

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

Unidad 1: Programación multiproceso -

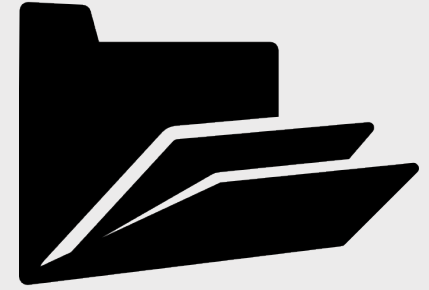
Profesor: Marcelino Penide Durán (sustituto)

marcelimpd@educastur.org

[Curso 2023 - 2024]

(DAM Distancia)

Contenidos unidad

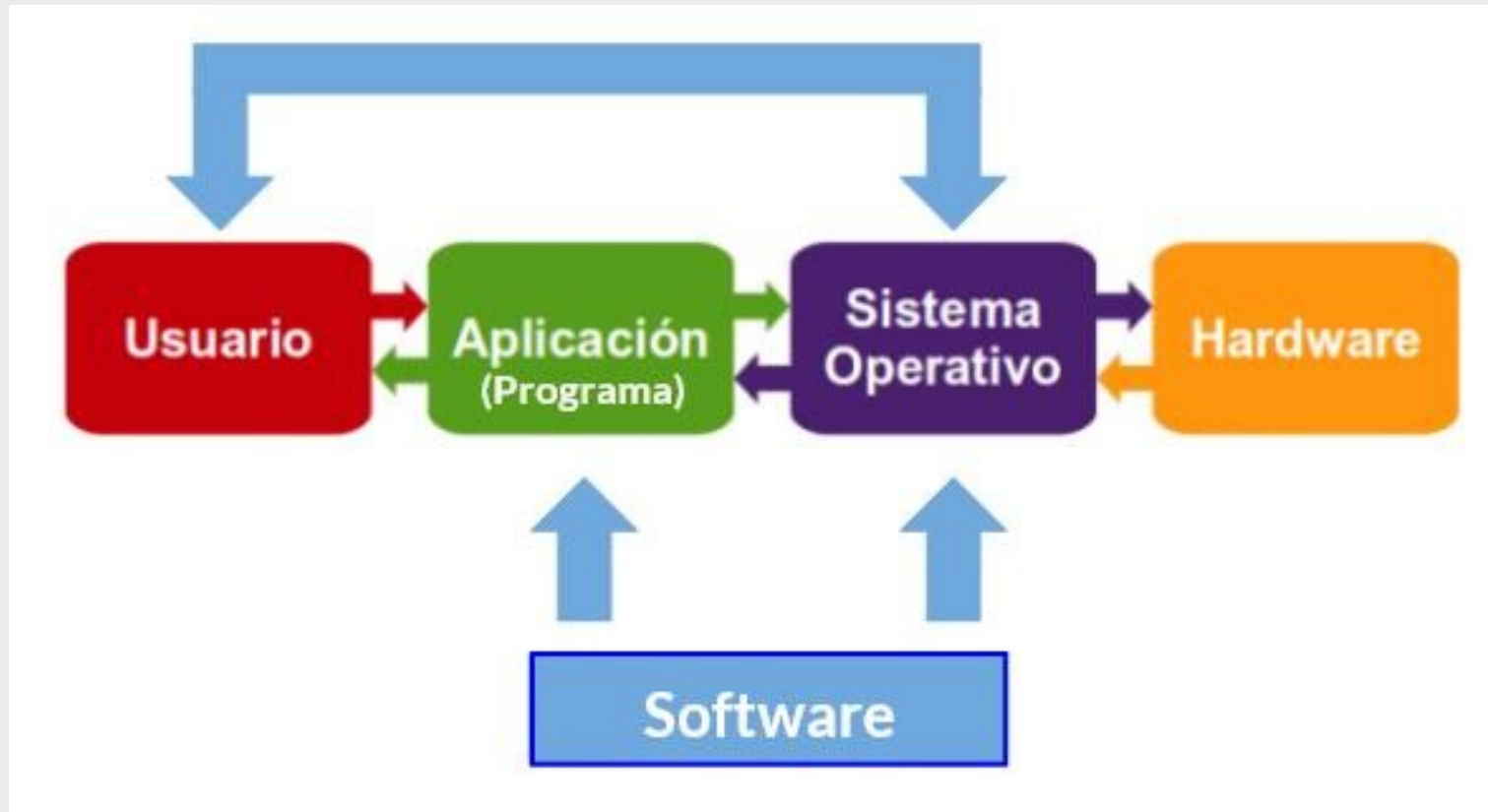


1. Introducción.
2. Gestión de procesos.
3. Procesos en Java.
4. Comandos gestión procesos.
5. Programación concurrente.
6. Comunicación entre procesos.
7. Sincronización entre procesos.
8. Programación paralela y distribuida.

1. INTRODUCCIÓN

Conceptos básicos (1/4)

- Sistema informático: hardware + software + usuario.



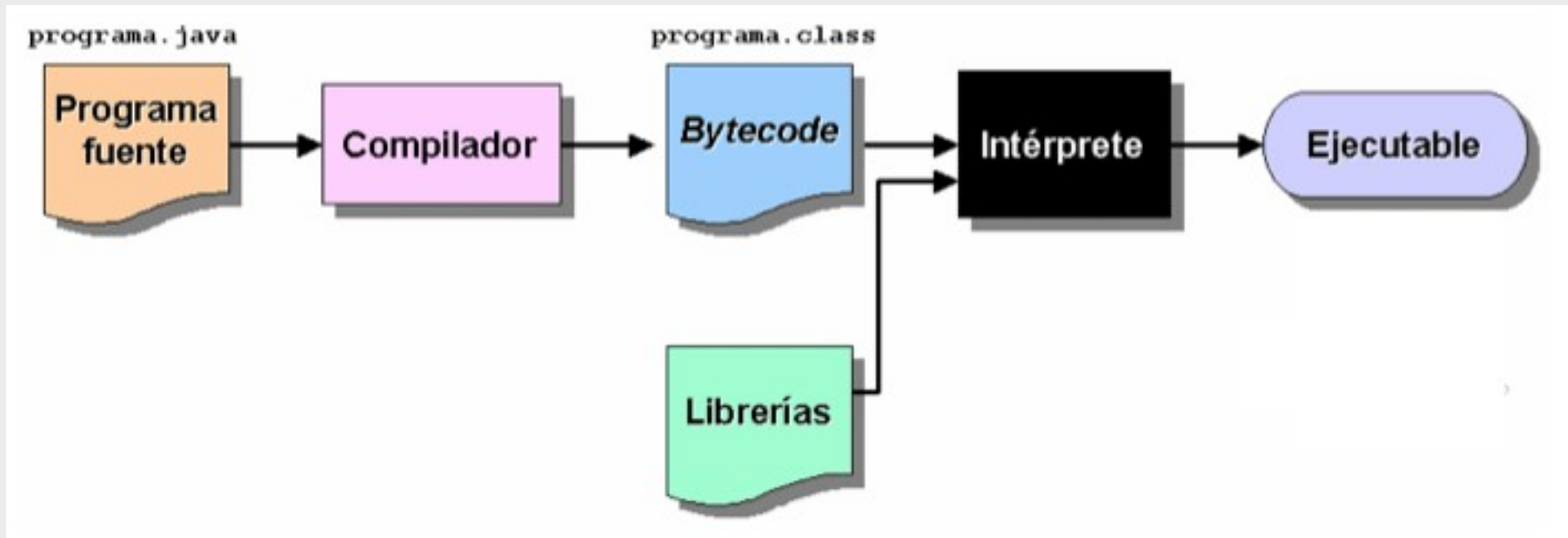
Conceptos básicos (2/4)

