# Sombrea la respuesta correcta (0,25 c/u).

1. **¿Con qué comando se accede a los procesos en Windows /Linux?**
2. Processlist/process
3. Tasklist/ps
4. Tasklist/process
5. Process/ps
6. **¿Qué información nos muestra PID?**
7. La ID del proceso padre
8. ID del proceso
9. El procesador que tiene asignado el proceso
10. Ninguna de las anteriores
11. **¿Qué información nos da PPID?**

a) La ID del proceso padre

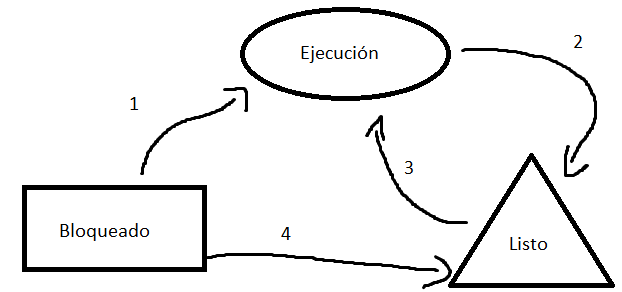
b) ID del proceso

c) El procesador que tiene asignado el proceso

d) Ninguna de las anteriores

1. **¿Qué información nos da STIME**
2. Tiempo que lleva ejecutándose un proceso
3. Hora a la que empezó a ejecutarse el proceso
4. Nos permite hacer un Set Time
5. Ninguna de las anteriores

# Explica el estado de los procesos, sus posibles transacciones y haz el gráfico. (1 punto).



Ejecución a Bloqueado: El proceso espera que termine o inicie un evento.

Bloqueado a Listo: El proceso que esperaba se ejecuta.

Listo a Ejecución: El Sistema Operativo le da tiempo de CPU al proceso.

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

# Escribe un código en Java que permita abrir Chrome. (2 puntos).

package ejecuciones;

public class Chrome {

  public static *void* main(String[] *args*) {

    //ProcessBuilder pb = new ProcessBuilder("\"C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe\"");

    ProcessBuilder pb = **new** ProcessBuilder("CMD","/C","start chrome");

    try {

      Process p = pb.start();

    } catch (Exception *e*) {

      e.printStackTrace();

    }

  }

}

# Escribe un código en Java que nos muestre por la consola de Eclipse los procesos que tengamos abiertos. (2 puntos).

package ejecuciones;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class ListarProcesos {

    public static *void* main(String[] *args*) {

        listarProcesosWindows();

    }

    private static *void* listarProcesosWindows() {

        try {

            Process process = Runtime.getRuntime().exec("tasklist");

            BufferedReader reader = **new** BufferedReader(

**new** BufferedReader(**new** InputStreamReader(process.getInputStream())));

            System.out.println("Procesos en ejecución:");

            String line;

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

                System.out.println(line);

            }

            reader.close();

        } catch (IOException *e*) {

            e.printStackTrace();

        }

    }

}

# Crea el cuadro de Bernstein e indica qué instrucciones son concurrentes. (2 puntos).

Instrucción 1: p1=a\*square;  
Instrucción 2: p2=b\*x;  
Instrucción 3: square=x\*x;  
Instrucción 4: z= m1 + m2;  
Instrucción 5: y= z + c;

|  |  |  |
| --- | --- | --- |
| Instrucción | Lectura | Escritura |
| p1=a\*square | a,square | p1 |
| p2=b\*x | b,x | p2 |
| square=x\*x | x,x | square |
| z= m1 + m2 | m1,m2 | z |
| y= z + c | z,c | y |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instrucciones | Lectura y escritura | Escritura y lectura | Escritura y escritura | Concurrencia |
| 1 y 2 | a,square∩p2 | p1∩b,x | p1∩p2 | Concurrente |
| 1 y 3 | a,square∩square | p1∩x,x | p1∩square | No Concurrente |
| 1 y 4 | a,square∩z | p1∩m1,m2 | p1∩z | Concurrente |
| 1 y 5 | a,square∩y | p1∩z,c | p1∩y | Concurrente |
| 2 y 3 | b∩square | p2∩x,x | p2∩square | Concurrente |
| 2 y 4 | b∩z | p2∩m1,m2 | p2∩z | Concurrente |
| 2 y 5 | b∩y | p2∩z,c | p2∩y | Concurrente |
| 3 y 4 | m1,m2∩z | square∩m1,m2 | square∩z | Concurrente |
| 3 y 5 | m1,m2∩y | square∩z,c | square∩y | Concurrente |
| 4 y 5 | z,c∩y | z∩z,c | z∩y | No concurrente |

# Desarrolla las ventajas de la programación concurrente en los monoprocesadores. (1 punto).

Respuesta más rápida: Permite la ejecución simultánea de tareas para un sistema más ágil.

Eficiencia en E/S: Facilita la gestión eficiente de operaciones de entrada/salida, optimizando el uso del procesador.

Ejecución de tareas en segundo plano: Permite realizar tareas no intrusivas mientras se mantiene la capacidad de respuesta principal.

Mantenimiento de la capacidad de respuesta: Asegura que el sistema pueda manejar eficientemente múltiples operaciones sin sacrificar la experiencia del usuario.

# Desarrolla los dos problemas inherentes a la programación concurrente. (1 punto).

Condiciones de Carrera: Ocurren cuando múltiples se interponen entre ellos por modificar datos compartidos, haciendo que esos resultados sean impredecibles. Se resuelven mediante la sincronización, que regula el acceso a los datos.

Deadlocks: Situación donde los hilos quedan atrapados esperando que otros liberen recursos necesarios. Se deben prevenir y gestionar para que la aplicación que se esté usando no se congele