1. Introducción

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:

- La aparición del concepto de **thread o hilo** que hace que los programas puedan ejecutarse con mayor velocidad.
- La aparición de lenguajes como **Java**, lenguaje orientado a objetos de propósito general que da soporte directamente a la programación concurrente mediante la inclusión de primitivas específicas.
- La aparición de **Internet** que es un campo abonado para el desarrollo y la utilización de programas concurrentes. Cualquier programa de Internet en el que podamos pensar tales como un navegador, un chat, etc, están programados usando técnicas de programación concurrente.

Para definir correctamente la programación concurrente debemos diferenciar entre programa y proceso.

PROGRAMA

Un **programa** es un conjunto de instrucciones. Es, simplemente, un texto que consiste en una secuencia de líneas de código que dicen qué hacer con un conjunto de datos de entrada para producir algún tipo de salida. Se trata de algo **estático**. Para que el programa pueda hacer algo de verdad hay que ponerlo en ejecución.

PROCESO

Un **proceso** es un programa en ejecución.

Se debe tener muy presente que un proceso asocia **programa+actividad**. Cuando decimos que un proceso es un programa en ejecución, nos referimos al hecho de llevar a cabo o realizar las instrucciones que contiene el programa en el orden establecido. Un programa es una lista de instrucciones escritas en un papel, un fichero en disquete, disco duro, memoria RAM o cualquier otro soporte, pero el simple hecho de que estas instrucciones estén escritas no implica que se estén llevando a cabo. Pues bien, cuando se leen estas instrucciones y se hacen ejecutar, entonces ya tenemos programa+actividad, es decir, un proceso.

Hacemos hincapié en actividad para diferenciarlo bien de un mero programa, ya que una característica básica de **los programas** es que **son estáticos** (una secuencia de órdenes que, por mucho que la miremos, no varía nunca). Las variables no tienen valores, las rutinas no tienen dirección, las condiciones están sin evaluar. Un programa es simplemente un algoritmo a ejecutar. En cambio, un proceso es **dinámico**. Tiene vector de estado indicando el momento y estado de su ejecución, las variables tienen valores, las rutinas se encuentran en alguna dirección de memoria, y las condiciones son evaluables. El proceso es la *ejecución del algoritmo.*

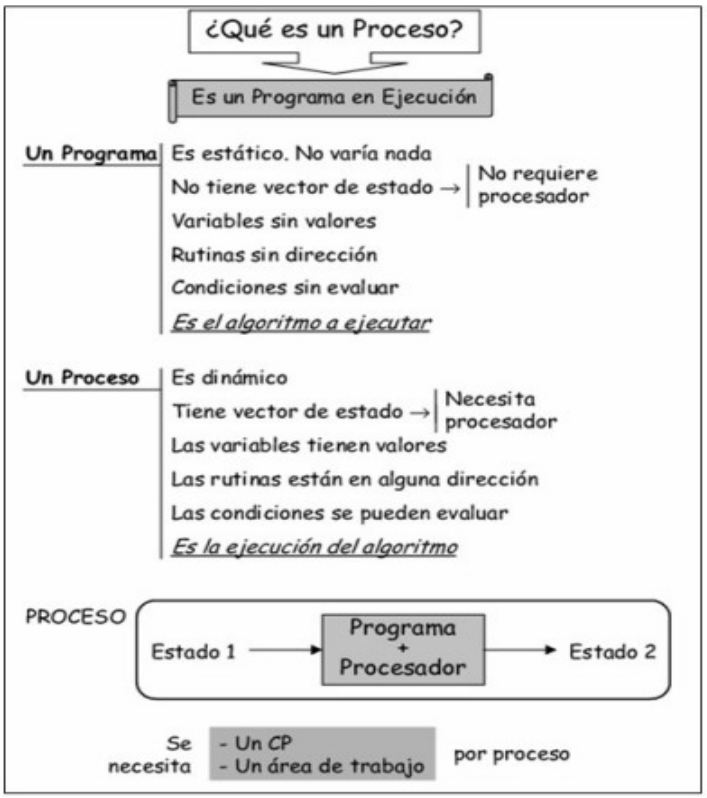
Un ejemplo para describir la relación programa-proceso puede ser el manual de montaje de un juguete que viene despiezado. Las instrucciones de montaje que vienen en la caja son algo estático (por eso aunque dejemos mucho tiempo las instrucciones en la caja, el juguete no aparece construido al cabo de un tiempo). Sin embargo, cuando cogemos las instrucciones y empezamos a hacer lo que indican, es cuando empieza la actividad, entonces comienza el proceso de montaje.

