UNIDAD 1: PROGRAMACIÓN MULTIPROCESO

Módulo Profesional:
Programación de Servicios y Procesos

Índice

RESUMEN INTRODUCTORIO

En esta unidad vamos a introducir todos los conceptos necesarios para la comprensión de los sistemas multiproceso.

Por lo tanto, ahondaremos en la gestión de los procesos ejecutados en paralelo: como la concurrencia, por un lado, hace mucho más eficiente nuestros sistemas, pero como, por otro, debemos solventar inconvenientes como la sincronización y el uso de recursos compartidos.

También estudiaremos los principales conceptos sobre hilos, sus características compartidas con los procesos y sus diferencias, lo cual lo trabajaremos con más profundidad en unidades posteriores.

Por último, introduciremos los conceptos y ejemplos sobre programación concurrente, paralela y distribuida, conceptos necesarios para comenzar con la programación de aplicaciones multiproceso, las problemáticas que nos encontramos y algunas soluciones para ejecutar procesos de forma concurrente.

INTRODUCCIÓN

Debido a la necesidad de optimizar el uso de los recursos del sistema se han desarrollado técnicas complejas que permiten explotar al máximo el uso de los componentes de un sistema informático.

Actualmente los computadores secuenciales se basan en la arquitectura de Von Neumann, la cual presenta dos grandes desventajas: la velocidad de ejecución de las instrucciones y la velocidad de transferencia de la información entre la memoria y la CPU. Para solventar estas dificultades surgió la programación concurrente cuya evolución ha dado lugar a la programación paralela y distribuida.

Hoy en día los ordenadores, principalmente las CPUs, y en consonancia los sistemas operativos, implementan mecanismos y políticas para la gestión de múltiples programas. Como programadores y desarrolladores, es necesario conocer las diferentes técnicas de programación multiproceso para, por lo tanto, obtener el mayor rendimiento de nuestras aplicaciones.

CASO INTRODUCTORIO

Como desarrolladores nos encontramos inmersos en el desarrollo de una aplicación de escritorio, usando el lenguaje de programación Java, que se comunicará con otros sistemas software a través de una red de computadores.

Para ello, necesitamos conocer cómo identificar los procesos y la información relevante de los distintos sistemas de software involucrados en el desarrollo de su aplicación, la forma en que el sistema gestiona estos procesos y los distintos estados por los que estos pueden pasar. De la misma forma, necesitaremos conocer cómo se gestionan los recursos del sistema que ejecutará la aplicación que se está desarrollando para prevenir situaciones de error y conseguir desarrollar un producto que tenga un buen rendimiento.

¿Debemos pensar en una programación multiprogramada? ¿El lenguaje usado tiene librerías para realizar esa programación multiproceso?

Al finalizar la unidad tendremos los recursos y habilidades para responder a esas preguntas.

1. SISTEMAS MULTITAREA

El primer paso es conocer toda la terminología y conceptos sobre la programación multiproceso y también los diferentes modelos que nos encontramos en los sistemas para desarrollar una aplicación multiproceso.

Nos planteamos preguntas como: ¿Es importante conocer el sistema operativo en el que vamos a programar nuestra aplicación? ¿Qué arquitectura y metodología nos encontramos en la programación multiplataforma?

A partir de estas cuestiones estableceremos el entorno de trabajo y las metodologías adecuadas para la programación multiproceso de nuestra aplicación.

En la medida que los ordenadores incrementaron sus capacidades y, por lo tanto las de procesar información, pasaron de ejecutar un único programa para poder ejecutar un grupo o lote de programas de forma secuencial. Es evidente que, aunque este paso fue importante, la ejecución secuencial hace que un programa tenga que esperar a otro en el caso de que se necesiten recursos como un disco duro o datos de la red de ordenadores.

Una mejor opción a la de esperar sin realizar nada es la de planificar la programación para que se realice otras acciones, deteniendo las acciones que esperaban esa operación bloqueante y después recuperándola en el momento que se quedó.

Los equipos continuaron evolucionando y creciendo, y a su vez su capacidad de ejecución, llegando a la conclusión de que no solo un programa podía realizar varias tareas a la vez, sino que podían ser varios programas los que se ejecutaran al mismo tiempo, de esta forma aparecen los primeros **sistemas multiprogramados.**

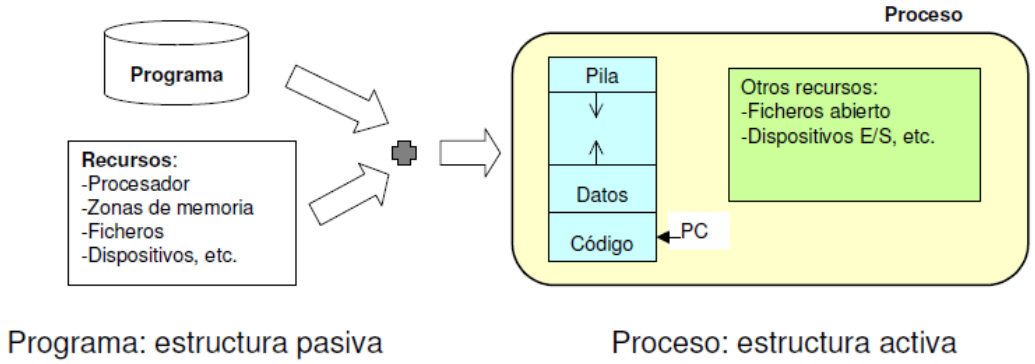
1.1 Programas, ejecutables, procesos y servicios

Es importante realizar una definición inicial de todos los conceptos y nomenclatura que usaremos a lo largo de todo el módulo.

- Un **programa** podemos definir que es la implementación de un algoritmo donde se usa un lenguaje de programación y que pueda ser entendido por un ordenador. A partir de esta definición:

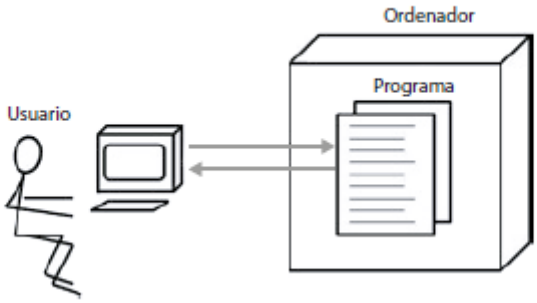
- Un programa ejecutable o simplemente un **ejecutable** es aquel que puede ser interpretado o ejecutado directamente en el ordenador. Podríamos distinguir también tipos de equipos y/u ordenadores, y en ese caso podríamos distinguir en qué capa se ejecuta, si por el sistema operativo, si por la BIOS o remotamente.
- Los **programas concurrentes** son aquellos que pueden ser iniciados por el ordenador y estar ejecutándose de forma simultánea, pero en estados diferentes.
- Los **programas del sistema** son el conjunto de programas que sirven de base a otros programas para actividades fundamentales como, por ejemplo, la de interacción con el hardware.

- Sobre los **procesos e hilos** hablaremos largo y tendido a lo largo del módulo. Como vemos en la siguiente imagen, un proceso podemos definirlo como la instancia de un programa en ejecución.



