MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Gestión de procesos

03

ÍNDICE

OBJETIVOS

Crear y lanza procesos.

Terminar procesos.

Conocer los mecanismos de comunicación entre procesos.

Conocer los mecanismos de sincronización entre procesos.

/ 1. Introducción y contextualización práctica

En esta unidad, veremos cómo podemos realizar una gestión de procesos mediante nuestros programas en Java.

Estudiaremos cómo crear un proceso, lanzarlo y ejecutarlo. Además, veremos cómo podemos ejecutar una clase mediante un proceso independiente.

También aprenderemos cómo podemos hacer que un proceso finalice su ejecución mediante código.

Seguidamente, comprobaremos cómo podemos hacer que varios procesos que hemos lanzado se comuniquen entre ellos.

Por último, veremos cuáles son los mecanismos de sincronización de los que disponemos para la gestión de procesos.

Como puedes comprobar, nos basamos en conceptos básicos ya estudiados, en lo que profundizaremos un poco más para que puedas aprender lo máximo posible.

Fig. 1. Gestión de procesos.

Escucha el siguiente audio, en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad:

Audio Intro. "Usos de los procesos"

https://bit.ly/2X20V7F

/ 2. Creación de procesos en Java I

Para crear procesos en Java, vamos a utilizar la clase Process. Esta clase nos ofrece los siguientes métodos: ProcessBuilder.start() y Runtime.exec(), los cuales pueden crear un proceso nativo del sistema operativo donde se está ejecutando la máquina virtual de Java. Ambos devolverán un objeto de la clase Process.

- **ProcessBuilder.start():** Este método iniciará un proceso nuevo. Este va a ejecutar el comando y los argumentos que le indiquemos en el método command(), y se va a ejecutar en el directorio de trabajo que le indiquemos con el método directory(). Además, podrá utilizar las variables de entorno del sistema operativo que estén definidas en environment().

- **Runtime.exec(String[] cmdarray, String[] envp, File dir):** Este método ejecutará el comando que le especifiquemos con sus correspondientes argumentos en el parámetro cmdarray, en un proceso hijo que será totalmente independiente. Además, se ejecutará el entorno de trabajo indicado en el parámetro envp, y en el directorio de trabajo especificado en el parámetro dir.

Estos dos métodos comprobarán que el comando que les hayamos indicado sea un comando o fichero ejecutable válido en el sistema operativo que estemos utilizando. A fin y al cabo, el hecho de crear un nuevo proceso va a depender del sistema operativo que estemos utilizando, y pueden ocurrir diversos problemas como, por ejemplo:

- No encontrar el ejecutable debido a la ruta indicada.

- No tener permisos de ejecución.

- No ser un ejecutable válido en el sistema.

En la mayoría de los casos, se lanzará una excepción, dependiendo del sistema operativo que estemos utilizando, aunque siempre va a ser una subclase de IOException.

```
public static void main(String[] args) {
    // Ruta del ejecutable de Google Chrome
    String RUTA_PROCESO = "C:\\Program Files "
            + "(x86)\\Google\\Chrome\\Application\\chrome.exe";
    // Creamos el proceso de Google Chrome
    ProcessBuilder pb = new ProcessBuilder(RUTA_PROCESO);
    try {
        // Lanzamos el proceso
        Process process = pb.start();
        // Obtenemos su estado de ejecución
        int retorno = process.waitFor();
        System.out.println("La ejecución de " + RUTA_PROCESO +
                " devuelve " + retorno);
    } catch (IOException ex) {
        System.err.println("Error: " + ex.toString());
        System.exit(-1);
    } catch (InterruptedException ex) {
        System.err.println("El proceso hijo finalizó de forma incorrecta");
        System.exit(-1);
    }
}
```

Fig. 2. Lanzando Google Chrome en un proceso.

/ 3. Creación de procesos en Java II: lanzar método

Java nos ofrece la posibilidad de lanzar, en un proceso, un método de una clase creada por nosotros.

Para ello, deberemos crear una clase, como hemos hecho habitualmente, y el método o métodos que queremos ejecutar en forma de proceso independiente.

A continuación, crearemos un método main en el que llamaremos al método que queremos ejecutar. En el caso de que este tenga parámetros, utilizaremos el parámetro del método main String[] args para enviar los parámetros que necesitemos.