- Programa informático: Conjunto de instrucciones que ejecutadas en un ordenador realizarán una tarea.
- Aplicación: Tipo de programa informático, diseñado como herramienta para resolver de manera automática un problema específico del usuario.



Conceptos básicos (3/4)

- **Ejecutable:** Fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.



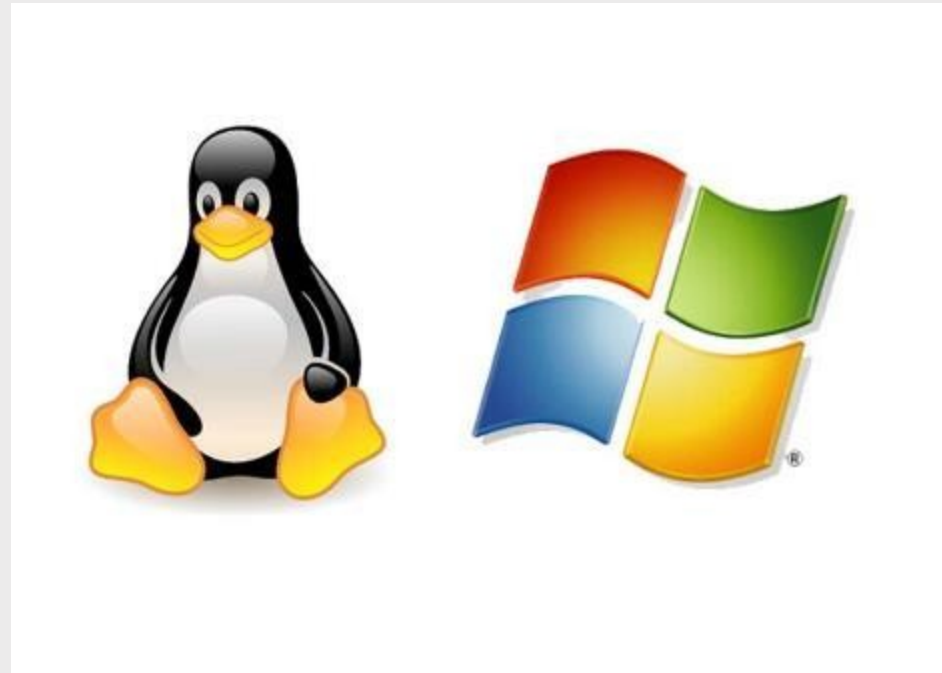
Conceptos básicos (4/4)

- **Proceso:** Programa en ejecución (entidad activa).
 - *En el SO > unidad de trabajo completa.*
 - *Habitualmente, un programa implica que se arranquen varios procesos (mínimo 1).*



Ejecutables

- Windows > .exe
- Linux > tienen activado permiso ejecución.



Tipos ejecutables



- **Binarios:** Conjunto de instrucciones que directamente son ejecutadas por el procesador del ordenador (código compilado). Ej.: C.
 - **Interpretados:** Conjunto de instrucciones que son interpretadas antes de ser ejecutadas por el procesador del ordenador (código interpretado). Ej.: Java.
-
- **Librerías:** Conjunto de funciones que permiten dar modularidad y reusabilidad a nuestros programas.



CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

2. GESTIÓN DE PROCESOS

Procesos

- En nuestro equipo se están ejecutando, al mismo tiempo, muchos procesos.
- Nuestros SO son multitarea.





Tipos procesos

- **Por lotes:** Formados por una serie de tareas, de las que el usuario sólo está interesado en el resultado final. Ej.: imprimir.
- **Interactivos:** Aquellas tareas en las que el proceso interactúa continuamente con el usuario y actúa de acuerdo a las acciones que éste realiza, o a los datos que suministra. Ej.: procesador textos.
- **Tiempo real:** Tareas en las que es crítico el tiempo de respuesta del sistema. Ej.: ordenador coche.

Funcionamiento (1/4)

- En nuestros equipos ejecutamos distintos procesos: por lotes e interactivos.
- Microprocesador > capaz de ejecutar miles de millones de instrucciones básicas en un segundo (es decir, muchas tareas).
- SO > encargado de decidir qué proceso puede entrar a ejecutarse o debe esperar.



Funcionamiento (2/4)

- SO > tiene que repartir el uso del microprocesador entre los distintos procesos:
 - *¿Qué le sucede a un proceso cuando no se está ejecutando?*
 - *¿Por qué el equipo hace otras cosas mientras que un proceso queda a la espera de datos?*



Funcionamiento (3/4)

■ Posible solución:



- *Procesos van avanzando posiciones en la cola de procesos activos hasta que les toca el turno.*
- *SO concede el uso de la CPU, a cada proceso durante un tiempo determinado y equitativo > quantum.*

