

Programación de Servicios y Procesos



PSP UT1

01 Programación Multiproceso

1. Ejecutables, procesos y servicios

1.1 Procesador y núcleo (core).

1.2 Programas y ejecutables.

1.3 Procesos, servicios e hilos.

1.1 Procesador y núcleo (core).

Procesador

El **procesador** es el encargado ejecutar las instrucciones de los programas.

Podríamos llamarlo el cerebro del dispositivo.

Núcleo (core)

El **núcleo** es una unidad con capacidad de ejecución dentro del procesador.

Existen **multitud de modelos de procesador**, tanto como necesidades de capacidad.

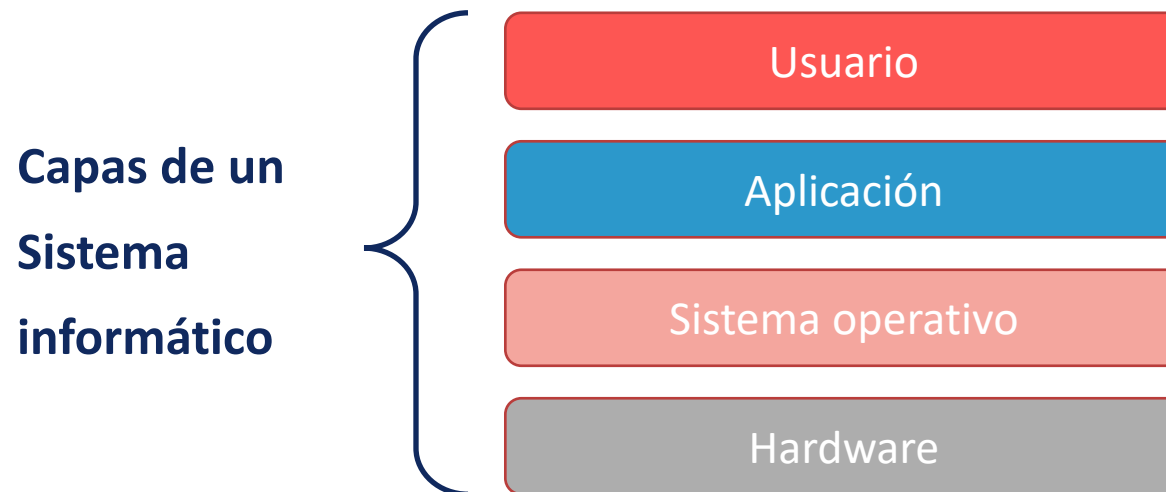
¿En qué se diferencian los procesadores? Si tomamos como ejemplo los procesadores de Intel, Core i5, Core i7 y Core i9, se diferencian, en primer lugar, por el número de cores o núcleos. Los Core i5 vienen con 6 núcleos, los Core i7 vienen con 8 núcleos, y los Core i9 vienen con 10 núcleos.

¿Qué significa que un procesador tenga 2 núcleos? Lo explicamos con un ejemplo. Un Pentium G4560 tiene dos núcleos físicos, lo que significa que puede manejar dos procesos simultáneos (**multitarea**).

1.2 Programas y ejecutables

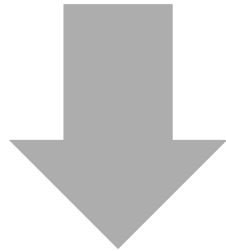
Podemos definir un **programa** como el conjunto de instrucciones que resuelven una tarea específica en un dispositivo. En terminología moderna, se le suele llamar **aplicaciones** y también **apps**.

Un fichero **ejecutable** es aquel que contine la información necesaria para crear un proceso de memoria capaz de ser ejecutado por el procesador.



