

# PROGRAMACIÓN DE SERVICIOS Y PROCESOS

1º CFGS DAM

Tema 01

Programación  
multiproceso

# PROGRAMACIÓN MULTIPROCESO

## Introducción

Los ordenadores son capaces de realizar muchas tareas de manera simultánea, pese a que el número de procesadores que tienen no es muy alto habitualmente. En parte es un engaño, ya que las tareas se van ejecutando por turnos, pero a tal velocidad que el ojo humano no es capaz de apreciar las ínfimas discontinuidades en la ejecución.

En otras ocasiones, en cambio, el procesamiento es realmente simultáneo. Los procesadores modernos tienen varios núcleos de ejecución que funcionan como procesadores de facto. Cada núcleo es capaz de ejecutar una instrucción en un instante dado, por lo que se podrán ejecutar a la vez tantas instrucciones como núcleos de manera real.

También existe la posibilidad de utilizar varios ordenadores y construir una red en la que los elementos se distribuyan el trabajo para hacer bueno aquello de la unión hace la fuerza. Varios procesadores con distintos núcleos trabajan a la vez resolviendo diversos procesos de manera simultánea. En esta unidad se presentan las bases de la programación multiproceso para comprender el funcionamiento de esta área de la informática.

### 1. Introducción a los sistemas multitarea

Un ordenador actual de potencia media con un procesador con cuatro núcleos y correctamente configurado es capaz de realizar simultáneamente varias tareas. Por ejemplo, puede reproducir un fichero de sonido, imprimir un documento, descargar un

programa de internet, recibir un correo electrónico, actualizar el sistema operativo y monitorizar la temperatura de la CPU. Este hecho sugiere una pregunta, ¿cómo puede un ordenador ejecutar a la vez más tareas que el número de unidades de proceso que tiene disponibles? La respuesta es gracias a la **multitarea**, que es la capacidad de ejecutar varias tareas simultáneamente por parte de un computador.

Los primeros ordenadores domésticos eran muy limitados y algunos sistemas operativos reflejaban de forma evidente esa limitación: o se estaba editando un documento de texto, o ejecutando un programa que realizaba un cálculo o jugando a un videojuego. Solo permitían hacer una tarea en un momento determinado. Estos sistemas se denominan **monotareas** y el sistema operativo MS-DOS, publicado en 1981, sería uno de sus principales ejemplos. No obstante, esa fecha no debe ser una referencia del paso de la monotarea a la multitarea, ya que sistemas como UNIX, desarrollado varios años antes, ya contemplaban la multitarea.

La **multitarea** es, por lo tanto, la capacidad que tienen los ordenadores de realizar varias tareas (ejecutar varios programas) al mismo tiempo, independientemente del número de elementos de procesamiento. Esta multitarea puede ser real (hay tantas unidades de proceso como procesos a ejecutar) o simulada (hay menos unidades de proceso que procesos a ejecutar).

La capacidad de realizar multitarea de un sistema depende principalmente del procesador y del sistema operativo. El primero tiene la capacidad física y el segundo, la capacidad lógica. Curiosamente, unas determinadas características del sistema operativo son necesarias para que exista multitarea mientras que las necesidades del procesador son menos restrictivas: un

procesador simple con un sistema operativo adecuado puede disponer de multitarea; un procesador con múltiples núcleos con un sistema operativo inadecuado no aprovechará los recursos disponibles.

