**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES-FAST**

**KARACHI CAMPUS**



**COURSE INSTRUCTOR:**  
**MISS NAUSHEEN SHOAIB & MISS SUMAIYAH**

**OPERATING SYSTEMS**

**PROJECT NAME:**

**COMPARISON BETWEEN MULTIPROCESSING AND MULTITHREADING**

**SECTION: F**

**GROUP MEMBERS:**

**TABAN SHAUKAT (16K-3937)**

**FAREEHA SATTAR SHAIKH (16K-3934)**

**HAFIZ MOHAMMAD HUZAIFAH PUNJANI (16K-3924)**

# OBJECTIVE:

Our project will mainly focus on the differences between multithreading and multiprocessing. The performance comparison will be observed by the implementation of merge sort.

# INTRODUCTION:

We will show the comparison by implementing sorting algorithm i.e**. Merge Sort** in different ways: using multithreading and multiprocessing. We will compare the performance of this sorting algorithm with respect to time, number of inputs and speed in relation with multiple threads and multiple processes.

# PROGRAMMING PLATFORM USED:

* The programming platform used is C Language
* The operating system that we worked on is Ubuntu.

# METHODOLOGY:

For a single thread and a single process, we implemented a single of merge sort.

For multithreading, we have used the library of pthread to create multiple threads and have joined the threads after merging of split parts while applying merge sort, whereas, for process, we have used the ipc and shm library with fork function to create 2 child processes then we called merge sort for both the processes on equal halves of number of elements attaching shared memory so that both processes concurrently work on the same memory space. Their sorted results were then merged and the shared memory was detached.

to split the number of elements in equal halves to make them sorted through merge sort technique with shared memory implemented through ipc and shm library functions. We tested out our codes on the following data and the results are tabulated as follows:

|  |  |  |
| --- | --- | --- |
| Number of elements | Time (in seconds) taken by | |
| 2-Process | 2-Thread |
| 2 | 0.0003x10^-2 | 0.000604 |
| 4 | 0.0005 x10^-2 | 0.000753 |
| 8 | 0.0279 x10^-2 | 0.002328 |
| 16 | 0.0427 x10^-2 | 0.004585 |
| 32 | 0.0251 x10^-2 | 0.000385 |
| 64 | 0.0293 x10^-2 | 0.024131 |
| 128 | 0.0003 x10^-2 | 0.065857 |
| 256 | 0.0483 x10^-2 | 0.243865 |
| 512 | 0.0381 x10^-2 | 0.237705 |
| 1024 | 0.0185 x10^-2 | 0.484888 |

The Data in table above is plotted in the following graphs:

Figure 1

TIME(S)

NUMBER OF ELEMENTS

The Above figure 1 explains a comparison between the Process and threads where Number of Threads and Process are both 2 on the sorting of randomly generated elements. number of elements are increasing in the powers of 2. Here, we have observed that the merge sort with 2-processes is clearly working faster as compared to the merge sort with 2-threads as the number of input is increased.

TIME(S)

Figure 2

Here, in figure 2 we have showed a different kind of comparison. Here, number of elements are kept constant that is 1024 and the number of threads and process are increased to observe the change. We can see that when number of threads and process were one no difference was there. But, when number of process and threads were increased to two, number of threads worked more slowly as compare to two processes for the same number of elements. The speed is achieved by dividing the execution time of the sequential version over the execution

# APPLICATION:

Multithreading and Multi Processing are two ways. Both have their respective usage, but nowadays multithreading is more widely used rather than multiprocessing. Because multiprocessing creates a overhead of PCBs in huge computations.

* Web servers, games like Angry birds, text editors to check errors and web browsers to create multiple tabs uses Multithreading.
* Multiprocessing were used before the threads. It is not widely used nowadays.

# CONCLUSION:

The conclusion which can be drawn from the graphs is that Merge sort is working faster in Multiprocessing as compared to multithreading. The reason is that, multithreading is creating an overhead of synchronization of the threads, secondly as merge sort works on divide and conquer rule so we have to merge all the parts at the end which is time consuming and there are architecture limitations too. Secondly, multithreading works faster on heavy computations.

# REFERENCES:

Following are the relevant references that guided us for our project:

* <http://www.8bitavenue.com/2012/10/difference-between-multiprogramming-multitasking-multithreading-and-multiprocessing/>
* <https://randu.org/tutorials/threads/>
* <https://ubuntuforums.org/showthread.php?t=811144>