Tema 1

- Programa secuencial: las instrucciones se ejecutan una tras otra, en secuencia.
- Programa concurrente: conjunto de programas secuenciales que se pueden ejecutar lógicamente en paralelo.
- Proceso: ejecución de un programa secuencial.
- Concurrencia: solapamiento real o virtual de varias actividades. Es el potencial para ejecución paralela.
- Programación concurrente (PC): notaciones y técnicas de programación utilizadas para expresar paralelismo y resolver problemas de sincronización y comunicación. Es una abstracción
- Programación paralela: tiene el objetivo de acelerar la resolución de problemas mediante el aprovechamiento de la capacidad de procesamiento paralelo.
- Programación distribuida: consiste en que varios componentes software de distintos ordenadores trabajen juntos.
- Programación en tiempo real: consiste en programar sistemas que funcionan continuamente, recibiendo entradas y enviando salidas a/desde hardware (sistemas reactivos). Se trabaja con restricciones muy estrictas en cuanto a la respuesta temporal (sistemas de tiempo real).

La programación concurrente mejora la eficiencia y la calidad.

Mejora de la eficiencia

- Sistemas con un procesador
  * Cuando la tarea que tiene el control de la CPU necesita realizar una E/S cede el control a otra, evitando la espera ociosa de la CPU.
  * Permite que varios usuarios usen el sistema de forma interactiva (SO multiusuario)

- Sistemas con varios procesadores
  * Se pueden repartir las tareas, minimizando el tiempo de ejecución.
  * Fundamental para acelerar complejos cálculos numéricos.

Mejora de la calidad

Ejemplos:

* Servidor web de reserva de vuelos: se considera cada petición como un proceso.

* Gasolinera: se consideran surtidores, clientes, vehículos y empleados como procesos que cambian de estado al participar en tareas comunes.

Mecanismos de implementación de concurrencia

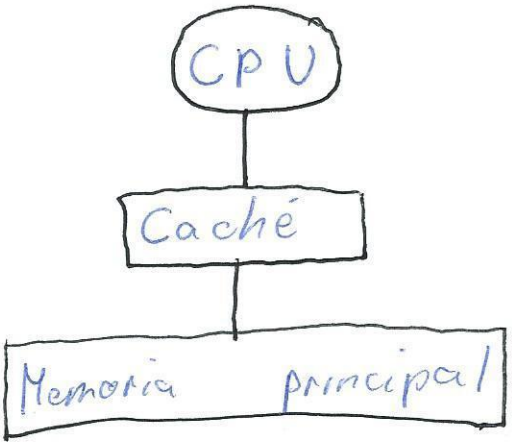
Dependen fuertemente de la arquitectura. Consisten en considerar una Máquina Virtual que representa un sistema multiprocesador/distribuido.

El tipo de paralelismo afecta a la eficiencia pero no a la corrección (el programa debe funcionar con independencia del entorno)

→ Concurrencia en sistemas monoprocesador.

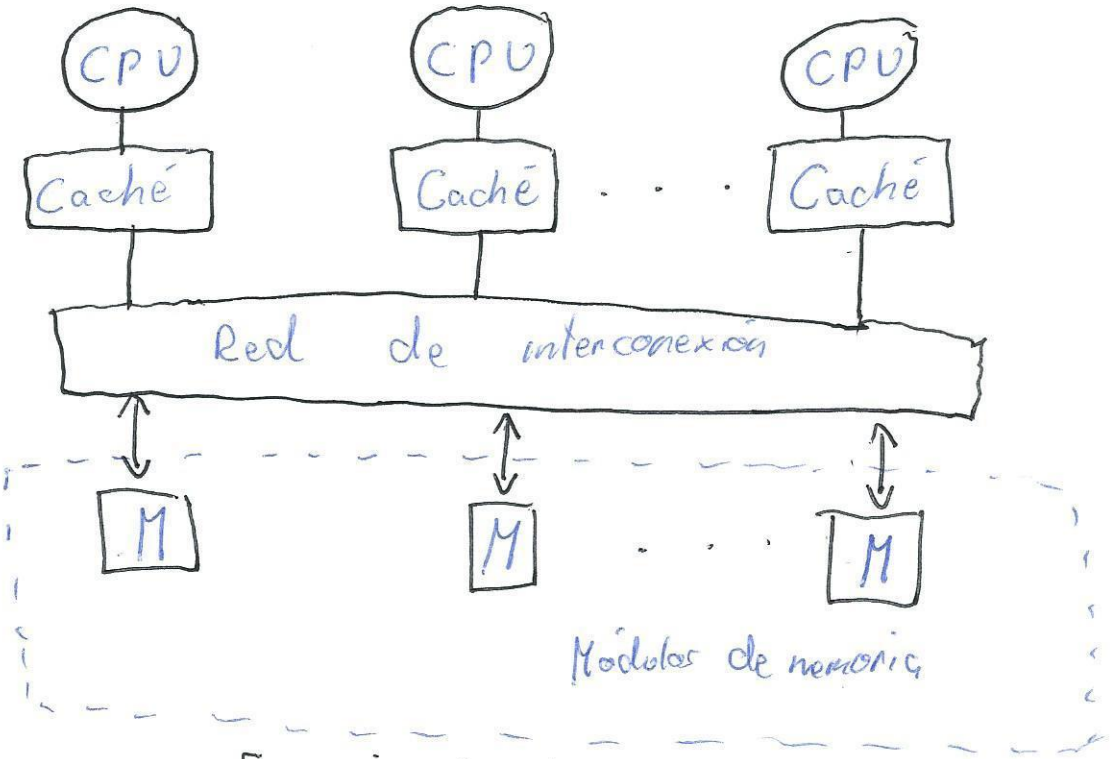
* Multiprogramación
  - Mejor aprovechamiento de CPU
  - Servicio interactivo a varios usuarios.
  - Permite usar un diseño concurrente
  - Sincronización y comunicación mediante variables compartidas

El SO gestiona como múltiples procesos se reparten la CPU



-> Concurrencia en multiprocesadores de memoria compartida

* Los procesadores tienen un espacio de direcciones compartido.
* La interacción entre procesos se puede implementar con variables compartidas.

