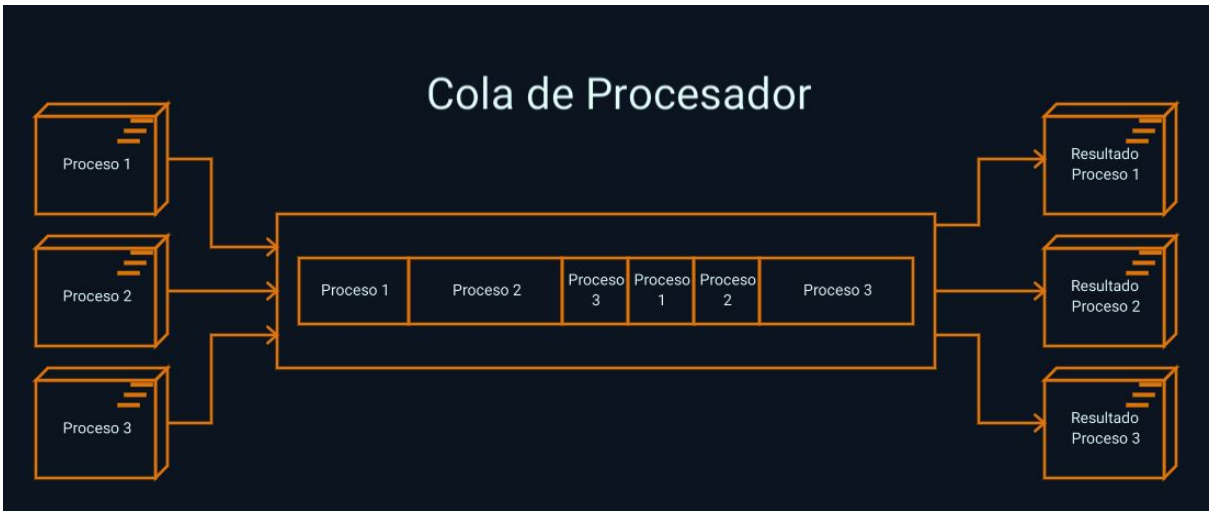
Prácticas 1



Pablo Ramilo Costal

1.-

La programación concurrente son unas técnicas de programación que permiten expresar la concurrencia entre tareas y solución de los problemas de comunicación y sincronización entre procesos. La programación concurrente es la ejecución simultánea de múltiples tareas interactivamente. Estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Las tareas se pueden ejecutar en una sola CPU (multiprogramación), en varios procesadores, o en una red de computadores distribuidos. El principal problema de la programación concurrente corresponde a no saber en qué orden se ejecutan los programas (en especial los programas que se comunican).

Los procesos concurrentes tienen las siguientes características:

- Indeterminismo: Las acciones que se especifican en un programa secuencial tienen un orden total, pero en un programa concurrente el orden es parcial, ya que existe una incertidumbre sobre el orden exacto de ocurrencia de ciertos sucesos, esto es, existe un indeterminismo en la ejecución. De esta forma si se ejecuta un programa concurrente varias veces puede producir resultados diferentes partiendo de los mismos datos.

- Interacción entre procesos: Los programas concurrentes implican interacción entre los distintos procesos que los componen:

  - Los procesos que comparten recursos y compiten por el acceso a los mismos.

  - Los procesos que se comunican entre sí para intercambiar datos.
  -

- Gestión de recursos: Los recursos compartidos necesitan una gestión especial. Un proceso que desee utilizar un recurso compartido debe solicitar dicho recurso, esperar a adquirirlo, utilizarlo y después liberarlo. Si el proceso solicita el recurso pero no puede adquirirlo en ese momento, es suspendido hasta que el recurso esté disponible. La gestión de recursos compartidos es problemática y se debe realizar de tal forma que se eviten situaciones de retraso indefinido (espera indefinidamente por un recurso) y de deadlock (bloqueo indefinido o abrazo mortal).

- Comunicación: La comunicación entre procesos puede ser síncrona, cuando los procesos necesitan sincronizarse para intercambiar los datos, o asíncrona, cuando un proceso que suministra los datos no necesita esperar a que el proceso receptor los recoja, ya que los deja en un buffer de comunicación temporal.

