EXAMEN PARCIAL - Bloque Unidades Didácticas 1 a 6 (15/04/2013)

APELLIDOS:		NOMBRE:	
DNI:		FIRMA:	

Este bloque tiene una puntuación máxima de **2.5 puntos.**

Indique, para cada una de las siguientes 50 afirmaciones, si éstas son verdaderas (V) o falsas (F). **Cada respuesta vale: correcta= 0.05, errónea= -0.05, vacía=0.** Al final de cada pregunta dispone de un espacio reservado para justificar mejor su respuesta, en caso de considerarlo necesario.

1. Un programa concurrente:

F	Siempre es determinista.
V	Suele ser más complejo de depurar que uno secuencial.
F	Siempre presentará condiciones de carrera.
F	Deberá ejecutarse únicamente en máquinas con más de un procesador.

JUSTIFICACIÓN:

2. La comunicación mediante memoria compartida entre hilos:

F	No se puede emplear. Con hilos hay que emplear intercambio de mensajes.
F	Provoca necesariamente condiciones de carrera.
V	Ocurre cuando diferentes hilos leen y modifican objetos compartidos.
V	Requiere habitualmente el uso de algún mecanismo de control de concurrencia, como semáforos, cerrojos, monitores, etc.

JUSTIFICACIÓN:

3. Diferentes hilos que ejecuten una sección crítica de forma correcta:

V	Deben hacerlo respetando exclusión mutua.
F	Deben hacerlo mediante un protocolo de entrada que garantice progreso entre otras propiedades.
F	Deben hacerlo mediante un protocolo de entrada que garantice espera limitada entre otras propiedades.
V	Lo harán evitando que se produzcan condiciones de carrera.

JUSTIFICACIÓN:

4. Al ejecutar el siguiente programa:

```
public class p1 extends Thread {
   private int id = 0;

   public p1 (int id) {
      this.id = id;
   }

   public void run () {
      new p1(id+1);
      System.out.println ("id:" + id);
   }

   public static void main (String args[]) {
      Runnable r1 = new p1 (1);
      Thread t1 = new p1 (2);
      t1.run();
      r1.run();
      t1.start ();
      try { t1.join (); } catch (Exception e) {}
   }
}
```

V	Veremos por la pantalla exactamente 3 líneas.
V	Veremos al menos 2 líneas que contengan "id:2".
F	Se crearán y se ejecutarán al menos 3 hilos.
F	La última línea que veremos será "id:1".

JUSTIFICACIÓN:

5. Sobre los monitores:

F	Tal y como los encontramos en Java de forma estándar, siguen el modelo de Hoare.
F	Deben vincularse a algún semáforo para lograr exclusión mutua.
F	No están presentes en Java.
F	Garantizan determinismo.

JUSTIFICACIÓN:

6. El siguiente código Java pretende implantar un monitor para controlar el acceso de hilos lectores (concurrentes) y escritores (de manera exclusiva) sobre un recurso compartido. Los escritores utilizarán `writeStart()` antes de acceder y `writeEnd()` tras haber accedido. Los lectores utilizarán `readStart()` antes de acceder y `readEnd()` tras haber accedido:

```
public class ReadersWriters {
   private int writersWaiting;
   private boolean writing;
   private int readers;

   public ReadersWriters() {
       writersWaiting=readers=0;
       writing=false;
   }

   public synchronized void readStart() {
       while (writing || writersWaiting>0)
          try { wait(); }
            catch(Exception e) { };
       readers++;
   }
   public synchronized void readEnd() {
       readers--;
       notifyAll();
   }
   public synchronized void writeStart() {
       writersWaiting++;
       while (writing || readers>0)
          try { wait(); }
            catch(Exception e) { };
       writersWaiting--;
       writing=true;
   }
   public synchronized void writeEnd() {
       writing=false;
       notifyAll();
   }
}
```

F	Este código es erróneo, pues podrá haber múltiples escritores accediendo simultáneamente al recurso.
F	Este código es erróneo, pues podrá haber lectores y escritores accediendo simultáneamente al recurso.
F	Este código es erróneo, pues se ha utilizado `wait()` y `notifyAll()` en lugar de `await()` y `signalAll()`.
F	Este código es erróneo, pues no admite que haya múltiples lectores accediendo simultáneamente.

JUSTIFICACIÓN:

7. Sobre las condiciones de Coffman:

F	Son necesarias para que se ejecute la sección crítica de forma mutuamente excluyente.
F	Si se dan todas ellas, se producirá siempre un interbloqueo.
V	Si alguna de ellas no se cumple, no se producirá interbloqueo.
V	Se podrán satisfacer en un programa que emplee semáforos como mecanismo de control de concurrencia.

JUSTIFICACIÓN:

8. Sobre los `ReentrantLock` (cerrojos) de `java.util.concurrent`:

V	Permiten la creación de tantas variables condición como se desee.
F	No pueden emplearse en programas donde empleemos semáforos.
F	Se abren automáticamente siempre que el cerrojo quede fuera de ámbito. Es decir, se ejecuta `unlock()` automáticamente aunque el programador olvide hacerlo.
V	Se pueden emplear para solucionar el problema del productor-consumidor.

