Programación de servicios y procesos

Unidad Didáctica 01: Programación multiproceso

Demostración: Programación multiproceso en Java

Resumen de planificación de actividad

Objetivos	Contenidos que se trabajan
• Poner en práctica la creación de procesos mediante el lenguaje de programación Java	• Ejecutables. Procesos. Servicios. • Programación concurrente. • Creación de procesos. • Comunicación entre procesos. • Gestión de procesos. • Comandos para la gestión de procesos en sistemas libres y propietarios. • Sincronización entre procesos. • Programación de aplicaciones multiproceso. • Documentación. • Depuración.
Objetivos didácticos	**Criterios de evaluación**
RA1: Desarrolla aplicaciones compuestas por varios procesos reconociendo y aplicando principios de programación paralela.	c) Se han analizado las características de los procesos y de su ejecución por el sistema operativo. e) Se han utilizado clases para programar aplicaciones que crean subprocesos. f) Se han utilizado mecanismos para sincronizar y obtener el valor devuelto por los subprocesos iniciados. g) Se han desarrollado aplicaciones que gestionen y utilicen procesos para la ejecución de varias tareas en paralelo. h) Se han depurado y documentado las aplicaciones desarrolladas.

Recursos necesarios	Agrupamiento	Duración estimada
• Ordenador Ubuntu con acceso a Internet	Grupo clase	2 sesiones

Secuencia de desarrollo de la actividad

1. Planteamiento y objetivos.
2. Explicación guiada de cada ejemplo
3. Resolución de dudas

INSTRUCCIONES

Esta demostración se hará de forma individual.

PREPARACIÓN DEL ENTORNO

Comprobamos si tenemos instalado el JRE de Java

```
java --version
```

Si en Linux no disponemos de Java instalado, ejecutar con permisos de root:

```
apt-get update
sudo apt install default-jre
sudo apt install default-jdk
```

CREACIÓN DE PROCESOS EN JAVA

Clases Process y ProcessBuilder

La clase **Process** de java.lang representa un proceso que es lanzado desde el código java de nuestra aplicación. Para crear e inicializar estos procesos se usa la clase ProcessBuilder

La clase **ProcessBuilder** proporciona métodos para configurar y lanzar procesos del sistema. La clase ProcessBuilder permite básicamente ejecutar un comando de shell desde el código Java. Mediante ProcessBuilder podemos hacer varias cosas con un proceso antes de lanzarlo como

- Establecer el comando a lanzar y sus argumentos.
- Crear un entorno personalizado de ejecución con determinados valores de las variables de entorno.
- Cambiar el directorio de trabajo de donde se ejecuta nuestro comando de shell.
- Redirigir los flujos de entrada, salida y error del proceso.

Al constructor de la clase es posible pasarle una lista con el comando y sus argumentos.

```
ProcessBuilder(List<String> command)
```

Este ejemplo lanza el ejecutable *notepad.exe* (el bloc de notas de Windows)

Abre el fichero EjemploNotepad.java con un editor de texto plano y revisa el código

Compila EjemploNotepad.java (javac EjemploNotepad.java)

Ejecuta EjemploNotepad (java EjemploNotepad)

Ejercicios:

Ejercicio 1: Crea un programa que muestre el usuario actual en Windows y sus privilegios (comando whoami /all).

Spoiler: con lo que sabes hasta ahora no te va a funcionar.

Leer la salida estándar del proceso

En comandos escriban en la salida estándar es necesario usar mecanismos de redirección. Veamos el siguiente ejemplo que crea un proceso que muestra los archivos de la carpeta actual en Windows:

Abre el fichero EjemploSalida.java con un editor de texto plano y revisa el código

Compila EjemploSalida.java (javac EjemploSalida.java)

Ejecuta EjemploSalida (java EjemploSalida)

Ejercicios:

Ejercicio 2: Modifica el ejercicio 1 para que funcione.

Leer la salida de error del proceso

Para redirigir la salida de error del proceso se usa el método de Process **getErrorStream()**. Por cierto, cuando hacemos un *java -version* desde un shell la salida es dirigida a la salida estándar de error, no a la salida estándar.

Ejercicios

Ejercicio 3: Crea un programa que muestre en pantalla la versión de java instalada mediante la ejecución del comando *java -version*.

Ejercicio 4: Crea un programa que muestre por pantalla los errores que se producen cuando, por ejemplo, intentamos crear un proceso que ejecute un programa que no existe.

Cambiar el directorio de trabajo del proceso

Otra cosa que podemos hacer es cambiar el directorio de trabajo donde se ejecuta el proceso, para ello se usa el método **directory()**

```
directory(File directory)
```

Usaremos **directory** para cambiar el directorio donde se ejecutará el comando anterior.

Abre el fichero EjemploDirectorio.java con un editor de texto plano y revisa el código

Compila EjemploDirectorio.java (javac EjemploDirectorio.java)

Ejecuta EjemploDirectorio (java EjemploDirectorio)

Ejercicios

Ejercicio 5: Modifica el ejemplo para que el proceso tome como directorio actual el que ha recibido como primer argumento desde la línea de comandos.

Ejercicio 6: Crea un programa que invoque al que has creado en el ejercicio anterior con el argumento "c:\program files".

Escribir en la entrada estándar del proceso

Si el comando necesita información de entrada por parte del usuario de forma interactiva. La clase process tiene el método **getOutputStream()** que permite escribir en el stream de entrada del proceso, así podemos enviar datos a un comando.

Por ejemplo, el comando **date** de Windows nos devuelve la fecha actual y nos pide que escribamos de forma interactiva una nueva fecha si queremos cambiarla. El siguiente ejemplo ejecuta el comando date y le da un nuevo valor para la fecha del sistema.

Abre el fichero EjemploEntrada.java con un editor de texto plano y revisa el código

Compila EjemploEntrada.java (javac EjemploEntrada.java)

Ejecuta EjemploEntrada (java EjemploEntrada)

Ejercicios

Ejercicio 7. Crea un nuevo programa partiendo del obtenido en el ejercicio 5 para que el directorio que tome sea el introducido por el teclado. Crea otro programa que permita invocarlo.

Redirigir la entrada, salida y error a ficheros

ProcessBuilder proporciona métodos para redirigir las salidas estándar a un fichero así como para tomar la entrada desde un fichero:

```
redirectOutput(File file)
redirectError(File file)
redirectInput(File file)
```

Abre el fichero EjemploRedireccion.java con un editor de texto plano y revisa el código

Compila EjemploRedireccion.java (javac EjemploRedireccion.java)

Ejecuta EjemploRedireccion (java EjemploRedireccion)

También es posible usar la subclase *Redirect* de *ProcessBuilder* que establece una serie de opciones interesantes:

- Redirect.INHERIT para indicar que se tomar la entrada o salida establecida para el proceso actual
- Redirect.appendTo(file): para añadir al final del fichero indicado

Ejercicios

Ejercicio 8. Haz una versión del ejercicio 2 usando redirecciones directas a ficheros con la clase *ProcessBuilder*.

Ejercicio 9. Haz una versión del ejercicio 7, usando redirecciones directas a ficheros con la clase *ProcessBuilder*, en la que el programa que invoca haga que se tome como entrada un fichero llamado *directorio.txt* que contenga el directorio a listar.