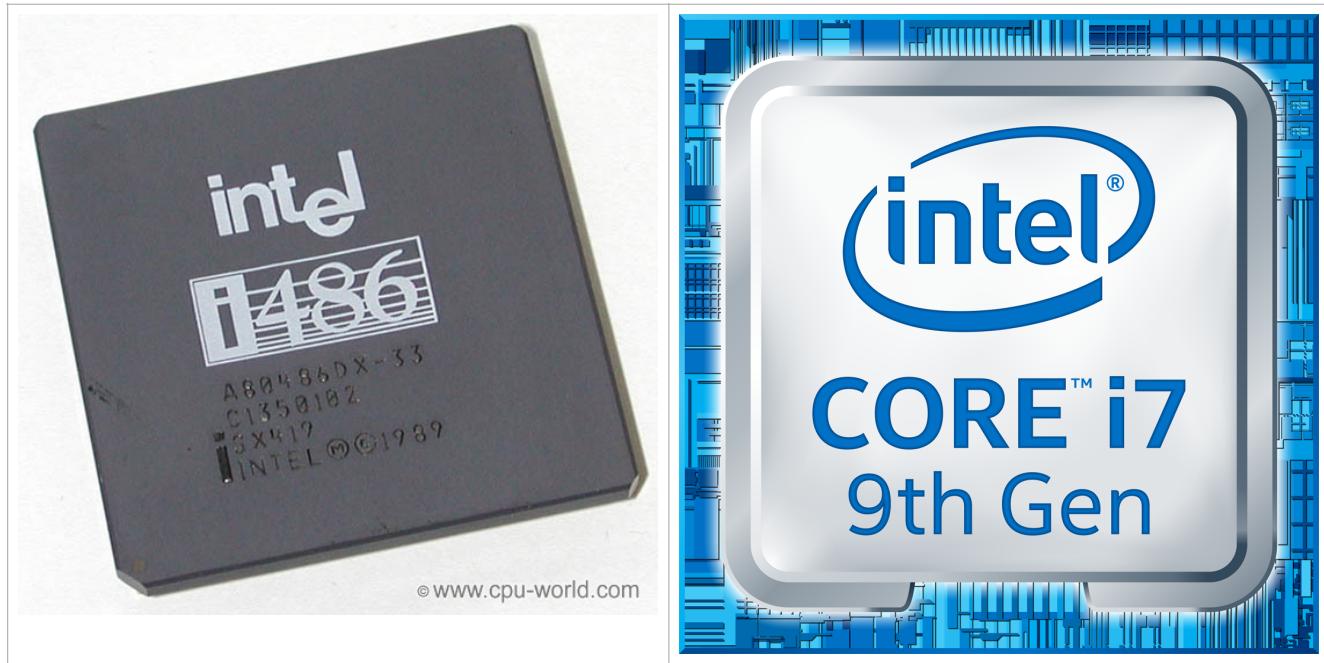
Alrededor de la multitarea existe una serie de conceptos que, en ocasiones, provocan confusión, ya que la frontera que los delimita es difusa. Concurrencia, paralelismo, proceso o hilo son algunos de los términos cuyo significado es fundamental explicar para comprender esta interesante disciplina de la computación.

## 1.1. El procesador

Uno de los elementos más importante de un ordenador (o de un teléfono móvil, o de una videoconsola...) es el procesador. Este componente es el que proporciona la capacidad de ejecutar las instrucciones de los programas. Es el "cerebro" del dispositivo. Existen muchos modelos de procesador, tantos como necesidades de capacidad, tamaño y coste. Desde el legendario Zilog Z80A de los ordenadores Sinclair ZX Spectrum de los años ochenta del siglo xx hasta los modernos Intel Core i9, por citar solo dos ejemplos representativos, ha habido un salto en potencia, gestión energética y reducción de tamaño inimaginable hace años. También es llamativo el incremento en el número núcleos y su impacto en la multitarea,

Un **núcleo** es una unidad con capacidad de ejecución dentro de un procesador. En un sistema con un procesador y cuatro núcleos se pueden ejecutar cuatro instrucciones simultáneamente. Esto no significa que se disponga de cuatro procesadores, pero sí que se tiene mucha más capacidad de procesamiento simultáneo.

Obviamente, no todos los procesadores son iguales. Sin ir a los extremos más distantes se pueden comparar los procesadores Intel i486 SX de 33 MHz y un solo núcleo con los Intel Core i7 de más de 2 GHz en su modelos más sencillos y cuatro núcleos. Los resultados de dicha comparación evidencian las diferencias entre distintos procesadores.



¿Significa todo esto que un procesador con un único núcleo no puede realizar multitarea?

La respuesta es no. De hecho, el procesamiento multitarea existe en sistemas operativos que se ejecutan sobre arquitecturas que disponen de un solo procesador con un único núcleo.

Cualquier procesador puede, potencialmente, ejecutar varias tareas y que parezca que las realiza todas simultáneamente. El truco consiste en dedicar una pequeña porción de tiempo de ejecución a cada una, de tal manera que no se aprecie el cambio entre tarea (denominado **cambio de contexto**). Si el ordenador es suficientemente rápido se producirá una ilusión de ejecución simultánea. La realidad es que todas las tareas avanzan sin tener

que esperar unas a que las otras terminen, pero en un instante dado solo alguna de ellas está realmente en ejecución.

Un procesador con varios núcleos, o un sistema basado en varios procesadores, podrá realizar una ejecución simultánea real, pero nunca será capaz de ejecutar más operaciones concurrentes que el número de unidades de procesamiento disponible.

## 1.2. Programas. Ejecutables. Procesos. Servicios

Programa, proceso, ejecutable y servicio son términos que hacen referencia a elementos distintos, pero íntimamente relacionados.



