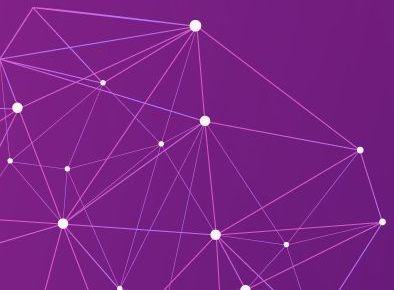
TEMA 1
CONCEPTOS BÁSICOS

01
CONCEPTOS BÁSICOS

NOTA IMPORTANTE: los procesos son entidades independientes, aunque estén ejecutando el mismo programa

CONCEPTOS BÁSICOS

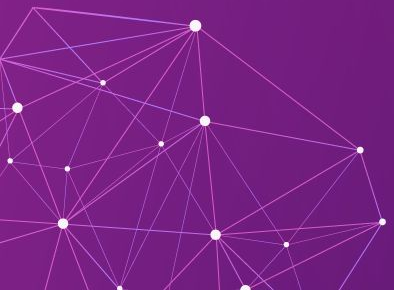
- **¿Qué es un programa?**
  - Toda la información (código y datos) almacenada en un ordenador y que resuelve una necesidad concreta de los usuarios
- **¿Qué es un proceso?**
  - De manera muy simple: **un programa ejecutándose**
  - No son sólo el código y los datos:
    - Identificador del programa
    - Permisos asignados
    - Contador de programa: por dónde se está ejecutando
    - Imagen de memoria: espacio de memoria asignado
    - Estado del procesador: valor de los registros del proceso



CONCEPTOS BÁSICOS

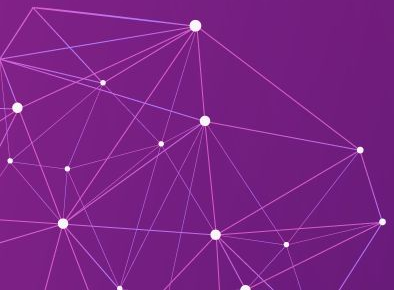
- **¿Qué es un ejecutable?**
  - Archivo que contiene la información necesaria para crear un proceso a partir de los datos almacenados de un programa.

- **¿Qué es un Demonio (daemon)?**
  - Proceso **no interactivo**.
  - Está ejecutándose en **segundo plano** continuamente.
  - Suele proporcionar un **servicio básico** para el resto de procesos



CONCEPTOS BÁSICOS

- **¿Qué es un Sistema Operativo?**
  - Programa intermediario entre el usuario y los programas y el hardware
  - Sus **objetivos**:
    - Ejecutar programas de usuario: **Crear procesos** a partir de ejecutables y gestionar su ejecución
    - Hacer el ordenador cómodo de utilizar: intermediario entre usuario y recursos, **interfaces**. Permite acceso a ficheros, memoria y hardware
    - Utilizar recursos de forma eficiente: **repartir los recursos** entre usuarios y programas para que todos sean servidos correctamente



