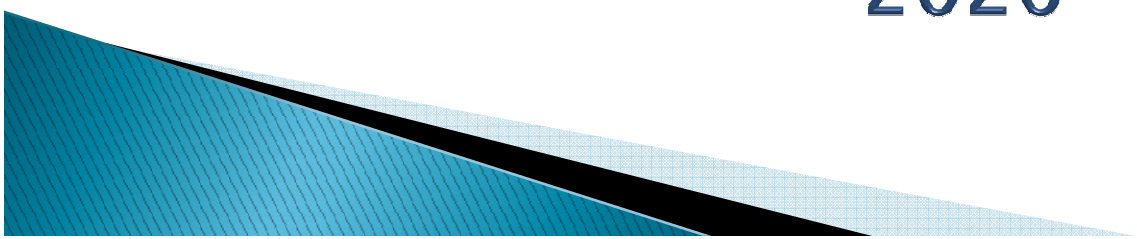
Programación Concurrente y Paralelo

3er año – Informática

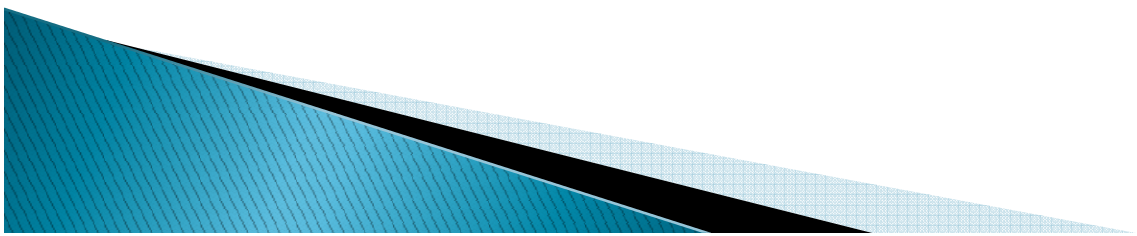
Ing. María del Milagro Paredes
Escuela de Minas

2020

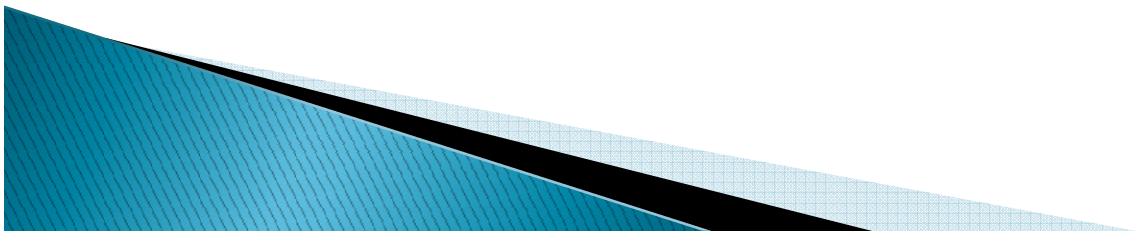


Contenido

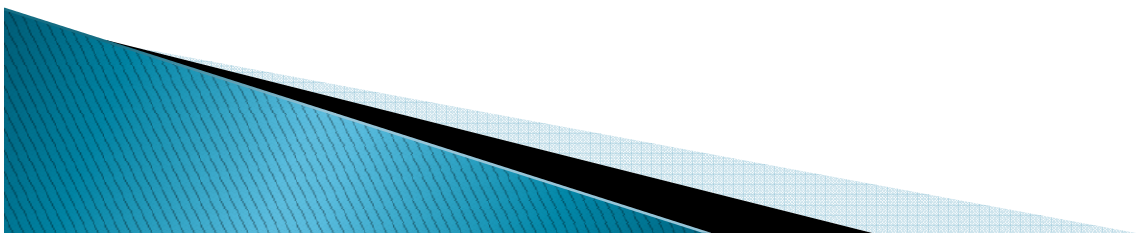
- Condiciones de Bernstein
- Grafos de Precedencia
- Sentencias Cobegin/Coend



- Una alternativa para aumentar el rendimiento de una máquina puede ser poner a trabajar varios elementos de un proceso simultáneamente , dicho de otra forma tratar de desarrollar el paralelismo; así el trabajo se repartirá y se tardará menos tiempo en completar el proceso de cálculo.
- La palabra concurrencia se refiere a la ejecución simultánea de diferentes acciones.
- Un proceso es un fragmento de un programa en ejecución.
- Un programa concurrente está compuesto por varios programas secuenciales que pueden ejecutarse simultáneamente, compartiendo el acceso a unos recursos compartidos.



- De la compartición de recursos surge la necesidad de arbitrar mecanismos adecuados para permitir el acceso, ya sea para leer, ya sea para modificar el recurso compartido, de manera que se garantice que la información que proporciona en cualquier momento sea coherente.
- Esta necesidad de coherencia, lleva a la cuestión de la sincronización entre los programas.
- Para ello se estudiarán diferentes herramientas propuestas históricamente para manejar la ejecución simultanea de programas.



ESPECIFICACIÓN DE EJECUCIÓN CONCURRENTE

¿Qué se puede ejecutar concurrentemente?

No todas las partes se pueden ejecutar en forma concurrente. Considerando el siguiente fragmento de programa:

x:=m +1;

y: = x +2;

-Esta claro que la primera sentencia debe ejecutarse antes que la segunda, sin embargo si consideramos ahora:

x:=1;

y:=2;

z:=3;

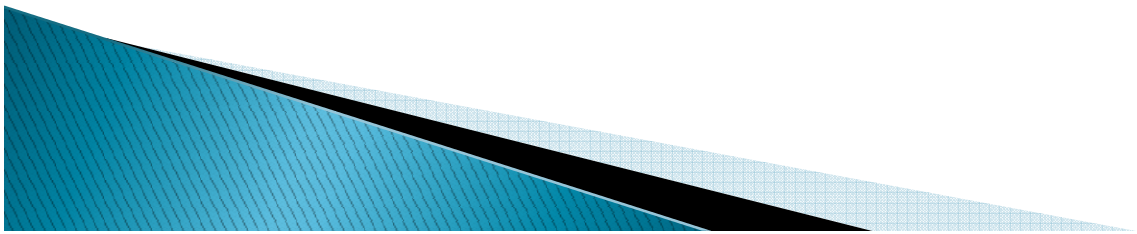
-Se puede observar que el orden en que se ejecuten no interviene en el resultado final. Si se dispusiera de 3 procesadores, se podría ejecutar cada una de las líneas en uno de ellos, incrementando la velocidad del sistema.

Bernstein, definió unas condiciones para determinar si dos conjuntos de instrucciones S_i y S_j se pueden ejecutar concurrentemente.

Condiciones de Bernstein.

- Bernstein (1966) descubrió una serie de condiciones para que dos procesos pudieran ejecutarse en paralelo.

- Las condiciones de Bernstein pueden ayudar a determinar qué es lo que se puede ejecutar de forma concurrente y lo que no, aunque como resumen, se puede decir que cualquier número de procesos puede leer de un determinado recurso de forma concurrente. Sin embargo, el que haya un proceso escribiendo en el recurso lo hace incompatible tanto con otros procesos escritores como con procesos lectores.



