

## Exámenes

### GrupoD-TestAula-ud4a6

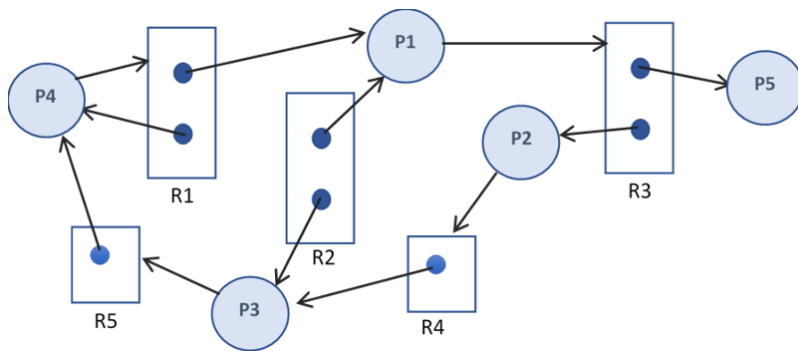
[Volver a la Lista de Exámenes](#)

---

## Parte 1 de 5 - Sobre los interbloqueos.

1.32/ 1.32 Puntos

Dado el siguiente grafo de asignación de recursos:



## Ficheros adjuntos

GAR1.pdf 38 KB

Preguntas 1 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

En el grafo mostrado existe una única secuencia segura.

☒ Verdadero

Habrán tantas secuencias seguras como procesos sin aristas de petición -&gt; SOLO UNA.

☐ Falso

Preguntas 2 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Se puede afirmar que en el grafo actual no hay un interbloqueo..

☒ Verdadero

Si hay una secuencia segura -&gt; NO hay interbloqueo.

☐ Falso

Preguntas 3 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Como existe un ciclo dirigido en el que hay recursos con una única instancia, podemos afirmar que existe un interbloqueo.

☒ Verdadero

Para poder hacer tal afirmación, todos los recursos deberían tener únicamente una instancia.

☐ Falso

Preguntas 4 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si el proceso P5 solicita una instancia de R4, entonces todos los procesos quedarán en situación de interbloqueo.

☒ Verdadero

Ya que todos tendrían aristas de petición, quedarán en interbloqueo.

☐ Falso

## Parte 2 de 5 - Sobre Sistemas de Tiempo Real

1.09/ 3.07 Puntos

Un sistema de tiempo real crítico se compone de 5 tareas (T1,T2 ...,T5) que utilizan 3 semáforos (M1,M2,M3) para sincronizar el acceso a sus secciones críticas (SC), y cuyas características se describen en las siguientes tablas:

Tarea	Semáforo	Duración de la SC
T1	M1	1
T2	M1	1
T2	M2	2
T4	M2	3
T4	M3	1
T5	M3	2

Tarea	t.CPU $C_i$	Periodo $T_i$	Plazo $D_i$
T1	3	10	7
T2	4	15	10
T3	6	30	23
T4	5	40	40
T5	3	60	60

Asumiendo que los semáforos utilizan el protocolo del techo de prioridad inmediato y el sistema se planifica por prioridades fijas expulsivas, con asignación de prioridades  $T1 > T2 > T3 > T4 > T5$ , indique para cada una de las siguientes **9 afirmaciones**, si éstas son verdaderas (V) o falsas (F) .

Preguntas 5 de 30

0.33/ 0.33 Puntos

El semáforo con el techo de prioridad mayor es M2 porque guarda la sección crítica de mayor duración.

☐ Verdadero

**El techo de prioridad es equivalente a la mayor prioridad de las tareas que lo utilizan.**

☐ Falso

Luego:

techo(M1) = 1.

techo(M2) = 2.

techo(M3) = 4.

**Por lo que el techo de mayor prioridad es el M1.**

Preguntas 6 de 30

0.33/ 0.33 Puntos

El factor de bloqueo de T1 es 1 y el de T2 es 3.