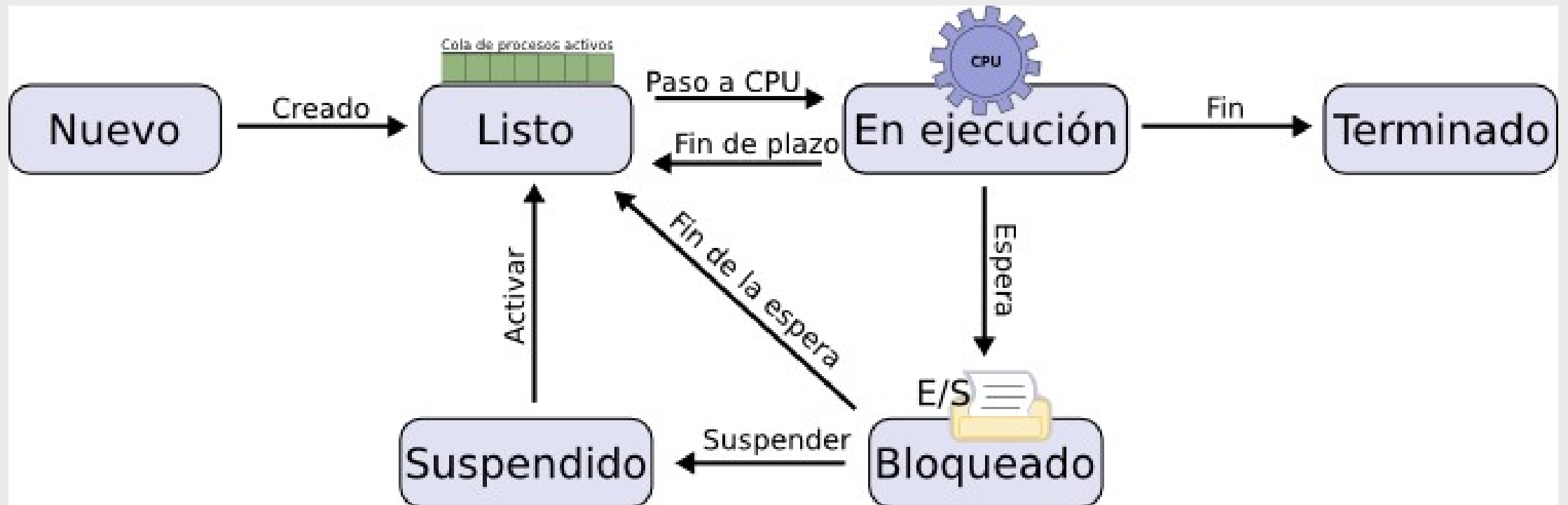
Funcionamiento (4/4)



■ Problemas:

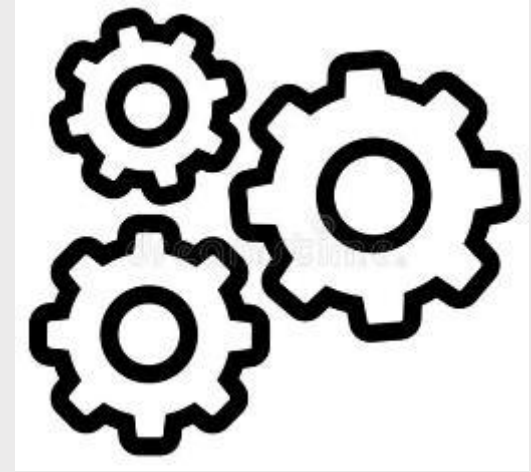
- *Cuando un proceso: necesita datos de un archivo, necesita una entrada de datos del usuario, tiene que imprimir o grabar datos,... > proceso bloqueado (en espera).*
- *Cuando la memoria RAM del equipo está llena, algunos procesos deben pasar a disco para dejar espacio > proceso suspendido.*
- *Procesos “críticos” (no pueden esperar) > procesos SO.*

Estados proceso



Ciclo vida proceso

- Conjunto de estados por los que transita.
- Estados básicos:
 - *Activo: en ejecución o listo.*
 - *En espera: bloqueado o suspendido.*



Recuerda: los procesos no pueden pasar por ellos mismos de listos a ejecución, es el SO el que decide.

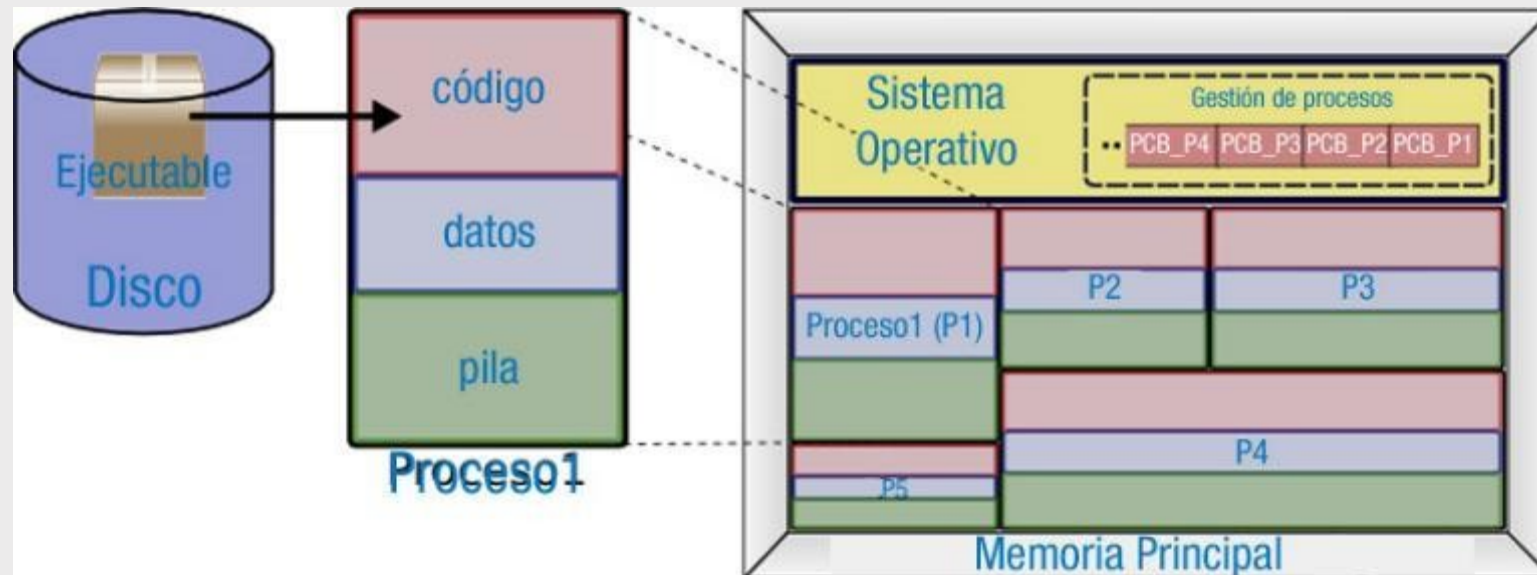
BCP / PCB

- Bloque de Control de Proceso / Process Control Block.
- Estructura de datos que contiene toda la información que requiere el SO para gestionar un proceso:
 - *PID: identificador del proceso.*
 - *Estado actual.*
 - *Espacio de direcciones de memoria.*
 - *Información para la planificación.*
 - ...



Componentes SO (1/2)

- Cargador: encargado de crear los procesos:
 - *Carga el proceso en memoria principal (RAM).*
 - *Crea una estructura de información llamada BCP / PCB.*

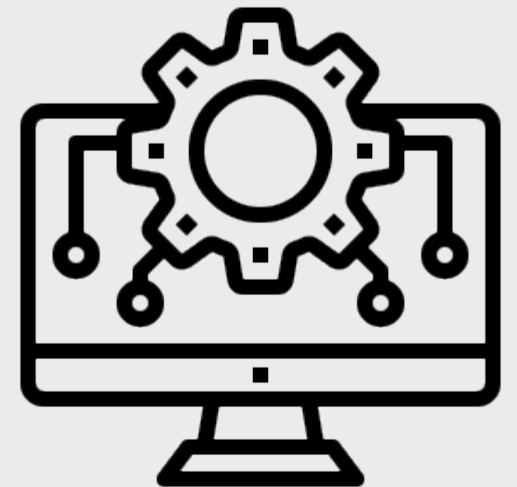


Componentes SO (2/2)

- Planificador: encargado de decidir el proceso que debe ejecutarse.



