

Programación Concurrente y de Tiempo Real  
Grado en Ingeniería Informática  
Examen Final Teórico de la Asignatura  
Febrero de 2018

Apellidos:

Nombre:

D.N.I.:

Grupo (A ó B):

## 1. Notas

1. Escriba su nombre, apellidos, D.N.I. y grupo en el espacio habilitado para ello, y en todos los folios blancos que utilice. Firme el documento en la esquina superior derecha de la primera página.
2. Dispone de diez minutos para leer los enunciados y formular preguntas o aclaraciones sobre ellos. Transcurrido ese tiempo, no se contestarán preguntas. Dispone de 60 minutos para completar el ejercicio.
3. No complete el documento a lápiz. Utilice bolígrafo o rotulador. Escriba con letra clara y legible. No podemos corregir lo que no se puede leer.
4. Utilice los folios blancos que se le proporcionan para resolver los enunciados, pero traslade a este documento únicamente la solución final que obtenga, utilizando el espacio específicamente habilitado para ello, sin sobrepasarlo en ningún caso, y sin proporcionar información o respuestas no pedidas. Entregue tanto el enunciado como los folios blancos. Únicamente se corregirá este documento.

## 2. Criterios de Corrección

1. El examen se calificará de cero a diez puntos, y ponderará en la calificación final al 40 % bajo los supuestos recogidos en la ficha de la asignatura.
2. Cada enunciado incluye información de la puntuación que su resolución correcta representa, incluida entre corchetes.
3. Un enunciado (cuestión teórica o problema) se considera correcto si la solución dada es correcta completamente. En cualquier otro caso se considera incorrecto y no puntúa.

4. Un enunciado de múltiples apartados (cuestión teórica o problema) es correcto si y solo si todos los apartados que lo forman se contestan correctamente. En cualquier otro caso se considera incorrecto y no puntúa.

### 3. Test de Alternativa Simple

#### INSTRUCCIONES:

Consteste a las preguntas tipo test que siguen, dedicando aproximadamente un minuto a cada una. Para cada pregunta, existe una única respuesta correcta, que aporta 0,25 puntos. Una respuesta incorrecta resta 0,15 puntos. Las preguntas sin contestar no puntúan. Si considera que todas o varias respuestas son correctas, escoja únicamente la letra que describa esta situación. Para la respuesta que considere correcta, traslade la letra que la identifica a las tablas situadas a continuación. Ejemplo: si la pregunta número 14 tiene como respuesta la opción b), debe localizar la casilla identificada con "14" y situar en la casilla de debajo la letra "b". Puede utilizar las letras de los ítems del test a modo de borrador, pero coloque en las tablas únicamente la opción que considere correcta. Se corregirá exclusivamente la tabla, y las marcas o señales efectuadas en el cuerpo del test no tendrán valor. El test completamente correcto aporta a la calificación del examen 5,0 puntos.

#### TABLA DE RESPUESTAS:

1	2	3	4	5	6	7	8	9	10

11	12	13	14	15	16	17	18	19	20

(PARA USO EXCLUSIVO DEL PROFESORADO)

TOTALES	Número	Puntos
Correctas		
Incorrectas		
Total Puntos		

1. Suponga que 1000 hebras comparten el acceso a un objeto común cada una a través de su propia referencia, y ejecutan el acceso a un método `inc()` de ese objeto que incrementa un contador que inicialmente vale -1. El programador protege el acceso haciendo que cada hebra ejecute la llamada a `inc()` envolviendo la misma en un bloque de la forma `synchronized(this){...miReferencia.inc();...}`. El valor final del contador es:
- a) 999.
  - b) 1000.
  - c) un número indeterminado entre 0 y 1000.
  - d) un número indeterminado entre 1 y 1001.
  - ☒ e) ninguna de las anteriores es correcta.
2. Considere un objeto en Java que dispone de tres métodos `m1`, `m2` y `m3`. El programador decide que deben ejecutarse en exclusión mutua, y para ello dispone todo el código de los métodos `m1` y `m3` dentro de un bloque `synchronized(this)`. Para el tercer método, utiliza un objeto de clase `Semaphore` que inicializa a cero, y engloba todo el código del mismo bajo el par de instrucciones `acquire()` y `release()`. La sección crítica en todos los casos implica incrementar una variable de tipo `int` que inicialmente vale 3. Tres hebras externas A, B y C activadas dentro de una co-rutina ejecutan lo siguiente: A.m1, C.m3 y B.m2. Terminada la co-rutina, el programa principal imprime el valor de la variable y este es:
- a) 3, ya que las tres hebras quedan bloqueadas y no lo modifican.
  - b) 6, pues las tres hebras hacen su incremento de modo segura.
  - ☒ c) el programa realmente no imprime nada porque queda bloqueado a causa de las hebras.
  - e) 5, porque dos hebras hacen su incremento y otra no, ya que queda bloqueada.
3. En Java, utilizando cerrojos de clase `ReentrantLock` y variables `Condition` es posible disponer
- a) de señalización SC y SX.
  - b) de señalización SA, SC y SX.
  - c) de señalización SC y SA.
  - ☒ d) ninguna de las anteriores correcta.
4. Una hebra de C++
- ☒ a) únicamente puede recibir su código mediante un puntero a función.
  - b) tras ser instanciada mediante el constructor de clase, deber ser lanzada explícitamente por programa.
  - ☒ c) puede estar sincronizada con el programa principal o con otras hebras mediante el método `join()`
  - d) Todas las anteriores son correctas.

5. Desarrollando con el *framework* RMI, el binario `rmiregistry` :

- a) permite registrar diferentes objetos de diferentes clases.
- b) permite registrar diferentes instancias de un mismo objeto.
- c) efectúa funciones de DNS mediante clientes que hace *dynamic-binding* con él.
- d) todo lo anterior es cierto.

6. Se desea implementar un sistema de control de una válvula de presión en una central térmica, y usted determina emplear para ello el API de JRTS. Dado lo anterior, estima que necesita controlar de forma periódica la presión en la válvula, mientras que ocasionalmente inyecta *fuel-oil* en las turbinas de generación. En consecuencia, deberá utilizar

- a) hebras de T.R. periódicas y gestores de eventos asíncronos ligados a hebras de T.R. esporádicas.
- b) hebras de T.R. esporádicas y gestores de eventos asíncronos ligados a hebras de T.R. periódicas.
- c) únicamente gestores de eventos asíncronos.
- d) Ninguna de la anteriores es correcta, y basta utilizar hilos de T.R.

7. Es conocido que la anidación de bloqueos en lenguaje Java sobre dos recursos compartidos puede llevar a dos hebras, en el peor de los casos, a una situación de *deadlock*. Para evitar lo anterior podemos

- a) evitar el anidamiento de bloqueos mediante un monitor.
- b) no utilizar recursos compartidos.
- c) utilizar el API de alto nivel del lenguaje, que ofrece en tiempo de compilación la capacidad para detectar y eliminar estas situaciones.
- d) reducir a cero la probabilidad de aquellos flujos de ejecución que conlleven al interbloqueo, cambiando la arquitectura de la codificación

8. Cuando se utiliza programación CUDA, el programa principal que lanza el *kernel* CUDA:

- a) está sincronizado con parte de los *kernels*, pero no con todos.
- b) puede ejecutar código paralelo y actuar como **coprocesador** de la GPU.
- c) puede tener hebras ejecutándose sobre la CPU, pero deben estar en exclusión mutua con los hebras que se ejecutan en los *kernels* de la GPU.
- d) nada de lo anterior es cierto.


9. El uso de variables *volatile* en java

- a) garantiza al programador que los datos que albergan son accedidos de forma segura frente a hebras concurrentes.
- b) elimina las cache temporales de esa variable asociadas al motor de ejecución de las hebras que la utilizan

- b) permite garantizar código parcialmente correcto, pero no totalmente correcto.
  - d) ninguna de las anteriores es cierta.
10. El resultado de paralelizar una aplicación sobre una máquina de dos *cores* ofrece un *speedup* de 0.75. Esto significa que
- a) el código original es inherentemente paralelo.
  - b) el coeficiente de bloqueo de la aplicación está próximo a 1.
  - c) su diseño paralelo es ineficiente.
  - d) ninguna de las anteriores es correcta.
11. Cuando se utiliza STM en Java sobre Clojure:
- a) el delimitador sintáctico `dosync` gestiona el acceso a regiones críticas bajo transacciones.
  - b) los interbloqueos siguen sin poder evitarse.
  - c) puede haber inconsistencias de la información.
  - d) es necesario utilizar, también a nivel sintáctico, el delimitador `LockingTransaction.runInTransaction()`.
12. En el lenguaje C++ el desarrollo de un monitor supone:
- a) disponer de una única cola de espera por condición para las tareas.
  - b) una semántica de señalización de tipo SX.
  - c) es imposible usarlos. C++ no lo permite.
  - d) ninguna de las anteriores es correcta.
13. La implementación de la primitiva teórica de región crítica en C++
- a) es imposible.
  - b) es posible utilizando un objeto auxiliar que actúe como cerrojo.
  - c) es posible encerrando la región entre `lock.lock()` y `lock.unlock()` donde `lock` es un objeto de clase `mutex`.
  - d) ninguna de las anteriores son ciertas.
14. El uso de ejecutores de *pool* de *threads* en Java
- a) optimiza el rendimiento para cualquier número de hebras.
  - b) permite al programador centrarse exclusivamente en las tareas
  - c) imposibilita el uso de clases contenedoras autosincronizadas.
  - d) sólo tiene sentido en problemas y tareas con  $C_b = 0$ .
15. Se desea paralelizar un algoritmo para el que se ha determinado que  $C_b = 0$ , y que tiene que trabajar sobre una máquina con 8 *cores* lógicos, para procesar un vector de  $10^6$  componentes sin interdependencia entre ellas. Con estos datos, usted decide:
- a) utilizar un contenedor sincronizado para optimizar el rendimiento.



- b) utilizar 16 tareas con 16 sub-vectores disjuntos y procesarlas utilizando un objeto de clase `ThreadPoolExecutor` con `corePoolSize=4`.
  - c) utilizar utilizar 16 tareas con 16 subvectores y procesarlas utilizando un ejecutor de tamaño fijo mediante `newFixedThreadPool(16)`.
  - d) ninguna de las anteriores es cierta.
16. La reentrancia para código concurrente seguro en el lenguaje C++:
- a) funciona exactamente igual que en Java con cualquier tipo de cerrojo que se escoja.
  - b) funciona únicamente escogiendo cerrojos de clase `mutex`.
  - c) no es posible disponer de ella.
  - d) ninguna de las anteriores es cierta.
17. Usted debe determinar el valor de  $C_b$  óptimo para un aplicación con paralelismo a nivel de hebras que acaba de escribir. Para ello:
- a) busca el mínimo de la curva  $Tiempo = f(C_b)$  mediante experimentación, lo determina de forma aproximada, y deduce a partir de aquí cuántas hebras necesita tener en su aplicación. busca el mínimo de la curva  $Tiempo = f(C_b)$  mediante técnicas analíticas.
  - b) busca el máximo de la curva  $Tiempo = f(C_b)$  mediante experimentación, y lo determina de forma aproximada.
  - ☒ c) queda determinada por la tipología del problema.
  - d) no es posible determinar esto de forma adecuada sin un *benchmarking* profesional. Puesto que su código tiene latencias de E/S y de red, su experiencia como programador le dice que  $C_b \approx 0,5$ .
18. El uso de la interfaz `Future` junto con tareas `Callable`
- a) es igual que con la interfaz `Runnable`, cambiando el método `run()` por el método `call()`.
  - b) permite el acceso a los datos de retorno del método `call()` bajo bloques explícitos.
  - ☒ c) permite el acceso a los datos de retorno del método `call()` bajo bloques implícitos.
  - d) permita procesar los objetos que la implementan dándolas como parámetros a objetos de la clase `Thread`.
19. El multiprocesamiento simétrico, contemplado en la jerarquía de *Flynn* en el nivel MIMD
- a) engloba a arquitecturas *multicore*.
  - b) engloba a arquitecturas *multicore* y *manycore*.
  - c) engloba a arquitecturas *multicore* y *manycore*, a excepción los *clusters* de procesadores.

- 
- d) En realidad todo lo anterior es cierto. El multiprocesamiento simétrico es tan general como para englobar a todas las tipologías de máquinas que disponen de múltiples *cores* de ejecución, con independencia de su número y organización.
20. El anidamiento de múltiples operaciones `wait(S)` sobre diferentes semáforos realizadas por hebras diferentes:
- a) es algo que la especificación estándar del lenguaje Java no permite.
  - b) conlleva irrevocablemente a un *deadlock*, con independencia del lenguaje que soporte a los semáforos.
  - c) es algo que la especificación estándar del lenguaje Java no permite pero la de tiempo real sí.
  - d) se puede hacer con Java, para ambas especificaciones, y con cualquier lenguaje que soporte semáforos, siempre que respeten las ecuaciones de invariancia de los mismos.

#### 4. Cuestiones de Desarrollo Corto

Conteste a las preguntas que se le formulan en el espacio habilitado para ello. Deberá razonar o justificar su respuesta siempre que se le indique. **La ausencia del razonamiento o de la justificación invalidarán la respuesta al no ser esta completa.**

1. Utilizando el API estándar de Java para escribir un monitor, para lograr una adecuada sincronización es obligatorio utilizar el método `notifyAll()` cuando se desea enviar una señal a las tareas para las cuales se cumple una determinada condición. ¿Es esto también obligatorio si utilizamos el API de alto nivel? ¿Por qué? [1 punto]



2. Java implementa el concepto de región crítica mediante bloques de código `synchronized`. ¿Cómo lo hace C#?. Explíquelo, y escriba un corto código de ejemplo que ilustre la respuesta.



3. La Figura 1 ilustra el comportamiento de la memoria transaccional frente al modelo de cerrojos estándar para controlar el acceso en modo seguro a una sección crítica, cuando hay dos hebras en ejecución concurrente. Explíquela, destacando las ventajas que el modelo STM ofrece frente al modelo de cerrojos, explicando igualmente los inconvenientes que tiene. [1 punto] Escriba aquí su respuesta:





4. La Figura 2 ilustra el flujo de funcionamiento de una solución *manycore* basada en el uso del lenguaje C-CUDA sobre una GPU. Para cada una de las fases rotuladas, indique qué instrucción C-CUDA permite efectuarla. [1 punto] Escriba aquí su respuesta:

5. Explique brevemente el API de control de prioridades en la especificación JRTS frente al API de control de prioridades de Java estándar, mediante una comparativa tabular entre ambos, que recoja un mínimo de cuatro características a comparar.



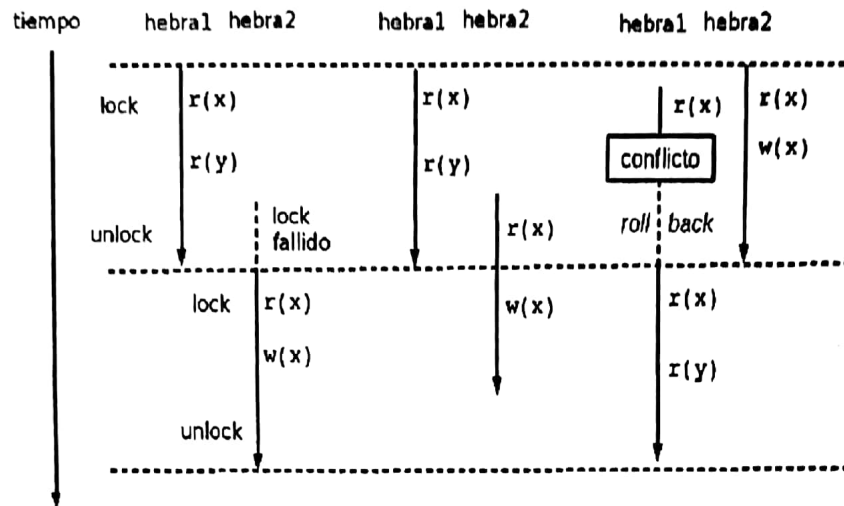


Figura 1: Transacciones vs. Cerrojos

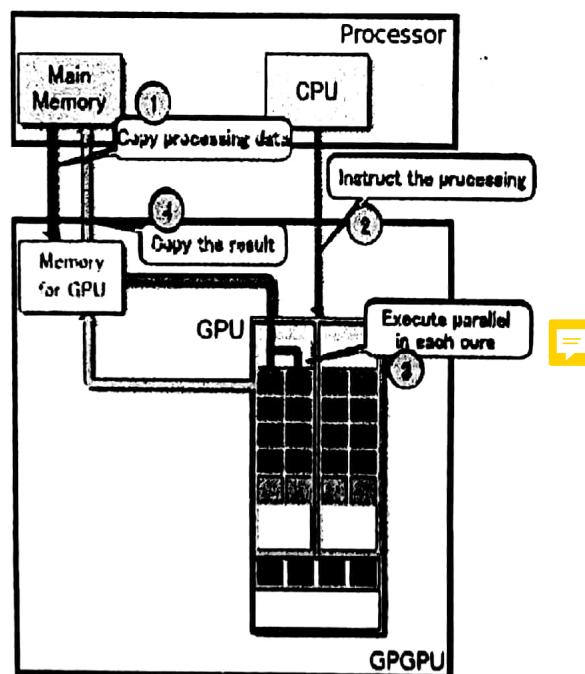


Figura 2: Flujo de Trabajo CUDA