Para poder determinar si dos conjuntos de instrucciones se pueden ejecutar de forma concurrente, se define en primer lugar los siguientes conjuntos:

- $L(S_k)=\{a_1,a_2,.....,a_n\}$, como el **conjunto de lectura** del conjunto de instrucciones S_k y que esta formado por todas las variables cuyos valores son referenciados (se leen)durante la ejecución de las instrucciones en S_k.
- $E(S_k)= \{b_1,b_2,.....,b_m\}$, como el **conjunto de escritura** del conjunto de instrucciones S_k y que esta formado por todas las variables cuyos valores son actualizados (se escriben) durante la ejecución de las instrucciones en S_k.

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

1) $L(S_i) \cap E(S_j)= \emptyset$
2) $E(S_i) \cap L(S_j)= \emptyset$
3) $E(S_i) \cap E(S_j)= \emptyset$

El estudio de estas condiciones requiere algunas definiciones previas:

- Definiremos el conjunto de entrada, conjunto de lectura o dominio de un proceso P_i, como el conjunto de todas las variables de entrada necesarias para la ejecución de ese proceso. A este conjunto lo denotaremos I_i.
- De forma similar, se llama conjunto de salida, conjunto de escritura o rango de un proceso P_i, al conjunto formado por todas las variables modificadas después de la ejecución de P_i. A este conjunto lo denominaremos O_i.
- Ambos conjuntos, el de entrada y el de salida, están formados por operandos o resultados que pueden residir en registros del procesador o memoria.
- Sean dos procesos P_1 y P_2, cuyos conjuntos de entrada son I_1 e I_2 y cuyos conjuntos de salida son O_1 y O_2, respectivamente. Estos dos procesos se podrán ejecutar en paralelo, y se denotará mediante $P_1||P_2$, si se verifican las siguientes condiciones:

$$O_1 \cap I_2 = \emptyset$$

La intersección entre el conjunto de vbles leidas por uno, con el conjunto de vbles que escribe el otro, debe ser nulo y viceversa.

$$I_1 \cap O_2 = \emptyset$$

$$O_1 \cap O_2 = \emptyset$$

La intersección del conjunto de vbles que escribe uno, y el conjunto que escribe el otro, debe ser nulo.

- Estas ecuaciones son conocidas como condiciones de Bernstein y expresan la independencia de salida de los procesos P_1 y P_2.
- Los resultados de dos procesos que pueden ejecutarse en paralelo son los mismos si se ejecutan secuencialmente, en cualquier orden.
- En general, un conjunto de procesos,P_1,P_2…P_n, pueden ejecutarse en paralelo, y se representará $P_1||P_2||\ldots||P_n$, si ,y sólo si , las condiciones de Bernstein se cumplen con todas las parejas de procesos, es decir: $\Longleftrightarrow$ $\forall$

$$P_1||P_2||\ldots||P_n \qquad P_i||P_j \qquad i \neq j$$

EJEMPLO 1:

- Consideremos el caso más sencillo en que los procesos son instrucciones simples. Sean los siguientes procesos:

P1 $w = a * b$
P2 $x = a + w$
P3 $y = b - w$
P4 $z = x - y$

- Determinaremos ahora los conjuntos de entrada y salida de cada uno de los procesos.

$L(P1) = \{a,b\}$ $E(P1) = \{w\}$

$L(P2) = \{a,w\}$ $E(P2) = \{x\}$

$L(P3) = \{b,w\}$ $E(P3) = \{y\}$

$L(P4) = \{x,y\}$ $E(P4) = \{z\}$

- Y ahora comprobaremos las condiciones de Bernstein

$L(P1) \cap E(P2) = \emptyset$ $L(P1) \cap E(P3) = \emptyset$ $L(P1) \cap E(P4) = \emptyset$

$E(P1) \cap L(P2) \neq \{w\}$ $E(P1) \cap L(P3) \neq \{w\}$ $E(P1) \cap L(P4) = \emptyset$

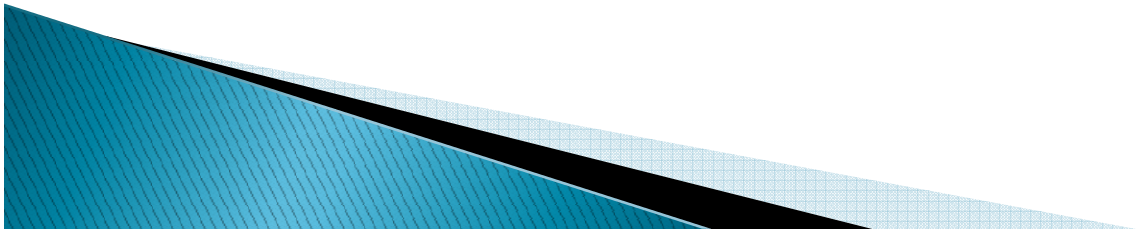
$E(P1) \cap E(P2) = \emptyset$ $E(P1) \cap E(P3) = \emptyset$ $E(P1) \cap E(P4) = \emptyset$

$L(P2) \cap E(P3) = \emptyset$ $L(P2) \cap E(P4) = \emptyset$ $L(P3) \cap E(P4) = \emptyset$

$E(P2) \cap L(P3) = \emptyset$ $E(P2) \cap L(P4) \neq \{x\}$ $E(P3) \cap L(P4) \neq \{y\}$

$E(P2) \cap E(P3) = \emptyset$ $E(P2) \cap E(P4) = \emptyset$ $E(P3) \cap E(P4) = \emptyset$

Donde $P1 \nparallel P2$, $P1 \nparallel P3$, $P1 || P4$, $P2 || P3$, $P2 \nparallel P4$ y $P3 \nparallel P4$



EJEMPLO 2:

$S_1 \longrightarrow$ a:=x + y;

$S_2 \longrightarrow$ b:=z - 1;

$S_3 \longrightarrow$ c:=a - b;

$S_4 \longrightarrow$ w:=c + 1;

Empleando las condiciones de Bernstein veremos que sentencias pueden ejecutarse de forma concurrente y cuáles no. Para ello, vamos a establecer los conjuntos de lectura y escritura correspondientes:

Denotamos por L(lectura) y E(escritura)

$L(S_1) = \{x,y\}$
$L(S_2) = \{z\}$
$L(S_3) = \{a,b\}$
$L(S_4) = \{c\}$

$E(S_1)=\{a\}$
$E(S_2)=\{b\}$
$E(S_3)=\{c\}$
$E(S_4)=\{w\}$

Se debe cumplir :

$L(Si) \cap E(Sj) = \emptyset$
$E(Si) \cap L(Sj) = \emptyset$
$E(Si) \cap E(Sj) = \emptyset$

Luego se aplica las condiciones de Bernstein a cada par de sentencias:

Entre $S_1 \cap S_2$:

1. $L(S_1) \cap E(S_2) = \emptyset$
2. $E(S_1) \cap L(S_2) = \emptyset$
3. $E(S_1) \cap E(S_2) = \emptyset$

Entre $S_1 \cap S_3$:

1. $L(S_1) \cap E(S_3) = \emptyset$
2. $E(S_1) \cap L(S_3) = a \neq \emptyset$
3. $E(S_1) \cap E(S_3) = \emptyset$

Entre $S_1 \cap S_4$:

1. $L(S1) \cap E(S4) = \emptyset$
2. $E(S1) \cap L(S4) = \emptyset$
3. $E(S1) \cap E(S4) = \emptyset$

Entre $S_2 \cap S_4$:

1. $L(S2) \cap E(S4) = \emptyset$
2. $E(S2) \cap L(S4) = \emptyset$
3. $E(S2) \cap E(S4) = \emptyset$

Entre $S_2 \cap S_3$:

