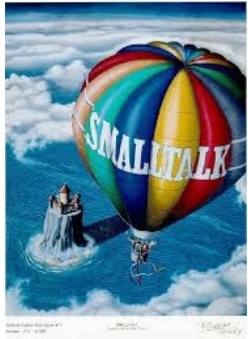




Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



Programación Concurrente



*Fundamentos de la
Concurrencia*



Temario

- Repasamos ...
- Programas, procesos y concurrencia
- Especificación de comportamiento concurrente
- Características de los sistemas concurrentes

¿Qué es concurrencia?

Según Real Academia Española: <Acaecimiento o concurso de varios sucesos en un mismo tiempo>

- La concurrencia es como un **conjunto de actividades** que se desarrollan de **forma simultánea**.
- En informática, cada una de esas actividades se suele llamar **proceso**.

Programa vs. Proceso

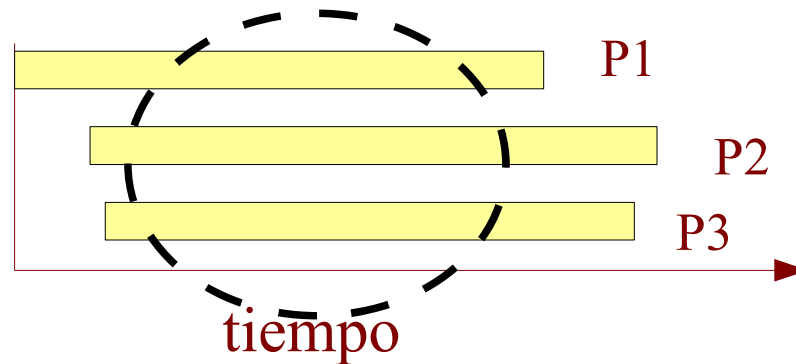
- **Programa:** Conjunto de sentencias/instrucciones que se ejecutan secuencialmente. Concepto estático.
- **Proceso:** Básicamente, se puede definir como un programa en ejecución. Líneas de código en ejecución de manera **dinámica**.

Se asemeja al concepto de **Clase** de la POO

Se asemeja al concepto de **Objeto** de la POO

Concurrencia

- Es la **existencia simultánea** de varios procesos en ejecución.
- Dos **procesos** son **concurrentes** cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última



Cómo expresar la concurrencia

- Sentencia concurrente:

cobegin

P; Q; R

coend;

- Objetos que representan procesos:

- **task A is** begin P; end;

- **task B is** begin Q; end;

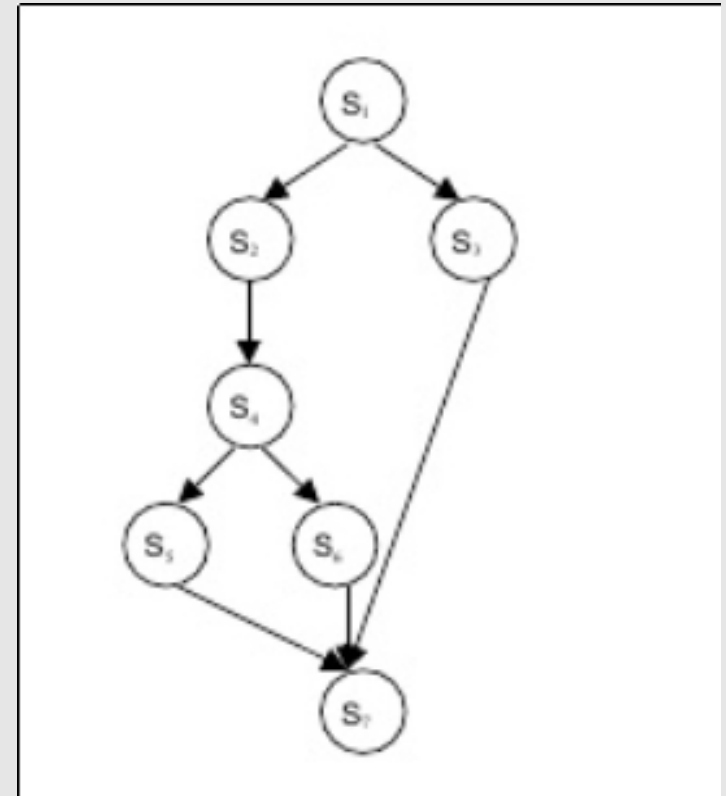
- **task C is** begin R; end;

