PROGRAMACIÓN DE SERVICIOS Y PROCESOS

Introducción a la concurrencia

Procesos

CONCEPTOS BÁSICOS

- Un **proceso** es un **programa en ejecución** siendo un programa un conjunto de instrucciones que el sistema operativo ejecuta para realizar determinadas acciones de un sistema informático.
- **Características**
  - *Para **iniciar su ejecución** ha de residir en memoria completamente y que le hayan sido asignados todos los recursos que necesita para su operación.*
  - *Los procesos se encuentran **protegidos** de otros procesos, es decir, ningún proceso puede acceder al área de memoria asignada a otro proceso.*
  - *Todos los procesos tienen un **identificador** que lo discrimina de los demás. Este id es asignado por el sistema operativo.*
  - *Todos los procesos tienen una estructura de datos denominada **bloque de control de proceso** que almacena información sobre dicho proceso, como por ejemplo: estado, identificador, prioridad, ubicación y recursos utilizados.*

Procesos

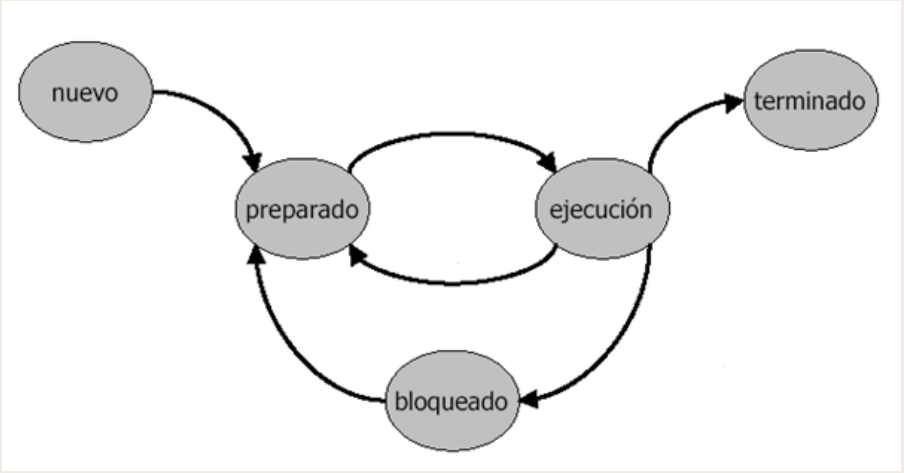
CONCEPTOS BÁSICOS

- Se denomina **hebra** o **hilo** (thread) a un punto de ejecución de un proceso.
- Cada proceso tendrá **una o varias hebras**.
- Las hebras representan un método para **mejorar la eficacia y rendimiento** del sistema operativo

Procesos

ESTADOS DE UN PROCESO

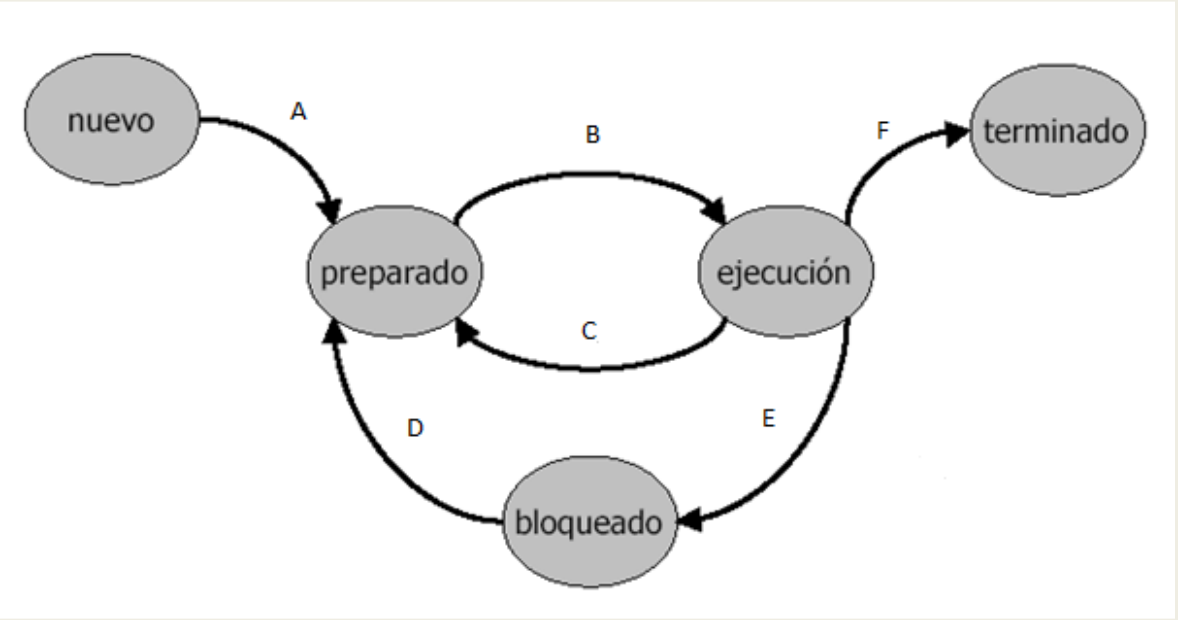
- **Nuevo**: el proceso acaba de ser creado.
- **Preparado**: se considera que un proceso está preparado para ejecutarse cuando tiene todos los recursos que necesita asignados y está esperando que la CPU le asigne tiempo de ejecución.
- **Ejecución**: un proceso está en ejecución cuando tiene asignado tiempo de CPU y está ejecutando las instrucciones que lo componen.
- **Bloqueado**: un proceso está bloqueado o detenido cuando estando en ejecución ocurren determinadas circunstancias como por ejemplo que otro proceso necesite algún recurso que este tenga asignado.
- **Terminado**: el proceso ha terminado de ejecutar todas las instrucciones.