Si en la construcción del juguete, quien lleva a cabo las instrucciones del manual de montaje es la persona, que lee las instrucciones y las va ejecutando; en un ordenador, el ente que va leyendo las instrucciones del programa (que está en memoria) y ejecutándolas, es la CPU. Así, tenemos que el tándem programa+procesador es el que hace que el estado de desarrollo de un determinado trabajo evolucione y cambie de un estado a otro.

Al igual que para la construcción del juguete se necesita un área de trabajo (una mesa o algo similar), para la ejecución de un programa, también se requiere un área de trabajo, esto es, la pila del proceso. La pila es la zona de memoria en la que se guardan los parámetros, variables locales, direcciones de retorno de subprogramas y, en general, los datos temporales del proceso. Un proceso también se caracteriza porque en un momento dado de su ejecución dispone de ciertos valores en los registros del procesador, entre ellos, el **contador de programa** que contiene la dirección de la instrucción que va a ejecutar el proceso a continuación. Así, el contador de programa es el registro que indica el punto de ejecución del programa en el que se encuentra el proceso.

Se debe tener en cuenta que la relación entre un programa y su proceso no es biunívoca. Esto se debe al hecho de que en un momento dado puede haber varios procesos correspondientes al mismo programa. Por ejemplo, en una clase de trabajos manuales pueden estar escritas en la pizarra las instrucciones de montaje de un juguete (un programa) y todos los alumnos están montando un juguete siguiendo las mismas instrucciones (varios procesos).

De igual manera que en POO puede haber múltiples objetos de una clase determinada, aquí puede haber múltiples procesos que corresponden al mismo programa. Como ejemplo consideremos un servidor de aplicaciones donde reside una aplicación de navegador de Internet y existen varios usuarios ejecutando ese navegador, cada uno de ellos navegando por un sitio diferente. Cada instancia del programa es un proceso. Cada proceso tendrá su propio contador de programa, así como sus propios registros, pila y variables.

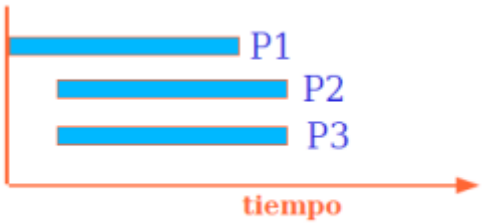


2. LA PROGRAMACIÓN CONCURRENTE

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

Los procesos pueden "competir" o colaborar entre sí por los recursos del sistema. Por lo 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**.



2.1 BENEFICIOS DE LA PROGRAMACIÓN CONCURRENTE

El trabajar con procesos concurrentes va añadir complejidad a la hora de programar, pero ¿cuáles son los beneficios que aporta la programación concurrente?

- 1. Velocidad de ejecución

  Cuando la puesta en ejecución de un programa conlleva la creación de varios procesos y el sistema consta de más de un procesador, existe la posibilidad de asignar un proceso a cada procesador de tal forma que el programa se ejecuta de una forma más rápida. Los programas de cálculo numérico son grandes beneficiados de este hecho.

- 2. Solución de problemas inherentemente concurrentes

  Existen algunos problemas cuya solución es más fácil utilizando esta metodología:

  – sistemas de control: captura de datos, análisis y actuación (por ej. sistemas de tiempo real).
  – Tecnologías web: servidores web que son capaces de atender varias peticiones concurrentemente, servidores de chat, email..
  – aplicaciones basadas en GUI (Interfaz Gráfica de Usuario): el usuario hace varias peticiones a la aplicación gráfica, por ej. un navegador web
  – simulación: programas que modelan sistemas físicos con autonomía.
  – Sistemas gestores de bases de datos: cada usuario un proceso.

- 3. Un mejor aprovechamiento de la CPU.

2.2 CONCURRENCIA Y ARQUITECTURAS HARDWARE

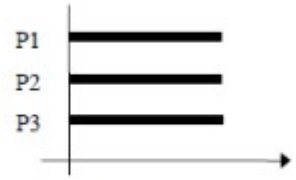
Cuando hablemos de hardware nos estamos refiriendo fundamentalmente al número de procesadores en el sistema.