2.-

Concurrencia y paralelismo son dos conceptos relacionados pero distintos.

La concurrencia es, esencialmente, Que dos o más cálculos ocurran dentro del mismo período de tiempo. Tanto la tarea A como la B deben suceder independientemente una de otra, A comienza a ejecutarse y B comienza antes de que termine A.
El paralelismo significa que dos o más cálculos ocurren simultáneamente, el paralelismo es una forma de implementar la concurrencia, pero no es la única.

El paralelismo consta en tener varias CPU trabajando en las diferentes tareas al mismo tiempo

La concurrencia describe un problema (dos cosas deben suceder juntas), mientras que el paralelismo describe una solución (dos núcleos de procesador se utilizan para ejecutar dos cosas simultáneamente).

3.-

-¿Qué entiendes por contexto de un proceso?
Es el mínimo conjunto de datos utilizado por una tarea que debe ser guardado para permitir su interrupción en un momento dado, y una posterior continuación desde el punto en el que fue interrumpida en un momento futuro.

- ¿A que se refiere el concepto de multiprogramación?
-

Al existir una memoria y muchos procesos que quieren acceder a ella, esta alternancia rápida entre estos procesos (CPU y muchos procesos quieren utilizarla) por acceder a la memoria es lo que llamamos multiprogramación que da ilusión de paralelismo.

- ¿Qué entiendes por demonio o daemon?

Es un tipo especial del proceso que se ejecuta en segundo plano, se encuentra cargado en memoria esperando alguna señal para ser ejecutado. Este tipo de programas se ejecutan de manera continua, y aunque se intente cerrar o matar el proceso, este continúa su ejecución o se reinicia automáticamente.

- ¿Cual es la diferencia principal entre UNIX y Windows a la hora de organizar los procesos?

  En windows no hay jerarquía: todos son iguales.y en UNIX un proceso (padre) crea otros (hijos).

- ¿Cuántos padres como máximo puede tener un proceso en UNIX?

Uno solo.

- ¿Cuantos hijos puede tener un proceso en UNIX?

Muchos hijos.

- ¿Cuales son los estados básicos de un proceso? ¿Cuando se encuentra un proceso en cada estado?

-Nuevo: programa que entra en ejecución, tiene el proceso de control asignado pero no ha sido cargado en memoria, con suficientes recursos entrará en listo.

-Listo: no se estará ejecutando pero estará asociado en el mapa de memoria.

-Ejecución: la CPU queda libre y el sistema tiene que decidir que procesos de los listos que están en memoria va a darle la CPU, es un proceso crítico.

-Bloqueado: momento en el que un proceso necesita entrar al disco o esta esperando algún evento y por tanto se bloquea y el sistema operativo le quita la cpu y la lleva a memoria o disco para posiblemente dársela a otro proceso y le da el estado bloqueado a el proceso que espera el evento.

-Terminado: estado saliente, es cuando termina el proceso.

-¿Qué es un cambio de contexto? Pon un ejemplo

Acción de realizar cambio entre distintos procesos, un ejemplo sería cuando tenemos un procesos que se está ejecutando en un momento que tiene una serie de información en la CPU utilizándose y al cambiar a otro proceso ponemos la información de ese proceso en la CPU.

4.-

Un bloqueo mutuo (interbloqueo, traba mortal, deadlock) es, en el sistema operativo, el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. A diferencia de otros problemas de concurrencia de procesos, no existe una solución general para los interbloqueos.

Todos los interbloqueos surgen de necesidades que no pueden ser satisfechas, por parte de dos o más procesos. En la vida real, un ejemplo puede ser el de cuatro autos que se encuentran en una intersección en el mismo momento. Cada uno necesita que otro se mueva para poder continuar su camino, y ninguno puede continuar. Los recursos compartidos en este caso son los cuatro cuadrantes. El auto que se dirije de oeste a este, por ejemplo, necesita de los cuadrantes suroeste y sureste.

