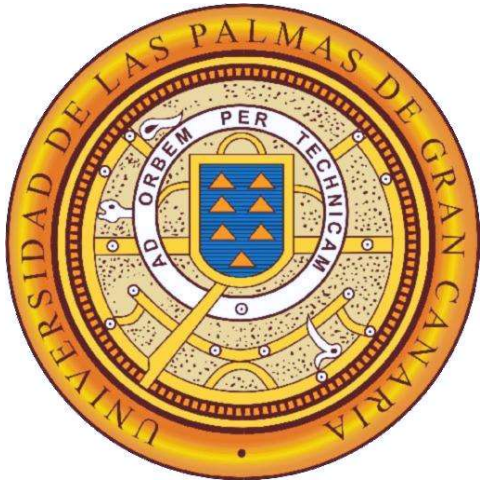


Universidad de Las Palmas de Gran Canaria

Departamento de Ingeniería Telemática

Centro de Tecnología Médica
http://www.ctm.ulpgc.es

Tema 1: Introducción a la Programación Concurrente

Programación Concurrente

Escuela Técnica Superior de Ingenieros de Telecomunicación

Conceptos Fundamentales

La idea de programación concurrente siempre ha estado asociada a los sistemas operativos: Un sólo procesador de gran capacidad debía repartir su tiempo entre muchos usuarios.

La programación de estos sistemas se hacía a bajo nivel (ensamblador). Posteriormente aparecerían lenguajes de alto nivel con soporte para este tipo de programación.

Su utilización y potencial utilidad se apoya en: threads o hilos, java e internet.

Definición: Se habla de concurrencia cuando ocurren varios sucesos de manera contemporánea.
En base a esto, la concurrencia en computación está asociada a la "ejecución" de varios procesos que coexisten temporalmente.

Programación Concurrente

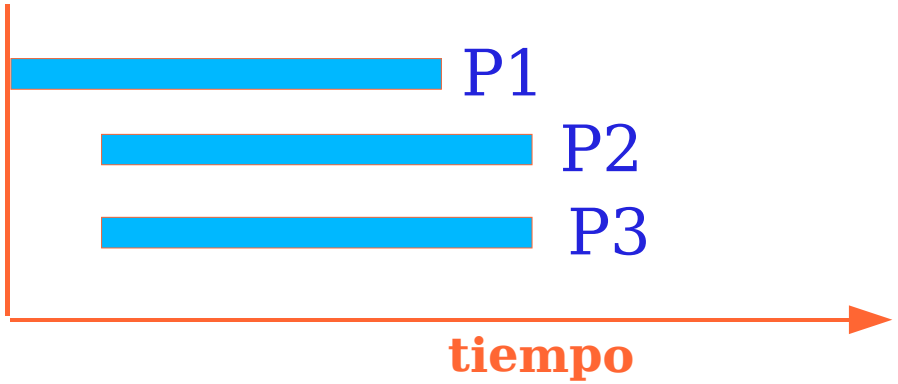
Para definirla correctamente, debemos diferencias entre programa y proceso.

Programa: Conjunto de sentencias/instrucciones que se ejecutan secuencialmente. Se asemeja al concepto de clase dentro de la POO. Es por tanto un concepto estático.

Proceso: Básicamente, se puede definir como un programa en ejecución. Líneas de código en ejecución de manera dinámica. Se asemeja al concepto de objeto en POO.

Concurrencia

La concurrencia aparece cuando dos o más procesos son contemporáneos. Un caso particular es el paralelismo (programación paralela).



Los procesos pueden “competir” o colaborar entre sí por los recursos del sistema. Por tanto, existen tareas de colaboración y sincronización.

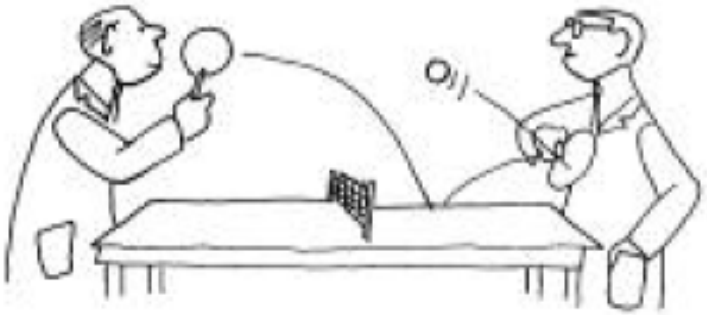
La programación concurrente se encarga del estudio de las nociones de ejecución concurrente, así como sus problemas de comunicación y sincronización.

