

Tema I: Introducción

PROGRAMACIÓN CONCURRENTE

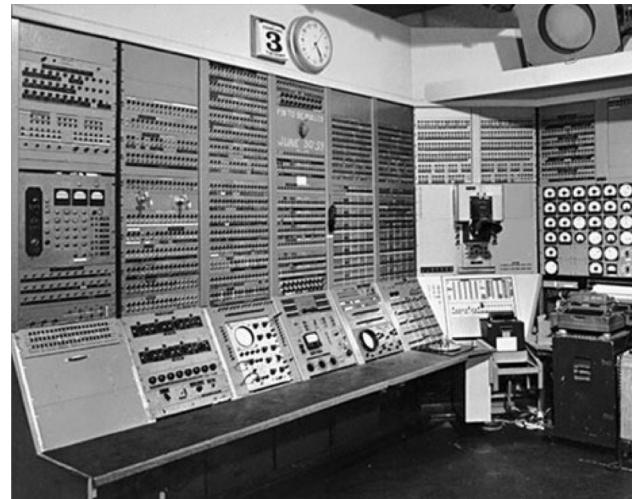


Universidad
Rey Juan Carlos

Introducción

Historia de la concurrencia

- Los primeros ordenadores **no** tenían sistema operativo.
- Ejecutaban un programa desde el **inicio** hasta el **final**.
- Este programa tenía acceso a **todos los recursos** del sistema



Introducción

Historia de la concurrencia

- Los sistemas operativos permiten ejecutar varios programas simultáneamente en **procesos**.
- Estos procesos pueden **comunicarse** entre ellos a través de diferentes mecanismos:
 - Sockets, señales, memoria compartida, semáforos, ficheros, etc.

Introducción

¿Por qué surge la concurrencia?

Utilización de recursos

- A veces un programa necesita esperar a operaciones externas (E/S) para poder continuar.
- Podemos aprovechar ese tiempo de espera para hacer otras tareas

Introducción

¿Por qué surge la concurrencia?

Justicia

- Varios usuarios y programas quieren acceder al mismo recurso.
- Es mejor compartir el recurso que tener que esperar a que todos terminen.

Introducción

¿Por qué surge la concurrencia?

Conveniencia

- Suele ser mejor disponer de varios programas que se encargan de tareas individuales y tienen que coordinarse entre sí, que tener un único programa que hace todas las tareas.

Introducción

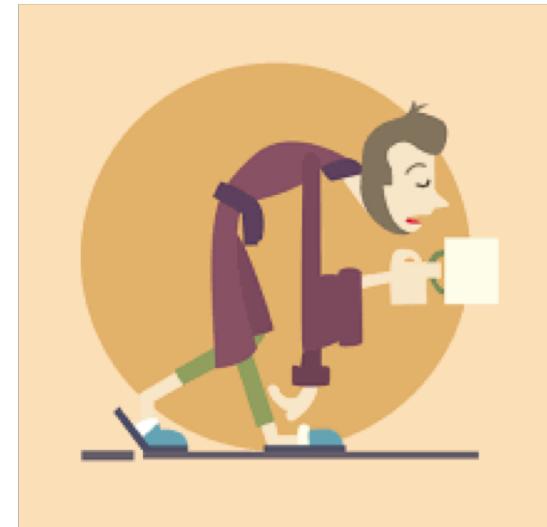
Modelo secuencial

- Inicialmente, cada proceso tenía su propio espacio de memoria para **instrucciones y datos**.
- Ejecutaban las instrucciones de forma **secuencial**, interactuando con el exterior a través de E/S.
- Es un modelo muy sencillo donde está **perfectamente definido** qué instrucción se ejecutará a continuación.

Introducción

Modelo secuencial

- Este modelo funciona como las personas.
- Por la mañana, nosotros:
 1. Nos levantamos
 2. Nos ponemos las zapatillas
 3. Vamos a la cocina
 4. Encendemos la cafetera



Introducción

Modelo secuencial

- Cada una de estas acciones es una abstracción de unos pasos más precisos.
- Por ejemplo, el paso de hacer café se puede dividir en:
 1. Abrir el armario
 2. Coger el café
 3. Abrir la cafetera
 4. Poner el café
 5. Encender la cafetera



Introducción

Modelo secuencial

- El último paso (*encender la cafetera*) implica cierta **asincronía**.
- Mientras se hace el café, podemos:
 - Esperar.
 - Hacer otra tarea: hacer unas tostadas, ver las noticias, etc.



Introducción

Modelo secuencial

- Los fabricantes de cafeteras y tostadoras conocen esta asincronía, y por eso sus productos emiten una **señal** (visual o auditiva) cuando han terminado.
- La gente eficiente debe buscar un **balance** entre secuencialidad y asincronía, igual que los programas.

Introducción

Threads

- La misma idea motivó el desarrollo de los **threads**.
- Permiten que diferentes flujos de control de un programa **coexistan** dentro de un mismo proceso.
- **Comparten** recursos como la memoria, pero cada uno tiene su propio contador de programa, pila y variables locales.

Introducción

Threads

- A veces se denominan **procesos ligeros**, y la mayoría de SO los consideran las unidades básicas de planificación.
- Los threads se ejecutan de manera **simultánea y asíncrona** respecto entre ellos.

Introducción

Threads

- Comparten el espacio de memoria de su proceso, por lo que todos tienen acceso a las **mismas variables del montículo**, lo que permite **compartir** datos de forma más eficiente.
- **Problema:** varios threads pueden modificar la misma variable **a la vez**, produciendo resultados inesperados.

Concurrencia

Paradigma

- **Conjunto de métodos y herramientas para construir y verificar programas concurrentes.**
- **Surge en los años 60**, cuando los SO permiten ejecutar varios procesos de forma concurrente.

Concurrencia

Evolución

- Inicialmente la concurrencia sólo podía utilizarse con herramientas de **bajo nivel** y por el sistema operativo
- Los lenguajes de programación y sus librerías han **evolucionado** para proporcionar herramientas de **alto nivel** para la programación concurrente

Concurrencia

Lenguajes

- Lenguajes de programación que permiten el desarrollo de programas concurrentes
- PascalFC, C/C++, Scala, OCAM, Java, Haskell, Erlang, C#...

¿Qué es la concurrencia?

Programa secuencial

- Conjunto de declaraciones de datos e **instrucciones ejecutables**, escrito en un lenguaje de programación.
- Estas instrucciones deben ejecutarse una a continuación de otra, siguiendo una **secuencia** determinada por un algoritmo, para resolver un cierto problema.

