

MP_0490
Programación de
servicios y procesos



TEMA 1: Programas, Ejecutables, Procesos y Servicios

En esta lección perseguimos los siguientes objetivos:

- 1 Comprender los conceptos básicos relacionados con la asignatura de "Programación de servicios y procesos".
- 2 Distinguir entre *programa*, *ejecutable*, *proceso* y *servicio*.
- 3 Distinguir entre los conceptos de *multiproceso* y *multitarea*.
- 4 Crear programas Java capaces de lanzar procesos.

Un programa es una secuencia de tareas cuya misión consiste en resolver un determinado problema de software.

Un programa está compuesto por...

- **Instrucciones.**
Conjunto de acciones que la computadora debe ejecutar para resolver el problema de software propuesto. Denominamos **algoritmo** al conjunto de instrucciones y la forma de ordenarlas para su ejecución.
- **Datos.**
Constituyen la información que las instrucciones requieren para llevar a cabo las acciones.

En función del objetivo que persiguen, los programas pueden dividirse en...

Programas del sistema

Forman parte del **sistema operativo** y permiten al usuario interactuar con la computadora. Algunos de estos programas sirven para gestionar los dispositivos, y otros para suministrar al usuario una interfaz para que pueda realizar las tareas básicas.

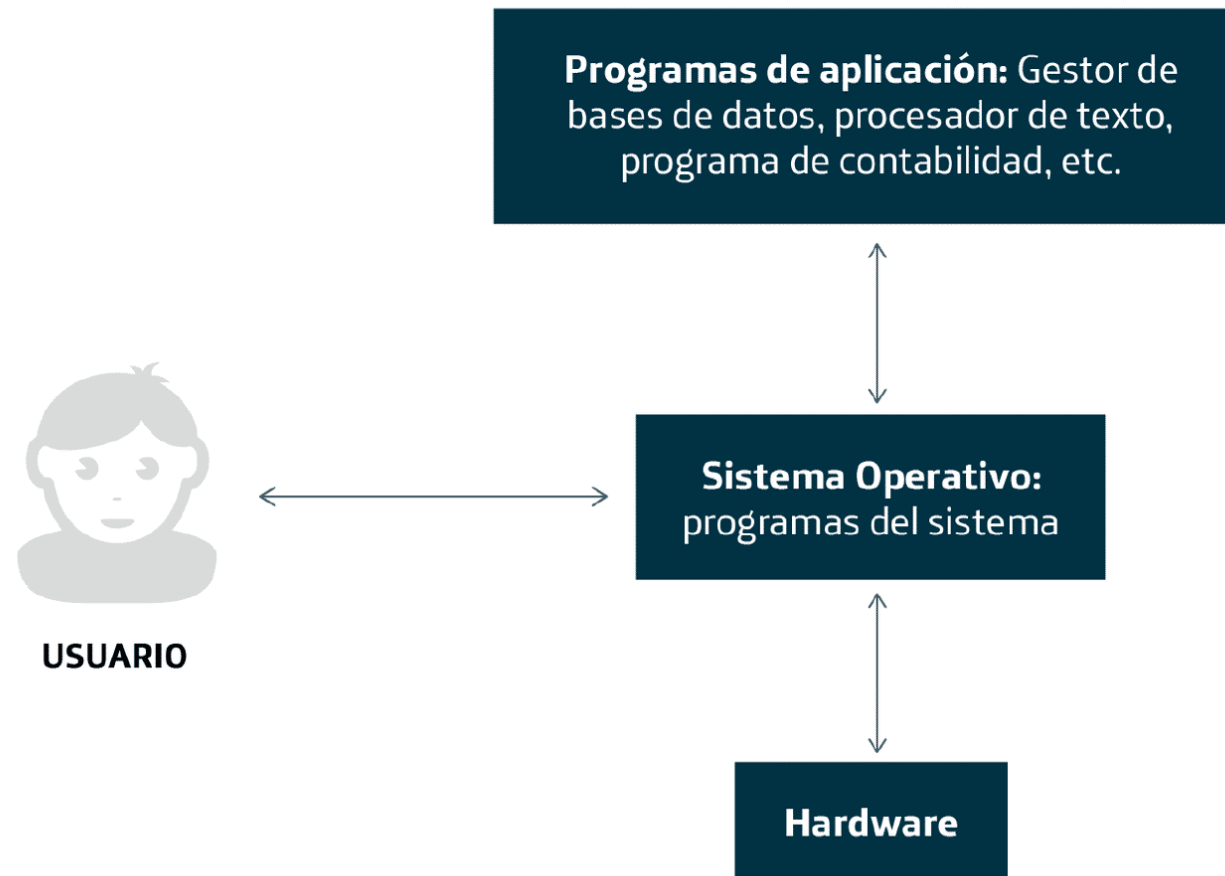
software de programación

Programas que forman parte del **software de programación**: utilizado por los programadores para escribir el código fuente de los programas, verificar la sintaxis, compilar, depurar, etc.