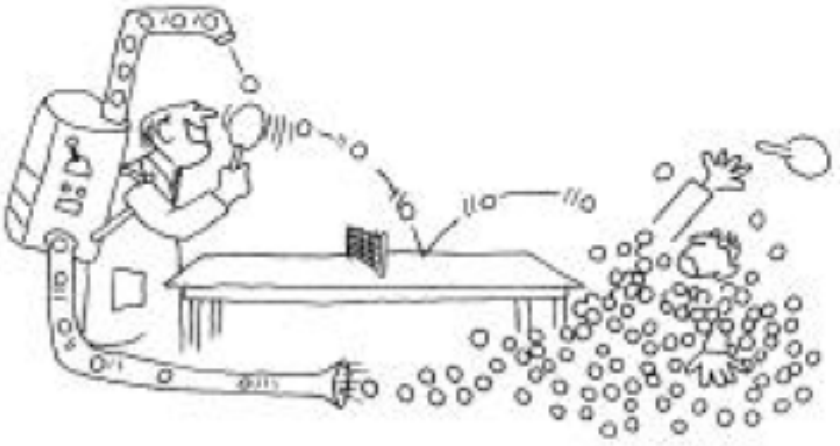
¿Cuales son sus beneficios?

- Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden "repartir" entre procesadores o gestionar en un único procesador según importancia.

- Solución a problemas de esta naturaleza. Existen algunos problemas cuya solución es más fácil utilizando esta metodología.
  - Sistemas de control: Captura de datos, análisis y actuación (p.ej. sistemas de tiempo real).
  - Tecnologías web: Servidores web que son capaces de atender varias peticiones concurrentemente, servidores de chat, email, etc.
  - Aplicaciones basabas en GUI: El usuario hace varias peticiones a la aplicación gráfica (p.ej. Navegador web).
  - Simulación: Programas que modelan sistemas físicos con autono--mía.
  - Sistemas Gestores de Bases de Datos: Cada usuario un proceso.

¿Cuales son sus beneficios?

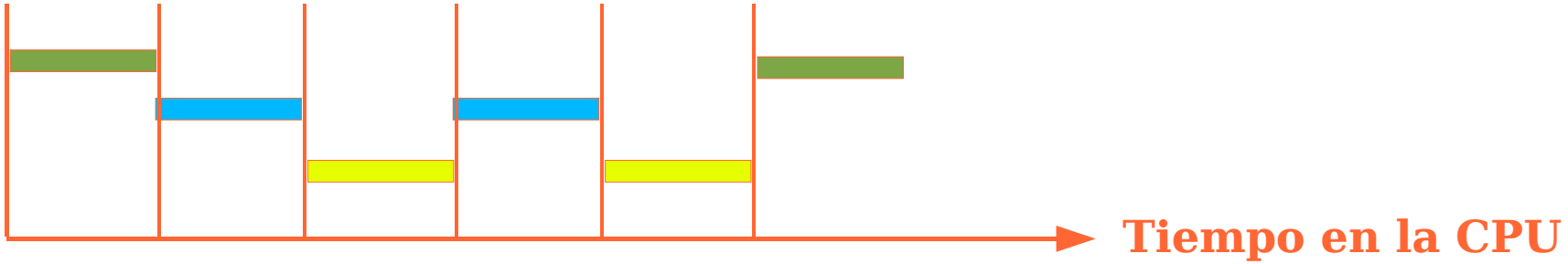




Concurrencia y hardware

Hasta ahora sólo hemos hablado del software, aunque el hardware y su topología es importante para abordar cualquier tipo de problema.

- Sistemas monoprocesador. Podemos tener concurrencia, gestionando el tiempo de procesador para cada proceso.



- Sistemas multiprocesador. Un proceso en cada procesador.

Éstos pueden ser de memoria compartida (fuertemente acoplados) o con memoria local a cada procesador (débilmente acoplados).
Un ejemplo muy conocido y útil son los sistemas distribuidos (p.ej. Beowulfs).
En relación a la concurrencia se pueden clasificar en aquellos que funcionan con variables/memoria compartida o paso de mensajes.

En resumen:

• Programa concurrente: Ejecución de acciones simultáneamente.

