Sesión 6: Propiedades en Concurrencia

Concurrencia

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

Terminología[1] 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

[1] En inglés: *atomic action*, *interleaving*, *race conditions*, *critical section*

Terminología[3] 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*[2] basado en un bucle de comprobación continua

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

[2] No requiere ayuda del sistema operativo o de bibliotecas

[3] *mutual exclussion* (*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*[4]

iv. Garantizar que los procesos progresan

Ausencia de *inanición*[5]

[4] *Deadlock* o *livelock*

[5] *Starvation*

Propiedades: seguridad y vivacidad[6]

Seguridad

Siempre se cumple P

Nunca se cumple N

Vivacidad

Alguna vez se cumple P

[6] *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) {}          ||  while (turnoInc) {}
x = x + 1;                    ||  x = x - 1;
turnoInc = false;             ||  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) {}          ||  while (turnoInc) {}
x = x + 1;                    ||  x = x - 1;
turnoInc = false;             ||  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;          ||   scDec = true;
while (scDec) {}       ||   while (scInc) {}
x = x + 1;             ||   x = x - 1;
scInc = false;         ||   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;          ||   scDec = true;
while (scDec) {}       ||   while (scInc) {}
x = x + 1;             ||   x = x - 1;
scInc = false;         ||   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[7]

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

[7] Extraido 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

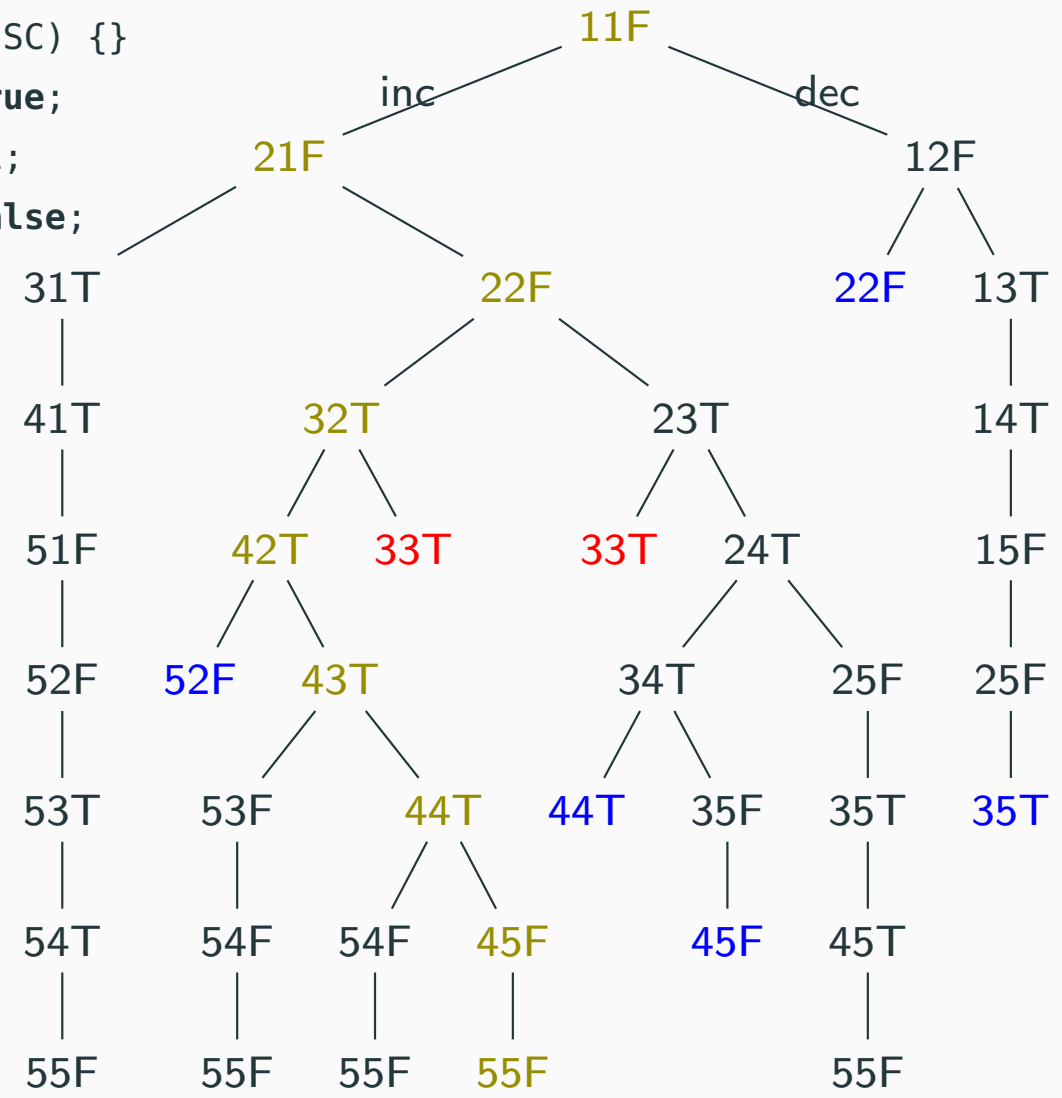
Siempre se cumple P
Nunca se cumple N

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 ejerccios 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`