☐ Verdadero

Factor de bloqueo de  $T_1$  -> Tareas de prioridad inferior ->  $T_2, T_4, T_5$  -> semáforos de techo con prioridad  $\geq$  semáforo de  $T_1$  ( $M_1$ ) ->  **$M_1$  de  $T_2$  ->  $B_1 = SC_{\text{semáforo-M1 de } T_2} = 1$ .**

☐ Falso

Factor de bloqueo  $T_2$  -> Tareas de prioridad inferior ->  $T_4, T_5$  -> semáforos de techo con prioridad  $\geq$  semáforo de  $T_2$  ( $M_2$ ) ->  **$M_2$  de  $T_4$  ->  $B_2 = SC_{\text{semáforo-M2 de } T_4} = 3$ .**

Preguntas 7 de 30

0.33/ 0.33 Puntos

El factor de bloqueo de T3 y T5 es 0.

☐ Verdadero

$T_5$  es la tarea menos prioritaria -> **Factor de bloqueo de  $T_5 = 0$** , ya que la tarea de prioridad más baja, por definición, no puede ser bloqueada por otras tareas.

☐ Falso

Factor de bloqueo  $T_3$  -> Tareas de prioridad inferior ->  $T_4, T_5$  ->  **$B_3 = \text{mayor}(SC_{\text{semáforos-}T_4, T_5}) = 3$ .**

Preguntas 8 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

El factor de bloqueo de T4 es 2.

☐ Verdadero

☐ Falso

Factor de bloqueo  $T_4$  -> Tareas de prioridad inferior ->  $T_5$  -> semáforos de techo con prioridad  $\geq$  semáforo de  $T_4$  ( $M_3$ ) ->  $M_3$  de  $T_5$  ->  **$B_4 = SC_{\text{semáforo-M3 de } T_5} = 2$ .**

## Preguntas 9 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

El tiempo de respuesta de T1 es 3.

☐ Verdadero $w_1^0 = 3 \rightarrow$  Para prioridades fijas (sin semáforos).☐ Falso $R_1 = w_1^0 + B_1 = 3 + 1 = 4 \rightarrow$  Prioridades expulsivas.

## Preguntas 10 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

La tarea T3 tiene un tiempo de respuesta mayor que su plazo.

☐ Verdadero $w_3^0 = \sum C_i = C_1 + C_2 + C_3 = 3 + 4 + 6 = 13$ ☐ Falso $w_3^1 = 6 + [13/15]4 + [13/10]3 = 6 + 4 + 2 * 3 = 16$  $w_3^2 = 6 + [16/15]4 + [16/10]3 = 6 + 2 * 4 + 2 * 3 = 20$  $w_3^3 = 6 + [20/15]4 + [20/10]3 = 6 + 2 * 4 + 2 * 3 = 20$ 

## Preguntas 11 de 30

 $R_3 = 20 + B_3 = 20 + 3 = 23 = 23.$  0.43/ 0.43 Puntos. Puntos descontados por fallo: 0.33

El tiempo de respuesta de T5 es 59.

☐ Verdadero

Mismo proceso que el anterior ejercicio. Da verdadero.

☐ Falso

## Preguntas 12 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

El sistema es planificable.

☐ VerdaderoEl sistema será planificable si todos los  $\sum R_i < \sum D_i$ .☐ Falso $R_1 = 4$  y no es menor que  $D_1 = 7 \rightarrow$  No es planificable.

## Preguntas 13 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

El hiperperiodo es 120.

☐ VerdaderoHiperperiodo =  $\text{mcm}(\sum T_i) = \text{mcm}(10, 15, 30, 40, 60) = 120.$ ☐ Falso

## Parte 3 de 5 -

0.0/ 0.66 Puntos

Sobre la clase Executors de java.util.concurrent:

Preguntas 14 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

La sentencia `ExecutorService miejecutor = Executors.newFixedThreadPool(3);` crea un pool de 3 hilos que se utilizarán para ejecutar las tareas que se le deleguen a `miejecutor`.

☐ Verdadero☐ Falso

Preguntas 15 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Sea `miejecutor` el `ExecutorService` creado desde el hilo principal (main) con la instrucción: `ExecutorService miejecutor = Executors.newFixedThreadPool(3);`.