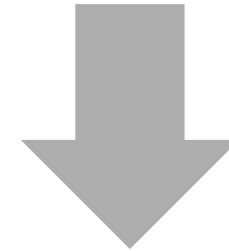
1.2 Programas y ejecutables

Programa



**Conjunto de
instrucciones**

Ejecutable



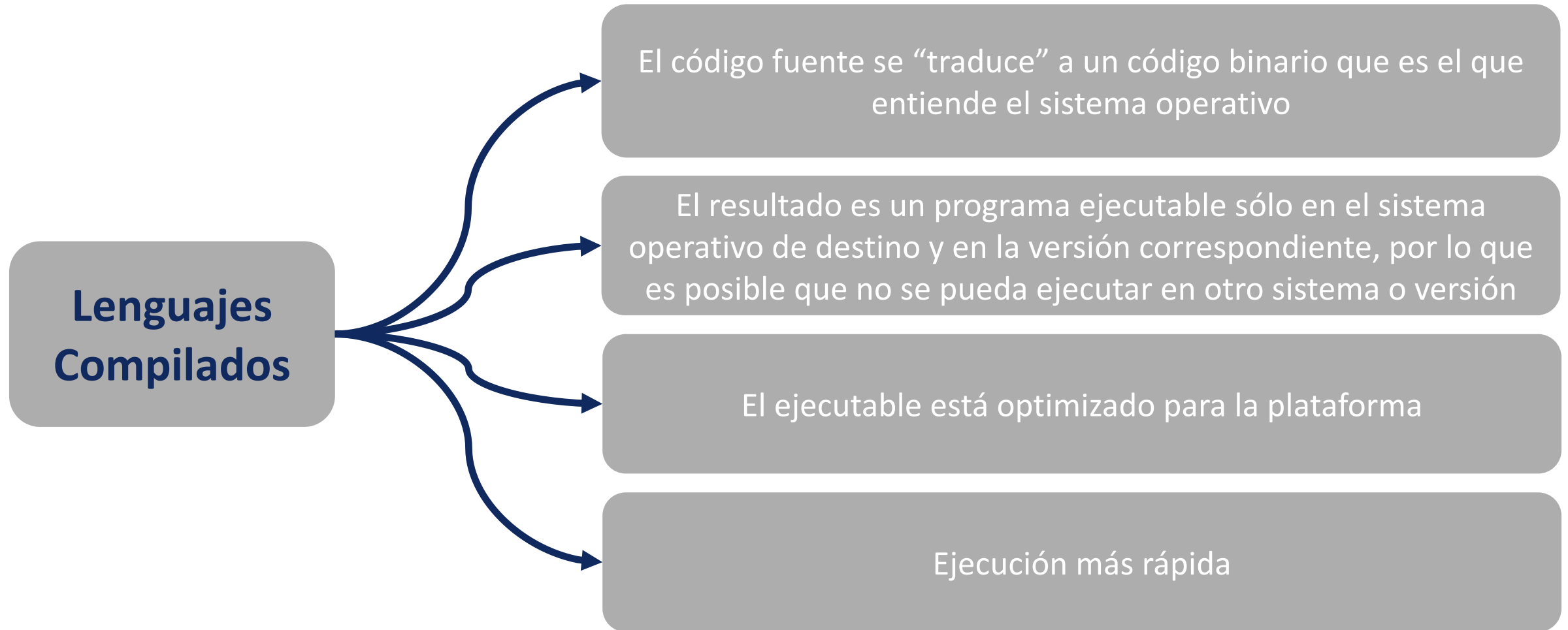
Es un fichero

1.2 Programas y ejecutables

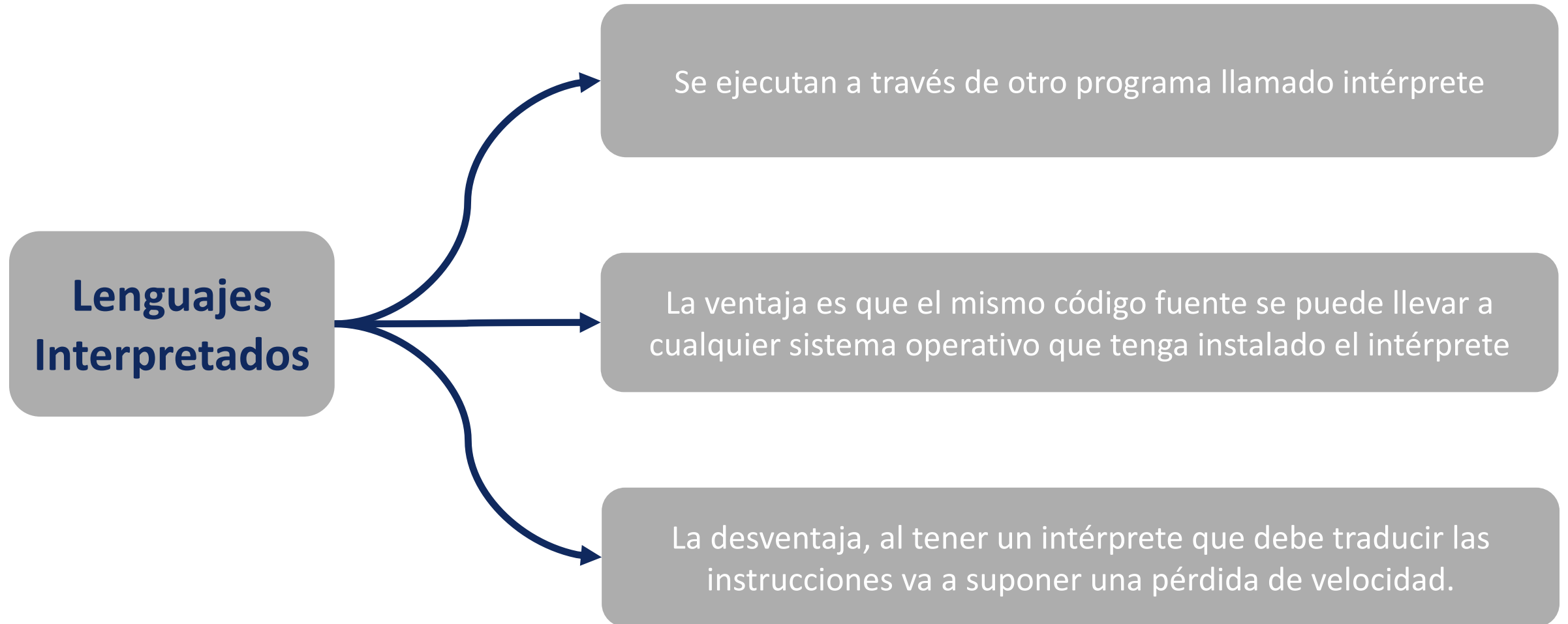
Los programas
se pueden
generar
mediante:

- **Compiladores**, los cuales transforman código de algún lenguaje fuente en un fichero que será ejecutable por un sistema operativo. Por ejemplo, C/C++.
- **Intérpretes**, que son programas que se encargan de transformar cada una de las líneas de un lenguaje de alto nivel en instrucciones máquina y ejecutarlas individualmente. Por ejemplo, PHP.
- **Técnicas híbridas**, que contienen un lenguaje de alto nivel en otro intermedio, que posteriormente se interpretará por una máquina virtual con base en el sistema operativo. Por ejemplo, Java o .net de Microsoft.