Concepto de proceso
Fuente: http://mermaja.act.uji.es/docencia/ii22/teoria/TraspasTema2.pdf

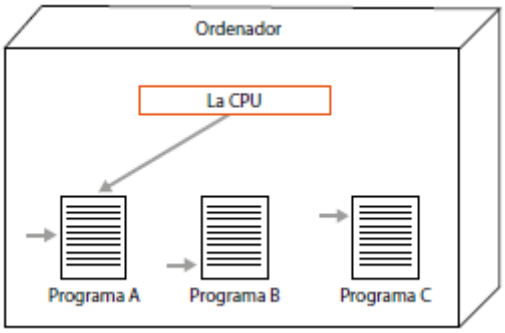
- La última definición es la de **servicio** y cuya diferencia entre programa o proceso de sistema es mínima. Un servicio de hecho es un proceso que es ejecutado y controlado por el sistema operativo y que se mantiene ejecutado en *backend* para que pueda ser utilizado por cualquier otro proceso. Normalmente, suele permitir peticiones a través de llamadas a un puerto mediante sockets, por lo que suele tener una arquitectura cliente/servidor. Un ejemplo de servicio podría ser un servidor de correo que se encuentra ejecutado en segundo plano y que permite manejar las transacciones sobre correo electrónico.



Sistema de Procesamiento Interactivo

Fuente: Sistemas operativos, Módulo de Formación Básica en Grado en Ingeniería Informática de la VIU

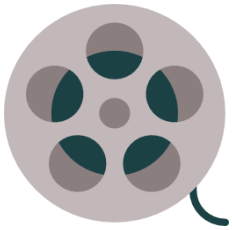
En un sistema multiprogramado, el sistema operativo principalmente puede decidir que, cuando uno de los programas no puede continuar, se sigue con otro programa y, por lo tanto, se tiende a tener el ordenador ocupado la mayor parte del tiempo siendo el ideal de ocupación del 100%.



Sistemas Multiprogramados

Fuente: Sistemas operativos, Módulo de Formación Básica en Grado en Ingeniería Informática de la VIU

Ya podemos imaginarnos que el hecho de que se ejecuten varios programas al mismo tiempo implica que el ordenador (sistema operativo principalmente) debe tener en cuenta y garantizar que cada programa cuenta con los recursos necesarios para poder ser ejecutado. Sino fuera así, un programa que se lanzara a ejecutarse en modo multiproceso podría quedar bloqueado y, por lo tanto, volver a la situación inicial de un programa bloqueando el sistema.



VIDEO DE INTERÉS

En el siguiente enlace encontrarás un interesante vídeo en el que se explica muy claramente la evolución de los sistemas operativos y su evolución en los sistemas multiproceso:

https://www.youtube.com/watch?v=E2fAqDFz9SY

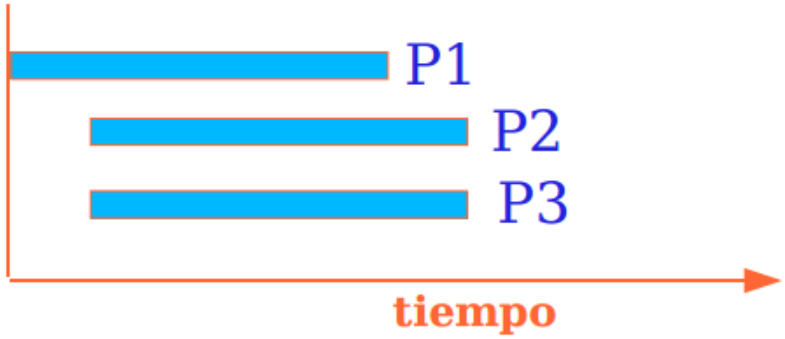
1.2 Programación concurrente, paralela y distribuida

Una vez vistos que los ordenadores y sistemas operativos han ido evolucionando a lo largo de la historia hacia modelos multitareas, multiprocesos y sistemas paralelos, concurrentes y distribuidos, es necesario que nos planteemos como estos cambios tecnológicos afectan al modelo de programación de aplicaciones.

Es evidente que los lenguajes de programación deben seguir las evoluciones tecnológicas y de esta forma aparecen nuevas librerías e incluso nuevos lenguajes de programación. Quizá el lenguaje más de moda dentro de la programación asíncrona y distribuida pueda ser JS, pero tal y como veremos, todos los lenguajes han desarrollado funcionalidades para estos nuevos paradigmas que vamos a estudiar: programación concurrente, paralela y distribuida.

1.2.1 Programación concurrente

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



Programación concurrente
Fuente: Elaboración propia

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 esta programación 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.

Como comentábamos anteriormente, 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.

Aunque todas las características mencionadas arriba son ventajas que hay que tener en cuenta, este paradigma presenta ciertos problemas como que, en ocasiones, varios procesos compiten por los mismos recursos:

- Exclusión mutua.
- 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.

1.2.2 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 realiza 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.

Este tipo de programación surge ante los inconvenientes encontrados con la programación secuencial debido a 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.

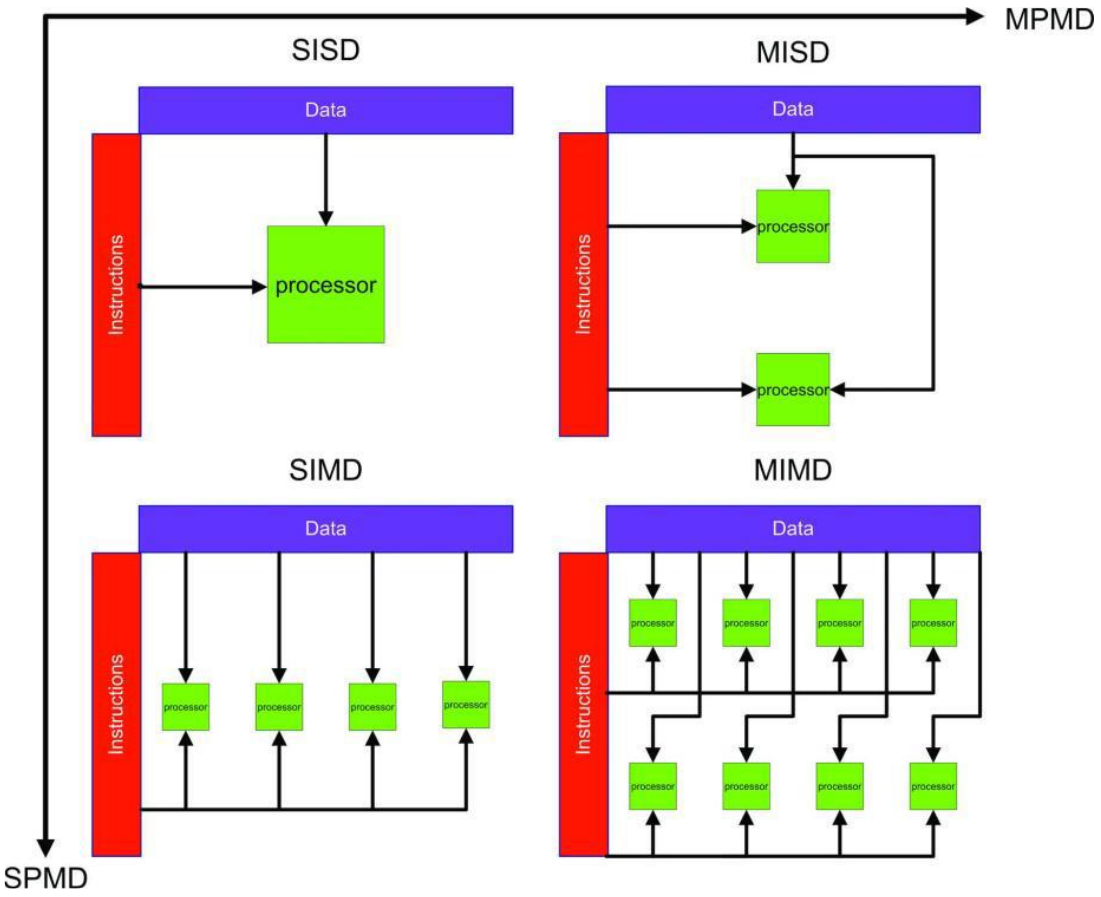
Existen varios tipos de computación paralela:

