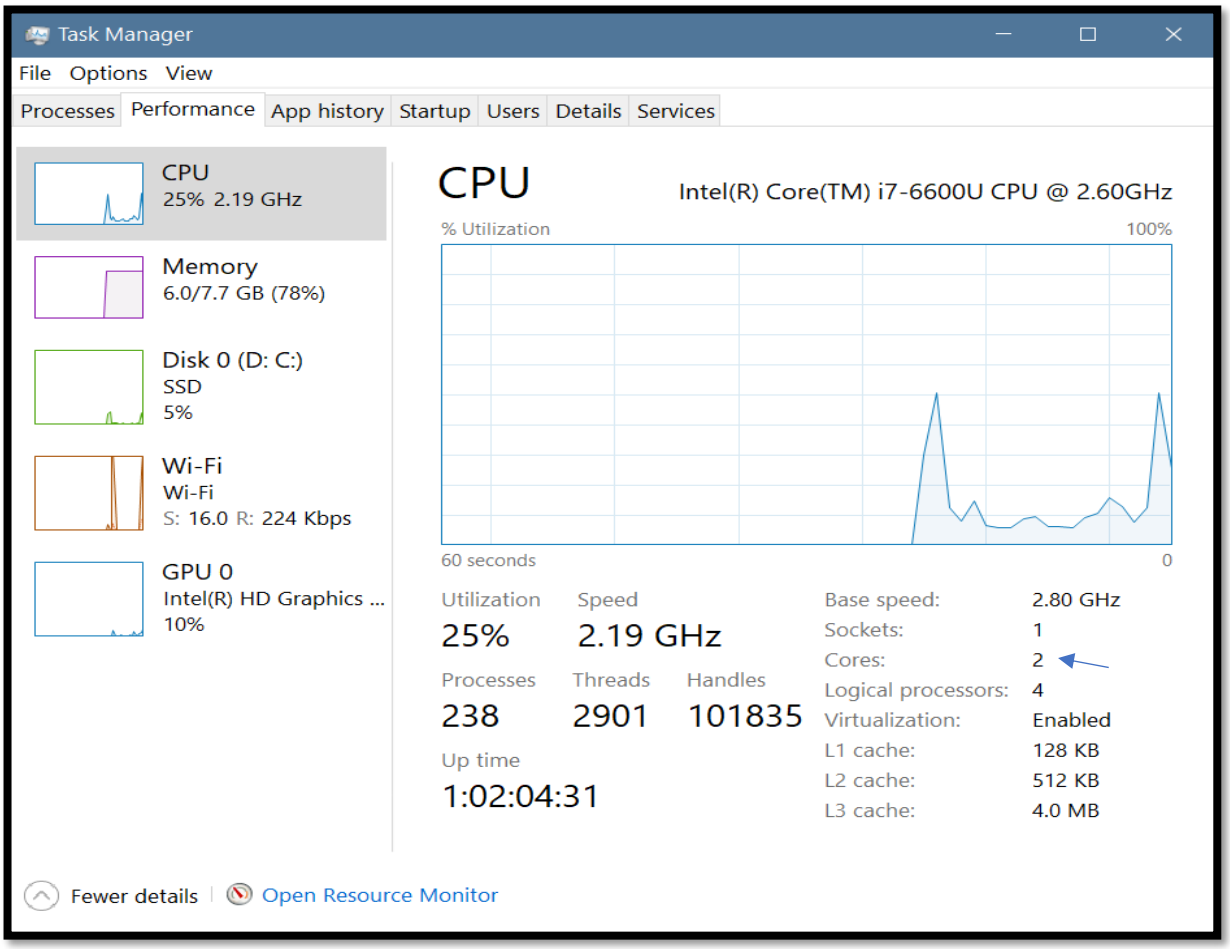
Merge sort performance can be enhanced using multi-threading. For this, check the processor cores of your system, i.e., the system processor has 4 cores. Now create 4 threads and divide the array among these threads and sort them using "*Merge Sort*". You have to take the size and elements of the array from the user.

Explanation

The system processor currently in use has 2 cores. Thus, 2 threads have been first initialized as ***tid1*** and ***tid2*** and then created using the command ***pthread_create.*** The threads are then joined together using the command ***pthread_join.***