Programas de aplicación

Forman parte del software de aplicación y facilitan al usuario la ejecución de determinados trabajos. Ejemplos: procesadores de texto, hojas de cálculo, programas de contabilidad, etc.

El siguiente esquema refleja cómo el Sistema Operativo actúa como intermediario:



Procesos

Un proceso es un programa en ejecución. Puesto que un proceso está en ejecución, está consumiendo recursos del sistema.

El sistema operativo es el encargado de gestionar los procesos, creándolos y borrándolos cada vez que sea necesario.

La ejecución de un proceso implica:

- 1 La reserva de una determinada memoria de trabajo.
- 2 Carga de trabajo para el procesador, que tendrá que ir ejecutando las instrucciones incluidas en el proceso.
- 3 Cambios de estado del proceso, que se reflejan en los valores de los registros de la CPU.

Estados que puede atravesar un proceso:

Nuevo

El proceso acaba de ser creado.

En ejecución

El proceso está haciendo uso del procesador en este momento.

Bloqueado

El proceso no seguirá ejecutándose hasta que el usuario responda a un evento externo.
Ejemplo: el proceso es creado por un programa con una interfaz de usuario que está en espera de que el usuario haga clic en algún botón para realizar alguna acción.

Listo

El proceso no está en este momento en ejecución, aunque puede pasar a estarlo en cualquier momento.

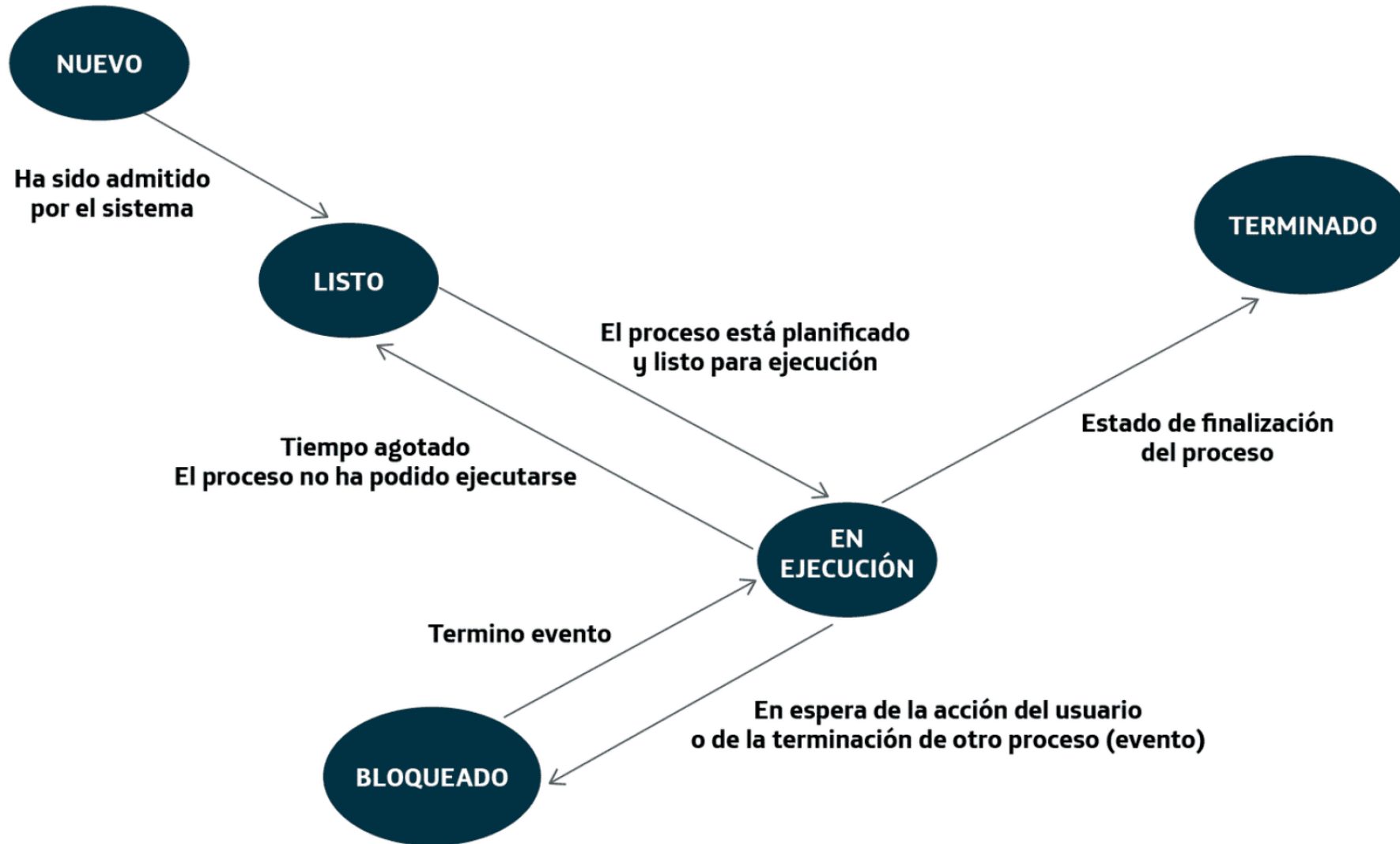
Estados que puede atravesar un proceso:

Terminado

El proceso ha finalizado su ejecución y se liberan los recursos del sistema que estaba consumiendo.

Cabe señalar que el estado de finalización de un proceso puede dar lugar a distintas circunstancias o sucesos, que se reflejan en el estado del proceso en el momento que va a ser eliminado:

- **Salida normal:** el proceso ha cumplido su misión y ha finalizado, a petición del usuario o de otro proceso que lo ha puesto en marcha.
- **Salida de error:** el proceso finaliza por un error en tiempo de ejecución.
- **Salida de error fatal:** similar al anterior, pero por causa más grave, por ejemplo: intento de escribir en un área de memoria ocupada.
- **Eliminado por otro proceso:** un caso típico podría ser cuando abortamos un proceso desde el administrador de tareas de Windows.



- **Programa:** se puede considerar un programa a toda la información (tanto código como datos) almacenada en disco de una aplicación que resuelve una necesidad concreta para los usuarios.
- **Proceso:** cuando un programa se ejecuta, podremos decir de manera muy simplificada que es un proceso. En definitiva, puede definirse “proceso” como un programa en ejecución. Este concepto no se refiere únicamente al código y a los datos, sino que incluye todo lo necesario para su ejecución. Esto incluye tres cosas:
 - *Un contador del programa:* algo que indique por dónde se está ejecutando.
 - *Una imagen de memoria:* es el espacio de memoria que el proceso está utilizando.
 - *Estado del procesador:* se define como el valor de los registros del procesador sobre los cuales se está ejecutando.

Es importante destacar que los procesos son entidades independientes, aunque ejecuten el mismo programa. De tal forma, pueden coexistir dos procesos que ejecuten el mismo programa, pero con diferentes datos (es decir, con distintas imágenes de memoria) y en distintos momentos de su ejecución (con diferentes contadores de programa).



EJEMPLO 1.1

Se pueden tener, por ejemplo, dos instancias del programa Microsoft Word ejecutándose a la vez, modificando cada una un fichero diferente. Para que los datos de uno no interfieran con los del otro, cada proceso se ejecuta en su propio espacio de direcciones en memoria permitiendo independencia entre los procesos.

Comprueba los procesos activos en Windows 10

El administrador de tareas de Windows te permite ver los procesos que el sistema operativo está gestionando.

Como práctica, inicia varios procesos, por ejemplo, puedes ejecutar la calculadora de Windows, el *bloc de notas* y el programa *paint*. Luego, pulsa **ctrl+alt+supr** para activar el administrador de tareas y ver los procesos.

Ejecutables

Llamamos ejecutable al archivo capaz de lanzar un programa, es decir, ponerlo en funcionamiento.

Un archivo ejecutable encierra un programa y, además, contiene la estructura e información necesarias para que el sistema operativo sea capaz de ejecutarlo. Su extensión en Windows es **.exe**.