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

A continuación, vamos a analizar 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: 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.
- 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.
- Modelo MISD: modelo que incorpora varios flujos de instrucciones al mismo tiempo trabajando sobre el mismo conjunto de datos.
- 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.



Modelos programación paralela
Fuente: Elaboración propia

Según la clasificación anterior, 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.

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 multicomputadoras: en este caso cada procesador cuenta con su propia memoria la cual es accesible solo por dicho procesador.

ENLACE DE INTERÉS

En el siguiente enlace encontrarás información ampliada sobre las arquitecturas paralelas:

https://eva.fing.edu.uy/pluginfile.php/192701/mod_resource/content/4/Tema2-2021.pdf

COMPRUEBA LO QUE SABES

Acabamos de estudiar diferentes modelos dentro de la programación paralela. ¿Serías capaz de poner un ejemplo lo más real posible en el que se tenga que aplicar programación paralela? ¿Qué modelo usarías?

Coméntalo en el foro.

VIDEO DE INTERÉS

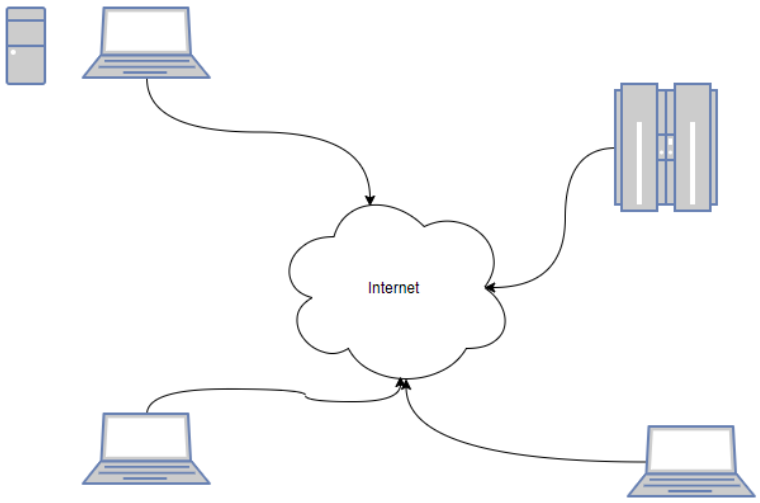
En este vídeo tenemos un interesante análisis entre programación concurrente y paralela:

https://www.youtube.com/watch?v=kMr3mF71Kp4

1.2.3 Programación distribuida

Un tipo especial de programación paralela es la denominada distribuida. Este tipo de programación depende también de dónde se realice la programación y la ejecución, ya que se produce cuando estos sistemas son distribuidos a su vez.

Podemos definir como sistema distribuido aquel donde los ordenadores están en lugares diferentes unidos a través de redes. Un ejemplo podría ser los cajeros de los bancos o los bancos que tienen oficinas a lo largo del mundo.



Arquitectura distribuida
Fuente: Elaboración propia

A partir de esta arquitectura, la programación distribuida es un tipo de programación concurrente donde los diferentes procesos van a ser ejecutados en la red de una forma distribuida.

Una de las grandes diferencias con los sistemas paralelos y concurrentes es que los recursos compartidos como la memoria no existen. Se crean copias

de forma distribuida, por ejemplo, de esta memoria y datos y, por lo tanto, se hace necesario controlar la coherencia de las múltiples copias de la información. Uno de los mecanismos que veremos que se utiliza para intercambiar información será el de comunicación distribuida, por los motivos que acabamos de explicar.

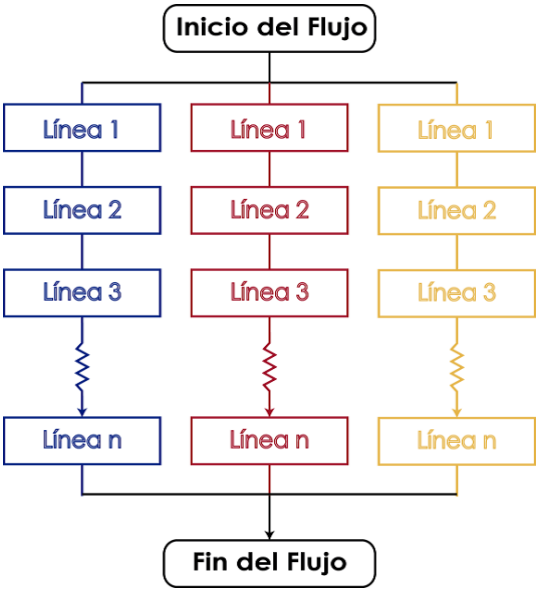
ARTÍCULO DE INTERÉS

Para conocer las ventajas y desventajas del uso de la programación distribuida visita:

http://teoriapa1112.blogspot.com.es/2011/10/ventajas-e-inconvenientes-de-la.html

1.3 Programación de aplicaciones multiproceso

Típicamente en los lenguajes de programación actuales las sentencias de un programa se ejecutan en forma secuencial, es decir, se ejecutan una tras otra. Estos lenguajes también cuentan con otras estructuras como las repetitivas (bucles) o selectivas (condicionales) que permiten modificar este flujo, pero siempre respetando el hecho de que las instrucciones se deben ejecutar una tras otra. Sin embargo, por las características de un programa o por optimizar el uso de recursos del sistema, se puede necesitar que dos o más tareas se lleven a cabo de manera simultánea, tal y como muestra la siguiente imagen:



Programación aplicaciones

Fuente: Elaboración propia

Esto es posible gracias a la utilización de hilos o *threads* que permiten una ejecución lo más cercana posible a la ejecución simultánea.

