Unidad 2. Programación de procesos

Descargar estos apuntes

Índice

Introducción

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.

Objetivos

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)
- Aprender la sintaxis y uso del comando curl para probar API REST desde un programa
- Crear programas que ejecuten tareas en paralelo.

Comunicación entre procesos

La comunicación entre procesos, IPC (Inter Process Communication), es una de las características principales de los sistemas operativos. En este apartado nos vamos a centrar en la comunicación de procesos que estén en el mismo dispositivo.

Comunicación mediante E/S

La comunicación entre procesos se puede realizar de muchas formas, pero una de las más sencillas y comunes es la comunicación mediante la entrada y salida estándar.

> **E/S en Java**
>
> En Java, la comunicación mediante la entrada y salida estándar se realiza mediante los flujos de entrada y salida estándar, `System.in` y `System.out` respectivamente.

Todo proceso tiene tres flujos de entrada y salida estándar que se pueden usar para la comunicación con otros procesos. Estos flujos son:

- **stdin** (entrada estándar): por donde el proceso recibe datos. Por defecto se corresponde con el teclado y el identificador del fichero asociado es 0.
- **stdout** (salida estándar): por donde el proceso envía datos. Por defecto se corresponde con la consola y el identificador del fichero asociado es 1.
- **stderr** (salida de error estándar): por donde el proceso envía mensajes de error. Por defecto se corresponde con la consola y el identificador del fichero asociado es 2.

Un mecanismo IPC, relativamente sencillo, es la comunicación de procesos mediante la redirección de las salidas y entradas estándar a/desde otras fuentes.

> **Redirección de E/S**
>
> La redirección de la entrada y salida estándar se puede hacer en la línea de comandos de los sistemas UNIX y Windows. En Java, se puede hacer mediante la clase `ProcessBuilder` que veremos en la siguiente sección de la unidad

Redirección de la entrada estándar

La redirección de la entrada estándar se puede hacer mediante el operador `<` en sistemas UNIX y Windows.

```
$> java MiClase < entrada.txt
```

En el ejemplo anterior, el programa `MiClase` recibe la entrada estándar desde el fichero `entrada.txt` en vez de desde el teclado.

> Cuando se redirige la entrada estándar, el programa no tiene que hacer nada especial para leer de un fichero en vez de del teclado. El sistema operativo se encarga de redirigir la entrada estándar del programa al fichero que se le indica.

Redirección de la salida estándar

La redirección de la salida estándar se puede hacer mediante los operadores `>` y `>>` en sistemas UNIX y Windows.

```
$>  java MiClase > salida.txt
$>  java MiClase >> salida2.txt
```

En el ejemplo anterior, la salida estándar del programa `MiClase` se redirige al fichero `salida.txt` en vez de a la consola. Si el fichero `salida.txt` no existe, lo crea, y si el fichero ya existe, sobrescribe su valor.

Si el operador es `>>`, la salida se añade al final del fichero en vez de sobrescribirlo.

> Cuando se redirige la salida estándar, el programa no tiene que hacer nada especial para escribir en un fichero en vez de en la consola. El sistema operativo se encarga de redirigir la salida estándar del programa al fichero que se le indica.

Redirección de la salida de error estándar

La redirección de la salida de error estándar se puede hacer mediante el operador `2>` en sistemas UNIX y Windows.

```
$> java MiClase 2> error.txt
$> java MiClase 2>> error2.txt
```

En el ejemplo anterior, la salida de error estándar del programa `MiClase` se redirige al fichero `error.txt` en vez de a la consola.

Si el operador es `2>>`, la salida de error se añade al final del fichero en vez de sobrescribirlo.

> Cuando se redirige la salida de error, el programa no tiene que hacer nada especial para escribir en un fichero en vez de en la consola. El sistema operativo se encarga de redirigir la salida de error del programa al fichero que se le indica.

Redirección de la salida de un proceso a la entrada de otro proceso

La redirección de la salida estándar a la entrada estándar de otro proceso se puede hacer mediante el operador `|` en sistemas UNIX y Windows.

Las tuberías (pipes) permiten conectar la salida estándar de un proceso con la entrada estándar de otro, estableciendo así una relación de productor-consumidor.

El uso de tuberías sigue la siguiente sintaxis:

```
$> java MiClase | java MiClase2
```

En el ejemplo anterior, la salida estándar del programa `MiClase` se redirige a la entrada estándar del programa `MiClase2`.

> Cuando se redirige la salida estándar de un proceso a la entrada estándar de otro, el sistema operativo se encarga de conectar los flujos de salida y entrada de los procesos.

Comunicación mediante señales

Las señales son una forma de comunicación entre procesos que se basa en la interrupción de la ejecución de un proceso para que realice una acción determinada.

Las señales son eventos asíncronos que se envían a un proceso para notificarle de un evento. Las señales pueden ser enviadas por el propio proceso, por otro proceso o por el sistema operativo.

Las señales se pueden enviar a un proceso mediante la línea de comandos o mediante un programa. En sistemas UNIX, se pueden enviar señales a un proceso mediante el comando `kill`.

```
$> kill -s SIGUSR1 1234
```

En el ejemplo anterior, se envía la señal `SIGUSR1` al proceso con PID `1234`.

Las señales en el shell de Windows se pueden enviar mediante el comando `taskkill`.

```
$> taskkill /pid 1234 /f
```

En el ejemplo anterior, se envía la señal de finalización forzada al proceso con PID `1234`.

> **Señales**
>
> Os dejo un enlace a la lista de señales de UNIX.
>
> Y un artículo de ampliación sobre Gestión de procesos en Windows.

Comunicación mediante sockets

Los sockets se pueden usar para la comunicación entre procesos en el mismo dispositivo o en dispositivos diferentes.

Los sockets los estudiaremos en la Unidad 4, donde veremos cómo se pueden usar para la comunicación entre procesos en dispositivos diferentes.