Algoritmo de planificación



Criterios planificación

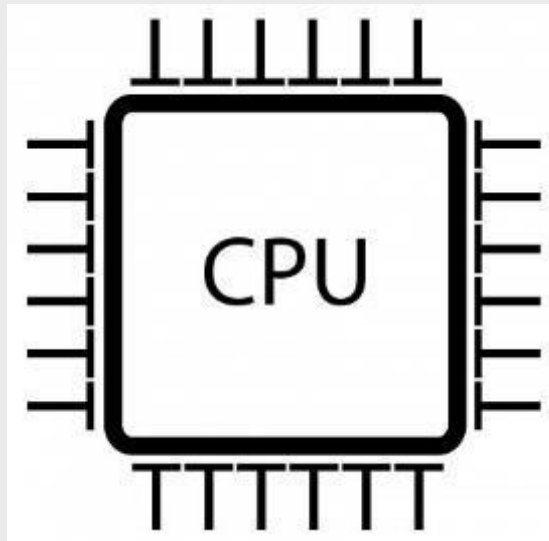


- Justicia. Todos deben tener CPU.
- Eficiencia. Minimizar CPU inactiva.
- Rendimiento. Maximizar trabajos/tiempo
- Tiempo de retorno. Min. tiempo de respuesta final.
- Tiempo de espera. Mínimo tiempo en espera (listo).
- Tiempo de respuesta. Tiempo de respuestas intermedias.

Objetivo: optimizar ejecución procesos.

Políticas planificación

- Expulsoras / expropiativas: Proceso puede perder la posesión del procesador sin solicitarlo.
- No expulsoras / no expropiativas: Proceso NO puede perder la posesión del procesador sin solicitarlo.



Algoritmos planificación



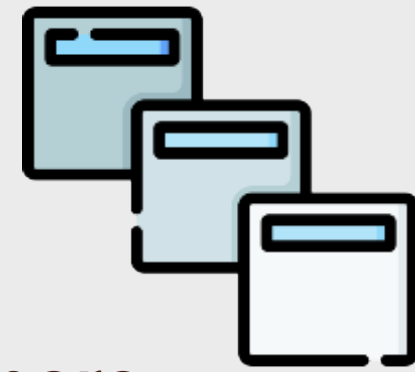
■ No expulsores / no expropiativos:

- *Algoritmo FIFO / FCFS [1].*
- *Algoritmo SJF [2].*
- *Algoritmo por prioridades [3].*
- ...

■ Expulsores / expropiativos:

- *Algoritmo por prioridades [3].*
- *Algoritmo Round-Robin [4].*
- ...

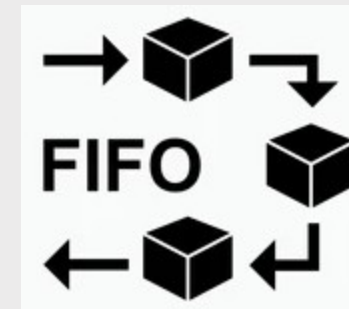
Algoritmo FIFO / FCFS [1]



- First In First Out / First Come First Served > primero en entrar – llegar, primero en salir - servir.

- Ventajas: 

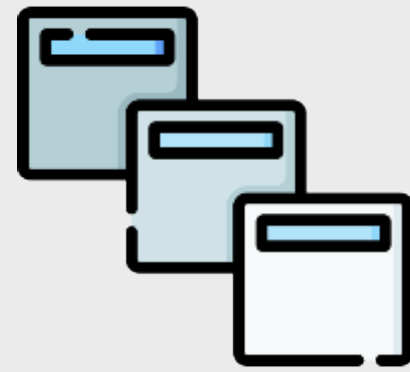
- *Fácil de implementar (simple).*
- *Eficiente si hay pocos procesos.*





- Inconvenientes: 

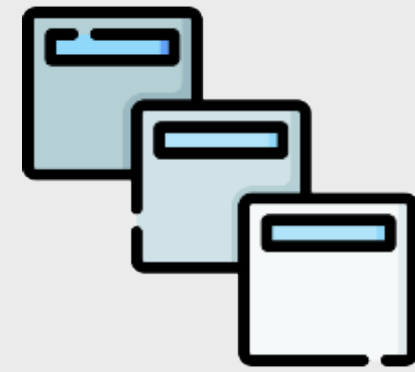
- *Mucho volumen de procesos > mucha cola.*
- *No eficiente con procesos interactivos.*



Algoritmo SJF [2]



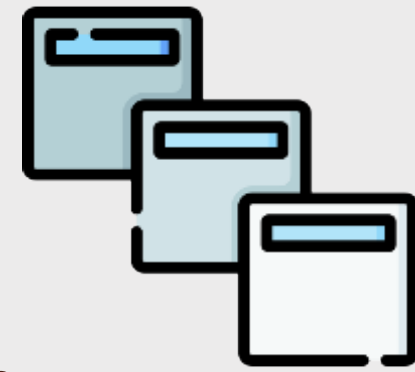
- Shortest Job First > primero el más corto.
- 2 procesos iguales > usa FIFO.
- Ventaja: 
 - *Procesos más cortos acaban primero.*
- Inconveniente: 
 - *Procesos más largos podrían llegar a no ejecutarse.*



Algoritmo por prioridades [3]



- Criterios establecer prioridades $>$ variados.
- Puede ser: expulsor / no expulsor.
- Ventaja: 
 - *Procesos más importantes (con prioridad alta) van antes.*
- Inconveniente: 
 - *Procesos con prioridad baja podrían no llegar a ejecutarse.*

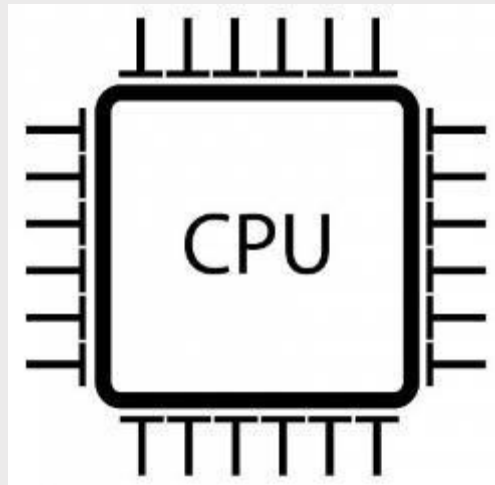
Algoritmo Round-Robin [4]



- Se usa un temporizador > cada proceso tiene un tiempo determinado (q = quantum).
- Similar a FIFO pero expulsor / expropiativo.
- Ventaja: 
 - *Más “justo”.*
- Inconvenientes: 
 - q grande > “se tarda mucho en atender” (FIFO).
 - q pequeño > sobrecarga / degradación aprovechamiento CPU.