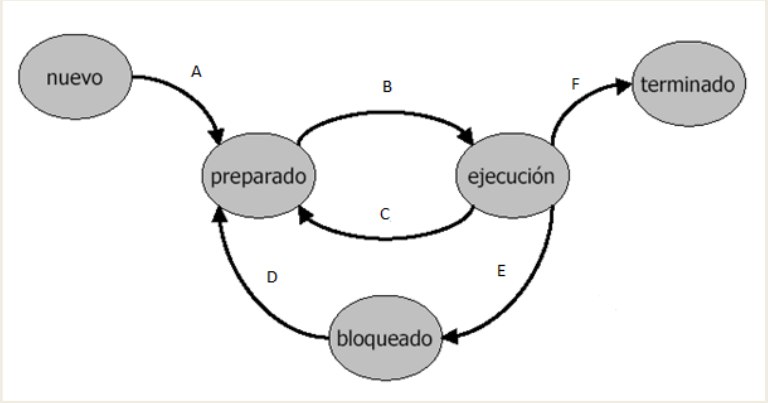
Procesos

CONCEPTOS BÁSICOS

- Los cambios de estado en los que se va encontrando un proceso se denominan transiciones



Procesos



CONCEPTOS BÁSICOS

- Transición A: un proceso nuevo nunca pasa directamente al estado de ejecución sino que pasa antes por un estado intermedio denominado preparado. Esta transición ocurre cuando se le asignan los recursos necesarios para su ejecución.
- Transición B: ocurre cuando un proceso que tiene asignados los recursos necesarios para ejecutarse recibe el tiempo de CPU para pasar a ejecución.
- Transición C: un proceso que está ejecutándose ya terminado el tiempo asignado por la CPU y tiene que dejar paso a otro proceso.
- Transición D: un proceso que estaba bloqueado por ejemplo porque necesitaba un recurso no disponible pasa a preparado cuando se le asigna dicho recurso.
- Transición E: tiene lugar cuando un proceso que se está ejecutando necesita por ejemplo un recurso que no está disponible para seguir ejecutándose.
- Transición F: un proceso que ha terminado su tiempo de CPU y que además ha terminado de ejecutar todas las instrucciones.

Procesos

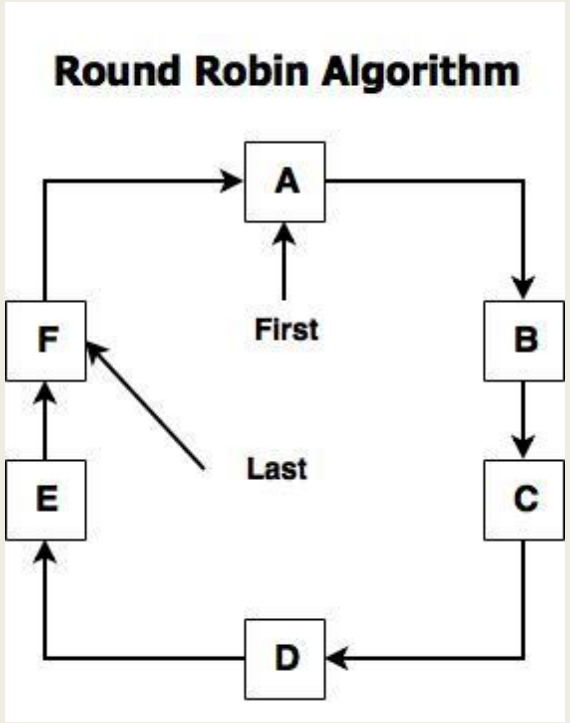
CONCEPTOS BÁSICOS

- Los estados de los procesos están íntimamente relacionados con el concepto de **prioridad** que les asigna el administrador del sistema o el propio sistema operativo a cada proceso.
- Mediante los **algoritmos de planificación** se decide qué procesos deben ejecutarse en cada momento y por qué. Para esto, la CPU se basa en el tipo de algoritmo y la prioridad de cada proceso.

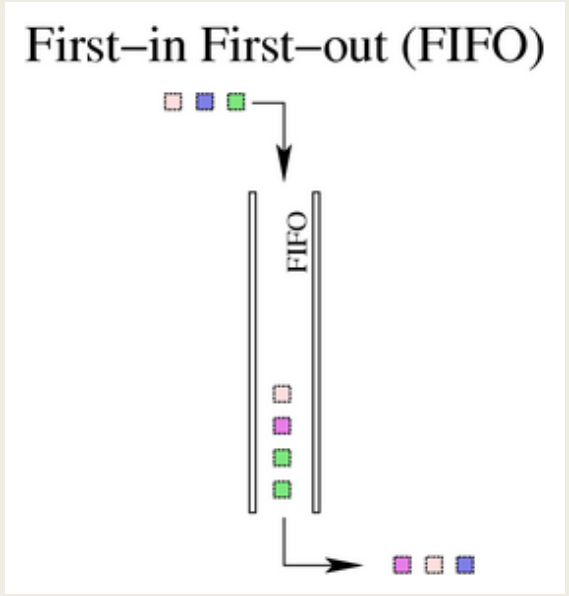
Procesos

ALGORITMOS DE PLANIFICACIÓN

- **Algoritmo de rueda o Round Robin**: este algoritmo asigna rotativamente los ciclos de CPU a los distintos procesos. A cada proceso se le asigna un quantum o intervalo de tiempo de ejecución.



- **Algoritmo FIFO (First In First Out):** este algoritmo asigna los ciclos de CPU a cada proceso en función de una cola FIFO, es decir, el primer proceso en llegar coge el tiempo de la CPU y lo suelta cuando termina completamente.