- Sentencia concurrente múltiple:

forall i:=1 to 1000 do P(i);

- Notación gráfica:

- Grafos de precedencia



Cómo expresar la concurrencia

- Especificación concurrente:

cobegin/coend

S1

cobegin

S3

begin

S2

S4

cobegin

S5

S6

coend

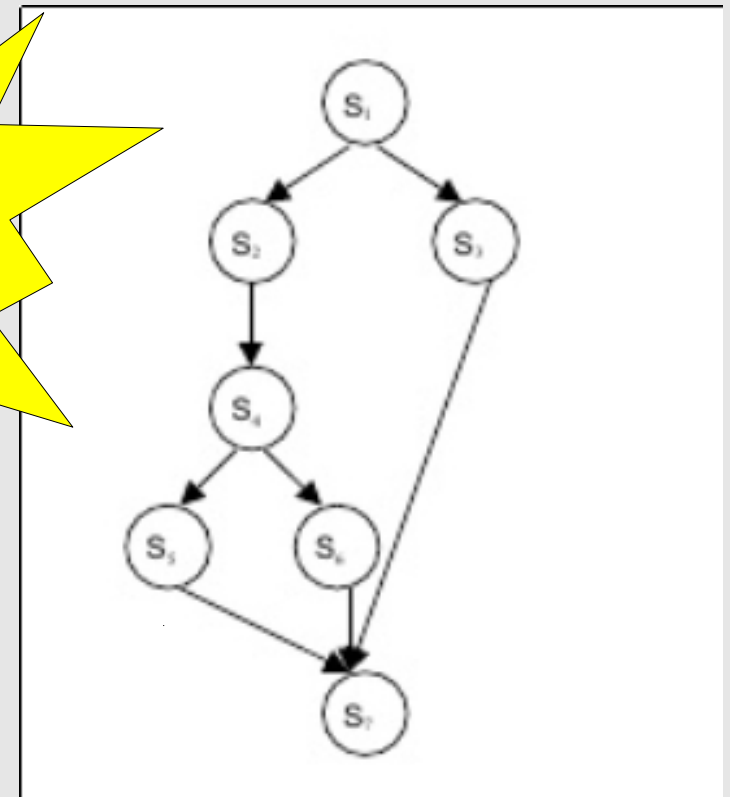
end

coend;

S7

No se puede
representar
TODOS
los grafos de
precedencia

- Grafos de precedencia



Cómo expresar la concurrencia

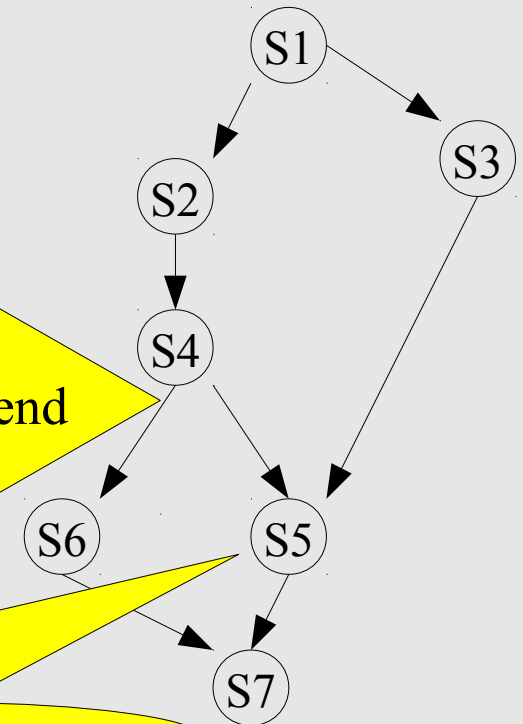
- Sentencia concurrente:

fork/join

Programas con estructura de control inadecuada - confusa
(no la vamos a considerar)

- Notación gráfica:

- Grafos de precedencia



No se puede con cobegin/coend

S5 no solo es precedido por S4, también por S3,
por lo que no puede ser encerrado en cobegin/coend con S6

Problemas de la PC

¿qué significa?

