Examen PCTR Junio 2017.pdf

blackmamba

Programación Concurrente y de Tiempo Real

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

Examen PCTR Junio 2017

Tests

1. **Suponga que 500 hebras comparten el acceso a un objeto común cada una a través de su propia referencia, y ejecuta el acceso a un método inv() de ese objeto que incrementa un contador que inicialmente vale cero. El programador protege el acceso haciendo que cada hebra ejecuta la llamada a inc() envolviendo la misma en un bloque de la forma sychronized(this) { ...miReferencia.inc(); ... }. El valor final del contador es:**

   A) 500
   B) 499
   C) Un número indeterminado entre el 0 y 499.
   D) Un número indeterminado entre el 1 y 500.

2. **En Java, los cerrojos de la clase ReentrantLock se utilizan:**

   A) Para tener regiones críticas de grano muy grueso.
   B) Para no utilizar regiones synchronized.
   C) Para poder utilizar variables de condición si hacen falta.
   D) Para optimizar los accesos a los datos que protegen.

3. **Una hebra**

   A) Puede estar detenida en un punto de su código por un bloqueo de exclusión mutua, mientras ejecuta código de una zona diferente.
   B) Puede acceder a varios objetos diferentes de forma concurrente.
   C) Puede ser procesada a través de un ejecutor.
   D) Ninguna de las anteriores es correcta.

4. **Usted debe desarrollar una aplicación bajo el framework RMI que controla un sistema de reserva de billetes de avión y sabe:**

   A) Que el sistema deberá ser concurrente.
   B) Que habrá que utilizar necesariamente control de exclusión mutua explícito en el lado del servidor.
   C) Que deberá crear una hebra de servicio por cada petición procedente de un cliente.
   D) Que utilizando generación dinámica de resguardos puede prescindir de utilizar un DNS local a la máquina que ejecuta el servidor.

5. **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 la clase Semaphore que inicializa a cero, y engloba todo el código del mismo bajo el par de instrucciones acquire() y reléase(). La**

sección crítica en todos los casos implica incrementar una variable del 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) 5, porque dos hebras hacen su incremento y otra no, ya que queda bloqueada.
D) El programa realmente no imprime nada porque queda bloqueado a causa de las hebras.

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 las 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**

opi

A) Evitar el anidamiento de bloqueos mediante un monitor.
B) Anidar transacciones.
C) Verificar de manera formal las propiedades de corrección de nuestro código.
D) Utilizar el API de alto nivel del lenguaje.

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

A) Está sincronizado con parte de los kernels, per no con todos.
B) No puede ejecutar código paralelo y actuar como coprocesador de la …
C) Puede tener hebras sobre la CPU, pero deben estar en exclusión mutua con los kernels de la GPU.
D) Nada de lo anterior es cierto.

9. **El uso de ejecutores en lenguaje Java supone**

A) Optimizar siempre el rendimiento de la aplicación.

B) Procesar indistintamente tareas Callable o Runnable a través de ellos.
C) No controlar la sincronización, pues son los ejecutores quiénes controlan el ciclo de la vida de las tareas.
D) Asegurar la exclusión mutua de las tareas sobre los datos compartidos.

10. El uso de variables volatile en el ámbito de la concurrencia

A) Nos garantiza que los datos son accedidos de forma segura.
B) Sirven solo para implementar algoritmos de control de exclusión mutua con espera ocupada.
C) Únicamente tiene sentido cuando esas variables son estáticas.
D) Ninguna de las anteriores es cierta.

11. El resultado de paralelizar una aplicación sobre una máquina de ocho cores ofrece un speedup de 2.75. Esto significa que

A) El código original es inherentemente paralelo.
B) El coeficiente de bloqueo de la aplicación está próximo a cero.
C) No se han utilizado ejecutores en el procesamiento de las hebras paralelas.
D) Ninguna de las anteriores es correcta.

12. Cuando se utiliza STM en Java sobre Clojure:

A) Desaparece la necesidad de controlar la exclusión mutua.
B) Los interbloqueos siguen sin poder evitarse.
C) Tenemos garantizada la consistencia de la información.
D) Tenemos escrituras concurrentes y lecturas no concurrentes.

13. En el lenguaje C++ el desarrollo de un monitor exige:

A) El uso de sincronización call_once-
B) El uso exclusivo de recursive_mutex.
C) Utilizar unique_lock y variables de condición si son necesarias.
D) Ninguna de las anteriores es correcta.

14. La implementación del concepto teórico de región crítica en java.

A) Exige que los métodos sean synchronized.
B) Exige utilizar un objeto auxiliar que actúe como cerrojo.
C) Exige acotar con synchronized(this) la región.
D) Todas las anteriores son ciertas.

15. Los contenedores autosincronizados de Java

A) Optimizan el rendimiento de los productores-consumidores.
B) Elevan el nivel de abstracción para el programador.
C) Pueden manejarse desde tareas gestionadas por ejecutores.
D) Todas las anteriores son ciertas.

16. Se desea paralelizar un algoritmo para el que se ha determinado que Cb = 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 objetos de clase ThreadPoolExecutor con corePoolSize=4.
C) Utilizar 16 tareas con 16 sub-vectores y procesarlas utilizando un ejecutor de tamaño fijo mediante newFixedThreadPool(16).
D) Ninguna de las anteriores es cierta.

17. La reentrancia para códico synchronized en el lenguaje Java

A) Tiene sentido cuando un objeto concreto es un monitor.
B) Funciona cuando el código synchronized reentrante es recursivo.
C) Funciona cuando un objeto concreto tiene código synchronized reentrante y código no synchronized.
D) Todas las anteriores son ciertas.

18. Usted debe determinar el valor de Cb óptimo para una aplicación con paralelismos a nivel de hebras que acaba de escribir. Para ello:

A) Busca el mínimo de la curva Tiempo = f(Cb) mediante técnicas analíticas.
B) Busca el máximo de la curva Tiempo = f(Cb) mediante experimentación y lo determina de forma aproximada.
C) Busca el mínimo de la curva Tiempo = f(Cb) mediante experimentación lo determina de forma aproximada, y deduce a partir de aquí cuántas hebras necesita tener en su aplicación.
D) No es posible determinar esto de forma adecuada sin un benchmark profesional. Puesto que su código tiene latencias de E/S y de red, su experiencia como programador le dice que Cb es aproximado a 0,5.

19. Los monitores escritos en lenguaje C#

A) Permiten varias colas de espera por condición.
B) No permiten colas de espera por condición; funcionan únicamente para proteger recursos compartidos bajo control de exclusión mutua.
C) Funcionan igual que los monitores en Java con el API de alto nivel.
D) Ninguna de las anteriores es cierta.

20. Los hilos daemon de Java

A) Ejecutan tareas de alto y bajo nivel que no terminan nunca.
B) Tienen sentido en sistemas reactivos.
C) Permanecen activos hasta que se apaga la máquina.
D) Solo hay realmente un hilo daemon correspondiente al garbage co*

Problemas

1.

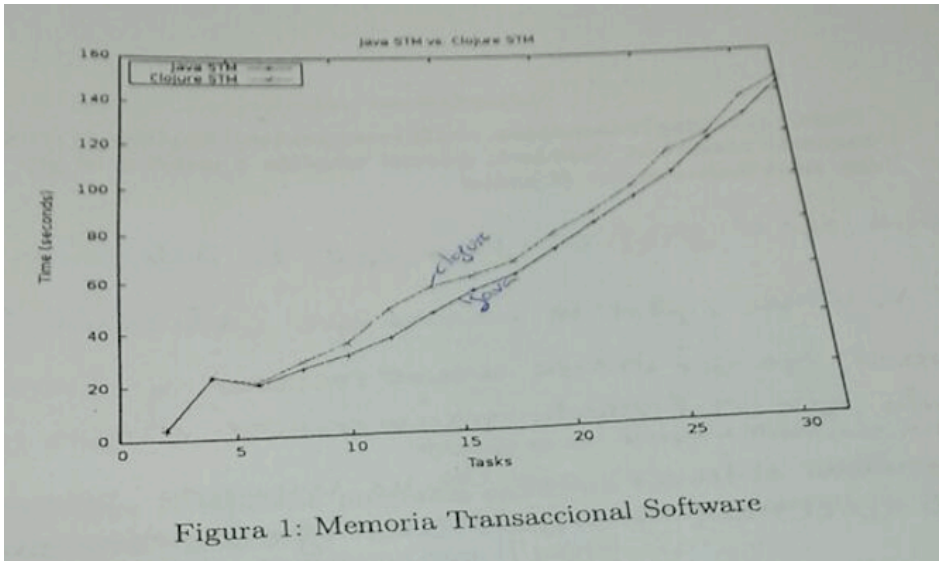

Figura 1: Memoria Transaccional Software

Se aprecia en el gráfico, hace que la diferencia entre las transacciones sean mínima, pero se observa que utilizando Java sobre ------ cupo menor ya que utiliza menos recursos y el acceso a la memoria es más rápido que utilizando transacciones Clojure(dosync), ya que la exclusión mutua mediante bloques no es eficiente. Las dos empiezan a la vez, pero llega un momento en que se ha distanciado en el tiempo, siendo más eficiente Java sobre Clojure.

2. En un lenguaje con soporte a STM (Software Transactional Memory), el algoritmo básico de procesamiento de una transacción es fundamental. Escriba un pseudocódigo que ilustre el funcionamiento de ese algoritmo. [1 punto]

¿Cómo funciona entonces el modelo transaccional frente al modelo estándar de exclusión mutua?

El modelo transaccional es un mecanismo de control de la concurrencia análoga al sistema de transacciones de las bases de datos para controlar el acceso a la memoria compartida mientras que la exclusión mutua consiste en evitar la condición de concurso sobre un recurso compartido forzando la ejecución atómica de las secciones críticas de las entidades concurrentes que lo usan. Elimina entrelazado.

¿Qué precio para el modelo STM para liberarnos del control de la exclusión mutua mediante bloqueos?

Es más lenta y menos eficiente en espacio. Puede llevar a inconsistencia si implican cualquier estructura fuera de control de la transacción.

(Escriba su algoritmo en la plantilla, usando las líneas que necesite)

Algoritmo STM:

1. Inicio

2. Hacer una copia privada de los datos compartidos.

3. Hacer actualizaciones en la copia privada.

4. Ahora:
    4.1 Si datos compartidos no modificados->actualizar datos compartidos con copia privada y goto(Fin)
    4.2 Si hay conflictos, descartar copia privada y goto(Inicio)

5. Fin

6.

3. Considere el siguiente programa escrito en C++11-14. Utiliza objetos atómicos correspondientes al API de concurrencia. Complételo, de manera que siempre se imprima cero. [1 punto]

```
#include <iostream>
#include <vector>
#include <thread>
#include <atomic>
struct ContadorAtomico
{
    std::atomic<int> valor;
    void incremento(){ ++valor; }
    void decremento(){ --valor; }
    int obtener(){ return valor.load(); }
};

int main()
```

```
{
    ContadorAtomico contadoratomico;
    contadoratomico.valor.store(0);
    std::vector<std::thread> hilos;
    for(int i = 0; i < 3; ++i)
    {
        hilos.push_back(std::thread([&contadorato
    mico](){
                for(int i = 0; i < 100; ++i)
                {
                  contadoratomico.incremento();
                }
            }));
            }
for(int i = 0; i < 3; ++i)
{
    hilos.push_back(std::thread([&contadoratomico
]
(){
for(int i = 0; i < 100; ++i){
contadoratomico.decremento();
}
}));
}
for(auto& thread : hilos){ thread.join(); }
std::cout << contadoratomico.obtener() <<
std::endl;
return 0;
}
```

*ROJO: Soluciones que te pedían en el examen. Sacadas del seminario de C++.

4. Considere el siguiente código escrito en c#. Explique el comportamiento y la salida impresa que produce, justificando la respuesta. [1 punto]

```
using System;
using System.Threading;

  public class PrintingThread{
    private static object threadLock = new object();
    private static int val = 1000000;

    public void tJob(){
     lock (threadLock){
       Console.WriteLine();
       Console.WriteLine(Thread.CurrentThread.Name);
       val++;
       Console.WriteLine("El cocherito lere...: ",
              Thread.CurrentThread.Name);
      for (int i = 0; i < 100; i++){
Random r = new Random();
Thread.Sleep(1000 * r.Next(2));
Console.Write(i+", ");
}
Console.WriteLine();
}
}

public static void Main(){
PrintingThread pt = new PrintingThread();
val++;
Thread[] ts = new Thread[10];
for (int i = 0; i < 10; i++){
ts[i] = new Thread(new ThreadStart(pt.tJob));
ts[i].Name = string.Format("Hebra currante [{0}]", i);
}
foreach (Thread t in ts) t.Start();
lock (threadLock){val++; Monitor.Wait(threadLock);}
foreach (Thread t in ts) t.Join();
Console.WriteLine(val++);
Console.ReadLine();
}
}
```

Solución:

Se crean 10 hilos que ejecutan bajo exclusión mutua el método tJob(). En primer lugar, la variable estática val se incrementa en el main, después se muestra por pantalla "Hebra currante x", donde x será un número del 0 al 9 sin repetirse, cada hilo volverá a incrementar val, seguidamente se mostrará por pantalla 10 veces lo mismo que antes pero con "El cochecito lere … : " de lante, después se mostrará los números del 1 al 100 repetidos 10 veces cada uno. Finalmente se volverá a incrementar val y se bloquea el programa. Por lo que este no termina y no muestra ni incrementa por última vez el valor de val.

5. El siguiente código de programa se considera bien escrito si ofrece una salida impresa igual a 110 o a 50. Se pide analizar, desde el punto de vista de la salida impresa, si el programa es correcto o no. Justifique su respues. [1 punto]

```
program output
var x: integer;

process P1;
begin
 x:=x+10;
end;

                    process P2;
                    begin
                    if x>100
                      then write(x);
                      else write(x-50);
                    end;

begin (*principal*)
x:=100;
  cobegin
    P1; P2;
  coend;
end.
```

Solución:

Al no preservar la exclusión mutua pueden darse varios casos:

- Caso 1: Se incrementa en 10 la variable x primero por lo que el proceso P2 entraría en el if y mostraría por pantalla 110, por lo que según el enunciado estaría bien escrito.

- Caso 2: Entra primero el proceso P2 y al no cumplirse el if, pasa directamente al else que decrementa en 50 la variable, mostrando 50 por pantalla, por lo que según el enunciado estaría bien escrito.

- Caso 3: Igual que el caso 2 pero antes de decrementar 50, el proceso P1 incrementa 10, por lo que no estaría bien escrito.