Usando Java disponemos de las siguientes opciones para implementar técnicas de programación multiproceso:

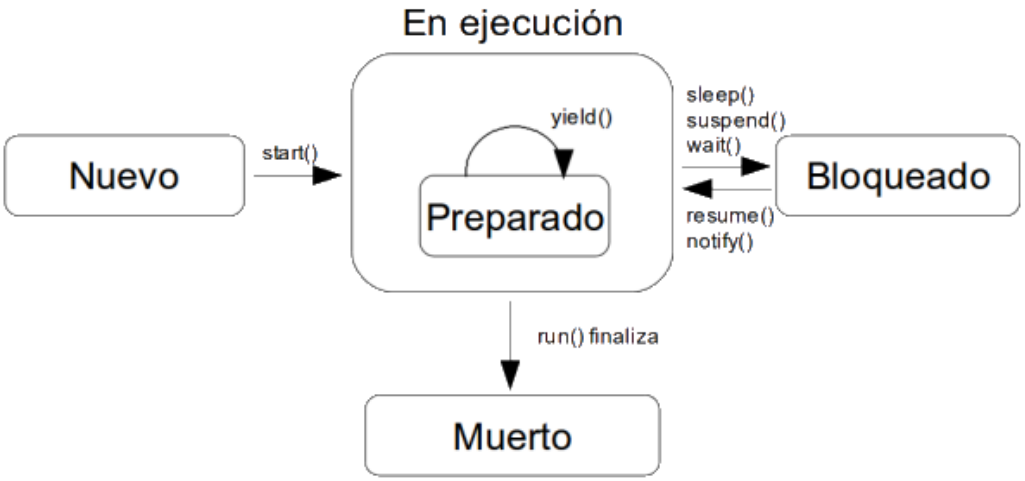
- Implementar la interfaz Runnable.
- Heredar de la clase Thread,

A continuación, se va a estudiar esta última opción.

Para crear una clase que herede de la clase Thread utilizamos la sintaxis:

```
public class NombreClase extends Thread
```

La clase Thread nos ofrece un conjunto de métodos para trabajar con hilos. En la siguiente imagen se observan los estados (que se estudiarán más adelante) de los procesos y los métodos de la clase Thread para pasar de uno a otro:



Estado de procesos con Thread
Fuente: Elaboración propia

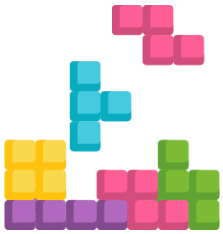
- **Yield()**: para permutar la ejecución de una tarea y la siguiente disponible.
- **Sleep(long)**: para pausar la tarea en curso durante un número de milisegundos indicados por la variable "long".
- **Start()**: para iniciar un proceso o tarea. Llama automáticamente al método run().
- **Run()**: cuerpo de una tarea o hilo.
- **Stop()**: para la ejecución de una tarea y la destruye.
- **Suspend()**: para la ejecución de una tarea sin destruirla.

- **Resume()**: para revivir una tarea que ha sido parada o suspendida.
- **Wait()**: para pausar la ejecución de una tarea hasta recibir una señal.
- **isAlive()**: para conocer el estado de un thread.

COMPRUEBA LO QUE SABES

Acabamos de estudiar los métodos que usamos para pasar de un estado a otro en un proceso. ¿Qué diferencias hay entre stop y suspend? ¿Puedes poner un ejemplo para una aplicación desktop?

Coméntalo en el foro.



EJEMPLO PRÁCTICO

En este ejemplo creamos una clase que denominamos Hilo que hereda de la clase Thread y sobrescribe su método run(). Este método muestra el nombre que le asignamos al objeto Hilo en el momento de su creación y le añade un retardo que pasándolo como parámetro al método sleep() retrasará su ejecución, determinando así qué hilo se ejecutará primero.

En el método principal creamos dos objetos de esta clase Hilo y lanzamos su ejecución con el método start() de la clase thread (de la cual hereda Hilo).

```
public class Hilo extends Thread{

    private String nombre;
    private int retardo;

    public Hilo(String s, int d) {
        nombre = s;
        retardo = d;
    }

    public void run() {
        try {
            sleep(retardo);
        } catch (InterruptedException e) {
        }
        System.out.println(nombre +"-> retardo:"+retardo);
    }
}
```

Creación de hilos

Fuente: Elaboración propia

Tras la ejecución de este programa obtendremos dos líneas de salida en las que se muestran los nombres de los hilos y su retardo (asignado aleatoriamente).

Observa que, para cada ejecución, los retardos varían y el orden de ejecución de los hilos también. El que tenga un retardo menor terminará su ejecución primero y el que tenga un retardo mayor terminará último.

```
public class CreadorHilos {
    public static void main(String[] args) {
        Hilo h1, h2;
        h1 = new Hilo("hilo 1", (int)(Math.random() * 2000));
        h2 = new Hilo("hilo 2", (int)(Math.random() * 2000));
        h1.start();
        h2.start();
    }
}
```

Creación de hilos
Fuente: Elaboración propia

1.4 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 estos 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.

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 gastar 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.

2. PROCESOS

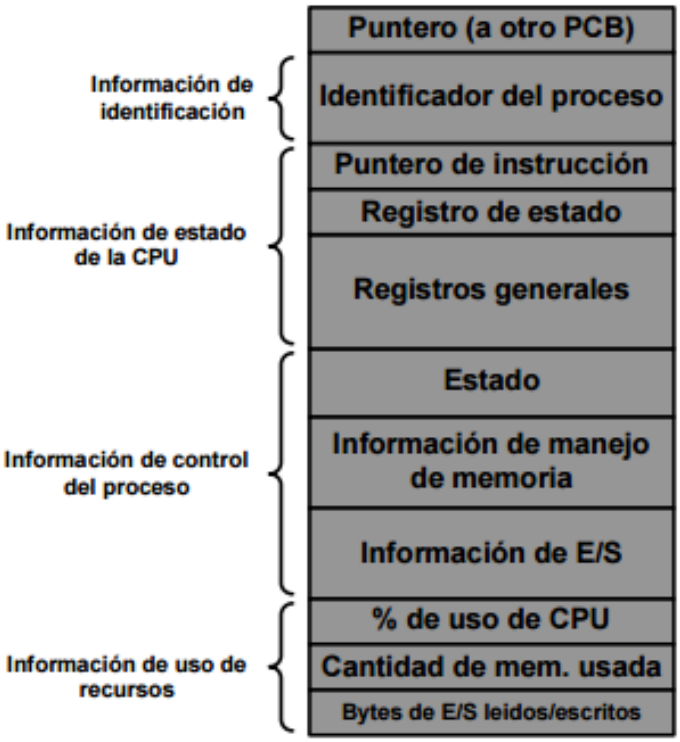
El siguiente paso es conocer que recursos tenemos a nivel de lenguaje de programación para la gestión de una aplicación multiproceso y cuáles son los mejores mecanismos que hay que tener en cuenta para el desarrollo.

Nos planteamos preguntas como: ¿Es importante programar parte o toda la aplicación como multiprocesos? ¿Qué partes de nuestra aplicación es importante programar como un proceso? ¿Qué son los hilos?

A partir de estas cuestiones tendremos una idea más clara de cómo estructurar nuestra aplicación y los mecanismos que se deben tener en cuenta.