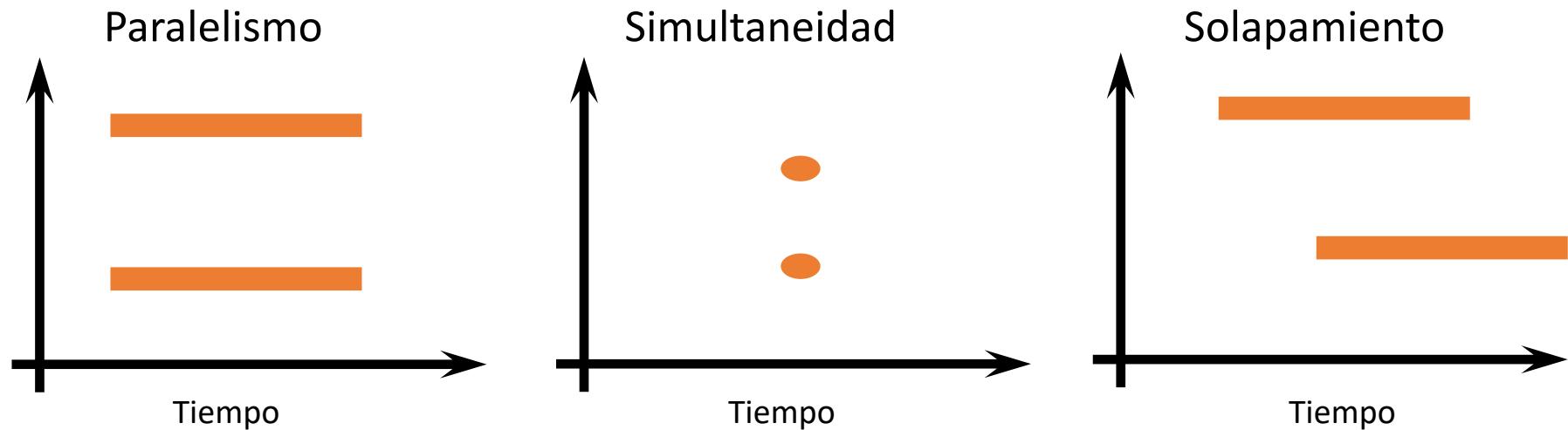
¿Qué es la concurrencia?

Proceso

- Es la **ejecución** de un programa secuencial en un sistema informático.
- Pueden existir **varios procesos** de un mismo **programa secuencial** porque se puede ejecutar el mismo programa secuencial varias veces, con cierto **solapamiento** y de forma **simultánea**.

¿Qué es la concurrencia?

Comparativa



Gestión de hilos en Java

Creación de hilos

- Un hilo es un objeto de la clase `java.lang.Thread`
- El código que se ejecutará en el nuevo hilo se indica en el constructor de una clase que implementa la interfaz `java.lang.Runnable`
- Podemos asignar un nombre a cada hilo (para depurar)

Gestión de hilos en Java

Creación de hilos

- Toda aplicación Java tiene al menos un hilo *main* que ejecuta el método estático **main**
- Existen **otros hilos** gestionados por la JVM (como el recolector de basura)
- Pueden existir otros hilos si se usan librerías o frameworks (GUI, Servidores web, ...)
- Un programa **finaliza** su ejecución cuando
 - Todos sus hilos han **finalizado** su ejecución o
 - Se ejecuta el método **System.exit(...)**

Gestión de hilos en Java

Ciclo de vida

- Los hilos **comienzan** su ejecución cuando se invoca el método `start()` de la clase `Thread`
- Un hilo **finaliza** su ejecución
 - Cuando se han ejecutado todas sus sentencias o
 - Cuando se eleva una excepción no chequeada (`RuntimeException`) en el método `run()`

Gestión de hilos en Java

Creación de hilos

- El lenguaje Java dispone de una sintaxis más **compacta** para la creación de hilos
- Se pueden usar las **clases anónimas**
- Estas clases se **declaran** en el método de otra clase
- En la misma sentencia se declara la clase y se **crea** un objeto de dicha clase
- Una clase anónima no tiene **nombre**

Gestión de hilos en Java

Creación de hilos

- Además, la variable intermedia que almacena Runnable no es necesaria, podemos eliminarla

Gestión de hilos en Java

Creación de hilos

- Las clases anónimas pueden acceder a variables y parámetros **final** del método en el que se encuentran.
- Una variable final es aquella que **no puede cambiar de valor** una vez inicializada.
- Desde Java 8, se puede acceder a **cualquier variable** que no cambie de valor, aunque no esté declarada como final (*effectively final*).

Gestión de hilos en Java

Creación de hilos

- Si la clase anónima está en un método **estático**, podrá acceder a los atributos y métodos estáticos de la clase **contenedora**
- Si la clase anónima está en un método **no estático** (método de instancia), entonces podrá acceder:
 - a los atributos y métodos no estáticos de la clase **contenedora**
 - al **objeto** de la clase **contenedora**

Gestión de hilos en Java

Creación de hilos

- Java 8 permite la creación de hilos de una manera más resumida, mediante el uso de expresiones lambda.
- Una interfaz funcional (con un solo método) se puede implementar de la siguiente manera:
`(argumentos) -> {cuerpo};`
- Runnable es una interfaz funcional

Gestión de hilos en Java

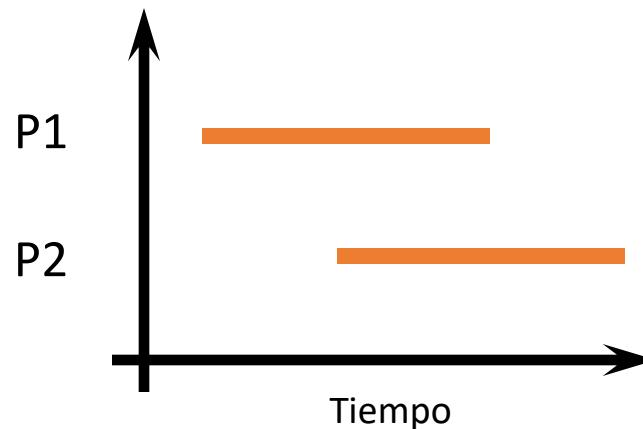
Creación de hilos

- Podemos crear un thread creando una clase hija de Thread y redefiniendo el método run().
- Puede ser útil, pero en general se prefiere la otra alternativa.

Programa y Sistema Concurrente

Proceso concurrente

- Se dice que P1 y P2 son dos procesos concurrentes si la primera instrucción de uno de ellos se ejecuta entre la primera y la última instrucción del otro



Programa y Sistema Concurrente

Programa concurrente

- Conjunto de varios programas secuenciales, cuyos procesos pueden ejecutarse concurrentemente en un sistema informático.
- Por ejemplo:
 - Windows y Linux son concurrentes, MS-DOS no.
 - Un programa en Pascal no es concurrente, en Java puede serlo.