Es necesario comprender las particularidades de cada uno de ellos para poder avanzar en el conocimiento de la programación multiproceso,

El **sistema operativo** es el elemento del ordenador que coordina el funcionamiento

del resto de componentes de este, tanto software como hardware. Es a él a quien se indica qué se quiere hacer o, siendo más precisos, qué programas se desean ejecutar.

Para llegar a poder ejecutar un programa primero hay que obtenerlo o, en el caso de los programadores, crearlo, para posteriormente proceder a su ejecución. El proceso de creación por parte del programador y de ejecución por parte del usuario final de un programa compilado es el siguiente:

1. El programador escribe el **código fuente** con un editor de texto y lo almacena en un fichero.
2. El programador compila el código fuente utilizando un compilador, generando un **programa ejecutable**. Este programa contiene instrucciones comprensibles por el sistema operativo para el cual se realizó la compilación.
3. El usuario ejecuta el programa ejecutable, generando un **proceso**.

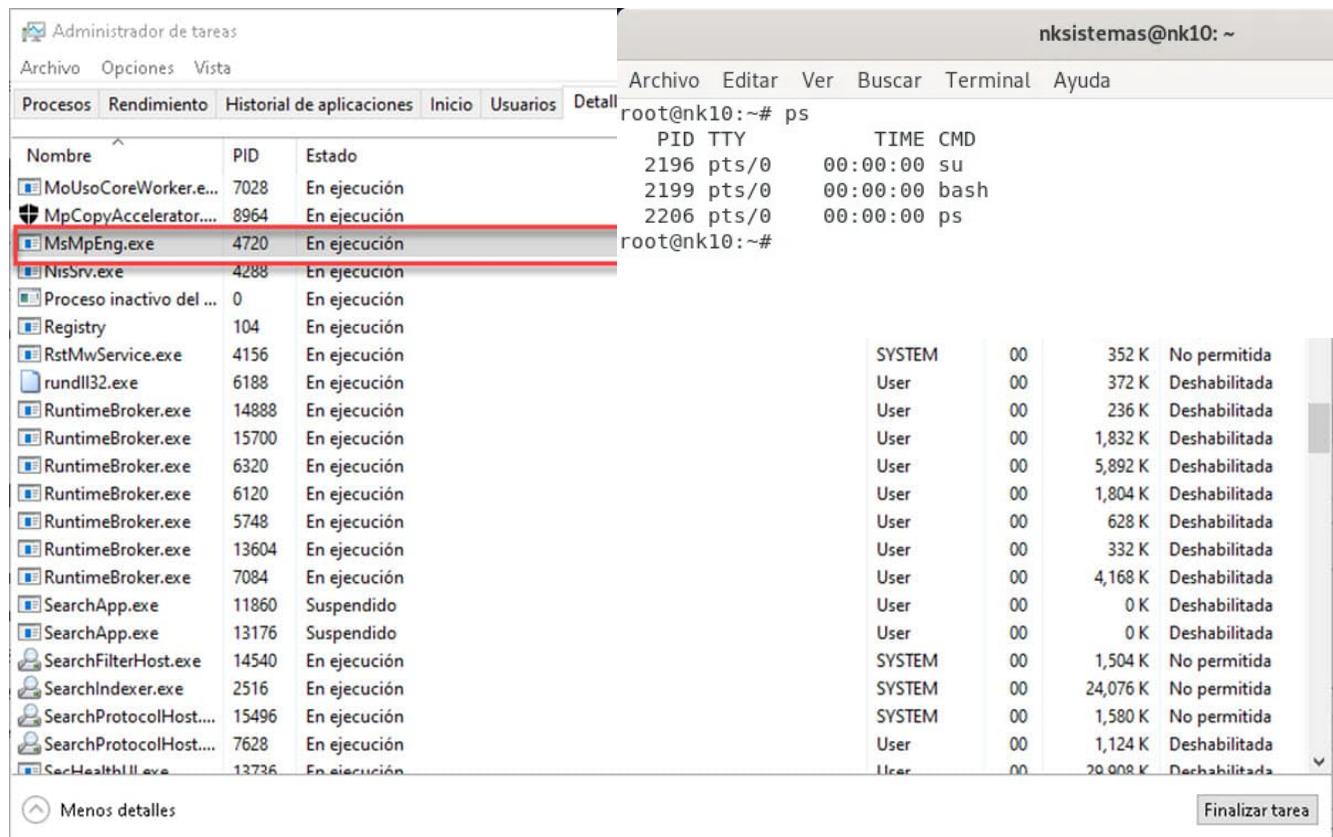
Por lo tanto, se puede afirmar que un programa, al ser ejecutado por un usuario, genera un proceso en el sistema operativo: **un proceso es un programa mientras se encuentra en ejecución**.

