package com.twitter.servo.forked

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

import com.twitter.servo.util.ExceptionCounter

import com.twitter.util.{Duration, Time, Local, TimeoutException}

import java.util.concurrent.{LinkedBlockingQueue, TimeUnit, CountDownLatch}

/\*\*

\* A forking action executor that executes the actions in a separate

\* thread, using a bounded queue as the communication channel. If the

\* queue is full (the secondary thread is slow to drain it), then the

\* items will be dropped rather than enqueued.

\*/

class QueueExecutor(maxQueueSize: Int, stats: StatsReceiver) extends Forked.Executor {

private val forkExceptionsCounter = new ExceptionCounter(stats)

private val enqueuedCounter = stats.counter("forked\_actions\_enqueued")

private val droppedCounter = stats.counter("forked\_actions\_dropped")

private val log = Logger.get("Forked.QueueExecutor")

@volatile private var isStopped = false

private val releaseCountDownLatch = new CountDownLatch(1)

private val queue = new LinkedBlockingQueue[() => Unit](maxQueueSize)

private val thread = new Thread {

override def run(): Unit = {

while (!isStopped) {

try {

queue.take()()

} catch {

// Ignore interrupts from other threads

case \_: InterruptedException =>

// TODO: handle fatal errors more seriously

case e: Throwable =>

forkExceptionsCounter(e)

log.error(e, "Executing queued action")

}

}

releaseCountDownLatch.countDown()

}

}

thread.setDaemon(true)

thread.start()

/\*\*

\* Interrupts the thread and directs it to stop processing. This

\* method will not return until the processing thread has finished

\* or the timeout occurs. Ok to call multiple times.

\*/

def release(timeout: Duration): Unit = {

if (!isStopped) {

isStopped = true

thread.interrupt()

releaseCountDownLatch.await(timeout.inMilliseconds, TimeUnit.MILLISECONDS) || {

throw new TimeoutException(timeout.toString)

}

}

}

/\*\*

\* Blocks until all the items currently in the queue have been

\* executed, or the timeout occurs. Mostly useful during testing.

\*/

def waitForQueueToDrain(timeout: Duration): Unit = {

val latch = new CountDownLatch(1)

val start = Time.now

queue.offer(() => latch.countDown(), timeout.inMilliseconds, TimeUnit.MILLISECONDS)

val remaining = timeout - (Time.now - start)

latch.await(remaining.inMilliseconds, TimeUnit.MILLISECONDS) || {

throw new TimeoutException(remaining.toString)

}

}

/\*\*

\* Queue the action for execution in this object's thread.

\*/

def apply(action: () => Unit) =

if (queue.offer(Local.closed(action)))

enqueuedCounter.incr()

else

droppedCounter.incr()

}