Análisis y diseño de software

dit
UPM

Tema 3: Concurrencia /A to B /java

José A. Mañas

11.2.2017

ETSIT UPM

caso de estudio

- Una thread A quiere pasarle un datos a otra thread B
  - usando un objeto compartido

solución 0: no funciona

```
public class A extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      dato.setS(“mal");
   }
```

```
public class B extends Thread

   @Override
   public void run() {
      System.out.println(dato.getS());
   }
```

```
Dato dato = new Dato();
Thread A = new A(dato);
Thread B = new B(dato);
A.start();
B.start();
```

```
public class Dato {
   private String s = null;

   public String getS() { return s; }

   public void setS(String s) { this.s = s; }
}
```

solución: monitor + wait condicional

```
public class Am extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      dato.setS(“monitor");
   }
```

```
public class Bm extends Thread

   @Override
   public void run() {
      System.out.println(dato.getS());
   }
```

```
DatoM dato = new DatoM();
Thread A = new Am(dato);
Thread B = new Bm(dato);
A.start();
B.start();
```

```
public class DatoM {
   private String s = null;

   public synchronized String getS() {
      while (s == null) waiting();
      return s;
   }

   public synchronized void setS(String s) {
      this.s = s;
      notifyAll();
   }
```

solución: semáforo

```
public class As extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      dato.setS(“semaforo");
   }
```

```
public class Bs extends Thread

   @Override
   public void run() {
      System.out.println(dato.getS());
   }
```

```
public class DatoS {
 private final Semaphore semaphore = new Semaphore(0);
   private String s = null;

 public String getS() {
       try { semaphore.acquire();
       } catch (InterruptedException ignored) {  }
       return s;
   }

   public void setS(String s) {
       this.s = s;
       semaphore.release();
   }
```

```
DatoS dato = new DatoS();
Thread A = new A1(dato);
Thread B = new B1(dato);
A.start();
B.start();
```

A then B

solución: busy wait

```
public class A2 extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      dato.setS(“busy wait");
   }
```

```
public class B2 extends Thread

 @Override
   public void run() {
      String s;
      while (true) {
         s = dato.getS();
         if (s != null)
            break;
         Nap.sleep(50);
      }
      System.out.println(dato.getS());
   }
```

```
Dato dato = new Dato();
Thread A = new A2(dato);
Thread B = new B2(dato);
A.start();
B.start();
```

```
public class Dato {
   private String s = null;

   public String getS() { return s; }

   public void setS(String s) { this.s = s; }
}
```

solución: CAS

```
public class A3 extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      dato.setS(“CAS");
   }
```

```
public class B3 extends Thread

   @Override
   public void run() {
      System.out.println(dato.getS());
   }
```

```
public class Dato3 {
   private AtomicInteger state = new AtomicInteger(0);
   private String s = null;

   public void setS(String s) {
      this.s = s;
      state.incrementAndGet();
   }

   public String getS() {
      while (!state.compareAndSet(1, 2))
         Nap.sleep(50);
      return s;
   }
}
```

```
Dato3 dato = new Dato3();
Thread A = new A3(dato);
Thread B = new B3(dato);
A.start();
B.start();
```

A then B

CAS

- wait-free
- primitiva atómica en la CPU
  - Intel x86: [lock] cmpxchg reg, reg/mem

simulated in java

```
synchronized int compareAndSwap(int expectedValue, int newValue) {
        int oldValue = value;
        if (value == expectedValue)
            value = newValue;
        return oldValue;
}
```

event-driven concurrency

- La receptora, B,
  - se apunta como parte interesada
  - y espera sin hacer nada
- La emisora, A,
  - calcula el valor
  - manda el dato
  - avisa a B de que hay un dato listo

solución 4: event-driven

A

```
@Override
  public void run() {
     Nap.sleep(500);
     dato.setS("event driven");
     event.set();
  }
```

B

```
@Override
  public void run() {
     event.waiting();
     System.out.println(dato.getS());
  }
```

```
MyEvent event = new MyEvent();
Dato dato = new Dato();
Thread A = new AE(dato, event);
Thread B = new BE(dato, event);
A.start();
B.start();
```

```
public class MyEvent {
   private boolean state = false;

   synchronized void set() {
      state = true;
      notifyAll();
   }

   synchronized void waiting() {
      while (!state)
         try { wait(); } catch (Exception ignored) { }
   }
}
```

solución 4: event-driven

```
public class A4 extends Thread

   @Override
   public void run() {
      Nap.sleep(500);
      channel.setS("hola 4");
   }
```

```
public class B4 implements Runnable

   @Override
   public void run() {
      System.out.println(channel.getS());
   }
```

```
Channel channel = new Channel(
Thread A = new A4(channel);
Runnable B = new B4(channel);
channel.setListener(B);

A.start();
```

```
public class Channel {
   private String s = null;
   private Runnable listener;

   public void setListener(Runnable listener) {
      this.listener = listener;
   }

   public String getS() { return s; }

   public void setS(String s) {
      this.s = s;
      listener.run();
   }
}
```

java UI - swing

```
MyListener listener = new MyListener();

JButton button = ...;
button.addActionListener(listener);
```

```
class MyListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        // do whatever you want here ...
    }
}
```