Arquitecturas Avanzadas

Grupo AA-2-1

25/11/2019

Julio César Velasquez Cardenas 1397896
Sergio Prada Maeso 1459122
Juan Carlos Bermúdez Rodríguez 1455486

Para la primera versión del programa, añadimos threads al código y ponemos a cada uno a trabajar en la ordenación de la lista sin preocuparnos por las regiones críticas.
El problema aquí radica en que aunque el tiempo que obtenemos es acorde a la ejecución que hacemos (con dos threads obtenemos un speed up de casi x2), el resultado obtenido cambia y pasa a ser incorrecto. Esto se debe a que mientras un thread lee o escribe en una posición (utilizando los punteros), el otro puede haber hecho (a mitad de la lectura/escritura del otro thread) , una inserción o una modificación de los punteros dando lugar a búsquedas o a inserciones que se ejecutan de manera incorrecta; lo que se traduce en que la lectura/escritura del otro thread sea incorrecta ya que puede crear la falsa idea al otro thread de que la posición actual es la correcta y no tiene en cuenta la nueva versión del nodo que implementa el otro thread.

Para la segunda versión del programa, añadimos una sección crítica al código para que al hacer la escritura/lectura de las posiciones, no haya interacciones negativas que hagan que un nodo se pierda o se cambie la posición de este mientras el otro thread lo lea.
Para controlarlo, hacemos que esta sección sea ejecutada por un thread a la vez, eliminando la paralelización del código (hacemos un código casi secuencial) pero hacemos que el resultado de la ejecución sea correcta.

Para la tercera versión del programa, hemos de considerar que la sección crítica para limitar la ejecución de los threads de la forma que se hace en el segundo ejercicio, genera un problema de ejecución secuencial del programa, ya que limita que la ejecución de los threads sea secuencial.
Para resolverlo, hemos de encontrar una manera de no limitar la sección del código entera, sino las posiciones las cuales cada thread mire/escriba a cada vez. Para ello, cada vez que un thread lea o escriba en un nodo, impide que el otro lo haga y se espere para que cuando vaya a leer o escribir, dicho thread tenga en consideración su inserción.
De esta manera no obligamos a que el código se ejecute en secuencial, sinó que solamente cierta parte del código crítico lo haga. (ello hará que un thread pueda insertar en la posición 7 mientras que otro lo hace en la 2 sin que uno limite a otro).
Sin embargo, tendrá en consideración la situación en la que un thread intente leer/escribir en la posición 3 mientras el otro está modificando dicha posición, haciendo que este espere a hacer la lectura/escritura hasta que el otro thread no acabe. Para esto mismo utilizamos los locks, que bloquean el acceso a la variable o variables para el resto de threads hasta que se libere el candado, que previamente se ha cerrado para este efecto. Una vez el thread que cerró el candado lo vuelva a abrir, aquel thread que primero acceda a la variable una vez se ha abierto el candado lo volverá a cerrar, permitiendo así un acceso seguro a los punteros que controlan la búsqueda e inserción de elementos en la lista, haciéndose correctamente a diferencia de en la primera versión.