- ¿Pueden P1 y P2 ejecutarse de forma concurrente y determinista?
- ¿Cómo puedo saberlo?

```
proceso P1;  
    var i: integer;  
begin  
    for i:=1 to 5 do ...  
end;  
  
proceso P2;  
    var j: integer;  
begin  
    for j:=6 to 15 do ...  
end;
```

```
begin  
    x= 0;  
cobegin  
    P1;  
    P2;  
coend  
Escribir x;  
end;
```

depende...
¿quien es x?
¿P1 actúa sobre x?
¿P2 actúa sobre x?



Problemas de la PC

- ¿Pueden P1 y P2 ejecutarse de forma concurrente y determinista?
- Completamos P1 y P2

```
proceso P1;  
    var i: integer;  
begin  
    for i:=1 to 5 do x:= x+1;  
end;  
  
proceso P2;  
    var j: integer;  
begin  
    for j:=6 to 15 do x:= x+1;  
end;
```

```
begin  
    x= 0;  
    cobegin  
        P1;  
        P2;  
    coend;  
    Escribir x  
end;
```

x es una *variable compartida*
P1 actúa sobre x,
con acciones de
lectura y escritura
P2 también

Problemas de la PC

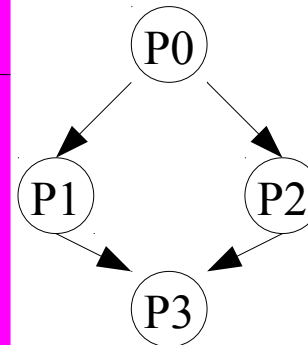
- Es más difícil analizar y verificar un algoritmo concurrente por el no determinismo.
- Ojo, que existan varias posibilidades de salida NO significa necesariamente que un programa concurrente sea incorrecto

```
Proceso P0  
  x := 100;  
End;
```

```
proceso P1;  
  x := x+10;  
end;
```

```
proceso P2;  
  Si x > 100 escribir(x)  
  Sino escribir (x-50)  
end;
```

```
Proceso P3  
  escribir ('fin')
```



```
begin  
  P0;  
  cobegin  
    P1;  
    P2:  
  coend  
  P3;  
end;
```

¿Cual es la salida?

Problemas de la PC

Proceso P0

```
x := 100;  
end;
```

Proceso P1

```
x := x + 10;  
end;
```

proceso P2;

```
Si x > 100 escribir(x)  
Sino escribir (x-50)  
end;
```

Proceso P3

```
escribir ('fin')  
end
```

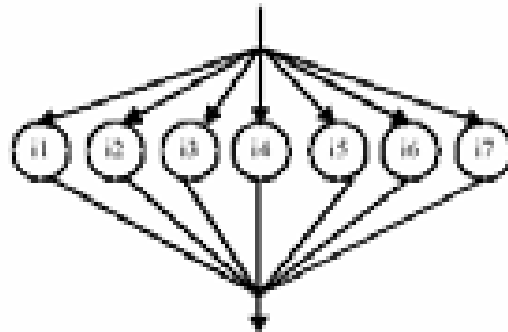
- Se ejecuta P0, P1, P2, P3
 $x=100$, $x = x+10$ (110), $\text{escribir}(x) \rightarrow 110$,
 escribir "fin"
- Se ejecuta P0, P2, P1, P3
 $x=100$, $x > 100$ (no), $\text{escribir}(x-50) \rightarrow 50$
 escribir "fin"
- Se ejecuta P0, P2 (solo $x > 100$), P1,
continúa P2, P3
 $x=100$, $x > 100$ (no), $x = x+10$ (110),
 $\text{escribir}(x-50) \rightarrow 60$
 escribir "fin"

INDETERMINISMO

Un problema propio de la PC

- **Indeterminismo:** Un programa concurrente define un orden **parcial** de ejecución. Ante un conjunto de datos de entrada no se puede saber cual va a ser el flujo de ejecución

Sentencia concurrente:
cobegin
....
coend;

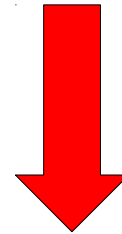


```
begin  
  cobegin  
    i1;i2;i3;i4;i5;i6;i7  
  coend  
end;
```

Los programas concurrentes pueden producir diferentes resultados en ejecuciones repetidas sobre el mismo conjunto de datos de entrada

Programación Concurrente

- ¿Cómo implementar concurrencia en el paradigma orientado a objetos?