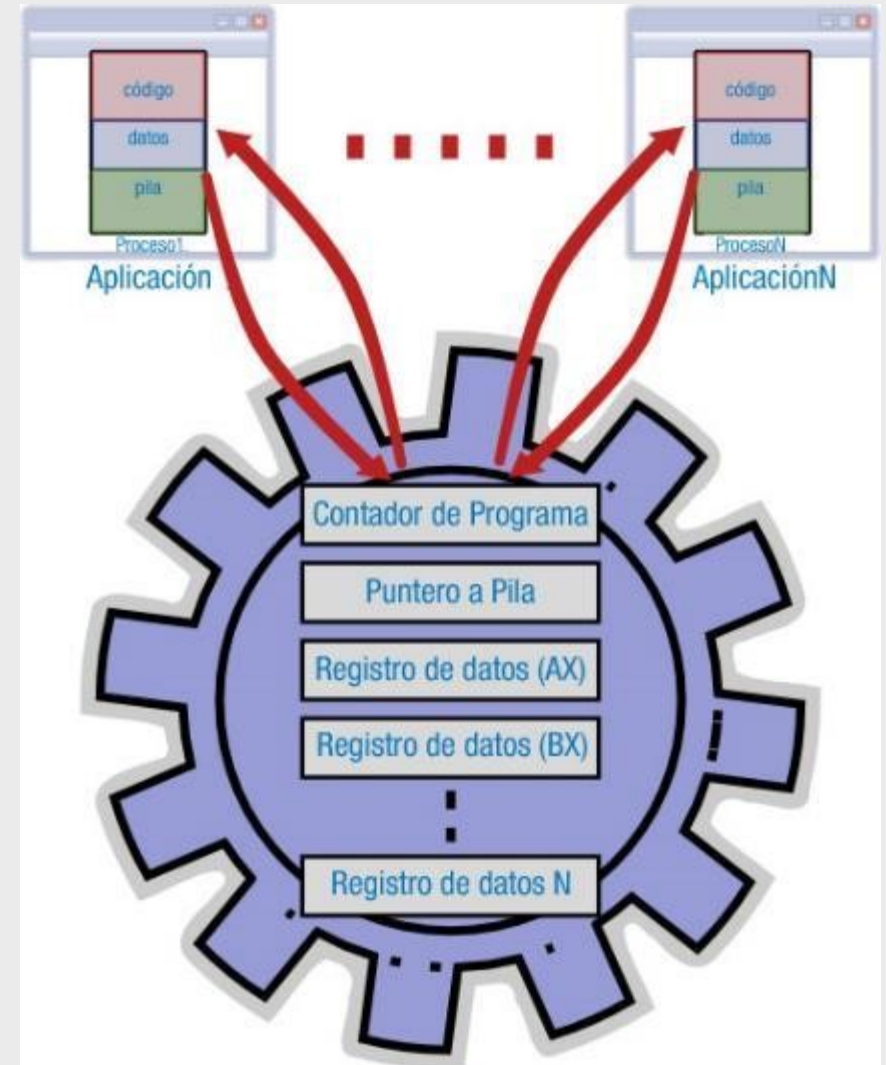
Cambio contexto CPU (1/2)

- Se realiza cada vez que la CPU cambia la ejecución de un proceso a otro distinto. Hay que:
 - *Guardar el estado actual de la CPU.*
 - *Restaurar el estado de CPU del proceso que se va a ejecutar.*



Cambio contexto CPU (2/2)

- Estado CPU > conjunto registros:
 - *Contador programa > almacena la dirección de la siguiente instrucción a ejecutar.*
 - *Puntero a pila > almacena el contexto de la CPU.*
 - ...





CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

3. PROCESOS EN JAVA

Creación de procesos

- Clase principal abstracta: `Process` (representa un proceso en Java).
- Clases (crear + interaccionar):
 - *`ProcessBuilder`*
 - *`Runtime`*



Clase: ProcessBuilder

| Método | Funcionalidad |
|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ProcessBuilder(String... comando)</code> <code>ProcessBuilder(List<String> comando)</code> | Construye un <code>ProcessBuilder</code> para un comando con los correspondientes argumentos de línea de comandos, en su caso. |
| <code>ProcessBuilder directory(File directorio)</code> <code>File directory()</code> | El primer método establece el directorio de trabajo del proceso, y el segundo método lo devuelve. Por defecto, el segundo método normalmente devuelve <code>null</code> . Si es así, se puede obtener el directorio de ejecución del proceso actual con <code>System.getProperty("user.dir")</code> . |
| <code>Map<String,String> environment()</code> | Devuelve el entorno de trabajo del proceso, que es una lista de asignación de valores para variables de entorno. |
| <code>Process start()</code> | Crea e inicia un proceso. |

Clase: ProcessBuilder (1/2)

■ Crear un proceso (bloc de notas) [1ª forma]:

```
import java.io.IOException;

public class E01_CrearProceso1 {

    public static void main(String[] args) {
        String nombreProceso = "notepad.exe";
        ProcessBuilder pb = new ProcessBuilder(nombreProceso);
        try {
            pb.start(); // Crear proceso
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Clase: ProcessBuilder (2/2)

■ Crear un proceso (VLC) [2ª forma]:

```
import java.io.IOException;

public class E01_CrearProceso1 {

    public static void main(String[] args) {
        String nombreProceso = "C:\\Archivos de programa\\VideoLAN\\VLC\\vlc.exe";
        ProcessBuilder pb = new ProcessBuilder();
        try {
            pb.command(nombreProceso); // Establecer proceso
            pb.start(); // Crear proceso
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Clase: Runtime

| Método | Funcionalidad |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>static Runtime getRuntime()</code> | Devuelve el objeto <code>Runtime</code> asociado con la aplicación de Java actual. |
| <code>Process exec(String command)</code> <code>Process exec(String[] comandoYArgs)</code> <code>Process exec(String[] comandoYArgs, String[] envp)</code> <code>Process exec(String[] cmdarray, String[] envp, File dir)</code> | Ejecuta un comando. Algunas variantes permiten especificar el entorno de ejecución y el directorio para ejecutar el comando. |
| <code>void exit(int status)</code> <code>void halt(int status)</code> | Termina la ejecución de la máquina virtual. El primer método lo hace de manera ordenada, y el segundo, de manera abrupta. |
| <code>int availableProcessors()</code> | Devuelve el número de procesadores disponibles para la máquina virtual de Java. |
| <code>long freeMemory()</code> | Devuelve la cantidad de memoria disponible para la máquina virtual. |