Por su parte, un **servicio** es también un programa cuya ejecución se realiza en segundo plano y que no requiere la interacción del usuario. Normalmente, se arranca de manera automática por el sistema operativo y está en constante ejecución.

Todos los sistemas operativos modernos distinguen entre procesos y servicios, aunque en esta unidad se pondrá el foco en los primeros.

En la figura se puede observar el Administrador de tareas de Windows 10. En la pestaña "Procesos" se muestran las aplicaciones que están en ejecución, junto con su PID (identificador de proceso) y datos relacionados con el consumo de recursos.

En Unix y sus derivados, el comando ps (process status) muestra, con distinto nivel de detalle, información sobre los procesos que están en ejecución en un momento determinado, incluyendo también su identificador de proceso (PID).



```
nksistemas@nk10: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@nk10:~# ps
    PID TTY      TIME CMD
  2196 pts/0    00:00:00 su
  2199 pts/0    00:00:00 bash
  2206 pts/0    00:00:00 ps
root@nk10:~#
```

En Windows se puede ejecutar este comando desde la consola PowerShell.

Es interesante reseñar que cuando el programa que se ejecuta no es compilado, el proceso que se arranca no es el propio programa, sino el del intérprete (como en Python) el de la máquina virtual (como en Java). En ambos casos el nombre del proceso no coincidirá con el nombre del programa.

Como se ha comentado anteriormente, el número de procesos que puede estar conviviendo en un sistema es alto y muy superior al número de núcleos de ejecución. Es el sistema operativo el encargado de conseguir que sea posible repartir los limitados recursos del ordenador de forma ordenada, justa y eficiente.

## 1.4. Computación concurrente y paralela

Salvo los sistemas más antiguos, sencillos o limitados, como MS-DOS o los embebidos en las placas de desarrollo Arduino, todos los sistemas operativos modernos disponen de la capacidad de realizar multitarea o multiproceso.

La **multitarea**, como indica su nombre, es la capacidad de realizar varias tareas simultáneamente, frente a la restricción de la monotarea, en donde las tareas se ejecutan una detrás de otra. Cuando se trabaja en un sistema multitarea, varias tareas avanzan a la vez, aunque pueden hacerlo de diferentes maneras.

Un sistema basado en un único procesador con un único núcleo es capaz de realizar multitarea mediante el uso de la **conurrencia**. En la computación concurrente, los tiempos de CPU se reparten entre los distintos procesos según una planificación dirigida por el sistema operativo. Si la CPU es suficientemente rápida, el número de procesos limitado y el planificador tienen un buen diseño, todas las tareas avanzarán a la vez (aparentemente) pese a que en un ciclo de CPU en un momento dado solo se puede ejecutar una instrucción. En una unidad de tiempo de computación solo avanza un proceso. La velocidad en la asignación de la CPU a los distintos procesos logra que no se perciba el cambio, pero, aunque este se apreciase, seguiría siendo multitarea, ya que todas las tareas avanzan sin tener que esperar unas a que las otras terminen. Este tipo de computación se denomina **concurrente**.

Los sistemas basados en varios procesadores o en procesadores de varios núcleos aportan una mejora sustancial: permiten ejecutar varias instrucciones en un único ciclo de reloj. Esta capacidad hace posible ejecutar en paralelo varias instrucciones, lo que da origen al término **procesamiento paralelo**. En este tipo de procesamiento, los procesos se dividen en pequeñas subtareas (hilos) que se ejecutan en los diferentes núcleos, con lo que se consigue una reducción en los tiempos de ejecución de los procesos.

Como resumen, se pueden definir los conceptos de procesamiento concurrente y paralelo de la siguiente manera:

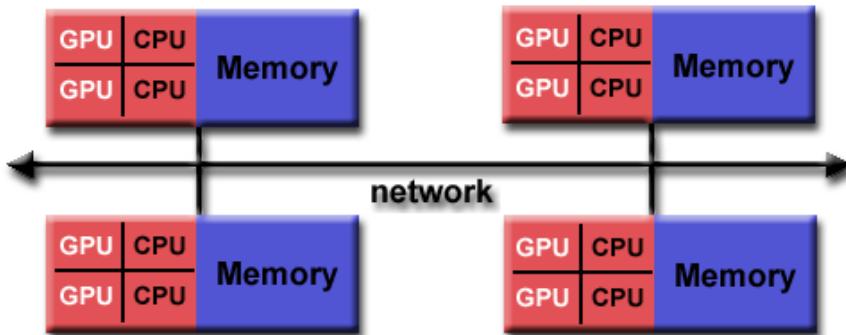
- **Procesamiento concurrente.** Es aquel en el que varios procesos se ejecutan en una misma unidad de proceso de manera alterna, provocando el avance simultáneo de los mismos y evitando la secuencialidad.
- **Procesamiento paralelo.** Es aquel en el que divisiones de un proceso se ejecutan de manera simultánea en los diversos núcleos de ejecución de un procesador o en diversos procesadores.

