

# Sesión 5: Exclusión mutua con espera activa

Concurrencia

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

# Concurrencia

 Simultaneidad

+

Sincronización +  Comunicación<sup>1</sup>

---

<sup>1</sup>Sólo con memoria compartida.

# Concurrencia

 Simultaneidad

+

Sincronización +  Comunicación<sup>1</sup>

---

<sup>1</sup>Sólo con memoria compartida.



# Terminología<sup>2</sup>

- 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>2</sup>En inglés: *atomic action*, *interleaving*, *race conditions*, *critical section*

## Terminología: Exclusión mutua<sup>3</sup>

- Exclusión mutua (*mutex*): propiedad deseable de nuestros programas que dice que nunca hay dos procesos ejecutando una sección crítica *al mismo tiempo*

---

<sup>3</sup>Mutual exlusion



# Terminología: **Espera activa**<sup>5</sup>

- Espera activa: mecanismo de sincronización *autónomo*<sup>4</sup> basado en un bucle de comprobación continua

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

---

<sup>4</sup>No requiere ayuda del sistema operativo o de bibliotecas

<sup>5</sup>Busy waiting

## *Invariante* con nuevas variables

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

```
x = x + 1;
```



```
x = x - 1;
```

## Invariante con nuevas variables

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

*// enSC  $\Leftrightarrow$  un proceso está en sección crítica*

```
public static volatile boolean enSC = false;
```

```
x = x + 1;
```

```
x = x - 1;
```



## Invariante con nuevas variables

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

*// enSC  $\Leftrightarrow$  un proceso está en sección crítica*

```
public static volatile boolean enSC = false;
```

```
enSC = true;
```

```
x = x + 1;
```

```
enSC = false;
```





```
enSC = true;
```

```
x = x - 1;
```

```
enSC = false;
```

# Primer intento: no hay mutex

```
1 public class EsperaActiva1 {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```

 variables compartidas,  secciones críticas,  protocolo de entrada a la sección crítica,  protocolo de salida

```
17 static class Dec extends Thread {
18     public void run() {
19         for (int i = 0; i < N_OPS; i++) {
20             while (enSC) {}
21             enSC = true;
22             x = x - 1;
23             enSC = false;
24         }
25     }
26 }
27
28 public static void main(String[] args)
29     throws InterruptedException {
30     Thread i = new Inc();
31     Thread d = new Dec();
32     i.start(); d.start();
33     i.join(); d.join();
34     System.out.println(x);
35 }
36 }
```

# Segundo intento: esperas innecesarias

```
1 public class EsperaActiva2 {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean turnoInc;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (!turnoInc) {}
10                x = x + 1;
11                turnoInc = false;
12            }
13        }
14    }
15}
```

```
17     public void run() {
18         for (int i = 0; i < N_OPS; i++) {
19             while (turnoInc) {}
20             x = x - 1;
21             turnoInc = true;
22         }
23     }
24 }
25
26 public static void main(String[] args)
27     throws InterruptedException {
28     Thread i = new Inc();
29     Thread d = new Dec();
30     i.start(); d.start();
31     i.join(); d.join();
32     System.out.println(x);
33 }
34 }
```

## Tercer intento: *¡más madera!*

```
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

x = x + 1;

x = x - 1;

## Tercer intento: *¡más madera!*

```
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;
```

```
x = x + 1;
```

```
scInc = false;
```

```
scDec = true;
```

```
x = x - 1;
```

```
scDec = false;
```

## Tercer intento: *¡más madera!*

```
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 (...) {}  
x = x + 1;  
scInc = false;
```

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

# Tercer intento: *¡quiero acceder pero...!*

```
1 public class EsperaActiva3 {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean scInc = false;
5     static volatile boolean scDec = false;
6
7     static class Inc extends Thread {
8         public void run() {
9             for (int i = 0; i < N_OPS; i++) {
10                 scInc = true;
11                 while (!scDec) {}
12                 x = x + 1;
13                 scInc = false;
14             }
15         }
16     }
```

```
18     static class Dec extends Thread {
19         public void run() {
20             for (int i = 0; i < N_OPS; i++) {
21                 scDec = true;
22                 while (scInc) {}
23                 x = x - 1;
24                 scDec = false;
25             }
26         }
27     }
28
29     public static void main(String[] args)
30         throws InterruptedException {
31         Thread i = new Inc();
32         Thread d = new Dec();
33         i.start(); d.start();
34         i.join(); d.join();
35         System.out.println(x);
36     }
37 }
```

# Cuarto intento: *¡te dejo un momento!*

```
7  static class Inc extends Thread {
8      public void run() {
9          for (int i = 0; i < N_OPS; i++) {
10             scInc = true;
11             while (!scDec) {
12                 scInc = false;
13                 scInc = true;
14             }
15             x = x + 1;
16             scInc = false;
17         }
18     }
19 }
```

```
21 static class Dec extends Thread {
22     public void run() {
23         for (int i = 0; i < N_OPS; i++) {
24             scDec = true;
25             while (scInc) {
26                 scDec = false;
27                 scDec = true;
28             }
29             x = x - 1;
30             scDec = false;
31         }
32     }
33 }
```



# Cuarto intento: *¡te dejo un momento!*

```
7  static class Inc extends Thread {
8      public void run() {
9          for (int i = 0; i < N_OPS; i++) {
10             scInc = true;
11             while (!scDec) {
12                 scInc = false;
13                 scInc = true;
14             }
15             x = x + 1;
16             scInc = false;
17         }
18     }
19 }
```

```
21 static class Dec extends Thread {
22     public void run() {
23         for (int i = 0; i < N_OPS; i++) {
24             scDec = true;
25             while (scInc) {
26                 scDec = false;
27                 scDec = true;
28             }
29             x = x - 1;
30             scDec = false;
31         }
32     }
33 }
```

Se puede ver que los protocolos de entrada  
tienen un punto de **cooperativos**