ExamenFebreroCompleto.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

1. Los hilos creados por otro hilo
   a) No necesitan implementar un acceso en exclusión mutua para acceder a los recursos compartidos.
   b) No necesitan implementar un acceso en exclusión mutua para acceder a los recursos compartidos, siempre y cuando el hilo padre acceda a dichos recursos en exclusión mutua.
   c) Siempre deben acceder a los recursos compartidos en exclusión mutua.
   d) Ninguna de las respuestas anteriores es correcta.

2. Los hilos creados por otro hilo pueden compartir información con el hilo padre
   a) A través de variables estáticas definidas en el hilo padre o el hilo hijo.
   b) Pasando por referencia a los hilos hijos las variables compartidas.
   c) A través de un archivo.
   d) Todas las respuestas son correctas.

3. Cada hilo comparte con su hilo padre
   a) El contador de programa.
   b) El contador de programa y la pila de llamada.
   c) La pila de llamadas y las variables estáticas.
   d) Ninguna de las respuestas anteriores es correcta

4. Un hilo creado por otro
   a) Es siempre reentrante, por lo que no se bloquea al realizar llamadas recursivas.
   b) Nunca se bloquea al tratar de adquirir un cerrojo bloqueado por el hilo padre, pues comparten el mismo espacio de direcciones.
   c) Debe esperar para adquirir el cerrojo de los métodos sincronizados del (falta terminación)

5. El entrelazado de instrucciones
   a) Solo se produce entre hilos, pero no entre procesos.
   b) Solo se produce en arquitecturas multinúcleo
   c) Se produce en sistemas concurrentes y paralelos.
   d) D) no se produce cuando se ejecutan instrucciones del sistema operativo.

6. ¿Cuál es el número máximo de clases que pueden heredar de la clase Thread en una aplicación escrita en Java?
   a) No hay límite.
   b) Solo una clase puede heredar de Thread. El resto debe implementar la interfaz Runnable.
   c) Solo una, si se desea compartir información entre los hilos a través de variables estáticas.
   d) Todas las clases deben heredar de la clase Thread.

7. El acceso en exclusión mutua a los recursos compartidos
   a) Produce aplicaciones más eficientes.
   b) Produce aplicaciones más eficientes cuando se utilizan primitivas de sincronización reentrantes.
   c) Produce aplicaciones menos eficientes.

d) No afecta al rendimiento de las aplicaciones.

8. Si se aplica en Java la instrucción H.notify()
   a) La señal se perderá si el hilo H no se encuentra bloqueado.
   b) La señal se perderá si no hay ningún hilo en el Wait-Set asociado a la variable H.
   c) Se incrementará en una unidad el número de veces que se puede realizar la operación H.wait() sin que el hilo se quede bloqueado.
   d) Ninguna de las respuestas anteriores es correcta.

9. Dadas dos variables A y B en Java, que apuntan a la misma dirección de memoria (A=B;),
   a) Cada una de ellas dispondrá de su propio Wait-Set asociado.
   b) Cada una de ellas dispondrá de su propio Wait-Set asociado si ambas son de tipo Object.
   c) Ambas comparten el mismo Wait-Set asociado.
   d) Ambas variables comparten el mismo Wait-Set, pero solo si ambas son de tipo Object.

10. Los métodos bloqueantes que se pueden aplicar sobre un hilo en Java son
   a) Join, suspend.
   b) Join, wait, yield.
   c) Sleep, join, wait.
   d) Sleep, join, wait, suspend.

11. El método shutDownNow() de un objeto de la calseExecutorService
   a) Es un método equivalente a invocar al método shutdown seguido de una llamada al método awaitTermination().
   b) Es un método bloqueantes equivalente a invocar al método shutdown() seguido de un bucle que finalice cuando el método isTerminates() devuelva un valor cierto.
   c) Es un método bloqueante que espera a la finalización de los hilos que actualmente se encuentran ejecutándose en el pool de hilos, pero no al resto de hilos que se encuentran pendientes de ejecución.
   d) Es un método que finaliza los hilos del pool de forma inmediata, sin esperar a la finalización de los que actualmente se encuentran ejecutándose.

