OPERATING SYSTEM ASSIGNMENT #03

Particulars:

Name:	Asad Abbas
Reg No:	200901024
Date:	27-12-2022

Code:

```
import threading        # a library used in Python specifically for the use
of multithreading
import time             # library used to measure run time


def merge(arr, l, m, r):          # where 'l' is left, 'm' is middle and 'r'
is right
  # helper function to merge two sorted subarrays
  n1 = m - l + 1
```

```
  n2 = r - m
  L = [0] * (n1)
  R = [0] * (n2)

  for i in range(0 , n1):
    L[i] = arr[l + i]

  for j in range(0 , n2):
    R[j] = arr[m + 1 + j]

  i = 0
  j = 0
  k = l

  while i < n1 and j < n2:
    if L[i] <= R[j]:
      arr[k] = L[i]
      i += 1
    else:
      arr[k] = R[j]
      j += 1
    k += 1

  while i < n1:
    arr[k] = L[i]
    i += 1
    k += 1

  while j < n2:
    arr[k] = R[j]
    j += 1
    k += 1

# The merge_sort function in this code is a implementation of the merge sort
algorithm.
# It takes an array arr and two indices l and r that mark the start and end
of the portion of the array that needs to be sorted.
# Each of these threads is started using the start() method and then joined
using the join() method after they have completed their task.

def merge_sort(arr, l, r):
    # The function first checks if l is less than r.  If it is,
    # then it divides the portion of the array between l and r into two
halves and creates four threads to sort each half concurrently using the
merge_sort function again.
    if l < r:
        m = (l+(r-1))//2
```

```
        t1 = threading.Thread(target=merge_sort, args=(arr, l,
m))           # Creating threads via threading module
        t2 = threading.Thread(target=merge_sort, args=(arr, m+1, r))
        t3 = threading.Thread(target=merge_sort, args=(arr, m+1, r))
        t4 = threading.Thread(target=merge_sort, args=(arr, m+1, r))

        t1.start()
        t2.start()
        t3.start()
        t4.start()

        t1.join()
        t2.join()
        t3.join()
        t4.join()

        merge(arr, l, m, r)    # Calls the merge function, which takes the
sorted subarrays produced by the threads and combines them into a single
sorted array.


# The sort function is used to start the sorting process by calling the
merge_sort function and passing it the entire array to be sorted,
# along with the indices marking the start and end of the array.

def sort(arr):
    l = 0
    r = len(arr) - 1

    # The sort function also starts a timer before the sorting begins and
stops the timer after the sorting is complete, printing the elapsed time.

    # Start the timer
    start_time = time.time()

    merge_sort(arr, l, r)

    # End the timer and print the elapsed time
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Elapsed time:", elapsed_time)

# test the implementation
size = int(input("Enter the size of the array: "))
arr = []

for i in range(size):
    element = int(input("Enter element {}: ".format(i+1)))
```

```
    arr.append(element)

sort(arr)
print("The sorted array becomes: ")
print(arr) # should print the sorted array

# MAC ADDRESS of my system having 4 cores: 10-C3-7B-6C-DC-8A
import uuid

# joins elements of getnode() after each 2 digits.

print ("The MAC address for the current system is : ", end="")
print (':'.join(['{:02x}'.format((uuid.getnode() >> ele) & 0xff)
for ele in range(0,8*6,8)][::-1]))
```

Output:

```
OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\Asad\Desktop\OpenCV>  c:; cd 'c:\Users\Asad\Desktop\OpenCV'; & 'C:\Users\Asad\AppData\L
ms-python.python-2022.20.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '50543' '
y.py'
Enter the size of the array: 5
Enter element 1: 34
Enter element 2: 345
Enter element 3: 3
Enter element 4: 54
Enter element 5: 4
Elapsed time: 0.009993791580200195
The sorted array becomes:
[3, 4, 34, 54, 345]
The MAC address for the current system is : 10:c3:7b:6c:dc:8a
PS C:\Users\Asad\Desktop\OpenCV>
```

Why implement it on Python:

Due to some difficulties faced by me in C++, I chose to implement this in Python in order to gain the core understanding of the task and to avoid unnecessary syntax related complexities of C and C++.

Explanation:

The code provided is a multithreaded implementation of the merge sort algorithm in Python. It has several functions and modules that work together to perform the sorting operation.

The **threading** module is imported to allow the use of multithreading in the code. Multithreading is a way to execute multiple threads concurrently, in the same process. This can be useful in situations where you want to perform multiple tasks simultaneously, such as sorting different parts of an array concurrently.

The **time module** is imported to measure the elapsed time of the sorting operation. This can be used to compare the performance of different sorting algorithms or to optimize the code to make it run faster.