Los programas `.exe` son ejecutados por el sistema operativo, por lo que, generalmente, tienen la problemática de que son dependientes de la plataforma hardware y software. Un ejecutable creado para correr en una máquina Intel con un sistema operativo Windows, no podrá funcionar en un Mac o, simplemente, en un sistema operativo Linux. Java soluciona este problema con su **bytecode** interpretado por la máquina virtual de Java.

Un programa ejecutable está **traducido a código máquina**, es decir, todas las instrucciones y los datos han sido traducidos al sistema binario, una codificación a base de ceros y unos.

No hay que olvidar que este es el único lenguaje que entiende la computadora. *A priori*, no podemos saber las instrucciones que están almacenadas en un archivo ejecutable, aunque existen programas desensambladores capaces de realizar la traducción desde código máquina a lenguaje ensamblador.

Instrucciones básicas

MOV: Mueve datos de un lugar a otro. Ejemplo: `MOV A, #55h` (Mueve el valor 55h al acumulador).

ADD: Suma el contenido de un registro con el acumulador. Ejemplo: `ADD A, R1` .

SUBB: Resta con acarreo. Ejemplo: `SUBB A, R2` .

INC/DEC: Incrementa o decrementa el valor de un registro. Ejemplo: `INC A` .

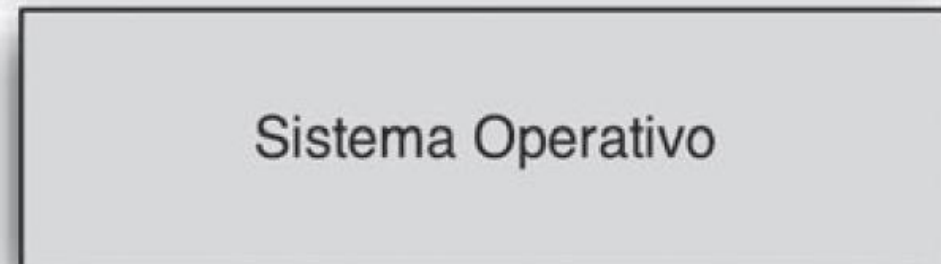
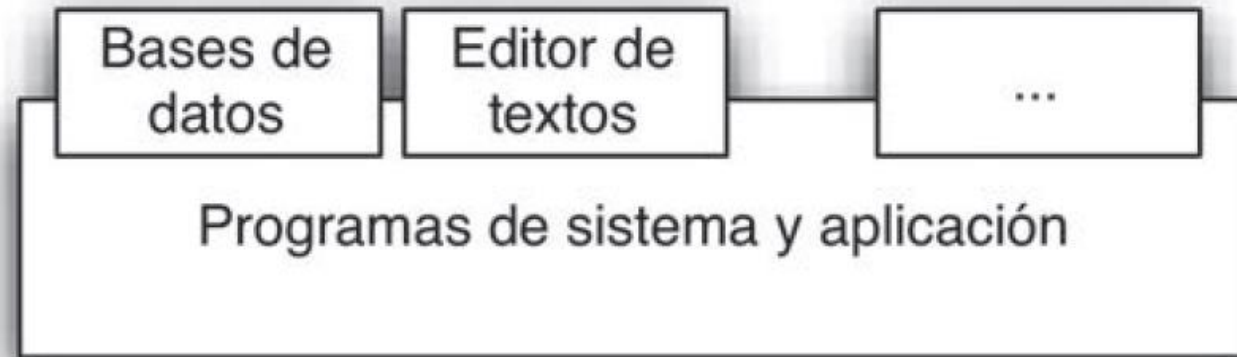
JMP: Salto incondicional a una etiqueta. Ejemplo: `JMP START` .

CJNE: Compara y salta si no es igual. Ejemplo: `CJNE A, #55h, LABEL` .