1. $L(S2) \cap E(S4) = \emptyset$
2. $E(S2) \cap L(S4) = b \neq \emptyset$
3. $E(S2) \cap E(S4) = \emptyset$

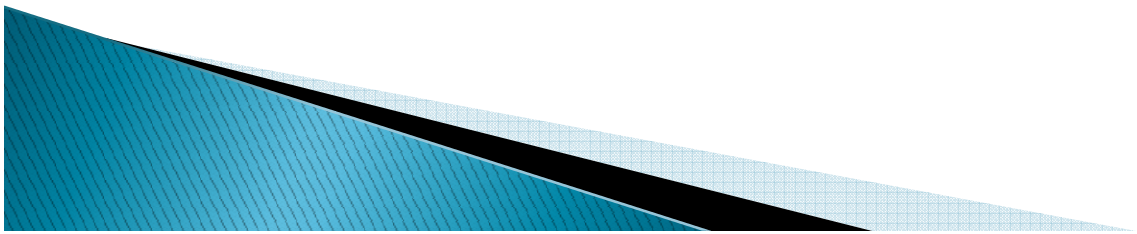
Entre $S_3 \cap S_4$:

1. $L(S3) \cap E(S4) = \emptyset$
2. $E(S3) \cap L(S4) = c \neq \emptyset$
3. $E(S3) \cap E(S4) = \emptyset$

De todo se deduce la siguiente tabla en la que puede verse que pares de sentencias pueden ejecutarse en forma concurrente:

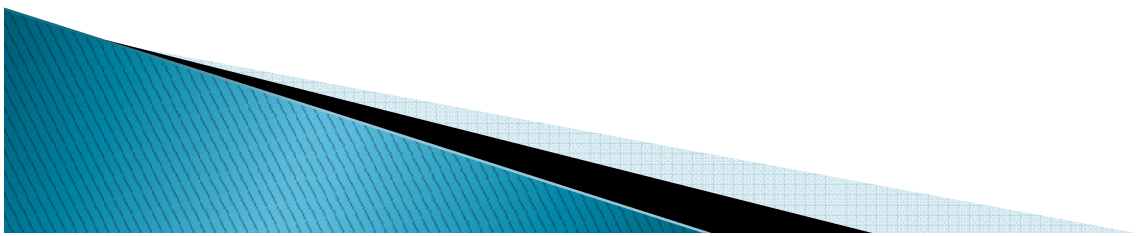
	S1	S2	S3	S4
S1	–	SI	NO	SI
S2	–	–	NO	SI
S3	–	–	–	NO
S4	–	–	–	–

Una vez que sabemos qué se puede y qué no se puede ejecutar concurrentemente, se hace necesario algún tipo de notación para especificar qué partes de un programa puede ejecutarse concurrentemente y qué partes no.



Como especificar la ejecución concurrente

Veremos 2 formas de especificar la ejecución concurrente de instrucciones basadas en una notación gráfica (**grafo de precedencia)** y otra basada en lo que suelen utilizar diversos lenguajes de programación, el par **cobegin/end**.

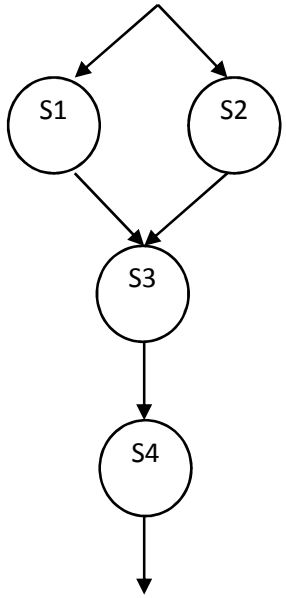


1-Grafos de precedencia de datos

Se trata de una notación gráfica. Es un grafo dirigido acíclico, cuyos nodos corresponden a sentencias individuales. Cada nodo representará una parte (conjunto de instrucciones) de nuestro sistema. Una flecha o arco desde un nodo S_i a un nodo S_j, significa que la sentencia S_j puede ejecutarse sólo después de que S_i haya terminado su ejecución. Si aparecen dos procesos en paralelo, querrá decir que se pueden ejecutar concurrentemente.

Ejemplo de grafo de precedencia:

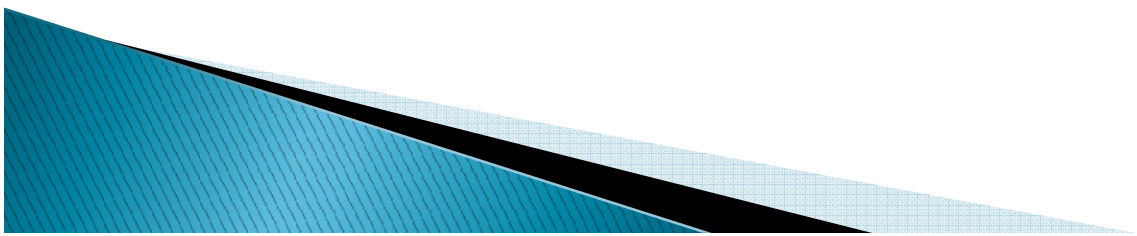
$S_1 \longrightarrow$ a:=x + y;
$S_2 \longrightarrow$ b:=z - 1;
$S_3 \longrightarrow$ c:=a - b;
$S_4 \longrightarrow$ w:=c + 1;



Grafos de precedencia de datos

Las condiciones de Bernstein son un poco engorrosas de aplicar, por lo que veremos una forma más práctica de detectar el paralelismo de los procesos.

- Un grafo de precedencia es un grafo dirigido cuyos nodos representan a procesos y cuyos arcos describen las relaciones de dependencia de datos entre ellos.
- Un grafo de precedencia es un grafo sin ciclos donde cada nodo representa una única sentencia. Un arco que parte del nodo S1 hacia el S2 indica que S2 puede ser ejecutado sólo si S1 ha completado su ejecución.