Se puede, para finalizar, concluir que el procesamiento concurrente es responsabilidad del sistema operativo mientras que el procesamiento paralelo es una responsabilidad compartida entre el sistema operativo y el programa.

## 1.4. Programación distribuida

La programación distribuida es otro de los paradigmas de la programación multiproceso. En este tipo de arquitectura, la ejecución del software se distribuye entre varios ordenadores, consiguiendo así disponer de una potencia de procesamiento

mucho más elevada, escalable y, normalmente, económica. Si en un único ordenador el número de núcleos viene determinado por el procesador que se está utilizando y es inmutable, en un sistema distribuido se elimina dicha restricción.



Para construir un sistema distribuido se requiere una red de ordenadores entre los que distribuir el trabajo. No todas las tareas son susceptibles de

distribuirse ni en todos los casos se obtendrá beneficio respecto de una ejecución convencional, pero en el caso en el que la ventaja es posible, estos tipos de sistemas son altamente eficientes y no requieren de una inversión tan elevada como la resultante de conseguir la misma potencia de cálculo con un único ordenador.

Se puede definir el **procesamiento distribuido** como aquel en el que un proceso se ejecuta en unidades de computación independientes conectadas y sincronizadas.

## 1.5. Hilos

Un programa básico está compuesto por una serie de sentencias que se ejecutan de manera secuencial y síncrona: hasta que no se completa la ejecución de la primera de las sentencias no se comienza con la ejecución de la segunda, y así sucesivamente hasta terminar la ejecución del programa completo.

En muchos casos, es necesaria esta secuencialidad y sincronía, ya que los diferentes pasos del algoritmo programado son dependientes entre sí y no existe la posibilidad de invertir el

orden de ejecución sin generar un resultado del proceso erróneo. En otros casos, en cambio, un algoritmo podría trocearse en varias unidades más pequeñas, ejecutar cada una de ellas por separado y en paralelo, juntar los resultados sin que importe el orden en que se obtengan y generar el resultado final. Esta técnica se conoce como **programación multihilo**.

Los hilos de ejecución son fracciones de programa que, si cumplen con determinadas características, pueden ejecutarse simultáneamente gracias al procesamiento paralelo. Al formar parte del mismo proceso son extremadamente económicos en referencia a los recursos que utilizan

Los programas que se ejecutan en un único hilo se denominan **programas monohilo** mientras que los que se ejecutan en varios hilos se conocen como **programas multihilo**

## 1.7. Bifurcación o fork

Una bifurcación, referenciada habitualmente por su término en inglés **fork**, es una copia idéntica de un proceso. El proceso original se denomina **padre** y sus copias, **hijos**, teniendo todos ellos diferentes identificadores de proceso (PID). La copia creada continua con el estado del proceso original (padre), pero a partir de la creación cada proceso mantiene su propio estado de memoria.