SISTEMA OPERATIVO

02

FUNCIONAMIENTO DEL SISTEMA OPERATIVO

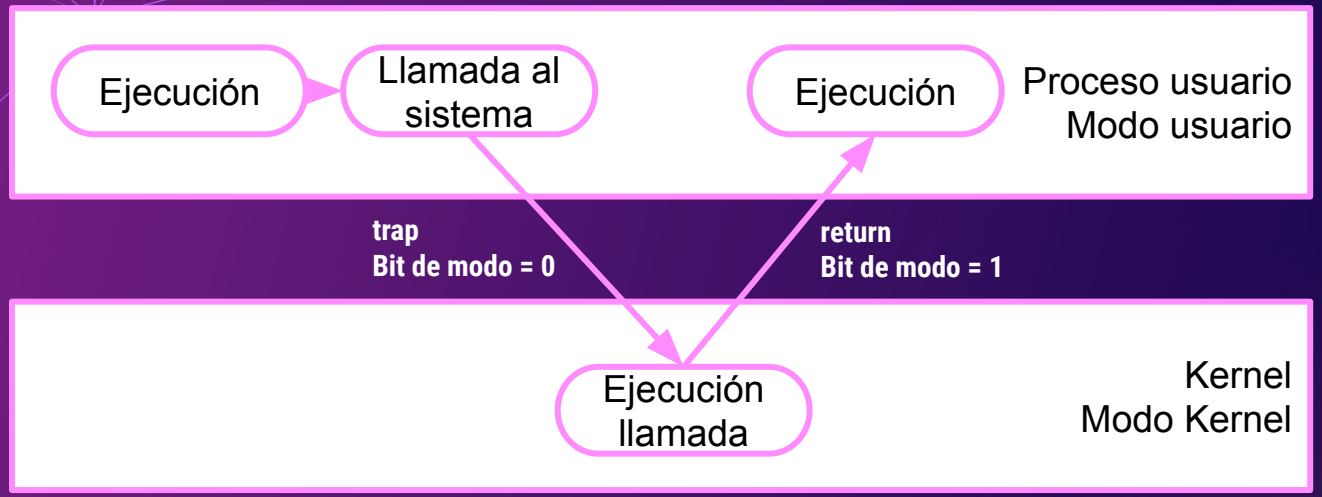
EL KERNEL

- Parte central del S.O. que realiza la funcionalidad más básica.
- Es el responsable de **gestionar los recursos** del ordenador.
- Funciona a base de **interrupciones**
  - Interrupción: suspensión temporal de la ejecución de un progreso para pasar a ejecutar una rutina del S.O.
  - Mientras se **atiende** una interrupción no pueden llegar otras interrupciones a no ser que sea de **nivel superior**
  - Cuando finaliza se reanuda el proceso justo donde se quedó.
- El S.O. **no es un proceso demonio**, si no que sólo se ejecuta en respuesta a interrupciones.
- Cuando salta una interrupción se transfiere el control a la rutina que trata dicha interrupción.
- Todas las rutinas de tratamiento de interrupciones componen el Kernel.

LLAMADAS AL SISTEMA

- Son la interfaz que proporciona el Kernel para que los programas de usuario puedan hacer uso de determinadas partes del S.O.
- Errores de un programa podrían afectar a otros programas.
- El sistema implementa el sistema de **llamadas** para evitar que instrucciones peligrosas se ejecuten directamente por el programa de usuario
- **Toda llamada** al sistema en un programa de usuario provoca una **interrupción**
- El procesador tiene dos modos de ejecución:
  - **Modo usuario** (1): para ejecutar programas de usuario
  - **Modo kernel** (0): sólo en este modo se pueden ejecutar instrucciones delicadas