5.-

ProcessBuilder.

```
package io.github.picodotdev.blogbitix.javaprocess;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.math.BigDecimal;
import java.text.MessageFormat;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

public class Main {

   public static void main(String[] args) throws Exception {
      // Iniciar el proceso
      ProcessBuilder builder = new ProcessBuilder().command("cat",
"/proc/uptime");
      Process process = builder.start();

      // Alternativa a ProcessBuilder
      //Process process = Runtime.getRuntime().exec(new String[] { "cat",
"/proc/uptime" });

      // Esperar a que termine el proceso y obtener su valor de salida
      process.waitFor(10, TimeUnit.SECONDS);
```

```
        int value = process.exitValue();
        if (value != 0) {
            throw new Exception(MessageFormat.format("Código de salida con
error (%d)", value));
        }

        // Obtener la salida del proceso
        BufferedReader br = new BufferedReader(new
InputStreamReader(process.getInputStream(), "UTF-8"));
        String result = br.lines().collect(Collectors.joining("\n"));
        br.close();

        // Obtener el tiempo desde el inicio del sistema
        String seconds = result.split(" ")[0];
        System.out.printf("Segundos desde el inicio del sistema: %.2f", new
BigDecimal(seconds));
    }
}
```

6.-

Los algoritmos de planificación se encargan de asegurar que un proceso no monopoliza el procesador. Un proceso puede estar en 3 estados distintos “Listo” “Bloqueado” y “En Ejecución”. Los procesos son almacenados en una lista junto con la información que indica en qué estado está el proceso, el tiempo que ha usado el CPU, etc.

Tipos:

FCFS (*First-Come, First-Served*)

Los FCFS o también llamado FIFO (*First In, First Out*) es muy sencillo y simple, pero también el que menos rendimiento ofrece, básicamente en este algoritmo el primer proceso que llega se ejecuta y una vez terminado se ejecuta el siguiente.

SJF (*Shortest Job First*).

Este algoritmo siempre prioriza los procesos más cortos primero independientemente de su llegada y en caso de que los procesos sean iguales utilizara el método FIFO anterior, es decir, el orden según entrada. Este sistema tiene el riesgo de poner siempre al final de la cola los procesos más largos por lo que nunca se ejecutarán, esto se conoce como **inanición**.

SRTF (*Short Remaining Time Next*).

Añadiendo la expulsión de procesos al algoritmo SJF obtenemos SRTF, éste será capaz de expulsar un proceso largo en ejecución para ejecutar otros más cortos. El problema que puede surgir es que un proceso largo puede llegar a expulsarse muchas veces y nunca terminar debido a la ejecución de otros más cortos.

Round Robin.

Por último el *Round Robin*, es uno de los más complejos y difíciles de implementar, asigna a cada proceso un tiempo equitativo tratando a todos los procesos por igual y con la misma prioridad.

Este algoritmo es circular, volviendo siempre al primer proceso una vez terminado con el último, para controlar este método a cada proceso se le asigna un intervalo de tiempo llamado *quantum* o cuanto (para definirlo se utiliza esta regla, el 80% de los procesos tienen que durar menos tiempo que el *quantum* definido).

Pueden suceder dos casos con este método:

-El proceso es menor que el quantum: Al terminar antes se planifica un nuevo proceso.

-El proceso es mayor que el quantum: Al terminar el quantum se expulsa el proceso dando paso al siguiente proceso en la lista. Al terminar la iteración se volverá para terminar el primer proceso expulsado.

Código java:

```
package batch;

import java.io.BufferedReader;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;



import javax.swing.JOptionPane;



public class HolaUsuario {

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



		List<String> list = new ArrayList<String>();



		String line;

		String ruta="helloBatch.bat";
```

```
String mensaje = JOptionPane.showInputDialog("Cual es tu nombre?:
");



try {

File bat = new File(ruta);

Process pb = new ProcessBuilder(bat.getAbsolutePath(),
mensaje).start();



InputStream is = pb.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);



while ((line = br.readLine()) != null) {

list.add(line);

}



String mostrar = null;

for (String saludo : list) {

mostrar = saludo;

}
```

```
                JOptionPane.showMessageDialog(null, mostrar);


            } catch (Exception e) {
                e.printStackTrace();
            }



        }
}
```