Si main ejecuta la sentencia `miejecutor.awaitTermination(20L, TimeUnit.SECONDS)`, todos los hilos del pool terminarán su ejecución en 20 segundos .

☐ Verdadero☐ Falso

Se bloquea hasta que todas las tareas hayan acabado, o hayan pasado 20 segundos, o que el hilo sea interrumpido, en definitiva, por lo que suceda primero.

## Parte 4 de 5 - Monitor Orden

0.99/ 2.97 Puntos

Se pretende desarrollar un monitor para ordenar el acceso de un conjunto de hilos a un recurso compartido del que sólo hay una instancia. El enfoque es el mismo que se utiliza en los mercados para determinar el siguiente cliente al que atender entre los que esperan. Los hilos antes de intentar acceder al recurso deben solicitar un número de orden. El monitor debe asegurar que siempre accederá el hilo con el menor número. Analice la siguiente propuesta de código y responda a las siguientes 9 afirmaciones con V/F:

```
public class numerosOrden {
    long nActual = 0;
    long numeroSiguiente () {
        nActual ++;
        return nActual;
    }
}
```

```
import java.util.concurrent.locks.*;
public class monitorOrdenRL {
    numerosOrden numeroOrden = new numerosOrden();
    long numeroActual = 1;
    Lock milock=new ReentrantLock();
    Condition espera=milock.newCondition();

    public long solicitarTurno() {
        try { milock.lock();
            return numeroOrden.numeroSiguiente();
        } finally { milock.unlock();}
    }

    public void accederRecurso(long numeroOrden) throws InterruptedException {
        try { milock.lock();
            while (numeroOrden > numeroActual) { espera.await();}
        } finally { milock.unlock();}
    }
}
```

```

    }
    public void liberarRecurso() {
        try { milock.lock();
            numeroActual ++;
            espera.signalAll();
        } finally { milock.unlock();}
    }
}

```

```

public class hiloRecurso extends Thread{
    int id = 0;
    static final int n = 3;
    monitorOrdenRL unMonitorOrden;
    public hiloRecurso (int unId, monitorOrdenRL unMonitorOrden){
        id = unId;
        this.unMonitorOrden = unMonitorOrden;
    }
    public void run () {
        long turno = 0;
        for (int i=0; i < n; i++ ){
            turno = unMonitorOrden.solicitarTurno();
            System.out.println(">Hilo " + id + " obtiene numero " + turno);
            try {Thread.sleep(Math.random()*100);}catch (InterruptedException e) {};
            try {unMonitorOrden.accederRecurso(turno);
            } catch (InterruptedException e) {}
            System.out.println("<Hilo " + id + " accede al recurso. Turno " + turno);
            unMonitorOrden.liberarRecurso();
        }
    }
}


```

```

public class pruebaMonitorOrdenRL {
    public static void main(String[] args) {
        int n = 5;
        hiloRecurso unHilo;
        monitorOrdenRL unMonitorOrden = new monitorOrdenRL();
        for (int i=1; i <= n; i++){
            unHilo = new hiloRecurso(i, unMonitorOrden);
            unHilo.start();
        }
    }
}

```

## Ficheros adjuntos

 [MonitorOrden.pdf](#) 36 KB

Preguntas 16 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Se crean 5 hilos aparte del principal, y la cadena "Hilo" no forma parte del nombre del hilo.

☐ Verdadero

$main + n = 5 \rightarrow 1 + 5 = 6.$

☐ Falso

## Preguntas 17 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Como los hilos se crean en orden según su id, la primera vez que tomen el turno siempre obtendrán como turno el valor de su id, es decir el hilo 1 obtendrá el turno 1, el hilo 2 el turno 2 y así sucesivamente.

☐ Verdadero☐ Falso

## Preguntas 18 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

El lock `milock` debería haberse creado con el argumento `fair` a `true`, para que los hilos accedan en orden al recurso (ej: `ReentrantLock milock=new ReentrantLock(true);`)

☐ Verdadero☐ Falso

El argumento `fair` en un `ReentrantLock` lo que hace es garantizar el acceso al hilo que más esté esperando.