GRAFO DE PRECEDENCIA

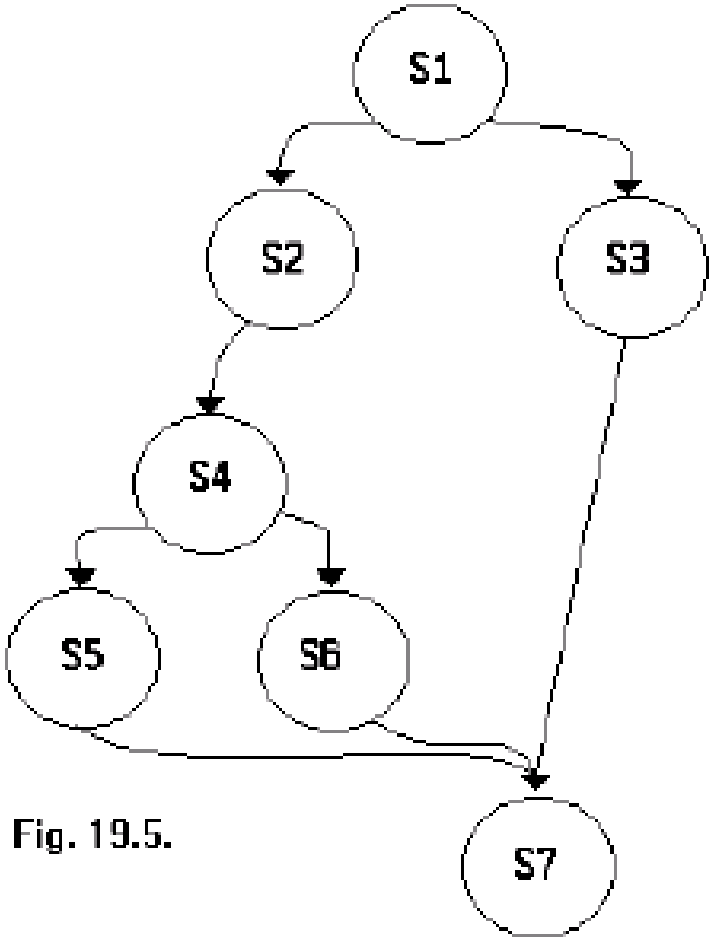
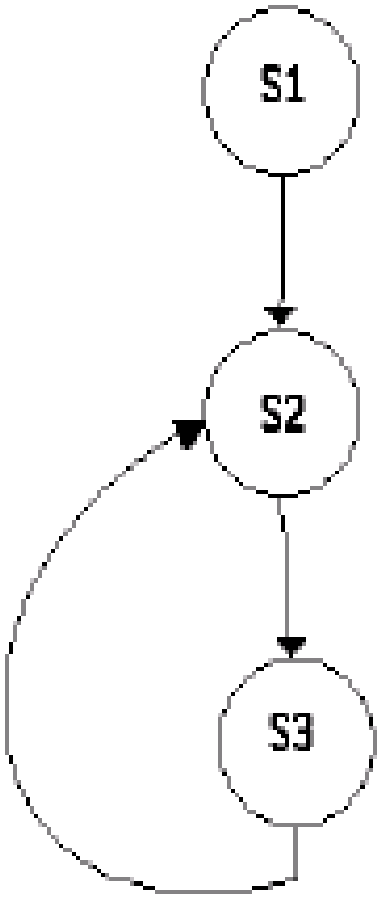


Fig. 19.5.

GRAFO DE NO PRECEDENCIA

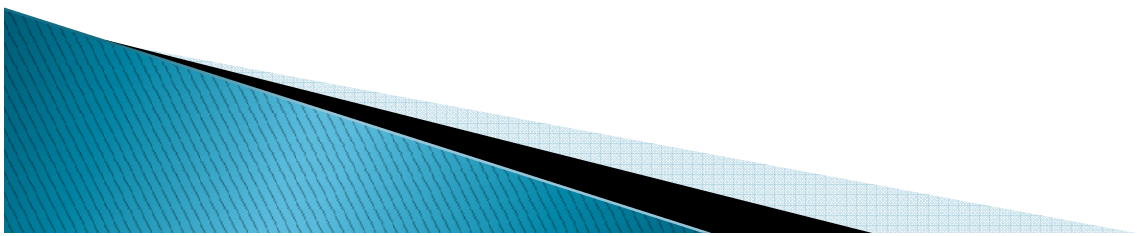


Grafos de precedencia de datos

Ejemplo:

Supongamos que debemos evaluar la expresión:

$$X = \frac{a * b + c}{a * b - d}$$



Grafos de precedencia de datos

Un programa para hacerlo constaría de la siguiente secuencia de procesos:

P1 u = a*b

P2 v = u + c

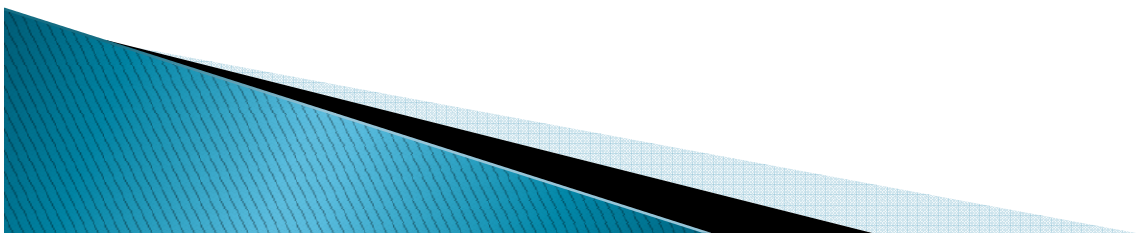
P3 w = u – d

P4 x = v/w

En un computador convencional, estos procesos deben ejecutarse de uno en uno, aunque no necesariamente en el orden indicado(los procesos P2 y P3 podrían intercambiarse)

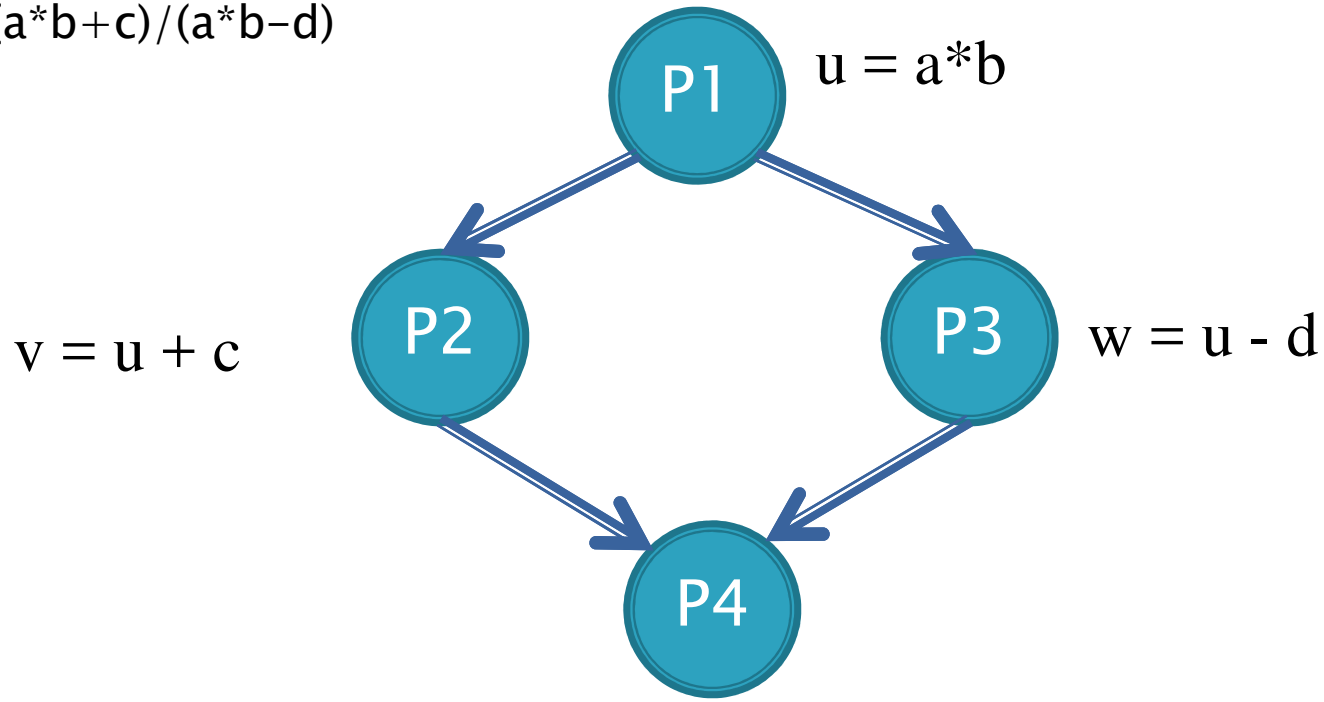
Sin embargo, en un computador paralelo, los procesos P2 y P3, podrían ejecutarse concurrentemente siempre que antes se haya terminado de ejecutar el proceso P1, debido a la existencia de datos existente.

