Thread Pool

What is the Thread Pool in Java?

As the name suggests, the thread pool in Java is actually a pool of Threads. In a simple sense, it contains a group of worker threads that are waiting for the job to be granted. They are reused in the whole process.

In a Thread Pool, a group of fixed size threads is created. Whenever a task has to be granted, one of the threads is pulled out and assigned that task by the service provider, as soon as the job is completed the thread is returned back to the thread pool.

Thread pool is preferably used because active threads consume system resources, when is JVM creates too many threads at the same time, the system could run out of memory. Hence the number of threads to be created has to be limited. Therefore the concept of the thread pool is preferred!

Thread Pool in Java

In Java, a thread pool is created using ExecutorService

```
ExecutorService executorService = Executors.newFixedThreadPool(3)
```

This will create a pool of 3 threads

Using the execute() function, a thread can be requested from the executorService

```
class TaskThread implements Runnable {
   public void run() {
      // doWork()
   }
}
Runnable taskThread = new TaskThread()
executorService.execute(taskThread)
```

Risk in the Java Thread Pool

There are a few risks while you are dealing with the thread pool, like;

- **Thread Leakage:** If a thread is removed from the pool to perform a task but not returned back to it when the task is completed, thread leakage occurs.
- **Deadlock:** In thread pool is executing thread is waiting for the output from the block the thread waiting in the queue due to unavailability of thread for execution, there's a case of a deadlock.
- **Resource Thrashing:** More number of threads than the optimal number required can cause starvation problems leading to resource thrashing.

Disadvantages of the Thread Pool

Some of the disadvantages of using thread pool when programming are:

- There is no control over the priority and state of the thread you are working with.
- When there is a high demand for the thread pool, the process may be deleted.
- The thread pool can not work well when two threads are working in parallel.
- There are several situations where the application code can be affected by another application code, despite robust application isolation.

Formula to calculate Thread Pool Size:

$Numberofthreads = NumberofAvailableCores * (1 + Waittime/Servicetime)$

Wait Time: The total time spent when no processing is done, for example I/O operations, network calls Service Time: Actual processing time

For example, in a quad-core CPU, if a microservice spends 500ms on spent on I/O operations, and 50 ms on actual compute,

$numberOfhreads = 4 * (1 + 500/50) = 44$