Clase: Runtime

■ Crear un proceso (bloc de notas):

```
import java.io.IOException;

public class E02_CrearProceso2 {

    public static void main(String[] args) {
        String nombreProceso = "notepad.exe";
        Runtime rt = Runtime.getRuntime();
        try {
            rt.exec(nombreProceso); // Crear proceso
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Clase: Process

■ Métodos interesantes (1/2):

- pid()
- info()



```
import java.io.IOException;

public class E03_ObtenerInfoProceso {

    public static void main(String[] args) {
        String nombreProceso = "notepad.exe";
        Runtime rt = Runtime.getRuntime();
        try {
            Process miProceso = rt.exec(nombreProceso); // Crear proceso
            // Mostrar PID proceso
            System.out.println("Número proceso: " + miProceso.pid());
            // Mostrar información proceso
            System.out.println("Información proceso: " + miProceso.info());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

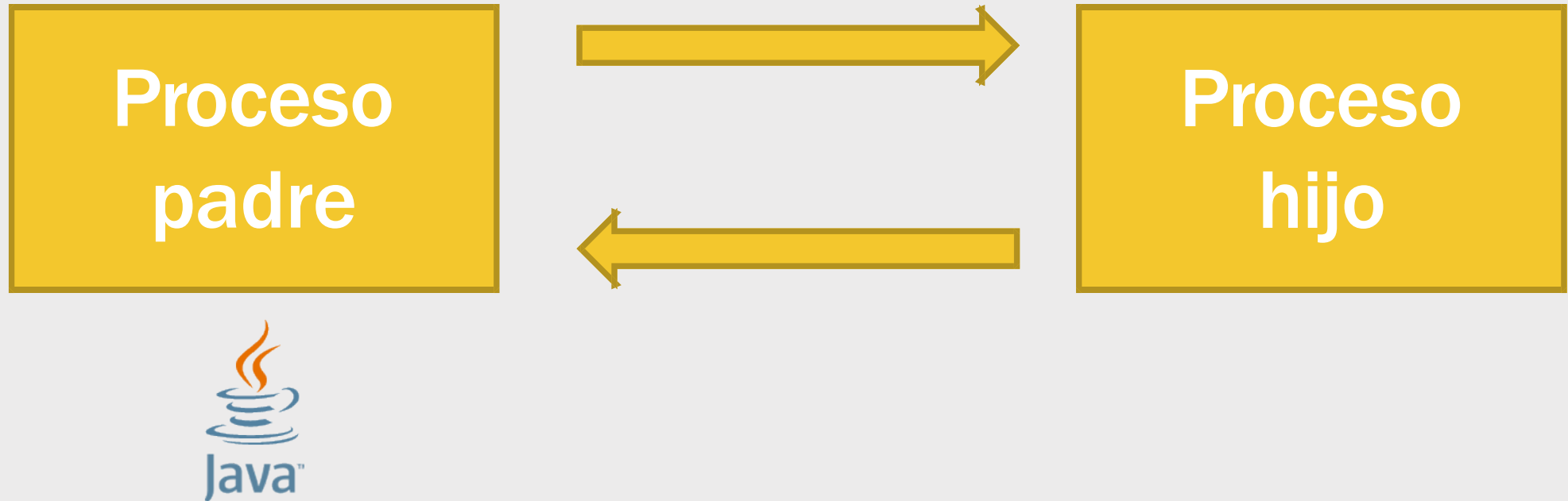
Clase: Process

■ Métodos interesantes (2/2):

- `isAlive()` : Comprobar si proceso está vivo [boolean].
- `waitFor()` : Esperar hasta que proceso termine.
- `destroy()` : Matar proceso.



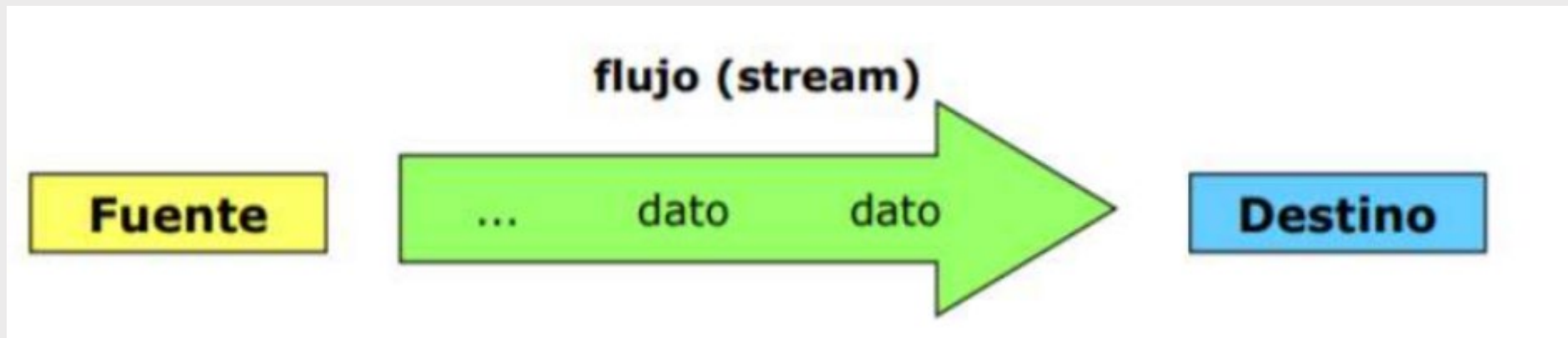
Árbol de procesos



- Todos los procesos son lanzados por otro proceso (padre).
- Padre e hijo NO comparten espacios de memoria. No se pueden comunicar directamente.

Comunicación entre procesos (1)

- En Java se define la abstracción de stream (flujo) para tratar la comunicación de información entre una fuente y un destino donde fluye una secuencia de datos.
- La fuente y el destino pueden ser dispositivos de distintas clases (ficheros, teclado, red, procesos, ...).



Comunicación entre procesos (2)

Es importante saber que todo proceso tiene una serie de flujos estándar:

- Una entrada estándar (**stdin**): habitualmente el teclado.
- Una salida estándar (**stdout**): habitualmente la consola.
- Una salida de error estándar (**stderr**): habitualmente la consola.

Comunicación entre procesos (3)

En Java estos flujos están implementados por los objetos:

- **System.in:** es una instancia de InputStream que representa un flujo de bytes de entrada.
- **System.out:** que es una instancia de PrintStream (subclase de OutputStream) que representa un flujo de bytes de salida y permite formatearla. Los métodos más importantes son print() y println() o flush() que vacía el buffer de salida haciendo efectivo su envío.
- **System.err:** funcionamiento similar a System.out y se utiliza para enviar mensajes de error.

Estos flujos estándar los cierra la propia JVM.



Comunicación entre procesos (4)

La comunicación se puede ver como un tubo (pipe) por donde va el flujo de información que se intercambian los dos dispositivos que están conectados a sus extremos.