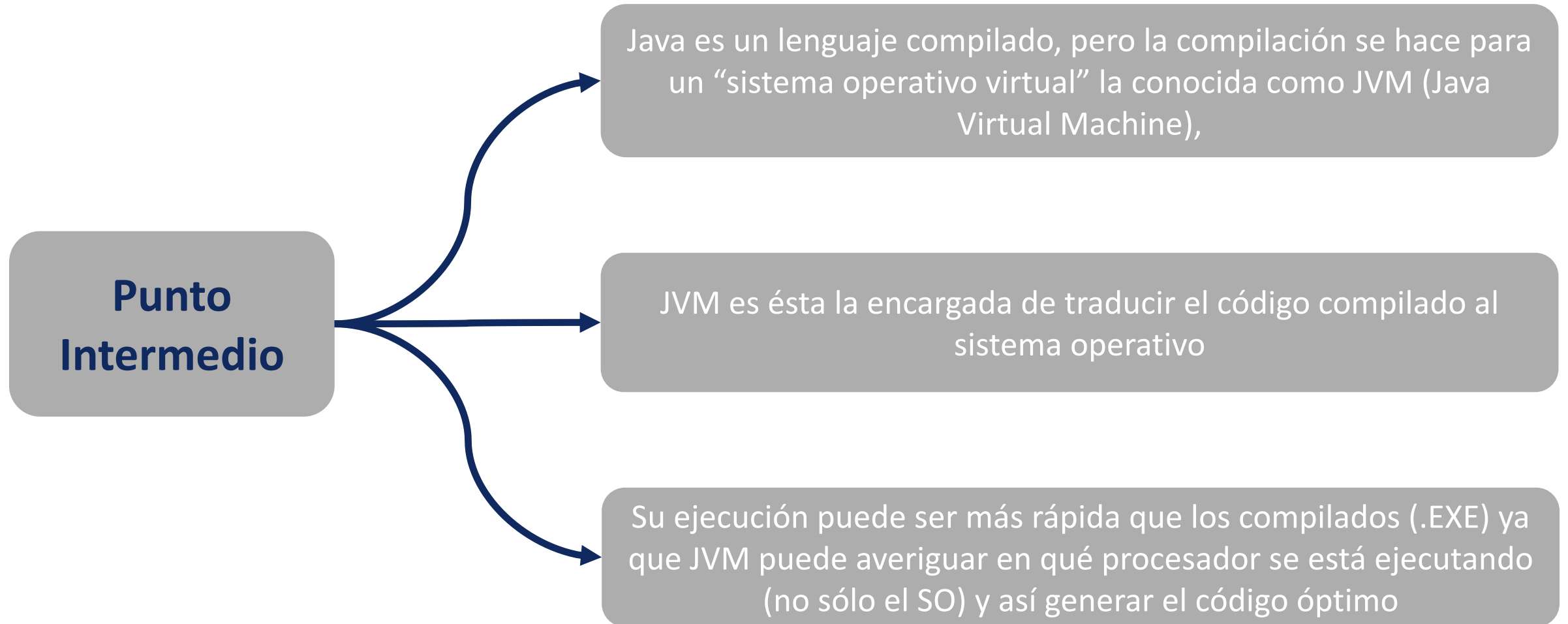
1.2 Programas y ejecutables



1.2 Programas y ejecutables



1.2 Programas y ejecutables



1.3 Procesos, servicios e hilos

Procesos

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

El proceso es la unidad mínima que trata un sistema operativo.

Servicios

En servicio es un proceso que no muestra ninguna ventana ni gráfico en la pantalla, esto es debido a que no está pensado para el usuario interactúe con el.

Hilos

Un proceso se puede descomponer en otros **subprocesos más ligeros** llamados hilos (threads).

Un hilo sólo puede existir dentro de un proceso y no puede ejecutarse por sí mismo, por lo tanto, todo proceso tiene un hilo principal del cual pueden crearse otros hilos secundarios.

Un proceso no tiene acceso a datos de otro proceso, sin embargo, un hilo si tiene acceso a datos de otros hilos (cuidado al programar).

2. Sistema multitarea

2.1 Multiproceso / Multihilo

2.2 Paralelismo. Programación concurrente y distribuida

2.3 Problemas inherentes a la programación concurrente

2.1 Multiproceso / Multihilo

Multiproceso

Consiste en la ejecución de varios procesos diferentes de forma simultánea para la realización de una o varias tareas relacionadas o no.

El caso más conocido es aquel en el que nos referimos al Sistema Operativo (Windows, Linux, MacOS, . . .) y decimos que es multitarea puesto que es capaz de ejecutar varias tareas o procesos al mismo tiempo.

Cada proceso es de una **aplicación diferente**.

