



# Apuntes de clase

## Unidad 1 Programación de Servicios

### Tipos de sistemas:

**Monotarea:** Sólo puede ejecutarse una tarea a la misma vez. En el sistema monotarea no puede alterarse el tiempo de gestión del procesador entre varios programas distintos.

**Multitarea:** Se subdivide en distintas tareas que pueden ser ejecutadas de forma simultánea. En realidad se trata de una simultaneidad aparente, puesto que el microprocesador dedica a cada trabajo una fracción de segundo; algo, por otra parte, inapreciable para el ser humano. Los recursos y el tiempo pueden ser asignados de forma igualitaria o estableciendo correcciones según un orden de prioridades.

### Tipos de lenguaje

#### Compilados:

El código fuente se traduce a código binario. El resultado del proceso de compilación es un programa ejecutable solamente en el sistema operativo de destino y en la versión correspondiente. A pesar de esto, si el lenguaje es estándar, el mismo código fuente puede ser compilado para distintos sistemas sin necesitar modificaciones. **C y C++**

### **Interpretados:**

No se compilan para generar un programa binario ejecutable directamente sobre el sistema operativo. Este tipo de programa se ejecuta por parte del intérprete, que si está programado para un sistema operativo concreto. El mismo código fuente se puede llevar a cualquier ordenador que tenga instalado el intérprete. **PHP y Python**

### **Java:**

Existe un punto intermedio entre ambos, su máximo exponente es Java. Es un lenguaje de programación compilado, pero realiza la compilación para su propia JVM(Java Virtual Machine), encargada de traducir el código compilado en Java al lenguaje del sistema operativo durante la ejecución de los programas.

## **Tipos de procesamiento**

### **Procesamiento concurrente.**

En la computación concurrente, los tiempos de CPU se reparten entre los distintos procesos según una planificación dirigida por el SO.

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.**

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, haciendo posible ejecutar varias instrucciones en paralelo.

Es aquel en el que las divisiones de un proceso se ejecutan de manera simultánea en los diversos núcleos de ejecución de un procesador o en diversos procesadores.

---

## Procesos

Es un archivo que está en ejecución y bajo el control del sistema operativo: Puede atravesar diversas etapas en su ciclo de vida:

- Nuevo: No es técnicamente un estado, es cuando el proceso se añade.
- En ejecución. Se considera programa cuando está en ejecución.
- Bloqueado: el proceso no puede hacer nada hasta que ocurra un evento externo.
- Listo: el proceso está listo para ejecutarse cuando el SO le pase el turno.
- Terminado: Igual que el nuevo, no es técnicamente un estado, es cuando finaliza.

PID: Identificador del proceso.

## Servicios

Es un programa cuya ejecución se realiza en segundo plano y que no requiere la interacción del usuario. Normalmente, se arranca de manera automática con el SO y está en constante ejecución.

## Hilos

Un programa básico está compuesto por una serie de sentencias que se ejecutan de manera secuencial y síncrona.

En muchos casos, es necesaria esta secuencialidad y sincronía, en otros casos, 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 pueden ejecutarse simultáneamente gracias al procesamiento paralelo.

## Bifurcación o fork

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

## Procesos: conceptos teóricos:

Cada proceso está compuesto por:

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

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.

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

Los pasos para realizar 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.

## Gestión y estados de los procesos.

Los procesos necesitan recursos y estos son limitados. Por eso el sistema operativo cuenta con 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. 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.

## Comunicación entre procesos

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

- **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 diferente lenguaje.
- **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(deben estar relacionados previamente).
- **RPC.** Llamada a procedimiento remoto(Remote Process Call). Consiste en realizar llamadas a métodos de otros procesos que pueden estar ejecutándose en otras máquinas. En Java, este tipo de llamada se realiza mediante la tecnología conocida como RMI(Remote Method Invitación), equivalente a RPC, pero orientada a objetos.
- **Mediante el uso de sistemas de persistencia.** Consiste en realizar escrituras y lecturas desde los distintos procesos en cualquier sistema de persistencia(base de datos, fichero, etc).
- **Mediante el uso de servicios proporcionados a través de internet.** Pueden utilizar servicios de transferencia de archivos FTP, aplicaciones o servicios web, o la tecnología cloud como mecanismo de conexión para el intercambio de información

---

## Sincronización entre procesos

Todos los sistemas en los que participan múltiples actores de manera concurrente exigen que exista una sincronización entre ellos.

