Arq. Y Sistemas Operativos

Procesos Concurrentes

Procesos Concurrentes

La ejecución de procesos

***Puede* ser**

Secuencial

Concurrente

Se caracteriza por:

Secuencia de instrucciones que se ejecutan sobre un procesador

Independiente de la velocidad.

Idénticos resultados para el mismo conjunto de datos

Dos o más procesos o hilos secuenciales que pueden ejecutarse concurrentemente como tareas paralelas

Actividades superpuestas en el tiempo.

Requiere de mecanismos de sincronización y comunicación

Condiciones de Concurrencia

Notación

Sentencia: Si

Sentencia o conjunto secuencial de instrucciones agrupadas

Conjunto de Lectura: **R (Si) = (a1, a2,... , am)**

El conjunto de lectura de la sentencia Si es aquel formado por todas las variables que son referenciadas por la sentencia Si durante su ejecución sin sufrir cambios

Condiciones de Concurrencia

Notación

Conjunto de escritura: W (Si) = (b1, b2,... , bn)

El conjunto de escritura de la sentencia S_i es aquel formado por todas las variables cuyos valores son modificados durante la ejecución de S_i.

Condiciones de Concurrencia (Bernstein)

Dos sentencias cualesquiera Si y Sk pueden ejecutarse concurrentemente produciendo el mismo resultado que si se ejecutaran secuencialmente sí y sólo sí se cumplen las siguientes condiciones

1. $R(S_i) \cap W(S_k) = (\emptyset)$
2. $W(S_i) \cap R(S_k) = (\emptyset)$
3. $W(S_i) \cap W(S_k) = (\emptyset)$

Condiciones de Concurrencia

S1: x = a+b **S2:** y = c+b **S3:** z = x + b

$R(S_1) = (a,b)$ $R(S2) = (c,b)$ $R(S3) = (x,b)$

$W(S_1) = (x)$ $W(S2) = (y)$ $W(S3) = (z)$

1. $R(S_1) \cap W(S_2) = (a,b) \cap (y) = (\emptyset)$
2. $W(S_1) \cap R(S_2) = (x) \cap (c,b) = (\emptyset)$
3. $W(S_1) \cap W(S_2) = (x) \cap (y) = (\emptyset)$

S1 y S2 pueden ejecutarse en paralelo o concurrentemente

Condiciones de Concurrencia

S1: x = a+b **S2:** y = c+b **S3:** z = x + b

$R(S_1) = (a,b)$ $R(S2) = (c,b)$ $R(S3) = (x,b)$

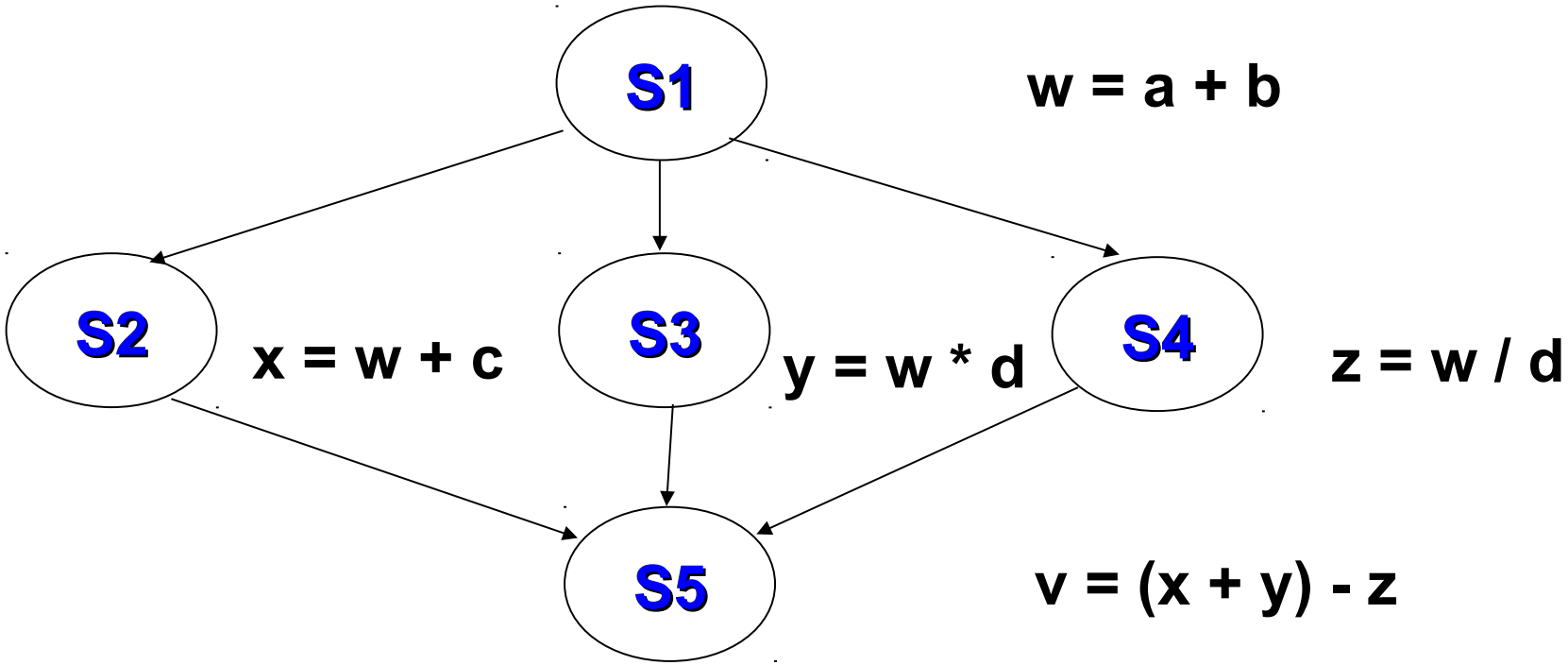
$W(S_1) = (x)$ $W(S2) = (y)$ $W(S3) = (z)$