DJNZ: Decrementa y salta si no es cero. Ejemplo: `DJNZ R0, LOOP` .

```
ORG 0000h      ; Inicia en la dirección 0
    MOV P1, #0FFh ; Configura el puerto P1 como salida
    SETB P1.0    ; Enciende el LED conectado al pin P1.0
    SJMP $       ; Bucle infinito
END
```

- **Ejecutable:** un fichero ejecutable contiene la información necesaria para crear un proceso a partir de los datos almacenados de un programa. Es decir, llamaremos “ejecutable” al fichero que permite poner el programa en ejecución como proceso.
- **Sistema operativo:** programa que hace de intermediario entre el usuario y las aplicaciones que utiliza y el hardware del ordenador. Entre sus objetivos, se pueden destacar:
 - *Ejecutar los programas del usuario.* Es el encargado de crear los procesos a partir de los ejecutables de los programas y de gestionar su ejecución para evitar errores y mejorar el uso del computador.
 - *Hacer que el computador sea cómodo de usar.* Hace de interfaz entre el usuario y los recursos del ordenador, permitiendo el acceso tanto a ficheros y memoria como a dispositivos hardware. Esta serie de abstracciones permiten al programador acceder a los recursos hardware de forma sencilla
 - *Utilizar los recursos del computador de forma eficiente.* Los recursos del ordenador son compartidos tanto por los programas como por los diferentes usuarios. El sistema operativo es el encargado de repartir los recursos en función de sus políticas a aplicar.



- **Demonio:** proceso no interactivo que está ejecutándose continuamente en segundo plano, es decir, es un proceso controlado por el sistema sin ninguna intermediación del usuario. Suelen proporcionar un servicio básico para el resto de procesos.



¿SABÍAS QUE...?

La palabra "demonio" fue usada en 1963 por primera vez en el área de la informática, para denominar a un proceso que realizaba en segundo plano *backups* en unas cintas. El nombre proviene de Fernando J. Corbató, director del proyecto MAC del MIT basándose en el famoso demonio de James Maxwell. Este demonio era un elemento biológico que residía en medio de un recipiente dividido en dos, lleno de moléculas. El demonio se encargaba de permitir, dependiendo de la velocidad de la molécula, que estas pasaran de un lado al otro. Los demonios actúan de forma similar ya que están continuamente vigilando y realizando acciones en función de las necesidades del sistema. En Windows, los demonios se denominan "servicios", estableciendo una clara relación entre los procesos no interactivos, y los servicios que se estudiarán en el Capítulo 4.

Los ejecutables Java

En Java, los programas fuentes (ficheros *.java*) deben ser compilados, pero no para crear código máquina directamente, sino para crear un código que queda a medio camino entre el código fuente y el código máquina (fichero *.class*), al que denominamos *bytecode* (o *bycode*).

El *bytecode* es un tipo de codificación, parecida al ensamblador, que deberá ser interpretada por una máquina virtual, línea a línea, según se vaya ejecutando. El encargado de esta interpretación línea a línea es el compilador JIT (*just-in-time*) que va traduciendo el *bytecode* instrucción a instrucción para el microprocesador y sistema operativo en el que nos encontremos. Este proceso es lo que hace que Java sea independiente de la plataforma.

Programa.java

COMPILACIÓN

Programa.class

INTÉRPRETE



WINDOWS



MACINTOSH



UNIX

Multiproceso

Se denomina multiproceso a la ejecución simultánea de varios procesos.

Para que exista **multiprocesamiento** en el sentido estricto, es necesario un ordenador que contenga dos o más procesadores, lo que permitirá la ejecución simultánea de varios programas que compartirán el uso de la memoria central y de los dispositivos periféricos. Además de varios procesadores, será necesario un sistema operativo capaz de gestionar el multiprocesamiento.

¿Qué es la programación concurrente?

Un **programa concurrente** es el que lanza varios procesos al mismo tiempo, que colaboran entre sí para realizar una tarea compartida.

Pongamos un ejemplo: queremos crear un programa que realice un dibujo despacio, de modo que el usuario vaya viendo los trazos. Imagina que el dibujo contendrá un árbol y una casa. Si queremos que la casa y el árbol se vayan trazando simultáneamente, será necesaria la ejecución de dos procesos concurrentes, aunque ambos colaboren para lograr el mismo fin, la confección del dibujo.



EJEMPLO 1.3

A la vez que se modifica un documento en Microsoft Word, se puede estar escuchando música en iTunes y navegando a través de la red con Google Chrome. Si los cambios entre los procesos se producen lo suficientemente rápido, parece que todo se ejecuta al mismo tiempo, y así la música se escucha sin cortes.

Y, ¿qué son la programación paralela y la programación distribuida?