• Programa paralelo: Programa que se ejecuta en un sistema multi-procesador.

• Programa distribuido: Programa paralelo para ejecutarse en sistemas distribuidos.

¿Qué se puede ejecutar concurrentemente?

```
x=x+1;
y=x+1;
```

La primera instrucción se debe ejecutar antes de la segunda!!

```
x=1;y=2;z=3;
```

El orden no interviene en el resultado final!!

Condiciones de Bernstein

Para que dos conjuntos de instrucciones se puedan ejecutar concurrentemente se tiene que cumplir que:

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

Ejemplo:

Denotamos por L(lectura) y E(escritura)

P1: a=x+y;	L(P1)={x,y}	E(P1)={a}
P2: b=z-1;	L(P2)={z}	E(P2)={b}
P3: c=a-b;	L(P3)={a,b}	E(P3)={c}
P4: w=c+1;	L(P4)={c}	E(P4)={w}

Se debe cumplir:

$L(Pi) \cap E(Pj)=\emptyset$;
$E(Pi) \cap L(Pj)=\emptyset$;
$E(Pi) \cap E(Pj)=\emptyset$;

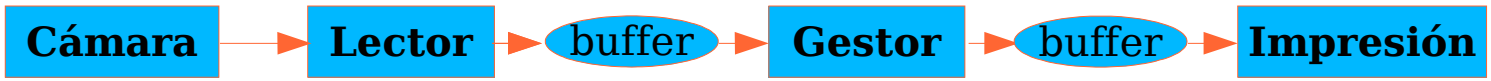
Características de los sistemas concurrentes

• Orden de ejecución: A diferencia de los programas secuenciales el flujo del programa sigue un orden parcial. Ante una misma entrada no se sabe cuál va a ser el orden seguido.

• Indeterminismo: El orden parcial produce consecuentemente un comportamiento indeterminista. Es decir, repetidas ejecuciones sobre un mismo conjunto de datos resultan “diferentes resultados”.

Problemas inherentes a los sistemas concurrentes

• Exclusión mutua: Como lo que realmente se ejecuta concurrentemente son las instrucciones de ensamblador, cuando se comparten variables se excluyen los valores. Por ejemplo, dos bucles que hacen x=x+1.
• Condición de sincronización: La necesidad de coordinar los procesos. Por ejemplo un capturador de imágenes con colas de impresión, el juego del pañuelo, etc.