JUSTIFICACIÓN:

9. Los semáforos de Java:

F	No permiten programar soluciones a la sincronización condicional, pues no tienen variables Condición.
V	Permiten que se establezca el número de permisos inicial en su constructor.
V	Pueden emplearse para solucionar el problema de los 5 filósofos.
F	Al destruirlos, se invoca a la operación `release()`.

JUSTIFICACIÓN:

10. Sobre la programación en tiempo real:

V	Establece dificultades adicionales a la programación concurrente, pues deben satisfacerse restricciones temporales en los diferentes hilos.
F	Está soportada de forma nativa por lenguajes como Java y puede emplearse sin mayor problema sobre sistemas operativos de propósito general tales como Linux, MacOS o Windows, para implementar sistemas de tiempo real duros o blandos.
V	Suele requerir de una planificación de tareas donde se conozcan a priori el uso de la CPU de cada una de ellas.
F	Suele ejecutarse en sistemas con planificación round-robin.
V	La sincronización entre tareas en un sistema de tiempo real no resulta necesaria siempre y cuando las tareas de estos sistemas sean independientes entre sí.

JUSTIFICACIÓN:

11. Dado el siguiente código en Java

```
public class p3 extends Thread {
   private A a = null;
   private B b = null;

   /*
   public static void duerme () {
      long tpo = Math.round(Math.random()*100);
      try {Thread.sleep(tpo);} catch (Exception e) {}
   }
   */

   public static class A {
      private B b = null;
      int d=0;
      //public synchronized void a1 () {if(d++ <0) duerme(); b.b2();}
      public synchronized void a1 () {b.b2();}
      public synchronized void a2 () {}
      public void setB (B b) {this.b = b;}
   }

   public static class B {
      private A a = null;
      public synchronized void b1 () {a.a2();}
      public synchronized void b2 () {}
      public void setA (A a) {this.a = a;}
   }

   public p3 (A a, B b) {
      this.a = a; this.b=b;
   }

   public void run () {
      a.a1();
      b.b1();
   }

   public static void main (String args[]) throws Exception {
      A a = new A ();
      B b = new B ();
      a.setB(b); b.setA(a);
      Thread t1 = new p3 (a, b);
      Thread t2 = new p3 (a, b);
      t1.start ();
      t2.start ();
      t1.join ();
      t2.join ();
   }
}
```

F	Al ejecutarlo, se producirá siempre un interbloqueo.
V	Al ejecutarlo es posible que se produzca un interbloqueo.
F	Es incorrecto pues se producirán condiciones de carrera en las clases `A` y `B`. Ello es debido a que las operaciones `setA()` y `setB()` no están prefijadas con la palabra `synchronized`.
F	Es incorrecto pues la clase `p3` no puede tener una operación `run()`, al tratarse de una clase que extiende de Thread.

JUSTIFICACIÓN:

12. Si se aplica el análisis de planificabilidad de estas tareas, con pri(τ1)>pri(τ2)>pri(τ3), se obtiene... :

Tarea	T_i	C_i	D_i
τ_1	4	2	4
τ_2	9	2	5
τ_3	20	3	11

V	... que R1 es 2.
V	... que R2 es 4.
F	... que R3 es 11.
F	... que todos los plazos están garantizados y, por tanto, el sistema es planificable.
F	… que se produce el problema de inversión de prioridades.

JUSTIFICACIÓN (y cálculos):

EXAMEN PARCIAL - Bloque Prácticas 1y 2 (15/04/2013)

APELLIDOS:		NOMBRE:	
DNI:		FIRMA:	

Este bloque tiene una puntuación máxima de **0.5 puntos.**

Indique, para cada una de las siguientes 10 afirmaciones, si éstas son verdaderas (V) o falsas (F). **Cada respuesta vale: correcta= 0.05, errónea= -0.05, vacía=0.** Al final de cada pregunta dispone de un espacio reservado para justificar mejor su respuesta, en caso de considerarlo necesario.

1. Sobre los servicios de dominio de Active Directory

V	Almacenan información sobre los ordenadores, grupos y usuarios que forman parte de cada dominio.
F	No requieren de servidores, pudiendo ejecutarse de forma totalmente descentralizada en base únicamente a ordenadores con sistemas operativos Windows tipo Home Edition.
V	Emplean protocolos tales como Kerberos y LDAP entre otros.
V	Un usuario puede formar parte de varios grupos de usuarios de forma simultánea.
V	Organizan los dominios en bosques y árboles.

JUSTIFICACIÓN:

2. Sobre la práctica de la piscina vista en el laboratorio:

F	Debe ejecutarse sobre ordenadores diferentes para comprobar su correcta ejecución distribuida.
V	Nunca producirá interbloqueos al ejecutar la piscina de tipo 0.
V	Nunca producirá condiciones de carrera al seleccionar la piscina de tipo 0.
F	Todas las piscinas definidas por el alumno (tipos 1, 2, 3 y 4) utilizan las mismas variables para representar el estado interno de la piscina.
F	Al implementar las operaciones de la piscina de tipo 1, no será necesario usar la palabra `synchronized`, pues esta piscina no tiene estado interno.

JUSTIFICACIÓN: