Tema 1. Conceptos fundamentales

1. Concepto de programación concurrente

- Avances en HARDWARE (procesadores específicos de E/S)
- Nuevos SO que optimizan uso del procesador (E/S concurrente)
- Nuevos problemas de sincronización
- Programación concurrente como disciplina

> *Los primeros sistemas concurrentes fueron los propios sistemas operativos, un solo procesador atendía multiples usuarios*

- Hitos:
  - Aparición del concepto de *thread* o *hilo de ejecución*, que permite que los programas se ejecuten más rápido que los que utilizan el concepto de proceso
  - Aparición de lenguajes de propósito general como Java que dan soporte directo a la programación concurrente
  - Aparición de Internet: campo abonado para el desarrollo y la utilización de programas concurrentes. Navegadores, chats, etc..., están programados usando técnicas de programación concurrente

- Concurrencia (RAE): Acaecimiento o concurso de varios sucesos en un mismo tiempo
- Si sustituimos **suceso** por **proceso** ya tenemos la definición de concurrencia en computación

- ***Programa ≠ proceso***
- Programa:
  - Conjunto de instrucciones
  - Estático (para que pueda hacer algo hay que ponerlo en ejecución)
  - Secuencia de líneas de código que dicen qué hacer con un conjunto de datos

    - Comparable con una clase en POO

- Proceso:
    - Programa en ejecución (definición incompleta)
    - Dinámico
    - Representado por un valor de contador de programa, el contenido de los registros del procesador, una pila y una sección de datos que contiene variables globales.
    - Se puede comparar con un objeto en POO
    - Puede haber múltiples procesos que corresponden al mismo programa igual que pasa en POO muchos objetos que provienen de una misma clase.

- POO: Una sola clase y muchos objetos instancias de la clase
- PC: Un solo programa y muchos procesos que corresponden al mismo programa
    - Un proceso no tiene por qué ser todo un programa en ejecución sino que puede ser una parte de él. Por ejemplo un programa, al ponerse en ejecución puede dar lugar a más de un proceso, cada uno de ellos ejecutando una parte del programa (navegador *Firefox*, *Chrome*, etc...).

- Concurrencia:
    - Dos procesos serán concurrentes cuando la primera instrucción de uno de ellos se ejecute después de la primera instrucción del otro y antes de la última
        - Existe solapamiento en la ejecución de sus instrucciones, aunque no se ejecuten al mismo tiempo

    - Si los proceso se ejecutan exactamente al mismo tiempo → programación paralela
    - La programación concurrente es un paralelismo potencial, no real. Dependerá del hardware subyacente

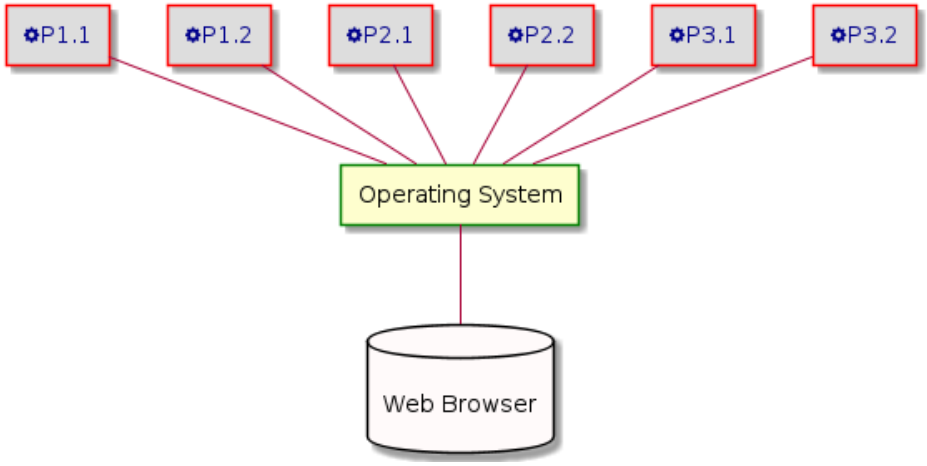
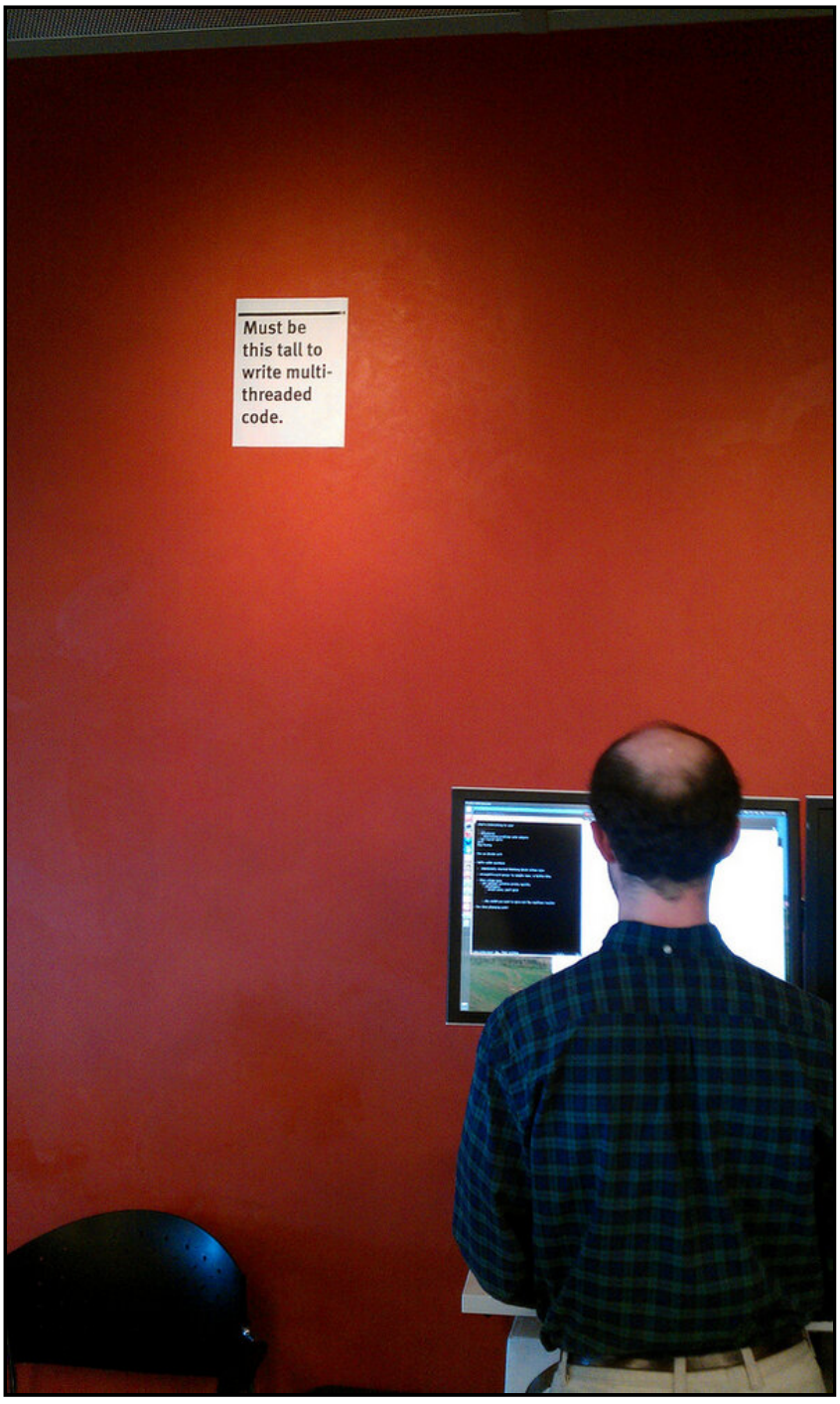


Figure 1: Programación concurrente.

- Cuando varios procesos se ejecutan concurrentemente pueden
  - Colaborar
  - Competir por recursos

- En ambos casos es necesario introducir mecanismos de comunicación y sincronización
- Programación concurrente:
  - Permite especificar la ejecución concurrente de las acciones de un programa, así como las técnicas para resolver problemas inherentes a la ejecución concurrente → mecanismos de comunicación y sincronización
  - Es más compleja que la programación tradicional

- David Baron puso esta nota en su oficina en Mozilla hace un tiempo:



2. Beneficios de la programación concurrente

- Cuando los procesos se ejecutan concurrentemente puede haber procesos que colaboran entre sí mientras que otros pueden competir por los mismo recursos.
- Entonces si añade estos problemas, ¿cuales son los beneficios de la programación concurrente?
  - Mejor aprovechamiento de la CPU
  - Velocidad de ejecución: cuando hay mas de 1 procesador podemos tener cada proceso en un procesador
    - Por ejemplo programas cálculo numérico, tratamiento de imágenes, la propia compilación de código con `make`, etc...

- Solución de problemas inherentemente concurrentes:
  - Sistemas de control
  - Tecnologías web: servidores web, chat, servidores correo, navegadores.
  - Aplicaciones basadas en interfaces de usuario (e.g. juegos)
  - Simulación realista de objetos físicos: la simulación de objetos físicos no se puede hacer de manera secuencial porque no se correspondería con la vida real
  - SGBD: la programación concurrente permite aplicar políticas adecuadas para mantener la integridad de los datos.

3. Concurrencia y arquitecturas hardware

- Cuando hablamos de hardware nos referimos al número de procesadores del sistema:
  - Sistemas monoprocesador: 1 solo procesador
  - Sistemas multiprocesador: más de un procesador

- La programación concurrente presenta beneficios tanto en arquitecturas hardware monoprocesador (comunicación mediante variables compartidas) como en arquitecturas hardware multiprocesador

- Arquitecturas hardware monoprocesador:
  - También es posible tener ejecución concurrente de procesos
  - No todos los procesos se ejecutan al mismo tiempo, sólo uno de ellos los estará haciendo, pero la sensación del usuario es de estar

ejecutándose al mismo tiempo
    - El SO va alternando el tiempo de procesador entre los distintos procesos → *Multiprogramación*
    - Todos los procesos comparten la misma memoria. La forma de sincronizar y comunicar es mediante el uso de variables compartidas.

- Arquitecturas hardware multiprocesador: permiten que exista el paralelismo real, aunque la realidad es que siempre haya más procesos que procesadores.
- Las clasificamos en:
  Fuertemente acopladas :
  Se comunican mediante variables en memoria compartida, ya que los procesadores y otros dispositivos están conectados a un bus: *Multiproceso*

  Débilmente acopladas :
  No existe memoria compartida por los procesadores, cada procesador tiene su memoria local. Se trata de un *Procesamiento distribuido*. El mecanismo de comunicación más común es el paso de mensajes.

- El paso de mensajes también se puede aplicar en sistemas de multiproceso y de multiprogramación.
- La unión del paso de mensajes y la programación concurrente ha llevado a la *programación concurrente orientada a objetos*. El caso más representativo es el de JAVA

3.1. Concurrencia y arquitecturas hardware (cuadro resumen)

Programa Concurrente :
Conjunto de acciones que pueden ser ejecutadas simultáneamente.

Programa Paralelo :
Programa concurrente para sistema multiprocesador

Programa Distribuido :
Programa paralelo para sistemas sin memoria compartida

En la siguiente tabla tenemos en las filas el *número de procesadores* y en la columna correspondiente si se trata de *memoria compartida* o no.

NP. vs. MC.	Sí	No
1	Multiprogramación	-
n	Multiproceso	Programación distribuida

4. Especificación de ejecución concurrente

- ¿Qué se puede ejecutar concurrentemente?
  - No todas las partes de un programa se pueden ejecutar concurrentemente:

No	Sí
x:=x+1;	x:=1
y:=x+2;	y:=2
z:=y+1;	z:=3

  - En la primera columna está claro que la primera sentencia que debe ejecutar antes que la segunda
  - En la segunda columna, el orden en que se ejecuten no tiene importancia

Condiciones de Bernstein :

Para poder determinar si dos conjuntos de instrucciones se pueden ejecutar concurrentemente, sea S_k un conjunto de instrucciones a ejecutar, llamamos:

- $L(S_k) = \{a_1, a_2, \ldots a_n\}$ al *conjunto de lectura*, formado por todas las variables cuyos valores son referenciados (se leen) durante la ejecución de las instrucciones S_k
- $E(S_k) = \{b_1, b_2, \ldots b_m\}$ al *conjunto de escritura*, formado por todas las variables cuyos valores son actualizados durante la ejecución

- Para que se puedan ejecutar concurrentemente dos conjuntos de instrucciones S_i y S_j se debe cumplir que ninguno escriba lo que el otro lee o escribe:
  1. $L(S_i) \cap E(S_j) = \varnothing$
  2. $E(S_i) \cap L(S_j) = \varnothing$
  3. $E(S_i) \cap E(S_j) = \varnothing$

- Dadas las siguientes instrucciones, decidir cuáles se pueden ejecutar de forma concurrente.

```
S1 a := x + y;
S2 b := z - 1;
S3 c := a - b;
S4 w := c + 1;
```

- Procedemos así:
  1. Calculamos los conjuntos de lectura y escritura
  2. Aplicamos las condiciones a cada par de sentencias:

	S1	**S2**	**S3**	**S4**
S1	-	S	N	S
S2	-	-	N	S
S3	-	-	-	N
S4	-	-	-	-

4.1. Ejercicio

- Usando las condiciones de Bernstein, construye una tabla como la anterior para el siguiente código:

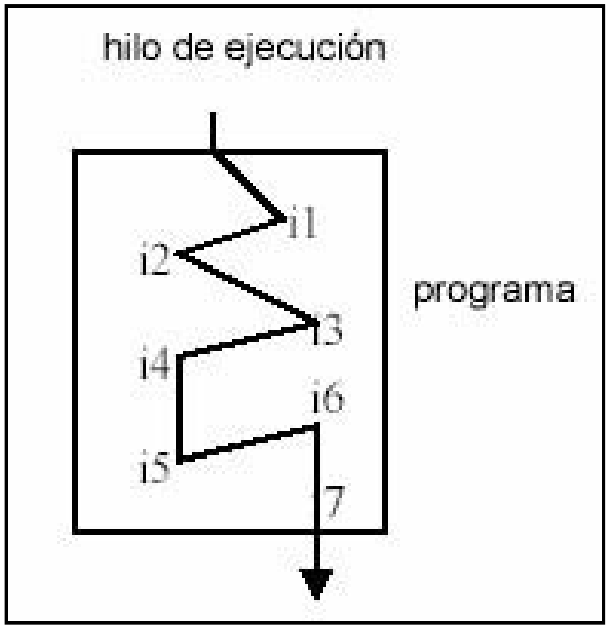
```
S1 cuad := x * x;
S2 m1 := a * cuad;
S3 m2 := b * x;
S4 z := m1 + m2;
S5 y := z + c;
```

5. Especificación de ejecución concurrente

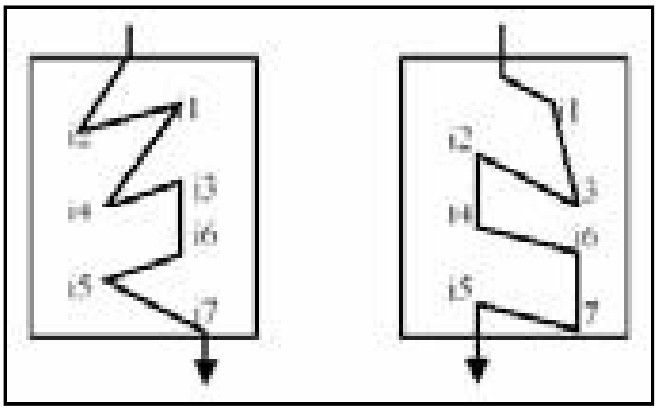
- La ejecución concurrente tiene dos características:
  - Orden parcial
  - Indeterminismo (causado por el orden parcial)
  - **El indeterminismo está en la raíz de los problemas de la programación concurrente**

6. Orden de ejecución de las instrucciones

- Programas secuenciales: orden total, ante un conjunto de datos de entrada siempre se sabe el flujo del programa:



- Programas concurrentes: orden parcial, ante el mismo conjunto de datos no se puede saber cual será el flujo de ejecución. En cada ejecución el flujo puede ir por un sitio distinto:



7. Indeterminismo

- El orden parcial lleva a los programas a un comportamiento indeterminista → arrojar diferentes resultados cuando se ejecuta el programa repetidamente con los mismos datos de entrada

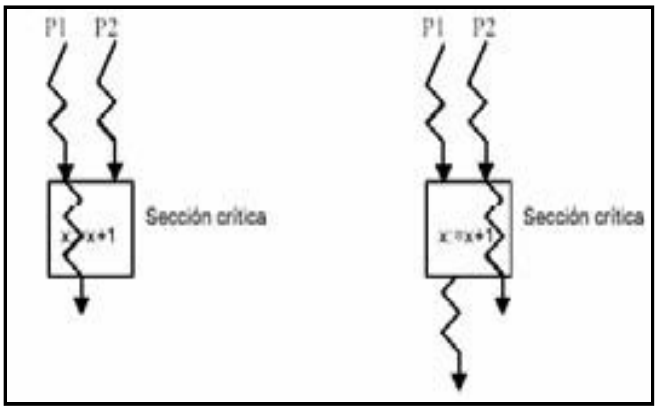
8. Problemas inherentes a la programación concurrente

8.1. Exclusión mutua:

- Situación en la que un recurso sólo puede ser accedido por un proceso (recurso crítico)
  - Ojo! Lo que realmente se ejecuta concurrentemente son las líneas de código generadas por el compilador. Cada línea es atómica o indivisible, p.e. la instrucción "`x := x + 1`", genera el ensamblador:

```
LOAD X R1;
ADD R1 1;
STORE R1 X
```

  - Este programa no podría lanzar dos procesos concurrentes salvo que la línea de código se defina como **Sección Crítica**



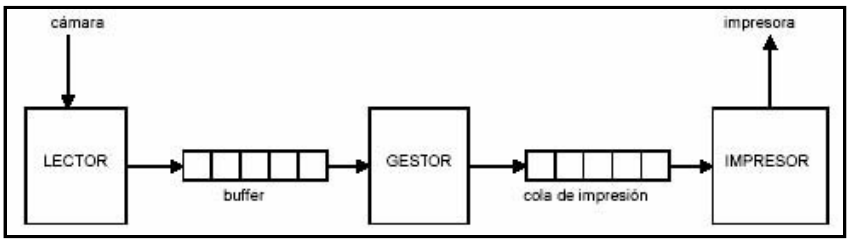
8.2. Condición de sincronización:

- Situaciones en las que un recurso compartido por varios procesos se encuentra en un estado en el que un proceso no puede hacer una determinada acción con él hasta que no cambie su estado
- La programación concurrente debe permitirnos bloquear procesos a la espera de un evento y desbloquearlos cuando el evento haya ocurrido

- Supongamos el siguiente sistema:
  Lector:
  va almacenando en el buffer las imágenes capturadas por una camara

  Gestor:
  recoge las imágenes del buffer, las trata y las deja en la cola de impresión

  Impresor:
  va imprimiendo las imágenes de la cola de impresion



- Supongamos que las tres funciones se ejecutan de manera concurrente:

```
1: def lector():                    def impresor():
2:         while True:                      while True:
3:                 captura imagen                   coge imagen de cola de
4:                 almacena en buffer               imprime imagen
5:
6: def gestor():
7:         while True:
8:                 coge imagen del buffer
9:                 trata imagen
10:                almacena imagen en cola de impresion
```

- Esta solución está incompleta:
  - ¿Qué ocurre cuando el proceso lector o el proceso gestor tratan de poner una imagen y el buffer o la cola están llenos?
  - ¿Qué ocurre cuando el proceso gestor o el proceso impresor tratan de coger una imagen y el buffer o la cola están vacios?

