Joan Gracia Taribó - 219176
Josep Tuñí Domínguez - 216737

Operative Systems: Laboratory 3
Threaded Programming

· Threaded Mandelbrot

Vamos a crear R^2 threads, y cada thread se encargará de crear cada uno de los cuadrados en los que se divide la imagen, es decir, tendremos un thread por cuadrado. Para esto, tenemos dos funciones:

1. setPixel: Esta función recibe las coordenadas cartesianas (x, y) de ese píxel y la posición en memoria del primer píxel. En este píxel ponemos el valor azul, en el píxel+1 ponemos el valor verde y en el píxel+2 ponemos el valor rojo. Mediante las posiciones (x, y) calculamos el valor del Mandelbrot de esa iteración.

2. fillSquare: Esta función recibe un struct (Ya que lo llamamos con los threads), este struct contiene la posicón en memoria del primer píxel del cuadrado, la posición del vértice arriba a la izquierda del píxel y el tamaño de un lado del cuadrado. Iremos recorriendo pixel a pixel nuestro cuadrado y llamaremos a setPixel para dibujar en pantalla ese píxel, la posicón en memoria del pixel la vamos actualizando en cada iteración.

La ejecución de generar el dibujo Mandelbrot sin threads, el tiempo medio de 300ms. Con threads, el tiempo se reduce a 119ms. Claramente podemos ver que el tiempo usando threads es mucho menor, esto es debido a que nuestro programa se puede ejecutar de forma paralela gracias al uso de threads.

· Scrambled Mandelbrot

1. Esta ejecución con threads, tiene un tiempo medio de 35ms, pero tiene un grave problema. Al hacer esta ejecución con threads que acceden a las mismas zonas de memoria, tenemos un grave problema de Race Condition. Este problema causa que nuestra imagen no sea como debería:

2. Gracias al uso de locks, no tenemos ningún problema de Race Condition, ya que cuando un thread va a acceder a una zona de la imagen, se hace un lock que no permite que nadie acceda a esa imagen. Esto tiene un problema, que es el hecho de que nuestro programa es más lento, ahora tiene un tiempo medio de 60ms, ya que nuestros threads siempre tienen que esperar a que nadie esté usando la imagen. Esta espera no es necesaria en algunos casos, ya que la mayoría de veces un Thread1 estará trabajando sobre una zona de la imagen, distinta a la que quiere trabajar el Thread2. Así que Thread1 y Thread2 podrian trabajar paralelamente y no tendrían Race Condition.

3. Ahora, vamos a solucionar el problema anterior, haciendo que cada thread haga un lock solo de la zona sobre la que trabaja, de esta forma, un thread solo esperará cuando sea estrictamente necesario. Nuestro tiempo medio, es de 30ms, es decir, es el doble de rápido que el anterior.
Para hacer lock de algunas zonas, hemos implementado una matriz de locks. Cada celda de nuestra matriz controla la situación de ese cuadrado de la imagen.

4. Finalmente, hacemos uso de los monitores. La solución será muy parecida a la anterior, pero en este caso, nuestra matriz no estará formada por locks, estará formada por monitores. Nuestro tiempo medio es 40ms, que es superior al del caso 3 pero inferior al del caso 2.