Programación de Procesos - Java

En este apartado vamos a ver cómo es posible escribir programas en lenguaje Java para trabajar con Procesos.

Sincronización de Procesos en Java

Java permite trabajar directamente sobre **Procesos** mediante una serie de clases y métodos.

Mecanismo	Clase	Método
Ejecución	Runtime	exec ()
Ejecución	ProcessBuilder	start ()
Espera	Process	waitFor ()
Terminación	System	exit (valor)
Esperar Terminación	Process	waitFor ()

Vamos a ver un **Ejemplo**. Definimos en primer lugar una Clase con **un main ()** que realiza una serie de tareas. Coloca esa clase en un paquete "es". Si se lanzase esta clase, se ejecutaría el programa (se crearía un nuevo **Proceso**). Sin embargo, algo ha ido mal y se va a retornar el código de error 103.

```
public class ProcesoSecundario {

    public static void main(String[] args) {
        System.out.println("Proceso secudnario....");

        // -- El proceso hace cosas pero genera error --//

        System.exit(103);
    }

}
```

Ahora, vamos a crear otra clase en ese mismo paquete "es" con **otro main ()** que va a ejecutar al ProcesoSecundario, va a esperar a que termine, va a recoger el valor de su terminación. En base a ese valor, va a tomar una u otra decisión.

```java
public class ProcesoPrincipal {

    public static void main(String[] args) {
        System.out.println("Ejecutamos el proceso secundario....");

        try {

            // Proceso que queremos lanzar
            String [] processInfo = {"java", "es.ProcesoSecundario"};
            Process process = Runtime.getRuntime().exec(processInfo);

            // Esperamos a que termine (Se bloquea la ejecución del proceso principal)
            int valorRetorno = process.waitFor();

            // ¿Qué ha pasado con el proceso secundario?
            if (valorRetorno == 0) {
                System.out.println("Proceso secundario finalizado con éxito");
            } else {
                System.out.println("El proceso secundario ha fallado");
                System.out.println("Código de error: " + valorRetorno);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Obviamente, ese ejemplo es bastante simple. En esencia es similar a la sincronización que se hacía en C mediante wait (). Podría ser útil para crear una aplicación de gestión con un interfaz de ventana que permitiese lanzar uno u otro proceso a conveniencia.

Programación Multiproceso en Java (Runtime)

Toda aplicación Java tiene una única instancia de la clase **Runtime** que permite que la propia aplicación interactúe con su entorno de ejecución. De esta forma, la aplicación puede interactuar con el Sistema Operativo mediante el método **exec**.

Métodos de la clase Runtime	Descripción
destroy ()	Destruye el Proceso
exitValue ()	Devuelve el valor de retorno
getErrorStream ()	Proporciona un InputStream conectado a la salida de error del Proceso
getInputStream ()	Proporciona un InputStream conectado a la salida del Proceso
getOutputStream ()	Proporciona un OutputStream conectado a la entrada de error del Proceso
isAlive ()	¿El Proceso está en ejecución?
waitFor ()	Espera a que el Proceso termine

Vamos a ver un **ejemplo**. Lanzamos el Notepad desde nuestro Proceso Java.

```java
public class ExecNotepad {

    public static void main(String[] args) {
        System.out.println("Vamos a lanzar el Notepad...");

        String [] infoProceso = {"Notepad.exe"};

        try {
            // Ejecutamos el nuevo Proceso
            Process proceso = Runtime.getRuntime().exec(infoProceso);

            // Esperamos a que finalice el proceso. Cogemos el codigo de retorno
            int codigoRetorno = proceso.waitFor();

            System.out.println("Fin del Proceso con el codigo " + codigoRetorno);

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Programación Multiproceso en Java (ProcessBuilder)

Al igual que **Runtime**, es posible crear procesos con **ProcessBuilder**.

Métodos de ProcessBuilder	Descripción
start ()	Inicia un Proceso
command ()	Obtiene o asigna el programa
directory ()	Obtiene o asigna el directorio de trabajo al Proceso
environment ()	Retorna información de entrono de ejecución
redirectError ()	Determina el destino de la salida de errores
redirectInput ()	Determina el destino de la entrada estándar
redirectOutput ()	Determina el destino de la salida estándar

Vamos a ver un **ejemplo**. Lanzamos el Notepad desde nuestro Proceso Java.

```java
public class ProcessBuilderNotepad {

    public static void main(String[] args) {
        System.out.println("Vamos a lanzar el Notepad...");

        String infoProceso = "Notepad.exe";
        try {

            // Preparamos el Proceso
            ProcessBuilder processBuilder = new ProcessBuilder (infoProceso);

            // Obtenemos informacion del Proceso
            Map <String, String> environment = processBuilder.environment();
            System.out.println("Numero de Procesadores: " + environment.get("NUMBER_OF_PROCESSORS"));

            // Ejecutamos el Proceso
            Process proceso = processBuilder.start();

            // Esperamos a que finalice el proceso. Cogemos el codigo de retorno
            int codigoRetorno = proceso.waitFor();
            System.out.println("Fin del Proceso con el codigo " + codigoRetorno);

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Ejercicio 9

Crea un programa en Java que obtenga todas las variables de entorno de tu equipo y las muestre por consola.