The **merge()** function is a helper function that takes in an array, arr, and indices l, m, and r, where l is the left index, m is the middle index, and r is the right index. It is used to merge two sorted subarrays into a single sorted array.

The **merge_sort()** function is the implementation of the merge sort algorithm. It takes an array, arr, and indices l and r, marking the start and end of the portion of the array that needs to be sorted. If l is less than r, the function divides the portion of the array between l and r into two halves and creates four threads using the threading module's Thread class to sort each half concurrently using the **merge_sort()** function again. The four threads are started using the **start()** method and then joined using the **join()** method after they have completed their task. Finally, the **merge()** function is called to combine the sorted subarrays produced by the threads into a single sorted array.

The **sort()** function is used to start the sorting process by calling the merge_sort() function and passing it the entire array to be sorted, along with

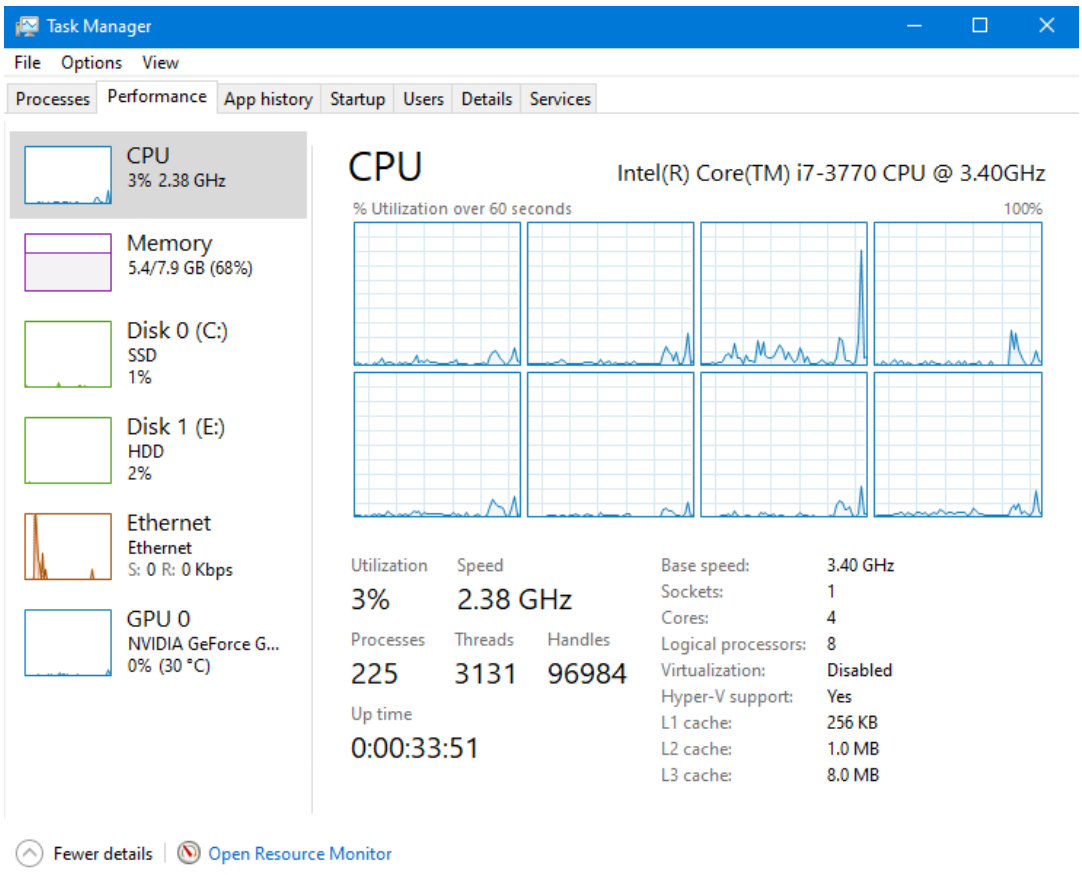
the indices marking the start and end of the array. The **sort()** function also starts a timer before the sorting begins and stops the timer after the sorting is complete, printing the elapsed time.

The code also includes a test of the implementation, where the user is prompted to enter the size of the array and its elements. The **sort()** function is then called on the array, which prints the elapsed time and the sorted array.

Finally, the code imports the **uuid** module and uses it to print the MAC address of the system and the number of cores it has. This information is obtained using the **getnode()** and **cpu_count()** functions of the **uuid** module.

To Find CPU Cores:

As we can see from the image below, my system has 4 cores.



MAC Address:

The MAC address of the system I am using is: 10-C3-7B-6C-DC-8A

Note that I have used an additional Python module which shows the MAC address of whatever system the code is being run from at the end.

GitHub Link:

https://github.com/asadnaqviii/Semester-5--OS---CC