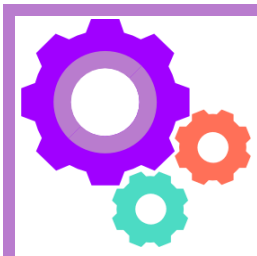
Un proceso es un programa en ejecución bajo el control del sistema operativo, siendo un programa un conjunto de instrucciones que el sistema operativo ejecuta para realizar determinadas acciones de un sistema informático. Los procesos presentan las siguientes características:

- 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 (salvo casos excepcionales).
- Todos los procesos tienen un identificador que lo discrimina de los demás. Este PID (process id) es asignado por el sistema operativo.
- Todos los procesos tienen una estructura de datos denominada bloque de control de proceso (BCP) que almacena información sobre dicho proceso como, por ejemplo: estado, identificador, contador del programa, prioridad, ubicación y recursos utilizados. EL BCP es dependiente del sistema operativo.



Pila de un proceso

Fuente: Elaboración propia



PARA SABER MÁS

Puede ampliar información sobre los procesos visitando el siguiente enlace:

http://www.dc.fi.udc.es/~so-grado/2_PROCESOS.pdf

SABÍAS QUE...

Los BCP de los distintos procesos del sistema son almacenados por el sistema operativo de forma conjunta en una estructura denominada tabla de procesos.

VIDEO DE INTERÉS

En este vídeo puedes profundizar de una forma más visual sobre los conceptos de tabla y bloque de control de procesos:

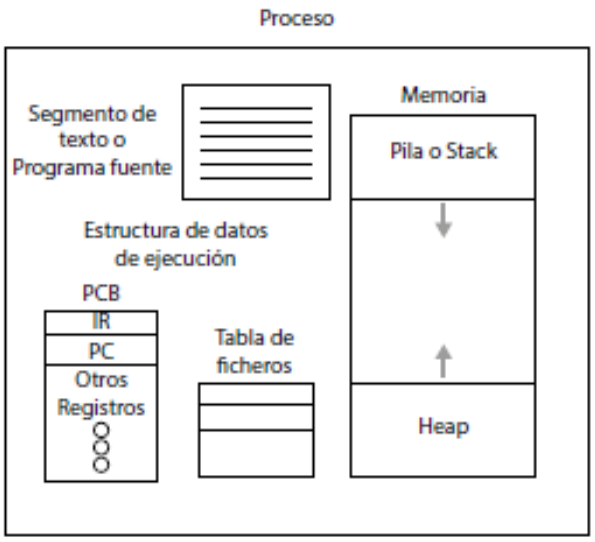
https://www.youtube.com/watch?v=ewovrPs_PNw

2.1 Elementos de un proceso

En primer lugar, sería interesante conocer todo lo que necesita un programa para poderse ejecutar en un ordenador. Un programa está compuesto por instrucciones que es lo que se denomina programa ejecutable y que, para que el ordenador pueda entenderlo y ejecutarlo, deben estar escritas estas instrucciones en lenguaje máquina. El lenguaje máquina está muy cerca del procesador, pero es un lenguaje muy alejado del lenguaje humano y, por lo tanto, difícil de manejar y mantener. Las instrucciones en lenguaje máquina también se suelen denominar segmento de texto.

En segundo lugar, un programa necesita de espacios de memoria donde se almacenen los datos que se van a usar, uno para las variables y otro para las estructuras de datos. La primera de estas áreas se denomina Pila o Stack y la otra se denomina Heap.

Existen otras zonas y partes dentro de los elementos de un proceso como vemos en la imagen, pero exceden nuestros objetivos.



Elementos de un proceso

Fuente: Sistemas operativos, Módulo de Formación Básica en Grado en Ingeniería Informática de la VIU

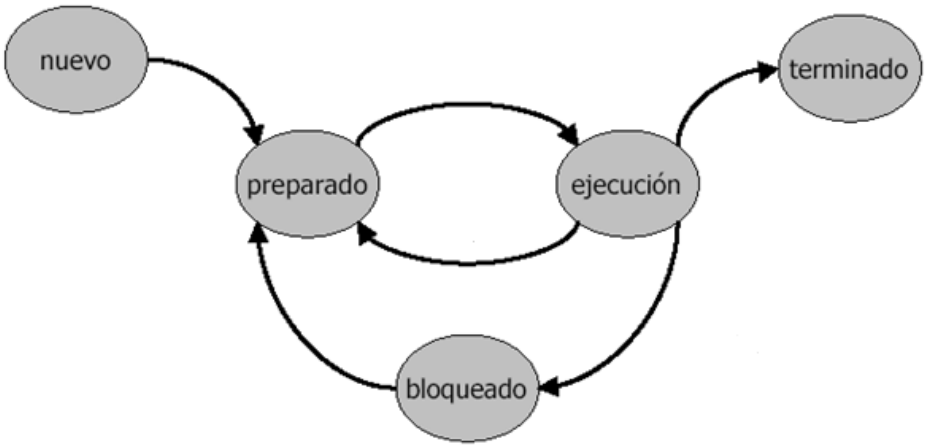
COMPRUEBA LO QUE SABES

Acabamos de introducir los diferentes elementos de un proceso. Dentro de los elementos nos encontramos con los registros de estado. ¿Para qué se usan a nivel de CPU los registros de estado? ¿Puedes buscar y explicar uno de esos registros y su uso?

Coméntalo en el foro.

2.2 Estados y planificación de los procesos

Una vez que un programa se ha ejecutado y se ha convertido en proceso, pasa por distintas fases o estados hasta que finalmente termina de ejecutar todas las instrucciones que le corresponden. Básicamente los estados de un proceso serían los mostrados en la siguiente imagen:



Estados de los procesos
Fuente: Elaboración propia

- 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.

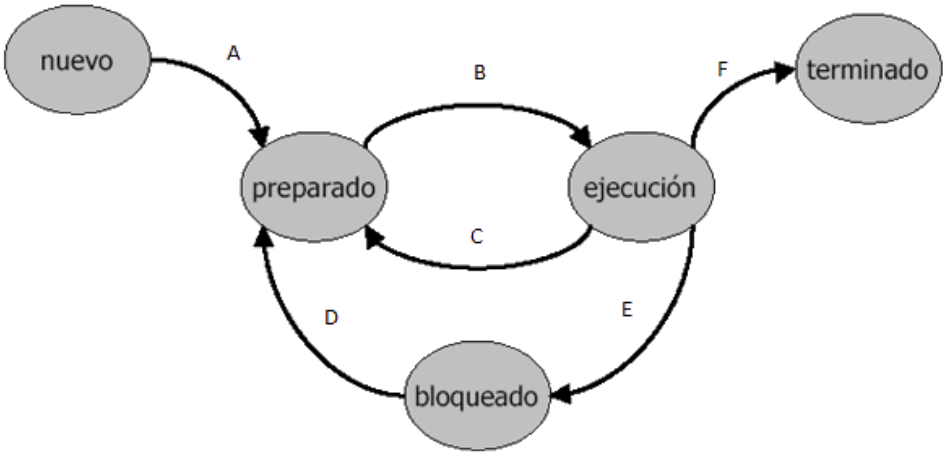
- Terminado: el proceso ha terminado de ejecutar todas las instrucciones.