Multihilo

Hablamos de multihilo cuando se ejecutan varias tareas relacionadas, o no, dentro de una **misma aplicación**.

En este caso no son procesos diferentes, sino que dichas tareas se ejecutan dentro del mismo proceso del Sistema Operativo.

A cada una de estas tareas se le conoce como hilo o thread (en algunos contextos también como procesos ligeros).

2.2 Proceso paralelo, concurrente y distribuido

Paralelismo

Un procesamiento es paralelo cuando se procesa de manera simultánea en los diversos núcleos de un procesador o en varios procesadores.

Concurrente

Un procesamiento es concurrente cuando varios procesos se ejecutan en una misma unidad de proceso de manera alterna.

Los sistemas operativos actuales son capaces de realizar avances pseudosimultáneos de varios procesos y evitando por ello la secuencialidad.

Distribuido

Es un caso particular del multiproceso, pero en este caso la ejecución del software se distribuye entre varios ordenadores, consiguiendo de una potencia de procesamiento mucho mayor.

Por lo tanto, el procesamiento distribuido es aquel en el que un proceso se ejecuta en unidades de computación independientes, conectadas (red) y sincronizadas.

2.3 Problemas inherentes a la programación concurrente

Sincronización



```
graph LR; A[Sincronización] --> B[Hace referencia a la necesidad de coordinar los procesos con el fin de sincronizar sus actividades. Imaginemos un proceso P1 que llega a un estado X y que no pueda continuar su ejecución hasta que otro proceso P2 llegue a un estado Y de su ejecución]; A --> C[La programación concurrente proporciona mecanismos para bloquear procesos a la espera de que ocurra un evento y para desbloquearlos cuando éste ocurra. Algunas herramientas para manejar la concurrencia son: la región crítica, los semáforos, monitores... que veremos más adelante];
```

Hace referencia a la necesidad de coordinar los procesos con el fin de sincronizar sus actividades.


Imaginemos un proceso P1 que llega a un estado X y que no pueda continuar su ejecución hasta que otro proceso P2 llegue a un estado Y de su ejecución

La programación concurrente proporciona mecanismos para bloquear procesos a la espera de que ocurra un evento y para desbloquearlos cuando éste ocurra.

Algunas herramientas para manejar la concurrencia son: la región crítica, los semáforos, monitores... que veremos más adelante

2.3 Problemas inherentes a la programación concurrente

Intercambio de información



Cada procesador realiza una parte del proceso y puede necesitar intercambiar información con el resto. Según cómo se realice este intercambio tendremos distintos modelos de programación:

- **Modelo de memoria compartida:** los procesadores comparten físicamente la memoria.
- **Modelo de paso de mensajes:** cada procesador dispone de su propia memoria independiente al resto y accesible sólo por él. Para realizar el intercambio de información es necesario que cada procesador realice la petición de datos al procesador que los tiene y éste haga el envío.

3. Procesos

3.1 Programas, proceso y ejecutables

3.2 Estados de un proceso

3.3 Algoritmos de planificación

3.4 Cambio de contexto

3.1 Programas, proceso y ejecutables

De forma sencilla, se puede decir que un proceso, es un programa en ejecución. Pero, es más que eso, un proceso en el sistema operativo (SO), es una **unidad de trabajo completa**, y es el **SO** quién gestiona los distintos procesos que se encuentren en ejecución en el equipo

Es muy importante diferencia ejecutable y proceso. Un ejecutable es un **fichero** y un proceso es una **entidad activa**, es el contenido del ejecutable ejecutándose

Por lo tanto, un proceso existe mientras que se esté ejecutando una aplicación. Es más, la ejecución de una aplicación puede implicar que se arranquen varios procesos en nuestro equipo; los cuales puedes estar formada por varios ejecutables y librerías.

3.1 Programas, proceso y ejecutables

Programa

Código + datos almacenados en soporte digital, que resuelve una necesidad concreta.

Proceso

Programa en ejecución, tiene: código ejecutable, datos, pila del programa (lista ordenada que permite almacenar y recuperar datos, LIFO), puntero de la pila, contador de programa (dirección de la siguiente instrucción que se va a ejecutar) y registros. Por lo tanto, siempre que hablemos de procesos, estamos hablando de memoria RAM. Los procesos son entidades independientes, podemos encontrar dos procesos que sean del mismo programa, pero cada uno tendrá su imagen de memoria, su contador de programa y sus datos independientes.

Ejecutable

Contiene la información necesaria para crear un proceso, es decir, el ejecutable es el fichero que permite poner el programa en ejecución y convertirlo en proceso. Según el código que contengan los ejecutables pueden ser: binarios, interpretados o librerías. Algunos ejemplos de librerías son: las librerías estándar de C, los paquetes compilados DLL en Windows; las API (Interfaz de Programación de Aplicaciones), la J2EE de Java (Plataforma Java Enterprise Edition versión 2); las librerías que incorpora el framework de .NET; etc.

3.2 Estados de un proceso



Nuevo. Proceso en creación

Preparado. Proceso a la espera de ejecución de sus instrucciones por parte de la CPU

Activo. Proceso cuyas instrucciones están siendo ejecutadas por la CPU

Terminado. Proceso que ha finalizado

Bloqueado. Proceso a la espera de una operación de E/S.

