COMPUTACIÓN CONCURRENTE
PRÁCTICA 3, SPIN LOCKS

Prof. Manuel Alcántara Juárez
manuelalcantara52@ciencias.unam.mx

Leonardo Hernández Cano
leonardohernandezcano@ciencias.unam.mx

Ricchy Alain Pérez Chevanier
alain.chevanier@ciencias.unam.mx

Fecha Límite de Entrega: 07 de Octubre de 2019 a las 23:59:59pm.

1. Objetivo

El objetivo de esta práctica es realizar un análisis de los tiempos de ejecución obtenidos al utilizar diferentes implementaciones de *spin locks* en la solución a un problema que utiliza *locks*.

2. Desarrollo

En esta práctica trabajarás con una base de código construida con Java 8[1] y Maven 3, también proveemos pruebas unitarias escritas con la biblioteca `Junit 5.5.1` que te darán retrospectiva inmediata sobre el correcto funcionamiento de tu implementación[2] .
Para ejecutar las pruebas unitarias necesitas ejecutar el siguiente comando:

```
$ mvn test
```

Para ejecutar las pruebas unitarias contenidas en una única clase de pruebas, utiliza un comando como el siguiente:

[1]De nuevo puedes utilizar cualquier versión de java que sea mayor o igual a Java 8 simplemente ajustando el archivo pom.xml

[2]Bajo los casos que vamos a evaluar, mas estas no aseguran que tu es implementación es correcta con rigor formal

```
$ mvn -Dtest=LockTest test
```

En el código que recibirás la clase `App` tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y finalmente ejecutar el jar generado. Utiliza un comando como el que sigue:

```
$ mvn package
...
...
$ java -jar target/practica03.jar
```

3. Actividades

3.1. Implementación de Locks

Completa la implementación de las clases que derivan de la interfaz `Lock`.

- `TASLock`
- `TTASLock`
- `BackoffLock`
- `CLHLock`
- `MCSLock`

Estas implementaciones de `Lock` fueron cubiertas en clase, pero también es posible revisar el código de estas en el capítulo 7 "Spin Locks"del libro "The Art of Multiprocessor Programming"[3].

Para implementar la clase `BackoffLock` prueba con algunos valores arbitrarios pero pequeños, y ejecuta las pruebas unitarias para ver qué valores te proporcionan un menor tiempo de ejecución en promedio.

Para verificar que tus implementaciones funcionan necesitas pasar las pruebas unitarias contenidas en la clase `LockTest`, hay una prueba por cada implementación de `Lock`.

[3]Aquí tambien encontrarás la descripción de cada implementación en donde explica en que contexto/arquitectura es buena cada una de ellas.

3.2. Análisis

Las pruebas unitarias de la implementación que realizaste en la actividad anterior, estresan la implementacion de cada `Lock` utilizando un contador que se debe de incrementar de manera atómica, la elección de este problema como parte de las pruebas tiene como razón el hecho de que un contador compartido con una *sección crítica* muy pequeña tiene alta *contención*. Una vez que pases todas las pruebas unitarias, estas te proporcionarán el tiempo de ejecución de cada una de tus implementaciones. Por omisión estas pruebas utilizan solamente 3 *threads* y cada *thread* realiza 10 veces el mismo procedimiento.

Ejecuta todas las pruebas unitarias y registra los tiempos de ejecución obtenidos para los valores de `THREADS` $[3, 7, 13, 23, 29]$, divide cada tiempo de ejecución entre 10 y registra lo que tomó cada ejecución en cada caso, crea una tabla con esta información y también crea una gráfica, ambas comparan el tiempo de ejecución en milisegundos versus el número de *threads* que fueron utilizados.

Incluye esta tabla y la gráfica en un documento, y provee las concluciones de por qué algunas implementaciones de `Lock` son más veloces o más lentas en tu computadora. Con esta información trata de describir cómo funciona la arquitectura de nucleos y caches en tu computadora.

4. Evaluación

Tu entrega debe de incluir el código de la implementación de cada una de las variantes de `Lock`, así como también un documento con el *análisis* que realizaste, siendo el análisis la parte más importante de las dos actividades.