LLAMADAS AL SISTEMA



03

LOS PROCESOS

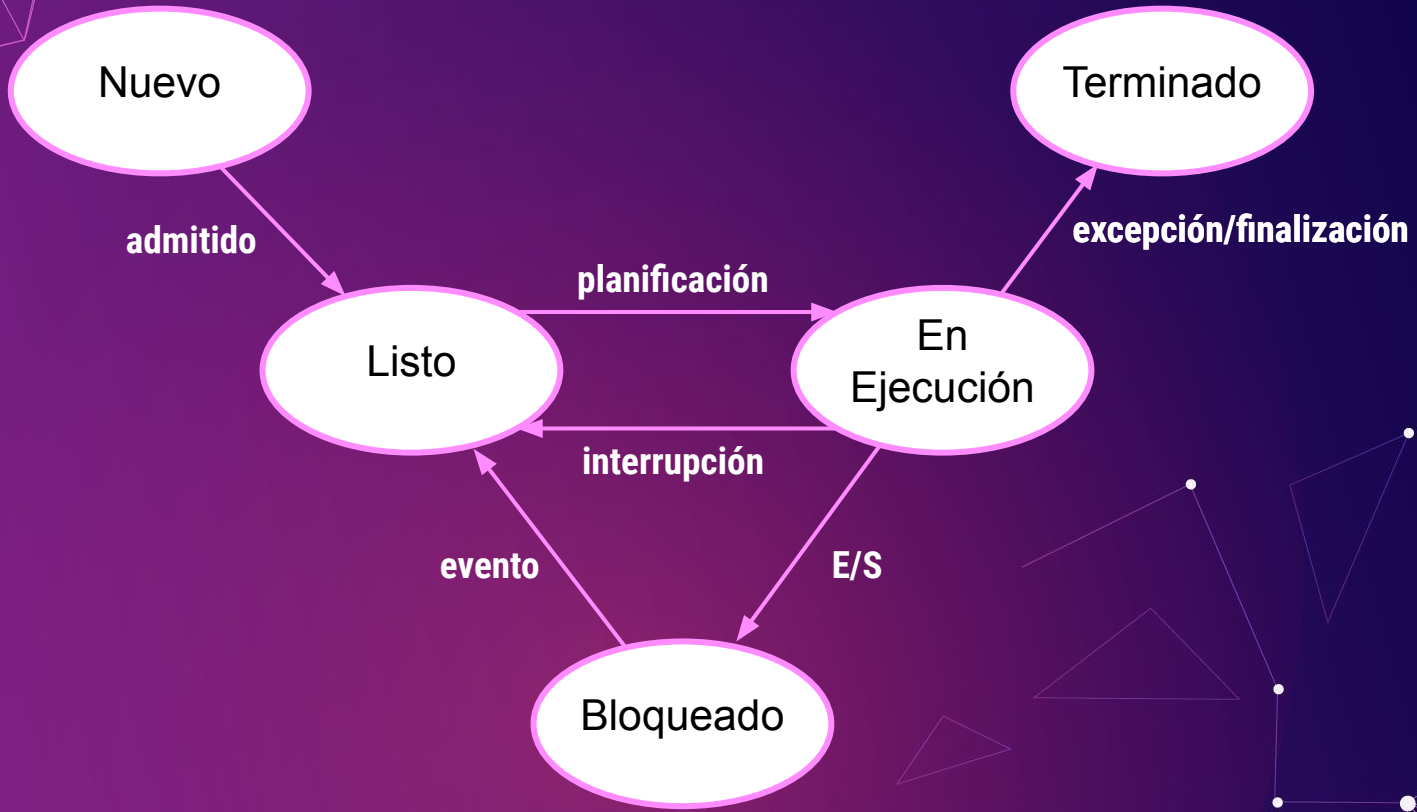
LOS PROCESOS

- El S.O. es el encargado de **ejecutar y gestionar** los procesos.
- A lo largo de su vida un proceso cambiará de estado a medida que se ejecuta.
- Es el S.O. el encargado de cambiar el estado de un proceso
- Los **estados** de un proceso son:
  - **Nuevo**
  - **Listo**
  - **En ejecución**
  - **Bloqueado**
  - **Terminado**

ESTADOS DE UN PROCESO

- **Nuevo**: el proceso está siendo creado a partir del ejecutable
- **Listo**: el proceso no está en ejecución, pero está preparado para hacerlo. El S.O. no le ha asignado todavía un procesador para ejecutarse
- **En ejecución**: el proceso está siendo ejecutado. Si necesitara un recurso cualquiera (incluso E/S) llamará a la llamada del sistema correspondiente
- **Bloqueado:** el proceso está esperando a que ocurra algún suceso (E/S, sincronización, etc). Cuando ocurre el evento lo desbloquea y pasará a estar listo
- **Terminado**: el proceso ha terminado de ejecutarse y libera los recursos

ESTADOS DE UN PROCESO



COLAS DE PROCESOS

- Principal objetivo de la multiprogramación es admitir varios procesos en memoria.
- Los procesos se irán intercambiando el uso del procesador para su correcta ejecución.
- El S.O. gestiona varias colas de procesos:
  - **Cola de procesos**: todos los procesos del sistema.
  - **Cola de procesos preparados**: los procesos listos esperando para ejecutarse.
  - **Colas de dispositivos**: los procesos que están a la espera de una E/S

PLANIFICACIÓN DE PROCESOS

- Para gestionar las colas de procesos es necesario un planificador de procesos.
- Es el encargado de seleccionar qué proceso va a qué cola.
- Existen dos tipos de planificadores:
- **A corto plazo:**
  - Elije qué proceso de la cola de preparados irá a ejecutarse.
  - Se ejecuta muy frecuentemente => debe ser rápido.
- **A largo plazo:**
  - Elije qué proceso nuevo debe pasar a la cola de preparados.
  - Se ejecuta pocas veces
  - Controla el grado de multiprogramación.

CAMBIO DE CONTEXTO

- Cuando un proceso sale del procesador, el S.O. debe guardar el contexto de dicho proceso y restaurar el contexto del proceso que el planificador a corto plazo haya elegido.
- El contexto de un proceso es:
  - El estado del proceso
  - El estado del procesador: valores de los registros.
  - Información de gestión de memoria: espacio de memoria que había reservado para el proceso.
- El cambio de contexto se considera tiempo perdido ya que el procesador no hace trabajo útil.