- Es por tanto necesario establecer *condiciones de sincronización*

9. Corrección de programas concurrentes

- Además de las especificaciones funcionales, un programa concurrente debe cumplir…
- Propiedades de seguridad:
  - Son aquéllas que aseguran que nada malo va a pasar durante la ejecución del programa

- Propiedades de viveza:
  - Son aquellas que aseguran que algo bueno pasará eventualmente durante la ejecución del programa

9.1. Ejemplo de las propiedades:

- En el *juego del pañuelo* hay dos equipos `A` y `B`, y un juez con un pañuelo. Cada jugador de un equipo tiene un número del `1` al `3`.
- No puede haber dos jugadores en el mismo equipo con el mismo número. El juez dice un número y entonces los dos rivales con el mismo número salen corriendo a coger el *pañuelo*.
- El jugador que lo coja ha de volver corriendo a su sitio sin que su rival logre tocarle la espalda. En este escenario plantearemos las propiedades de seguridad y viveza

10. Corrección de programas concurrentes

- Propiedades de seguridad:
  - **Exclusión mutua**
  - **Condición de sincronización**
  - **Interbloqueo:** se produce cuando todos los procesos están esperando que ocurra un evento que nunca se producirá. Se le denomina también interbloqueo pasivo. (*deadlock* o abrazo mortal)

10.1. En relación con el juego anterior

Exclusion mutua:

Hay recursos que deben ser accedidos en exclusión mutua y si otros procesos quieren acceder a este recurso tienen que esperar a que sea liberado.

Ejemplo: el pañuelo ha de adquirirse por un jugador o por otro, en exclusion mutua, ya que si lo adquieren los dos puede llegar a romperse → *mal funcionamiento del sistema*.

Condición de sincronización:

Hay situaciones en las que un proceso debe esperar a que se produzca un determinado evento para poder avanzar y no debería de avanzar antes de que ese evento ocurra.

Ejemplo: el jugador debe esperar a que digan su número para poder salir en busca del pañuelo.

Interbloqueo:

En nuestro juego se daría si uno de los jugadores coge el pañuelo y se lo lleva a su casa. El juez debería esperar a que le devuelva el pañuelo y los jugadores a que el juez diga su número, pero esto nunca va a pasar.

11. Corrección de programas concurrentes

- Propiedades de vivacidad:
  Interbloqueo activo:
  Se produce cuando un sistema ejecuta una serie de instrucciones sin hacer ningún progreso (*livelock*)

  Inanición:
  Existen un grupo de procesos que nunca progresan pues no se les otorga tiempo de procesador para avanzar

11.1. En relación con el juego anterior

Interbloqueo activo:
En el ejemplo si los jugadores que salen a por el pañuelo hacen como que lo cogen pero nunca llegan a cogerlo.

Inanición:
En nuestro ejemplo se produciría si el juez nunca dice el número de un jugador en concreto. Hay que garantizar cierta equidad en el trato a los procesos a no ser que las especificaciónes del sistema digan lo contrario.

12. Corrección de programas concurrentes

- Ejemplo: supongamos que una variable se actualiza con dos funciones que se ejecutan concurrentemente, ¿qué valor tendría x al acabar el programa?

```
1: x = 0
2:
3: def incrementa_5_A():   def incrementa_5_B():
4:                 i = 0                   i = 0
5:                 while i < 5:            while i < 5:
6:                         i = i + 1               i = i + 1
7:                         x = x + 1               x = x + 1
```

Ejercicio:

plantea una traza que haría que `x` valiese `5` al final de la ejecución. ¿Qué rango de valores crees que podría acabar teniendo `x` ?

Pregunta:

¿De qué problemas de los nombrados en esta sesión adolece el programa?

13. Cuestiones breves

1. ¿Cuál es la ventaja de la concurrencia en los sistemas monoprocesador?
2. ¿Cuáles son las diferencias entre programación concurrente, paralela y distribuida?
3. ¿Cuáles son los dos problemas principales inherentes a la programación concurrente?
4. ¿Qué es una sección crítica?
5. ¿Cuáles son las características de un programa concurrente?
6. ¿Qué se entiende por un programa concurrente correcto?

13.1. Aclaraciones

- **En ningún caso estas transparencias son la bibliografía de la asignatura**, por lo tanto debes estudiar, aclarar y ampliar los conceptos que en ellas encuentres empleando los enlaces web y bibliografía recomendada que puedes consultar en la página web de la ficha de la asignatura y en la web propia de la asignatura.