Relaciones entre procesos

Competencia y cooperación

- **Independencia:** Sin relación entre procesos
 - No existe ninguna relación entre los procesos
- **Interacción:** Con relación entre procesos
 - **Competencia:** Varios procesos deben **compartir recursos** comunes del sistema (procesador, memoria, disco, impresoras,...) por lo que compiten entre ellos para conseguirlo
 - **Cooperación:** Varios procesos deben trabajar sobre distintas partes de un problema para **resolverlo conjuntamente**

Relaciones entre procesos

Competencia y cooperación

- **Competencia (Cámara web)**
 - Una webcam es un recurso de uso exclusivo por un único programa
 - El primer programa que use la cámara la controla hasta que decida dejar de usarla
- **Cooperación (Antivirus)**
 - Un antivirus en modo residente monitoriza el disco duro para detectar nuevos ficheros
 - Cuando otro programa crea un fichero, el antivirus lo analiza en busca de virus e informa al usuario en caso de encontrarlo
 - Los procesos colaboran entre sí

Relaciones entre procesos

Sincronización y Comunicación

- La competencia y la cooperación son **relaciones de interacción** entre procesos
- Se llevan a cabo a través de las siguientes actividades
 - Comunicación
 - Sincronización

Relaciones entre procesos

Sincronización y Comunicación

- **Sincronización**

- La sincronización impone restricciones a la ejecución de las sentencias de los procesos
- Dos tipos de sincronización:
 - Sincronización Condicional
 - Exclusión Mutua

Relaciones entre procesos

Sincronización y Comunicación

- **Sincronización Condicional**

- Uno o más procesos deben esperar a que se cumpla cierta **condición antes de continuar** con su ejecución
- **Otro proceso** es el que establece esa **condición**
- Por ejemplo, un servidor web está bloqueado hasta que le hagan una petición

Relaciones entre procesos

Sincronización y Comunicación

- **Exclusión Mutua**

- Varios procesos compiten por un recurso común de acceso exclusivo
- Sólo uno de los procesos puede estar accediendo al recurso a la vez y los demás tienen que esperar
- Por ejemplo, solo un proceso puede utilizar a la vez un lector de huellas digitales

Relaciones entre procesos

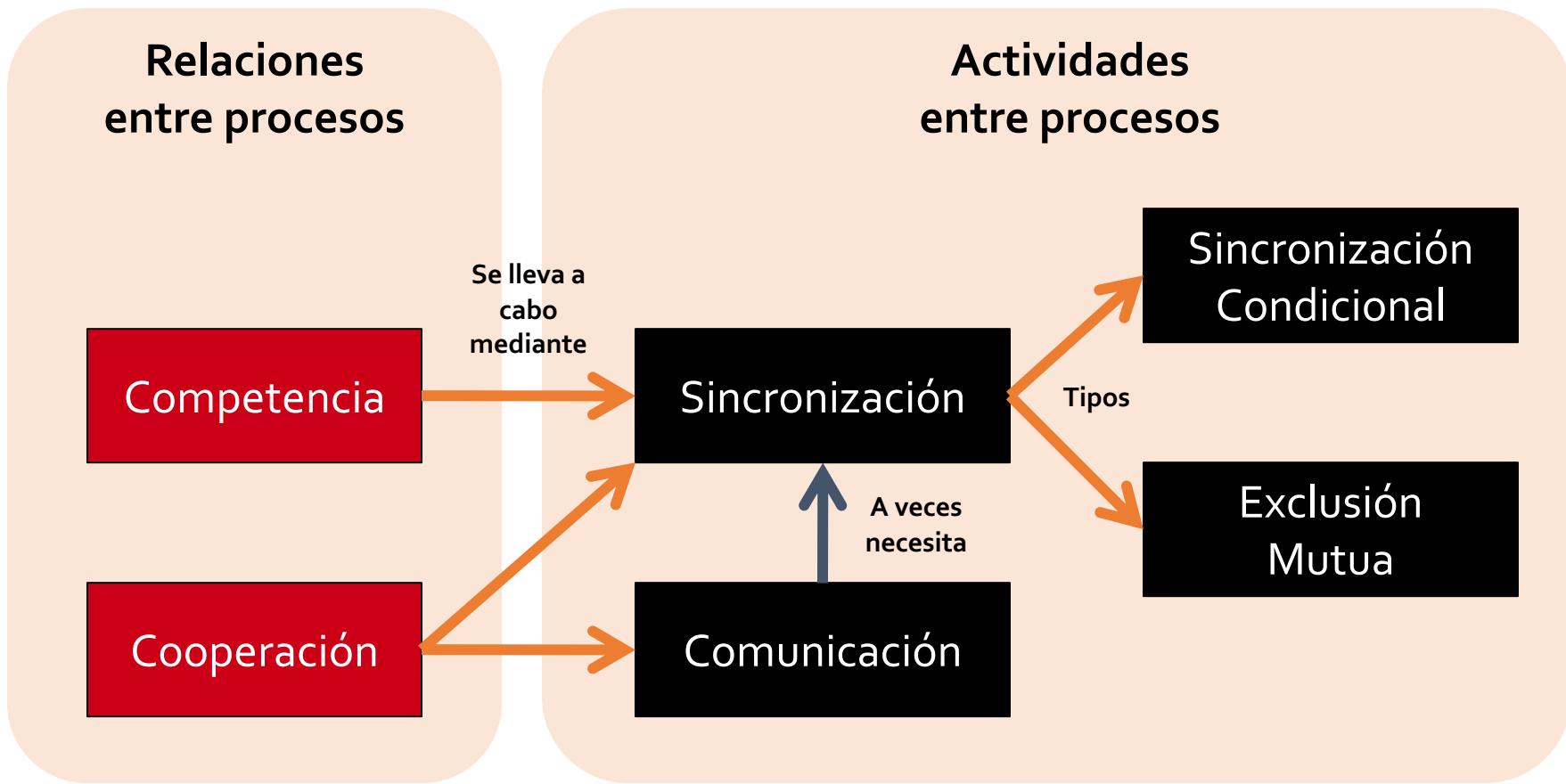
Sincronización y Comunicación

- **Comunicación**

- Es el intercambio de información entre procesos
- Habitualmente cuando dos procesos cooperan entre sí intercambian información de algún modo
- Por ejemplo, para imprimir, necesitamos enviar la información a la cola de impresión

Relaciones entre procesos

Sincronización y Comunicación



Relaciones entre procesos

Modelos de concurrencia

- **Memoria compartida**

- Los procesos pueden acceder a una memoria común
- Existen variables compartidas que varios procesos pueden leer y escribir

- **Paso de mensajes**

- Los procesos se intercambian mensajes entre sí
- Un proceso envía mensaje y otro proceso lo recibe

