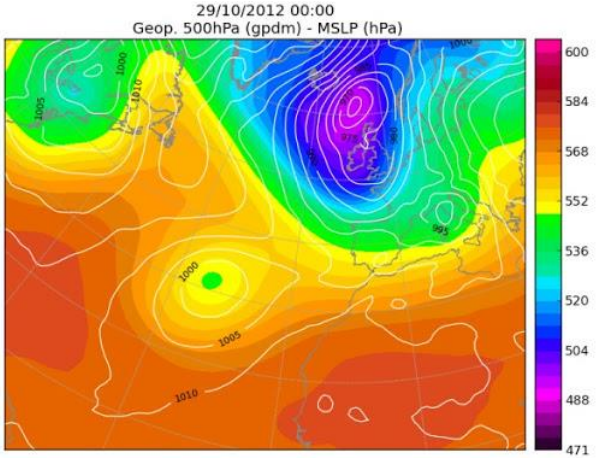
1. PROGRAMACIÓN MULTIPROCESO

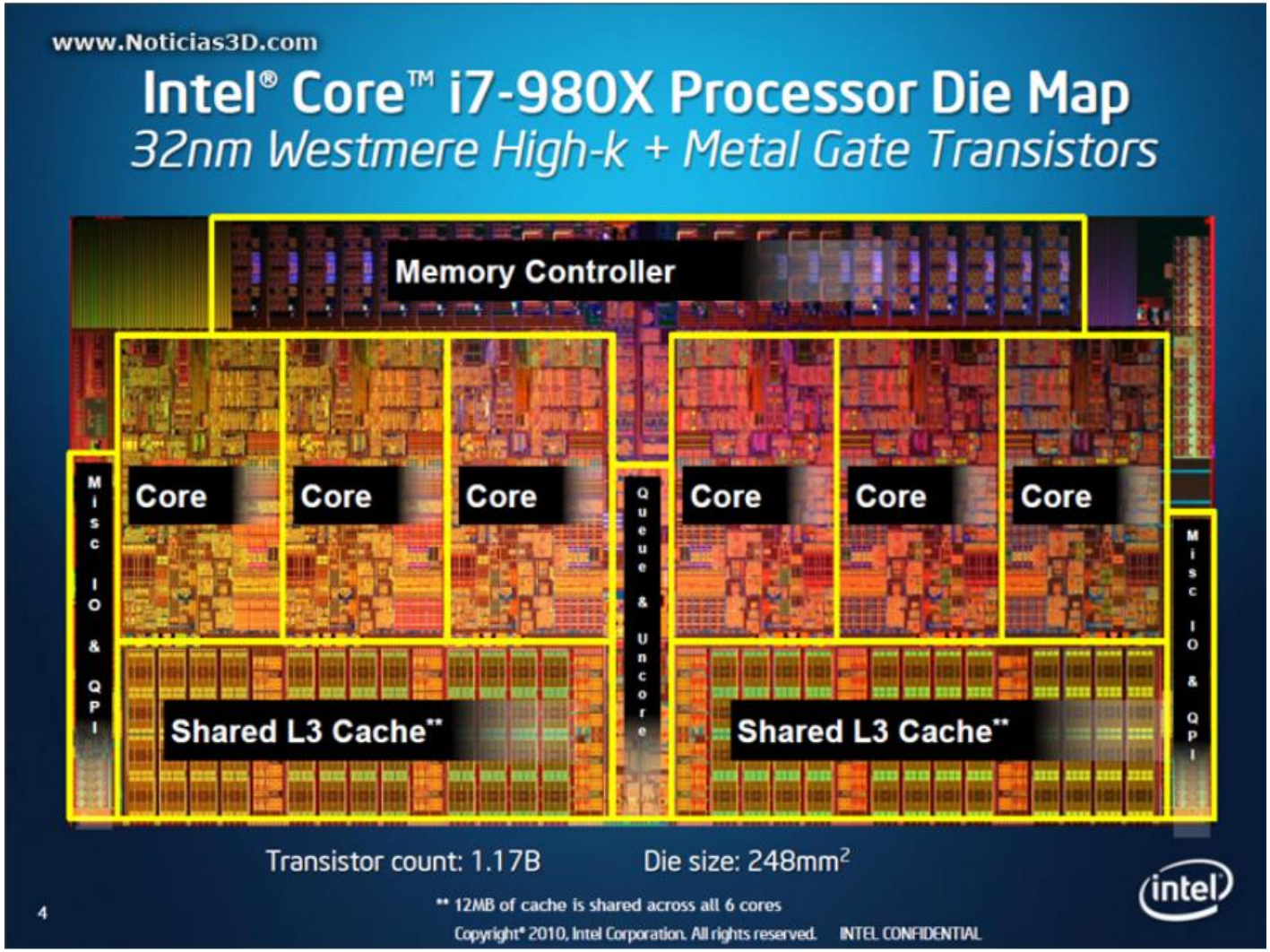
Joaquín Franco

1. Introducción

- Cada día se pide más y más potencia de cálculo



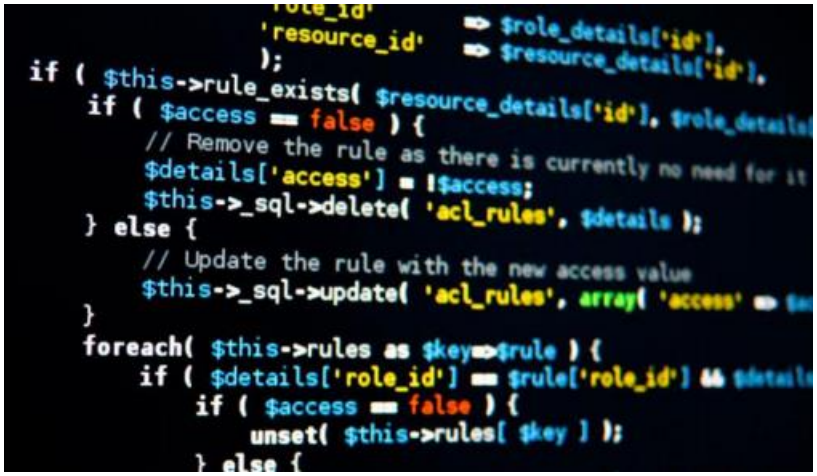
1.Introducción

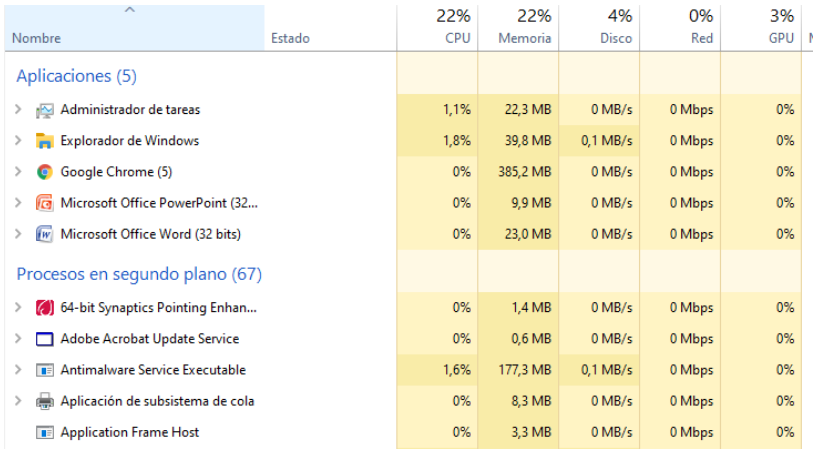


2. Ejecutables, procesos y servicios

- Un **programa** es un conjunto de instrucciones, unas líneas de código escritas en un lenguaje de programación.

- Un **proceso** es un programa en ejecución. Los procesos consumen recursos del ordenador: memoria, cpu, disco, red, etc. El sistema operativo es el encargado de gestionarlos (crearlos, destruirlos, planificarlos, controlar los recursos que utilizan, etc).





Nombre	Estado	22% CPU	22% Memoria	4% Disco	0% Red	3% GPU
Aplicaciones (5)						
Administrador de tareas		1,1%	22,3 MB	0 MB/s	0 Mbps	0%
Explorador de Windows		1,8%	39,8 MB	0,1 MB/s	0 Mbps	0%
Google Chrome (5)		0%	385,2 MB	0 MB/s	0 Mbps	0%
Microsoft Office PowerPoint (32...		0%	9,9 MB	0 MB/s	0 Mbps	0%
Microsoft Office Word (32 bits)		0%	23,0 MB	0 MB/s	0 Mbps	0%
Procesos en segundo plano (67)						
64-bit Synaptics Pointing Enhan...		0%	1,4 MB	0 MB/s	0 Mbps	0%