Por otra parte, el proceso P4 no puede ejecutarse si antes no se han terminado de ejecutar los procesos P2 y P3.



Grafo de dependencia de datos para evaluar la expresión

x=(a*b+c)/(a*b-d)

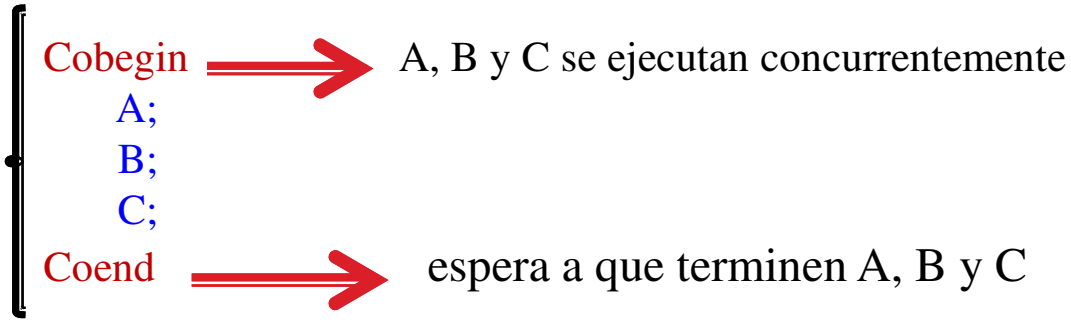


El grafo muestra de una manera clara las dependencia de datos existentes entre las operaciones

2-Creación de procesos estructurada: Sentencias COBEGIN-COEND

- Es otra forma de expresar la concurrencia.
- Todas aquellas acciones que puedan ejecutarse concurrentemente las introducimos dentro del par cobegin/coend.
- Cobegin S1| S2|…|Sn coend; ⟹ Ejecución concurrente de S1, S2,…, Sn
  - Termina cuando terminen todos los S_i.
  - Hace explícito qué rutinas van a ejecutarse concurrentemente.
  - Estructura: 1 única entrada y 1 única salida.

Ejemlo: Construcción estructurada



Conseguiremos ejecutar A, B, C de forma simultánea.

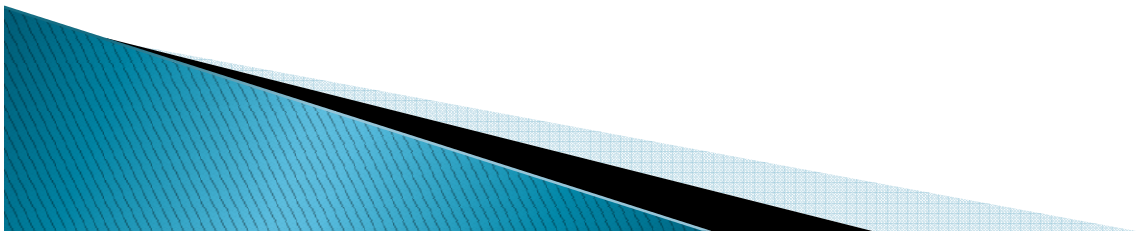
Sentencias COBEGIN-COEND:

El ejemplo anterior quedaría:

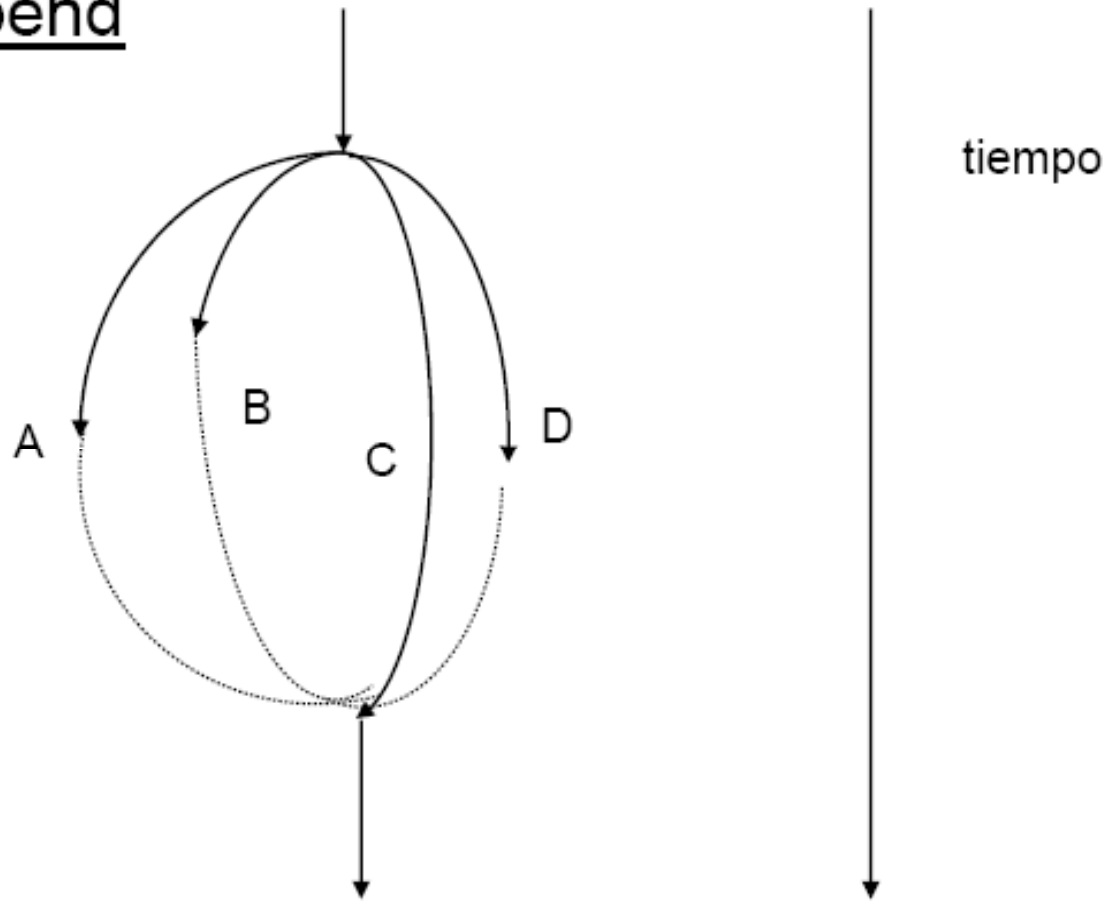
S_1 a:=x + y;
S_2 b:=z - 1;
S_3 c:=a - b;
S_4 w:=c + 1;

```
begin
        cobegin
                a:=x+y;
                b:=z-1;
        coend;
    c:=a-b;
   w:=c+1;
end.
```

Las instrucciones dentro del par cobegin/coend pueden ejecutarse en cualquier orden, mientras que el resto se ejecuta en manera secuencial.



cobegin / coend

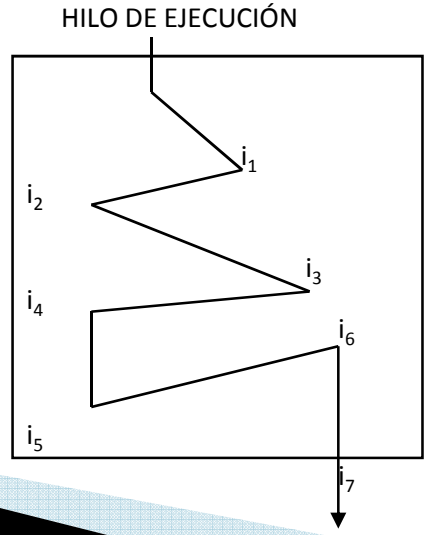


CARACTERÍSTICAS DE LOS SISTEMAS CONCURRENTES

La ejecución de sistemas concurrentes tiene algunas características que los diferencian de los sistemas secuenciales.