Relaciones entre procesos

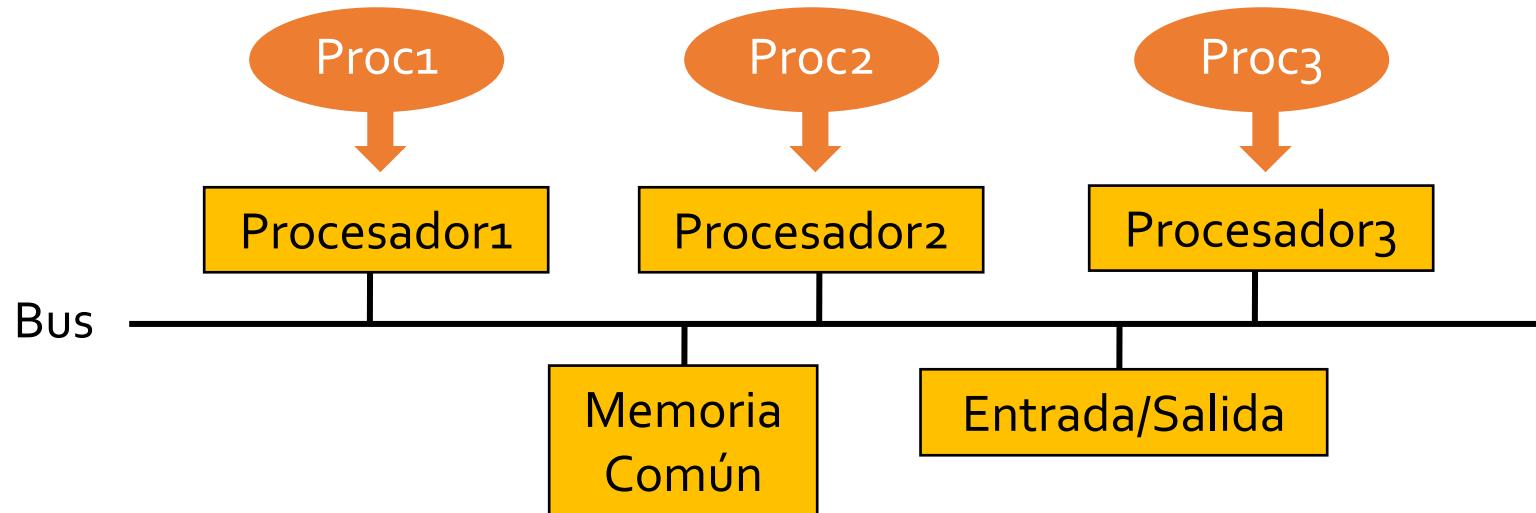
Asignación de procesos a procesadores

- Un procesador sólo puede ejecutar **un proceso a la vez**
- ¿Qué ocurre si hay que ejecutar **más procesos** que los procesadores disponibles? (Que es lo más habitual)

Relaciones entre procesos

Asignación de procesos a procesadores

- **Multiproceso**
 - Cada proceso se ejecuta en su **propio procesador** en un sistema de memoria compartida

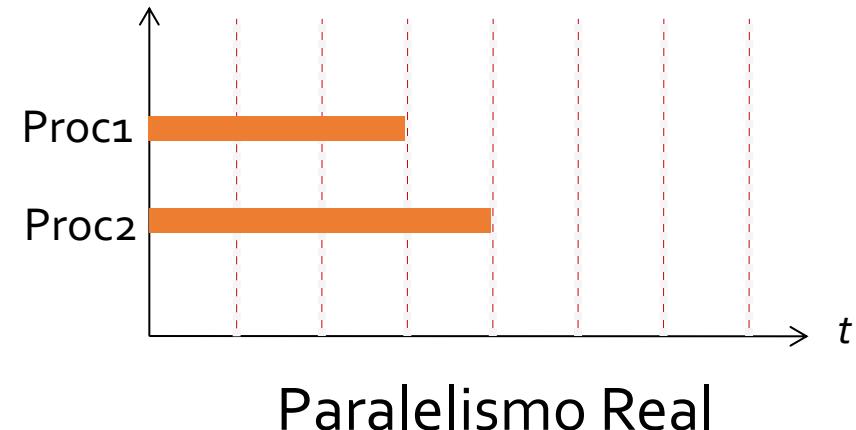
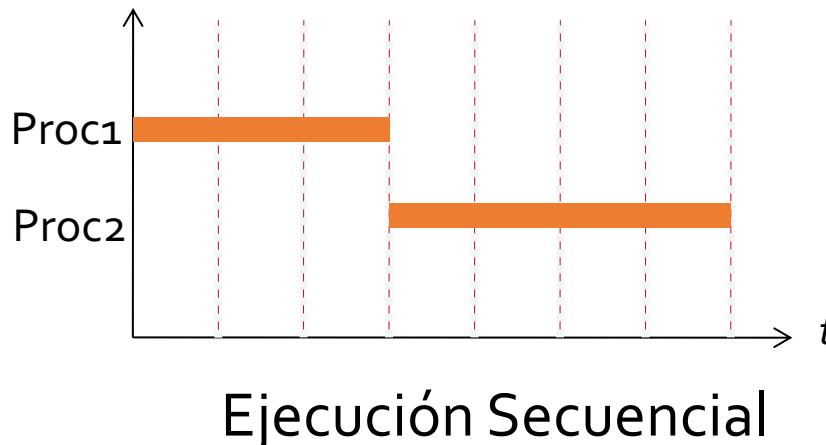


Relaciones entre procesos

Asignación de procesos a procesadores

- **Paralelismo Real**

- Se obtiene cuando hay un procesador por cada proceso
- Se consigue un aumento de la velocidad de ejecución del programa con respecto a la ejecución secuencial

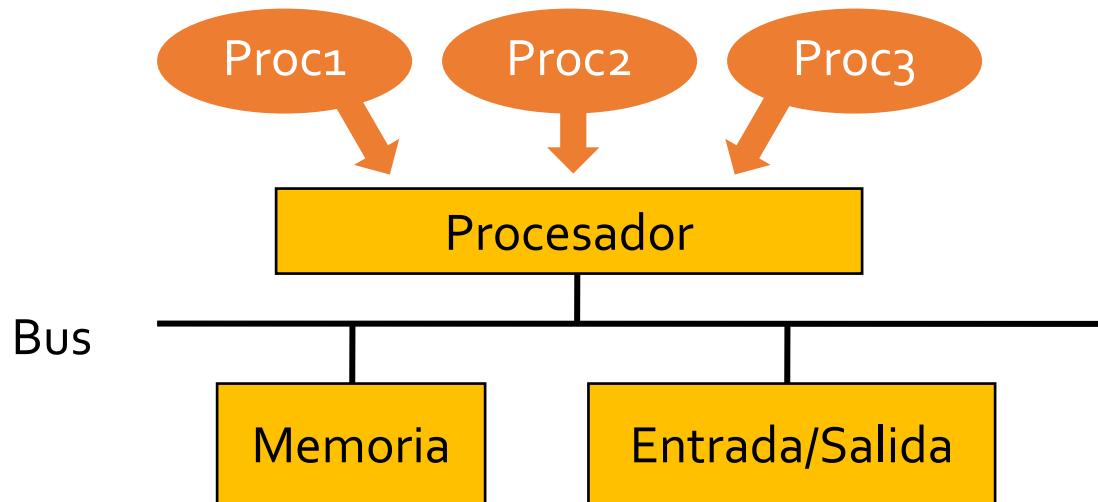


Relaciones entre procesos

Asignación de procesos a procesadores

- **Multiprogramación**

- Varios procesos se ejecutan en el **mismo** procesador
- Cada proceso se ejecuta durante un tiempo y luego pasa a ejecutarse el siguiente proceso (**Compartición de tiempo**)