SISTEMAS MONOPROCESADOR

Incluso en un sistema con un solo procesador podemos tener una ejecución concurrente de procesos. Evidentemente todos los procesos no pueden estar ejecutándose al mismo tiempo sobre el procesador, sólo uno de ellos podrá estar haciéndolo, pero la sensación que le da al usuario o grupo de usuarios es la de estar ejecutándose al mismo tiempo. Esto es debido a que el Sistema Operativo (SO) va alternando el tiempo de procesador entre los distintos procesos. De esta forma, cuando un proceso que ocupa el procesador en un momento determinado necesita hacer una operación de entrada/salida, puede abandonar el procesador para que otro proceso pueda ocuparlo y aprovechar ciclos de procesador. En la figura puede verse cómo el tiempo de procesador es repartido entre tres procesos. Esta forma de gestionar los procesos en un sistema monoprocesador recibe el nombre de **multiprogramación** y es otro de los beneficios de la programación concurrente: un mayor aprovechamiento del procesador.

SISTEMAS MULTIPROCESADOR

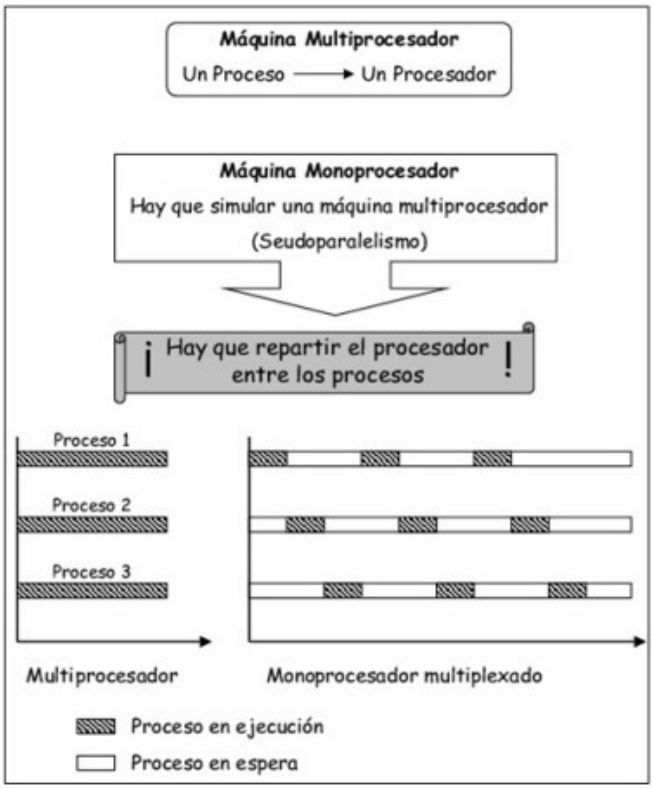
Un sistema **multiprocesador** es aquel en el que existe más de un procesador. Esto permite que exista un paralelismo real entre los procesos ya que idealmente cada procesador podría ejecutar un proceso.



Paralelismo.

Suele denominarse **multiproceso** a la gestión de varios procesos dentro de un sistema multiprocesador donde cada procesador puede acceder a una memoria común y **procesamiento distribuido** a la gestión de varios procesos en procesadores separados, cada uno con su memoria local.

2.3 PROGRAMACIÓN PARALELA



Un **programa paralelo** es un tipo de programa concurrente diseñado para ejecutarse en un sistema multiprocesador.

NECESIDAD DE LA PROGRAMACIÓN PARALELA

- Resolver problemas que no caben en una CPU.
- Resolver problemas que no se resuelven en un tiempo razonable.
- Se pueden ejecutar:
  - Problemas mayores.
  - Problemas mas rápidamente.
- Decrementa la complejidad de un algoritmo al usar varios procesadores.

VENTAJAS / INCONVENIENTES

- Rápida.
- Ahorro de recursos.
- Los programas de ordenador paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de software, siendo las condiciones de carrera[1] los más comunes.
- El incremento de velocidad que consigue un programa.

1Múltiples procesos se encuentran en **condición de carrera** si el resultado de los mismos depende del orden de su ejecución. Una condición de carrera se da principalmente cuando varios procesos acceden al mismo tiempo y cambian el estado de un recurso compartido (por ejemplo una variable), obteniendo de esta forma un valor no esperado de ese recurso.

