Exclusión Mutua y Sincronización en Java (I)

Exclusión Mutua en Java

Modificador synchronized

Los objetos Java disponen de un “cerrojo” o sistema de bloqueo que es activado mediante el modificador synchronized.

Si un hilo invoca código de un objeto synchronized “echa el cerrojo”

Cualquier otro hilo no podrá acceder hasta que el primero lo libere

Se puede utilizar sobre

- Sobre Métodos
- Sobre un bloque de código
- Sobre objetos

Exclusión Mutua y Sincronización en Java (II)

- Exclusión Mutua en Java
  - Sobre Métodos

```
public synchronized void met1() {
        // código synchronized para este objeto
}
```

  - Sobre un bloque de código

```
...
synchronized(this) {
        // código synchronized para este objeto
} ...
```

  - Sobre otros objetos

```
...
synchronized(obj) {
        // código synchronized para objeto obj
} ...
```

Exclusión Mutua y Sincronización en Java (III)

Exclusión Mutua en Java

Simulación de un Monitor

```
class ejMonitor {
        // Atributos privados

        // Métodos Synchronized públicos
        public synchronized void met1() {
                // código synchronized
        }
        ...
        public synchronized void metN() {
                // código synchronized
        }
}
```

Exclusión Mutua y Sincronización en Java (IV)

Sincronización en Java

Los objetos en Java disponen de un mecanismo parecido (aunque con diferencias) a las variables de condición de los Monitores

- Cada objeto sólo dispone de una variable de condición, denominada conjunto de espera (wait set).
- Un hilo que haya adquirido el “cerrojo” del objeto puede detenerse en ella. Liberando el cerrojo.
- Los hilos detenidos en una espera, cuando son seleccionados deben competir, de nuevo, por el “cerrojo”.

Bloqueo en el conjunto de espera: wait();

```
...
try {  wait();
} catchg (InterruptedException e) { }
...
```

Activación de un hilo del conjunto de espera: notify() o notifyAll()

Exclusión Mutua y Sincronización en Java (V)

Sincronización en Java

Uso tradicional

```
while ( ! cond ) {
        try {
            wait();
        }catch(InterruptedException e) {}
}
```

try-catch:

el método wait puede lanzar un excepción en caso de producirse una interrupción.

while (en lugar de if):

- Los hilos deben volver a adquirir el cerrojo
- Existe una única variable de condición

Exclusión Mutua y Sincronización en Java (VI)

Simulación de múltiples variables de condición

```
class  EjMultCondiciones  {

// Crear un objeto por cada variable de condición
Object c1 = new Object();
boolean cond1 = false;

public void met1() { // Los métodos no serán sinchronized

    synchronized(c1) {
        while ( ! cond1 ) {
            try  {
                c1.wait();
            }catch(InterruptedException e) {}
        }
        synchronized(this) {
            // Código a ejecutar en Exclusión Mutua
            //  cuando se cumpla la condición
        }
    }

}
```

otro hilo desde otro método

c1.notify()

Exclusión Mutua y Sincronización en Java (VII)

Diferencias entre Monitores y el Modelo Java

Semántica de c.resume() vs notify()

Monitor: Un proceso despertado tiene preferencia sobre otros que desean acceder al Monitor

Java: Un hilo despertado debe volver a adquirir el cerrojo, no garantizándose su continuación inmediata

Objetos vs Variables de Condición

Monitor: Facilidad de utilización de múltiples variables de condición.

Java: Un objeto sólo dispone de un conjunto de espera

La simulación de múltiples conjuntos de espera requiere gran habilidad del programador

Exclusión Mutua y Sincronización en Java (VIII)

- Ejemplo: Simulación de Semáforos en Java

```
class semaforo {
        private int cont;

        public semaforo(int valorInicial) {
                cont = valorInicial;
        }

        public synchronized void jWait() {
                while(cont==0) {
                        try {  wait();
                        }
                        catch(InterruptedException e) {};
                }
                cont=cont-1;
        }

        public synchronized void jSignal() {
                cont=cont+1;
                notify();
        }

}
```

Exclusión Mutua y Sincronización en Java (y IX)

- Ejemplo: Productor-Consumidor en Java

```
class Buffer {
    static final int MaxBuff = 5;
    private int alm[] = new int[MaxBuff];
    private int ent, sal, cont;

    public buffer() {
            ent=0; sal=0; cont=0;
    }

    public synchronized void poner(int dato) {
            while(cont==MaxBuff) {
                try { wait();
                }catch(InterruptedException e) {}
            }
            alm[ent] = dato;
            ent = (ent + 1) % MaxBuff;
            cont++;
            notifyAll();
    }
```

```
    public synchronized int coger() {
            int dato;
            while(cont==0) {
                try { wait();
                }catch(InterruptedException e)
                  {}
            }
            dato = alm[sal];
            sal = (sal + 1) % MaxBuff;
            cont--;
            notifyAll();
            return dato;
    }
```