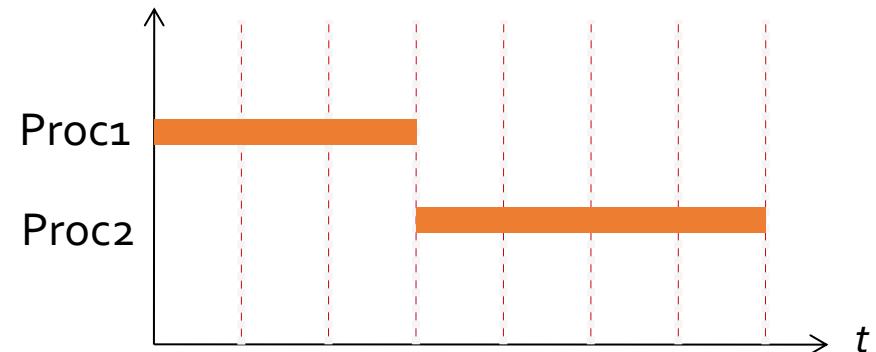


Relaciones entre procesos

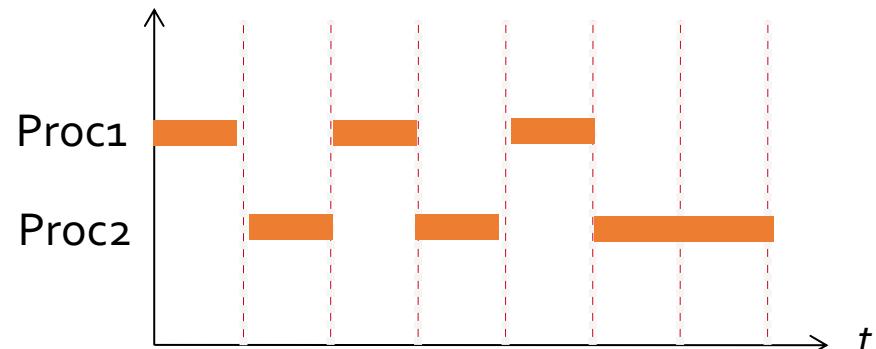
Asignación de procesos a procesadores

- **Paralelismo Simulado (Pseudoparalelismo)**

- Se obtiene cuando varios procesos comparten el mismo procesador
- El usuario percibe una sensación de paralelismo real
- No se consigue un aumento de la velocidad de ejecución del programa con respecto a la ejecución secuencial



Ejecución Secuencial



Paralelismo Simulado

Relaciones entre procesos

Asignación de procesos a procesadores

- Ejemplos de programación concurrente.
Problemas de sincronización (condición de carrera).
- Programa concurrente que incremente (hilo 1) y decremente (hilo 2) una variable
- Programa concurrente que escriba 5 veces XXX (hilo 1), otras 5 --- (hilo 2) y una vez *** (hilo 3)

Gestión de hilos en Java

Bloqueo durmiendo (*sleep*)

- La forma más **sencilla** de **bloqueo** se tiene cuando un hilo **suspende** su ejecución durante un intervalo de tiempo
- Para ello, un hilo puede invocar el método estático `sleep(long millis)` de la clase **Thread**
- El método que lo ejecuta quedará **bloqueado** durante el **tiempo** indicado
- Como Java no es sistema de tiempo real, no garantiza que se cumpla de forma **exacta** el tiempo indicado

Gestión de hilos en Java

Bloqueo durmiendo (*sleep*)

- El método `sleep(...)` eleva un excepción de tipo `InterruptedException`
- Veremos para qué sirve esa excepción. De momento, la ignoramos, elevándola de nuevo.

Gestión de hilos en Java

Asignación de procesos a procesadores

Cada procesador ejecuta un proceso

- Multiproceso
- Procesamiento Distribuido

Paralelismo Real

Aumenta la velocidad de ejecución

Cada procesador ejecuta varios procesos

- Multiprogramación

Paralelismo Simulado

No aumenta la velocidad de ejecución

- En un sistema informático lo más habitual es que se use la multiprogramación aunque se disponga de varios procesadores (más procesos que procesadores).

Gestión de hilos en Java

Asignación de procesos a procesadores

- **Multiproceso**

- Aumenta la velocidad de ejecución
- Se aprovechan los recursos disponibles

- **Multiprogramación**

- No aumenta la velocidad de ejecución
- Incluso puede disminuir debido a que el reparto del procesador implica un coste adicional

¿Para qué sirve la multiprogramación?

Gestión de hilos en Java

Ventajas de la multiprogramación

- Dar un servicio interactivo a varias tareas/usuarios simultáneamente

- Servidor Web

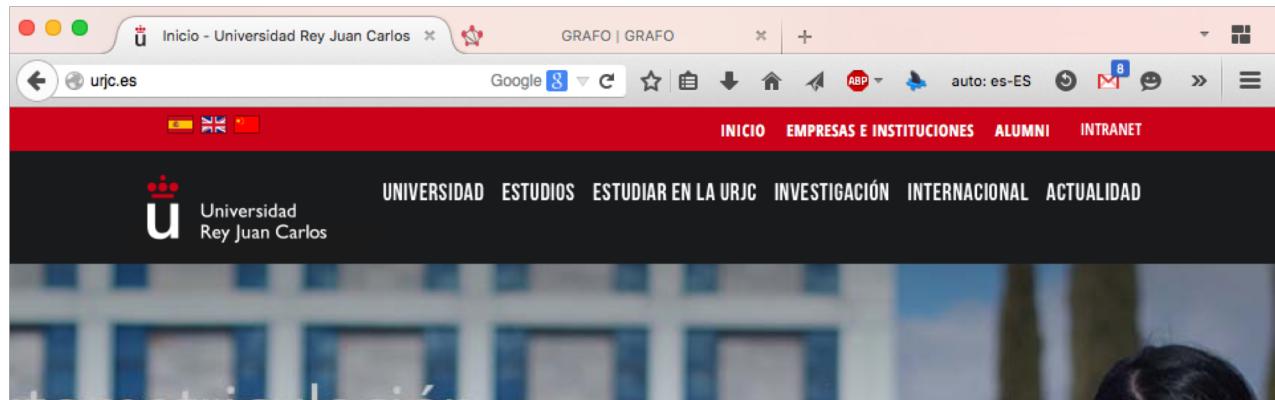
- Atiende a varios cientos de usuarios a la vez

- Hangouts, Skype, WhatsApp...