Métodos de la clase Process:

- **getOutputStream():** se obtiene el flujo de salida (OutputStream) conectado a la entrada estándar del subprocesso. Este flujo servirá normalmente para que el proceso padre le mande información al proceso hijo.
- **getInputStream():** se obtiene el flujo de entrada (InputStream) conectado a la salida estándar del subprocesso. Este flujo servirá normalmente para que el proceso padre obtenga información del proceso hijo.
- **getErrorStream():** devuelve el flujo de salida conectado a la entrada normal del subprocesso. En Java hay que saber que stderr y stdout están conectados ambos al mismo dispositivo (consola).

Comunicación de procesos (5)

- Proceso padre recibe salida proceso hijo:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class E04_ComunicarProcesos {

    public static void main(String[] args) {
        String linea;
        Runtime rt = Runtime.getRuntime();
        String nombreProceso = "cmd.exe /c dir";
        try {
            Process miProceso = rt.exec(nombreProceso); // Crear proceso
            // Recoger resultado proceso
            BufferedReader br = new BufferedReader(
                new InputStreamReader (miProceso.getInputStream()));
            // Mostrar resultado proceso
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Comunicación entre procesos (6)

1 - Ver Ejemplos/ComunicacionProcesos

2 - Prácticas 1 y 2 para hacer en clase.



CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

4. COMANDOS GESTIÓN PROCESOS

Comandos: aspectos básicos (1/2)

- Formato: nombreComando opciones
 - *nombreComando* > *relacionado con la tarea (en inglés).*
 - *Ejemplos (muestra listado procesos):*
 - Windows > tasklist
 - Linux > ps (process status)



Comandos: aspectos básicos (2/2)

- Comandos tienen diversas opciones > uso del manual.
 - Windows > nombreComando /?
 - Linux > man nombreComando



Gestión procesos (Windows)

- tasklist > muestra listado procesos.
 - taskkill > mata proceso.
-

- Ejemplo uso:
 - taskkill /pid numeroProceso
 - taskkill /pid 11472



Gestión procesos (Linux)

- `ps` > muestra listado procesos.
- `pstree` > muestra listado procesos (forma árbol).
- `kill` > mata proceso.
- `nice` > cambia prioridad proceso.



5. PROGRAMACIÓN CONCURRENTE



Concurrencia

- Coincidencia de varios sucesos al mismo tiempo.
- Situaciones:
 - *Procesos que necesitan comunicarse entre ellos.*
 - *Procesos que necesitan acceder al mismo recurso (fichero,...).*
 - Importante: controlar la forma para que no haya errores, resultados incorrectos o inesperados.

Procesos concurrentes



- Dos procesos son concurrentes, cuando la primera instrucción de un proceso se ejecuta después de la primera y antes de la última de otro proceso.
- La planificación alternando los instantes de ejecución en la gestión de los procesos, hace que estos se ejecuten de forma concurrente.

Multiproceso = concurrencia

Programación concurrente

- Proporciona mecanismos de comunicación y sincronización entre procesos que se ejecutan de forma simultánea.
 - Instrucciones (tipos):
 - Que se pueden ejecutar de forma simultánea.
 - Que deben ser sincronizadas.



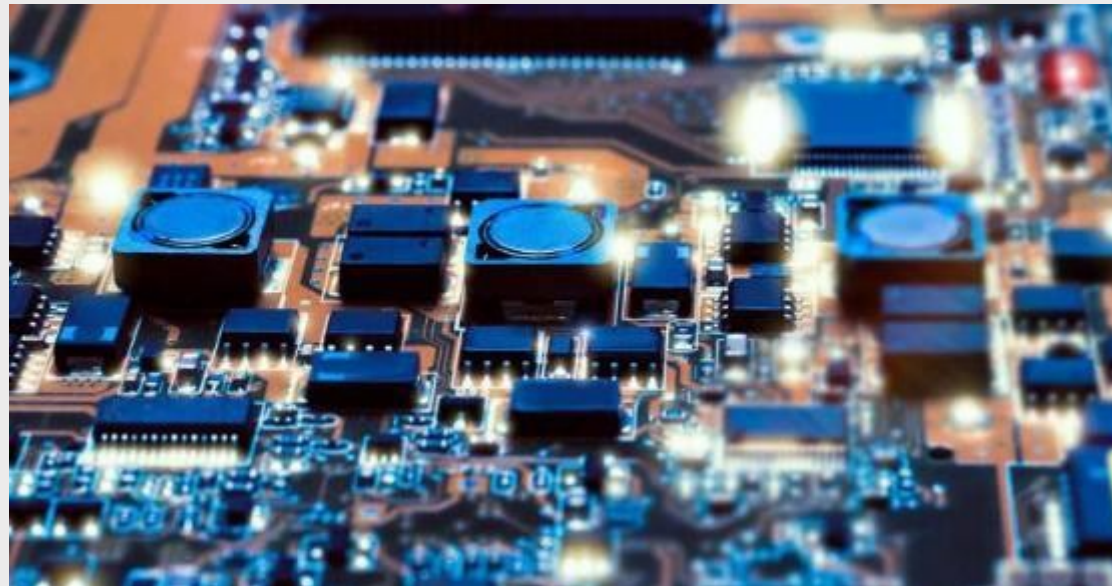
Razones uso

1. Optimizar la utilización de los recursos.
2. Proporcionar interactividad a los usuarios (y animación gráfica).
3. Mejorar la disponibilidad.
4. Conseguir un diseño conceptualmente más comprensible y mantenible.
5. Aumentar la protección.



Nuevos entornos hardware

1. Microprocesadores con múltiples núcleos.
2. Entornos multiprocesador.
3. Entornos distribuidos.



Beneficios

1. Aporta gran claridad.
2. Conduce a una reducción del tiempo de ejecución.
3. Permite una mayor flexibilidad de planificación.
4. Permite un mejor modelado del comportamiento del programa.



Tipos básicos interacción



- **Procesos independientes:** solo interfieren en el uso de la CPU.
- **Procesos cooperantes:** un proceso genera la información o proporciona un servicio que otro necesita.
- **Procesos competidores:** procesos que necesitan usar los mismos recursos de forma exclusiva.



Sincronización

Procesos competidores: conceptos (1/2)

- **Región exclusión mutua / región crítica:** conjunto de instrucciones en las que el proceso utiliza un recurso y que se deben ejecutar de forma exclusiva con respecto a otros procesos competidores por ese mismo recurso.





Procesos competidores: conceptos (2/2)

- Lock (bloqueo) de un recurso: cuando un proceso ha obtenido su uso en exclusión mutua.

-
- Deadlock (interbloqueo): cuando los procesos no pueden obtener los recursos que necesitan.
 - Ej.: 2 procesos que necesitan 2 recursos.



Situación muy peligrosa > puede llevar al sistema a su caída o cuelgue.