Adobe Acrobat Update Service		0%	0,6 MB	0 MB/s	0 Mbps	0%
Antimalware Service Executable		1,6%	177,3 MB	0,1 MB/s	0 Mbps	0%
Aplicación de subsistema de cola		0%	8,3 MB	0 MB/s	0 Mbps	0%
Application Frame Host		0%	3,3 MB	0 MB/s	0 Mbps	0%

2. Ejecutables, procesos y servicios

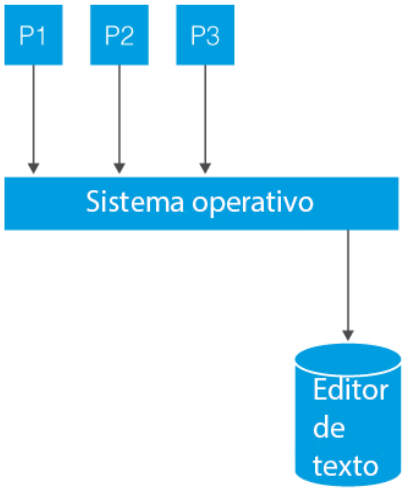
- **Servicio**: un tipo de proceso que no tiene interfaz con el usuario y que se ejecuta en segundo plano.
- **Ejecutable**: Un fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.

3. Programación concurrente

- Multiproceso: ejecutar muchos procesos al mismo tiempo.

3.1 Procesos concurrentes

- El sistema operativo es el encargado de la gestión de los **procesos**.
  - Los crea, los elimina y los provee de instrumentos que permitan la ejecución y también la comunicación entre ellos.
  - Cada proceso **tiene su propio espacio de memoria** y no se puede compartir con la de otro.

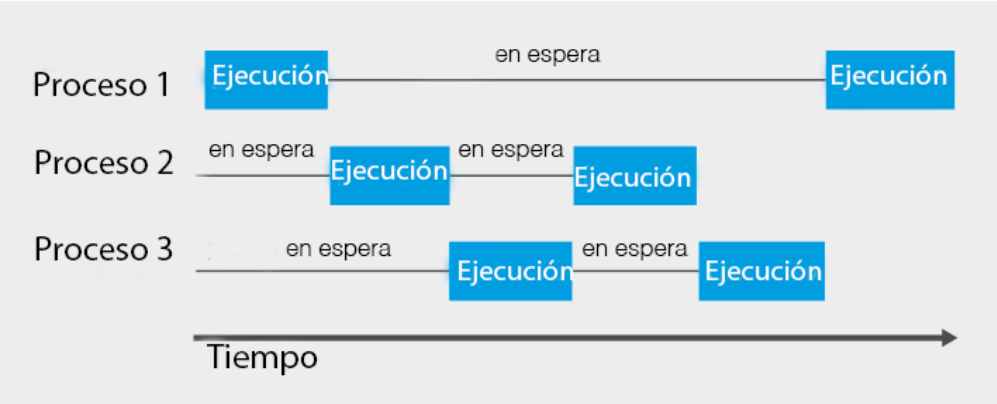


3.1 Procesos concurrentes

- La programación concurrente:
  - Da mecanismos para la comunicación y sincronización de procesos.
  - Permite definir qué instrucciones de nuestros procesos se pueden ejecutar de forma simultánea con los otros procesos sin que haya errores.