Suspendido. Proceso que se ha guardado en una memoria secundaria (disco dura, etc.) para liberar recursos de la memoria principal (RAM)

3.2 Estados de un proceso

Vamos a destacar dos componentes del SO que llevan a cabo toda la tarea:

Es el encargado de **crear los procesos**. Al iniciar un proceso (para cada proceso), el cargador realiza las siguientes tareas:

Cargador

➤ **Carga el proceso en memoria principal.**

- ✓ Reserva un espacio en la RAM para el proceso.
- ✓ En ese espacio, copia las instrucciones del fichero ejecutable de la aplicación y las constantes
- ✓ Deja un espacio para los datos (variables) y la pila (llamadas a funciones).

Nota. Un proceso, durante su ejecución, no podrá hacer referencia a direcciones que se encuentren fuera de su espacio de memoria; si lo intentara, el SO lo detectará y generará una excepción (produciendo, por ejemplo, los típicos pantallazos azules de Windows).

3.2 Estados de un proceso

Cargador

- **Crea una estructura de información llamada PCB (Bloque de Control de Proceso).**
La información del PCB, es única para cada proceso y permite controlarlo. Esta información, también la utilizará el planificador. Entre otros datos, el PCB estará formado por:
 - ✓ Identificador del proceso o PID. Es un número único para cada proceso, como un DNI de proceso.
 - ✓ Estado actual del proceso: en ejecución, listo, bloqueado, suspendido, finalizando.
 - ✓ Espacio de direcciones de memoria, donde comienza la zona de memoria reservada al proceso y su tamaño.
 - ✓ Información para la planificación: prioridad, quantum, estadísticas, ...
 - ✓ Información para el cambio de contexto: valor de los registros de la CPU, entre ellos el contador de programa y el puntero a pila. Esta información es necesaria para poder cambiar de la ejecución de un proceso a otro.

3.2 Estados de un proceso

Planificador

Es el encargado de decidir **qué proceso se ejecuta y cuánto tiempo se ejecuta.**

La política en la toma de decisiones del planificador se denominan **algoritmo de planificación.**

3.3 Algoritmos de planificación

FCFS

First Come First Served

El primer proceso que llegue al procesador se ejecuta antes y de forma completa. Hasta que su ejecución no termina, no podrá pasarse a ejecutar otro proceso.

RR

Round Robin

Se le conoce también como algoritmo de turno rotatorio. En este caso se designa una cantidad corta de tiempo (quantum) de procesamiento a todas las tareas. Las que necesiten más tiempo de proceso deberán esperar a que vuelva a ser su turno para seguir ejecutándose

SPF

Shortest Process First

En este algoritmo, de todos los procesos listos para ser ejecutados, lo hará primero el más corto.

SRT

Shortest Remaining Time

De todos los procesos listos para ejecución, se ejecutará aquel al que le quede menos tiempo para terminar.

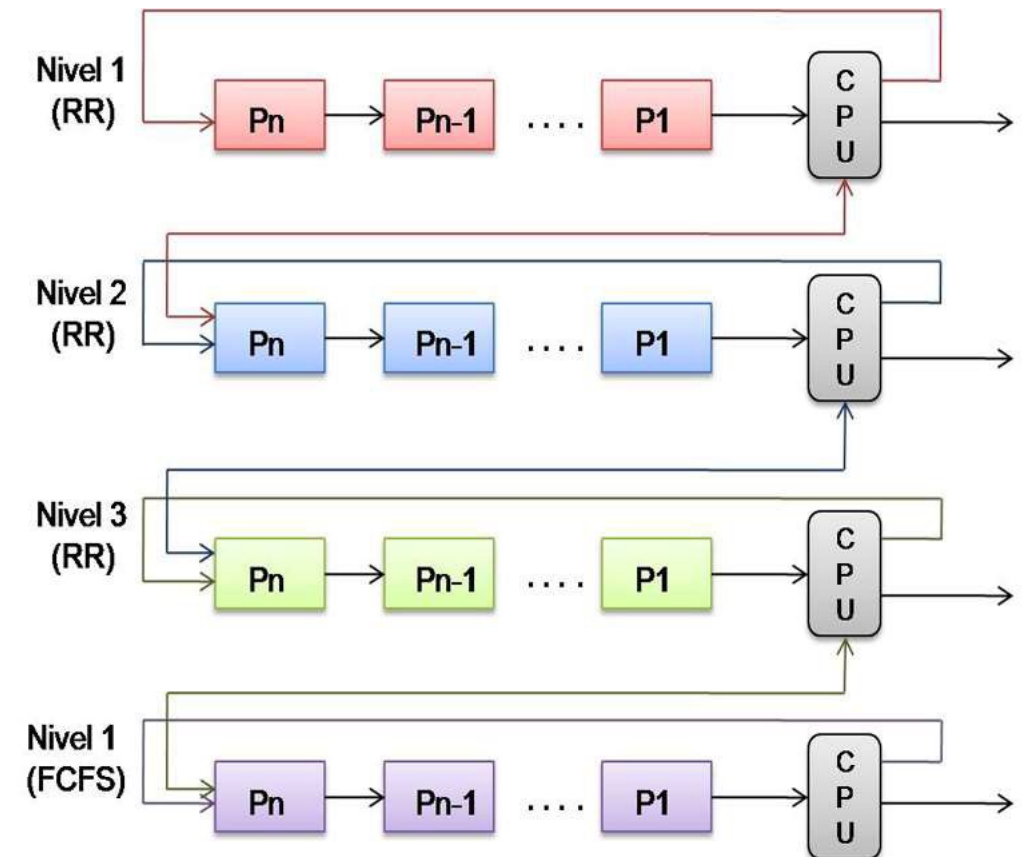
3.3 Algoritmos de planificación

Varias colas con realimentación

Es un algoritmo más complejo que todos los anteriores y, por tanto, más realista.

