Programación de Procesos en Java

Ejercicios

Ejercicio 1: Proceso Guardian

Desarrolla un programa en Java para lanzar el navegador Firefox abriendo la canción *"Never Gonna Give You Up"* en YouTube. Cada vez que el proceso de Firefox termine (es decir, cada vez que el navegador sea cerrado por el usuario), el programa debe lanzar nuevamente el proceso de Firefox con la misma URL. El objetivo es que el navegador siempre esté ejecutándose con la canción abierta, a menos que el programa de Java sea detenido manualmente.

Ejercicio 2: Contador de apariciones

En el aula virtual encontrarás un shell script llamado `contar_palabras.sh` que recibe como argumentos un archivo de texto y una palabra, y devuelve el número de apariciones de esa palabra en el archivo.

Escribe un programa en Java que utilice procesos para contar el número de apariciones de una determinada palabra en varios archivos de texto diferentes.

Cada proceso debe escribir su resultado en un fichero llamado `resultados.txt`. Una vez que todos los procesos hayan terminado, el programa Java debe leer el fichero `resultados.txt`, sumar el número total de apariciones de la palabra en todos los archivos procesados, e imprimir el total por consola

Ejercicio 3: Menú Interactivo

Escribir un programa en Java que ofrezca un menú interactivo al usuario con las siguientes opciones:

1. **Lanzar un sleep de duración determinada por el usuario en foreground**:
   El programa debe pedir al usuario que ingrese el tiempo en segundos para ejecutar el comando `sleep`. El proceso `sleep` debe ejecutarse en **foreground** (primer plano), lo que significa que el programa Java esperará a que el proceso termine antes de continuar.
2. **Lanzar un sleep de duración determinada por el usuario en background**:
   Similar al primer caso, pero el proceso `sleep` debe ejecutarse en **background** (segundo plano). El programa Java no debe esperar a que el proceso termine, sino que debe continuar ejecutándose y permitir al usuario realizar otras acciones mientras el proceso `sleep` sigue activo.
3. **Mostrar el PID de los procesos aún en ejecución**:
   El programa debe mantener un registro de los procesos lanzados en background y mostrar al usuario una lista de los PID (Process IDs) de estos procesos que aún están en ejecución.

4. **Salir**:
   Esta opción finaliza el programa Java. Antes de salir, el programa debe asegurarse de que todos los procesos lanzados en background han finalizado.

Ejercicio 4: Suma de Cuadrados

Dado los siguientes scripts de shell, los cuales puedes encontrar en el aula virtual:

1. `cuadrados.sh`: Dado una lista de números como argumentos, este script devuelve los números elevados al cuadrado.
2. `suma.sh`: Dado una lista de números como argumentos, este script devuelve la suma de dichos números.

Escribe un programa en Java que reciba una lista de números y calcule la suma de los cuadrados de esta haciendo uso de los shell scripts dados.
Por ejemplo, si el programa recibe la siguiente lista:

```
1 2 3
```

Deberá devolver la suma de los cuadrados, es decir:

```
14
```

Ejercicio 5: Editorial ChapiChapuzas

La editorial ChapiChapuzas está pensando en expandir su negocio y traducir sus obras a nuevos idiomas para aumentar su nicho de mercado. Para ello, quiere automatizar la línea de procesos que va desde la correción ortográfica de la obra hasta su traducción.

Se pide a los alumnos diseñar un programa en Java que gestione una **línea de procesos** para corregir y traducir ficheros de texto de manera automática. El flujo de trabajo consiste en:

1. **Corregir las faltas de ortografía** del fichero de texto.
2. **Traducir el texto corregido** al idioma seleccionado (inglés, italiano o francés).

Para realizar estas tareas, cuentan con dos scripts:

- `check.sh`: Este script corrige las faltas de ortografía del texto.
- `translate.sh`: Este script traduce un fichero de texto al idioma seleccionado, con las siguientes opciones:
  - `-en` para inglés.
  - `-fr` para francés.
  - `-it` para italiano.

El objetivo es crear un programa que permita gestionar múltiples trabajos de forma concurrente, aplicando estas dos operaciones (corrección y traducción) a varios archivos de texto.

Se debe crear la clase `Trabajo`, que representará una tarea individual de corrección y traducción de un fichero. Esta clase debe contener los siguientes atributos:

- `sourcePath`: Ruta del fichero de texto de origen que se desea procesar.
- `language`: Idioma al que se desea traducir el fichero. Puede ser "en" (inglés), "fr" (francés) o "it" (italiano).
- `destinationPath`: Ruta del fichero donde se guardará el texto corregido y traducido.

El programa debe permitir procesar varios trabajos simultáneamente o en serie.

NOTA: Para que los scripts funcionen correctamente este necesario instalar las siguientes librerías

```
sudo apt install hunspell hunspell-es
sudo apt install translate-shell
```