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 concurrentemente. Para esto durante la creación de un `ConcurVector` se deben tomar los siguientes parámetros (todos 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
3. `load`, que indica la cantidad de elementos en la que puede diferir la asignación a cada thread

Por ejemplo, un vector de `dimension` 11, con 5 `threads` y `load` 1 realizando la operación `abs` puede distribuir la carga entre los 5 threads trabajando con 2 elementos en 4 threads y 3 elementos en el thread restante. Por el contrario si el `load` fuera 11, todos los elementos podrían ser procesados en un sólo thread.

Tenga en cuenta que las operaciones que no dependen del resto de los valores son simples de distribuir, pero las operaciones con dependencias (como `sum`) son más complejas. Considerando que estas operaciones son asociativas es posible hacer la distribución siempre y cuando el resultado de cada procesamiento luego sea unido correctamente. La operación de unión de los resultados parciales producto de la división está sujeta a las mismas restricciones que el vector (`threads` y `load`), por lo que múltiples iteraciones de distribución y unión pueden ser necesarias.

Por ejemplo, un vector de `dimension` 8, con 4 `threads` y `load` 2 realizando la operación `sum`, realiza una primera iteración distribuyendo la carga entre los 4 threads, tabajando con 2 elementos en cada thread. Esto produce 4 resultados parciales, que para ser unidos deben ser nuevamente procesados de a 2 elementos en 2 threads. Finalmente con los últimos 2 resultados parciales pueden ser asignados a un sólo thread para concluir el proceso de sumatoria y retornar el resultado.

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.

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.