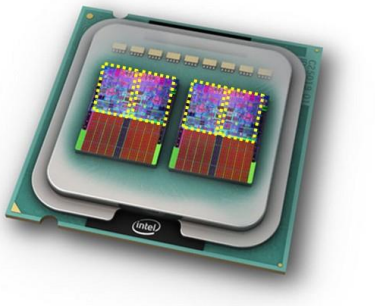
3.2 Procesos, sistemas monoprocesador y multiprocesador

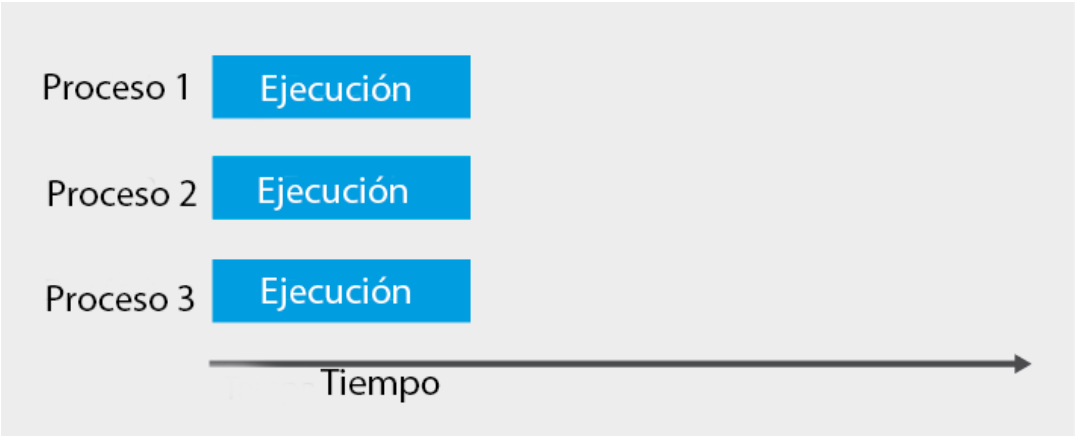
- En un **sistema informático monoprocesador** se va alternando la ejecución de los procesos muy rápido. Tan rápido que parece que se están ejecutando al mismo tiempo.
- A esto se llama **multiprogramación**.



3.2 Procesos, sistemas monoprocesador y multiprocesador

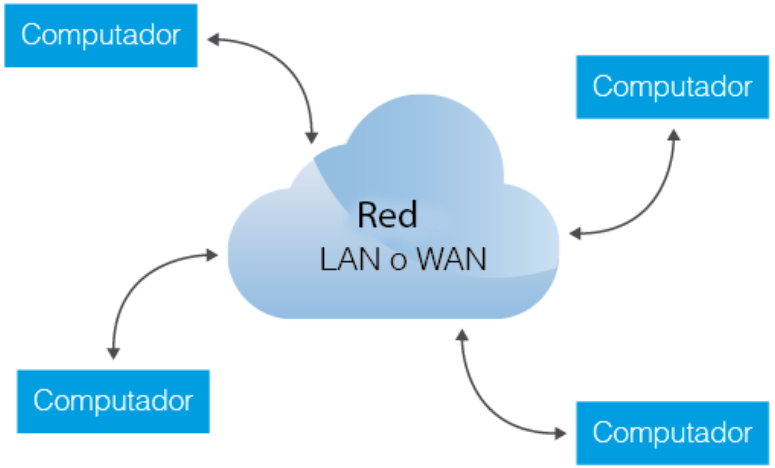
- En un **sistema informático multiprocesador** existen dos o más procesadores, por lo tanto, se pueden ejecutar simultáneamente varios procesos.
- A esto se llama **programación paralela o multiproceso**.





3.2 Procesos, sistemas monoprocesador y multiprocesador

- La **programación distribuida** es un tipo especial de programación paralela.
- Los procesos se ejecutan en distintos ordenadores conectados en red.



3.3 Ventajas de la programación concurrente

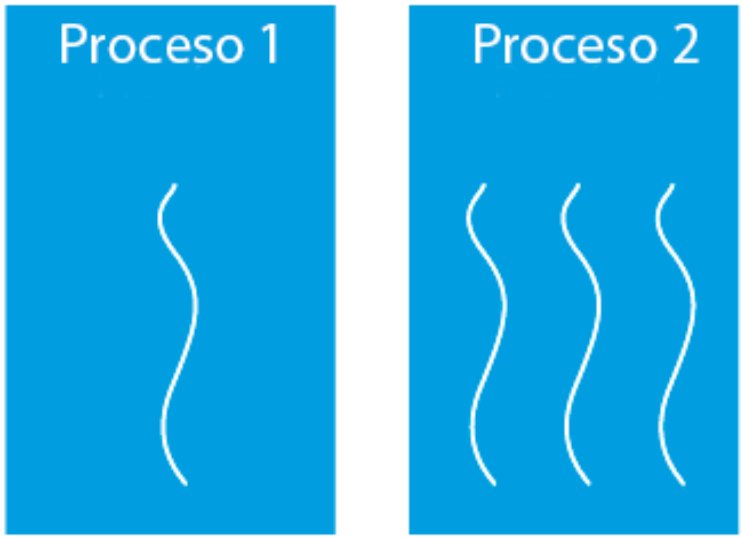
- Mejorar aprovechamiento de la CPU.
- Mayor velocidad de ejecución.
- Problemas de naturaleza concurrente:
  - Sistemas de control en tiempo real.
  - Tecnología web: servidores con múltiples peticiones simultáneas.
  - Aplicaciones basadas en GUI.
  - Simulaciones.
  - Sistemas Gestores de Bases de Datos.
- Etc.