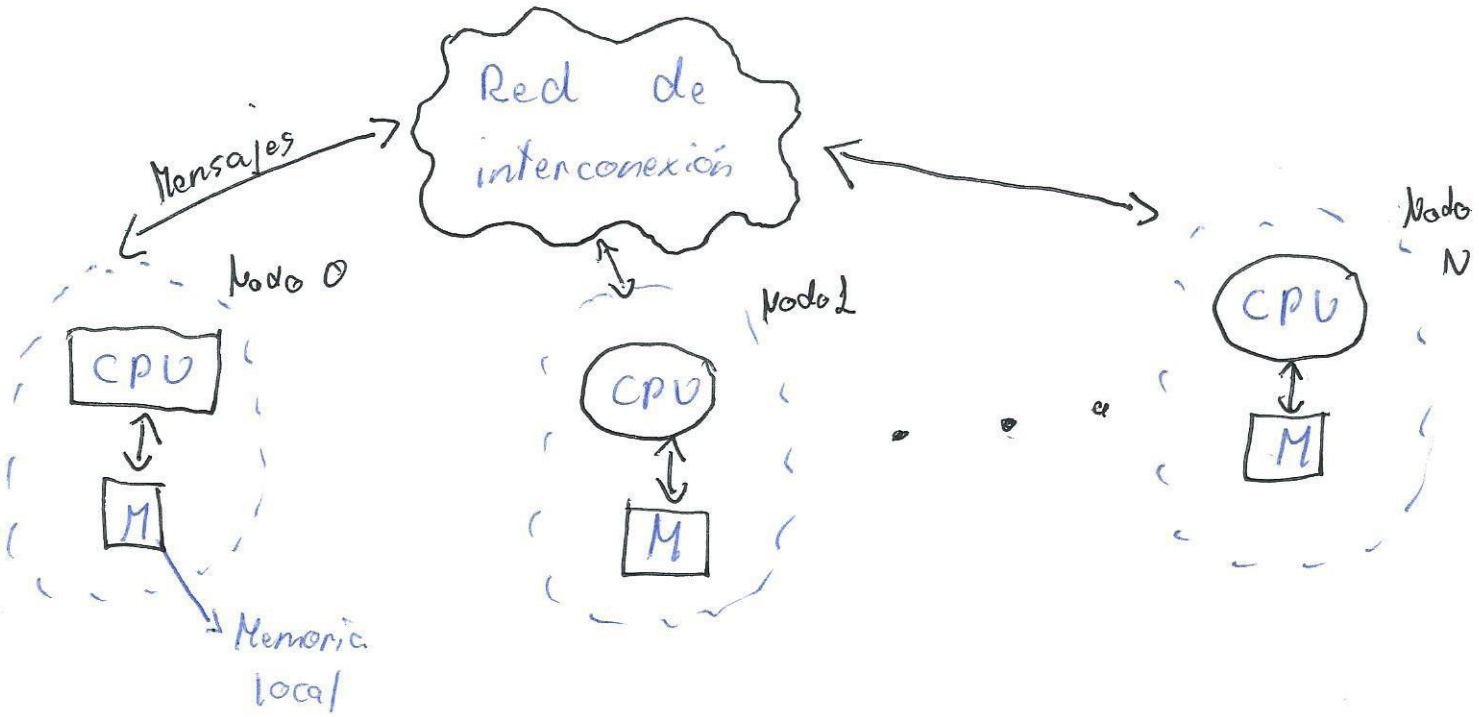


Espacio de direcciones compartido

Ejemplo: PC con procesador multi-core

→ Concurrencia en sistemas distribuidos

* Cada CPU tiene su espacio de direcciones privado
* Los procesadores interactúan con un paso de mensajes por red de interconexión.
* Programación distribuida: trata problemas como el tratamiento de fallos, transparencia, etc. además de la concurrencia.



Ejemplos: internet, intranet, clusters...

- Sentencia atómica: al ejecutarse de principio a fin nunca se ve afectada por otras secuencias en ejecución de otros procesos del programa.

  * El funcionamiento de una instrucción se define por su efecto en el estado de ejecución del programa al acabar
  * El estado de ejecución está formado por los valores de las variables y los registros de los procesos.

Ejemplos)

* Cargar un valor de memoria en un registro.
* Incrementar el valor de un registro.
* Escribir el valor de un registro en memoria.

Sentencia no atómica: $x := x + 1$

El compilador debe dar tres pasos:

1. Cargar x en un registro
2. Incrementar r
3. Escribir r en x

El resultado final puede variar si hay otras sentencias que actúen sobre x ejecutándose.

- Secuencia de interfoliación: mezcla de sentencias atómicas de dos procesos que se ejecutan de forma concurrente. Cada proceso debe ejecutar sus sentencias en orden consecutivo.

Ej) P_A y P_B procesos que se ejecutan concurrentemente

$A_1\ B_1\ A_2\ A_3\ B_2\ A_4\ B_3\ B_4\ B_5\ A_5$ ✓

$A_1\ B_1\ A_3\ A_2\ B_2\ B_3\ B_5\ B_4\ A_5\ A_4$ ✗

El entrelazamiento preserva la consistencia

Sea P un proceso compuesto por dos instrucciones atómicas I_0, I_1 tal que $P = (I_0 \| I_1)$

→ Si I_0 e I_1 no acceden a la misma celda de memoria, el orden de ejecución no afecta al resultado final.

→ Si $I_0 \equiv M \leftarrow 1$ e $I_1 \equiv M \leftarrow 2$, el resultado debe ser consistente. Es decir, al final $M = 1$ o $M = 2$, pero nunca $M = 3$.

Progreso finito

No se pueden hacer suposiciones sobre las velocidades de ejecución de los procesos. Un programa concurrente se entiende en base a sus procesos e interacciones, sin tener en cuenta el entorno de ejecución.

Si hiciéramos suposiciones temporales, la corrección sería difícil y dependería de la ejecución y su configuración.

Según la hipótesis del progreso finito, la velocidad de ejecución es no nula, lo que desencadena:

- Desde el punto de vista global: en cualquier momento de la ejecución de un programa concurrente habrá al menos un proceso preparado.
- Desde el punto de vista local: una sentencia de un proceso completará su ejecución en un espacio de tiempo finito.
- Estado de un programa concurrente: valores de las variables en un momento dado (incluyendo PC, registros, etc.).
- Historia/Traza de un programa concurrente: secuencia de estados $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$ producida por una secuencia de interpolación