- Permite conversaciones simultáneas

- Pestañas Firefox o Microsoft Edge

- Permite descargar simultáneamente varias páginas (cada una en una pestaña) mientras navegamos



Gestión de hilos en Java

Ventajas de la multiprogramación

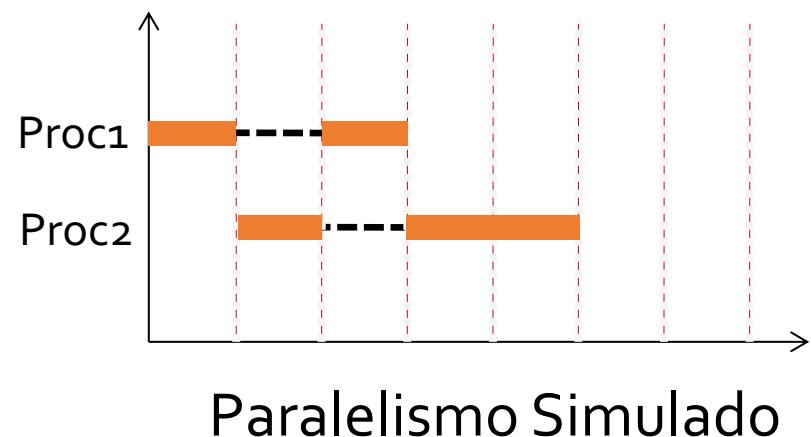
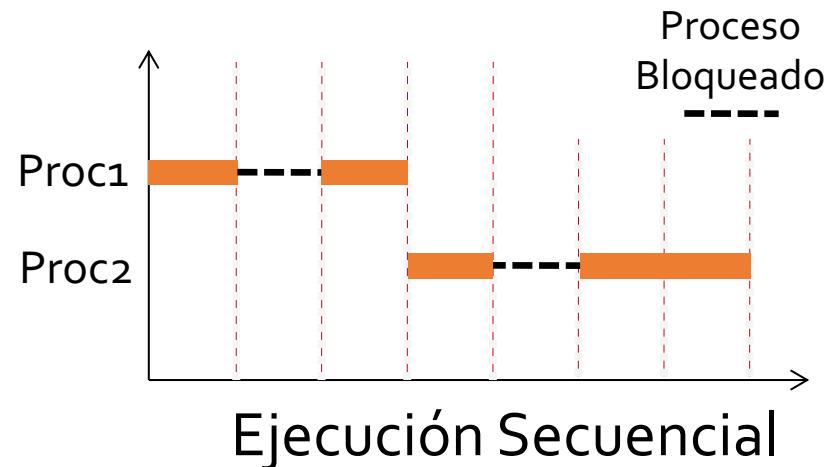
- Ciertos tipos de aplicaciones se implementan de manera natural con programación concurrente
- Aplicaciones gráficas
 - El programa realiza un proceso en segundo plano y también permite seguir trabajando
 - Eclipse compila en segundo plano
- Intercambio entre pares P2P (BitTorrent)
 - Se realizan descargas simultáneas de múltiples usuarios para completar el mismo fichero



Gestión de hilos en Java

Ventajas de la multiprogramación

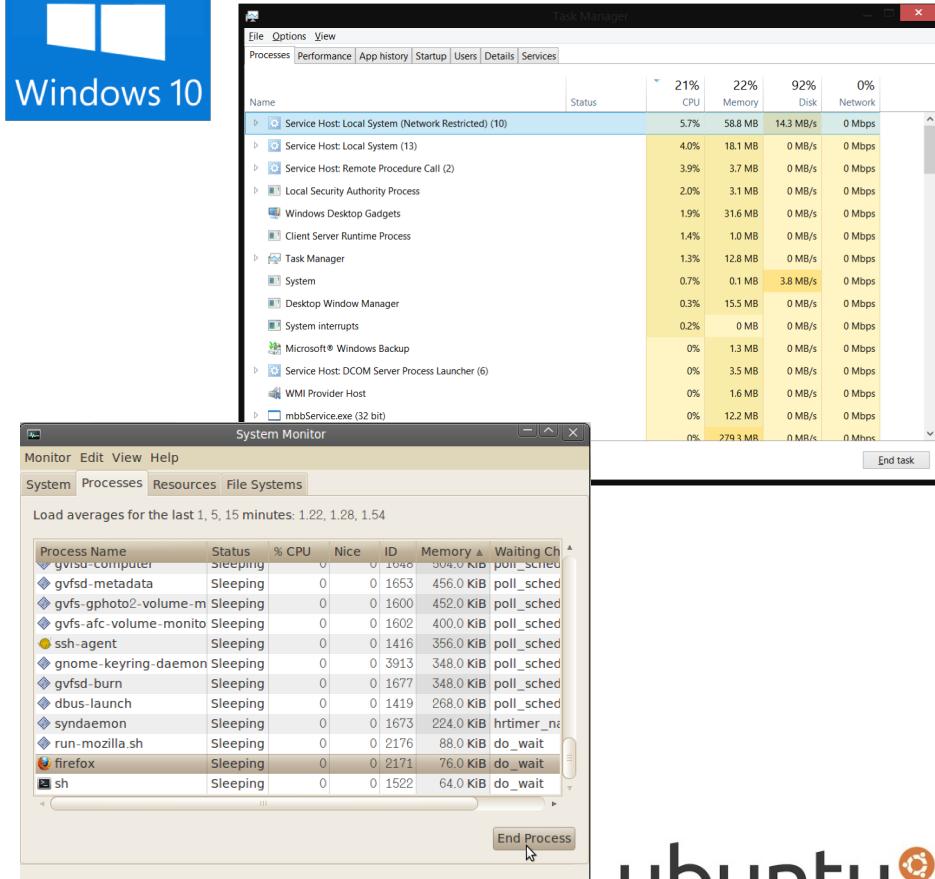
- Con Multiprogramación se puede aprovechar el procesador mientras un proceso espera por entrada/salida (acceso a datos del disco)
- En este caso se reduce el tiempo de ejecución del conjunto de procesos



Gestión de hilos en Java

Ventajas de la multiprogramación

- Los SO la usa para ejecutar varios procesos simultáneamente.
- Se pueden tener varios programas abiertos.



ubuntu

Gestión de hilos en Java

Ventajas de la multiprogramación

- Aplicaciones para varios usuarios
- Aplicaciones que realizan varias tareas a la vez de forma natural
- Sistemas operativos con muchas aplicaciones
- Aprovechamiento del procesador cuando los procesos necesitan esperar
- Tiempo total de ejecución menor

Gestión de hilos en Java

Abstracciones

- Hay **muchas arquitecturas** diferentes en las que se puede ejecutar un programa **concurrente**
- Al diseñar e implementar programas concurrentes no se tiene en cuenta si existe **multiproceso** o **multiprogramación**
- Usamos simplificaciones o **abstracciones** que nos ayudan a centrar nuestra atención en los **procesos y sus relaciones**, y nos evitan pensar en la arquitectura del sistema
- De esta forma los programas serán **portables** y se podrán ejecutar en cualquier sistema / arquitectura

Gestión de hilos en Java

Abstracciones

I^a Abstracción de la Programación Concurrente

Se considera que cada proceso se ejecuta en su propio procesador

- Esta abstracción permite tener en cuenta únicamente las **interacciones entre los procesos** derivadas de sus relaciones de competencia y cooperación
- No nos tenemos que preocupar de si hay **parallelismo real o paralelismo simulado**

Gestión de hilos en Java

Abstracciones

2^a Abstracción de la Programación Concurrente

Se ignoran las velocidades relativas de cada proceso, lo que posibilita considerar sólo las secuencias de instrucciones que se ejecutan

- Tenemos que pensar lo que ocurriría con nuestro programa si el procesador de cada **proceso** tuviese una **velocidad igual**
- También tenemos que pensar que pasaría si un procesador fuese **muy lento** y otro procesador fuese **muy rápido**
- Pensar **en todas las posibles situaciones** permite que nuestros programas concurrentes funcionen correctamente **en cualquier tipo de arquitectura**

¿Cómo se usa la concurrencia?