Requisitos

- Seguridad.
- Vivacidad.
- Eficiencia.
- Reusabilidad.



Objetivo > desarrollar software de calidad.

Requisitos (1/4): seguridad

- En cada instante de la ejecución no debe haberse producido algo que haga entrar al programa en un estado erróneo.
 - Ejemplos:
 - Dos procesos no deben entrar simultáneamente en una sección crítica.
 - Se respetan las condiciones de sincronismo.



Requisitos (2/4): vivacidad

- Cada sentencia que se ejecuta conduce en algún modo a un avance constructivo para alcanzar el objetivo funcional del programa.
 - Ejemplos (a evitar):
 - Aplazamiento indefinido.
 - Interbloqueo (deadlock).



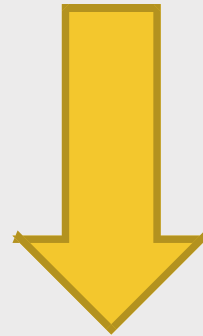
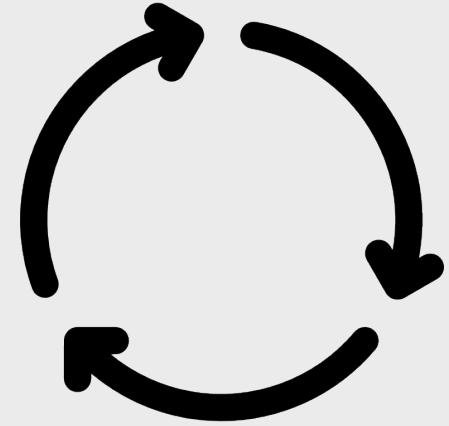
Requisitos (3/4): eficiencia

- No utilizar más recursos de los necesarios.
- Buscar la rigurosidad en la implementación: toda la funcionalidad esperada de forma correcta y concreta.



Requisitos (4/4): reusabilidad

- Implementar el código de forma modular.
- Documentar correctamente el código.



Patrones de diseño: soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en otras ocasiones.



CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

6. COMUNICACIÓN ENTRE PROCESOS

Comunicación entre procesos

- Un proceso da / deja información y/o recibe / recoge información.
- Los lenguajes de programación y los SO nos proporcionan primitivas de sincronización que facilitan la interacción entre procesos de forma sencilla y eficiente.





Mecanismos básicos

1. Intercambio de mensajes: tendremos las primitivas enviar (send) y recibir (receive o wait).
2. Recursos (o memoria) compartidos: tendremos las primitivas escribir (write) y leer (read) datos en o de un recurso.

Formas:

1. Buffer de memoria: misma máquina.
2. Socket: distintas máquinas.

Tipos de comunicación (1/2)



Tipos de comunicación (2/2)

- **Síncrona:** emisor queda bloqueado hasta que el receptor recibe el mensaje.
- **Asíncrona:** emisor continúa con su ejecución inmediatamente después de emitir el mensaje, sin quedar bloqueado.



7. SINCRONIZACIÓN ENTRE PROCESOS

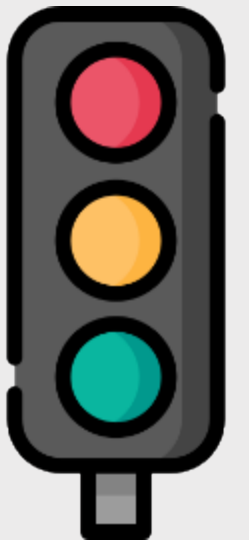
Sincronismo

- Sistema operativo: lo hace posible.
- Lenguajes programación alto nivel: encapsulan los mecanismos que proporciona cada SO en:
 - *Objetos.*
 - *Métodos.*
 - *Funciones.*



Soluciones (1/3): semáforos

- **Semáforo:** componente de bajo nivel de abstracción que permite arbitrar los accesos a un recurso compartido.
- **Ventaja:** funcionamiento sencillo.
- **Desventaja:** implementación.
- **Java (hilos):**
 - `java.util.concurrent.Semaphore`



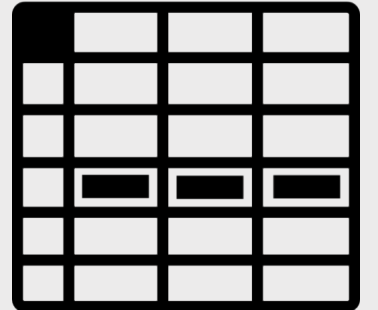
Soluciones (2/3): monitores

- **Monitor:** componente de alto nivel de abstracción que permite arbitrar los accesos a un recurso compartido.
- **Funcionamiento:** encierran en su interior los recursos o variables compartidas como componentes privadas y garantizan el acceso a ellas en exclusión mutua.
- **Java:** no existen paquetes.



Soluciones (3/3): memoria compartida

- SO actuales: implementan mecanismos que permiten proteger la zona de memoria de cada proceso.



- Solución: tener varios flujos de ejecución dentro de un mismo proceso (dividir un problema grande en varios pequeños).

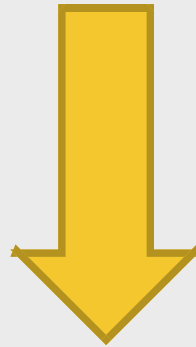


CENTRO INTEGRADO de FORMACIÓN PROFESIONAL
AVILÉS

8. PROGRAMACIÓN PARALELA Y DISTRIBUIDA

Programación paralela y distribuida

- Dos procesos se ejecutan de forma paralela, si las instrucciones de ambos se están ejecutando realmente de forma simultánea.

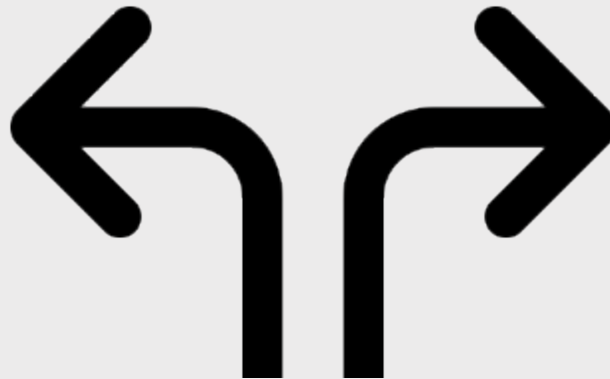


(Objetivo: ejecución simultánea de tareas que resuelven un problema común)



Diferencia

- Programación paralela: microprocesadores multinúcleo.
- Programación distribuida: conjuntos de ordenadores heterogéneos interconectados entre sí.



Características



- Se reparte el problema en tareas (porciones).
 - Programación paralela: intercambio datos a través de direcciones de memoria compartidas.
 - Programación distribuida: intercambio datos y sincronización mediante intercambio de mensajes.

El sistema se presenta como una unidad ocultando la realidad de las partes que lo forman.