Teoría PSP

Índice

BCP Linux y Comandos Linux.

El **BCP** es una estructura de datos llamada **Bloque de Control de Procesos** y almacena información acerca del proceso.

PS

Mediante el comando ps en Linux podemos ver parte de la información asociada a cada proceso.

- **PID:** Identificador del proceso.
- **TTY:** Terminal de ejecución asociado.
- **TIME:** Tiempo de ejecución asociado. (El tiempo total).
- **CMD:** Nombre del proceso.

PS -F

La orden ps –f muestra más información:

- **UID:** Nombre de usuario
- **PPID:** PID del padre de cada proceso.
- **C:** Porcentaje de recursos de CPU utilizado por el proceso.
- **STIME:** Hora de inicio del proceso.

PS -AF

Con la orden ps –AF podemos ver información más detallada:

- **SZ:** tamaño virtual de la imagen del proceso.
- **PSR:** Procesador que el proceso tiene asignado actualmente.

Windows TaskList

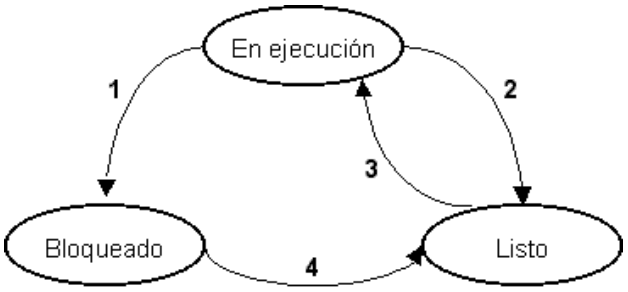
Para ver los procesos de Windows utilizamos la orden tasklist.
Aunque lo más típico es utilizar las teclas [CTRL + ALT + SUPR].

Estados de Procesos

Aunque los procesos son entidades independientes, a veces su salida genera la entrada para otro proceso.
Un proceso también puede parar porque el S.O decida asignar el procesador a otro proceso.

Los procesos se pueden encontrar en tres estados:

- **En ejecución:** Si se está ejecutando, es decir, usando el procesador.
- **Bloqueado:** No puede hacer nada hasta que otro proceso le dé la entrada.
- **Listo:** El proceso está esperando su turno en el procesador para ejecutarse.



Las transacciones entre los estados son las siguientes:

De Ejecución a Bloqueado: El proceso espera que ocurra un evento.

De Bloqueado a Listo: Ocurre el evento que el proceso esperaba.

De Listo a Ejecución: El S.O le otorga tiempo de CPU.

De Ejecución a Listo: Se le ha acabado el tiempo asignado por el S.O.

Creación Procesos

JAVA dispone en el paquete java.lang varias clases para la gestión de procesos. Una de ellas es la clase **ProcessBuilder.**
Cada instancia ProcessBuilder gestiona una colección de atributos del proceso.
El método .start() crea una instancia de **Process** con esos atributos.

La clase Process proporciona métodos para realizar la entrada desde el proceso, obtener la salida del proceso, esperar a que el proceso se complete y destruir el proceso.

Process	
Métodos	**Misión**
InputStream getInputStream()	Nos permite leer lo que el comando que ejecutamos escribió en la consola.
Int waitFor()	Provoca que el proceso actual espere hasta que el subproceso representado por el objeto Process finalice. Devuelve 0 si finaliza correctamente.
InputStream getErrorStream()	Nos permite leer los errores que se produzcan al lanzar el proceso.
OutputStream getOutputStream()	Nos permite escribir en la consola.
Void destroy ()	Elimina el subproceso.
Int exitValue()	Devuelve el valor de salida del subproceso.
Boolean isAlive()	Comprueba si el subproceso representado por Process está vivo.
Process Builder	
Métodos	**Misión**
ProcessBuilder command (“CMD”, “/C”, “DIR”)	Define el programa que se quiere ejecutar indicando sus argumentos como una lista de cadenas separadas por comas.
Process start()	Inicia un nuevo proceso.
File directory()	Devuelve el directorio de trabajo del objeto ProcessBuilder.
ProcessBuilder redirectOutput (File file)	Almacena la información de salida en un fichero.
ProcessBuilder redirectError (File file)	Almacena la información de un error en un fichero.

