Programación multiproceso I

Programación de servicios y procesos

Objetivos:

Una vez hemos aprendido a diferenciar entre programas, procesos e hilos, en este segundo tema vamos a aprender cómo desde un programa creado por nosotros podemos lanzar otros programas, es decir, desde un proceso en ejecución, podemos crear otro proceso.

Además de lanzarlos, al establecerse una relación padre-hijo estos procesos pueden comunicarse entre sí intercambiando información. De esta forma nuestros programas podrán lanzar otras aplicaciones, comandos del SO e incluso otras aplicaciones nuestras, permitiendo cierto grado de sincronización y comunicación entre ellas

Los objetivos que alcanzaremos tras esta unidad son:

- Conocer las clases de Java para la creación de procesos
- Monitorizar y controlar el ciclo de vida de un proceso
- Controlar la comunicación entre procesos padre/hijo
- Usar métodos para la sincronización entre procesos y subprocesos
- Entender el mecanismo de comunicación mediante tuberías (pipes)

Contenidos:

1) **Gestión de procesos. Conceptos básicos: Creación, ejecución y finalización de procesos.**
2) Sincronización entre procesos. Exclusión mutua. Condiciones de sincronización.
3) Recursos compartidos.
4) Mecanismos de comunicación y sincronización de procesos (semáforos, monitores, paso de mensajes.)
5) Problemas. Inanición, interbloqueos.

Gestión de procesos. Introducción.

En la actualidad prácticamente todos los sistemas de computación (ordenadores, smartphones, tablets, smartwatches, consolas,...) son multiproceso, por lo que tienen la capacidad de mantener en ejecución simultánea varios procesos / programas.

Los procesos necesitan recursos y estos son limitados. El procesador, la memoria, el acceso a sistemas de almacenamiento o a los diferentes dispositivos son algunos de ellos. La pregunta que surge es ¿cómo se consigue que la convivencia entre los procesos, que compiten entre sí por los limitados recursos, sea posible?

Como ya vimos la respuesta está en el sistema operativo y más concretamente en el planificador de procesos.

Gestión de procesos. Ejecución.

El sistema operativo es el encargado de crear los nuevos procesos siguiendo las directrices del usuario.

- La puesta en ejecución de un nuevo proceso se produce debido a que hay un proceso en concreto que está pidiendo su creación en su nombre o en nombre del usuario.

- Cualquier proceso en ejecución (proceso hijo) siempre depende del proceso que lo creó (proceso padre), estableciéndose un vínculo entre ambos. A su vez, el nuevo proceso puede crear nuevos procesos, formándose lo que se denomina un árbol de procesos.

Gestión de procesos. Finalización.

Operaciones básicas:

- **Creación de procesos**: Cuando se crea un nuevo proceso hijo, ambos procesos, padre e hijo se ejecutan concurrentemente.
  - Ambos procesos comparten la CPU y se irán intercambiando siguiendo la política de planificación asignada
  - Si el proceso padre necesita esperar hasta que el hijo termine su ejecución, puede hacerlo mediante la operación wait.
  - Los procesos son independientes y tienen su propio espacio de memoria asignado, llamado imagen de memoria.
- **Terminación de procesos**: Al terminar la ejecución de un proceso, es necesario avisar al sistema operativo de su terminación para liberar los recursos que tenga asignados.
  - En general, es el propio proceso el que le indica al sistema mediante la operación exit que quiere terminar, pudiendo aprovechar para mandar información de su finalización al padre.

Gestión de procesos en Java.

En el paquete java.lang tenemos dos clases para la gestión de procesos.

- java.lang.ProcessBuilder
- java.lang.Process

Las instancias de **ProcessBuilder** gestionan los atributos de los procesos, mientras que las instancias de **Process** controlan la ejecución de esos mismos procesos cuando se ejecutan.

Antes de ejecutar un nuevo proceso, podemos configurar los parámetros de ejecución del mismo usando la clase ProcessBuilder. La clase ProcessBuilder define un par de constructores:

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

El funcionamiento de ambos es el mismo. En el primer constructor se pasa el comando a ejecutar y la lista de argumentos como una lista de cadenas. Por contra, en el segundo constructor, el comando y sus argumentos se pasan a través de un número variable de cadenas (String ... es lo que en Java se llama varargs).

Gestión de procesos en Java.

Tabla 1.3. Métodos de la clase java.lang.ProcessBuilder

Método	Descripción
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 la salida de errores.
redirectInput	Permite determinar el origen de la entrada estándar.
redirectOutput	Permite determinar el destino de la salida estándar.

https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html

Gestión de procesos en Java.

Tabla 1.2. Métodos de la clase java.lang.Process

Método	Descripción
destroy()	Destruye el proceso sobre el que se ejecuta.
exitValue()	Devuelve 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 del proceso.
getInputStream()	Proporciona un InputStream conectado a la salida normal del proceso.
getOutputStream()	Proporciona un OutputStream conectado a la entrada normal del proceso.
isAlive()	Determina si el proceso está o no en ejecución.
waitFor()	Detiene la ejecución del programa que lanza el proceso a la espera de que este último termine.

https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html

Gestión de procesos en Java.

Además de ProcessBuilder, la clase Runtime también me permite ejecutar procesos y comandos del SO desde java.

La principal diferencia es que utilizando la clase de ProcessBuilder debe ser en la creación de este objeto y antes de arrancar el propio proceso donde debes asignar el proceso a ejecutar, directorio de trabaj y las variables de entorno y finalmente ejecutarlo y es cuando se crea el proceso. Utilizando la clase Runtime el proceso se crea “en vacío” y al ejecutarlo es cuando le pasaras los parámetro necesarios para ejecutar el proceso que necesites.

Gestión de procesos en Java.

Operaciones básicas:

- 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 y devuelven un objeto de la clase **Process** que puede ser utilizado para controlar dicho proceso.

- El método ***start()*** inicia un nuevo proceso utilizando los atributos indicados en el objeto. El nuevo proceso ejecuta el comando y los argumentos indicados, ejecutándose en el directorio de trabajo especificado por ***directory()***, utilizando las variables de entorno definidas en ***environment()***.

- El método *exec(cmdarray, envp, 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.

Configuraciones adicionales de un proceso.

Algunos de los atributos que podemos configurar para un proceso son:

- Establecer el directorio de trabajo donde el proceso se ejecutará: podemos cambiar el directorio de trabajo por defecto llamando al método directory y pasándole un objeto de tipo File. Por defecto, el directorio de trabajo se establece al valor de la variable del sistema **user.dir**. Este directorio es el punto de partida para acceder a ficheros, imágenes y todos los recursos que necesite nuestra aplicación.

- Configurar o modificar variables de entorno para el proceso con el método environment()

Gestión de procesos. Ejemplos

- Ejemplo creación de procesos con ProcessBuilder => 1_EjecutarAplicacionProcessBuilder.java
- Ejemplo de creación de procesos con Runtime => 2_EjecutarAplicacionRuntime.java
- Ejemplo de destrucción de procesos => 3_DestruccionProceso.java
  - Cuando el proceso termine su ejecución, liberará sus recursos.
  - Esto se produce cuando realiza la operación **exit** para finalizar su ejecución.
  - En Java, los recursos correspondientes los eliminará el **garbage collector** cuando considere.
- Ejemplo de consulta variable entorno de procesos => 4_Environment.java

Gestión de procesos. Ejercicio

Escribe un programa Java con la siguiente lógica:

- Que le pida al usuario que introduzca una web a la que quiera navegar
- Una vez el usuario la introduce, abra un navegador y vaya a esta url introducida por el usuario