Quick Sort Report

By Jashn Arora

2018114006

1. PROBLEM STATEMENT

Implement a Concurrent and Threaded version of Quicksort algorithm and compare the performance of Concurrent, Threaded and Normal Quicksort.

2. SOLUTION EXPLAINED

Randomized Quicksort is used to sort the array in increasing order in all versions.

(a) sort

Runs all the sorts and stores the running time of each version of sort.

(b) threaded_quicksort

(i) Two threads are made recursively, one of which will sort the low subarray and the other will sort the high subarray.

(ii) pthread_create and pthread_join functions are used.

(c) multiprocess_quicksort

(i) Two children are made recursively, one of which will

sort the low subarray and the other will sort the high subarray.

(ii) Shmget and Shmat functions are for accessing the shared memory.

(d) normal_sort

Sorts the elements using normal randomised quicksort.

3. Performance Analysis:

N	Normal Sort	Threaded Sort	Concurrent Sort
200	0.000024	0.009182	0.009810
2000	0.000274	0.089823	0.177128
10000	0.002473	0.327690	1.005230
50000	0.009231	1.545258	6.583130
100000	0.023355	Seg. Fault	10.991516

1) The Worst Case Complexity is taken as array was sorted in decreasing order.

2) Randomized Quicksort with threading takes much more time due to the creation of too many threads. The SegFault at large input therefore occurs due to the creation of too many threads (O(log(n)) number of threads).

3) Similarly, concurrent sort takes more time than normal sort due to the creation of a lot of processes which increase the number of page faults, context switches and CPU migration.

4) Threaded Sort takes less time than Concurrent Sort as threads share memory and caches but for concurrent sort, we are creating two processes which require the os to make virtual memory and Shared Memory region and accessing data in that region takes time.