Ejemplo 1 : Ejecutar Notepad por Java

```
public static void main(String[] args) throws IOException{

      ProcessBuilder pb=new ProcessBuilder ("Notepad");
      Process p=pb.start();
   }
}
```

Ejemplo 2 : Leer la Consola desde Eclipse

```
     public static void main(String[] args) throws IOException {

      ProcessBuilder pb=new ProcessBuilder ("CMD", "/C", "ipconfig");
      Process p=pb.start();
      InputStream is=p.getInputStream();
      int c;
      while ((c=is.read()) != -1) {
            System.out.print((char) c);
      }
      is.close();
      System.exit(0);
     }
}
```

Ejemplo 3 : Leer Errores de la Consola

```
public static void main(String[] args) throws Exception {

      ProcessBuilder pb=new ProcessBuilder ("CMD","/C", "IPCONFIGURAR");
      Process p=pb.start();
      InputStream is=p.getInputStream();
      int c;
      while ((c=is.read()) !=-1) {
            System.out.print((char) c);
}
      is.close();
//Lo mismo que el InputStream pero getErrorStream
      InputStream isError =p.getErrorStream();
      int cError;
      while ((cError=isError.read()) != -1) {
            System.out.print((char) cError);
      }
      isError.close();
    }
}
```

Por favor, no hacer un “throws Exception” en el Main, que eso no se hace.
También hará falta importar “ **java.io.*** ”

Programación recurrente

Consiste en la existencia simultánea de varios procesos en ejecución.
Puedo escuchar un video de Youtube mientras programo con Eclipse.

Programas recurrentes

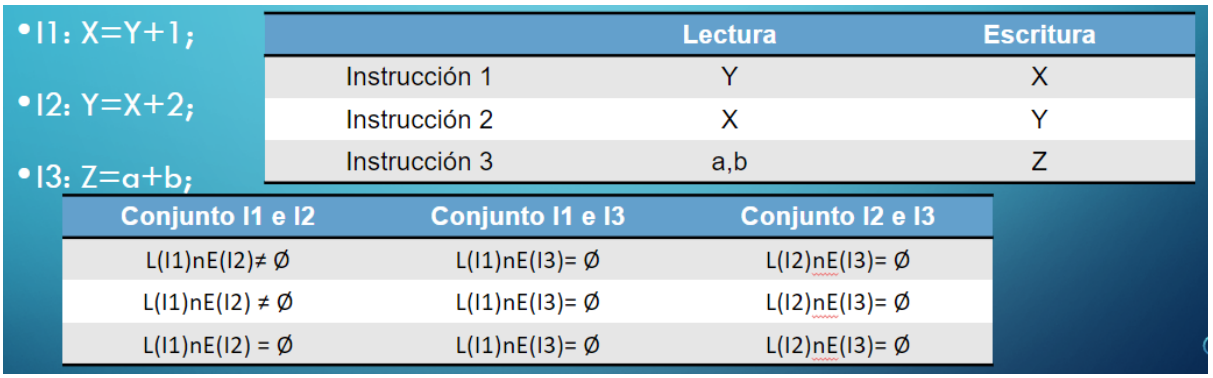
Un programa concurrente define un conjunto de acciones que pueden ser ejecutadas simultáneamente.
Supongamos que tenemos estos dos conjuntos de instrucciones.

Condiciones de Bernstein

- Necesitaremos dos instrucciones (mínimo).
- **Conjunto de lectura:** Las variables que toma como dato.
- **Conjunto de escritura:** Variable a la que se accede y se sobrescribe.

Condiciones:
L(Ii)nE(Ij)= Ø
E(Ii)nL(Ij)= Ø
E(Ii)nE(Ij)= Ø

O sea, que no pisemos las lecturas de una variable con las escrituras de la otra; ni ambas escrituras.



	Lectura	Escritura
Instrucción 1	Y	X
Instrucción 2	X	Y
Instrucción 3	a,b	Z

Conjunto I1 e I2	Conjunto I1 e I3	Conjunto I2 e I3
L(I1)nE(I2)≠ Ø	L(I1)nE(I3)= Ø	L(I2)nE(I3)= Ø
L(I1)nE(I2) ≠ Ø	L(I1)nE(I3)= Ø	L(I2)nE(I3)= Ø
L(I1)nE(I2) = Ø	L(I1)nE(I3)= Ø	L(I2)nE(I3)= Ø

En los programas secuenciales hay un orden fijo de ejecución, sin embargo, en los programas concurrentes puede haber solapamiento de instrucciones dando lugar a diferentes resultados.

Diferencias entre Multiprogramación y Multiproceso

- La multiprogramación da la sensación de estar haciendo varios procesos a la vez.
- La multiprogramación maximiza el uso de la CPU, mientras que el multiproceso ahorra tiempo al ejecutar varios procesos.
- En multiprogramación los procesos no están diseñados para trabajar entre sí, mientras que en multiproceso se permite la comunicación.
- En la multiprogramación los recursos se comparten, mientras que en el multiproceso cada proceso tiene sus recursos asignados.