3.4 Hilos

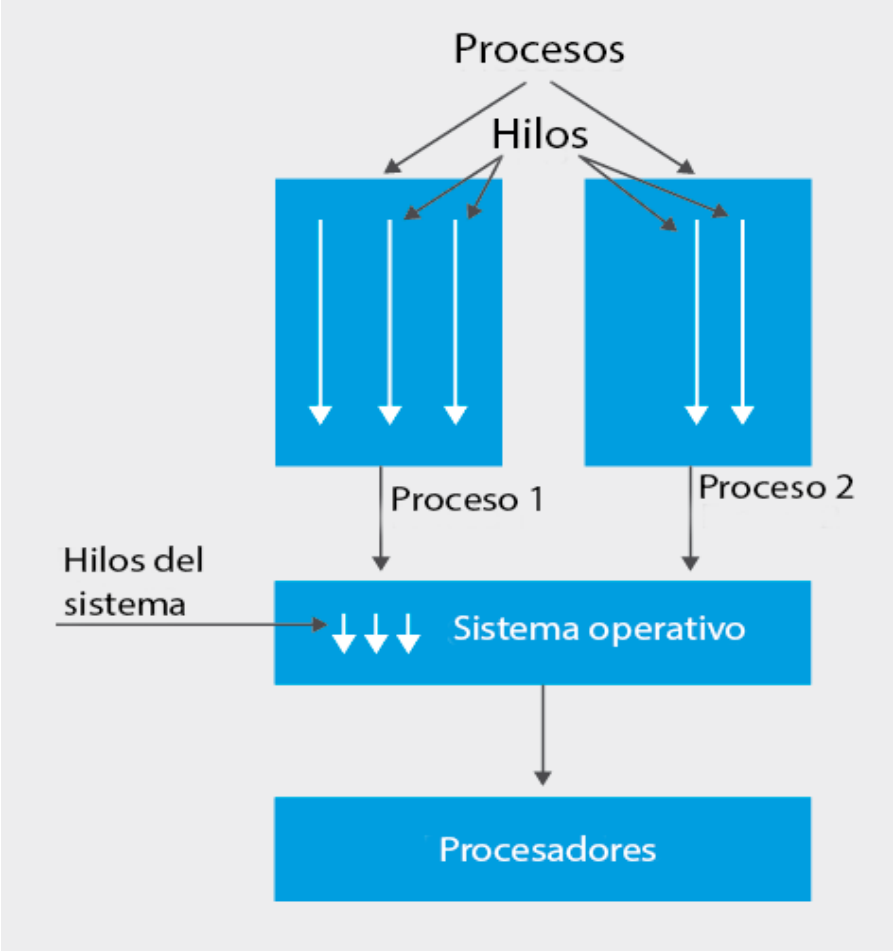
- Además de la concurrencia de procesos, dentro de cada proceso pueden ejecutarse varios **hilos**.
- Dentro del mismo proceso, instrucciones independientes pueden ejecutarse a la vez. Es decir, un proceso se puede dividir a su vez en subprocesos.
- Los hilos comparten los recursos del proceso (datos, código, **memoria**, etc).
- Procesos -> **Entidades pesadas**.
  - Creación y comunicación consume muchos recursos.
- Hilos->**Entidades ligeras**.
  - Creación y comunicación consumen pocos recursos.

3.4 Hilos

- Todo proceso tiene al menos un hilo de ejecución



3.4 Hilos


Procesos
Hilos
Proceso 1
Proceso 2
Hilos del sistema
Sistema operativo
Procesadores

4. Procesos

- Un proceso es **cualquier programa en ejecución.**
- Un proceso necesita ciertos **recursos** para realizar satisfactoriamente su tarea:
  - Tiempo de CPU.
  - Memoria.
  - Archivos.
  - Dispositivos de E/S.

4. Procesos

- Las obligaciones del SO como gestor de procesos son:
  - Creación y eliminación de procesos.
  - Planificación de procesos.
  - Establecimiento de mecanismos para la sincronización y comunicación de procesos.