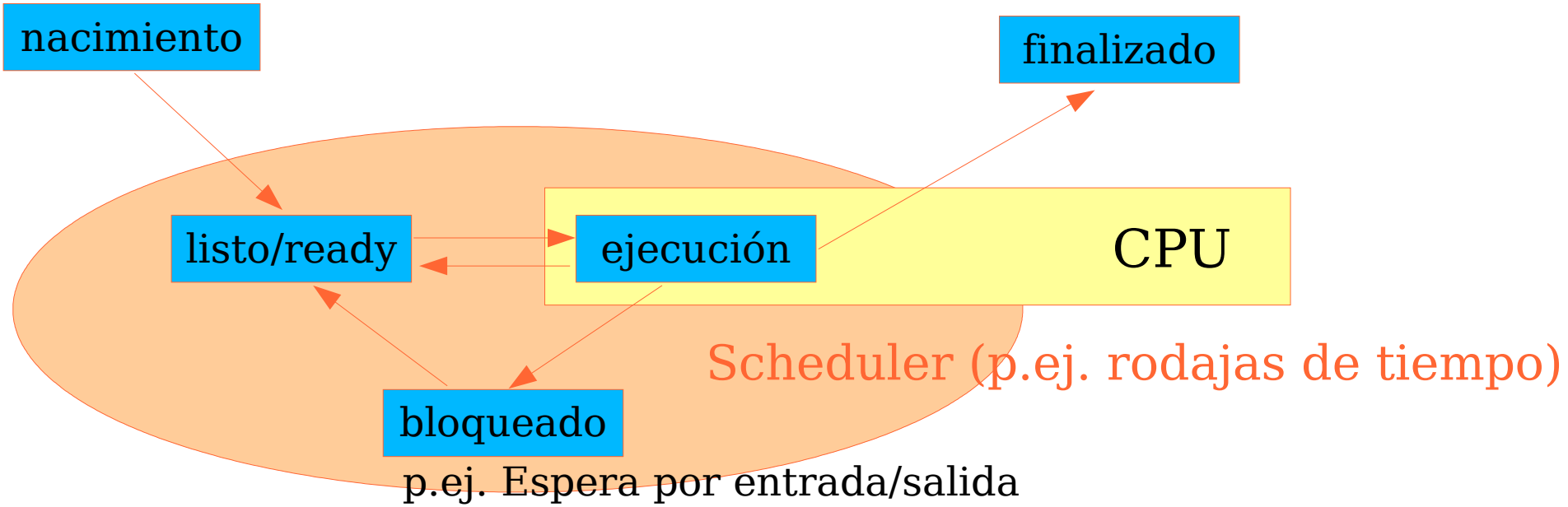
Procesos

Habíamos definido un proceso como un programa en ejecución. Pero, ¿cómo se ejecuta?, ¿cuándo?, ¿finaliza?, ¿nace, crece y se reproduce?.

Procesos

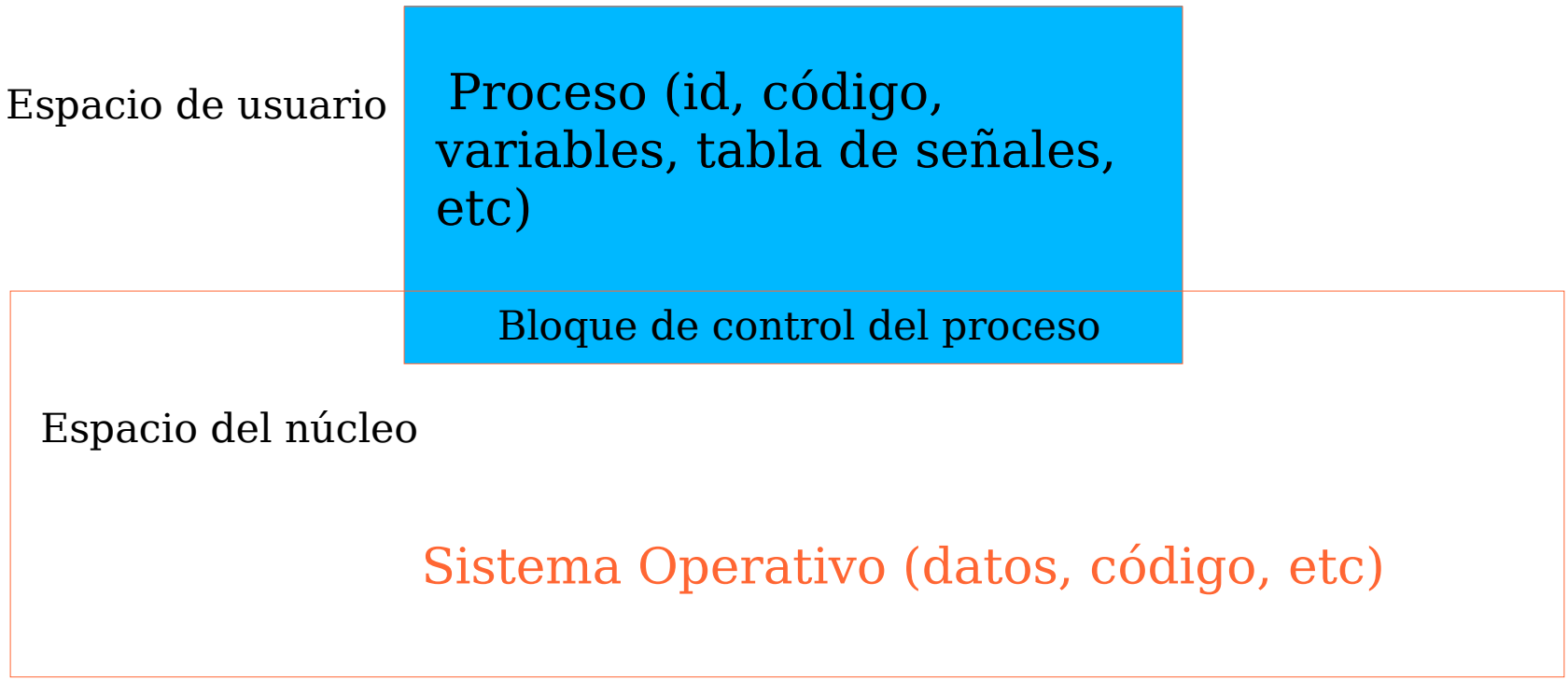
Habíamos definido un proceso como un programa en ejecución. Pero, ¿cómo se ejecuta?, ¿cuándo?, ¿finaliza?, ¿nace, crece y se reproduce?.

