Quick Sort Report

By Manish

2018101073

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

The program takes N (Number of elements in the array) as input. And then takes N element values. The given array is then sorted using Quick sort and time taken by different methods is printed.

(a) ThreadedQuicksort

(i) Two threads are made recursively, one of which will sort the first half of subarray and the other will sort the second half subarray.
(ii) pthread_create and pthread_join functions are used.

(c) ConcurrentQuicksort

(i) Two child processes are made recursively, one of which will sort the first half of subarray and the other will sort the second half

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

* When the number of elements in the array for a process is less than 5, insertion sort is performed.

3. Performance Analysis:

N	Normal Sort	Threaded Sort	Concurrent Sort
100	0.000007	0.001057	0.001922
1000	0.000072	0.012307	0.023924
10000	0.000844	0.110086	0.400798
50000	0.004950	0.509846	3.291965
100000	0.008423	Seg. Fault	4.412180

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.