**Parallel programming in Java by Fork/Join:**

**Fork/Join in Java**

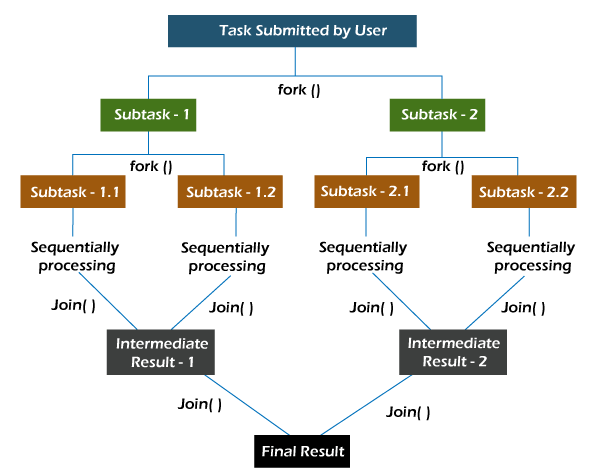
Nowadays, systems are launching with multicore processors.

The multicore processors make the computation faster. Hence, it becomes necessary for a programmer to use multicore processors effectively so that the result can be generated in less span of time.

**Fork/Join in Java** is used to make use of the cores (brain of CPU that process the instructions) in an efficient manner. The fork/join splits a bigger task into smaller sub-tasks. These sub-tasks are then distributed among the cores.

The results of these subtasks are then joined to generate the final result.

The splitting of a task and joining the results imitates the **divide-and-conquer algorithm**. The fork is responsible for splitting the task, and join is responsible for merging the results of the task to generate the final result.



It is worth noting here that various threads that are responsible for the completion of the sub-tasks never sit idle. In fact, they implement the work-stealing algorithm, where an idle thread steals the work from those threads that are busy.

Pseudocode

**if** (sizeOfProblem < threshold)

{

solve the problem directly without doing any split

}

**else**

{

split the problem into smaller subproblems

solve recursively each of the subproblems

 merge the results generated from the subproblems

 }

An important point to remember is that one should not be blindly splitting a problem into sub-problems. **Splitting a problem into sub-problems has its overhead**. If the overhead and the time consumed in solving the sub-problems is greater than solving the problem itself, then one should not be splitting the problem. The limit that makes logical sense to split a problem into subproblem is known as threshold.