4. Procesos

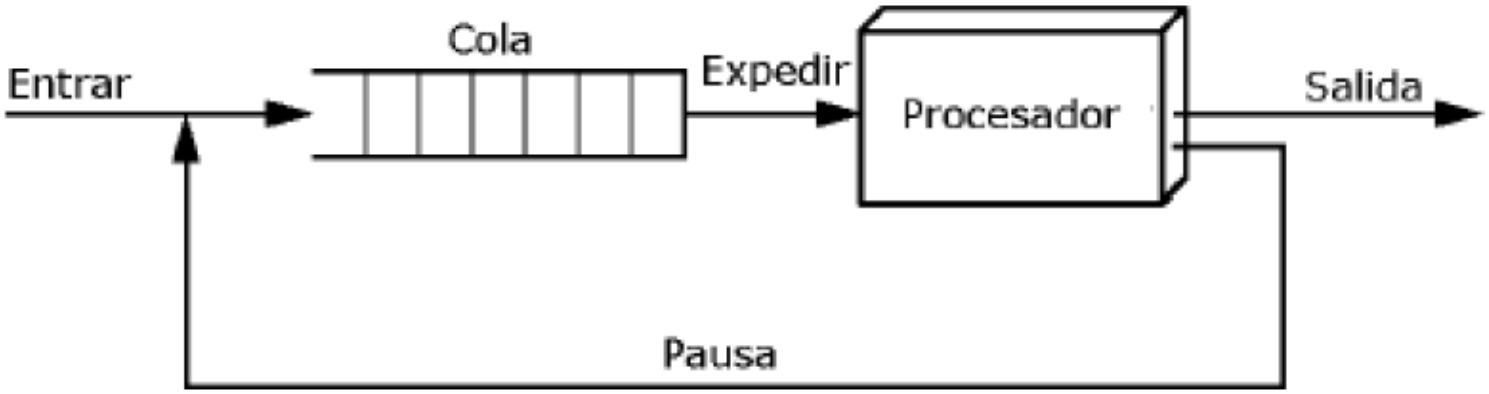
- Un proceso está formado por:
  - Sección de texto (código del programa).
  - Actividad actual representada por:
    - Valor del contador del programa.
    - Contenido de registros del procesador.
  - Además también incluye:
    - Pila, que contiene datos temporales (parámetros de subrutinas, direcciones de retorno y variables locales).
    - Sección de datos, que contiene variables globales y memoria utilizada por el programa.

4.1 Tipos de procesos

- **Por lotes**: Están formados por una serie de tareas, de las que el usuario sólo está interesado en el resultado final (imprimir documentos).
- **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 (procesador de textos).
- **Tiempo real**: Tareas en las que es crítico el tiempo de respuesta del sistema (programa que controla un brazo mecánico).

4.2 Estados de un proceso

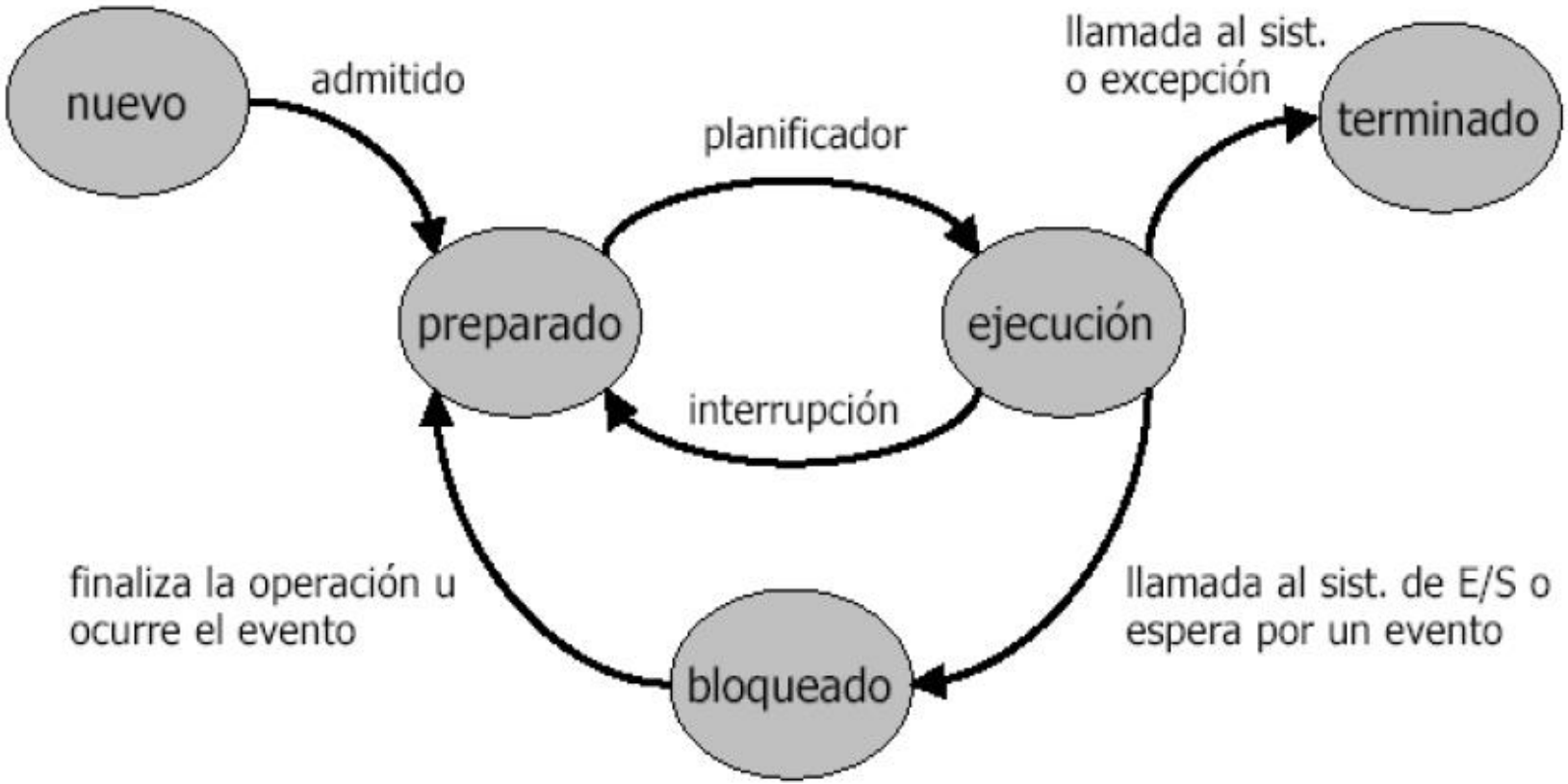
- Modelo de proceso con dos estados (Ejecución y no ejecución).