Procesos

OPERACIONES

El sistema operativo puede crear y eliminar proceso de manera dinámica. Los tipos de operaciones que se pueden realizar son:

- *CREACIÓN*
- *ELIMINACIÓN*
- *INTERACCIÓN*

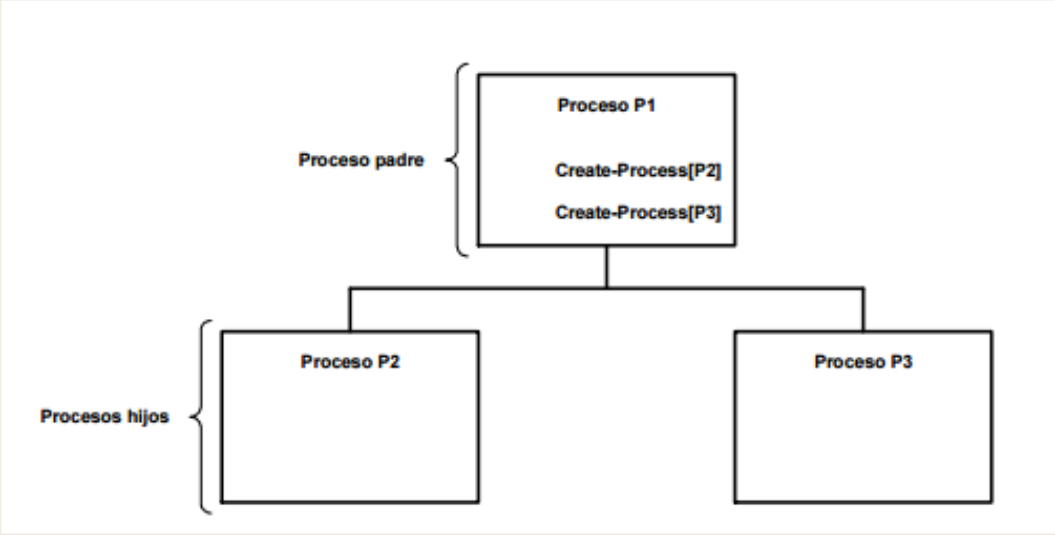
Procesos

OPERACIONES

Creación de procesos

El sistema operativo proporciona un **servicio** para la creación de procesos por parte de otros. Cada proceso que se lanza a ejecución depende, normalmente, de otro proceso denominado proceso ***padre***. El nuevo proceso es llamado proceso ***hijo***.

Debido a este mecanismo de creación de procesos, las relaciones entre los procesos tienen estructura de árbol.



Procesos

OPERACIONES

Eliminación de procesos

Al contrario de lo que ocurre con la creación, un proceso puede terminar de ejecutarse **por sí mismo** o **por otro proceso** que, normalmente, solo puede ser su proceso padre.

- Para que un proceso termine de ejecutarse por sí mismo debe llamar al servicio del sistema **Exit**.
- Para que un proceso padre termina la ejecución de su proceso hijo debe llamar al servicio del sistema **Abort**.

Procesos

OPERACIONES

Interacción entre procesos

Aunque normalmente los procesos **no se comunican entre sí**, en ocasiones es necesario por ejemplo para intercambiar información. Los **mecanismos básicos** de comunicación entre procesos son:

- **Memoria compartida**: mecanismo en el cual la **comunicación entre los procesos** recae en los **procesos** ya que el sistema operativo solo proporciona las llamadas necesarias para manipular la **memoria compartida**.
- **Paso de mensajes**: en este caso la responsabilidad de comunicación entre procesos recae en el **sistema operativo** que se encarga de proporcionar un **enlace lógico entre los procesos**. Los procesos únicamente tienen que invocar a las llamadas **send** y **receive** para comunicarse entre sí.

Procesos

OPERACIONES

Interacción entre procesos

La comunicación entre procesos puede ser:

- **Síncrona**: los procesos tienen que **sincronizarse para intercambiar datos**.
- **Asíncrona**: un proceso que suministra datos **no tiene que esperarse** a que el proceso que los necesita los coja. En este caso se usa un **buffer temporal** de comunicación.

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

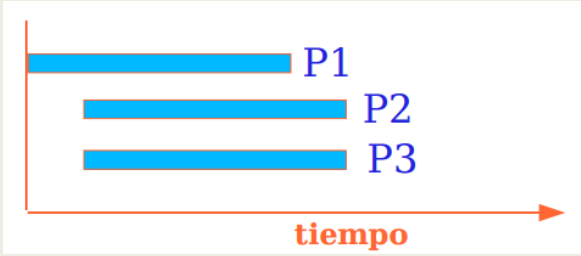
Programación concurrente, paralela y distribuida.

Programación concurrente, paralela y distribuida

Programación concurrente

Dos tareas o procesos son **concurrentes** cuando transcurren durante el mismo intervalo de tiempo.

La **programación concurrente**, es el conjunto de técnicas que permiten el paralelismo potencial en los programas además de resolver problemas de comunicación y sincronización.



En la programación concurrente **varios procesos** trabajan en la **resolución de un problema**. Una de las principales características de este paradigma es la velocidad de ejecución fruto de **dividir un programa en procesos** y del reparto de estos. Además, y a diferencia de la programación secuencial, **el orden de ejecución de los programas es indeterminado** ante una misma entrada.