El orden de las instrucciones

- En un programa **secuencial**, todas las instrucciones están ordenadas:
 - Está claro el orden en el que se van ejecutando las instrucciones y la forma en la que van cambiando los valores de las variables
- En la programación **concurrente**, diferentes ejecuciones del mismo programa pueden ejecutar las sentencias en orden diferente

¿Cómo se usa la concurrencia?

El orden de las instrucciones

```
public class MaxMin {  
  
    static volatile double n1, n2, min, max;  
  
    public static void min() {  
        min = n1 < n2 ? n1 : n2;  
    }  
  
    public static void max() {  
        max = n1 > n2 ? n1 : n2;  
    }  
  
    public static void main(String[] args) {  
        n1 = 3;  
        n2 = 5;  
        min(); // I1  
        max(); // I2  
        println("MIN = "+min+", MAX = "+max); //I3  
    }  
}
```

- Programación secuencial:
- Conocemos el orden total

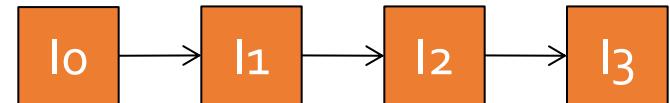


Diagrama de Precedencia

- La relación de precedencia define una relación de orden

¿Cómo se usa la concurrencia?

El orden de las instrucciones

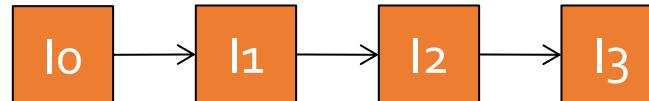


Diagrama de Precedencia

- Existe determinismo
- Al ejecutar el programa con los **mismos datos de entrada** se obtienen los **mismos resultados**
- Pero hay veces que no es necesario que una sentencia se execute antes que otra, se podrían ejecutar en **cualquier orden o simultáneamente** ¿Cómo lo podemos especificar en el código?

¿Cómo se usa la concurrencia?

El orden de las instrucciones

```
public class MaxMinConcurrent {  
  
    static volatile double n1, n2, min, max;  
  
    public static void min() {  
        new Thread(() -> min = n1 < n2 ? n1 : n2).start();  
    }  
  
    public static void max() {  
        new Thread(() -> max = n1 > n2 ? n1 : n2).start();  
    }  
  
    public static void main(String[] args) {  
        n1 = 3;  
        n2 = 5;  
        min(); // l1  
        max(); // l2  
        System.out.println("MIN = "+min+", MAX = "+max); // l3  
    }  
}
```

- Programación concurrente:
- Orden parcial

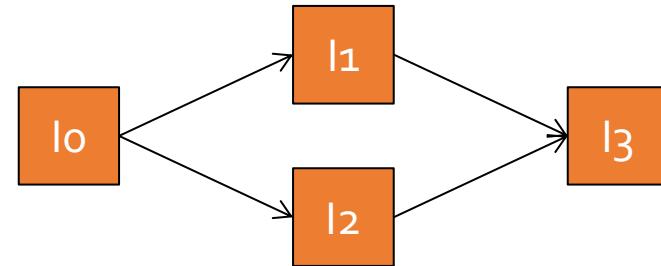


Diagrama de Precedencia

- No sabemos el orden en el que se ejecutarán las instrucciones

¿Cómo se usa la concurrencia?

El orden de las instrucciones

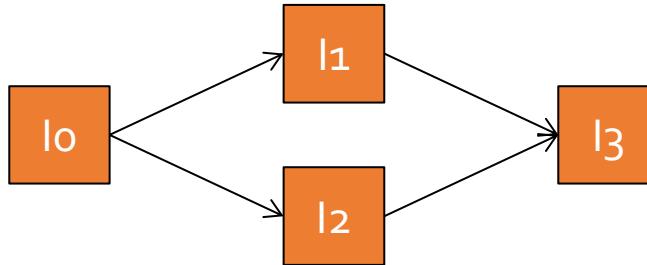


Diagrama de Precedencia

- No existe determinismo
- No se restringe el orden de ejecución de las instrucciones. Podrían ejecutarse en cualquier orden.

Gestión de hilos en Java

Espera por la finalización de un hilo

- Si un hilo quiere esperar a que otro hilo finalice, debe invocar el método `join()` en el objeto de la clase `Thread` que representa a ese hilo
- El método eleva la excepción `InterruptedException` si otro hilo le interrumpe mientras espera
- Existen versiones de `join()` para indicar el tiempo máximo de espera

Instrucciones atómicas

Recordemos...

I^a Abstracción de la Programación Concurrente

Se considera que cada proceso se ejecuta en su propio procesador

2^a Abstracción de la Programación Concurrente

Se ignoran las velocidades relativas de cada proceso, lo que posibilita considerar sólo las secuencias de instrucciones que se ejecutan

Instrucciones atómicas

¿Para qué nos sirven las abstracciones?

- Estas ideas nos permiten **abstraernos** de detalles como el **número de procesadores, su velocidad**
- Y nos permiten centrarnos en las diferentes posibles **secuencias de instrucciones** que ejecuta cada proceso
- ¿Exactamente qué **instrucciones** ejecuta un proceso?

Instrucciones atómicas

¿Qué son?

- Una **instrucción atómica** es aquella cuya ejecución es indivisible
- Independientemente del número de procesadores, una instrucción atómica de un proceso se **ejecuta completamente sin interferencias**
- Durante la ejecución de una sentencia atómica, otros procesos **no pueden interferir** en su ejecución

Instrucciones atómicas

Ejemplo

- Ejemplo: Reserva de vuelo con escala
 - Un viajero quiere ir de Madrid a Los Ángeles
 - Tiene que hacer escala en New York
 - Se debe reservar el billete Madrid-New York y también de New York-Los Ángeles
 - La reserva no se puede quedar a medias, reservando sólo un trayecto
 - O se reservan ambos trayectos o no se reserva ninguno

Instrucciones atómicas

¿Para qué sirven?

- En programación concurrente, es muy importante conocer las **instrucciones atómicas que ejecuta el procesador**
- Esas instrucciones atómicas serán las que se ejecuten **completamente sin interferencias** de otros procesos
- Para desarrollar aplicaciones concurrentes en un lenguaje de programación, hay que conocer **qué sentencias del lenguaje se corresponden a sentencias atómicas**
- Las sentencias **no atómicas** ejecutadas por un proceso, podrán verse **afectadas** por la ejecución de otros procesos

Instrucciones atómicas

¿En qué consisten?

Sentencia Java

```
x = x + 1
```



Corresponde a las
instrucciones atómicas

Instrucciones Atómicas

```
LOAD R,x  
ADD R,#1  
STR R,x
```

- Carga la variable x en el registro R del procesador
- Suma 1 al registro R del procesador
- Guarda el valor del registro R en la variable x

Intercalación (*Interleaving*)

¿Qué es?