Finalmente, crearemos la función que podemos ver en la siguiente figura, que nos permitirá ejecutar una clase en un proceso independiente.

```
public static int ejecutarClaseProceso(Class clase, int n1, int n2)
        throws IOException, InterruptedException {
    // Defino dónde está el home de java
    String javaHome = System.getProperty("java.home");
    String javaBin = javaHome
            + File.separator + "bin"
            + File.separator + "java";
    String classpath = System.getProperty("java.class.path");
    // Obtengo el nombre canónico de la clase que se va a ejecutar
    String className = clase.getCanonicalName();

    ProcessBuilder builder = new ProcessBuilder(javaBin, "-cp",
            classpath, className, String.valueOf(n1), String.valueOf(n2));
    Process process = builder.start();
    process.waitFor();
    return process.exitValue();
}
```

Fig. 3. Método para ejecutar una clase en un proceso.

Como podemos ver, al método le pasamos tres atributos, uno con el nombre de la clase que queremos ejecutar, y dos parámetros de tipo int, que serán los parámetros que necesite la clase para ejecutar su método. Estos parámetros podrán cambiar y podrán ser tantos como necesitemos.

A su vez, cuando estamos creando el proceso con ProcessBuilder, los dos últimos parámetros que pasamos son los dos enteros citados, ya que son los que necesitará la clase que queremos ejecutar. Al igual que antes, aquí podremos pasarle todos los parámetros que necesitemos (o ninguno, si se da el caso), que deben ser, obligatoriamente, del tipo String, ya que, en el main de la clase, utilizaremos el array de Strings que recibe para recuperar todos los parámetros que estamos pasando en este punto.

Una vez hecho todo esto deberemos llamar al método de la siguiente forma:

int estado = ejecutarClaseProceso(Sumador.class, numero1, numero2);

Siendo Sumador el nombre de la clase que queremos ejecutar en un proceso y numero1 y numero2 los dos enteros que queremos sumar.

Vídeo 1. "Ejecución de una clase en un proceso I"
https://bit.ly/2WIOcpW

/ 4. Caso práctico 1: "Cuidado con los procesos"

Planteamiento: Pilar y José están repasando cómo pueden crear procesos, ya que tienen que realizar un ejercicio relacionado con esto.

En este ejercicio, deberán lanzar el proceso que decida el usuario, ofreciendo una lista de posibles procesos que se lancen hasta que este decida parar.

El ejercicio no es muy complicado, pero a José, cada vez que ejecuta uno de los procesos, se le queda el ordenador congelado. «Qué raro, si ejecuto cualquiera de los procesos, se me queda el ordenador congelado y tengo que forzar un reinicio», le comenta a Pilar.

Fig. 4. Es importante tener cuidado con los procesos.

Nudo: ¿Crees que José está realizar de forma incorrecta el lanzamiento de los procesos? ¿O acaso su ordenador no es lo suficientemente potente?

Desenlace: Cuando se trata de gestionar procesos, hay que tener muchísimo cuidado, ya que podemos provocar casos extraños, como el que le está ocurriendo a nuestro amigo José.

Ya sabemos que un proceso ocupará cierta parte de la memoria RAM del ordenador. Los que nosotros lanzamos también, así que, si lanzamos procesos muy grandes, podemos hacer que el ordenador se quede sin memoria RAM.

Volviendo al error de José, investigando un poco en el código del programa, se ha dado cuenta de que no ha programado de forma correcta el bucle para detener el programa cuando el usuario lo indique, sino que lo que está haciendo, en realidad, es lanzar el mismo proceso hasta que el usuario indique que quiere parar. Esto, claro está, no ocurrirá, ya que el proceso se lanzará tantas veces, que dejará al ordenador sin memoria RAM disponible para trabajar, quedándose congelado y forzando un reinicio.

/ 5. Terminar un proceso

En Java, al igual que podemos crear y lanzar un proceso, vamos a poder finalizar, en el momento en el que necesitemos, un proceso hijo que haya creado.

Esto lo podremos hacer ejecutando el método destroy, que pertenece a la clase Process. Este método va a eliminar el proceso hijo que indiquemos liberando, a su vez, todos los recursos que estuviera utilizando, dejándolos disponibles para que el sistema operativo pueda asignarlos de nuevo. En nuestro caso, como estamos utilizando el lenguaje de programación Java, todos los recursos correspondientes al proceso que hayamos eliminado serán liberados por el recolector de basura cuando se ejecute la próxima vez.

