1-¿PROGRAMACIÓN CONCURRENTE?:

La concurrencia en software implica la existencia de diversos flujos de control en un mismo programa colaborando para resolver un problema.
Un programa concurrente da lugar, durante su ejecución, a un proceso con varios hilos de ejecución, las operaciones (software) en un programa son concurrentes si pueden ejecutarse en paralelo, aunque no necesariamente deben ejecutarse así.

2-¿CUALES SON LAS DIFERENCIAS ENTRE PROGRAMACIÓN CONCURRENTE Y PROGRAMACIÓN PARALELA?

La programación CONCURRENTE permite tener en ejecución varias tareas sin que se ejecuten simultaneamente, mientras que LA PARALELA, permite que se ejecuten simultaneamente varias tareas.

3-VISUALIZA LOS SIGUIENTES VÍDEOS Y CONTESTA:

3.1-¿A QUE SE REFIERE EL CONCEPTO DE MULTIPROGRAMACIÓN?

Se llama multiprogramación a una técnica por la que dos o más procesos pueden alojarse en la memoria principal y ser ejecutados concurrentemente por el procesador o CPU.

3.2-¿QUÉ ENTIENDES POR DEMONIO O DAEMON?

Proceso no interactivo que se esta ejecutando en 2º plano sin que el usuario intervenga

3.3-¿CUAL ES LA DIFERENCIA PRINCIPAL ENTRE UNIX Y WINDOWS A LA HORA DE ORGANIZAR LOS PROCESOS?

Un proceso es básicamente un entorno formado por todos los recursos necesarios para ejecutar programas. Desde el punto de vista del SO, un proceso es un objeto más que hay que gestionar y al cual hay que dar servicio, en unix encontramos.
Los procesos EN LINUX pueden ser divididos en tres categorías: interactivos, tiempo real o por lotes, mientras EN WINDOWS, la planificación de procesos se basa en la utilización de colas múltiples de prioridades. Posee 23 niveles de colas clasificadas de la 31-16 en clase de tiempo real y las demás en clase variable.Cada cola es manejada mediante

Round-Robin, pero si llega un proceso con mayor prioridad, se le es asignado el procesador.

3.4-¿CUANTOS PADRES COMO MÁXIMO PUEDE TENER UN PROCESO EN UNIX?

Cada proceso hijo es un clon del proceso padre, por lo que tiene un proceso padre unico, si el proceso hijo terminara antes que el padre, tendriamos un proceso zombi.

3.5-¿CUANTOS HIJOS PUEDE TENER UN PROCESO EN UNIX?

Cada vez que ejecutamos un comando "fork" se crea un proceso hijo, pudiendose crear tantos como se necesiten

3.6-¿CUALES SON LOS ESTADOS BÁSICOS DE UN PROCESO? ¿CUANDO SE ENCUENTRA UN PROCESO EN CADA ESTADO?

ACTIVO: el proceso está empleando la CPU, por tanto, está ejecutándose. Puede haber tantos procesos activos como procesadores haya disponibles.
PREPARADO: el proceso no está ejecutándose, pero es candidato a pasar a estado activo. Es el planificador el que, en base a un criterio de planificación, decide qué proceso selecciona para pasar a estado activo.
BLOQUEADO: el proceso está pendiente de un evento externo que le ha hecho bloquear, tales como una operación de lectura/escritura, la espera de finalización de un proceso hijo, una señal o una operación sobre un semáforo. El dispositivo/hecho externo "avisa" al S.O. cuando ha terminado la acción que realizaba mediante una INTERRUPCIÓN, dejando el S.O. lo que está haciendo para atender a esta última. Tras esto, el S.O. comprueba cuales son los procesos que fueron bloqueados por ese evento externo, cambiándolos al estado de preparado.

3.7-¿QUÉ ES UN CAMBIO DE CONTEXTO? PON UN EJEMPLO

Ejecucion de una rutina cuyo propósito es parar la ejecución de un hilo o proceso para dar paso a la ejecución de otro distinto.

Por ejemplo, La entrada de un proceso en el estado dormido.

4-¿QUE ES UN INTERBLOQUEO (DEADLOCK)? ¿CUALES SON LAS CONDICIONES NECESARIAS PARA QUE SE PRODUZCA? EXPLÍCALAS BREVEMENTE

Es 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.
Estas condiciones deben cumplirse simultáneamente y no son totalmente independientes entre ellas.

