Lección 5: Diseño de programas concurrentes

- Introducción
- Pautas para el diseño de programas concurrentes
- La instrucción **await**
- Ejemplos de diseño de programas con la instrucción **await**

Introducción

- El diseño de programas concurrentes (correctos) es una tarea compleja
  - A la dificultad del diseño de programas secuenciales hay que añadir las posibles interferencias entre los procesos
  - Los posibles entrelazados pueden dar resultados distintos en distintas ejecuciones
  - Lo que hace que la depuración sea complicada
- Es indispensable poder "asegurar" la corrección por el diseño
- Podemos plantear una solución de grano grueso y refinamientos posteriores
  - Hasta llegar a una solución fácilmente implementable

Pautas para el diseño de programas concurrentes

- Algunas pautas
  - Identificar muy claramente los procesos y los datos compartidos
  - Diseñar los procesos, determinando claramente la separación entre los datos compartidos y los que vamos a añadir para gestionar los aspectos de sincronización
  - Plantear un diseño en base a instrucciones atómicas de tipo **await**
  - Llevar a cabo la traducción del diseño en base a las instrucciones permitidas en nuestro lenguaje (semáforos, monitores, etc.)

La instrucción **await**

- Se trata de una instrucción de alto nivel ("grano grueso")
  - Potente, pero costosa de implementar
- Sintaxis

```
< await B
    S
>
```

  - **B**: es una expresión booleana
  - **S**: es un conjunto de instrucciones
- Nos evitará el uso de esperas activas

La instrucción **await**

< await **B**
S
>

- Semántica: cuando se pasa el control a un proceso para ejecutar una instrucción **await**
  - Se evalúa la guarda
  - Si es cierta, se ejecuta S, y se pasa a la instrucción que sigue al **await**
  - Si es falsa, no se hace nada, y no se pasa a la siguiente instrucción
    - Por lo que el **await** seguirá siendo la instrucción elegible para este proceso
  - Y todo esto es **atómico**

La instrucción **await**

```
< await B
    S
>
```

- Las formas habituales de sincronización son casos particulares de la instrucción:
  - Exclusión mutua

```
< await true
    S
>
```

```
< S >
```

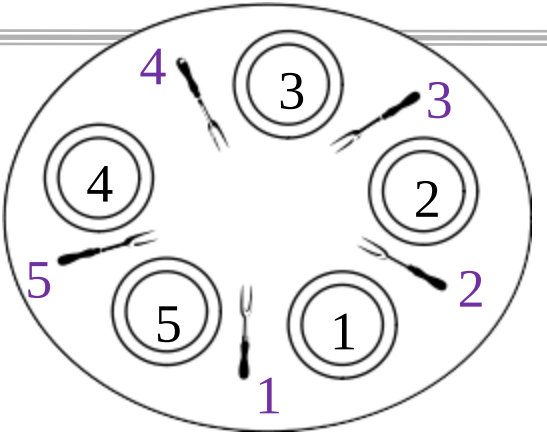
  - Sincronización por condición

```
< await B
    // nada
>
```

```
< await B >
```

Ejemplos de diseño: el problema de los filósofos

- Dijkstra 65, Hoare 85
  - Prototipo de sistema en que los procesos comparten recursos conservativos
- Hay 5 filósofos sentados alrededor de una mesa para comer espaguetis
- Hay 5 tenedores, e infinita pasta
- Hacen falta dos tenedores para comer
  - Cada filósofo puede usar los que tiene más cerca
  - Cada filósofo coge primero el de su izda., y luego el de su dcha.
- Se pide un programa que simule el comportamiento del sistema




```
Process filosofo(i:1..5):
   while true
      //piensa
      //coge tenedores
      //come
      //deja tenedores
   end
end
```

Ejemplos de diseño: el problema de los filósofos

- ¿Cómo modelar el estado de los tenedores?

```
constant integer N := 5
boolean array[1..N] libre := (1..N,true)

Process filosofo(i:1..N):
   while true
      //piensa
      //espera a que tenedor izda. libre y lo coge
      //espera a que tenedor dcha. libre y lo coge
      //come
      //deja tenedor a la izda.
      //deja tenedor a la dcha.
   end
end
```