En caso de que no forcemos la finalización de la ejecución del proceso hijo, este se ejecutará de forma normal y completa, terminando y liberando sus recursos al finalizar. Esto también se puede producir cuando el proceso hijo ejecuta la operación exit para forzar la finalización de su ejecución.

En la siguiente figura, podemos ver el código necesario para lanzar un proceso de la aplicación Google Chrome y después preguntar al usuario si quiere destruirlo.

```java
public static void main(String[] args)
{
    // Ruta del ejecutable de Google Chrome
    String RUTA_PROCESO = "C:\\Program Files (x86)\\Google\\Chrome"
            + "\\Application\\chrome.exe";
    // Creamos el proceso
    ProcessBuilder pb = new ProcessBuilder(RUTA_PROCESO);
    Scanner teclado = new Scanner(System.in);

    try
    {
        // Lanzamos el proceso
        Process process = pb.start();
        System.out.println("¿Terminar el proceso? (S/N)");
        if (teclado.nextLine().charAt(0) == 'S')
        {
            // Destruimos el proceso
            process.destroy();
        }
    }
    catch (IOException ex)
    {
        System.err.println("Error: " + ex.toString());
        System.exit(-1);
    }
}
```

Fig. 5. Colas de procesos.

Como podemos ver, primero lanzamos el proceso de forma normal y, en caso de que queramos terminar su ejecución, llamaremos al método destroy.

/ 6. Comunicación entre procesos

Tenemos que tener en cuenta que, como ya sabemos, los procesos no son más que un programa que se encuentra en ejecución y, como programa que es, podrá pedir y leer información, procesarla y mostrarla por pantalla.

Para leer y mostrar información, contamos con:

- **La entrada de datos estándar (*stdin*):** Mediante la entrada de datos, podremos obtener los datos necesarios para que nuestro proceso se ejecute. Podremos leer de teclado, de un fichero, de un socket, etc.
- **La salida estándar (*stdout*):** Mediante la salida de datos, podremos mostrar información. Podremos mostrar información en la pantalla, en un fichero, en un socket, etc.
- **La salida de error (*stderr*):** Mediante la salida de error, podremos mostrar la información de los errores que ocurran en la ejecución de nuestros procesos. Podremos mostrar los errores igualmente en la pantalla, en un fichero, en un socket, etc.

Lo normal es que las entradas y las salidas de datos de un proceso hijo sean las mismas que las de su proceso padre, es decir: si creamos un proceso hijo dentro de un programa que lee la información de un fichero y muestra los resultados por pantalla, el proceso hijo que creemos lo hará de igual forma.

En Java, como es nuestro caso, esto no ocurre así. El proceso hijo que creemos no va a tener su propia interfaz de comunicación, por lo que no podrá comunicarse con el proceso padre directamente. Como programadores, deberemos redireccionar todas sus salidas y entradas (***stdin***, ***stdout*** y ***stderr***) mediante:

- **OutputStream:** Será el flujo de **salida**. Está conectado a la **salida** estándar del proceso hijo. Se redirecciona con el método *redirectOutput*.
- **InputStream:** Será el flujo de **entrada**. Está conectado a la **entrada** estándar del proceso hijo. Se redirecciona con el método *redirectInput*.
- **ErrorStream:** Será el flujo de **salida** para los errores. Está conectado a la **salida** estándar de errores del proceso hijo. Se redirecciona con el método *redirectError*.

Una vez hayamos redirigido todos estos flujos, podremos hacer una comunicación entre los procesos padre e hijo.

Si queremos redireccionar a la salida/entrada estándar, deberemos utilizar *Redirect.INHERIT*.

Vídeo 2. "Ejecución de una clase en un proceso II"
https://bit.ly/2CYJE7Q

/ 7. Invocar al bash del sistema operativo

Uno de los procesos que vamos a poder invocar y utilizar es el bash o consola del sistema operativo.

Mediante el bash podremos ejecutar multitud de comandos que nos permitirán realizar operaciones muy potentes.

Para ejecutar el bash de Windows, deberemos crear un proceso con la ruta CMD, siendo CMD el proceso que se encargará de ejecutar el bash.

Una vez hecho esto, deberemos obtener los flujos de entrada y salida del proceso que ejecutará el bash, para poder interactuar con él de forma correcta.

Cuando ya tengamos el proceso lanzado, podremos indicarle que ejecute comandos mediante la orden println. Después de cada comando, deberemos limpiar el flujo del proceso, para evitar errores, lo cual podremos hacer mediante el método flush.

