

# Sesión 6: Propiedades en Concurrency

Concurrency

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid



# Terminología<sup>1</sup> i

- Acción atómica: instrucción *mínima* que no pueden ser *interrumpida* por otro proceso
- Entrelazado: intercalado posible de acciones atómicas de diferentes procesos (*semántica*)
- Condición de carrera: resultados *indeseados* por *interacción* de dos o más procesos que *leen y modifican* datos *compartidos*
- Sección crítica: porción de código que puede dar lugar a una condición de carrera

---

<sup>1</sup>En inglés: *atomic action*, *interleaving*, *race conditions*, *critical section*



# Terminología<sup>3</sup> ii

- Exclusión mutua: propiedad deseable de nuestros programas que dice que **nunca** hay dos procesos ejecutando una **sección crítica** *al mismo tiempo*
- Espera activa: mecanismo de sincronización *autónomo*<sup>2</sup> basado en un bucle de comprobación continua

```
while (!C) { // no hacer nada }  
// ¡Aquí se cumple C!
```

---

<sup>2</sup>No requiere ayuda del sistema operativo o de bibliotecas

<sup>3</sup>mutual exclusion (mutex), busy-waiting



## 4 intentos = 4 problemas i

```
public static volatile int x = 0;  
public static volatile boolean enSC = false;
```

enSC  $\Leftrightarrow$  un proceso está en sección crítica

```
while (enSC) {}  
enSC = true;  
x = x + 1;  
enSC = false;
```

```
while (enSC) {}  
enSC = true;  
x = x - 1;  
enSC = false;
```



## 4 intentos = 4 problemas ii

```
public static volatile int x = 0;  
public static volatile boolean turnoInc;
```

turnoInc  $\Leftrightarrow$  el turno es del incrementador

```
while (!turnoInc) {}  
x = x + 1;  
turnoInc = false;
```

```
while (turnoInc) {}  
x = x - 1;  
turnoInc = true;
```



## 4 intentos = 4 problemas iii

```
public static volatile int x = 0;  
public static volatile boolean scInc = false;  
public static volatile boolean scDec = false;
```

scInc  $\Leftrightarrow$  incrementador quiere acceder a SC

scDec  $\Leftrightarrow$  decrementador quiere acceder a SC

```
scInc = true;  
while (scDec) {}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {}  
x = x - 1;  
scDec = false;
```



## 4 intentos = 4 problemas iv

scInc  $\Leftrightarrow$  incrementador quiere acceder a SC

scDec  $\Leftrightarrow$  decrementador quiere acceder a SC

```
scInc = true;
while (scDec) {
    scInc = false;
    scInc = true;
}
x = x + 1;
scInc = false;
```

```
scDec = true;
while (scInc) {
    scDec = false;
    scDec = true;
}
x = x - 1;
scDec = false;
```

# Propiedades deseables

i. Garantizar exclusión mutua

Exclusión mutua

ii. Ausencia de alternancia estricta

Ausencia de esperas innecesarias

iii. Garantizar que los procesos no queden *atascados*

Ausencia de *interbloqueo*<sup>4</sup>

iv. Garantizar que los procesos progresan

Ausencia de *inanición*<sup>5</sup>

---

<sup>4</sup>*Deadlock o livelock*

<sup>5</sup>*Starvation*



# Propiedades: seguridad y vivacidad<sup>6</sup>

## Seguridad

Siempre se cumple  $P$

Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

---

<sup>6</sup>*Safety y Liveness*

# Exclusión mutua

```
while (enSC) {}  
enSC = true;  
x = x + 1;  
enSC = false;
```

```
while (enSC) {}  
enSC = true;  
x = x - 1;  
enSC = false;
```

dos procesos ejecutan la  
sección crítica al mismo tiempo

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Exclusión mutua

```
while (enSC) {}  
enSC = true;  
x = x + 1;  
enSC = false;
```

```
while (enSC) {}  
enSC = true;  
x = x - 1;  
enSC = false;
```