Programación Orientada a Objetos
CONCURRENTE

Programación Concurrente

Modelos de Objetos

- *Características*: clase, estado, referencia, método, identidad, encapsulación
- *Operaciones básicas*:
 - Aceptar un mensaje
 - Actualizar el estado
 - Enviar un mensaje
 - Crear nuevos objetos
- *Categorías de objetos* (reglas para las operaciones)
 - Activo / pasivo
 - El modelo concurrente incluye características de las 2

Programación Concurrente

¿Qué es un *proceso*? y ¿Qué es un *hilo*?

- ★ Programa: entidad pasiva - código
- ★ Proceso: entidad activa – programa en ejecución. Ocupa recursos, tiene asignado un espacio de memoria, ...
- ★ Concurrencia con múltiples hilos o multihilos (multithreading). Un proceso puede tener varios hilos.
- ★ Concurrencia de entornos livianos de ejecución: eventos, tareas que no pueden bloquearse unas a otras

Programación concurrente

¿Cómo implementar concurrencia en el paradigma orientado a objetos?

- se asocian las tareas a los métodos y los hilos que utilizan esas tareas a objetos
- un sistema es un conjunto de objetos autónomos que colaboran, activa y concurrentemente.
- modelos de objetos
 - Modelo activo
 - Modelo pasivo

Concurrencia en Java

Repasamos ...

Programa Concurrente creado usando la interfaz *Runnable*

```
public class EjemploRunnable implements Runnable {  
  
    public static void main(String args[]) {  
        EjemploRunnable re = new EjemploRunnable();  
        Thread hilo = new Thread(re);  
        hilo.start();  
    }  
}
```

- ¿qué sucede cuando se tiene el siguiente código?

```
Public class PruebaHilo extends Thread {
```

```
...
```

```
Thread hilo= new Thread();
```

```
hilo.start();
```

```
}
```

Se crea un nuevo
hilo de ejecución

Se lanza el nuevo
hilo de ejecución, que utiliza
su propia pila de llamadas

Pero ... ¿qué hace?

...nada... el hilo creado NO HACE NADA

El hilo muere y su pila de llamadas desaparece

¿que nos está faltando?



...ah...
el trabajo del hilo

... o sea
que código queremos que se ejecute
por separado en ese hilo.
Falta implementar el método **run()**!!



Concurrencia en Java

Proceso P0

```
x := 100;  
end;
```

Proceso P1

```
x := x + 10;  
end;
```

proceso P2;

```
Si x > 100 escribir(x)  
Sino escribir (x-50)  
end;
```

Proceso P3

```
escribir ('fin')  
end
```

Para implementarlo en Java ...
tendría que considerar:

- un hilo para P1 y
- otro hilo para P2
- para P0 y P3 no necesito crear hilos,
se ocupa de ellos el
hilo principal de ejecución



El *hilo principal de ejecución* corresponde a la ejecución del método `main` de cualquier programa Java.

Concurrencia en Java

```
package hilos;

public class Datos {
    private int dato = 0;
    public Datos(int nro) {
        dato = nro;
    }
    public int getDato() {
        return dato;
    }
    public void setDato(int valor) {
        dato = valor;
    }
    public boolean verificar(int valor) {
        return dato > valor;
    }
}
```

Concurrencia en Java

```
package hilos;
```

```
public class TestMiHilo{  
    public static void main(String[] args) {
```

```
        Datos x= new Datos(100);  
        ProcesoUno pUno = new ProcesoUno(x);  
        ProcesoDos pDos = new ProcesoDos(x);  
        Thread hilo1 = new Thread(pDos);  
        Thread hilo2 = new Thread(pUno);  
        hilo1.start();  
        hilo2.start();  
        System.out.println("fin");  
        System.out.println("vuelta en el main");
```

```
    }
```

```
}
```

Se creará un hilo Principal de ejecución que se ejecutará concurrentemente con los otros hilos creados

Thread es la unidad de concurrencia en Java

Concurrencia en Java

```
package hilos;
```

```
public class TestMiHilo{  
    public static void main(String[]
```

```
        Datos x= new Datos(100);  
        ProcesoUno pUno = new ProcesoUno(  
        ProcesoDos pDos = new ProcesoDos(x);  
        Thread hilo1 = new Thread(pDos);  
        Thread hilo2 = new Thread(pUno);  
        hilo1.start();  
        hilo2.start();  
        System.out.println("fin");  
        System.out.println("vuelta");  
    }  
}
```