El proceso se estará ejecutando en segundo plano e irá ejecutando todos y cada uno de los comandos que le indiquemos, que pueden ser estos tan complejos como necesitemos.

```
public static void main(String[] args) {
    try {
        // Creo el proceso hijo
        ProcessBuilder builder_echo = new ProcessBuilder("CMD");
        Process proceso_echo = builder_echo.start();

        // Escribo la salida del proceso hijo que ejecuta el bash
        final Scanner in = new Scanner(proceso_echo.getInputStream());
        new Thread() {
            public void run() {
                while (in.hasNextLine()) {
                    System.out.println(in.nextLine());
                }
            }
        }.start();

        // Obtengo la salida del proceso hijo
        PrintWriter salida = new PrintWriter(proceso_echo.getOutputStream());
        // Hago una llamada al proceso hijo con el comando ipconfig
        salida.println("ipconfig");
        salida.flush();

        salida.close();
    } catch (IOException ex) {
        System.err.println("Excepción de E/S!!: " + ex.toString());
    }
}
```

Fig. 6. Ejecutando el comando ipconfig, que nos muestra nuestra IP, desde un proceso en Java.

Hay que tener mucho cuidado si queremos ejecutar el bash, ya que, según el sistema operativo que estemos utilizando, tanto la forma de invocarlo como los comandos a ejecutar cambiarán.

Vídeo 3. "Ejemplo de invocación del bash de Windows"
https://bit.ly/30zR9KF

/ 8. Caso práctico 2: "¿Podemos comprobar que nuestros procesos son los correctos?"

Planteamiento: Una vez Pilar y José le han cogido el truco a la gestión de procesos, parece que no es tan complicado como parecía.

«Me pregunto si hay alguna forma de ver que los procesos que estamos lanzando y con los que estamos trabajando están ahí de verdad», le dice Pilar a José, a lo que este le responde que no había pensado en ello, pero que sería una muy buena forma de comprobar que lanzan los procesos correctos.

Nudo: ¿Qué piensas al respecto? ¿Crees que los sistemas operativos ofrecen alguna herramienta para comprobar qué procesos se están ejecutando en cada momento?

Desenlace: Cuando trabajamos con procesos, hay algunos que no podemos comprobar a simple vista si están ejecutándose o no.

Si lo que hacemos es lanzar un proceso de una aplicación en concreto, por ejemplo, el bloc de notas, podremos saber que ese proceso se está ejecutando porque estaremos viendo abierto dicho software. Pero ¿qué ocurre si tenemos una aplicación que lanza procesos que no vemos para realizar algún cálculo?

Estos procesos «invisibles» a nuestros ojos son muy fáciles de ver en el sistema operativo, ya que cualquiera de los tres grandes sistemas operativos que existen (GNU/Linux, Windows y MacOS) tiene una herramienta de gestión de procesos que nos permitirá comprobar cuáles están en ejecución, tanto de forma gráfica como en consola.

La aplicación que nos lo va a permitir es el Administrador de Tareas. Con esta herramienta será muy fácil ver si hemos lanzado los procesos correctamente en nuestras aplicaciones.

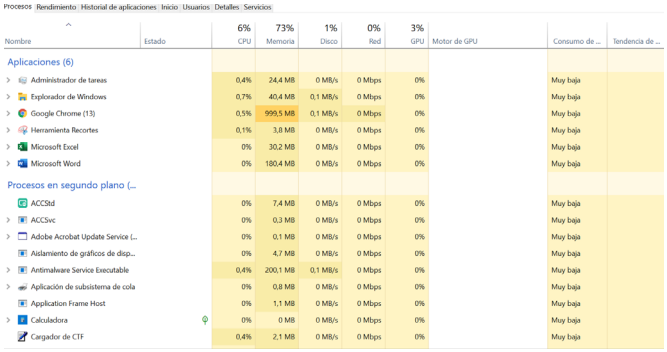

Fig. 7. Comprobando procesos en Windows.

/ 9. Sincronización entre procesos

Puede ocurrir que, en nuestra aplicación Java, estemos lanzando más de un proceso al mismo tiempo. En ese caso, no tenemos forma de saber qué línea de código estará ejecutando cada uno de ellos en un momento dado.