Ejemplos de diseño: el problema de los filósofos

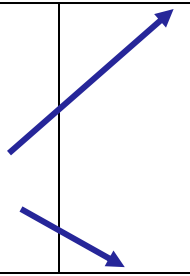
- Versión 1: primero un tenedor, y luego el otro

```
const integer N := 5
boolean array[1..N] libre := (1..N,true)

Process filosofo(i:1..N):
   while true
      //piensa
      //espera a que tenedor izda. libre y lo coge
      //espera a que tenedor dcha. Libre y lo coge
      //come
      //deja tenedor a la izda.
      //deja tenedor a la dcha.
   end
end
```

```
< await libre[i]
      libre[i] := false
>
```

```
< await libre[1 + i mod N]
      libre[1 + i mod N] := false
>
```



Ejemplos de diseño: el problema de los filósofos

- Si estudiamos el comportamiento de la solución, vemos que se puede alcanzar un bloqueo total, en el que todos los filósofos mueren de hambre
  - Encontrar la situación de bloqueo
- Plantear una solución alternativa, **V2**, en que los tenedores se cojan siempre por orden.
  - No se bloqueará porque no se puede cerrar ciclo de espera por recursos
- Plantear tercera solución alternativa, **V3**, en que cada filósofo coge ambos tenedores a la vez (o no los coge)
  - No se bloqueará porque los estados no son ***hold-and-wait***

Ejemplos de diseño: máximo de un vector

- Considérese el código que se muestra a continuación, en el que se lanzan M procesos concurrentes con el objetivo de que entre todos encuentren el valor máximo de un vector
  - cada proceso se encarga de un trozo del vector.
- Una vez todos han acabado, el proceso informador muestra el valor máximo del vector
- Se pide completar el diseño de acuerdo a la especificación.

```
constant integer N := ... //lo que sea, >= 1
                 M := ... //lo que sea, >= 1
                 NM := N*M
integer array[1..NM] val := ... //lo que sea, ya inicializado
integer max
// código a completar

Process P(i: 1..M)::
    // código a completar
    // al terminar todos los procesos, "max" contiene el
    // valor máximo del vector "val"
end

Process informador::
    // código a completar
    write('Max= ', max)
end
```

```
//Pre:  1 <= i1 <= i2 <= NM
//Post: maxTrozo() = el valor máximo de v[i1],...,v[i2]
operation maxTrozo(integer array[1..NM] v, integer i1,i2): integer
```

Ejemplos de diseño: máximo de un vector

Considérese un programa concurrente compuesto por 10 procesos: 5 procesos **lectores** y 5 procesos **calculadores**, que comparten una matriz 5x50 de números enteros. El ejercicio pide completar el esquema de programa que se muestra a continuación, de manera que:

- Cada proceso lector lee de la entrada estándar una secuencia de enteros compuesta por 50 números y rellena una fila completa de la matriz
- Cada proceso calculador busca el máximo parcial de una fila de la matriz
- Una vez que todos los procesos calculadores han hecho su trabajo, el proceso con identificador 1 muestra por la salida estándar el máximo global de la matriz

Ejemplos de diseño: máximo de un vector

Considérese un programa concurrente compuesto por 10 procesos: 5 procesos **lectores** y 5 procesos **calculadores**, que comparten una matriz 5x50 de números enteros. El ejercicio pide completar el esquema de programa que se muestra a continuación, de manera que:

- Cada proceso lector lee de la entrada estándar una secuencia de enteros compuesta por 50 números y rellena una fila completa de la matriz
- Cada proceso calculador busca el máximo parcial de una fila de la matriz
- Una vez que todos los procesos calculadores han hecho su trabajo, el proceso con identificador 1 muestra por la salida estándar el máximo global de la matriz

```
integer array[1..5,1..50] D
. . .
Process Lector(i:1..5)::
  . . .
end
Process Calculador(i:1..5)::
  . . .
end
```