En el siguiente código se realiza la creación de un nuevo proceso (proceso hijo) mediante la llamada a la instrucción `fork` de un proceso escrito en lenguaje C y ejecutándose en un sistema Linux.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(void) {
    int contador=0;
    printf("Comenzando la ejecución\n");
    pid_t idHijo;
    pid_t idPadre;
    idPadre = getpid();
    printf("Antes de bifurcar\n");
    contador++;
    idHijo = fork();
    contador++;
    printf("Después de bifurcar\n");
    if (idHijo == 0) {
        printf("Id. hijo:%ld Id. padre:%ld Contador:%d \n ", (long)getpid(),
(long)idPadre, contador);
    } else {
        printf("Id. padre:%ld Id. hijo:%ld Contador:%d \n ", (long)getpid(),
(long)idPadre, contador);
    }
    return 0;
}
```

La salida generada por la ejecución es la siguiente:

Comenzando la ejecución

Antes de bifurcar

Después de bifurcar

Después de bifurcar

Id. padre:143 Id. hijo:143 Contador:2

Id. hijo: 144 Id. padre 143 Contador:2

Sin entrar en detalles sintácticos sobre el lenguaje C. los aspectos relevantes de la creación de la bifurcación son los siguientes:

- ◆ La variable contador se declara e inicializa a 0 en la línea 5.
- ◆ En la linea 6 se muestra el texto "Comenzando la ejecución una única vez, ya que en este punto el proceso es único.
- ◆ En la linea 10 se muestra el texto "Antes de bifurcar" una única vez, ya que en este punto el proceso es único.

- ◆ En la línea 11 el valor de contador se incrementa en 1, tomando el valor 1.
- ◆ En la linea 12 se crea la bifurcación.
- ◆ En la línea 13 el valor de contador se incrementa en 1, una vez por cada proceso. No obstante, cada proceso dispone de su propio espacio de memoria por lo que variable de ambos procesos es distinta.
- ◆ En la linea 14 se muestra el texto "Después de bifurcar" dos veces, una por cada copia del proceso.
- ◆ En la línea 16 se muestra la información referente al proceso hijo. Se puede observar que el PID es 144 y el valor de la variable contador es 2.
- ◆ En la línea 18 se muestra la información referente al proceso padre. Se puede observar que el PID es 143 y el valor de la variable contador es 2.

## 2. Procesos: conceptos teóricos

De manera simplificada, se puede definir un **proceso** como un programa en ejecución. Algunos ejemplos de proceso podrían ser las instancias de un navegador web, de un procesador de textos, de un entorno de desarrollo o de una máquina virtual de Java.

Cada proceso está compuesto por:

- Las instrucciones que se van a ejecutar.
- El estado del propio proceso.
- El estado de la ejecución, principalmente recogido en los registros del procesador.
- El estado de la memoria.

Los procesos están constantemente entrando y saliendo del procesador. Se denomina **contexto** a toda la información que

determina el estado de un proceso en un instante dado. Es una especie de fotografía que permite quitar al proceso del procesador y restaurarlo en otro momento en el mismo estado en el que se encontraba.

Sacar a un proceso del procesador para meter a otro se conoce como **cambio de contexto**.

Un cambio de contexto implica capturar el estado de la CPU y de sus registros, de la memoria y de la propia ejecución del proceso saliente para restaurar la información equivalente del proceso entrante y poder continuar en el punto en el que este último abandonó el procesador en el cambio de contexto anterior. Lógicamente, el cambio de contexto implica un consumo de recursos y en circunstancias extremas y con un **planificador** (dispatcher) mal diseñado, el sistema podría estar constantemente realizando cambios de contexto sin que los procesos implicados avanzasen en la ejecución.

Los pasos que se han de realizar para llevar a cabo un cambio de contexto se pueden resumir en los siguientes:

- Guardar el estado del proceso actual.
- Determinar el siguiente proceso que se va a ejecutar
- Recuperar y restaurar el estado del siguiente proceso.
- Continuar con la ejecución del siguiente proceso.

Es el sistema operativo el encargado de la gestión de los procesos, quedando en la responsabilidad del programador el crear los programas que van a dar lugar a los procesos y en manos de los usuarios ejecutarlos.

## 2.1. Gestión y estados de los procesos

En la actualidad, prácticamente todos los sistemas de computación (ordenadores, teléfonos móviles inteligentes -smartphones-, tabletas, relojes inteligentes -smartwatches-, videoconsolas, etc.) son multiproceso, por lo que tienen la capacidad de mantener en ejecución simultánea varios programas o procesos. Normalmente, el número de estos es mucho mayor que el número de núcleos de ejecución que hay disponibles en el procesador o procesadores que tenga el dispositivo.