- La **programación paralela** se ejecuta en un único ordenador, ya se disponga de varios procesadores o uno solo.
- La **programación distribuida** se ejecuta en varios ordenadores, que se comunican en red y colaboran para realizar una tarea común.

- Varios núcleos en un mismo procesador (multitarea). La existencia de varios núcleos o *cores* en un ordenador es cada vez mayor, apareciendo en *Dual Cores*, *Quad Cores*, en muchos de los modelos *i3*, *i5* e *i7*, etc. Cada núcleo podría estar ejecutando una instrucción diferente al mismo tiempo. El sistema operativo, al igual que para un único procesador, se debe encargar de planificar los trabajos que se ejecutan en cada núcleo y cambiar unos por otros para generar multitarea. En este caso todos los *cores* comparten la misma memoria por lo que es posible utilizarlos de forma coordinada mediante lo que se conoce por **programación paralela**.
- La programación paralela permite mejorar el rendimiento de un programa si este se ejecuta de forma paralela en diferentes núcleos ya que permite que se ejecuten varias instrucciones a la vez. Cada ejecución en cada *core* será una tarea del mismo programa pudiendo cooperar entre sí. El concepto de “tarea” (o “hilo de ejecución”) se explicará en más profundidad a lo largo del Capítulo 2. Y, por supuesto, se puede utilizar conjuntamente con la programación concurrente, permitiendo al mismo tiempo multiprogramación.



EJEMPLO 1.4

Mientras se está escribiendo un documento en Microsoft Word, al mismo tiempo se puede estar ejecutando una tarea del mismo programa que comprueba la ortografía.

- Varios ordenadores distribuidos en red. Cada uno de los ordenadores tendrá sus propios procesadores y su propia memoria. La gestión de los mismos forma parte de lo que se denomina **programación distribuida**.

La programación distribuida posibilita la utilización de un gran número de dispositivos (ordenadores) de forma paralela, lo que permite alcanzar elevadas mejoras en el rendimiento de la ejecución de programas distribuidos. Sin embargo, como cada ordenador posee su propia memoria, imposibilita que los procesos puedan comunicarse compartiendo memoria, teniendo que utilizar otros esquemas de comunicación más complejos y costosos a través de la red que los interconecte. Se explicará en más profundidad en el Capítulo 3.

Funcionamiento básico del S.O.

La parte central que realiza la funcionalidad básica del sistema operativo se denomina *kernel*. Es una parte software pequeña del sistema operativo, si la comparamos con lo necesario para implementar su interfaz (y más hoy en día que es muy visual). A todo lo demás del sistema se le denomina **programas del sistema**. El *kernel* es el responsable de gestionar los recursos del ordenador, permitiendo su uso a través de llamadas al sistema.

En general, el *kernel* del sistema funciona en base a interrupciones. Una **interrupción** es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una rutina que trate dicha interrupción. Esta rutina será dependiente del sistema operativo. Es importante destacar que mientras se está atendiendo una interrupción, se deshabilita la llegada de nuevas interrupciones. Cuando finaliza la rutina, se reanuda la ejecución del proceso en el mismo lugar donde se quedó cuando fue interrumpido.

Es decir, el sistema operativo no es un proceso demonio propiamente dicho que proporcione funcionalidad al resto de procesos, sino que él solo se ejecuta respondiendo a interrupciones. Cuando salta una interrupción se transfiere el control a la rutina de tratamiento de la interrupción. Así, las rutinas de tratamiento de interrupción pueden ser vistas como el código propiamente dicho del *kernel*.

Instalar y configurar intellj

1. Descargar e Instalar IntelliJ IDEA

Si no lo tienes instalado, puedes descargar IntelliJ IDEA desde [su sitio oficial](#). Asegúrate de elegir la versión **Community** si buscas una opción gratuita.

Instalar y configurar intellj

2. Instalar JDK (Java Development Kit)

Si no tienes el JDK instalado, necesitarás instalarlo antes de comenzar a programar en Java.

Ve a la página de descargas de [Oracle JDK](#) o usa una alternativa como [OpenJDK](#).

