Programación Concurrente

Trabajo Práctico

Se desea implementar en Java usando métodos *synchronized* una clase "monitor" que encapsule el comportamiento de un vector de números de punto flotante. Para esto, el enunciado viene acompañado de una clase `SeqVector`. Se debe proveer una clase `ConcurVector` que implemente las mismas operaciones, pero donde su funcionamiento se resuelva de manera concurrente. Para esto durante la creación de un `ConcurVector` se deben tomar dos parámetros estrictamente mayores a cero:

1. `dimension`, que indica la cantidad de elementos que puede almacenar el vector.
2. `threads`, que indica la cantidad máxima de threads a utilizar para realizar las operaciones concurrentemente.

El mecanismo por el cual los threads toman y procesan nuevo trabajo debe ser llevado a cabo por un "thread pool" con la siguiente estructura:

1. Una clase `Buffer` (implementada como un monitor utilizando métodos synchronized) que actúa como una cola FIFO concurrente de capacidad acotada. Es decir, bloquea a un consumidor intentando sacar un elemento cuando está vacía y bloquea a un productor intentando agregar un elemento cuando está llena. La capacidad del Buffer debe ser un parámetro configurable.
2. Una clase `Worker` que extiende de `Thread` y realiza la operación deseada. Un `Worker` debe tomar una cantidad de elementos para trabajar de un `Buffer` conocido al momento de su creación. Si estos elementos son inválidos el `Worker` debe preparase para finalizar su ejecución.
3. Una clase `ThreadPool`, que se encarga de instanciar e iniciar la cantidad de `Workers` correspondiente a los valores de los parámetros `threads` y `load`.
4. Cualquier otra clase auxiliar que considere necesaria.

La repartición de trabajo debe realizarse de manera justa: No deberían generarse tareas que tengan más de dos celdas de diferencia (Es decir, si se requiere procesar un `ConcurVector` de 10 posiciones con dos threads, no es correcto generar una tarea para 2 posiciones y una para 8). Tomar en cuenta que mientras los métodos como `norm` sólo requieren hacer una única pasada por el vector, aquellos que son de agregación, como `sum` o `max`, requerirán de procesamiento de valores intermedios adicionales, y en estos casos la distribución de la carga también tiene que ser lo más justa posible (Si 10 threads trabajan sobre un vector de 200 posiciones y cada uno genera la suma parcial para 20 valores, estas sumas parciales también deben ser repartidas para su combinación de manera equitativa). Sólo en el caso en el que el trabajo a repartir es demasiado pequeño para distribuirlo a todos los threads existentes se podrá terminar generando menos unidades de trabajo que la cantidad de threads.

Pautas de Entrega

- El TP se hace en grupos de a lo sumo dos personas (Salvo por un grupo que potencialmente puede ser 3 personas). Es posible realizar el trabajo de manera invidual, aunque no es recomendable.
- Se debe hacer entrega del código Java que resuelva el enunciado (con todas las clases y paquetes utilizados). Si por alguna razón el armado del entorno requiere hacer algo más que sólo la importación, en la entrega deben estar listadas las instrucciones necesarias para lograrlo.
- La entrega se debe hacer por email a las direcciones de los docentes con el asunto `TP-PCONC-2018S1-Apellido1-Apellido2` (Donde `Apellido1` y `Apellido2` son los apellidos de los integrantes del grupo), y se debe adjuntar el código en un archivo `.zip` con nombre:

  `tp-pconc-2018s1-apellido1-apellido2.zip`
- Se reciben TPs para la primera entrega hasta el 24/10/2018 a las 23:59.