Se utiliza en entornos donde se desconoce el tiempo de ejecución de un proceso al inicio de su ejecución. En este caso, el sistema dispone de varias colas que a su vez pueden disponer de diferentes políticas unas de otras.

Los procesos van pasando de una cola a otra hasta que terminan su ejecución. En algunos casos, el algoritmo puede adaptarse modificando el número de colas, su política, ...



3.3 Ejemplo

Dado el siguiente listado de procesos

Proceso	Tiempo de ejecución	Tiempo de llegada
A	8	0
B	4	1
C	9	2
D	5	3
E	2	4

Utilizando los algoritmos FCFS, SPF y RR para cumplimentar la siguiente tabla

Proceso	Tiempo de ejecución	Tiempo de llegada	Tiempo de comienzo	Tiempo de finalización	Tiempo de retorno	Tiempo de retorno
A	8	0				
B	4	1				
C	9	2				
D	5	3				
E	2	4				

3.3 Ejemplo

[Ir a la Ejemplo Algoritmo 1](#)

3.4 Cambio de contexto

De forma general, el contexto de un proceso en un cierto instante de tiempo se puede definir como **la información relativa al proceso que el núcleo debe conocer para poder iniciar o continuar su ejecución.**

Cuando se ejecuta un proceso se dice que el sistema se ejecuta en el contexto de dicho proceso.

Por lo tanto, cuando el núcleo decide pasar a ejecutar otro proceso debe de cambiar de **contexto**, de forma que el sistema pasará a ejecutar el contexto del nuevo proceso.

3.4 Cambio de contexto

Dentro de la información que se guarda, podemos destacar:

Pila del programa

El Contador de Programa, que almacena la dirección de la siguiente instrucción a ejecutar

El Puntero a Pila, apunta a la parte superior de la pila del proceso en ejecución, donde será almacenado el contexto de la CPU. Y de donde se recuperará cuando ese proceso vuelva a ejecutarse

Registros

3.4 Cambio de contexto

El cambio de contexto que se hace al **cambiar un proceso es tiempo perdido**, ya que no se hace trabajo útil.

Cambiar el estado del proceso, significa cambiar el estado del procesador (cambio valores de registro), la información de la gestión de memoria y, por muy rápido que se haga, si se hace con mucha frecuencia puede provocar una ralentización del sistema. Por eso tener muchos programas abiertos provoca una disminución importante en el rendimiento del sistema.

4. Hilos

4.1 Conceptos y características

4.2 Hilo. Ventajas y uso

4.3 Hilo vs Proceso

4.1 Hilo. Concepto y características

Un hilo, denominado también **subproceso**, es un flujo de control secuencial independiente dentro de un proceso y está asociado con una secuencia de instrucciones, un conjunto de registros y una pila.

4.1 Hilo. Concepto y características

Podemos hacer las siguientes observaciones sobre los hilos:

- Un hilo no puede existir independientemente de un proceso.
- Un hilo no puede ejecutarse por sí solo.
- Dentro de cada proceso puede haber varios hilos ejecutándose.
- Un único hilo es similar a un programa secuencial.
- La habilidad de ejecutar varios hilos dentro de un proceso ofrece algo nuevo y útil, ya que cada uno de estos hilos puede ejecutar actividades diferentes al mismo tiempo.

4.1 Hilo. Concepto y características

Por otra parte, un hilo **puede compartir con otros hilos**, del mismo proceso, los siguientes recursos:

- Código.
- Datos (como variables globales).
- Otros recursos del sistema operativo, como los ficheros abiertos y las señales

El hecho de que los hilos compartan recursos (por ejemplo, pudiendo acceder a las mismas variables) implica que sea necesario utilizar esquemas de bloqueo y sincronización, lo que puede hacer más difícil el desarrollo de los programas, así como su depuración.

4.2 Hilo. Ventajas y uso

Como consecuencia de compartir el espacio de memoria, los hilos aportan las siguientes ventajas sobre los procesos:

- Se consumen **menos recursos** en el lanzamiento y la ejecución de un hilo que en el lanzamiento y ejecución de un proceso.
- Se tarda **menos tiempo** en crear y terminar un hilo que un proceso.
- La conmutación entre hilos del mismo proceso es bastante **más rápida** que entre procesos.