04

GESTIÓN DE PROCESOS

ÁRBOL DE PROCESOS

- Cuando un usuario quiere abrir un programa el S.O. crea y pone en ejecución el proceso correspondiente.
- Aunque quien crea un proceso es el S.O. lo hace siempre **a petición de otro proceso**
- Un proceso en ejecución siempre **depende del proceso que lo creó**, estableciéndose un vínculo entre ambos.
- Cuando se arranca el ordenador y se carga el Kernel, se crea el proceso **inicial del sistema**. A partir de este proceso, se crea el resto de procesos de forma **jerárquica** (padres, hijos, abuelos, etc)

ÁRBOL DE PROCESOS

- Para identificar a los procesos el S.O. asigna un identificador de proceso único para cada proceso. (PID)
- La utilización del PID es fundamental para gestionar procesos, porque es la forma que tiene el SO de referirse a los procesos

Ubuntu Online

1. Abrir una terminal
2. Ejecutar `ps -ef`
3. Ejecutar `pstree -Sp`
4. Ejecutar `kill -9 1` ¿que ocurre?

OPERACIONES BÁSICAS CON PROCESOS

- Cuando se crea un nuevo proceso hay que tener en cuenta que el proceso **hijo** y el proceso **padre** se van a ejecutar **concurrentemente**
- El proceso padre crea un hijo mediante la operación **CREATE**
- Compartirán CPU teniendo en cuenta la política de planificación del SO
- No comparten memoria, son procesos **independientes**. Si necesitan intercambiar información, deberán compartir recursos para hacerlo.
  - Lo más común es mediantes Ficheros
  - Colas FIFO

OPERACIONES BÁSICAS CON PROCESOS

- Por lo general, el hijo ejecuta un programa diferente que el padre
- Si el proceso padre necesita esperar hasta que el hijo termine su ejecución para poder continuar, puede hacerlo mediante la operación **WAIT**
- El proceso hijo depende tanto del SO como del padre, así que el padre puede detenerlo cuando quiera. Lo hará mediante la operación **DESTROY**
- Normalmente si un padre muere lo hacen todos sus hijos en cascada
- Esto no pasa con la JVM en la que pueden sobrevivir los hijos y ejecutarse de forma asíncrona

OPERACIONES BÁSICAS CON PROCESOS

OPERACIÓN	DESCRIPCIÓN
CREATE	Creación de nuevos procesos
WAIT	El padre espera hasta que el hijo termina su ejecución
DESTROY	El padre termina la ejecución del hijo cuando quiera

EJERCICIO

- ¿Qué es la programación concurrente?
  - ¿Es lo mismo que la paralela?
- ¿Qué diferencia la programación paralela de la multiprogramación?

05

PROGRAMACIÓN CONCURRENTE

PROGRAMACIÓN CONCURRENTE

- Permite la posibilidad de tener en ejecución al “mismo tiempo” **múltiples tareas** interactivas, es decir realizar varias cosas al mismo tiempo.
- Estas tareas se pueden ejecutar en:
  - Un único procesador: **multiprogramación**
  - Varios procesadores: **paralelismo+multiprogramación**

“Un sistema concurrente es aquel donde un cálculo puede avanzar sin esperar a que el resto de los cálculos se complete”

MULTIPROGRAMACIÓN

Se denomina 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.

- El **sistema operativo** se encarga de ir **cambiado el proceso** en ejecución después de un período corto de tiempo
- **No se mejora el tiempo global** de ejecución
- **Aprovecha los tiempos** que los procesos pasan esperando a que se completen sus operaciones de E/S y por ende aumenta la eficiencia en el uso del CPU

PROGRAMACIÓN PARALELA

- **Memoria compartida**
  - **Cada núcleo** puede ejecutar una **instrucción diferente**.
  - El sistema operativo va **intercambiando los procesos igualmente**.
  - Todos los núcleos comparte la misma memoria => pueden **interactuar entre sí**
  - Pueden ejecutarse diferentes instrucciones del mismo proceso a la vez
  - **Se mejora el rendimiento**

PROGRAMACIÓN PARALELA

- **Varios ordenadores en red**
  - Cada ordenador tiene su propio procesador y Sistema Operativo
  - Esto es **programación distribuida**
  - Se alcanzan elevadas mejoras en el rendimiento.
  - **No comparten memoria** => hay que utilizar otros métodos de comunicación
    - paso de mensajes. Ej: MPI
      - Con funciones send y receive

06

CREACIÓN DE PROCESOS CON JAVA

CREACIÓN DE PROCESOS (CREATE)