- La 1^a y 2^a abstracción nos permiten pensar solo en la **secuencia de instrucciones** que ejecuta cada proceso en su procesador
- Es bastante complicado pensar en la **ejecución en paralelo** de múltiples secuencias de instrucciones (una secuencia por cada proceso)
- Lo que se hace para estudiar el comportamiento de un programa concurrente es considerar que todas las sentencias de todos los procesos se **intercalan en una única secuencia**

Intercalación (*Interleaving*)

¿Qué es?

- No hay solapamientos
- La ejecución de dos instrucciones atómicas en paralelo tiene los mismos resultados que una después de otra (de forma secuencial)

3^a Abstracción de la Programación Concurrente

Se considera que las secuencias de ejecución de las acciones atómicas de todos los procesos se intercalan en una única secuencia

Intercalación (*Interleaving*)

¿Qué es?

- No hay que estudiar una única intercalación de instrucciones
- Todas las intercalaciones son posibles
- Hay que estudiar lo que ocurre en todas las posibles intercalaciones de instrucciones atómicas para comprender el comportamiento del programa en cualquier ejecución

Intercalación (*Interleaving*)

¿Qué es?

- Multiprogramación
 - Realmente las instrucciones se ejecutan de forma intercalada porque sólo hay un procesador
- Multiproceso
 - Si dos instrucciones compiten por un mismo recurso, el hardware se encarga de que se ejecuten de forma secuencial
 - Si dos instrucciones no compiten, son independientes, el resultado es el mismo en paralelo que de forma secuencial

Intercalación (*Interleaving*)

¿Qué es?

- Todas las abstracciones se pueden resumir en sólo una que nos permite estudiar el comportamiento de los programas concurrentes

Abstracción de la Programación Concurrente

Es el estudio de las secuencias de ejecución intercalada de las instrucciones atómicas de los procesos secuenciales
(Ben-Ari, 1990)

Indeterminismo

¿Qué es?

- Al ejecutar un programa concurrente, se ejecutarán las sentencias con una intercalación determinada y se obtendrá un resultado
- Puesto que todas las intercalaciones de instrucciones atómicas son posibles, un mismo programa puede obtener **resultados diferentes en diferentes ejecuciones**
- Cuando un mismo programa obtiene resultados diferentes dependiendo de la ejecución concreta, se dice que es **indeterminista**

Indeterminismo

Ejemplo

```
public class IncDec {  
  
    static volatile double x;  
  
    public static void main(String[] args)  
        throws InterruptedException {  
        Thread inc = new Thread(() -> {  
            x = x + 1; ----->  
            try {  
                Thread.sleep(1);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        });  
        Thread dec = new Thread(() -> {  
            x = x - 1; ----->  
            try {  
                Thread.sleep(1);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        });  
        inc.start();  
        dec.start();  
        inc.join();  
        dec.join();  
        System.out.println("X = "+x);  
    }  
}
```

Incrementar

```
LOAD R, x  
ADD R, #1  
STR R, x
```

Decrementar

```
LOAD R2, x  
SUB R2, #1  
STR R2, x
```

Indeterminismo

Possible intercalación de instrucciones

	Incrementar	Decrementar	x	R	R2
1		LOAD R2, x	0		0
2	LOAD R, x		0	0	0
3		SUB R2, #1	0	0	-1
4	ADD R, #1		0	1	-1
5		STR R2, x	-1	1	-1
6	STR R, x		1	1	-1

Incrementar

```
LOAD R, x  
ADD R, #1  
STR R, x
```

Decrementar

```
LOAD R2, x  
SUB R2, #1  
STR R2, x
```

Indeterminismo

Possible intercalación de instrucciones

	Incrementar	Decrementar	x	R	R2
1		LOAD R2, x	0		0
2	LOAD R, x		0	0	0
3	ADD R, #1		0	1	0
4		SUB R2, #1	0	1	-1
5	STR R, x		1	1	-1
6		STR R2, x	-1	1	-1

Incrementar

```
LOAD R, x  
ADD R, #1  
STR R, x
```

Decrementar

```
LOAD R2, x  
SUB R2, #1  
STR R2, x
```

Indeterminismo

Possible intercalación de instrucciones

	Incrementar	Decrementar	x	R	R2
1		LOAD R2, x	0		0
2		SUB R2, #1	0		-1
3		STR R2, x	-1		-1
4	LOAD R, x		-1	-1	-1
5	ADD R, #1		-1	0	-1
6	STR R, x		0	0	-1

Incrementar

```
LOAD R, x  
ADD R, #1  
STR R, x
```

Decrementar

```
LOAD R2, x  
SUB R2, #1  
STR R2, x
```

Indeterminismo

¿Qué es?

- Hemos visto **tres intercalaciones** posibles de las instrucciones atómicas de un programa concurrente en las que se obtiene un **resultado diferente**
- Hay 20 posibles intercalaciones
 - En sólo 2 de ellas se obtiene un 0
 - En 9 intercalaciones se obtiene un 1
 - En 9 intercalaciones se obtiene un -1

Conclusiones

Ventajas de la concurrencia

- Aumento de la eficiencia: Aprovechamiento de procesador o procesadores
- Hay tipos de programas que necesitan llevar a cabo varias tareas de forma concurrente y por tanto es obligatoria la concurrencia
- La posibilidad de tener tareas “en primer plano” y tareas en segundo plano
 - Tareas en primer plano: Interfaz de usuario
 - Tareas en segundo plano: Antivirus, impresión, ...

Conclusiones

Ventajas de la concurrencia

- La tendencia actual para aumentar la potencia de cómputo de los sistemas informáticos consiste en la creación de procesadores con varios *cores* o núcleos
- Sólo las aplicaciones concurrentes, que dividan las tareas que se deben realizar en varios procesos podrán aprovechar la potencia de calculo de los nuevos sistemas
- Esto implica que aplicaciones que actualmente están desarrolladas de forma secuencial, sin concurrencia, deberán actualizarse para aprovechar la potencia de calculo

Conclusiones

Desventajas de la concurrencia

- Desarrollar programas concurrentes es más difícil que desarrollar programas secuenciales
 - Son más difíciles de programar
 - Es más difícil comprobar que su funcionamiento es el correcto, porque hay que considerar todas las posibles intercalaciones
 - Que una ejecución sea correcta con unos datos de entrada no implica que todas las ejecuciones sean correctas con esos mismos datos (como ocurre con los programas secuenciales).
 - Hay intercalaciones **problemáticas** que pueden aparecer sólo en raras ocasiones

Conclusiones

Indeterminismo

- Cuando se implementa un programa concurrente se necesita controlar el indeterminismo (eliminarlo)
- No obstante, hay veces que permitiendo cierto indeterminismo se consiguen programas más eficientes
 - Ejemplo: En la descarga de ficheros P2P, cada usuario puede descargarse cualquier fragmento de otro usuario, sin importar cual de ellos. Restringir e imponer un orden podría reducir el rendimiento de la aplicación

Tema I: Introducción

PROGRAMACIÓN CONCURRENTE



Universidad
Rey Juan Carlos