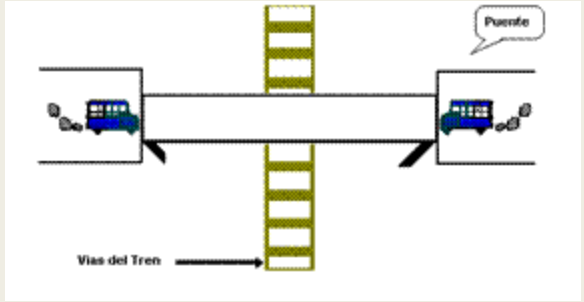
Programación concurrente, paralela y distribuida

Programación concurrente

Una de las características de la programación concurrente es el hecho de que los procesos deben disponer de un mecanismo para comunicarse entre sí, ya sea mediante memoria compartida o paso de mensajes.

Este paradigma presenta ciertos **problemas** a que en ocasiones varios procesos compiten por ejemplo por los mismos recursos:

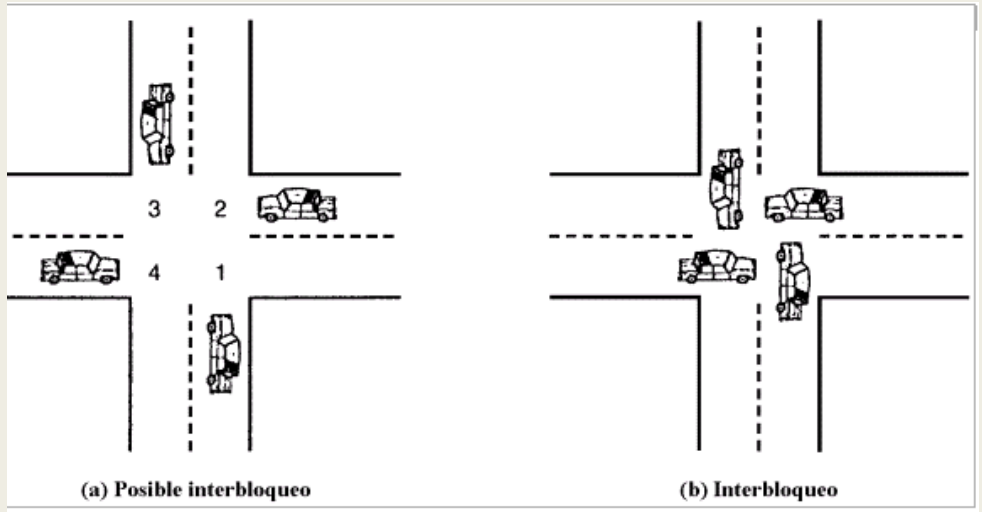
- ***Exclusión mutua.*** *La exclusión mutua es la actividad que realiza el sistema operativo para evitar que dos o más procesos ingresen al mismo tiempo a un área de datos compartidos o accedan a un mismo recurso.*



Programación concurrente, paralela y distribuida

Programación concurrente

- ***Bloqueo mutuo:*** *se da cuando dos procesos necesitan un recurso que tiene el otro proceso.*



La programación concurrente está íntimamente relacionada con la programación paralela pero enfatiza más en la interacción entre tareas o procesos. Digamos que la programación concurrente es un concepto **más general** que el paralelismo.

Programación concurrente, paralela y distribuida

Programación paralela

La **programación paralela** es un paradigma de programación que consiste en la explotación de **varios procesadores** para que trabajen de manera conjunta y simultánea en la resolución de un problema.

Normalmente, el funcionamiento se basa en que **cada procesador** ejecuta **una parte del problema** y realizan un **intercambio de datos** entre ellos según se necesita. Al contrario que con la programación concurrente, este paradigma enfatiza en la **simultaneidad de las tareas**.

Programación concurrente, paralela y distribuida

Programación paralela

Este tipo de programación surge ante los inconvenientes encontrados con la programación secuencial ante problemas que requerían de una mayor velocidad o disponer de más memoria. Sin embargo, presenta ciertas **desventajas**:

- Dificultades de integración de componentes.
- Incremento de la complejidad en el acceso a los datos.
- Incremento de la dificultad en el desarrollo de compiladores y entornos de programación eficientes.
- Dificultades de aprendizaje.

Programación concurrente, paralela y distribuida

Tipos de programación paralela

Existen varios tipos de computación paralela:

- Paralelismo a nivel de bit
- Paralelismo a nivel de instrucción
- Paralelismo de datos
- Paralelismo de tareas

Programación concurrente, paralela y distribuida

Tipos de programación paralela

Una de las clasificaciones de las computadoras más extendida, la **taxonomía de Flynn**, para estudiar donde se encuadraría la programación secuencial y paralela.

La clasificación de Flynn se basa en el **flujo de instrucciones y de datos** que en ellos se desarrolla.

Según la taxonomía de Flynn disponemos:

- *Modelo SISD*
- *Modelo SIMD*
- *Modelo MISD*
- *Modelo MIMD*

Programación concurrente, paralela y distribuida

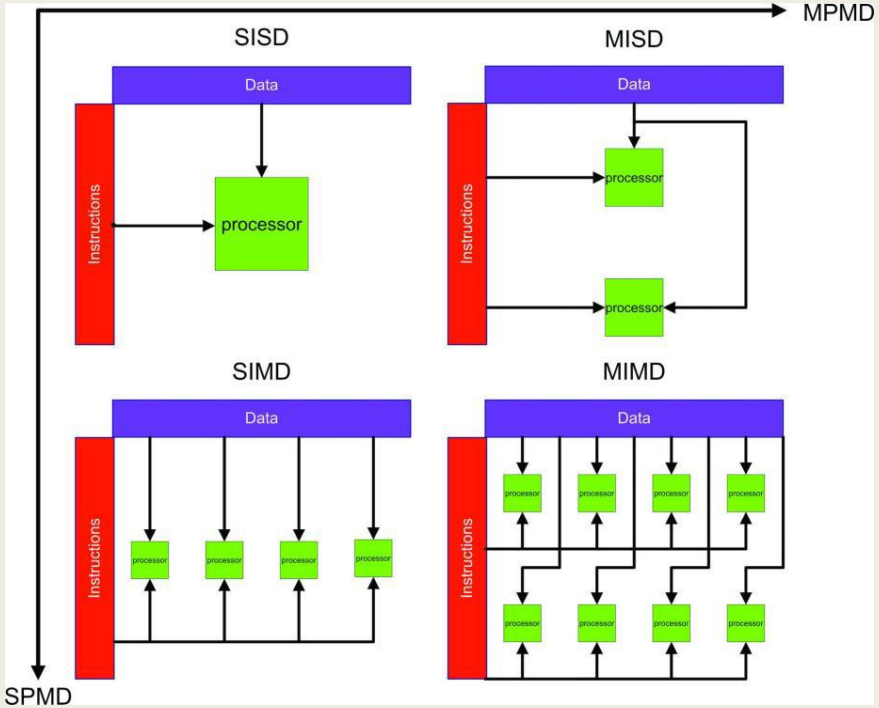
Tipos de programación paralela

Una de las clasificaciones de las computadoras más extendida, la **taxonomía de Flynn**, para estudiar donde se encuadraría la programación secuencial y paralela.

La clasificación de Flynn se basa en el **flujo de instrucciones y de datos** que en ellos se desarrolla.

Según la taxonomía de Flynn disponemos:

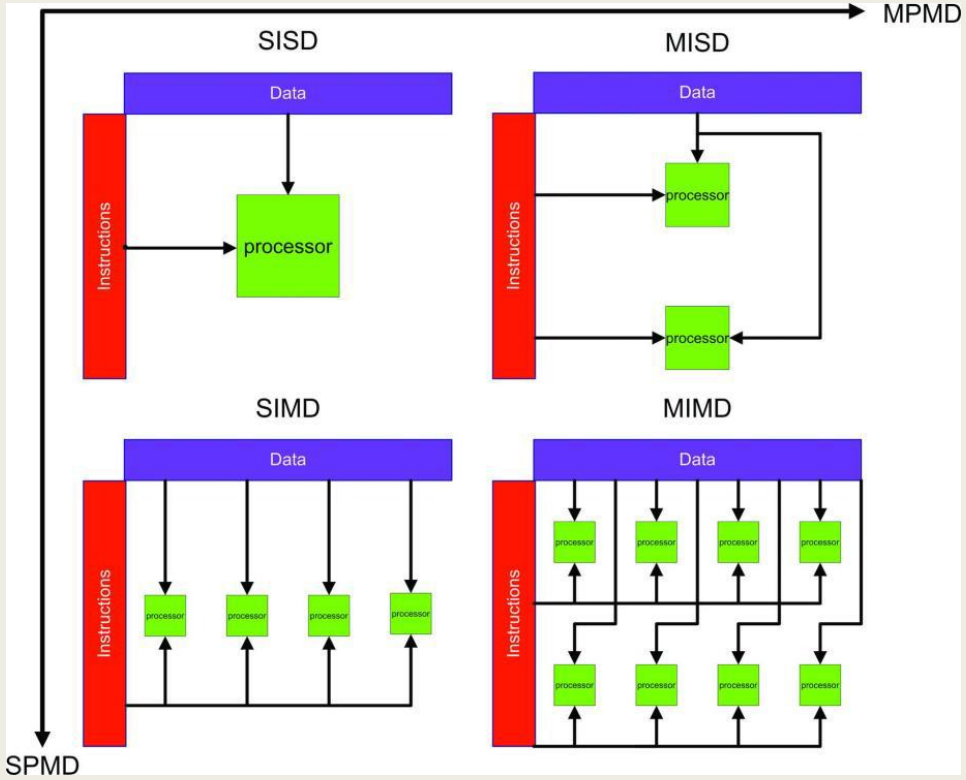
- *Modelo SISD*
- *Modelo SIMD*
- *Modelo MISD*
- *Modelo MIMD*



Programación concurrente, paralela y distribuida

Tipos de programación paralela

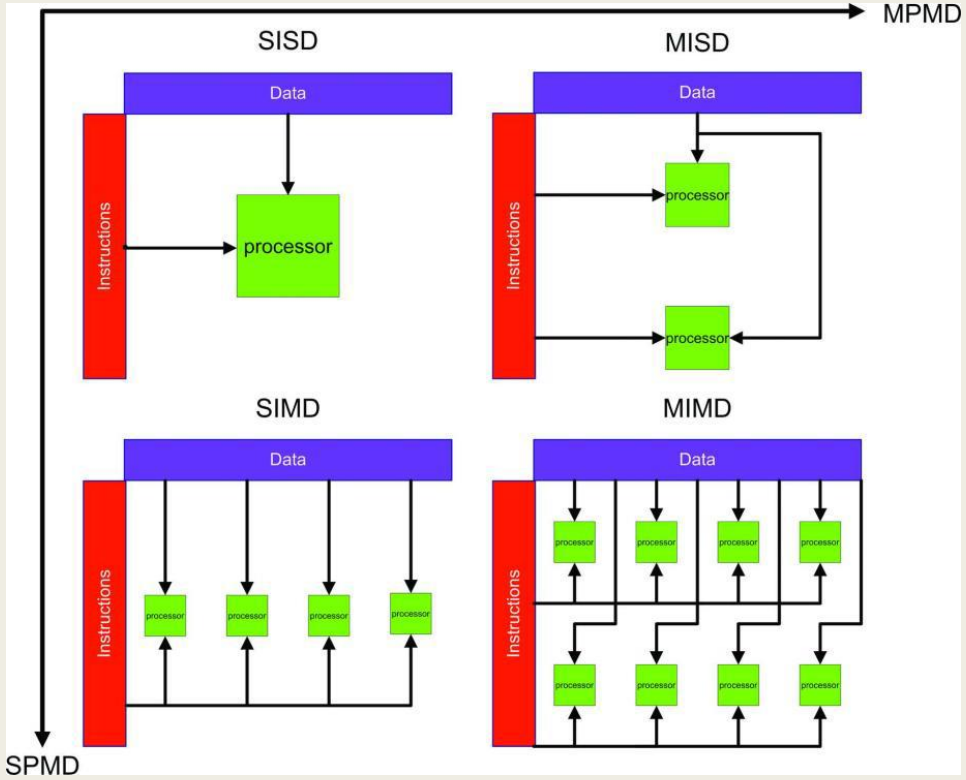
Modelo SISD: este modelo corresponde a la programación secuencial en la que se tiene un único flujo de instrucciones que trabajan sobre un único conjunto de datos por tanto no se explota el paralelismo ni en las instrucciones ni en los datos.