1. $R(S_1) \cap W(S_3) = (a,b) \cap (z) = (\emptyset)$
2. $W(S_1) \cap R(S_3) = (x) \cap (x,b) <> (\emptyset)$
3. $W(S_1) \cap W(S_3) = (x) \cap (z) = (\emptyset)$

S1 y S3 NO pueden ejecutarse en paralelo o concurrentemente

Grafos de precedencia

Un grafo de precedencia es un grafo sin ciclos donde cada nodo representa una única sentencia o un conjunto secuencial de instrucciones agrupadas.



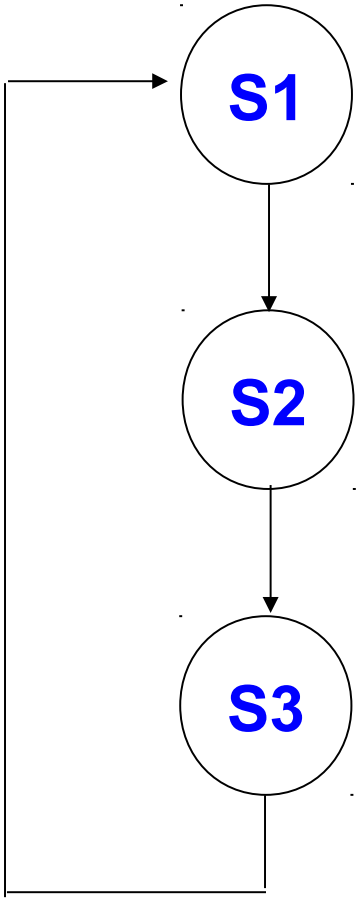
S2, S3 y S4 sólo podrán ejecutarse una vez finalizada S1

S2, S3 y S4 pueden ejecutarse en paralelo

S5 sólo podrá ejecutarse una vez finalizadas S2, S3 y S4

Grafos de precedencia

Un grafo de precedencia es un grafo sin ciclos donde cada nodo representa una única sentencia o un conjunto secuencial de instrucciones agrupadas.



x = a + b

y = x + c

a = y + 1

Grafo con ciclo.