Ciclo de vidad de un proceso

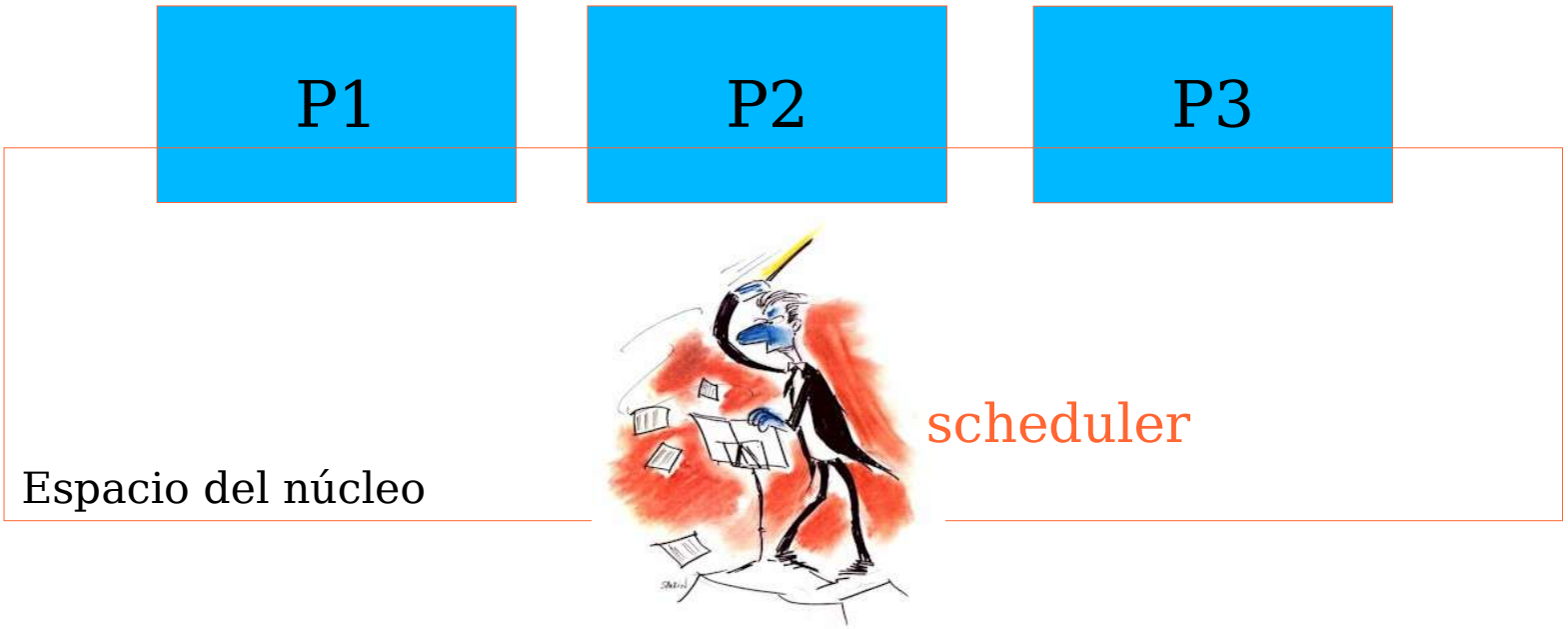


Disposición de memoria de un proceso

Básicamente, existe un espacio de usuario y un espacio de núcleo.



Cada proceso tiene sus propias características: código, variables, id, contadores, pila, etc. Son monohilo.



La gestión y el cambio de contexto de cada proceso es muy costoso. Se debe actualizar los registros de uso de memoria, y controlar los estados en los que quedan los procesos.

Procesos de procesos

- ¿Qué son?
- ¿Para qué?
- etc.