4.2 Estados de un proceso

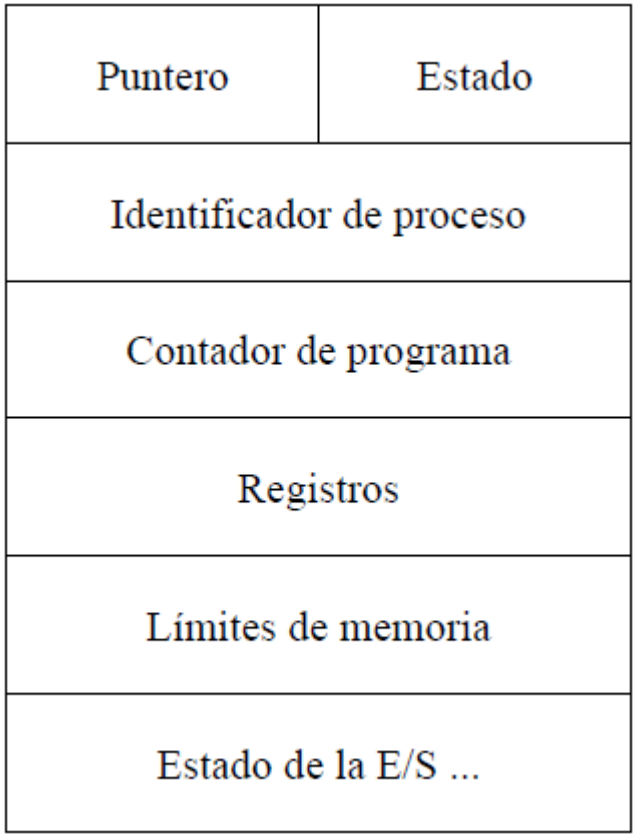
- A medida que un proceso se ejecuta cambia de estado. Cada proceso puede estar en uno de los estados:
  - **Nuevo** (new): el proceso se está creando.
  - En **Ejecución**: el proceso está en la CPU ejecutando instrucciones.
  - **Bloqueado**: el proceso está esperando a que ocurra un suceso (ej. terminación de E/S o recepción de una señal).
  - **Listo**: Esperando a que se le asigne un procesador.
  - **Terminado**: finalizó su ejecución, por tanto no ejecuta más instrucciones y el SO le retirará los recursos que consume.

2.3 Estados de un proceso



4.3 Bloque de control de proceso (PCB)

- Cada proceso se representa en el SO con un bloque de control de proceso(también llamado bloque de control de tarea).



Prueba: Administrador de procesos Windows

- El administrador de tareas en Microsoft Windows permite gestionar los procesos del sistema operativo. En él se puede ver qué procesos se están ejecutando con su uso de procesador y memoria.
- Abre el administrador de tareas y comprueba para tres procesos cuánta memoria y CPU están usando.
- Abre la calculadora de Windows y termina su proceso con el administrador.

4.4 Cambio de contexto de la CPU

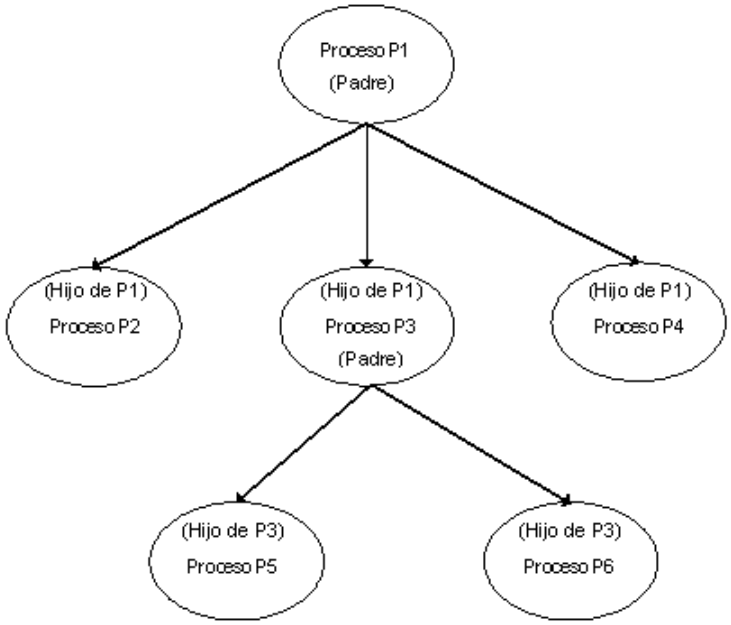
- Registros: se guarda la información que en cada instante necesita la instrucción que está ejecutando la CPU.
- Destacamos dos:
  - **Contador del programa**: dirección de la siguiente instrucción a ejecutar.
  - **Puntero a la pila**: apunta a un trozo de memoria donde se guarda el contexto de la CPU.

4.5 Planificación de procesos

- Planificador: se encarga de decidir qué proceso se ejecuta y durante cuanto tiempo.
- Algoritmos de planificación
  - **Round robin**: Cada proceso se ejecuta durante un quantum. Si no le ha dado tiempo a finalizar durante ese quantum se coloca al final de la cola de procesos listos y espera de nuevo su turno.
  - **Por prioridad**: Se asignan prioridades a los distintos procesos. Se ejecutan antes los que tienen mayor prioridad.
  - **Múltiples colas**: Combinación de los dos anteriores. Todos los procesos con una misma prioridad están en una misma cola y cada cola se gestiona con Round robin.