S2 sólo puede ejecutarse una vez finalizada S1. que depende de S3 la cual, a su vez, sólo podrá ejecutarse una vez finalizada S2

Ventajas y Desventajas

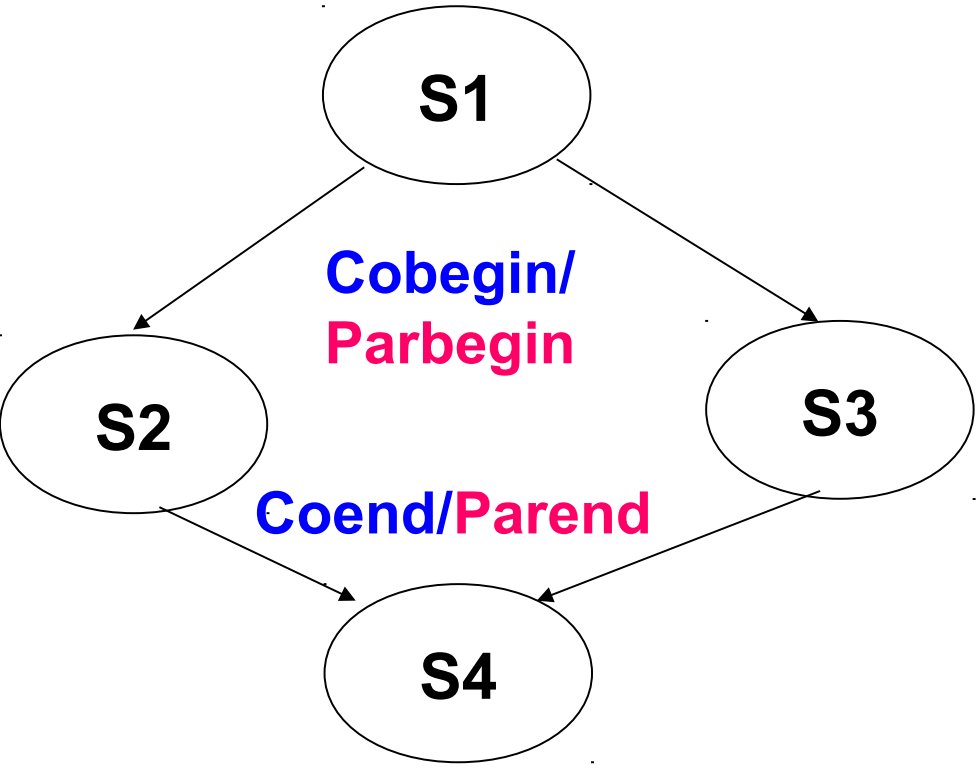
Permite sacar ventaja de :

- **Paralelismo de operaciones de CPU y E/S**
- **Existencia de varios procesadores**

- **La ejecución concurrente de procesos introduce nuevos problemas:**
  - **Comunicación y sincronización entre procesos.**
  - **Existencia de secciones criticas.**
  - **Posibilidad de interbloqueo (bloqueo o deadlock)**

Expresión Cobegin/Coend (Parbegin/Parend)

Las instrucciones ubicadas entre Cobegin y Coend (Parbegin y Parend) pueden ejecutarse concurrentemente.

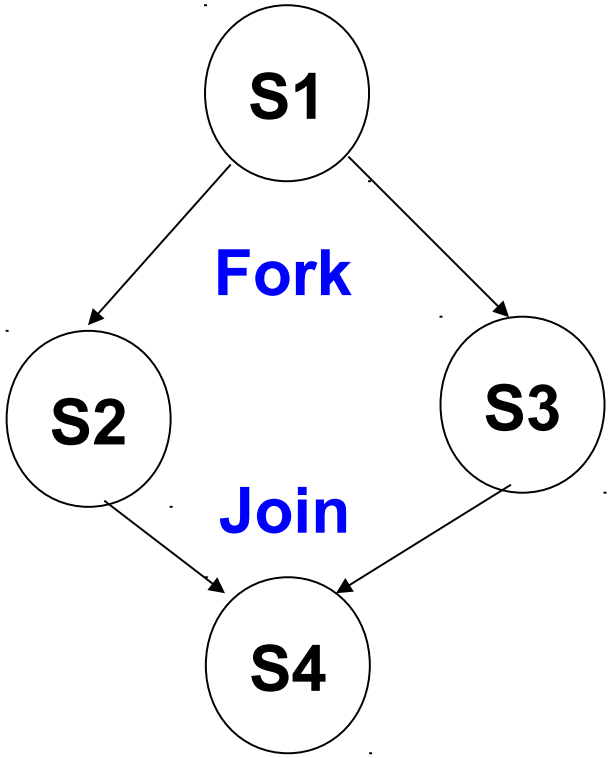



```
S1;
Cobegin
S2;
S3;
Coend
S4
```

```
S1;
Parbegin
S2;
S3;
Parend
S4
```