## Preguntas 19 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Los hilos que tarden demasiado en invocar `accederRecurso`, se les pasará el turno y otros hilos podrán acceder al recurso. Cuando los hilos tardones invoquen `accederRecurso`, entrarán directamente a utilizar el recurso aunque esté ocupado, pues el monitor sólo comprueba el número de turno y no si está ocupado.

☐ Verdadero☐ Falso

## Preguntas 20 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Una vez cogido el turno, no importa lo que el hilo H1 tarde en invocar el método `accederRecurso`, el resto de hilos no podrá acceder al recurso hasta que H1 lo haga y lo libere.

☐ Verdadero☐ Falso

Verdadero, ya que cuando accede al recurso, el lock se cierra, por lo que nadie más lo podrá usar.

## Preguntas 21 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Se pueden producir condiciones de carrera al obtener un número de orden, porque el método `numeroSiguiente` de la clase `numerosOrden` no está sincronizado ( no tiene la etiqueta `synchronized`).

☐ Verdadero☐ Falso

No se podrán producir condiciones de carrera, ya que aunque el método en sí no esté sincronizado, se ejecuta desde un método donde sí lo está (se ejecuta con el lock cerrado -> no se pueden producir condiciones de carrera).

## Preguntas 22 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

La variable `nActual` de la clase `numerosOrden` debería haberse declarado como un `AtomicLong`, y en el método `numeroSiguiente` hay que utilizar su método `incrementAndGet` para que no se produzcan condiciones de carrera en la solución propuesta.

☐ Verdadero☐ FalsoFalso, ya que de donde se accede (del método `accederRecurso`), está sincronizado.

## Preguntas 23 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si la clase `monitorOrdenRL` la sustituimos por el siguiente código, la solución es equivalente:

```
public class monitorOrdenRL {  
  
    numerosOrden numeroOrden = new numerosOrden();  
  
    long numeroActual = 1;  
  
    public synchronized long solicitarTurno() {return numeroOrden.numeroSiguiente();}  
  
    public synchronized void accederRecurso(long numeroOrden) throws InterruptedException {  
        while (numeroOrden > numeroActual) wait();  
    }  
  
    public synchronized void liberarRecurso() { numeroActual ++;notifyAll();}  
}
```

☐ Verdadero☐ Falso

Verdadero, ya que sigue sincronizado igual.

## Preguntas 24 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si todos los hilos solicitan el recurso al mismo tiempo, se producirá un interbloqueo porque sólo hay una instancia del recurso.

☐ Verdadero☐ FalsoHay 3 instancias -> `static final int n = 3`.

## Parte 5 de 5 - El Album de Fotos.

-1.32/ 1.98 Puntos

Asuma que existe una clase `Almacén` "thread-safe" con capacidad para 100 elementos (fotografías), cuyo método `put()` permite insertar un elemento y cuyo método `get()` extrae un elemento en el orden en que han sido insertados. Existen varios fotógrafos que utilizan el almacén para depositar sus fotografías. Se pretende crear un álbum obteniendo 100 fotografías del almacén, de las que al menos ha de haber 30 fotografías de cada fotógrafo. Supóngase que ya existen los métodos `tomar_fotografía` y `añadir_foto_album` implementados. Se dispone del siguiente código y etiquetas posibles para rellenarlo:



## Código a analizar y rellenar:

<pre> public class Main{     public static void main(String[] args){         Almacén d=new Almacén();         <b>SENTENCIA A</b>         Fotógrafo w1=new Fotógrafo (c,d,1);         Fotógrafo w2=new Fotógrafo (c,d,2);         Fotógrafo w3=new Fotógrafo (c,d,3);         w1.start();         w2.start();         w3.start();          try{             <b>SENTENCIA B</b>         }catch(InterruptedException e){}         for (int i=1; i&lt;=100; i++)             añadir_foto_album(d.get());         }     } </pre>	<pre> public class Fotógrafo extends Thread{     <b>SENTENCIA C</b>     private Almacén buf;     private fotografía foto;     private int number;      public Fotógrafo(<b>SENTENCIA D</b>, Almacén c, int id)     {         obj=a;         buf=c;         number=id;     }      public void run(){         for (int i=1; i&lt;=100; i++) {             foto=tomar_fotografía();             buf.put(foto);             if (i==30){                 try{ <b>SENTENCIA E</b>                 }catch(InterruptedException e){} }             }         }     } } </pre>
---	---