Esto puede ser muy peligroso, ya que, si dos o más procesos necesitan acceder a una variable en memoria, por ejemplo, es posible que alguno de ellos la modifique y los demás ya no puedan ver su valor original, sino el valor que ya ha sido modificado por el otro proceso que accedió a ella en primer lugar.

Esto es lo que se denomina una zona crítica, y las vamos a encontrar en todos y cada uno de los programas que tengan más de un proceso en activo. Estas zonas críticas son muy peligrosas y deben protegerse mediante una serie mecanismos.

```
int i = 0, j = 0;

synchronized(MiClase.class)
{
    if (i >= 2)
    {
        i++;
        j++;
    }
    System.out.println("Ok");
    i = i * 2;
    j--;
}
```

Fig. 8. Código sincronizado en Java. concurrente.

Los mecanismos más comunes para controlar este tipo de situaciones son los siguientes:

- Semáforos
- Colas de mensajes
- Pipes o tuberías
- Bloques de memoria compartida

Todos estos mecanismos nos van a ofrecer una solución al problema de las secciones críticas, pero unos necesitarán de una mayor elaboración que otros.

En el lenguaje de programación Java, tenemos una forma muy rápida de hacer que un bloque de código esté sincronizado entre varios procesos. Básicamente, consiste en incluir delante la palabra reservada synchronized; de esta forma, la propia máquina virtual de Java hace que ese código sea seguro en la ejecución de dos o más procesos.

Audio 1. "Depuración de programas multiproceso2
https://bit.ly/2D1PdT1

/ 10. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos visto cómo podemos llevar a cabo una gestión completa de procesos mediante nuestros programas en Java.

En primer lugar, hemos aprendido cómo crear y lanzar un proceso, pudiendo ejecutar una aplicación diferente a nuestro proyecto de una forma rápida y sencilla.

También hemos estudiado cómo ejecutar una clase en un proceso independiente, así como el procedimiento necesario para poder terminar un proceso.

Finalmente, hemos comprobado cómo podemos comunicar varios de los procesos que lanzamos en nuestra aplicación, además de los mecanismos de los que disponemos para sincronizar procesos.

Resolución del Caso práctico de la unidad

La programación multiproceso es algo que, por norma general, resulta bastante extraño cuando se estudia en un principio, ya que, en asignaturas anteriores, todo lo que hemos programado siempre ha estado en un único programa, es decir, en un único proceso.

En la vida real, la programación multiproceso está presente en multitud de situaciones y circunstancias, hasta ahora impensables por nosotros.

Pongamos como ejemplo nuestro procesador de textos favorito. Cuando escribimos y cometemos una falta de ortografía, vemos que debajo de la misma aparecerá una línea roja de forma automática. Esto se debe a que el corrector ortográfico se está ejecutando a la misma vez que el procesador de texto, en un proceso diferente, lo cual permite que nos corrija en tiempo real.

Si en este sencillo ejemplo el multiproceso está presente, imagina cuántos multiprocesos se estarán ejecutando cada segundo en cualquier aplicación que estemos utilizando.

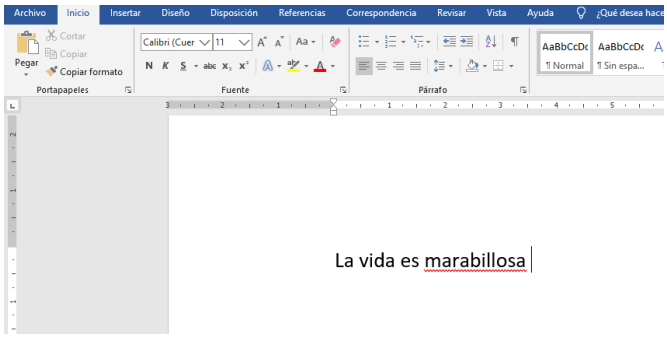

Fig. 9. Varios procesos trabajando conjuntamente.

/ 11. Bibliografía

Gómez, O. (2016). Programación de Servicios y Procesos Versión 2.1. Recuperado de: https://github.com/OscarMaestre/ServiciosProcesos/blob/master/build/latex/Servicios Procesos.pdf

Sánchez, J. M. y Campos, A. S. (2014). Programación de servicios y procesos. Madrid: Alianza Editorial.

Running Bash commands in Java. (9 de noviembre de 2014). [Entrada de Blog]. Stack Overflow. Recuperado de: https://stackoverflow.com/questions/26830617/running-bash-commands-in-java