Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final Teórico de la Asignatura
Junio de 2014

Apellidos: Nombre:

D.N.I.: Grupo (A ó B):

1. Notas

1. Escriba su nombre, apellidos, D.N.I. y grupo en el espacio habilitado para ello, y en todos los folios blancos que utilice. Firme el documento en la esquina superior derecha de la primera página.

2. Dispone de diez minutos para leer los enunciados y formular preguntas o aclaraciones sobre ellos. Transcurrido ese tiempo, no se contestarán preguntas. Dispone de 2:00 horas para completar el ejercicio.

3. No complete el documento a lápiz. Utilice bolígrafo o rotulador. Escriba con letra clara y legible. No podemos corregir lo que no se no puede leer.

4. Utilice los folios blancos que se le proporcionan para resolver los enunciados, pero traslade a este documento únicamente la solución final que obtenga, utilizando el espacio específicamente habilitado para ello, sin sobrepasarlo en ningún caso, y sin proporcionar información o respuestas no pedidas. Entregue tanto el enunciado como los folios blancos. Únicamente se corregirá este documento.

2. Criterios de Corrección

1. El examen se calificará de cero a diez puntos, y ponderará en la calificación final al 40 % bajo los supuestos recogidos en la ficha de la asignatura.

2. Cada enunciado incluye información de la puntuación que su resolución correcta representa, incluida entre corchetes.

3. Un enunciado (cuestión teórica o problema) se considera correcto si la solución dada es correcta completamente. En cualquier otro caso se considera incorrecto y no puntúa.

4. Un enunciado de múltiples apartados (cuestión teórica o problema) es correcto si y solo si todos los apartados que lo forman se contestan correctamente. En cualquier otro caso se considera incorrecto y no puntúa.

3. Cuestiones de Desarrollo Corto

Conteste a las preguntas que se le formulan en el espacio habilitado para ello. Deberá razonar o justificar su respuesta simpre que se le indique. La ausencia del razonamiento o de la justificación invalidará la respuesta al no ser esta completa.

1. En qué se diferencian las variables de condición de un monitor *Hoare* del *wait-set* asociado a los objetos Java?[1.0 puntos]

2. ¿Desarrolle en formato tabular una comparativa entre las diferencias arquitectónicas entre procesadores *multicore* y *manycore*. Escriba aquí la tabla y explíquela: [0.5 puntos]

3. ¿Es posible implantar monitores C++11 como primitiva de control de la concurrencia equivalentes a los monitores *Hoare*? Escriba y justifique su respuesta.[1 punto]

4. ¿Qué sucede cuando una CPU lanza un *kernel* sobre una GPU *nVidia* que actúa como coprocesador? Escriba aquí su respuesta razonada: [0.5 puntos]

5. Considere el siguiente programa. Indique la salida -si la hay- que produce y el comportamiento que tiene. Justifique su respuesta. [1 punto]

```
import java.util.concurrent.*;

class Task implements Runnable{
  public static int cont = 0;
  private boolean tHilo;

  public Task(boolean tHilo){this.tHilo=tHilo;}
  public void run(){
    if(tHilo)
      for(int i=0; i<1000000; i++) cont++;
      else for(int i=0; i<1000000; i++) cont--;
```

```
  }
}

public class E131406{
  public static void main(String[] args) throws Exception{
    ThreadPoolExecutor ejecutor = new ThreadPoolExecutor(
                                  1, 1, 60000L,
                                  TimeUnit.MILLISECONDS,
                                  new LinkedBlockingQueue<Runnable>());
    ejecutor.execute(new Task(true));
    ejecutor.execute(new Task(false));
    ejecutor.shutdown();
    while(!ejecutor.isTerminated()){}
    System.out.println(Task.cont);
  }
}
```

Indique aquí la salida y el comportamiento, justificando ambos:

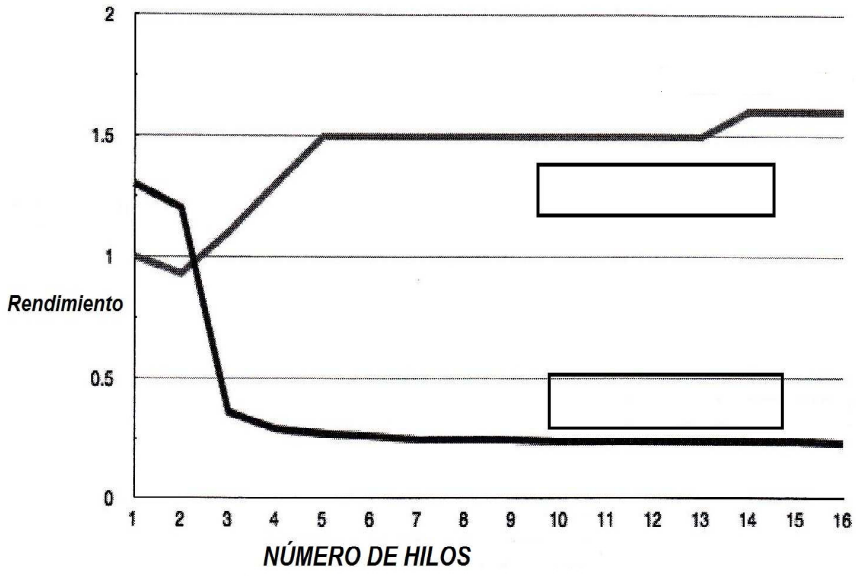
6. Considere el siguiente programa. Indique la salida -si la hay- que produce y el comportamiento que obtiene. Justifique su respuesta. [1 punto]

```
public class E131408 extends Thread
{
    Integer I;
    public E131408(Integer I) {this.I=I;}
    public void run(){
      System.out.println(this.getName());
      synchronized(I){
        try{I.wait();}catch (InterruptedException e){}
        System.out.println(this.getName()+" diCE: Hola...");}
    }

   public void m1(){synchronized (I){I.notify();}}
   public void m2(){synchronized (I){I.notifyAll();}}

   public static void main(String[] args) throws InterruptedException{
    Integer I = new Integer(0);
    E131408 [] AliBaba = new E131408[10];
    for(int i=0; i<10;i++){
```

```
      AliBaba[i]=new E131408(I); AliBaba[i].start();}
    AliBaba[5].m1();
    Thread SpeedyGonzalez = currentThread();
    SpeedyGonzalez.sleep(2000);
    AliBaba[5].m2();
    System.out.print("Todos terminaron...");
   }
}
```

Indique aquí la salida y el comportamiento, justificando ambos:

7. La siguiente curva ilustra el comportamiento en términos de rendimiento de una aplicación paralela donde las diferentes tareas utilizan contenedores de datos sincronizados para realizar los intercambios de información que necesitan. La misma versión del código se ha ejecutado con una instancia de la clase `HashMap` sincronizada de forma estándar (`synchronized`) y con una instancia de la clase `ConcurrentHashMap`, para un procesador multinúcleo de 8 cores.



Complete la figura rellenando los recuadros con `HashMap` o `ConcurrentHashMap` indicando para cada curva, a qué tipo de contenedor corresponde. Realice una

intepretación de forma justificada. [1 punto]

8. Considere el siguiente programa escrito en C++11. Indique la salida -si la hay- que produce y el comportamiento que tiene. Justifique su respuesta. [0.5 puntos]

```
#include <iostream>
#include <thread>
#include <vector>
#include <atomic>
using namespace std;
struct CuentaAtomica{
  int val;
  void inc(){
    ++val;
  }
  int verValor(){return(val);}
};

int main(){
  vector<thread> hilos;
  int nHilos = 100;
  CuentaAtomica contador;
  for(int i=0; i<nHilos; ++i){
    hilos.push_back(thread([&contador](){for(int i=0;i<1000;i++){contador.inc();}}));
  }
  for(auto& thread : hilos){thread.join();}
  cout << contador.verValor();
  return(0);
}
```

Indique aquí la salida y el comportamiento, justificando ambos:

4. Problemas

1. El conjunto de Mandelbrot es el más conocido de los conjuntos fractales y el más estudiado. Este conjunto se define en el plano complejo como sigue: sea c un número complejo cualquiera. A partir de c, se construye una sucesión por recursión dada por las siguientes ecuaciones [2 puntos]:

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

Si la sucesión está acotada, c pertenece al conjunto de Mandelbrot, pero no si diverge. Se sabe que los puntos cuya distancia al origen es superior a 2, es decir, $x^2 + y^2 = 4$ no pertenecen al conjunto. Por lo tanto basta encontrar un solo término de la sucesión que verifique $|zn| > 2$ para estar seguros de que c no está en el conjunto. Este hecho se utiliza para escribir un algoritmo sencillo que permite generar y pintar puntos del conjunto. Su seudocódigo se muestra a continuación:

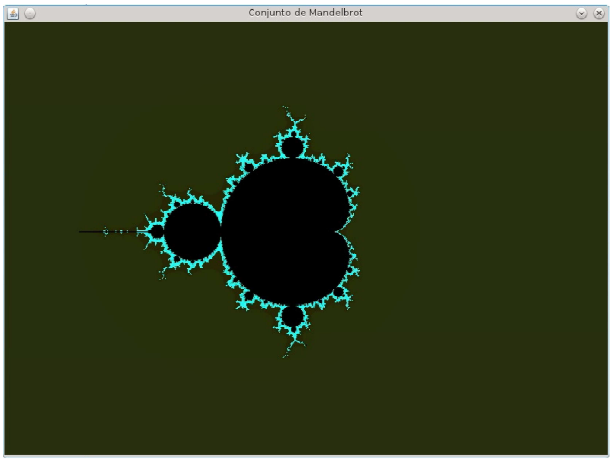
```
Para cada pixel (Px, Py) do:
{
  x0 = scaled x coordinate of pixel (Re part)
  y0 = scaled y coordinate of pixel (Im part)
  x = 0.0
  y = 0.0
  iteration = 0
  max_iteration = 100000
  while ( x*x + y*y < 4  AND  iteration < max_iteration )
  {
    xtemp = x*x - y*y + x0
    y = 2*x*y + y0
    x = xtemp
    iteration = iteration + 1
  }
  color = palette[iteration]
  plot(Px, Py, color)
}
```

Del cuál se deriva de forma directa el código Java siguiente:

```
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import javax.swing.JFrame;

public class Mandelbrot extends JFrame {

    private final int MAX_ITER = 100000;
    private final double ZOOM = 150;
    private BufferedImage Imagen;
```

```
    private double zx, zy, cX, cY, tmp;

    public Mandelbrot() {
        super("Conjunto de Mandelbrot");
        setBounds(100, 100, 800, 600);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Imagen = new BufferedImage(getWidth(), getHeight(),
                    BufferedImage.TYPE_INT_RGB);
        //aqui comienza la rutina a paralelizar
        for (int y = 0; y < getHeight(); y++) {
            for (int x = 0; x < getWidth(); x++) {
                zx = zy = 0;
                cX = (x - 400) / ZOOM;
                cY = (y - 300) / ZOOM;
                int iter = MAX_ITER;
                while (zx * zx + zy * zy < 4 && iter > 0) {
                    tmp = zx * zx - zy * zy + cX;
                    zy = 2.0 * zx * zy + cY;
                    zx = tmp;
                    iter--;
                }
                Imagen.setRGB(x, y, iter | (iter << 8));
            }
        }//aqui finaliza la rutina a paralelizar
    }

    public void paint(Graphics g) {
        g.drawImage(Imagen, 0, 0, this);
    }

    public static void main(String[] args) {
        new Mandelbrot().setVisible(true);
    }
}
```

El programa anterior permite obtener imágenes del conjunto como la de la Figura mediante un hilo único de ejecución secuencial:



Se desea escribir un programa paralelo `parMandelbrot.java` para resolver el problema en función del número de núcleos disponibles y del coeficiente de bloqueo que considere adecuado fijar, considerando tareas `Runnable` y utilizando un ejecutor `FixedThreadPool` para procesarlas. El programa debe fraccionar el trabajo y repartirlo entre los núcleos de forma automática.

(siga escribiendo aquí su programa)

2. El antiguo aeródromo de *Tempelhof* (Berlín) dispone de una pista para aterrizaje y tres para despegue (todo lo cuál es una exageración propia de un problema de examen). Cuando un avión desea despegar, debe pedir pista a la torre de control, que le indica cuál de ellas le corresponde. Una vez que ha despegado, el avión indica a la torre que ha dejado libre la pista. Si no hay pistas libres, el avión debe esperar para poder efectuar es despegue. El aterrizaje funciona de la misma manera, pero con una única pista para ello. Escriba un monitor que simule a la torre de control en `Torre.java`, utilizando cerrojos de clase `ReentrantLock` y variables `Condition`. Modele los aviones mediante hebras en una clase `Avion.java` por herencia de la clase `Thread`, suponiendo pilotos novatos que despegan, vuelan un rato y aterrizan. Ponga todo a funcionar en un programa ejecutable `Tempelhof.java`. [1.5 puntos]