Nunca se cumple que dos procesos ejecutan la sección crítica al mismo tiempo

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Esperas innecesarias

```
while (!turnoInc) {}  
x = x + 1;  
turnoInc = false;
```

```
while (turnoInc) {}  
x = x - 1;  
turnoInc = true;
```

hay un proceso que no puede avanzar  
inmediatamente por una condición ajena al problema  
que tratamos de resolver

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Esperas innecesarias

```
while (!turnoInc) {}  
x = x + 1;  
turnoInc = false;
```

```
while (turnoInc) {}  
x = x - 1;  
turnoInc = true;
```

Nunca hay un proceso que no puede avanzar inmediatamente por una condición ajena al problema que tratamos de resolver

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Interbloqueo (*deadlock*, )

```
scInc = true;  
while (scDec) {}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {}  
x = x - 1;  
scDec = false;
```

proceso que  
en el futuro no pueda avanzar

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Interbloqueo (*deadlock*, )

```
scInc = true;  
while (scDec) {}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {}  
x = x - 1;  
scDec = false;
```

Nunca hay un proceso que  
en el futuro no pueda avanzar

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Interbloqueo (*deadlock, livelock*)

```
scInc = true;  
while (scDec) {}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {}  
x = x - 1;  
scDec = false;
```

Nunca hay un proceso que  
en el futuro no pueda avanzar

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$



# Interbloqueo<sup>7</sup>

*When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.*

**Statute passed by the Kansas State Legislature  
early in the 20th century**

---

<sup>7</sup>Extraído de unas transparencias del profesor César Sánchez.

# Inanición

```
scInc = true;  
while (scDec) {  
    scInc = false; scInc = true;  
}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {  
    scDec = false; scDec = true;  
}  
x = x - 1;  
scDec = false;
```

*hay un entrelazado potencialmente infinito en el que uno de los procesos no avanza*

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

Alguna vez se cumple  $P$

# Inanición

```
scInc = true;  
while (scDec) {  
    scInc = false; scInc = true;  
}  
x = x + 1;  
scInc = false;
```

```
scDec = true;  
while (scInc) {  
    scDec = false; scDec = true;  
}  
x = x - 1;  
scDec = false;
```

Alguna vez uno de los procesos  
es seguro que avanzará

## Seguridad

Siempre se cumple  $P$   
Nunca se cumple  $N$

## Vivacidad

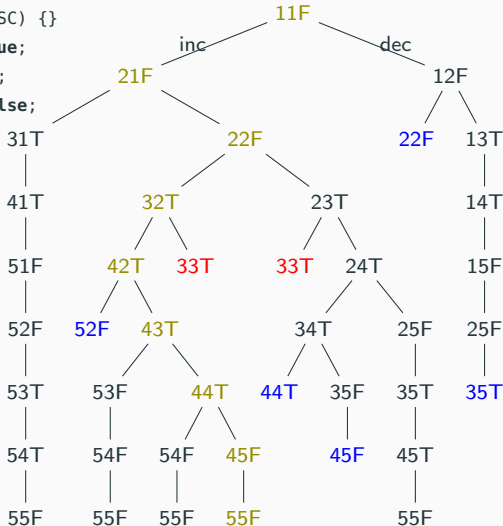
Alguna vez se cumple  $P$

# Todos los entrelazados posibles

```

1 while (enSC) {}
2 enSC = true;
3 x = x + 1;
4 enSC = false;
5

```



```

1 while (enSC) {}
2 enSC = true;
3 x = x - 1;
4 enSC = false;
5

```

Estado:

contadores de programa ( $CP_{inc}$  y  $CP_{dec}$ ) y variables ( $enSC$ )

- estado repetido
- estado indeseado
- ejemplo de entrelazado



# Ejercicio obligatorio semanal

Hoja de ejercicios en:

<http://babel.ls.fi.upm.es/teaching/concurrencia>

Ejercicio 3:

**Garantizar exclusión mutua con espera activa**

Fichero a entregar:

`CC_03_MutexEA.java`

Sistema de entrega:

<http://vps142.cesvima.upm.es>