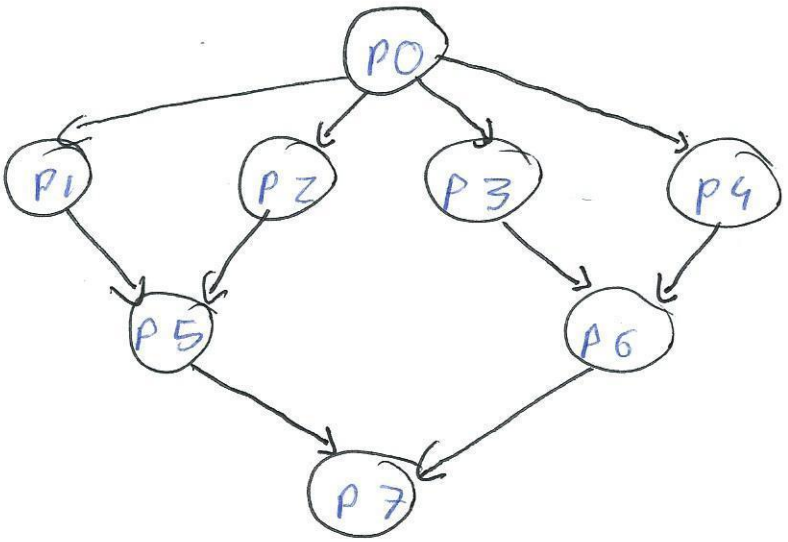
Notaciones de ejecución concurrente

- Propuestas iniciales: no separan la definición de los procesos de su sincronización
- Propuestas finales: separan conceptos e imponen estructura.
- Declaración de procesos: rutinas específicas de programación concurrente (hacen más clara la estructura).

- Sistemas estáticos
  - * Número de procesos fijado en el código del programa
  - * Los procesos se activan al lanzar el programa
  - Ejemplo: Message Passing Interface (MPI-1)
- Sistemas dinámicos
  - * Número variable de procesos/hebras que se pueden activar en tiempo de ejecución.
  - Ejemplos: OpenMP, MPI-2, etc.

Grafo de sincronización

- Dadas dos actividades A y B, una flecha de A a B significa que B no puede ejecutarse hasta que termine A



Estructura de un programa:

```
var ...        { vars. compartidas }

process Uno
  var ...      { vars. locales }
  begin
     { código }
  end
process Dos
  ...
end
...
```

El programa acaba cuando acaban todos los procesos.

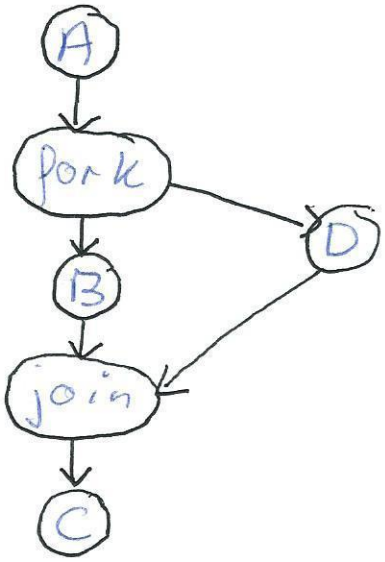
FORK - JOIN

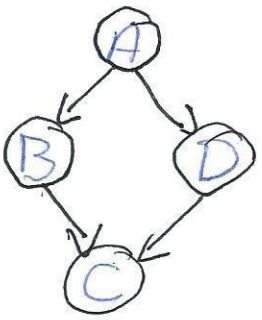
- fork: inicia una bifurcación: la rutina nombrada puede empezar a ejecutarse a la vez que la siguiente sentencia.

- join: espera a que termine la rutina nombrada antes de comenzar la siguiente sentencia.

→ Ventajas: práctica y potente

→ Inconvenientes: no estructurada, es difícil comprender los programas.





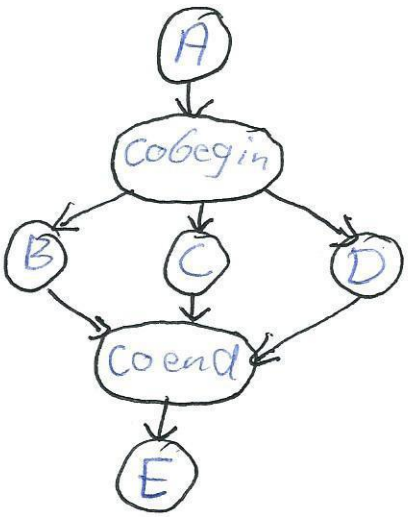

```
procedure P1
begin
    A;
    fork P2;
    B;
    join P2;
    C;
end
```