Programación concurrente, paralela y distribuida

Tipos de programación paralela

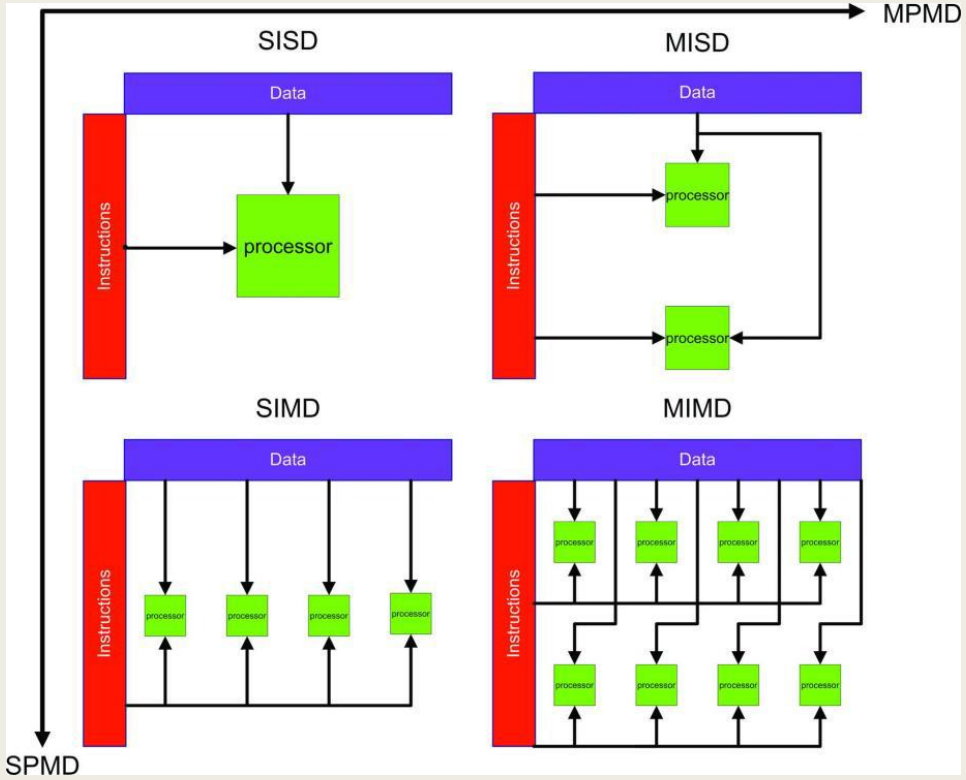
Modelo SIMD: este es el caso de un computador que explota varios flujos de datos dentro de un único flujo de instrucciones para llevar a cabo operaciones que pueden ser paralelizadas de modo natural como por ejemplo un procesador vectorial.



Programación concurrente, paralela y distribuida

Tipos de programación paralela

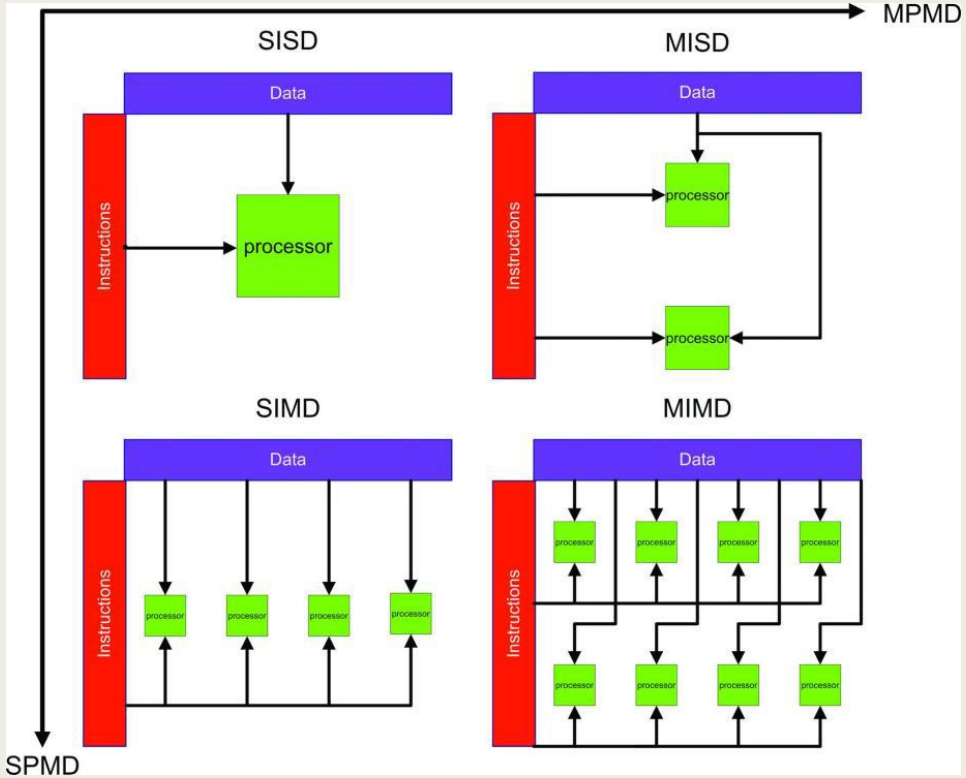
Modelo MISD: modelo que incorpora varios flujos de instrucciones al mismo tiempo trabajando sobre el mismo conjunto de datos.