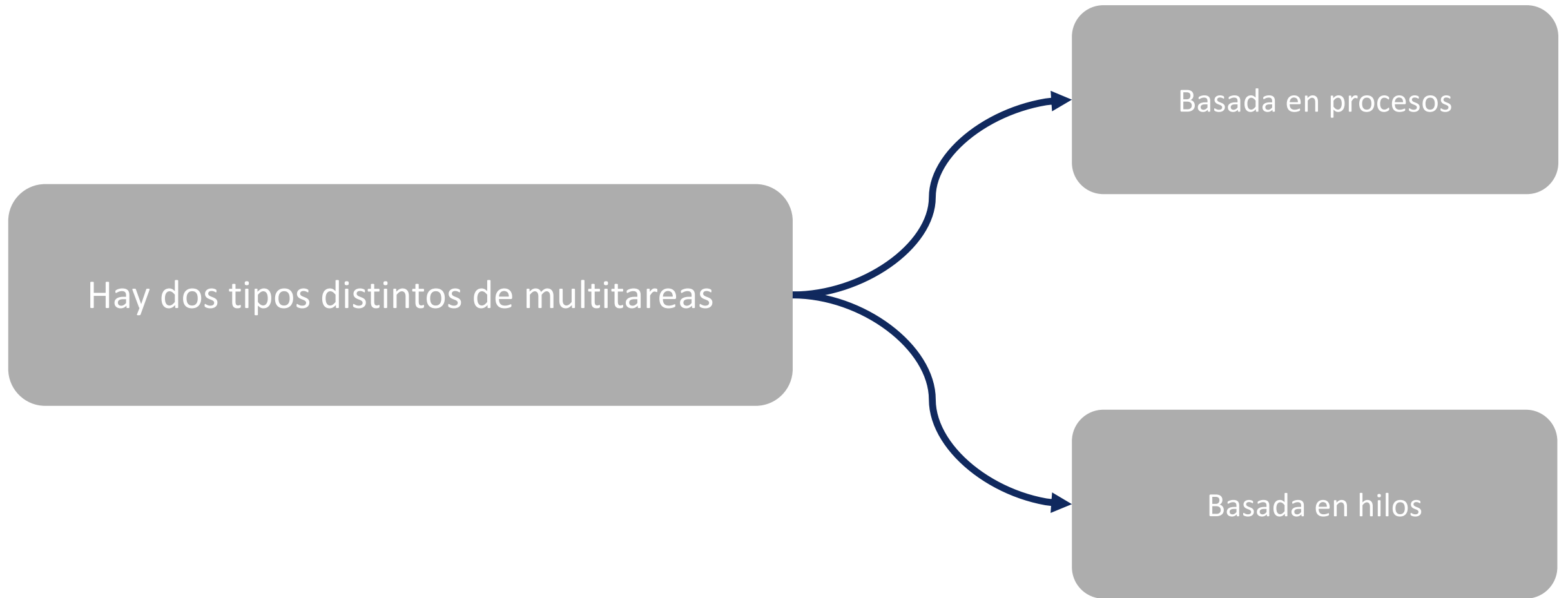
Por estas razones a los hilos se les denomina también **procesos ligeros**.

4.2 Hilo. Ventajas y uso

Se aconseja utilizar hilos en una aplicación cuando:

- La aplicación maneja entradas de varios dispositivos de comunicación.
- La aplicación debe poder realizar diferentes tareas a la vez.
- Interesa diferenciar tareas con una prioridad variada. Por ejemplo, una prioridad alta para manejar tareas de tiempo crítico y una prioridad baja para otras tareas.
- La aplicación se va a ejecutar en un entorno multiprocesador.

4.3 Hilo vs Proceso



4.3 Hilo vs Proceso

Un **proceso** es, en esencia, un **programa que se está ejecutando**.

La multitarea basada en procesos es la característica que le permite a su computadora **ejecutar dos o más programas al mismo tiempo**.

Por ejemplo, es una multitarea basada en procesos la que permite ejecutar el compilador de Java al mismo tiempo que utiliza un editor de texto o navegamos por Internet.

En la multitarea basada en procesos, un **programa es la unidad de código más pequeña** que puede enviar el planificador o bien ser gestionada con un sistema PVM (máquina virtual paralela, es una biblioteca para el cómputo paralelo en un sistema distribuido).

4.3 Hilo vs Proceso

En un entorno multitarea basado en hilos, **el hilo es la unidad más pequeña de código distribuible.**

Esto significa que **un solo programa puede realizar dos o más tareas a la vez.**

Por ejemplo, un editor de texto puede formatear texto al mismo tiempo que está imprimiendo, siempre que estas dos acciones se realicen mediante dos hilos separados

4.3 Hilo vs Proceso

La multitarea basada en procesos no está bajo el control del programa, sino del SO.

La multitarea multihilo sí está bajo el control del programa, en nuestro caso de Java

4.3 Hilo vs Proceso

Un hilo o thread es cada una de las tareas que puede realizar de forma simultánea una aplicación.

Por defecto, toda aplicación dispone de un único hilo de ejecución, al que se conoce como **hilo principal**. Si dicha aplicación no despliega ningún otro hilo, sólo será capaz de ejecutar una tarea al mismo tiempo en ese hilo principal.

Para cada tarea adicional que se quiera ejecutar en esa aplicación, se deberá lanzar un nuevo hilo o thread. Para ello, todos los lenguajes de programación, como Java, disponen de una API para crear hilos y trabajar con ellos.

5. Gestión de procesos en el SO

5.1 Gestión de procesos en Windows

5.2 Gestión de procesos en Linux

5 Gestión de procesos en el SO

La gestión de procesos se realiza de dos formas muy distintas en función del Sistema Operativo.

