APELLIDOS:		NOMBRE:	
DNI:		FIRMA:	

Examen final - Bloque primer parcial (18 junio 2013)

Este bloque tiene una puntuación máxima de **10 puntos.**

Indique, para cada una de las siguientes 50 afirmaciones, si éstas son verdaderas (V) o falsas (F). **Cada respuesta vale: correcta= 0.2, errónea= -0.2, vacía=0.**

1. Un programa concurrente:

F	Puede estar formado por una única actividad.
V	Necesita generalmente mecanismos de comunicación y sincronización para que sus actividades colaboren entre sí.
V	Puede implantar la comunicación mediante intercambio de mensajes.
F	Necesita planificación expulsiva basada en prioridades.

2. Una sección crítica:

V	Es una sección de código donde múltiples actividades puedan acceder a recursos (objetos, variables…) compartidos.
V	Podrá provocar condiciones de carrera si no se protege adecuadamente mediante un protocolo de entrada y un protocolo de salida.
F	Se protege adecuadamente al utilizar el algoritmo de Cristian.
F	No provocará ningún problema si se utiliza el algoritmo de Chandy y Lamport en sus protocolos de entrada y salida.

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

V	Deben hacerlo respetando exclusión mutua, progreso y espera limitada.
F	Deben hacerlo evitando la espera circular.
F	Deben hacerlo respetando no expropiación.
F	Deben hacerlo evitando retención y espera.

4. Dado el siguiente programa Java:

```
class Prueba extends Thread {
    private int i;
    public Prueba(int a) {i=a;}
    public void run() {
        for (int j=0; j<i; j++)
            new Prueba(j).start();
        System.out.println("i:"+ i);
    }
    static public void main(String args[]) {
        new Prueba(2).start();
    }
}
```

F	No se ejecutará ningún hilo (aparte del principal), pues no hay invocaciones del método run().
V	Veremos al menos 2 líneas que contengan "i:0".
V	Se crearán y se ejecutarán al menos 3 hilos.
F	Veremos al menos 2 líneas que contengan “i:1”.

5. Sobre los monitores:

F	Tal y como los encontramos en Java de forma estándar, siguen el modelo de Brinch Hansen.
V	Son una herramienta que permite combinar exclusión mutua con sincronización condicional.
F	No hay monitores en Java.
F	El modelo de Lampson y Redell no soporta sincronización condicional.

6. El siguiente código Java pretende implantar un monitor para controlar el acceso de múltiples 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 writers;
   private int readers;

   public ReadersWriters() {
       readers=0;
       writers=0;
   }

   public synchronized void readStart() {
       while (writers>0 || readers>5)
          try { wait(); }
            catch(Exception e) { };
       readers++;
   }
   public synchronized void readEnd() {
       readers--;
       notifyAll();
   }
   public synchronized void writeStart() {
       while (writers>0 || readers>0)
          try { wait(); }
            catch(Exception e) { };
       writers++;
   }
   public synchronized void writeEnd() {
       writers--;
       notifyAll();
   }
}
```

F	Este código admite múltiples escritores accediendo simultáneamente al recurso.
F	Este código admite lectores y escritores accediendo simultáneamente al recurso.
V	Este código permite que varios lectores accedan simultáneamente al recurso.
V	Este código puede causar inanición a los escritores.

7. Sobre las condiciones de Coffman:

F	Son suficientes para que haya un interbloqueo.
F	Si se dan todas ellas, se producirá siempre un interbloqueo.
V	Si alguna de ellas no se cumple, no habrá interbloqueo.
V	Se podrán cumplir en un programa que utilice monitores como mecanismo de control de concurrencia.

8. Sobre los `CountdownLatch` de `java.util.concurrent`:

F	Permiten la creación de variables condición mediante su método `newCondition()`.
F	No pueden emplearse en programas donde empleemos `CyclicBarrier`.
F	Se abren automáticamente cuando se han realizado suficientes invocaciones a su método `await()`.
V	Pueden abrirse cuando se invoca su método `countDown()`.

9. Los semáforos (clase `Semaphore`) de Java:

F	Siguen el modelo de Lampson y Redell.
F	Decrementan su contador interno cuando se invoca su método `await()`.
F	Incrementan su contador interno cuando se invoca su método `acquire()`.
F	Se puede reactivar a un hilo suspendido si se invoca su método `acquire()`.

10. Sobre la programación en tiempo real:

F	No es un tipo de programación concurrente, pues cada tarea de un programa en tiempo real es independiente de las demás y no se relaciona con ellas.
V	El problema de la inversión de prioridades puede aparecer al utilizar mecanismos de control de concurrencia.
V	Suele necesitar un análisis de planificabilidad.
F	Suele ejecutarse en sistemas con planificación FCFS.
V	Requiere la utilización del protocolo de techo de prioridad inmediato (u otro similar) cuando se utilicen semáforos para proteger las secciones críticas.

11. Dado el siguiente código en Java

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

public class p3 extends Thread {
     private ReentrantLock a,b;
     private int i;

     public p3 (ReentrantLock a1, ReentrantLock b1, int i1) {
        a=a1; b=b1; i=i1;
     }

     public void run () {
       try{
        if (i==0) {
              a.lock(); sleep(1000); b.lock();
        } else {
              b.lock(); sleep(1000); a.lock();
        }
        System.out.println("printing...");
       } catch (Exception e) {
       } finally {
        b.unlock();
        a.unlock();
       }
     }

     public static void main (String args[]) throws Exception {
        ReentrantLock c,d;
        p3 h1,h2;

        c=new ReentrantLock();
        d=new ReentrantLock();
        h1=new p3(c,d,0);
        h2=new p3(c,d,1);
        h1.start();
        h2.start();
     }
  }
```

F	No habrá interbloqueos pues jamás se cumplirá la condición de retención y espera.
V	Al ejecutarlo es posible que se produzca un interbloqueo.
F	No habrá interbloqueos pues jamás se cumplirá la condición de espera circular.
F	Es incorrecto pues la clase `ReentrantLock` solo puede ser utilizada para desarrollar monitores avanzados en Java.

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	3	1	2
τ_2	5	2	3
τ_3	10	2	6

F	... que R1 es 2.
V	... que R2 es 3.
F	... que R3 es 10.
F	... que todos los plazos están garantizados y, por tanto, el sistema es planificable.
F	… que se produce el problema de inversión de prioridades.

Espacio para cálculos:

R1= 1

R2= 3

R3= 9