Programación concurrente, paralela y distribuida

Tipos de programación paralela

Modelo MIMD: modelo utilizado por la mayoría de sistemas paralelos ya que se tienen varias unidades de proceso cada una con un conjunto de datos asociado y ejecutando un único flujo de instrucciones distinto.

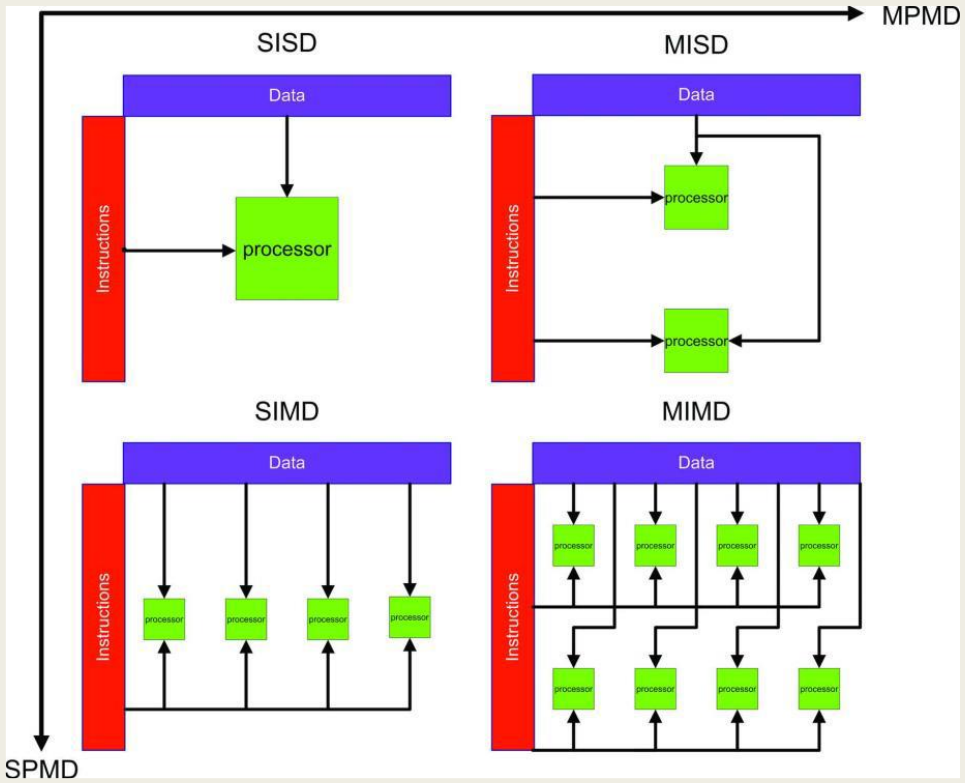


Programación concurrente, paralela y distribuida

Tipos de programación paralela

Podemos deducir que las arquitecturas paralelas pueden ser básicamente del tipo SIMD o MIMD por lo que vamos a ver las diferencias entre una y otra.

- Los sistemas SIMD necesitan **menos hardware** que los sistemas MIMD.
- Los sistemas SIMD necesitan **menos tiempo de inicio para la comunicación entre los procesadores**.
- Los sistemas SIMD **no pueden ejecutar diferentes instrucciones** en el mismo ciclo de reloj.
- Los sistemas MIMD **suelen tener un coste inferior** a los sistemas SIMD.