Veamos cómo los maneja Windows y Linux

5.1 Gestión de procesos en Windows

Tenemos dos maneras de verlo:

1. Para ver los procesos en Windows, iremos al Administrador de Tareas (Ctrl+Alt+Supr), y luego a la pestaña Procesos.
2. Si los queremos ver desde la línea de comandos, en CMD escribiremos tasklist.

5.1 Gestión de procesos en Windows

Si lo hacemos desde la línea de comandos, además de los procesos, veremos el PID, que es el identificador del proceso.

```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.18363.1256]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ usuario >tasklist

Nombre de imagen                PID Nombre de sesión Núm. de ses Uso de memor
=====
System Idle Process             0 Services          0          8 KB
System                          4 Services          0         4.752 KB
Registry                        96 Services          0        52.356 KB
smss.exe                       504 Services          0          724 KB
csrss.exe                      688 Services          0         2.884 KB
wininit.exe                    704 Services          0         2.720 KB
```

5.1 Gestión de procesos en Windows

En sistemas Windows, no existen apenas comandos para gestionar procesos:

- ***tasklist***. Lista los procesos presentes en el sistema. Mostrará, entre otros datos, el nombre del ejecutable, su correspondiente Identificador de proceso, así como el porcentaje de uso de memoria.
- ***taskkill***. Mata procesos. Con la opción /PID especificaremos el Identificador del proceso que queremos matar.
- ***start*** documento.pdf. Obliga al SO a arrancar la aplicación asociada a un archivo. En este caso se abrirá el visor de archivos PDF , que se cargará automáticamente con el fichero documento.pdf

5.2 Gestión de procesos en Linux

En GNU/Linux se puede utilizar un terminal de consola para la gestión de procesos, lo que implica que no solo se pueden arrancar procesos, si no también detenerlos, reanudarlos, terminarlos y modificar su prioridad de ejecución.

5.2 Gestión de procesos en Linux

ps : Obtiene un listado de todos los procesos que se encuentran activos en el sistema, mostrando su PID, usuario y ubicación de su fichero ejecutable.

Ps -f: también aparecerá el PPID, el identificador del proceso padre.

```
inazio@inazio-VirtualBox:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
inazio      1611    1604  0  00:31 pts/1        00:00:00 bash
inazio      2032    1611  0  00:35 pts/1        00:00:00 ps -f
```

PID: identificador del proceso

TIME: tiempo de ejecución asociado, es la cantidad total de tiempo CPU que el proceso ha utilizado desde que nació

C: porcentaje de CPU utilizado por el proceso.

STIME: hora de inicio del proceso

UID: nombre del usuario

CMD: nombre del proceso

5.2 Gestión de procesos en Linux

Ps -AF: aparecen todos los procesos lanzados en el sistema.

```
inazio@inazio-VirtualBox:~$ ps -AF
```

UID	PID	PPID	C	SZ	RSS	PSR	STIME	TTY	TIME	CMD
root	1	0	0	885	1980	0	00:30	?	00:00:00	/sbin/init
root	2	0	0	0	0	0	00:30	?	00:00:00	[kthreadd]
root	3	2	0	0	0	0	00:30	?	00:00:00	[ksoftirqd/0]
root	5	2	0	0	0	0	00:30	?	00:00:00	[kworker/0:0H]
root	7	2	0	0	0	0	00:30	?	00:00:00	[migration/0]
root	8	2	0	0	0	0	00:30	?	00:00:00	[rcu_bh]
root	9	2	0	0	0	0	00:30	?	00:00:00	[rcu_sched]
root	10	2	0	0	0	0	00:30	?	00:00:00	[watchdog/0]
root	11	2	0	0	0	0	00:30	?	00:00:00	[khelper]
root	12	2	0	0	0	0	00:30	?	00:00:00	[kdevtmpfs]
root	13	2	0	0	0	0	00:30	?	00:00:00	[netns]
root	14	2	0	0	0	0	00:30	?	00:00:00	[writeback]
root	15	2	0	0	0	0	00:30	?	00:00:00	[kintegrityd]
root	16	2	0	0	0	0	00:30	?	00:00:00	[bioset]

5.2 Gestión de procesos en Linux

kill : Detiene y/o terminar un proceso .

Se puede usar este comando para terminar un proceso sin guardar nada usando
kill -SIGKILL <numproceso> o ***kill -<numproceso>*** (sin preguntar)

Para pausar un proceso utilizaremos ***kill -SIGSTOP <numproceso>*** y para reanuncarlo utilizaremos
kill -SIGCONT <numproceso>

Nice: permite indicar prioridades entre -20 y 19. El -20 implica que un proceso reciba la máxima prioridad, y el 19 supone asignar la mínima prioridad

5.2 Gestión de procesos en Linux

En sistemas como GNU/Linux se puede modificar la prioridad con que se ejecuta un proceso. Esto implica dos posibilidades:

Si pensamos que un programa que necesitamos ejecutar es muy importante, podemos darle más prioridad para que reciba «más turnos» del planificador.

Y, por el contrario, si pensamos que un programa no es muy necesario podemos quitarle prioridad y reservar «más turnos de planificador» para otros posibles procesos.

MUCHAS GRACIAS A TODOS.