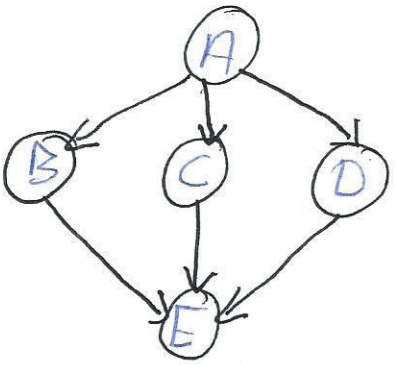
```
procedure P2;
begin
    D;
end
```

COBEGIN - COEND

Las sentencias del bloque delimitado por cobegin-coend se ejecutan a la vez. En coend se espera a que finalicen todas las sentencias

```
begin
    A;
    cobegin
        B; C; D;
    coend
    E;
end
```





- Ventajas: impone estructura → más fácil de entender.
- Inconveniente: no tan potente como fork-join

En un conjunto de procesos cooperativos (no independientes entre sí) algunas combinaciones de las secuencias de interpolación no son válidas:

→ Cuando esto ocurre, se dice que hay una condición de sincronización, una restricción en el orden de mezcla de las instrucciones de los procesos.

→ Un caso particular es la exclusión mutua: secuencias finitas de instrucciones que deben ejecutarse por un único proceso, sin que otro pueda ejecutarlas a la vez.

EXCLUSIÓN MUTUA

→ A las secuencias de instrucciones se las denomina sección crítica (SC).

→ Ocurre exclusión mutua (EM) cuando los procesos sólo funcionan correctamente si en cada instante de tiempo sólo uno de ellos ejecuta una instrucción de la sección crítica.

Ejemplos de exclusión mutua:

- Procesos con memoria compartida que acceden, leen y modifican estructuras de datos comunes usando operaciones no atómicas.
- Envío de datos a dispositivos no compartibles (ej. impresora).

Para indicar que una sentencia compuesta se debe ejecutar de forma atómica usaremos < >.

```
begin
    x:=0;
    cobegin
        < x:=x+1; >
        < x:=x-1; >
    coend
end
```

Con el uso de < >, x=0 en el fin del programa. Sin embargo, si no se ejecutaran de forma atómica, x podría valer -1, 0 ó 1.

CONDICIÓN DE SINCRONIZACIÓN

Establece que no todas las posibles interfoliaciones de las secuencias de instrucciones atómicas de los procesos son correctas.

Ejemplo: productor - consumidor.

PROPIEDADES DE LOS SISTEMAS CONCURRENTES

Propiedad de seguridad

- Son condiciones que deben cumplirse en todo momento.
- Son requeridas en especificaciones estáticas del programa
- Son fáciles de demostrar y para cumplirlas se restringen las interfoliaciones.

Ejemplos:

→ Exclusión mutua: nunca se entrelazan ciertas operaciones

→ Ausencia de Interbloqueo: los procesos nunca esperarán algo que nunca sucederá

→ Seguridad en Productor-Consumidor: el consumidor consumirá todos los datos una sola vez en el orden en el que se producen.

Propiedad de vivacidad

. Son propiedades dinámicas, más difíciles de probar.

Ejemplos:

-> Ausencia de inanición: un proceso no puede ser indefinidamente pospuesto, tendrá que avanzar en algún momento.

-> Equidad: un proceso debe progresar con justicia relativa con respecto a los demás. Existen varios grados.

VERIFICACIÓN DE PROGRAMAS CONCURRENTES

Se puede hacer de varias formas:

-> Posibilidad: realizar varias ejecuciones y comprobar que se verifica la propiedad.

- Problema: no se verifican todos los casos. Hay historias que son posibles pero podrían no ocurrir.

-> Enfoque operacional: análisis exhaustivo de casos. Se comprueban todas las historias.

- Problema: muy laborioso, el número de secuencias de interfolación crece exponencialmente.

VERIFICACIÓN
ENFOQUE AXIOMÁTICO

- Se define un sistema lógico formal que establece propiedades de programas en base a axiomas y reglas de inferencia.
- Se usan asertos (fórmulas lógicas) para caracterizar a un conjunto de estados.
- Las sentencias atómicas actúan como transformadores de predicados teoremas de la forma:

$$\{P\}\ S\ \{Q\}$$

"Si la ejecución de S comienza cuando P (precondición) es verdadero, entonces Q (postcondición) será verdadero en el estado final".

- Menor complejidad que en los métodos anteriores.

INVARIANTE GLOBAL

Predicado que debe ser verdadero en el estado inicial de cada proceso y durante la ejecución de todos ellos.

- Ejemplo: en Productor-Consumidor sería:

$$consumidos \leq producidos \leq consumidos + 1$$