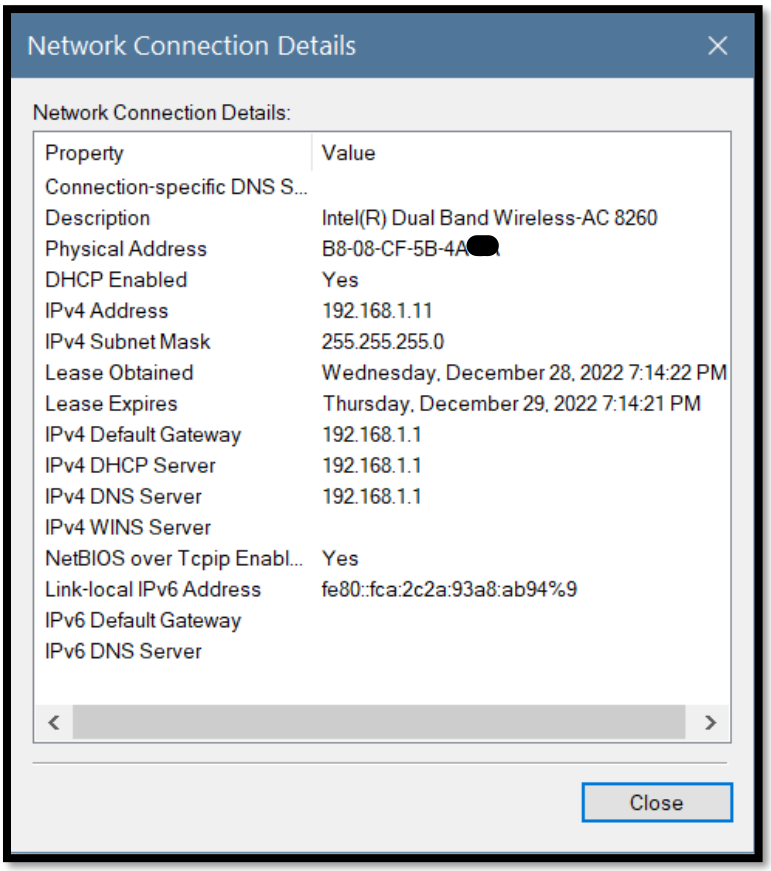
An array has been made, which is according the to size provided by the user. The user is then asked again to enter the elements in the array to be sorted using the '*Divide and Conquer*' technique; the merge sort in ***nlogn*** time. Multi-threading is used, which therefore enhances the performance of the merge sort.

- **System Processor Cores**



- **MAC Address**

For security purposes, I have hidden the last 2 bytes of my MAC address as every device has a unique MAC address and disclosing it can potentially enable hackers to fingerprint the user easily.



C++ Source Code

```
#include <iostream>
#include <pthread.h>
#include <time.h>

#define THREAD_MAX 2
#define MAX 10

using namespace std;

int a[MAX];
int part = 0;

// Function to merge elements.
void merge(int low, int mid, int high)
{
    //int *a;
    int i, j, k, temp[high - low + 1];
    i = low;
    k = 0;
    j = mid + 1;
```

```
21
22          // Merge the two parts into temp[].
23          while (i <= mid && j <= high)
24          {
25              if (a[i] < a[j])
26              {
27                  temp[k] = a[i];
28                  k++;
29                  i++;
30              }
31
32              else
33              {
34                  temp[k] = a[j];
35                  k++;
36                  j++;
37              }
38          }
39
40          // Insert all the remaining values from i to mid into temp[].
41          while (i <= mid)
42          {
```

```
42          {
43              temp[k] = a[i];
44              k++;
45              i++;
46          }
47
48          // Insert all the remaining values from j to high into temp[].
49          while (j <= high)
50          {
51              temp[k] = a[j];
52              k++;
53              j++;
54          }
55
56          // Assign sorted data stored in temp[] to a[].
57          for (i = low; i <= high; i++)
58          {
59              a[i] = temp[i - low];
60          }
61      }
62
63
64      // Function for merge sort.
65      void mergeSort(int low, int high)
66      {
67          int mid;
68          //mid = (low + high) / 2;
69          mid = low + (high - low) / 2;
70
71          if (low < high)
72          {
73              mergeSort(low, mid);
74              mergeSort(mid + 1, high);
75              merge(low, mid, high);
76          }
77
78      }
79
80      // Thread function for multi-threading.
81      void* mergeSort(void* arg)
82      {
83          int thread_part = part++;
```

```
    int thread_part = part++;

    int low = thread_part * (MAX / 2);
    int high = (thread_part + 1) * (MAX / 2) - 1;

    int mid = low + (high - low) / 2;

    if (low < high)
    {
        mergeSort(low, mid);
        mergeSort(mid + 1, high);
        merge(low, mid, high);
    }

    return 0;
}

// Main Code.
int main()
{
    int size, i;

    cout << "Enter the size of array: ";
    cin >> size;

    int arr[size];

    cout << "Enter the elements of array: ";
    for (i = 0; i < size; i++)
    {
        cin >> arr[i];
    }

    // t1 and t2 are used for calculating time.
    clock_t t1, t2;
    t1 = clock();

    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, mergeSort, (void*)NULL);
    pthread_create(&tid2, NULL, mergeSort, (void*)NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);


    // Merging the final parts.
    merge(0, (size / 2 - 1) / 2, size / 2 - 1);
    merge(size / 2, size / 2 + (size - 1 - size / 2) / 2, size - 1);
    merge(0, (size - 1) / 2, size - 1);


    t2 = clock();

    cout << "\nThe sorted array is: ";
    for (i = 0; i < size; i++)
    {
        cout << " " << arr[i];
    }

    // Time taken by merge sort in seconds.
    cout << "\nTime taken: " << (t2 - t1) / (double)CLOCKS_PER_SEC << endl;

    return 0;
}
```

Merge Sort Reference Link

https://www.geeksforgeeks.org/merge-sort/?ref=lbp