



Works based work-stealing algorithm

Why fork and Join

* Problem in executor framework , callable is free to submit a new task to executor and wait for result in synchronous or asynchronous fashion
* When callable waits for result of another callable ,it puts in waiting state and it is queued for execution.
* Java 7 given the concept of parallelism [fork-join framework)
* Added in java.util.concurrent package.
* Task are evenly distributed across the threads in the ForkJoinPool held in doubly-linked queue.

How ?

* Responsible for creating new task and create sub-task object and wating for sub task to get completed.
* Internally it maintains a thread pool and executor assign a pending task to this thread pool to complete when one task is waiting for another task to complete.
* Fork-join framework is leverage multiple processors of advanced machines.
* There ForkJoinTask object has 2 methods
* Fork() - allows ForJoinTask to be launched from existing one
* Join() allows ForkJoinTask to wait completion of another aone
* forkJoinTask object has 2 types
  + RecursiveAction(represent executions do not yield a return value) and RecursiveTask (Specialised form of this instance)
* Fork/Join framework contains thread pool and shared queue where the non-parallel client subkit the task. Each thread has its queue known as deque.

When to use Paraallelism (fork-join framework)

* Tasks are independent of each other and do not need any type of communication or result sharing
* Large amount of data and lot of processing is involved
* Hardware has too many cores and processors

if(task is small) {  
solve the task right away  
} else {  
divide the task into parts  
create new subtask to solve each part   
join all sub-task  
combine resullt from subresults  
}