PARA SABER MÁS

Para ampliar información sobre los procesos, sus estados y transiciones visite:

https://www.fing.edu.uy/inco/cursos/sistoper/recursosTeoricos/5-SO-Teo-Procesos.pdf

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



Estados de los procesos, pasos de un estado a otro
Fuente: Elaboración propia

- 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 ha 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.

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. Algunos de los principales algoritmos de planificación son:

- 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 intervalo de tiempo de ejecución (denominado quantum).
- 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.
- Algoritmo SJF (*Shortest Job First*): debe conocerse el tiempo de ejecución de cada proceso. Se selecciona al proceso en estado "preparado" con el menor tiempo de ejecución. El proceso se ejecutará hasta que haya terminado.
- Algoritmo de prioridad: se selecciona para ejecutar el proceso en estado "preparado" que tenga la máxima prioridad. Si las prioridades se mantienen fijas, algunos procesos podrían no ejecutarse nunca. Para solucionar esto se añade un mecanismo de forma que los procesos que llevan tiempo esperando van adquiriendo mayor prioridad.

ENLACE DE INTERÉS

En el siguiente enlace encontrarás ejemplos de estos y otros algoritmos de planificación:

http://www3.uji.es/~redondo/so/capitulo2_IS11.pdf

VIDEO DE INTERÉS

En este vídeo tienes una explicación detallada de los distintos estados de los procesos y de varios algoritmos de planificación del sistema operativo:

https://www.youtube.com/watch?v=jxGnKR3JoOw

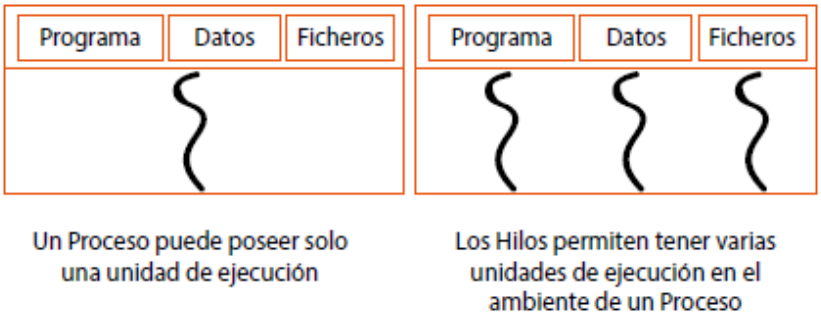
2.3 Hilos. Características y diferencias con los procesos

Los *threads* o hilos en castellano son unidades más pequeñas dentro de un proceso que pueden compartir algunas áreas como recursos, datos o zona de memoria, pero que tienen un entorno de ejecución propio, es decir, que pueden a su vez ser gestionados como los sistemas multiprocesos que estamos estudiando.

Los procesos son denominados entidades pesadas, ya que se encuentran en espacios de direccionamiento independientes, algo que consume bastantes recursos de la CPU. Por otro lado, los hilos al compartir y tener los mismos recursos ya generados para un proceso, la creación y el consumo de la CPU es menor, por lo que se denominan entidades ligeras.

Subdividir un proceso en hilos tiene ventajas y desventajas. En primer lugar, los hilos se pueden considerar procesos ligeros y, por lo tanto, los recursos que necesitan son mucho más ligeros. Además, permiten un mayor control de los programas y un mejor uso de los recursos del ordenador. Por otra parte, introducen una gestión mayor del sistema.

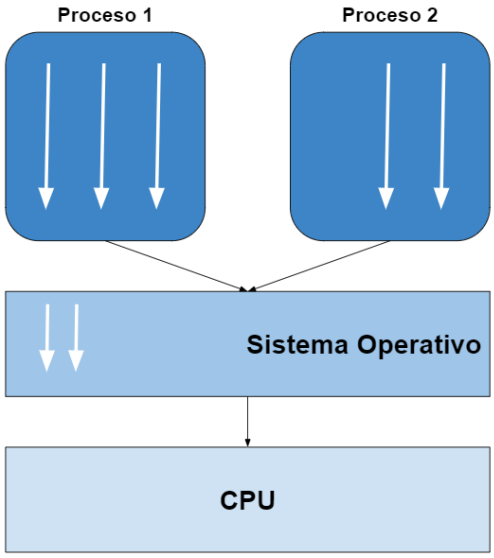
En la siguiente imagen podemos ver un diagrama con las características y diferencias entre un proceso y un hilo:



Procesos e hilos

Fuente: Tanenbaum (2009). *Sistemas operativos modernos*

En programación paralela y distribuida, la utilización de los hilos es uno de los mecanismos más usados, ya que permiten un mejor uso de los ordenadores multinúcleo, tal y como comentábamos. Todos los mecanismos y gestión que veamos para los procesos podrían, por lo tanto, ser aplicados con los hilos. Por ejemplo, en el caso de interbloqueo con recursos, la gestión con hilos aumenta la eficiencia de los procesos y programas, puesto que el hilo bloqueado quedaría a la espera de su ejecución, pero otros hilos continuarían ejecutándose, por lo que el proceso no quedaría bloqueado.



Procesos e hilos
Fuente: Elaboración propia

En la anterior imagen vemos el tratamiento de un ordenador respecto a los hilos y los procesos. Cada hilo se procesa de forma independiente, pertenezca o no al mismo proceso. La diferencia es que dos hilos de un mismo proceso tienen una misma dirección de memoria.

2.4 Gestión de procesos

Veremos qué estados, cómo se denominan y cómo un proceso pasa de un estado a otro. De hecho, dentro del sistema operativo, encontramos un componente que mueve los procesos de un estado a otro y, además de revisar las colas de procesos y de actualizar dichos estados, este componente dentro de un sistema operativo se llama despachador.

Asimismo, en los componentes que tenemos dentro de un sistema operativo, nos encontramos con otro que se encarga de saber qué proceso se debe ejecutar, cuál ejecutar y cuál detener. Este componente se denomina planificador de procesos.

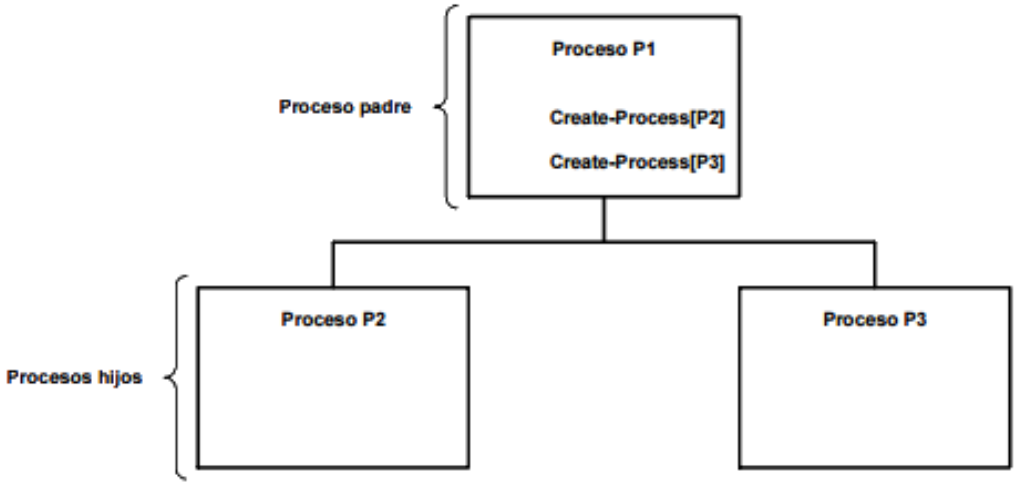
Existen muchas formas de planificar los procesos, su funcionamiento, su ejecución y su prioridad. Para la realización de la gestión, se definen unas **políticas y mecanismos** de planificación de procesos donde:

- Llamamos política a la estrategia que se utiliza para saber cuál es el próximo proceso que va a ser ejecutado.
- Llamamos mecanismo de planificación a la implementación de una determinada política.