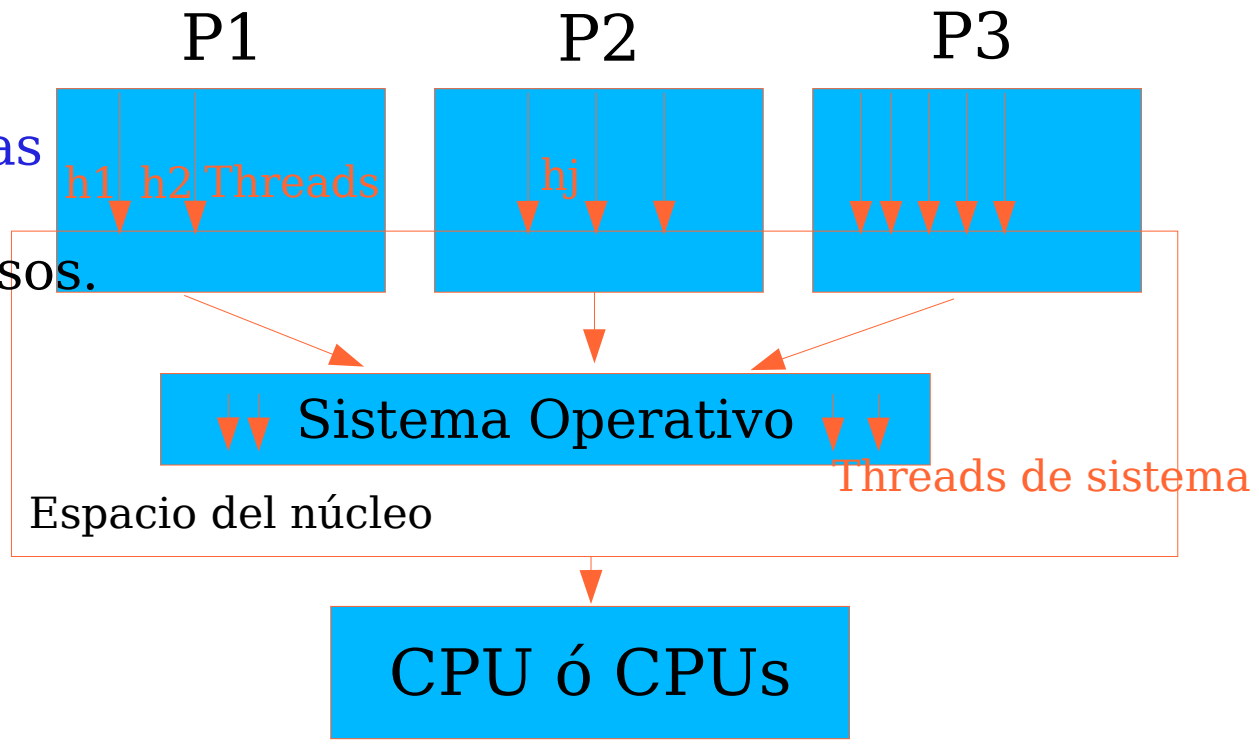
Threads/hilos

Definición: Una secuencia de control dentro de un proceso que ejecuta sus instrucciones de forma independiente.

Existe concurrencia a dos niveles: entre procesos y entre threads.

Procesos: entidades pesadas con espacio en el núcleo. Cambios de contexto costosos.

Threads: entidades ligeras en el espacio de usuario. Cambios de contexto poco costosos.



Los threads/hilos pueden estar en dos niveles: a nivel usuario (p.ej. java) o a nivel del sistema operativo (hilos del sistema).

Los threads/hilos de sistema dan soporte a los threads/hilos de usuario mediante un API (Application Program Interface).

Estándares

Cada sistema operativo implementa los threads/hilos de sistema de manera diferente: win32, OS/2 y POSIX (pthreads).

Implementación

A nivel usuario (librería) o a nivel de núcleo (llamadas al sistema). El estándar POSIX es del primer tipo.

Planificación

Existen procesadores lógicos (los threads compiten por cada procesador lógico), y los procesadores lógicos compiten por los físicos (SOLARIS).

Threads/hilos en Java

Java proporciona un API para el uso de hilos: clase Thread dentro del paquete java.lang.Thread.
Es de gran utilidad tener un lenguaje de alto nivel para programar concurrentemente utilizando threads/hilos, de ahí el potencial y la fama de Java.

Cuando arranca un programa existe un hilo principal (main), y luego se pueden generar nuevos hilos que ejecutan código en objetos diferentes o el mismo.

La clase Thread dispone de una serie de métodos para caracterizar el thread/hilo en el programa: isAlive(), isDaemon(), setName(), setDaemon(), run(), etc.