Los procesos necesitan recursos y estos son limitados. El procesador, la memoria, el acceso a los sistemas de almacenamiento o a los diferentes dispositivos son algunos de ellos. La pregunta que surge como consecuencia de esta afirmación es la siguiente: ¿cómo se consigue que la convivencia entre los procesos, que compiten entre sí por los limitados recursos del sistema de computación, sea posible? La respuesta está en el sistema operativo y, más concretamente, en el **planificador de procesos**.

El planificador de procesos es el elemento del sistema operativo que se encarga de repartir los recursos del sistema entre los procesos que los demandan. De hecho, es uno de sus componentes fundamentales, ya que determina la calidad del multiproceso del sistema y, como consecuencia, la eficiencia en el aprovechamiento de los recursos. Los objetivos del planificador son los siguientes:

- Maximizar el rendimiento del sistema
- Maximizar la equidad en el reparto de los recursos.
- Minimizar los tiempos de espera.
- Minimizar los tiempos de respuesta.

Se puede sintetizar que el objetivo del planificador es conseguir que todos los procesos terminen lo antes posible aprovechando al máximo los recursos del sistema. La tarea, como se puede suponer, es compleja.

Es posible imaginar al planificador como el responsable del almacén de las herramientas de un taller. El número de herramientas es limitado y mucho menor que el número de empleados que las necesitan para realizar su trabajo. El responsable del almacén debe satisfacer los siguientes requisitos:

- Todos los empleados tienen acceso a las herramientas para poder realizar su trabajo.
- Todos los empleados deben tener acceso a las herramientas durante aproximadamente la misma cantidad de tiempo.
- Ningún empleado puede quedarse la herramienta de forma permanente.
- Ningún empleado puede estar eternamente esperando a que le llegue el turno de usar la herramienta.
- Si hay una emergencia, el responsable debe tener la capacidad de recuperar las herramientas del empleado que las esté utilizando para dicha emergencia.

Si este proceso de gestión de recursos se realiza a la suficiente velocidad y los empleados fuesen muy rápidos, la percepción de un observador externo sería que todos los empleados avanzan en la ejecución de sus respectivas tareas a la vez. Eso es, en cierto modo, lo que ocurre en un sistema de computación gracias al planificador de procesos.

Existen muchos algoritmos para la planificación de los procesos, hay que considerar que cada sistema operativo utiliza sus propias estrategias de gestión de recursos a distintos niveles y que dichas

estrategias influyen de manera directa en el funcionamiento del sistema.

Para realizar la gestión de los procesos, el planificador necesita conocer el estado en que se encuentran. En un momento dado, un proceso se encuentra en un estado de un conjunto de estados posibles. Los estados básicos en los que puede estar un proceso son los siguientes:

- **Nuevo.** Técnicamente no es un estado, sino el reflejo del instante en el que se crea el proceso.
- **Listo.** El proceso está en memoria, preparado para ejecutarse. Está a la espera de que el planificador le conceda tiempo de procesamiento.
- **En ejecución.** El proceso se encuentra en ejecución.
- **Bloqueado.** Se encuentra a la espera de que ocurra un evento externo ajeno al planificador.
- **Finalizado.** Al igual que el estado Nuevo, no es técnicamente un estado. Refleja el instante posterior a la finalización del proceso.

Las posibles transiciones entre estos estados se recogen en el diagrama de transición de estados que representa su ciclo de vida mostrado en la siguiente figura



Este es un diagrama simplificado, ya que no recoge estados intermedios relacionados con la suspensión (listo y suspendido o bloqueado y suspendido, por citar algunos ejemplos clásicos), pero permite tener una visión panorámica del flujo de cambios de estado.

No es infrecuente que un sistema se detenga durante un breve instante de tiempo cuando está sometido a una excesiva carga de trabajo (fenómeno conocido como **lag**, también asociado a las fluctuaciones de las redes de datos). Esto puede deberse, entre otras razones, a que, en el establecimiento de prioridades del sistema operativo, alguna tarea ha acaparado los recursos del sistema durante una breve fracción de tiempo, dejando pausados al resto de procesos. Estas pausas se perciben especialmente durante la reproducción de contenidos multimedia o en la ejecución de videojuegos y son inevitables si la carga de trabajo del sistema excede a su capacidad.