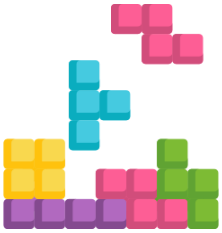
2.4.1 Creación de procesos

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. Veámoslo a través de una imagen:



Relación entre procesos
Fuente: Elaboración propia



EJEMPLO PRÁCTICO

En Java es posible crear procesos utilizando algunas clases que el entorno ofrece para esta tarea. En el ejemplo siguiente podemos ver cómo lanzar un proceso de Acrobat Reader con la clase ProcessBuilder:

```
package creador;
import java.io.IOException;

public class Creador {
    public void ejecutar(String ruta) {
        ProcessBuilder pb;
        try {
            pb = new ProcessBuilder(ruta);
            pb.start();
        } catch (IOException e) {
            System.out.println("Error");
        }
    }

    public static void main(String[] args) {
        String ruta
                = "C:\\Program Files (x86)\\Adobe\\Acrobat Reader DC\\Reader\\AcroRd32.exe";
        Creador cr = new Creador();
        cr.ejecutar(ruta);
        System.out.println("Finalizado");
    }
}
```

Uso de ProcessBuilder
Fuente: Elaboración propia

2.4.2 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.

2.4.3 Interacción entre procesos

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.

La comunicación entre procesos puede ser:

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

COMPRUEBA LO QUE SABES

Acabamos de ver que, dentro de la interacción entre procesos, esta puede ser asíncrona. ¿Puedes poner un ejemplo de esta interacción asíncrona? ¿El lenguaje Java tiene mecanismos para la programación asíncrona como en JS con las promesas?

Coméntalo en el foro.

2.4.4 Concurrencia y condiciones de sincronización

En un ordenador vamos a encontrarnos con diferentes procesos que se ejecuten de forma simultánea, es lo que se denomina procesos concurrentes. La concurrencia de los procesos puede pensarse en un primer momento como un funcionamiento óptimo de los ordenadores, ya que de esta forma se aprovecha mejor las capacidades y los recursos del mismo, pero, por otro lado, la concurrencia de los procesos genera inconvenientes que se deben tener en cuenta para una correcta planificación de los procesos.

Algunos de los conceptos que debemos definir en la sincronización de procesos son:

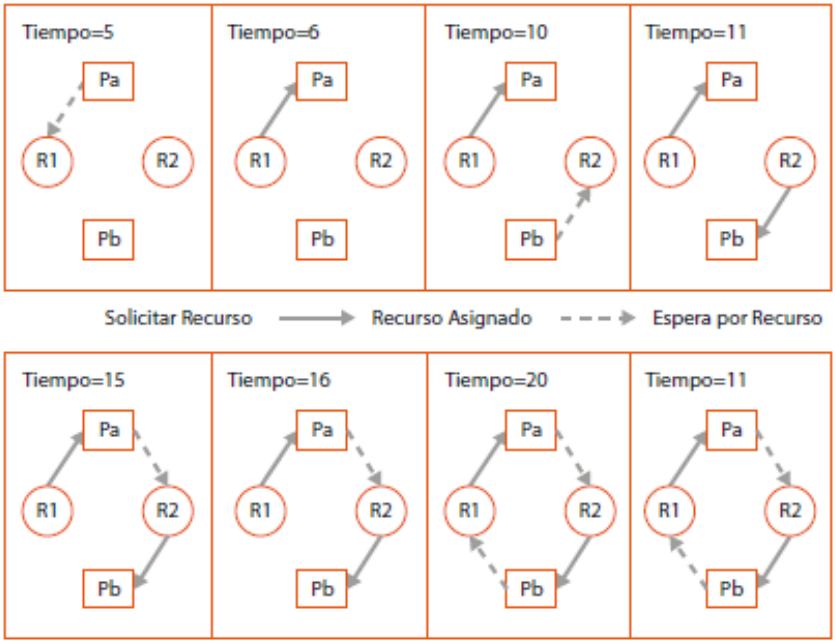
- Operación atómica: es aquella que puede suceder sin que sea interrumpida por el ordenador. Durante la realización de este tipo de operaciones, se marca como imposible interrumpir la operación o bien se inhibe al planificador para poder cambiar el estado del proceso.

- Sección crítica: como la parte del programa en la que se comparten recursos con otros programas o procesos y en los que se debe intervenir cuando hay dos o más procesos concurrentes. Los recursos compartidos pueden ser un área de memoria o un dispositivo, entre otros.

2.4.5 Recursos compartidos y secciones críticas

Como acabamos de definir, las secciones críticas son uno de los problemas más importantes cuando hablamos de programación concurrente, puesto que puede que parte del código de estos procesos o hilos necesite de forma exclusiva acceder a recursos compartidos como ficheros, variables, registros de una base de datos u otros.

La exclusión mutua, o mejor el bloqueo mutuo, es una condición que se puede dar cuando dos procesos necesitan de varios recursos para que se finalicen las operaciones que están llevando a cabo, sin embargo, el sistema operativo ha asignado dichos recursos de tal forma que se entra en un círculo vicioso o un bucle, ya que un proceso necesita un recurso que ha bloqueado otro proceso y viceversa, y a su vez para poderlos liberar, se necesita la finalización de los procesos.



Bloqueo mutuo

Fuente: Sistemas operativos, Módulo de Formación Básica en Grado en Ingeniería Informática de la VIU

De forma resumida, en el anterior ejemplo vemos como dos procesos, Pa y Pb, reservan los recursos R1 y R2 de forma circular, llegando a un momento en el que ambos procesos quedan bloqueados al necesitar del recurso que

está reservado de forma cruzada. A esta condición la llamamos bloqueo mutuo, y a la solución, exclusión mutua.

2.4.6 Problemas. Inanición, interbloqueos

El término inanición hace referencia a una debilidad por falta de un recurso. En el caso de un sistema operativo y siguiendo con el ejemplo que antes hemos presentado, el proceso (Pa*)* necesita un recurso (R1) que no es asignado por el sistema operativo y, por lo tanto, no puede continuar con su ejecución normal.

Puede darse porque haya un problema con el recurso R1, pero en el contexto que estamos trabajando, se produce porque otros procesos lo tienen bloqueado o porque llegan antes a su bloqueo por cuestiones de prioridad.