1. Orden de ejecución de las instrucciones:

- ✓ En los programas secuenciales, existe un orden total en la ejecución de las líneas de código. Ante un conjunto de datos de entrada se sabe siempre por dónde va a ir el programa(su flujo de ejecución).

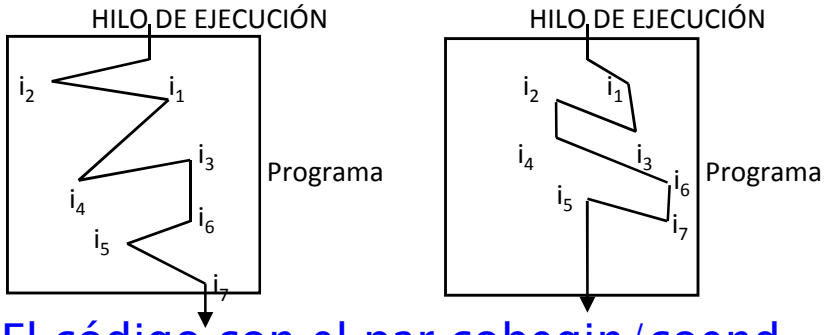


Por muchas veces que se ejecute el programa, el hilo de ejecución siempre tendrá el mismo recorrido.

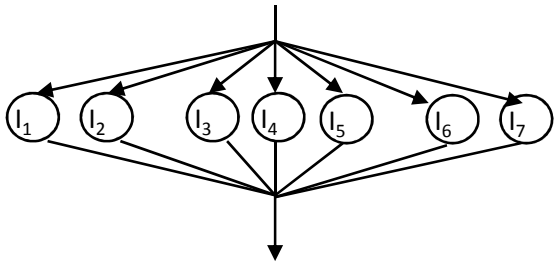
CARACTERÍSTICAS DE LOS SISTEMAS CONCURRENTES

- ✓ En los programas concurrentes hay un orden parcial. Ante el mismo conj. de datos de entrada no se puede saber cuál va a ser el flujo de ejecución. En cada ejecución del programa el flujo puede ir por distinto sitio.

En este ejemplo, se supone que todas las instrucciones pueden ejecutarse concurrentemente, por lo cual se puede ver como en dos ejecuciones distintas el orden en el que se ejecutan las instrucciones pueden variar.



El grafo de precedencia seria: sería:



El código con el par cobegin/coend

```
Begin
      Cobegin
       i1; i2; i3; i4; i5; i6; i7,
      coend;
end.
```

CARACTERÍSTICAS DE LOS SISTEMAS CONCURRENTES

2. **Indeterminismo:** el orden parcial lleva a que los programas concurrentes puedan tener un comportamiento indeterminista, es decir, puedan arrojar distintos resultados cuando se ejecutan repetidamente sobre el mismo conjunto de datos de entrada.

¿Qué valor tendrá la variable x al terminar la ejecución?. Todo hace pensar que seria 10, pero puede ser 5, 6, 7, 8, 9, 10.

```
Programa incógnita
  Var x:integer;
Process P1;
  Var i:integer;
Begin
    For i:=1 to 5 do  x:=x+1;
End;
Process P2;
  Var j:integer;
Begin
    For j:=1 to 5 do  x:=x+1;
End;
```

```
Begin
   x:=0;

cobegin
      P1;
      P2;
 Coend;

End;
```

x= vble compartida por ambos procesos

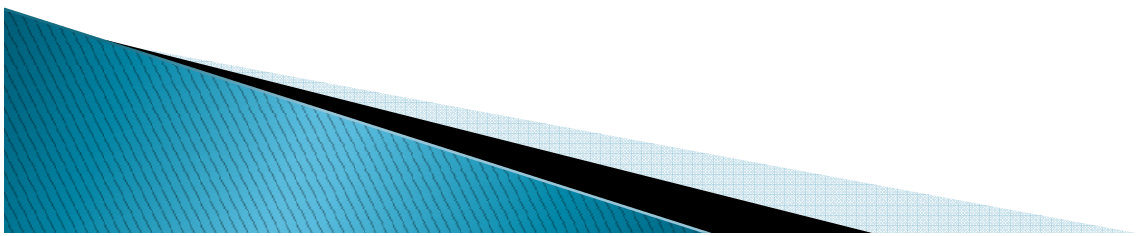
Programa en el que dos procesos se ejecutan concurrentemente para sumar 1 a la variable x.

Intuitivamente podemos observar que el error está en el acceso incontrolado a la variable compartida por parte de los dos procesos.

Paradigmas de resolución de programas concurrentes

La mayoría de las aplicaciones concurrentes utilizan cinco tipos de soluciones, o *paradigmas:*

1) Paralelismo iterativo
2) Paralelismo recursivo
3) Productores y consumidores (pipelines o workflows)
4) Clientes y servidores
5) Pares que interactúan o paralelismo entre iguales



1)Paralelismo Iterativo

En el **paralelismo iterativo** un programa consta de un conjunto de procesos (posiblemente idénticos) c/u de los cuales tiene 1 o más loops(bucles). Luego, cada proceso es un programa iterativo.

Un **programa interactivo** es aquél que necesita la realimentación continúa del usuario para poder ejecutarse.

Iteración significa el acto de repetir un proceso con el objetivo de alcanzar un resultado deseado. Cada repetición del proceso también se le denomina una "iteración", y los resultados de una iteración se utilizan como punto de partida para la siguiente iteración.

En un programa iterativo se emplea, necesariamente, ciclos o bucles para realizar las iteraciones o vueltas.

Un **bucle** o **ciclo**, en programación, es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

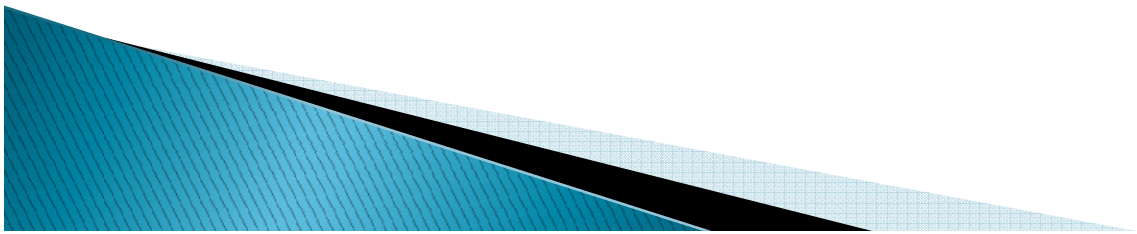
1)Paralelismo Iterativo

Los procesos cooperan para resolver un único problema (por ej. un sistema de ecuaciones), pueden trabajar independientemente, comunicarse y sincronizar por memoria compartida o Paso de mensaje.

```
var i=0, a := 0          // inicializo a antes de comenzar la iteración
for i from 1 to 3 {   // ciclo 3 veces
    a := a + i        // incremento a con el valor actual de i
    print a              // se imprime el número 6
    }
```