Sean los procesos P0, P1, ..., Pn y los recursos R0, R1, ..., Rm:

CONDICIÓN DE EXCLUSIÓN MUTUA: existencia de al menos un recurso compartido por los procesos, al cual solo puede acceder uno simultáneamente.
CONDICIÓN DE RETENCIÓN Y ESPERA: al menos un proceso Pi ha adquirido un recurso Ri, y lo retiene mientras espera al menos un recurso Rj que ya ha sido asignado a otro proceso.
CONDICIÓN DE NO EXPROPIACIÓN: los recursos no pueden ser expropiados por los procesos, es decir, los recursos solo podrán ser liberados voluntariamente por sus propietarios.
CONDICIÓN DE ESPERA CIRCULAR: dado el conjunto de procesos P0...Pm(subconjunto del total de procesos original),P0 está esperando un recurso adquirido por P1, que está esperando un recurso adquirido por P2,... ,que está esperando un recurso adquirido por Pm, que está esperando un recurso adquirido por P0.
Esta condición implica la condición de retención y espera.

5-¿QUÉ CLASE UTILIZARÍAS EN JAVA PARA CREAR UN PROCESO? PON UN EJEMPLO DE COMO LA EMPLEARÍA

public final class ProcessBuilder()
Cada ProcessBuilderinstancia gestiona una colección de atributos de proceso. El start()método crea una nueva Processinstancia con esos atributos.
El start()método se puede invocar repetidamente desde la misma instancia para crear nuevos subprocesos con atributos idénticos o relacionados.

Aquí hay un ejemplo que inicia un proceso con un directorio y entorno de trabajo modificado, y redirige la salida estándar y el error para que se agreguen a un archivo de registro:

```
ProcessBuilder pb =  new ProcessBuilder("myCommand", "myArg1", "myArg2");
        Map<String, String> env = pb.environment();
        env.put("VAR1", "myValue");
        env.remove("OTHERVAR");
        env.put("VAR2", env.get("VAR1") + "suffix");
        pb.directory(new File("myDir"));
        File log = new File("log");
        pb.redirectErrorStream(true);
        pb.redirectOutput(Redirect.appendTo(log));
Process p = pb.start();
        assert pb.redirectInput() == Redirect.PIPE;
        assert pb.redirectOutput().file() == log;
        assert p.getInputStream().read() == -1;
```

6-¿QUÉ ES UN ALGORITMO DE PLANIFICACIÓN (AP)? ¿CUANTOS TIPOS DE AP CONOCES? DESCRIBE CADA TIPO DE AP

Un algoritmo de planificación se utiliza para calcular los recursos que consume otro algoritmo o conjunto de algoritmos (programa) al realizar una determinada tarea.

Existen varios tipos:

1.FCFS “FIRTS-COME, FIRST-SERVED”:El procesador ejecuta cada proceso hasta que termina, por tanto, los procesos que en cola de procesos preparados permanecerán encolados en el orden en que lleguen hasta que les toque su ejecución

2.SJF “SHORTEST JOB FIRST”:En este algoritmo , da bastante prioridad a los procesos más cortos a la hora de ejecución y los coloca en la cola.

3.SRTF “SHORT REMAINING TIME FIRST”:es similar al sjf, con la diferencia de que si un nuevo proceso pasa a listo se activa el dispatcher para ver si es más corto que lo que queda por ejecutar del proceso en ejecución. si es así, el proceso en ejecución pasa a listo y su tiempo de estimación se decremento con el tiempo que ha estado ejecutándose.

4.ROUND ROBIN:Es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.

7-IMPLEMENTA EN JAVA EL EJERCICIO 9 CITADO EN LA SECCIÓN DE EJERCICIOS Y AÑADE EL PROYECTO A LA CARPETA P1 DEL REPOSITORIO PSP DE TU GITHUB, AÑADE ADEMÁS TU CÓDIGO AL FINAL DEL DOCUMENTO PDF DE LA PRÁCTICA.

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Main {

	public static final String RUTA =
	"D:\\eclipse\\workspace\\PruebaBatchJava\\batch.bat";

	 public static void main(String[] args) throws IOException {
		leeFichero(RUTA);
	}

		 public static void leeFichero(String RUTA) throws IOException{

			ProcessBuilder p = new ProcessBuilder(RUTA,"Juan");
				final Process command = p.start();
			String cadena;
	 BufferedReader b =
	newBufferedReader(newinputStreamReader(command.getInputStream()));
		 while((cadena = b.readLine())!=null) {
```

```
                System.out.println(cadena);
                }
          b.close();
        }

}
```