Una posible solución al problema de exclusión mutua que puede desembocar en los interbloqueos es implementar código que realice el siguiente mecanismo:

- Procedimiento para la entrada a la sección crítica. Este procedimiento debe ser atómico y, por lo tanto, asegurar que la entrada es única.
- Procedimiento para la salida a la sección crítica y, por consiguiente, liberación de los recursos.

```
public void entrada_seccion_critica(){
/*Codigo*/
}
public void salida_seccion_critica(){
/*Codigo*/
}
```

Para que este mecanismo tenga éxito se deben cumplir los siguientes requisitos:

- No puede haber más de un proceso o hilo simultáneamente en la zona de sección crítica.
- No puede existir inanición, pues un proceso no puede esperar un tiempo indefinido para entrar.
- No puede producirse un interbloqueo, ningún proceso fuera de la sección crítica puede impedir que otro proceso entre.

Otra de las posibles soluciones es incorporar como política el concepto de envejecimiento, donde cada vez que un proceso, en nuestro caso Pa, se le deniega el acceso al recurso R1, se le incrementaría la prioridad para poder

tener el recurso, llegando un momento en el que el proceso pueda alcanzar dicho recurso.

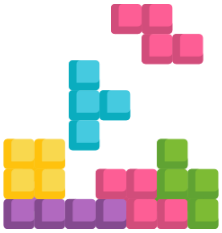
2.4.7 Sincronización entre procesos

Cuando nos encontramos con una situación de concurrencia, es necesario tener una serie de funciones para implementar la gestión y con ello evitar los problemas de la concurrencia de procesos.

Tal y como hemos definido en el apartado anterior, uno de los inconvenientes que debemos gestionar es la gestión de las condiciones de carrera que se puedan presentar al acceder a los recursos compartidos.

Para evitar estar condiciones de carrera bloqueantes, existen mecanismos en los sistemas operativos, y por lo tanto en la programación, que pueden ser utilizados de acuerdo a los programas y procesos, y así, gestionarlos cuando se alcance la ejecución de las regiones críticas.

A los mecanismos que permiten poner de acuerdo diferentes procesos y gestionar positivamente su correcto funcionamiento se denominan sincronización de la ejecución de los procesos.



EJEMPLO PRÁCTICO

A Juan le plantean buscar una solución a la siguiente situación a través de un pequeño programa en Java. En una tienda se ha implementado un contador de entrada de aforo y un contador de salida.
Ambos contadores son independientes, siendo la condición de aforo máximo de 20 personas. ¿Cómo resolver que salga y entre una persona al mismo tiempo a la tienda?

En primer lugar, deberíamos implementar las funciones que implementaría la entrada y salida de personas:

```
int numPersonas = 0; // Variable común

int agregarPersona() {
numPersonas = nupersonas + 1;
}

int restarPersona() {
numPersonas = nupersonas - 1;
}
```

Como se observa, al tener una variable común, denominada numPersonas, si ambas funciones se ejecutan por procesos diferentes y de forma simultánea, puede dar un error. Podemos usar dos funciones que garantizasen el uso atómico de la variable numPersonas:

```
int numPersonas = 0; // Variable común
int np = 0; // Variable para garantizar exclusión mutua

int agregarPersona() {
PedMutex(np);
numPersonas = nupersonas + 1;
LibMutex(np);
}

int restarPersona() {
PedMutex(np);
numPersonas = nupersonas - 1;
LibMutex(np);
} }
```

2.4.8 Compartición de información y mecanismos de comunicación

Comunicarse y sincronizarse son dos de las acciones más importantes cuando se ejecutan procesos concurrentes. Como hemos visto, cuando varios procesos se encuentran ejecutándose en paralelo, podemos planificar políticas y mecanismos que permitirán escoger el orden y la prioridad de ejecución de todos esos procesos.

Los procesos concurrentes los podemos clasificar de acuerdo con la comunicación entre ellos como:

- Procesos independientes, donde no necesitan ni ayuda ni cooperación de otros procesos.
- Procesos cooperantes, que se diseñan para trabajar conjuntamente con otros procesos y, por lo tanto, necesitan interactuar y comunicarse entre ellos.

Para poder sincronizar los procesos se pueden utilizar diferentes técnicas (alguna ya la hemos visto anteriormente):

- Sincronismo condicional: donde un proceso o un hilo que se encuentra en un estado de ejecución pasa a uno de bloqueo hasta que se cumpla una condición.
- Exclusión mutua: que se produce cuando dos o más procesos quieren acceder al mismo recurso.
- Comunicación por mensajes: donde los procesos se intercambian mensajes para comunicarse y sincronizarse. Se puede utilizar tanto en sistemas distribuidos como no distribuidos.

Como acabamos de ver, la solución de comunicación para los problemas de concurrencia es muy apropiada para sistemas distribuidos, ya que en estos sistemas no se comparte la memoria. Usando tareas básicas como el envío y recepción de mensajes, capacidades inherentes en cualquier sistema operativo, podremos implementar una comunicación.

Los tipos de comunicación pueden ser diversos:

- Comunicación directa donde emisor y receptor directamente se intercambian los mensajes y, por consiguiente, implementan todos los mecanismos para el envío y recepción. En la comunicación indirecta, se haría uso de un mecanismo intermedio.
- Comunicación síncrona, donde los procesos se quedan a la espera de la contestación, o asíncrona, donde continúan con otros procesos hasta que llegue la comunicación.

COMPRUEBA LO QUE SABES

Acabamos de ver la comunicación entre procesos. ¿En qué programas tienen sentido la comunicación entre procesos? ¿La arquitectura cliente-servidor sería un tipo de arquitectura de comunicación?

Coméntalo en el foro.

RESUMEN FINAL

En la medida que los ordenadores incrementaron sus capacidades, y por lo tanto la de procesar información, pasaron de ejecutar un único programa para poder ejecutar un grupo o lote de programas de forma secuencial. Los equipos continuaron evolucionando y creciendo y, a su vez, su capacidad de ejecución, llegando a la conclusión de que no solo un programa podía realizar varias tareas a la vez, sino que podían ser varios programas los que se ejecutaran al mismo tiempo. De esta forma aparecen los primeros sistemas multiprogramados.

Es necesario que nos planteemos como estos cambios tecnológicos afectan al modelo de programación de aplicaciones y es evidente que los lenguajes de programación deben seguir las evoluciones tecnológicas y, por ello, aparecen nuevas librerías e incluso nuevos lenguajes de programación.

La programación concurrente hace que aparezcan los procesos e hilos. Un proceso es un programa en ejecución bajo el control del sistema operativo y que podremos programar para que sea ejecutado de forma paralela a otros procesos.

Existen muchas formas de planificar los procesos, su funcionamiento, su ejecución y su prioridad. Para realizar la gestión, se definen unas políticas y mecanismos de planificación de procesos donde:

- Llamamos política a la estrategia que se utiliza para saber cuál es el próximo proceso que va a ser ejecutado.
- Llamamos mecanismo de planificación a la implementación de una determinada política.