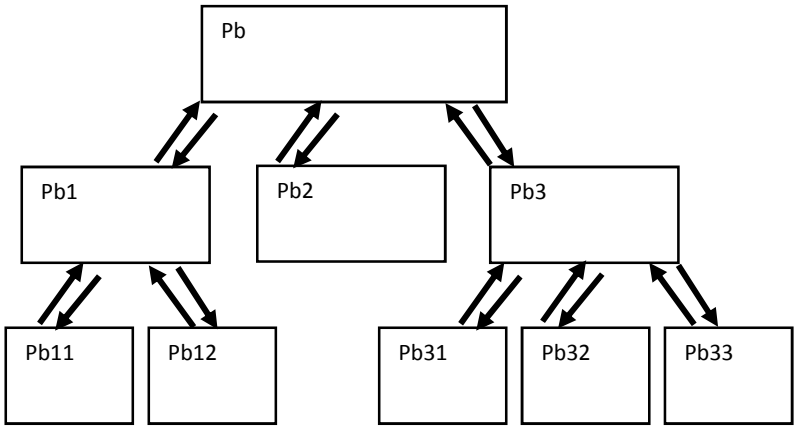
En este fragmento de programa, el valor de la variable *i* cambia a medida que la ejecución del programa progresa, tomando los valores 1, 2 y 3. Este cambio de valor —o *estado mutable*— es característico de una iteración.

Otro ejemplo seria , calcular la suma dentro una lista. Se necesita de una instrucción FOR, WHILE o REPEAT para recorrerla. En cada iteración se va acumulando la suma. Por tanto se hacen uso de variables acumuladores, iteradoras, banderas (o flags).

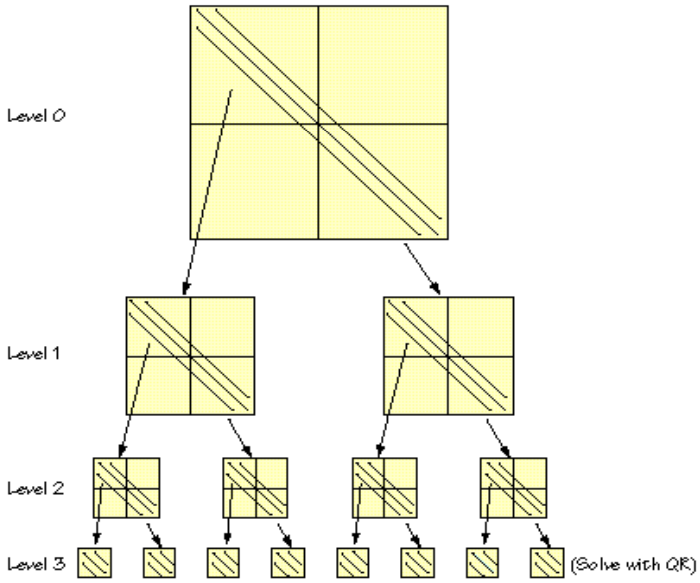


2)Paralelismo recursivo

En el **paralelismo recursivo** el problema general (programa) puede descomponerse en procesos recursivos que trabajan sobre partes del conjunto total de datos (Dividir y conquistar) .



Divide y venceras

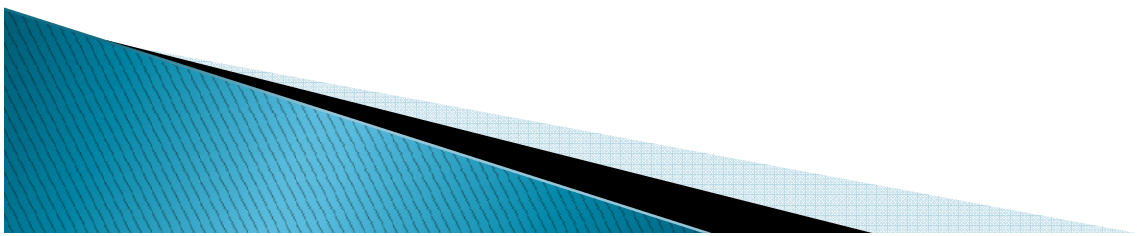


2)Paralelismo recursivo

En las ciencias de la computación, el término **divide y vencerás** (**DYV**) hace referencia a uno de los más importantes paradigmas de diseño algorítmico.

El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas independientes. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original.

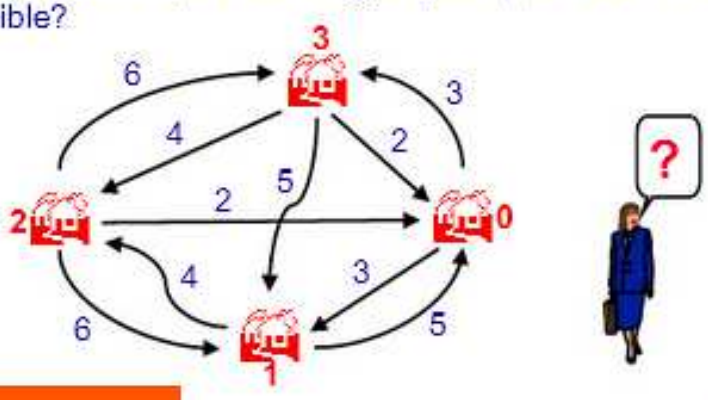
Ejemplos clásicos son el sorting by merging (ordenamiento por mezcla), el cálculo de raíces en funciones continuas, problema del viajante, juegos (tipo ajedrez) , multiplicar números grandes.



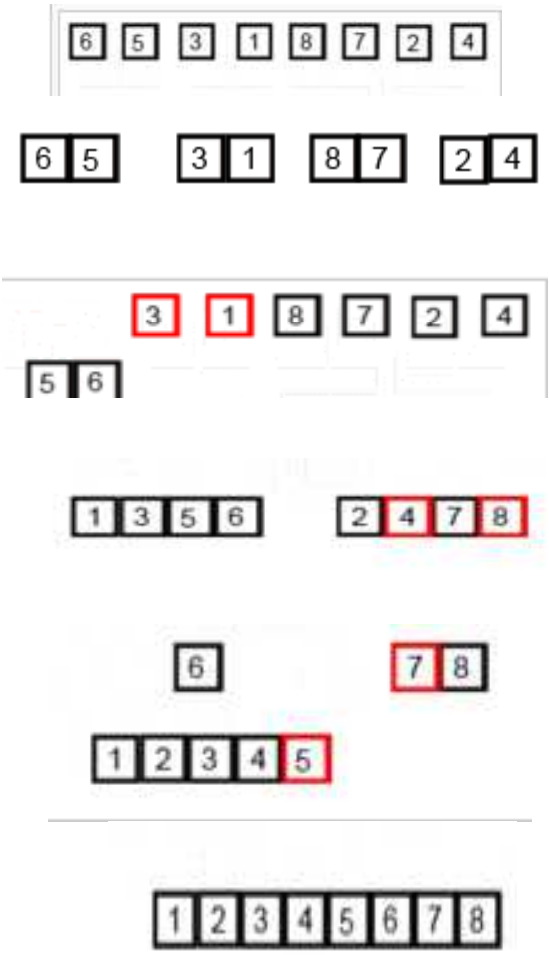
2)Paralelismo recursivo

El Problema del Viajante

- Un viajante tiene que recorrer n ciudades y regresar a la ciudad de partida, sin pasar dos veces por la misma ciudad.
- Se supone que los caminos son unidireccionales y se conoce la distancia de cada camino.
- ¿Cuál es recorrido que debe seguir para que sea lo más corto posible?



