Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Examen Final de Prácticas
Septiembre de 2013

Apellidos: Nombre:

Grupo (A ó B):

1. Notas

1. Escriba su nombre y apellidos con letra clara y legible en el espacio habilitado para ello.
2. Firme el documento en la esquina superior derecha de la primera hoja y escriba debajo su D.N.I.
3. Dispone de $2:30'$ horas para completar el ejercicio.
4. Puede utilizar el material bibliográfico (libros) y copia de API que estime convenientes. Se prohíbe el uso de apuntes, *pen-drives* y similares.
5. Entregue sus productos, utilizando la tarea de entrega disponible en el Campus Virtual, en un fichero (`.rar, .zip`) de nombre

   `Apellido1_Apellido_2_Nombre`

   que contendrá una subcarperta por cada enunciado del examen, la cual contendrá a su vez el conjunto de ficheros que den solución a ese enunciado en particular.

2. Criterios de Corrección

1. El examen práctico se calificará de cero a diez puntos y ponderará en la calificación final al 50 % bajos los supuestos recogidos en la ficha de la asignatura.
2. Las condiciones que una solución a un enunciado de examen práctico debe cumplir para considerarse correcta son:

a) Los ficheros subidos a través del Campus Virtual que conforman el examen práctico se ajustan al número, formato y nomenclatura de nombres explicitados por el profesor en el documento de examen.

b) El contenido de los ficheros es el especificado por el profesor en el documento de examen en orden a solucionar el enunciado en cuestión.

c) Los programas elaborados por el alumno, se pueden abrir y procesar con el compilador del lenguaje, y realizan un procesamiento técnicamente correcto, según el enunciado de que se trate.

d) Se entiende por un procesamiento técnicamente correcto a aquél código de programa que compila correctamente sin errores, cuya semántica dá soporte a la solución pedida, y que ha sido escrito de acuerdo a las convenciones de estilo y eficiencia habituales en programación

3. Enunciados

1. (Suavizado de Imagen, 4.0 puntos) Una de las operaciones más habituales en el tratamiento de imágenes digitales es el suavizado, que puede conseguirse de muy diversas maneras. Suponga que está representando su imagen mediante una matriz bidimensional de 500×500 píxels, con una escala de 20 niveles de gris, desde 0 para el color negro hasta 1 para el color blanco. Se entiende por suavizado de la imagen la aplicación de la siguiente transformación a todos los píxels que la forman:

$$x_{i,j} = (4 \cdot x_{i,j} + x_{i+1,j} + x_{i,j+1} + x_{i-1,j} + x_{i,j-1})/8$$

Se pide escribir un programa en Java que realice el suavizado de forma paralela de acuerda a las especificaciones siguientes:

- La imagen original se determinará de forma aleatoria.
- El programa leerá el número de núcleos de que dispone la máquina y ajustará el número de tareas paralelas en función del mismo y del coeficiente de bloqueo del problema.
- El usuario dispondrá de un corto menú de dos opciones: una para generar la imagen aleatoriamente, y otra que permitirá determinar el tamaño de la imagen, introducir de forma manual los píxels que la forman, efectuar el suavizado e imprimir en pantalla la imagen original y la final.
- Las tareas serán objetos que implementan la interfaz `Runnable` y serán ejecutadas por un `ThreadPoolExecutor`
- Todo su código estará en un fichero llamado `suavImagen.java`.

2. (Adivine Un Número, 1.5 puntos) Un conocido juego es el de adivinar el número que otra persona ha pensado. Deseamos implementar el juego mediante una arquitectura RMI de acuerdo a la siguiente interfaz:

```
import java.rmi.*;
public interface iJuego extends Remote
public void reset() throws RemoteException;
public boolean hacerIntento(int numero) throws RemoteException;
```

En ella, el método `reset()` permite al usuario cliente iniciar el juego, determinando en el lado del servidor un número entre 1 y 100 de forma aleatoria, y el método `hacerIntento(int numero)` permite enviar un número candidato para probar si se acierta o no. Se pide:

- Escribir un servidor `sJuego.java` que corra en el puerto 1066.
- Escribir un cliente `cJuego.java` que permita hacer intentos de adivinación y que ofreza un menú de usuario con las opciones ofrecidas por la interfaz.

3. (Análisis de Rendimiento, 4.5 puntos) Se nos ha dicho repetidas veces que el control de la exclusión mutua con cerrojos de clase `ReentrantLock` es más eficiente que el control con bloques de código sincronizado, y que ambos son menos eficientes que el uso de objetos con acceso atómico. Se pretende demostrar de forma empírica que esto es así verdaderamente. Para ello, se le pide que diseñe un programa experimental que analice el rendimiento del uso de las tres técnicas de control de la exclusión mutua descritas, utilizando variables de tipo `long` y objetos de clase `AtomicLong`. El programa deberá tener múltiples hilos que ciclan un número adecuado de veces para validar el experimento (recuerde fijar ese número en el código como una constante `llamada nIter`) y deber terminar imprimiendo una tabla que muestra las iteraciones de cinco mil en cinco mil iteraciones, junto con los tiempos requeridos para el control de la exclusión mutua con las tres técnicas descritas. Guarde su código en `aRendimiento.java`.