Se crean 2 objetos Thread,
pero ... no hay hilos de *ejecución*
hasta que se envia el mensaje
start a los objetos

Los hilos reciben el
mensaje de
“*estar listos*”,
pasan a estado
“*runnable*”

esperando por su turno para ejecutar el método *run()*

Concurrencia en Java

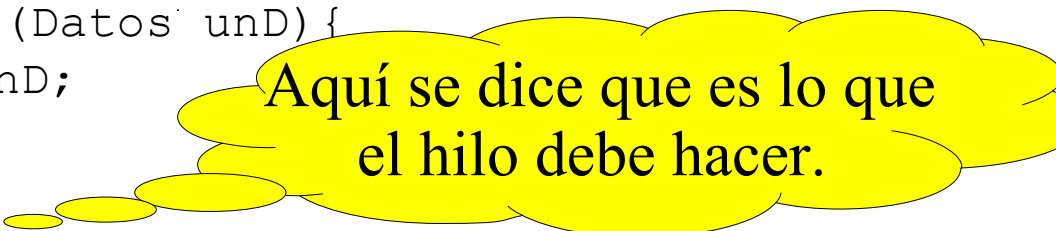
```
package hilos;

public class ProcesoUno implements Runnable{

    private Datos unDato;

    public ProcesoUno(Datos unD){
        unDato = unD;
    }

    public void run() {
        System.out.println("estoy en ProcesoUno");
        if (unDato.getDato() > 100)
            System.out.println(unDato.getDato());
        else
            System.out.println(unDato.getDato()-50);
    }
}
```



Aquí se dice que es lo que el hilo debe hacer.

Concurrencia en Java

```
package hilos;

public class ProcesoDos implements Runnable{
    private Datos unDato;

    public ProcesoDos(Datos unD) {
        unDato = unD;
    }

    public void run() {
        System.out.println("estoy en ProcesoDos");
        unDato.setDato(unDato.getDato()+10);
    }
}
```

Multihilos en Java: clase Thread

- Un *hilo* se crea en Java instanciando un objeto de la clase **Thread**.
- El código que ejecuta un *hilo* está definido por el método **run()** que tiene todo objeto que sea instancia de la clase Thread.
- La ejecución del *hilo se inicia* cuando, una vez recibido el mensaje **start()**, además el planificador de procesos le asigna tiempo para ejecutarse.
- De forma natural, un *hilo termina* cuando en **run()** se alcanza una sentencia **return** o el final del método.
(Existen otras formas de terminación forzada)

Concurrencia en Java

La unidad de concurrencia es el Thread, que comparte el mismo espacio de variables con los restantes hilos

Pero... cada hilo de ejecución tiene su propia pila de llamadas

Utiliza la clase Thread.

Concurrencia en Smalltalk

Proceso P0

```
x:= 100;  
end;
```

Proceso P1

```
x:= x + 10;  
end;
```

proceso P2;

```
Si x>100 escribir(x)  
Sino escribir (x-50)  
end;
```

Proceso P3

```
escribir ('fin')  
end
```

¿Cómo se hará en Smalltalk?

Mmm...!!,

tengo que utilizar el mensaje “fork”
sobre un bloque que debe contener el
código a ejecutar
(como el método run() en Java)



Concurrencia en Smalltalk

```
|x p1 p2|  
Transcript clear.  
x := 100.  
p1:= [x:= x+10].  
p2:= [(x>100)  
      ifTrue: [Transcript show: x asString]  
      ifFalse:[Transcript show: (x-50) asString]].  
p1 fork.  
p2 fork.  
Transcript show: 'fin'.
```

Recordemos...conurrencia

- **Proceso:** secuencia de acciones que se realizan independientemente de las acciones realizadas por otros procesos.
- Los **procesos** que se ejecutan en paralelo, **son objetos** que poseen una prioridad, la cual puede asignarse en un principio e ir modificándose a lo largo de la ejecución.
- La **prioridad** de un proceso, describe la importancia que tiene ese proceso por sobre los demás.