Descarga e instala la versión que prefieras (te recomiendo la versión 18 o superior)

ACTUALMENTE LA ULTIMA VERSION ES 23

Instalar y configurar intellj

Después de instalar, asegúrate de que la variable de entorno `JAVA_HOME` está configurada correctamente. Puedes verificar la instalación abriendo una terminal o símbolo de sistema y escribiendo:

```
java -version
```



Instalar y configurar intellj





Variables del sistema

Variable	Valor
DriverData	C:\Windows\System32\Drivers\DriverData
INTEL_DEV_REDIST	C:\Program Files (x86)\Common Files\Intel\Shared Libraries\
JAVA_HOME	C:\Program Files\Java\jdk-23
MIC_LD_LIBRARY_PATH	%INTEL_DEV_REDIST%\compiler\lib\mic
NUMBER_OF_PROCESSORS	16
NVM_HOME	C:\Users\CESAR\AppData\Roaming\nvm

Editar variable de entorno

C:\Program Files\nodejs
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\CESAR\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\CESAR\AppData\Local\GitHubDesktop\bin
C:\Users\CESAR\AppData\Local\Microsoft\WinGet\Packages\Schniz...
%NVM_HOME%
%NVM_SYMLINK%
%USERPROFILE%\dotnet\tools
C:\MinGW\bin
%JAVA_HOME%\bin

To Run code, press **Mayús** **F10** or click the  icon in the gutter.  1

```
3  public class Main {  
4      public static void main(String[] args) {  
        Press Alt Enter with your caret at the highlighted text to see how  
        IntelliJ IDEA suggests fixing it.  
7         System.out.printf("Hello and welcome!");  
8  
9         for (int i = 1; i <= 5; i++) {  
            Press Mayús F9 to start debugging your code. We have set one   
            breakpoint for you, but you can always add more by pressing  
            Ctrl F8.  
13  System.out.println("i = " + i);  
        }  
    }
```

Instalar Plugins (Opcional)

Si quieres mejorar tu experiencia en Java, puedes instalar algunos plugins útiles como **Lombok**, **CheckStyle**, o cualquier otro que te ayude a programar de manera más eficiente. Para instalarlos:

Ve a **File > Settings > Plugins**.

Busca el plugin que quieras instalar y haz clic en **Install**.

Lanzar procesos desde Java

Java tiene sus mecanismos para comunicarse con el sistema operativo y darle la orden de iniciar un proceso.

Veamos un ejemplo, utilizando la clase *ProcessBuilder* situada en el paquete *java.lang*.

❗ Recuerda que el paquete *java.lang* es el único que es importado de manera predeterminada en todos los proyectos.

La clase que representa un proceso en Java es la clase *Process*. Los métodos de *ProcessBuilder.start()* y *Runtime.exec()* crean un proceso nativo en el sistema operativo subyacente donde se está ejecutando la JVM y devuelven un objeto Java de la clase *Process* que puede ser utilizado para controlar dicho proceso.

- **Process ProcessBuilder.start():** inicia un nuevo proceso utilizando los atributos indicados en el objeto. El nuevo proceso ejecuta el comando y los argumentos indicados en el método **command()**, ejecutándose en el directorio de trabajo especificado por **directory()**, utilizando las variables de entorno definidas en **environment()**.
- **Process Runtime.exec(String[] cmdarray, String[] envp, File dir):** ejecuta el comando especificado y argumentos en *cmdarray* en un proceso hijo independiente con el entorno *envp* y el directorio de trabajo especificado en *dir*.

¿Qué es ProcessBuilder?

ProcessBuilder es una clase en Java que permite a los programas ejecutar procesos del sistema operativo.

Ofrece una interfaz más flexible que el método

- *Runtime.exec()*, permitiendo configurar el entorno, redirigir la entrada/salida de los procesos y más.

Características Principales:

Ejecuta comandos externos o scripts desde programas Java.

Permite redirigir la entrada, salida y error estándar.

Configura variables de entorno para los procesos.

Controla la secuencia de ejecución de varios procesos.

¿Cuándo usarlo?

Para ejecutar comandos del sistema operativo, lanzar scripts o procesos externos desde una aplicación Java.

Útil en automatización, tareas administrativas, o cuando se necesita interactuar con el sistema subyacente desde un programa.