Expresión Fork/Join

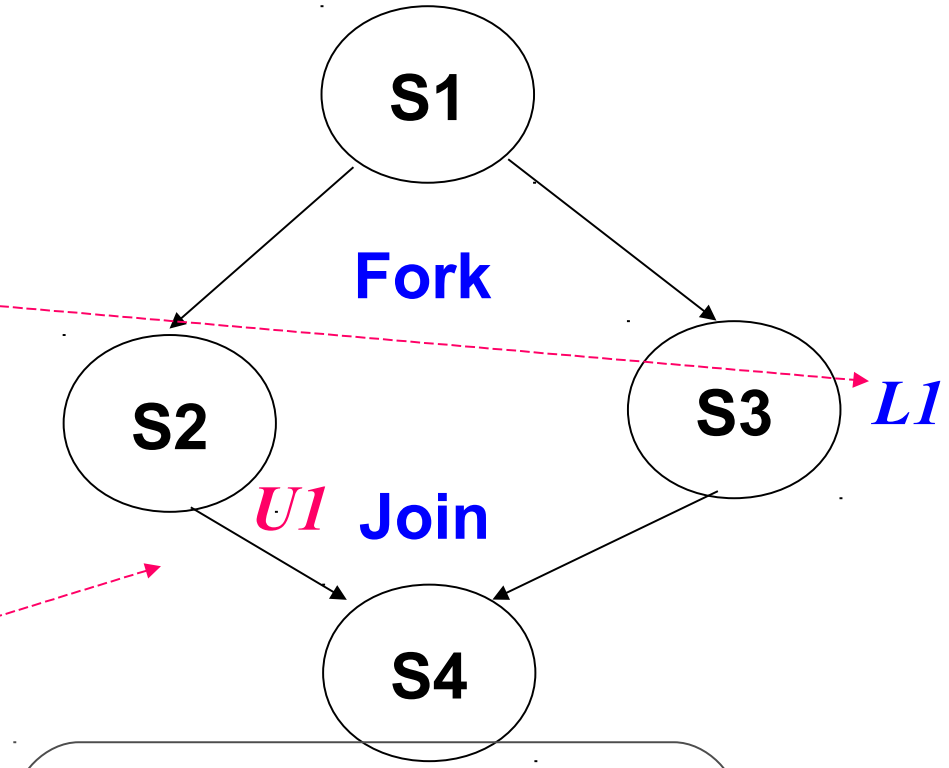
Un *fork* divide el flujo de control en dos threads, indicando con una etiqueta dónde comienza la ejecución del segundo thread.



Un *join* permite unir varios threads paralelos en uno solo.