2.4 CARACTERÍSTICAS DE UN PROGRAMA CONCURRENTE.

- **Orden de ejecución de las instrucciones.**

  La programación secuencial define un orden total de las instrucciones. Ante un conjunto de datos de entrada el flujo de ejecución es siempre el mismo

  Un programa concurrente define un orden parcial de ejecución Ante un conjunto de datos de entrada no se puede saber cual va a ser el flujo de ejecución

- **Indeterminismo**

  El orden parcial implica el no determinismo de los programas concurrentes. Es decir, puede producir diferentes resultados cuando se ejecuta repetidamente sobre el mismo conjunto de datos de entrada.

  El no determinismo es una propiedad inherente a la concurrencia. Por culpa del no determinismo, es más difícil analizar y verificar un algoritmo concurrente, pero que existan varias posibilidades de salida NO significa necesariamente que un programa concurrente sea incorrecto

2.5 PROBLEMAS DE LA PROGRAMACIÓN CONCURRENTE

A la hora de trabajar con programas concurrentes podemos encontrarnos con varios problemas. Se dice que los problemas inherentes a la programacion concurrente son la exclusión mutua y la condición de sincronización.

- **Exclusión Mutua:** Para explicar la exclusión mutua es necesario definir las regiones o secciones críticas. Se denomina **sección crítica**, a la porción de código de un programa de computador en la cual se accede a un recurso compartido (estructura de datos, variable o dispositivo) que no debe ser accedido por más de un proceso o hilo en ejecución. La exclusión mutua consiste en que un solo proceso tiene acceso a un recurso, hay que garantizar que si un proceso adquiere un recurso, otros procesos deberán esperar a que sea liberado.

  Un ejemplo, en el juego del pañuelo: el pañuelo ha de adquirirse en exclusión mutua, o lo coge un jugador o lo coge otro. Si lo cogen los dos a la vez puede llegar a romperse, llevando a un mal funcionamiento del sistema.

- **Condición de sincronización:** Hay situaciones en las que un recurso compartido por varios procesos, como puede ser el buffer o la cola de impresión, por ejemplo, se encuentra en un estado en el que un proceso no puede hacer una determinada acción con él hasta que no cambie su estado. A esto se le denomina **condición de sincronización.**

  En nuestro ejemplo, un jugador ha de esperar a que digan su número para poder salir corriendo.

  Si sale corriendo antes llevaría a un mal funcionamiento del sistema.

  La programación concurrente ha de proporcionarnos mecanismos para bloquear procesos que no puedan hacer algo en un momento determinado a la espera de algún evento (p. ej. que el buffer deje de estar vacío), pero también que permita desbloquearlos cuando ese evento haya ocurrido.

Aparte de estos se dan otros problema como:

- **Interbloqueo:** Se produce una situación de interbloqueo cuando todos los procesos están esperando porque ocurra un evento que nunca se producirá. Hay que garantizar que no se producen este tipo de situaciones. En nuestro ejemplo se produciría si un jugador se guarda el pañuelo y se va para su casa. El juez esperaría porque le devolvieran el pañuelo y los jugadores esperarían porque el juez dijese su número, pero ninguno de estos eventos va a ocurrir nunca. Se suele conocer también con el nombre de deadlock o abrazo mortal. Los procesos esperan unos por otros en círculo.
- **Postergación indefinida o inanición:** Se da cuando un proceso es desestimado a la hora de ejecutarse. Por ejemplo cuando un proceso está esperando a entrar en la CPU pero siempre hay otros procesos con mayor prioridad que se lo impiden.

Algunas herramientas para manejar estos problemas de concurrencia son:

- **Semáforos:** Está técnica permite resolver la mayoría de los problemas de sincronización entre procesos y forma parte del diseño de muchos sistemas operativos y de lenguajes de programación concurrentes.

  Un semáforo binario es un indicador de condición que indica si un recurso puede ser usado o no.

  Un semáforo binario trabaja de forma muy simple, tiene 2 estados: 0 y 1. Si S = 1 entonces el recurso está disponible, y el proceso lo puede tomar. Si S =0 el recurso no está disponible y el proceso debe esperar.

  Además tienen tres operaciones: Inicializar (Init), Espera (wait), Señal (signal).
- **Monitores:** Definimos un monitor como un conjunto de procedimientos que proporcionan el acceso garantizando exclusión mutua a un recurso compartido por un grupo de procesos. Los procedimientos van encapsulados dentro de un módulo que tiene la propiedad especial de que sólo un proceso puede estar activo cada vez para ejecutar un procedimiento del monitor.
- **Mensajes:** Los mensajes se pueden usar como una solución al problema de la concurrencia de procesos que brinda tanto sincronización como comunicación siendo útil para sistemas centralizados y distribuidos.