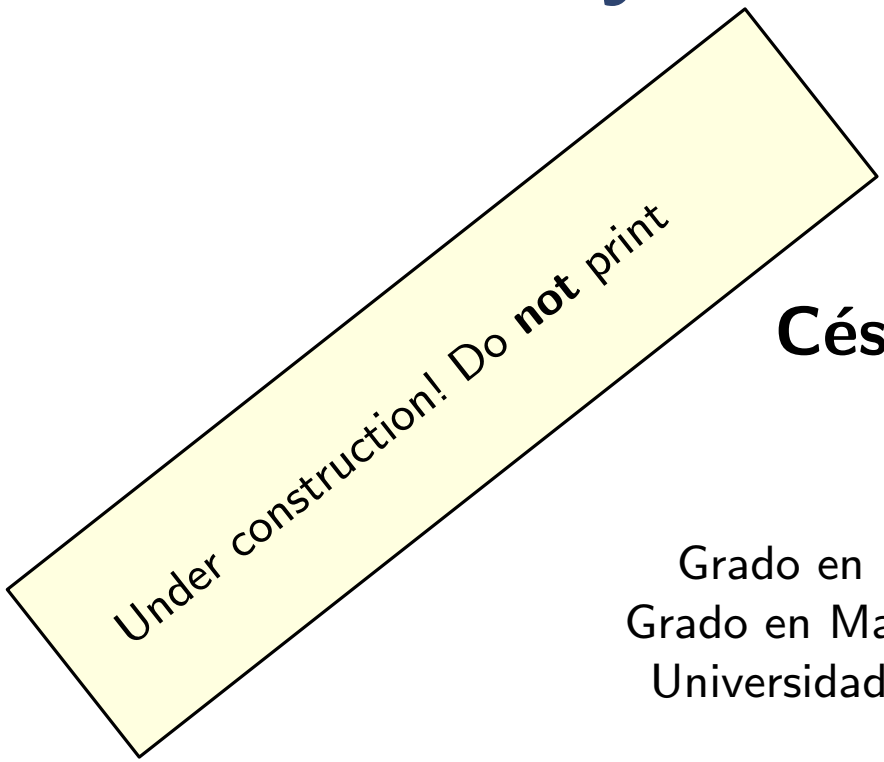
L9: Busy-wait (correct solutions)



César Sánchez

Grado en Ingeniería Informática
Grado en Matemáticas e Informática
Universidad Politécnica de Madrid

Wed, 4-March-2015

Mapa Conceptual

$$\text{Concurrency} = \text{Simultaneous} + \text{Nondeterminism} + \text{Interaction}$$

$$\text{Interaction} = \text{Communication} \mid \text{Synchronization}$$

$$\text{Synchronization} = \text{Mutual Exclusion} \mid \begin{array}{l}\text{Conditional}\\ \text{Synchronization}\end{array}$$

- Terminology:

atomic	race condition
interleaving	busy-wait
mutual exclusion	critical section
deadlock	livelock
liveness	

HW3: Mutex with Busy-Wait

3. Garantizar exclusión mutua con espera activa

Este ejercicio consiste en evitar la condición de carrera que se produjo en el ejercicio anterior. Para ello supondremos la existencia de **sólo dos procesos**, que simultáneamente ejecutan sendos bucles de N pasos incrementando y decrementando, respectivamente, en cada paso una variable compartida (la operación de incremento y la de decremento sobre esa misma variable compartida son secciones críticas). El objetivo es evitar que mientras un proceso modifica la variable el otro haga lo mismo (propiedad que se denomina exclusión mutua: no puede haber dos procesos modificando simultáneamente esa variable) y el objetivo es hacerlo utilizando sólo nuevas variables y "espera activa" (en otras palabras, está prohibido utilizar métodos synchronized, semáforos o cualquier otro mecanismo de concurrencia).

Material a entregar

El fichero fuente a entregar debe llamarse `CC_03_MutexEA.java`.

Solution 1: Peterson

```
static class Decrementador2 extends Thread {
  public void run() {
     for (int i = 0; i < N_OPS; i++) {
        quiere_dec = true;
        turno = INC;
        while (quiere_inc && turno == INC) {};
        cont--;
        quiere_dec = false;
     }
  }
}
```

Solution 1: Peterson

```
static class Decrementador2 extends Thread {
  public void run() {
     for (int i = 0; i < N_OPS; i++) {
        quiere_dec = true;
2 →     turno = INC;
        while (quiere_inc && turno == INC) {};
        cont--;
        quiere_dec = false;
     }
  }
}
```

mutex: $\begin{pmatrix} \texttt{quiere\_dec} = \texttt{false} \\ \vee \\ \texttt{turno} = \texttt{INC} \\ \vee \\ pc(dec) = 2 \end{pmatrix}$

Solution 2: Dekker

```
static class Incrementador2 extends Thread {
   public void run() {
      for (int i = 0; i < N_OPS; i++) {
         quiere_inc = true;
         while (quiere_dec) {
            if (turno!= INC) {
               quiere_inc = false;
               while (turno!=INC) {}
               quiere_inc = true;
            }
         }
         cont++;
         turno = DEC;
         quiere_inc = false;
      }
   }
}
```

Solution 3: Bakery

```
static class Decrementador2 extends Thread {
   public void run() {
      for (int i = 0; i < N_OPS; i++) {
         entering_dec = true;
         ticket_dec = ticket_inc + 1;
         entering_dec = false;
         while (entering_inc) {}
         while (ticket_inc !=0 && (ticket_inc <
             ticket_dec)) {}
         cont--;
         ticket_dec = 0;
      }
   }

    }
```