Expresión Fork/Join

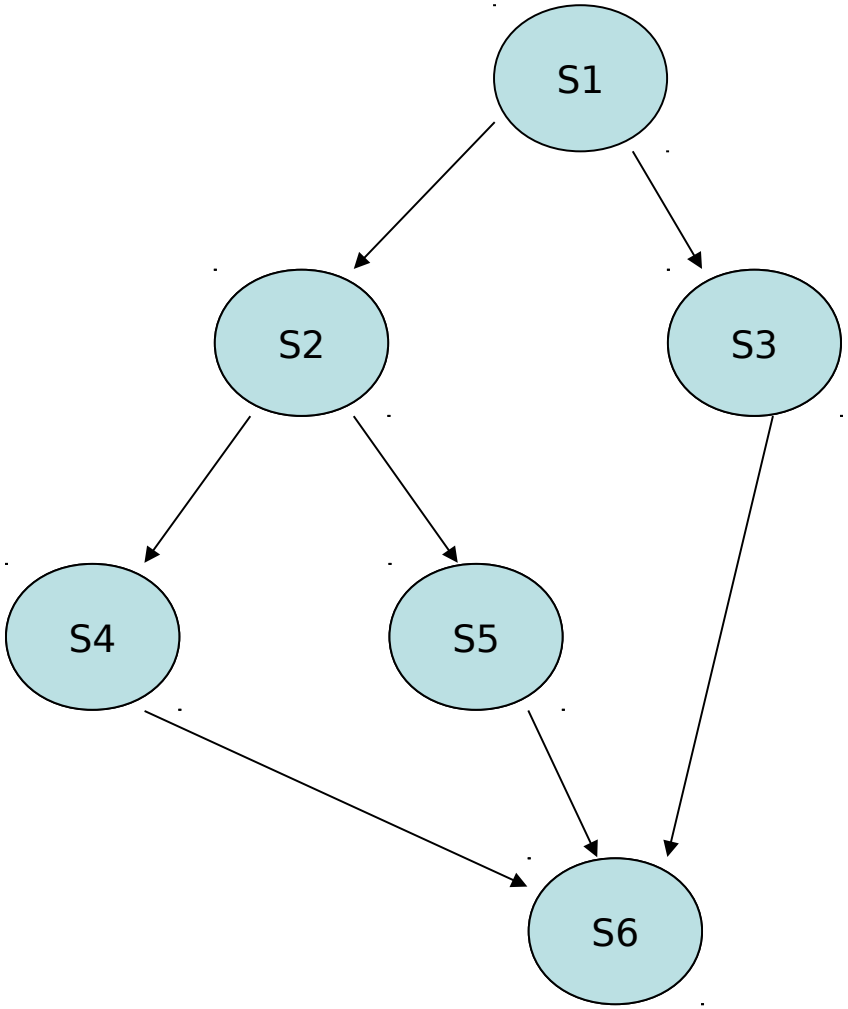
```
S1;
Cont = 2;
Fork L1;
S2;
Goto U1;
L1: S3;
U1: join Cont;
S4;
```



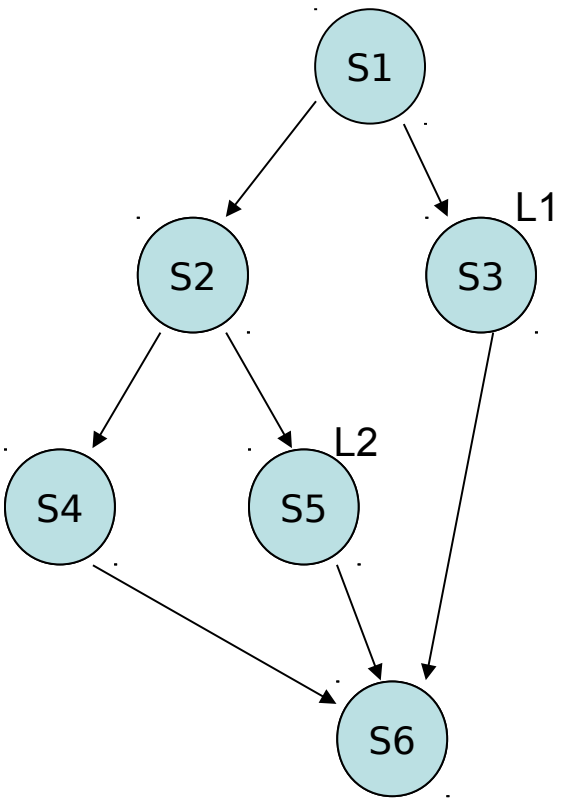
Cont:= Cont-1
¿Cont =0?
Sí → ejecutar siguiente
No → esperar

Ejercitación

Resolver usando: fork / join y cobegin / coend



Ejercitación




```
S1;
Cont = 3;
Fork L1;
S2;
Fork L2;
S4;
Goto U1;
L2: S5;
Goto U1;
L1: S3;
U1 join  cont;
S6;
```

```
Begin
    S1;
    Cobegin
        S3;
        Begin
            S2;
            Cobegin
                S4;
                S5;
            Coend
        End
    Coend
    S6;
End
```

S3 puede ejecutarse concurrentemente con S2, S4 y S5

Ejercitación

Confeccionar el grafo de precedencia a partir de los siguientes códigos

```
S0;
Cont = 4;
Fork L1;
Fork L2;
S1;
S4;
Goto U1;
L1: S2;
    Fork L3;
        S5;
        Goto U1;
L3: S6;
    Goto U1;
L2: S3;
U1 join  cont;
S7;
```

```
Begin
    S0;
    Cobegin
        S3;
        Begin
            S1;
            S4;
        End
        Begin
            S2;
            Cobegin
                S5; S6;
            Coend
        End
    Coend
    S7;
End
```