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.

## Programación de aplicaciones multiproceso en Java

Si se dispone de un sistema compuesto por un conjunto de procesos que deben ejecutarse de manera individual, pero que tienen dependencias entre sí, se necesita disponer de un mecanismo de gestión y coordinación.

Las necesidades que se deben satisfacer para poder programar un sistema basado en la ejecución de múltiples procesos son las siguientes:

- Poder arrancar un proceso y hacerle llegar los parámetros de ejecución
- Poder quedar a la espera de que el proceso termine.
- Poder recoger el código de finalización de ejecución para determinar si el proceso se ha ejecutado correctamente o no.
- Poder leer los datos generados por el proceso para su tratamiento.

En Java, la creación de un proceso se puede realizar de dos maneras diferentes:

- Clase `java.lang.Runtime`.
- Clase `java.lang.ProcessBuilder`.

## Creación de procesos con Runtime.

Toda aplicación Java tiene una única instancia de la clase Runtime que permite que la propia aplicación interactúe con su entorno de ejecución a través del método estático `getRuntime`. Este método proporciona un canal de comunicación entre la aplicación y su entorno, posibilitando la interacción con el sistema operativo a través del método `exec`.

Este código genera un proceso en Windows indicando al SO que ejecute el bloc de notas.

```
Runtime.getRuntime().exec("Notepad.exe");
```

En muchos casos, los procesos necesitan parámetros para iniciarse. El método `exec` puede recibir una cadena de caracteres con los diferentes parámetros.

```
Runtime.getRuntime().exec("Notepad.exe notas.txt");
```

El siguiente nivel consiste en gestionar el proceso. Para ello, se debe obtener la referencia a la instancia de la clase `Process` proporcionada por el método `exec`:

```
String[] infoProceso = {"Notepad.exe", "notas.txt"};  
Process proceso = Runtime.getRuntime().exec(infoProceso);
```

Si se necesita esperar a que el proceso ejecutado termine y conocer el estado en que ha finalizado dicha ejecución, se puede utilizar el método `waitFor`.

```
1 String[] infoProceso = {"Notepad.exe", "notas.txt"};  
2 Process proceso = Runtime.getRuntime().exec(infoProceso);  
3 int codigoRetorno = proceso.waitFor();  
4 System.out.println("Fin de la ejecución:" + codigoRetorno);
```

La clase `Process` representa al proceso en ejecución y permite obtener información sobre este. Los principales métodos que proporciona dicha clase:

- `destroy()` : destruye el proceso sobre el que se ejecuta.
- `exitValue()`: Destruye el valor de retorno del proceso cuando este finaliza. Sirve para controlar el estado de la ejecución.
- `getErrorStream()`: Proporciona un `InputStream` conectado a la salida de error.
- `getInputStream()`: Proporciona un `InputStream` conectado a la salida normal.
- `getOutputStream()`: Proporciona un `OutputStream` conectado a la entrada normal.
- `isAlive()`: Determina si un proceso está o no en ejecución.
- `waitFor()`: Detiene la ejecución del programa que lanza el proceso a la espera de que este último termine.

## Creación de procesos con Builder

La clase `ProcessBuilder` permite, al igual que `Runtime`, crear procesos.