12. Los hilos que se ejecutan en un pool de hilos no requieren de un acceso en exclusión mutua a los recursos compartidos
   a) Cuando se utiliza un pool de tipo CachedThreadPool.
   b) Cuando se utiliza un pool de tipo FixedThreadPool.
   c) Cuando se utiliza un pool de tipo SingleThreadExecutor.
   d) Ninguna de las respuestas anteriores es correcta.

13. Un hilo
   a) Puede estar ejecutando el protocolo de entrada a la sección critica mientras otro se encuentra ejecutando instrucciones del protocolo de salida.
   b) No puede estar ejecutando el protocolo de entrada a la sección critica mientras otro se encuentra ejecutando instrucciones del protocolo de salida.

c) Puede encontrarse ejecutando instrucciones de la sección critica mientras otro hilo ejecuta las del protocolo de salida.
d) Puede estar ejecutando instrucciones del protocolo de salida al mismo tiempo que otro hilo también las ejecuta.

14. Los protocolos de exclusión mutua basados en soluciones software
    a) Solo funcionan correctamente si se accede en exclusión mutua a las variables compartidas.
    b) Solo funcionan correctamente si se accede por turnos a las variables compartidas.
    c) Son más eficientes que los basados en soluciones hardware.
    d) Ninguna de las respuestas anteriores es correcta.

15. Cuando se utilizan métodos sincronizados para implementar un monitor en Java, todos sus métodos
    a) Deben ser sincronizados.
    b) Deben ser públicos y sincronizados.
    c) Deben ser públicos, estáticos y sincronizados.
    d) Que sean públicos deben ser sincronizados.

16. Los monitores en C++
    a) Solo pueden implementarse mediante el uso de las instancias de tipo resursive_mutex
    b) Pueden implementarse mediante el uso de instancias de tipo unique_lock.
    c) Solo pueden implementarse mediante el uso de primitivas reentrantes.
    d) Puede implementarse mediante el uso de la primitiva de sincronización call_once.

17. Un método sincronizado en Java
    a) Puede contener un bloque sincronizado que utilice la variable this como cerrojo.
    b) Puede contener un bloque sincronizado que utilice una variable estática como cerrojo.
    c) No puede contener un bloque sincronizado
    d) Las respuestas a) y b) son correctas.

18. En el modelo de memoria CUDA
    a) Las regiones de memoria per-block requieren control de la exclusión mutua.
    b) Existen regiones de memoria que no requieren control de exclusión mutua.
    c) La región de memoria global requiere control de exclusión mutua.
    d) Todas las respuestas anteriores son correctas.

19. En la especificación JRTS la interfaz Schedulable
    a) Modela las clases RealTimeThread y NoHeapRealTimeThread.
    b) Hereda de la interfaz Runnable de Java.
    c) Modela de la clase AsyncEventHandler.
    d) Todas las respuestas anteriores son correctas.

20. En el framework RMI el servidor
    a) Puede efectuar la llamada a un método del cliente, siempre y cuando el cliente quede asociado a un nombre de servicio en el registro RMI local.

b) Puede efectuar la llamada a un método del cliente, siempre y cuando este último implemente la interfaz Serializable.
c) Solo puede disponer de métodos públicos.
d) Ninguna de las respuestas anteriores es correcta.

1. Considere el siguiente código escrito en C#. Explique el comportamiento que tiene y la salida impresa que produce, justificando su respuesta.

```
using System;
using System.Threading;
    public class PrintingThread{
        private static object threadLock = new object();
        private static int val           = 1000000;
        public void threadJob(){
            lock(threadLock){
                Console.WriteLine();
                Console.WriteLine(Thread.CurrentThread.Name);
                val++;
                Console.WriteLine("Cuenta de la hebra: ", 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.threadJob));
            ts[i].Name = string.Format("Hebra currante[{0}]", i);
        }
        //Ahora un iterador para recorrer el array ts
        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();
        }
    }
```

En la función ThreadJob, se imprimen una serie de mensajes dentro de un bloque sincronizado (haciendo uso de la palabra reservada lock). El bloque sincronizado equivaldría a hacer un synchronized.

En el método principal, se realiza la operación Monitor.wait(), bloqueando al hilo principal. El hilo principal se va a quedar ahí bloqueado. Por este motivo, solamente se imprimen los mensajes del hilo hijo pero no se escribe el mensaje del hilo padre.