4.6 Árbol de procesos

- Cuando se arranca el ordenador y se carga el *kernel* del sistema en disco se crea el proceso inicial del sistema. A partir de este proceso se crea el resto de procesos de manera jerárquica, creando lo que se denomina el **árbol de procesos**.



4.7 Operaciones básicas con procesos

- **Create**: Creación de un nuevo proceso.
  - Cuando se crea un proceso, padre e hijo se ejecutan concurrentemente, ambos comparten la CPU y se irán intercambiando según la política de planificación.
- **Wait**: Espera de un proceso.
  - Si el proceso padre necesita esperar hasta que el hijo termine su ejecución para poder continuar la suya con los resultados obtenidos del hijo.
- **Exit**: Terminación de un proceso
  - Se liberan los recursos utilizados por el proceso.
  - Dependiendo del SO se produce “terminación en cascada”, es decir, si el padre termina tienen que terminar también todos sus hijos.
- **Destroy**: Terminación abrupta de un proceso hijo por el padre.

5. Programación concurrente orientada a objetos

- La creación de procesos en los sistemas operativos se realiza mediante las **llamadas al sistema**.
- Cada SO operativo gestiona sus procesos de manera diferente.
  - Windows: función createProcess() que crea un nuevo proceso hijo a partir de un programa distinto al que está en ejecución.
  - UNIX: función fork(), que crea un proceso hijo con un duplicado del espacio de direcciones del padre, es decir, un duplicado del programa que se ejecuta desde la misma posición.

5.1 Gestión de procesos en Java

- Cada sistema operativo tiene características únicas y la gestión de procesos es diferente.
- Java evita tener que depender del Sistema Operativo para la gestión de procesos.
- La JVM permite la ejecución de binarios de Java sobre cualquier sistema operativo.
- Write Once, Run Anywhere “escribe una vez y ejecuta en cualquier lugar”.
- Con Java los procesos padre e hijo no tienen por qué ejecutarse de manera concurrente y no se produce terminación en cascada.

5.2 Creación de procesos en Java

- Clase que representa a un proceso: Process.
- **Process Runtime.exec**(String[] cmdarray, String[] envp, File dir)
  - cmdarray: comando y argumentos.
  - envp: entorno
  - dir: directorio donde se ejecuta.
- **ProcessBuilder.start():** inicia un nuevo proceso utilizando los atributos indicados en el objeto.
  - command(): comando a ejecutar y argumentos.
  - directory(): directorio donde se ejecuta.
  - environment(): variables de entorno.

5.2 Creación de un proceso

String[] commandLine=…;

//Se crea un objeto de tipo Process

- Process process = Runtime.getRuntime().exec(commandLine,null);

5.2 Creación de un proceso

String[] commandLine=…;

//Se crea un objeto ProcessBuilder

- ProcessBuilder builder= new ProcessBuilder(commandLine);

//Se inicia un nuevo proceso que ejecuta el comando commandLine

Process process=builder.start();

5.3 Sincronización entre procesos en Java

- El padre puede esperar a que termine el hijo y obtener su información de finalización.
- La operación waitFor() de Java bloquea al padre hasta que el hijo finaliza mediante un exit.
- El valor de retorno de finalización es un número entero. Por convención se utiliza 0 para indicar que el hijo ha acabado de forma correcta.

5.3 Sincronización entre procesos en Java

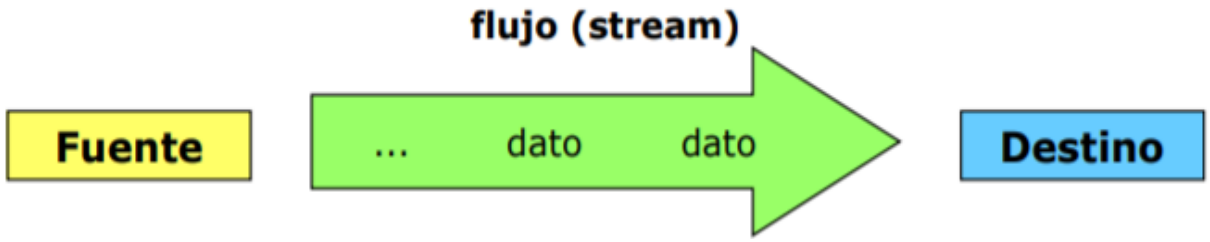
- Terminación normal
  - El proceso hijo realiza su ejecución completa y termina cuando ejecuta la operación exit.
- Terminación abrupta
  - Un proceso puede terminar de forma abrupta un proceso hijo que creó utilizando la operación destroy.
  - Destroy: elimina el proceso hijo liberando sus recursos en el sistema operativo subyacente.

5.4 Comunicación de procesos

- Un proceso da o deja información que otro proceso recibe o recoge.
- Interacciones entre procesos:
  1. **Sincronización**
  2. **Exclusión mutua**
  3. **Sincronización condicional**
- Mecanismos para intercambio de información
  1. **Sockets**: intercambio de información entre distintas máquinas (tema 3).
  2. **Buffer de memoria**: crear un canal de comunicación entre procesos utilizando la memoria principal del sistema.