Sintaxis Básica:

1.Crear y configurar el proceso:

```
ProcessBuilder pb = new ProcessBuilder("comando", "argumento1", "argumento2");
```

```
pb.redirectErrorStream(true); // Redirige errores al stream de salida
```

2.Ejecutar y obtener el resultado:

```
Process process = pb.start();
```

```
BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
```

```
String line;
```

```
while ((line = reader.readLine()) != null)
```

```
{
```

```
    System.out.println(line);
```

```
}
```

Ejemplo de Uso:

```
// Ejecutar un comando "ping" en la terminal del sistema operativo
ProcessBuilder processBuilder = new ProcessBuilder("ping", "google.com");
processBuilder.redirectErrorStream(true);

try {
    Process process = processBuilder.start();
    BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Lancemos la calculadora de Windows usando este comando de java.

Este sencillo programa lanza la *calculadora* de Windows.

```
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/calc.exe");
        try {
            proceso.start();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



Puede que la ruta "C:/Windows/System32/calc.exe" sea diferente en tu equipo. Comprueba la ruta donde se encuentra el archivo *calc.exe* y corrígela, si es necesario.

Hasta que no se ejecuta la sentencia *proceso.start()*, realmente no se lanza el proceso. Una vez que la *calculadora* de Windows se ha iniciado, nuestro programa ha finalizado. El proceso lleva consigo una operación de entrada/salida, ya que implica el acceso al fichero *calc.exe*, lo que podría ocasionar una excepción de tipo *IOException*.

Si queremos que nuestro programa se mantenga en ejecución a la espera de que termine el proceso iniciado, podemos utilizar el método *waitFor()*, lo que podría provocar una excepción de tipo *InterruptedException* en caso de que el proceso sea abortado por una situación de error.

```
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/calc.exe");
        try {
            Process p = proceso.start();
            p.waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Ahora, el programa no termina hasta que no se cierra la *calculadora* de Windows; lo demuestra el hecho de que no aparece el mensaje *Proceso lanzado con éxito* hasta ese momento. El método *start()* devuelve un objeto de tipo *Process* que dispone del método *waitFor()*. Nuestro programa queda detenido en la siguiente línea:

```
import java.io.IOException;
public class Principal
{
    public static void main(String[] args)
    {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/calc.exe");
        try {
            Process p = proceso.start().waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Lanzar procesos con argumentos

Algunos procesos requieren argumentos. El constructor de la clase *ProcessBuilder* admite la entrada de dichos argumentos.

El formato del constructor de *ProcessBuilder* tiene el siguiente formato:

```
new ProcessBuilder("ruta programa", |argumento1, argumento2, .....,  
argumentoN);
```



```
import java.io.IOException;
public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/notepad.exe","agenda.txt");
        try {
            proceso.start().waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Cambio del directorio de trabajo del proceso lanzado

En el programa anterior, hemos lanzado *notepad.exe* y dicho programa tiene un directorio de trabajo, es decir, un espacio donde guardará, de manera predeterminada, los archivos que se creen dentro de él. Como no hemos especificado ningún directorio de trabajo, ha utilizado como ubicación predeterminada la del proyecto Eclipse. Es posible cambiar dicho comportamiento, utilizando el método *directory()* antes de lanzar el proyecto.

```
import java.io.File;
import java.io.IOException;

public class Principal {
    public static void main(String[] args) {
        ProcessBuilder proceso;
        proceso = new ProcessBuilder("C:/Windows/System32/notepad.exe", "agenda.txt");
        try {
            proceso.directory(new File("C:\\tmp"));
            proceso.start().waitFor();
            System.out.println("Proceso lanzado con éxito");
        } catch (IOException | InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Operaciones de entrada y salida en los procesos

Cualquier programa típico lleva consigo una entrada de datos, un proceso o algoritmo, y una salida de resultados. Además, podrían producirse salidas de error.

El lanzamiento de un proceso desde un programa Java podría requerir de la gestión de estas entradas y salidas, lo que se efectúa a través de los siguientes métodos:

1. ***ProcessBuilder.redirectInput(File fich);***

Recibe un objeto *File*, que apunta al fichero que contiene los datos de entrada para que el proceso pueda realizar su tarea. De aquí recoge las entradas estándar (*System.in*).

2. ***ProcessBuilder.redirectOutput(File fich);***

Recibe un objeto *File*, que apunta al fichero donde se escribirán los datos de salida del proceso, es decir, los resultados. En este fichero escribe las salidas estándar (*System.out*).

3. ***ProcessBuilder.redirectError(File fich);***

Recibe un objeto *File* que apunta al fichero donde se escribirán los datos de salida de error del proceso. En este fichero escribe las salidas de error (*System.err*).

MUCHAS GRACIAS POR VUESTRA ATENCION!