La creación más sencilla de un proceso se realiza con un único parámetro en el que se indica el programa a ejecutar.

```
new ProcessBuilder("Notepad.exe")
```

El constructor de `ProcessBuilder` admite parámetros:

```
new ProcessBuilder("Notepad.exe", "datos.txt").start();
```

Al igual que ocurre con el método `exec` de la clase `Runtime`, el método `start` de `Process-Builder` proporciona un proceso como retorno, lo que posibilita la sincronización y gestión de este:

```
1 Process proceso = new
2   ProcessBuilder("Notepad.exe", "datos.txt").start();
3 int valorRetorno = proceso.waitFor();
4 System.out.println("Valor retorno:" + valorRetorno);
```



---

### Métodos ProcessBuilder:

- `start()`: Inicia un nuevo proceso usando los atributos especificados.
  - `command()`: permite obtener o asignar el programa y los argumentos de la instancia de `processbuilder`
  - `directory()`: permite obtener o asignar el directorio de trabajo del proceso
  - `environment()`: proporciona información sobre el entorno de ejecución del proceso.
  - `redirectError()`: permite determinar el destino de a la salida de errores
  - `redirectInput()`: Permite determinar el origen de la entrada estándar
  - `redirectOutput()`: Permite determinar el destino de la salida estándar.
-

## Actividades resueltas.

### Actividad resuelta 1.1

#### Conexión de procesos Java y Python con lectura de salida

Desde un programa escrito en Java, ejecutar un programa escrito en Python y leer los datos que genera como salida.

#### Solución

##### Proceso iniciador (Java). Fichero App.java

```

1 package es.paraninfo.gestorprocesos;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6
7 public class App {
8
9     public static void main(String[] args) {
10         try {
11
12             Process proceso = new
13                 ProcessBuilder("Python",
14                     "C:\\\\PSP_Unidad1\\\\proceso_python\\\\proceso_python.py").start();
15
16             BufferedReader br = new
17                 BufferedReader(new
18                     InputStreamReader(proceso.getInputStream()));
19             proceso.waitFor();
20             int exitStatus = proceso.exitValue();
21             System.out.println("Retorno:" + br.readLine());
22             System.out.println("Valor de la salida:" + exitStatus);
23         } catch (IOException | InterruptedException e) {
24             e.printStackTrace();
25         }
26     }
27 }

```

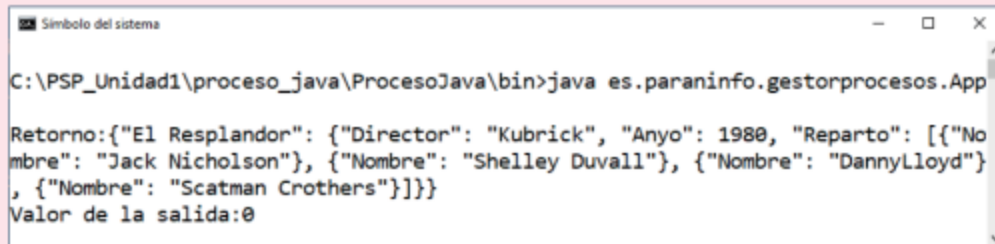
##### Proceso iniciado (Python). Fichero proceso\_python.py

```

1 import json
2 import sys
3
4 pelicula = {"El Resplandor":
5     {
6         "Director": "Kubrick",
7         "Anyo": 1980,
8         "Reparto": [
9             {"Nombre": "Jack Nicholson"},
10            {"Nombre": "Shelley Duvall"},
11            {"Nombre": "Danny Lloyd"},
12            {"Nombre": "Scatman Crothers"}
13        ]
14    }
15 }
16
17 print(json.dumps(pelicula))
18 sys.exit(0)

```

### Ejecución y salida:



```
Símbolo del sistema
C:\PSP_Unidad1\proceso_java\ProcesoJava\bin>java es.paraninfo.gestorprocesos.App
Retorno:{"El Resplandor": {"Director": "Kubrick", "Anyo": 1980, "Reparto": [{"Nombre": "Jack Nicholson"}, {"Nombre": "Shelley Duvall"}, {"Nombre": "DannyLloyd"}, {"Nombre": "Scatman Crothers"}]}}
Valor de la salida:0
```

Como se puede observar, desde el programa escrito en Java (proceso iniciador) se está ejecutando el intérprete de Python (proceso iniciado) que recibe un parámetro (el nombre del programa Python a ejecutar). El proceso iniciador queda a la espera de que el proceso iniciado termine para leer la salida que este proporciona a través del *stream* de salida, en este caso una cadena de caracteres que contiene un fichero JSON.