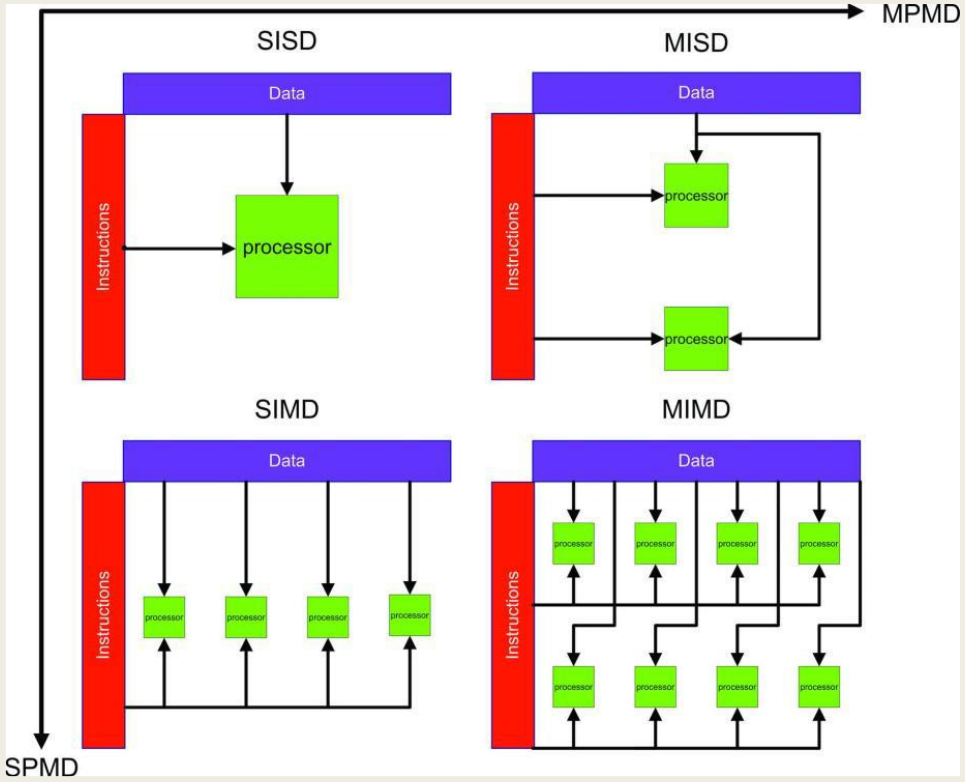


Programación concurrente, paralela y distribuida

Tipos de programación paralela

En general, los sistemas MIMD son **más generales y eficaces** por lo que suelen ser **los más usados** en la computación paralela. Los sistemas MIMD se pueden dividir, atendiendo a cómo se organiza el espacio de direcciones de memoria, en:

- Sistemas de **memoria compartida o multiprocesadores**: estos sistemas **comparten físicamente la memoria**, es decir, todos los procesadores acceden al mismo espacio de direcciones.
- Sistemas de **memoria distribuida o multicomputadores**: en este caso cada procesador cuenta con su **propia memoria** la cual es accesible solo por dicho procesador.



Programación concurrente, paralela y distribuida

Programación distribuida

La programación distribuida es un paradigma de programación para el desarrollo de sistemas distribuidos, abiertos, escalables, transparentes y tolerantes a fallos. Es fruto del uso de las computadoras y las redes.

Se basa en el uso de distintos computadores organizados y repartidos en una infraestructura.

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

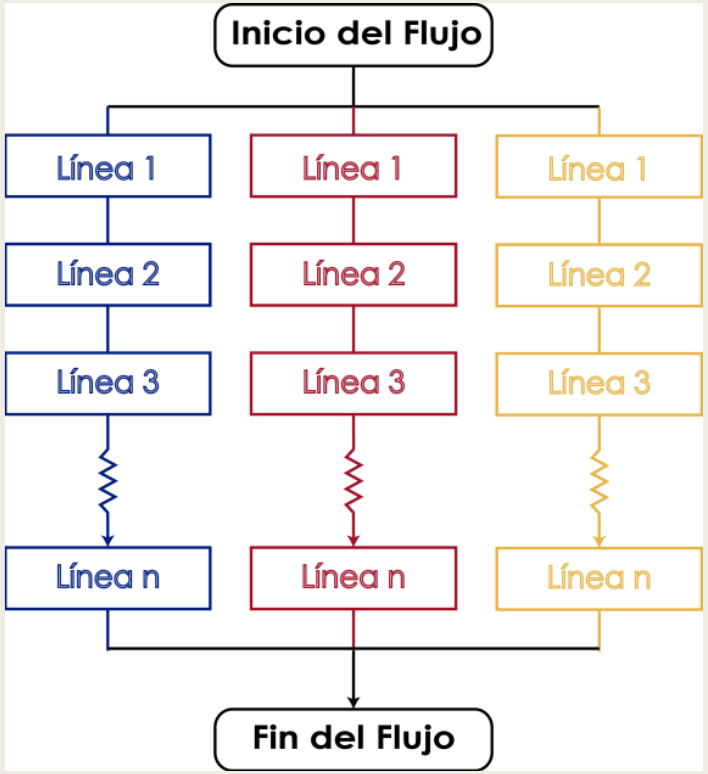
Programación de aplicaciones multiproceso.

Programación de aplicaciones multiproceso

Programación de aplicaciones multiproceso

La principal estructura de programación en los lenguajes de programación actuales es la secuencial donde una instrucción se ejecuta detrás de otra. También cuentan con otras estructuras como las repetitivas o selectivas que permiten modificar este flujo pero siempre respetando el hecho de que las instrucciones se deben ejecutar una tras otra. Sin embargo un usuario puede necesitar que **dos o más tareas se lleven a cabo de manera simultánea.**

Esto sería imposible si no fuera por la existencia de los **hilos** o **threads** que permiten una ejecución lo más cercana a la ejecución simultánea.



Depuración de aplicaciones multiproceso

Depuración de aplicaciones multiproceso

Conocemos como **depuración del software** al proceso de identificar y corregir defectos que pueda tener el programa que estamos desarrollando. Mientras programamos son muchos los errores humanos que podemos cometer y éstos evidentemente aumentan considerablemente con la complejidad del problema.

La mayoría de compiladores tienen la posibilidad de ejecutar un programa paso por paso, dando así la posibilidad al desarrollador de ir comprobando los valores intermedios de las variables, posibles salidas, etc.

Por tanto, los depuradores deben ser usados para realizar un seguimiento sobre el comportamiento dinámico del software.

Depuración de aplicaciones multiproceso

Depuración de aplicaciones multiproceso

Normalmente se utilizan cuando se encuentra un error para averiguar la causa de ese error e intentar corregirlo. Es por esto que antes de acudir a la depuración es necesario intentar delimitar al máximo la causa del error, y las posibles estrategias para solucionarlo.

Además debemos analizar cada error por separado como un proceso individualizado. Y cuidado porque al corregir un error puede que surjan otros por lo que es recomendable volver a realizar las pruebas necesarias y si se encuentran más errores volver a depurar.