## Etiquetas posibles para rellenarlo:

ID	ETIQUETA	ID	ETIQUETA
1	Semaphore c=new Semaphore(0);	2	Semaphore c=new Semaphore(4);
3	CyclicBarrier c=new CyclicBarrier(3);	4	CyclicBarrier c=new CyclicBarrier(4);
5	CountDownLatch c=new CountDownLatch(0);	6	CountDownLatch c=new CountDownLatch(3);
7	CountDownLatch c=new CountDownLatch(4);	8	int c=0;
9	c.await();	10	c.acquire();
11	c.release();	12	c.countDown();
13	c.acquire(); c.acquire(); c.acquire();	14	w1.join(); w2.join(); w3.join();
15	private Semaphore obj;	16	private CyclicBarrier obj;
17	private CountDownLatch obj;	18	private int obj;
19	Semaphore a	20	CyclicBarrier a
21	CountDownLatch a	22	int a
23	obj.release();	24	obj.acquire();
25	obj.await();	26	obj.countDown();

## Ficheros adjuntos

 [AlbumdeFotos.pdf](#) 52 KB

Preguntas 25 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Una solución consiste en utilizar en SENTENCIA B el método join de la clase Thread. Es decir, la etiqueta 14, y dejar el resto de sentencias vacías.

☐ Verdadero

La sentencia D, por ejemplo, no puede estar vacía.

☐ Falso

## Preguntas 26 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si se emplean semáforos para coordinar a los Fotógrafos y al hilo principal, deberíamos utilizar la combinación de sentencias-etiquetas siguiente: A-1, B-13, C-15, D-19, E-23

☐ Verdadero☐ Falso

Ya que  $A_1 \rightarrow \text{Semaphore } c = \text{new Semaphore}(0);$  y  $B_{13}$  son 3  $c.\text{acquire}();$   
Al no hacer ningún  $\text{release}()$  antes, para que tengan algún recurso disponible, el programa se quedará en un bucle infinito y no acabará.

## Preguntas 27 de 30

0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si se emplean barreras para coordinar a los Fotógrafos y al hilo principal, podríamos utilizar la combinación de sentencias-etiquetas siguiente: A-3, B-9, C-16, D-20, E-26.

☐ Verdadero☐ Falso

E-26 dice la siguiente línea de instrucción: `obj.countDown()`, que es únicamente posible para un `CountDownLatch`, y no barrera cíclica, por lo que es falso.

En una barrera cíclica, cuando todos los objetos llegan a `obj.await()`, la barrera se abre.

## Preguntas 28 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si se emplean barreras para coordinar a los Fotógrafos y al hilo principal, podríamos utilizar la combinación de sentencias-etiquetas siguiente: A-7, B-12, C-17, D-21, E-25.

☐ Verdadero☐ Falso

Como se crea una barrera de 4, y solo hay 3 objetos, la barrera nunca se abrirá -> Falso.

## Preguntas 29 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si se emplean barreras para coordinar a los Fotógrafos y al hilo principal, podríamos utilizar la combinación de sentencias-etiquetas siguiente: A-4, B-9, C-16, D-20, E-25.

☐ Verdadero☐ Falso

Funcionará, al llegar todos a `obj.await()` en Fotógrafo, la barrera se abre y el main continúa correctamente.

## Preguntas 30 de 30

-0.33/ 0.33 Puntos. Puntos descontados por fallo: 0.33

Si se emplean barreras para coordinar a los Fotógrafos y al hilo principal, podríamos utilizar la combinación de sentencias-etiquetas siguiente: A-6, B-9, C-17, D-21, E-12.

☐ Verdadero☐ Falso

- PoliformaT
- UPV

- Powered by Sakai
- Copyright 2003-2020 The Sakai Foundation. All rights reserved. Portions of Sakai are copyrighted by other parties as described in the Acknowledgments screen.