- En Java a través de las clases **Process** y **ProcessBuilder** podemos lanzar un proceso de cualquier comando del sistema y además acceder a su entrada estándar, su salida estándar y su salida de error.
- La clase **Process** representa el proceso que lanzamos.
- Hay dos formas de obtener una referencia a esta clase:
  - Con la clase **ProcessBuilder.start()**: inicia un nuevo proceso utilizando los atributos indicados en el objeto.
    - **Command()** => comando que se quiere ejecutar
    - **Directory()** => directorio de trabajo
    - **Environment()** => variables de entorno
  - **Runtime.exec(String[] com, String[] envp, File dir)**

CREACIÓN DE PROCESOS (CREATE)

- Ambos métodos comprueban que el **comando es válido** en el sistema operativo.
- Pero la **creación de un proceso depende al final del S.O.** que haya bajo la JVM, así que pueden ocurrir **problemas** que habrá que controlar:
  - No encontrar el ejecutable indicado en la ruta
  - No tener permisos de ejecución
  - No ser un ejecutable válido en el sistema.
  - Otros…
- En la mayoría de los casos se lanzará una **IOException** que controlaremos.

CREACIÓN DE PROCESOS (CREATE)

```
import java.io.IOException;
import static java.lang.Thread.sleep;
import java.util.Arrays;
public class RunProcess {
public static void main(String[] args) throws IOException {

    ProcessBuilder pb = new ProcessBuilder("calc.exe");
    try {
        Process process = pb.start();
        int retorno = process.waitFor();
        System.out.println("La ejecución de " +
        Arrays.toString(args) + " devuelve " + retorno);
    }catch(IOException ex){
        System.err.println("Excepción de E/S!!");
        System.exit(-1);
    }catch(InterruptedException ex){
        System.err.println("El proceso hijo finalizó de forma incorrecta");
        System.exit(-1);
    }
    }
}
```

TERMINACIÓN DE PROCESOS (DESTROY)

- Un proceso padre puede decidir terminar un proceso hijo sin esperar a que se termine de ejecutar.
- La operación destroy elimina el hijo y libera todos sus recursos del S.O.

```
import java.io.IOException;
public class RuntimeProcess {
    public static void main(String[] args) {
        if (args.length <= 0)
        {
            System.err.println("Se necesita un programa a ejecutar");
            System.exit(-1);
        }
        Runtime runtime = Runtime.getRuntime();
        try
        {
            Process process = runtime.exec(args);
            process.destroy();
        }
        catch(IOException ex)
        {
            System.err.println("Excepción de E/S!!");
            System.exit(-1);
        }
    }
}
```

07

COMUNICACIÓN ENTRE PROCESOS

COMUNICACIÓN ENTRE PROCESOS

- Los procesos (como programas que son) reciben información y producen resultados. Para ello se utilizan:
  - Entrada estándar (**stdin**): normalmente el teclado, pero podría ser fichero, tarjeta de red o hasta otro proceso.
  - Salida estándar (**stdout**): normalmente la pantalla, pero podría ser fichero, impresora o hasta otro proceso.
  - Salida de error (**stderr**): normalmente igual que la salida estándar, pero podría utilizarse otro lugar.
- En JAVA se utilizarán los procedimientos **System.out y System.err** para mandar información a las salidas estándar y de error.

COMUNICACIÓN ENTRE PROCESOS

- En JAVA, el proceso hijo creado de la clase Process no tiene interfaz de comunicación propia.
- El usuario no podrá comunicarse directamente con él.
- Todas sus salidas (stdin, stdout y stderr) se redirigen al proceso padre utilizando flujos de datos.
  - **OutputStream**: flujo de salida del proceso hijo. Este flujo envía la información directamente a la entrada de datos del padre (stdin).
  - **InputStream**: flujo de entrada del proceso hijo. Este flujo envía la información directamente a la salida de datos del padre (stdout)
  - **ErrorStream**: flujo de error del proceso hijo. Este flujo envía la información directamente a la salida de error del hijo (stderr) que suele ser la misma que la de salida.

COMUNICACIÓN ENTRE PROCESOS

- Si se quisiera separar stdout y stderr se puede utilizar **redirectErrorStream(bool)** de la clase ProcessBuilder
  - Con true los flujos stderr y stdout serán diferentes.
- Utilizando los flujos que hemos visto, el proceso padre podrá mandarle información a los procesos hijos y podrá recibir los resultados de la salida que genere (incluidos los errores)
- Hay que tener en cuenta que en algunos S.O. los buffer de entrada y salida tienen un tamaño limitado.

FIN

Tema 1