## 2.2 Comunicación entre procesos

Por definición, los procesos de un sistema son elementos estancos. Cada uno tiene su espacio de memoria, su tiempo de CPU asignado por el planificador y su estado de los registros. No obstante, los procesos deben poder comunicarse entre sí, ya que es natural que surjan dependencias entre ellos en lo referente a las entradas y salidas de datos.

La comunicación entre procesos se denomina **IPC** (Inter Process Communication) y existen diversas alternativas para llevarla a cabo.

Algunas de estas alternativas son las siguientes;

- **Utilización de sockets.** Los sockets son mecanismos de comunicación de bajo nivel. Permiten establecer canales de comunicación de bytes bidireccionales entre procesos alojados en distintas máquinas y programados con diferentes lenguajes. Gracias a los sockets dos procesos pueden intercambiar cualquier tipo de información.
- **Utilización de flujos de entrada y salida.** Los procesos pueden interceptar los flujos de entrada y salida estándar, por lo que pueden leer y escribir información unos en otros. En este caso, los procesos deben estar relacionados previamente (uno de ellos debe haber arrancado al otro obteniendo una referencia al mismo).
- **RPC.** Llamada a procedimiento remoto (Remote Process Call, en inglés). Consiste en realizar llamadas a métodos de otros procesos que, potencialmente, pueden estar ejecutándose en otras máquinas. Desde el punto de vista del proceso que realiza la llamada, la ubicación de los procesos llamados es transparente.
- **Mediante el uso de sistemas de persistencia.** Consiste en realizar escrituras y lecturas desde los distintos procesos en cualquier tipo de sistema de persistencia, como los ficheros o las bases de datos. Pese a su sencillez, no se puede ignorar esta alternativa, ya que puede ser suficiente en múltiples ocasiones.
- **Mediante el uso de servicios proporcionados a través de internet.** Los procesos pueden utilizar servicios de transferencia de ficheros FTP, aplicaciones o servicios web, así como la tecnología cloud como mecanismos de conexión entre procesos que permiten el intercambio de información.

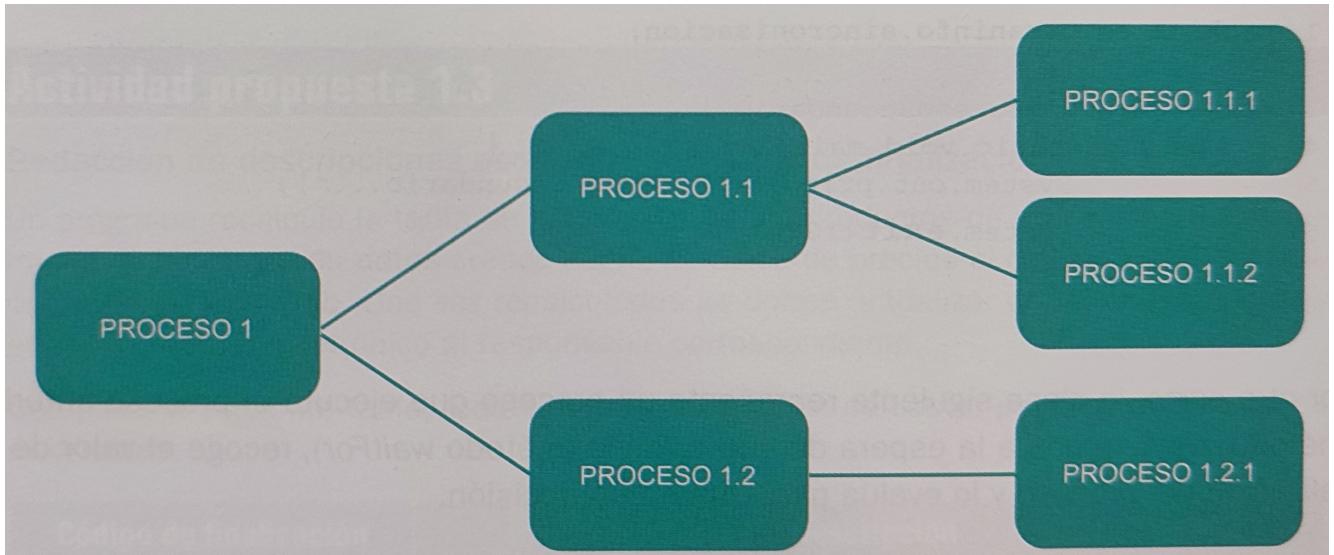
## 2.3. Sincronización entre procesos

Todos los sistemas en los que participan múltiples actores de manera concurrente están sometidos a ciertas condiciones que exigen que exista sincronización entre ellos. Por ejemplo, puede que sea necesario saber si un proceso ha terminado satisfactoriamente para ejecutar el siguiente que se encuentra en un flujo de procesos o, en caso de que haya ocurrido un determinado error, ejecutar otro proceso alternativo.

Es el planificador del sistema operativo el encargado de decidir en qué momento tiene acceso a los recursos un proceso, pero a nivel general, la decisión de crear y lanzar un proceso es humana y expresada a través de un algoritmo.

En la siguiente figura se muestra un posible ejemplo de flujo de ejecución de un conjunto de procesos. Las condiciones que determinan dicho flujo son las siguientes:

- ✓ El proceso "Proceso 1" se ejecuta inicialmente.
  - Si el código de finalización de "Proceso 1". es 0, se ejecuta el proceso "Proceso 1.1".
    - Si el código de finalización de "Proceso 1.1" es 0, se ejecuta el proceso "Proceso 1.1.1".
    - Si el código de finalización de "Proceso 1.1" es 1, se ejecuta el proceso "Proceso 1.1.2".
  - Si el código de finalización de Proceso 1 es 1, se ejecuta el proceso "Proceso 1.2".
    - Independientemente del código de finalización del proceso "Proceso 1.2". pero únicamente cuando haya finalizado, se ejecuta el proceso "Proceso 1.2.1".



Para gestionar un flujo de trabajo como el presentado en el ejemplo se necesita disponer de los siguientes mecanismos:

- **Ejecución.** Un mecanismo para ejecutar procesos desde un proceso.
- **Espera.** Un mecanismo para bloquear la ejecución de un proceso a la espera de que otro proceso termine.
- **Generación de código de terminación.** Un mecanismo de comunicación que permita indicar a un proceso cómo ha terminado la ejecución mediante un código.
- **Obtención de código de terminación.** Un mecanismo que permita a un proceso obtener el código de terminación de otro proceso.