Ordenamiento recursivo por mezcla



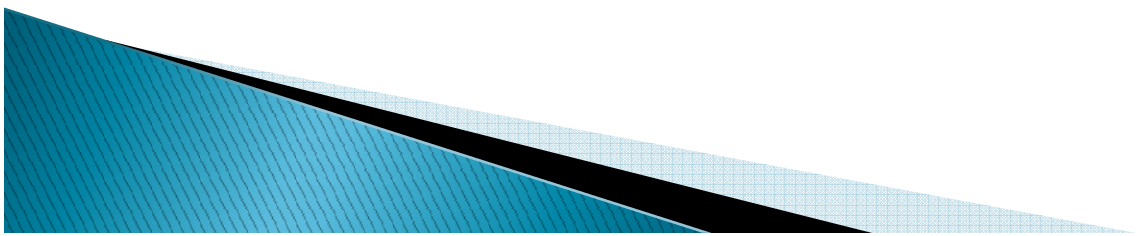
3)Productor-Consumidor

En el esquema productor-consumidor, los procesos productores producen elementos de información que se almacenan en un buffer compartido

hasta ser retirados por los procesos consumidores.

Si el buffer está vacío, los consumidores esperan; si está lleno, esperan los productores hasta que se libere espacio.

La comunicación por medio de tuberías se basa en la interacción **productor/consumidor**, los procesos productores (aquellos que envían datos) se comunican con los procesos consumidores (que reciben datos) siguiendo un orden FIFO Una vez que el proceso consumidor recibe un dato, éste se elimina de la tubería.



3)Productor-Consumidor

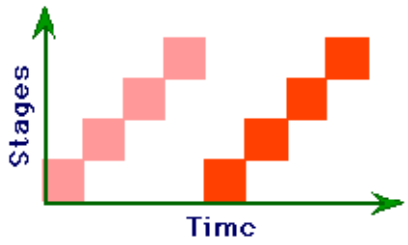
Los esquemas **productor-consumidor** muestran procesos que se comunican. Estos procesos generan datos que han de usar otros procesos.

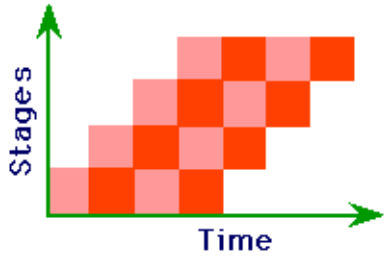
Es habitual que estos procesos se organicen en pipes a través de los cuales fluye la información.

Cada proceso en el pipe o tubería es un filtro que consume la salida de su proceso predecesor y produce una salida para el proceso siguiente.

Ejemplos a distintos niveles de SO.

Pipe





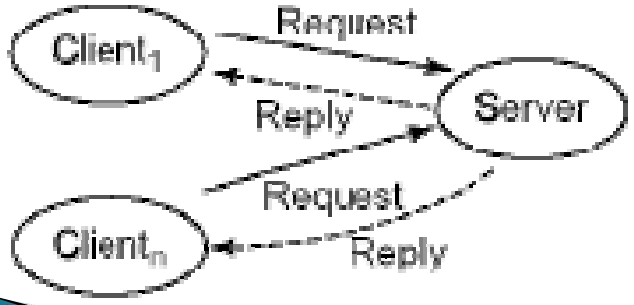
tubería (*pipe*,) consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos.

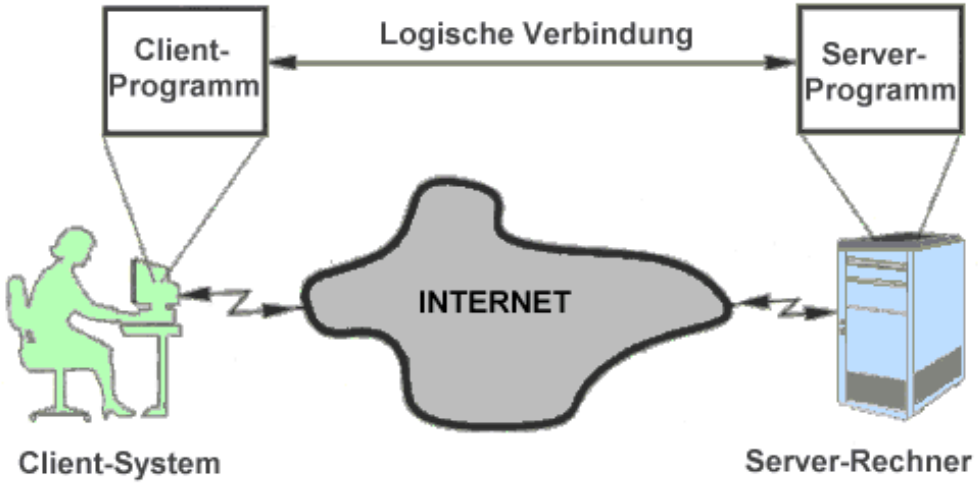
4)Cliente-Servidor

Cliente-servidor es el esquema dominante en las aplicaciones de procesamiento distribuido.

Los servidores son procesos que esperan pedidos de servicios de múltiples clientes. Unos y otros pueden ejecutarse en procesadores diferentes. Comunicación bidireccional. Atención de a un cliente o con multithreading a varios.

El soporte distribuido puede ser simple (LAN) o extendido a la WEB.



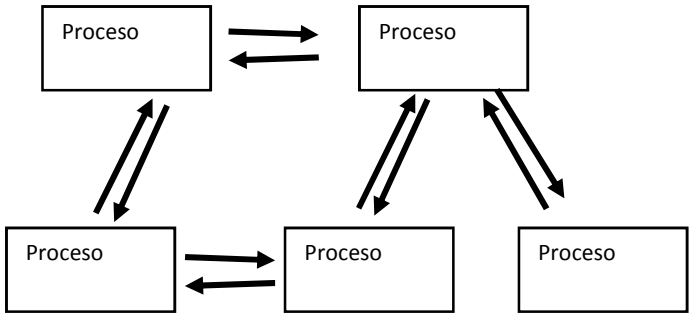


5)Pares que interactúan

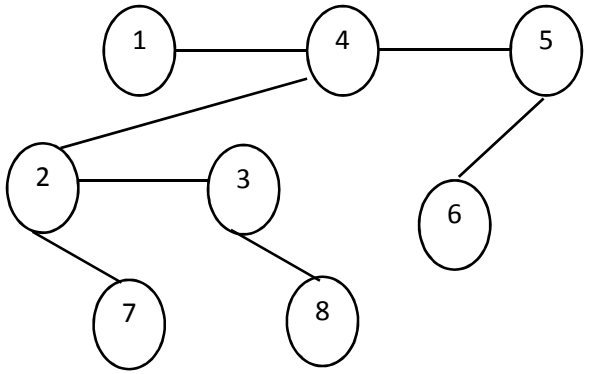
En los **esquemas de pares que interactúan** los procesos (que forman parte de un programa distribuido) resuelven partes del problema (normalmente mediante código idéntico) e intercambian mensajes para avanzar en la tarea y completar el objetivo.

Permite mayor grado de asincronismo que C/S.

Configuraciones posibles: grilla, pipe circular, uno a uno, arbitraria



Pares que interactúan



FIN TEORIA 2