Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 1)/clave: b

Curso 2014/2015 - 2º Semestre (Mayo 2015)
Grado en Ingeniería Informática / Grado en Matemáticas e Informática
UNIVERSIDAD POLITÉCNICA DE MADRID

NORMAS: Este es un cuestionario que consta de 7 preguntas. Todas las preguntas son de respuesta simple excepto la pregunta 7 que es una pregunta de desarrollo. La puntuación total del examen es de 10 puntos. La duración total es de una hora. El examen debe contestarse en las hojas de respuestas. No olvidéis rellenar apellidos, nombre y número de matrícula en cada hoja de respuesta.

Sólo hay una respuesta válida a cada pregunta de respuesta simple. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma.

Cuestionario

(1 punto) 1. Dado el siguiente programa concurrente:

```
static class Hilos {
  static class MiHilo
    extends Thread {
    volatile int n = 0;
    public void run () {
      n++;
    }
  }
```

```
  public static final void
    main(final String[] args) {
    Thread t = new MiHilo();

    t.start();
    t.run();
  }
}
```

Se pide marcar la afirmación correcta.

(a) El máximo número de procesos que pueden ejecutar simultáneamente es tres.

(b) El máximo número de procesos que pueden ejecutar simultáneamente es dos.

(1 punto) 2. Dado el programa concurrente de la pregunta 1.

Se pide marcar la afirmación correcta.

(a) `n++` no es una sección crítica.

(b) `n++` es una sección crítica.

(1 punto) 3. La clase `PorTurno` implementa un protocolo de acceso a una sección crítica:

```
static int volatile turno = 0;

static class PorTurno extends Thread {
    private int pid;

    public PorTurno(int pid) {
        this.pid = pid;
    }
```

```
    public void run() {
        while (true) {
            seccionNOcritica();
            while (turno != pid) {}
            seccionCritica();
            int aux = (turno + 1)
                    % MAX_THREADS;
            turno = aux;
        }
    }
}
```

Dado un programa concurrente con `MAX_THREADS` *threads* de la clase `PorTurno` compartiendo una variable `turno` inicializada a `0` y cada una de ellas con un índice `pid` distinto entre `0` y `MAX_THREADS-1`.

Se pide marcar la afirmación correcta.

(a) Los procesos pueden esperar innecesariamente para entrar en `seccionCritica()`.

(b) El programa no cumple la propiedad de exclusión mutua en `seccionCritica()`.

(1½ puntos) 4. Dado un programa concurrente en la que dos *threads* instancias de las clases `A` y `B` comparten una variable `n`:

```
static int n = 0;
static Semaphore s1 = new Semaphore(1);
static Semaphore s2 = new Semaphore(0);
```

```
static class A extends Thread {
    public void run() {
        s2.await();
        n = 3 * n;
        s1.signal();
    }
}
```

```
static class B extends Thread {
    public void run() {
        s1.await();
        n = n + 2;
        s2.signal();
    }
}
```

¿Cuál es el valor de `n` tras terminar los dos threads?

(a) 6

(b) 2

(1 punto) 5. Si en el código anterior el semáforo `s2` se inicializa a 1:

(a) No está garantizada la exclusión mutua en el acceso a `n`.

(b) No está garantizada la terminación de ambas tareas.

(1½ puntos) 6. Dado el siguiente **CTAD** (sólo se muestran las partes necesarias):

TIPO: T4 $= (a : \mathbb{B} \times b : \mathbb{B})$
INICIAL: $self = (falso, falso)$
CPRE: $\neg(self.a)$
x()
POST: $self^{pre} = (ae, be) \wedge self = (\neg ae, be)$
CPRE: $self.a$
y()
POST: $self^{pre} = (ae, be) \wedge self = (\neg ae, \neg be)$

Se pide marcar la afirmación correcta:

(a) El recurso puede pasar, a lo sumo, por 3 estados diferentes.

(b) Si el sistema solo contiene (además de un recurso de este tipo) un proceso que intenta ejecutar $x()$ repetidamente y otro que intenta ejecutar $y()$ repetidamente, ambos ejecutarán dichas acciones en alternancia estricta.

(c) El recurso podría estar en un estado dado y, tras ejecutarse una sola de sus acciones, continuar en el mismo estado.

(d) Si el sistema solo contiene (además de un recurso de este tipo) un proceso que intenta ejecutar $x()$ repetidamente y otro que intenta ejecutar $y()$ repetidamente, el sistema podría acabar en interbloqueo.

(Sugerencia: Dibuja aquí el grafo de los estados por los que puede pasar el recurso.)

Apellidos: **Nombre:** **Matrícula:**

(3 puntos) 7. **Se pide** completar la siguiente especificación de un recurso compartido que implementa la adquisición simultánea de pares de tokens adyacentes escogidos de un vector circular de tokens. Las operaciones *adquirir* y *soltar* reciben un índice válido del vector de tokens. La operación *adquirir(i)* corresponde a la adquisición de los tokens en las posiciones i e $i+1$ (o 0 si $i+1$ es igual al número total de tokens). La operación *soltar(i)* devuelve los tokens de las posiciones i e $i+1$.

Un proceso puede invocar a la adquisición de dos tokens adyacentes cuando éstos no estén disponibles, en cuyo caso tendrá que bloquearse hasta que lo estén. Por el contrario, no se permite que un proceso invoque *soltar* sobre tokens que no estén asignados (podéis usar la clásula PRE para expresar esta prohibición).

C-TAD AnilloTokens

OPERACIONES

ACCIÓN adquirir: *Indice[e]*
ACCIÓN soltar: *Indice[e]*

SEMÁNTICA
DOMINIO:

TIPO: *AnilloTokens* =

TIPO: *Indice* = {*0, 1, . . . , N-1*}

INVARIANTE:

INICIAL:

PRE:

CPRE:

adquirir(i)

POST:

PRE:

CPRE:

soltar(i)

POST:

(Página intencionadamente en blanco, puede usarse como hoja en sucio).