2. Considere el siguiente código escrito en C++. Explique el comportamiento que tiene y la salida impresa que produce, justificando su respuesta.

```cpp
#include <iostream>
#include <thread>
#include <vector>
#include <atomic>
#include <mutex>
using namespace std;
mutex Ingrid_Bergman;
struct CuentaAtomica{
    atomic<int> val;
    void inc()
    {
        ++val;
        Ingrid_Bergman.lock();
    }
    int verValor(){return(val.load());}
};

int main(){
    vector<thread> hilos;
    int nHilos = 100;
    CuentaAtomica contador;
    for(int i = 0; i < nHilos; ++i){
        hilos.push_back(thread([&contador](){for(int i = 0; i < 1000; i++) contador.inc();}))}
    for(auto& thread : hilos) {thread.join();}

    cout << contador.verValor();
    return 0;
}
```

//Todos lo tienen correcto, por lo que no se corrige

El siguiente código se bloquearía , ya que en el método inc() se adquiere el cerrojo mediante el método lock() pero no lo libera ya que no se utiliza el método unlock(), por tanto no se accede a la variable bajo exclusión mutua.

3. Considere el siguiente programa en Java que hace uso de cerrojos synchronized y de una barrera cíclica. Analícelo desde el punto de la vista de las propiedades de corrección de seguridad y vivacidad bajo los escenarios que se le plantean, indicando también la salida impresa resultante.

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

public class MargotRobbie extends Thread{
    public static Integer I = new Integer(1000);
    public static int     j = 0;
    private       int tThread;
    public static CyclicBarrier c;

    public MargotRobbie(int tThread){
        this.tThread = tThread;
        synchronized(I){j++;};
    }

    private void AmberHeard() throes InterruptedException{
    synchronized(this){this.await();}
    }

    public void run(){
        switch(tThread){
            case 0: for(int x = 0; x < 1000000; x++){
                        synchronized(I){
                            try{AmberHeard();}
                            catch(InterruptedException ex){}
                            j++;
            }}
            try {c.await();
                    } catch(InterruptedException ex){}
            catch(BrokenBarrierException ex){}
            break;

            case 1: for(int y = 0; y < 1000000; y++){
                     synchronized(I){
                     j++;
            }}
            try {c.await();
                     } catch (InterruptedException ex){}
            catch (BrokenBarrierException ex){}
            break;
        }
    }

    public static void main(String[] args) throws Exception{
        MargotRobbie A = new MargotRobbie(1);
        MargotRobbie B = new MargotRobbie(0);
        c              = new CyclicBarrier(Integer.parseInt(args[0]));
        B.start(); A.start();
        System.out.println(j);
    }
}
```

ESCENARIOS:

- El programa se ejecuta con $java E011516 0
- El programa se ejecuta con $java E011516 1
- El programa se ejecuta con $java E011516 2

En este caso, se pueden lanzar dos hilos, uno que tiene como parámetro 0 y otro que tiene como parámetro 1. Hay un switch dentro del método run de tal forma que el hilo que tiene el parámetro 0 va a entrar dentro del bloque sincronizado sobre la variable “Y”, echando el cerrojo sobre la variable “Y”. Pero al llamar a la función Amberheard llama a un método sincronizado sobre this, hace una llamada al wait y se bloquea, liberando el cerrojo sobre this, pero no sobre “Y”. Entonces si el otro hilo intenta entrar en el método MargotRobbie, “Y” sigue bloqueado, quedándose bloqueado el hilo.

De esta forma ambos hilos se quedan bloqueados. Si el orden de los hilos fuera al revés, el primer hilo si pasaría el synchronized y haría tantos incrementos como le fuera posible de la variable j, pero el primer hilo (argumento 0) siempre se queda bloqueado.

Como en el método principal no tiene ningún join(), se imprimiría por pantalla el valor que tenga la variable j en ese momento.

La primera llamada con 0 no tiene sentido, de hecho provoca una excepción porque no se puede tener una barrera inicializada a 0.

En el segundo caso si pasaría de la primera barrera en c.await() (si no se ha quedado bloqueado antes).

En el tercer caso el hilo se quedaría bloqueado siempre.