5.5 Comunicación de procesos en Java

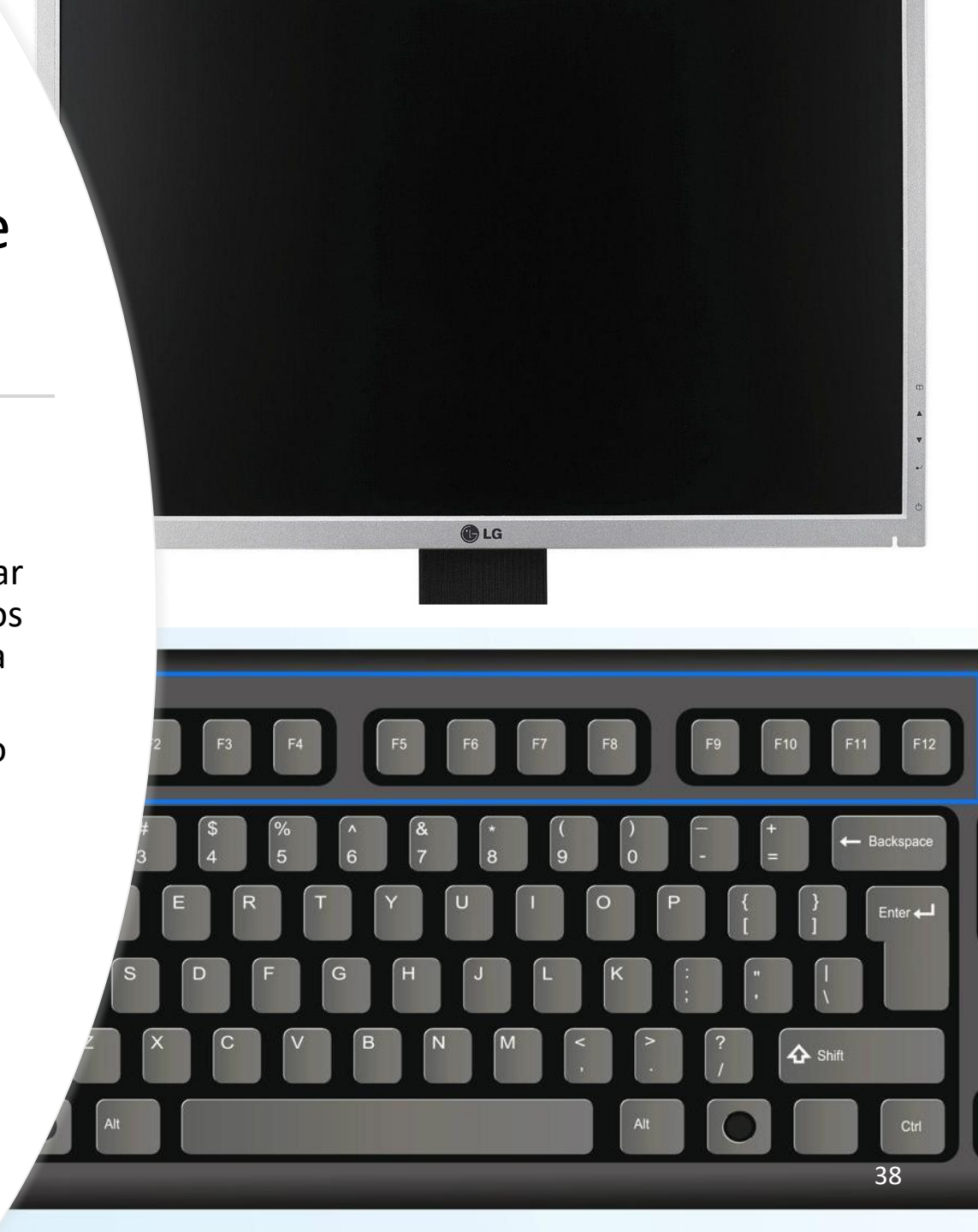
- En Java se utilizan los sockets y los buffers como cualquier otro stream.



- Las lecturas/escrituras **son bloqueantes**.

5.4 Comunicación de procesos en Java

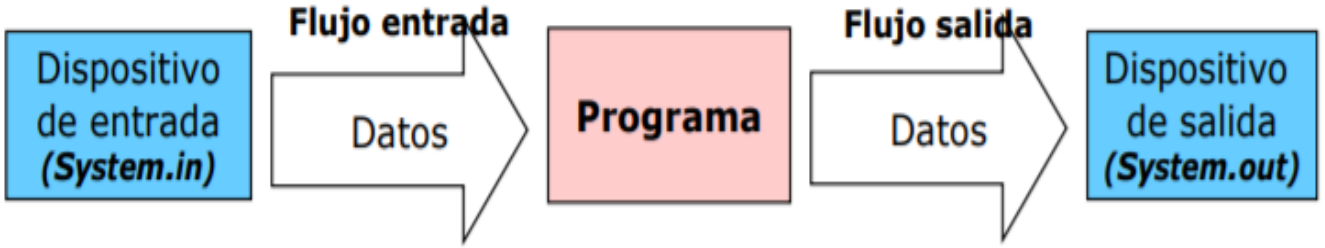
- Cada proceso en todo sistema operativo tiene:
  - Entrada estándar (**stdin**): lugar donde el proceso lee los datos de entrada que requiere para su ejecución.
  - Salida estándar (**stdout**): sitio donde el proceso escribe los resultados que obtiene.
  - Salida de error (**stderr**): sitio donde el proceso envía los mensajes de error

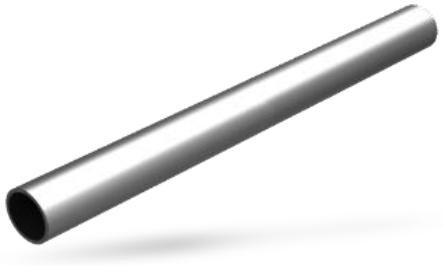


5.4 Comunicación de procesos en Java

- En Java:
  - **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.
  - ***System.err****:* funcionamiento similar a **System.out** y se utiliza para enviar mensajes de error.

5.4 Comunicación de procesos en Java





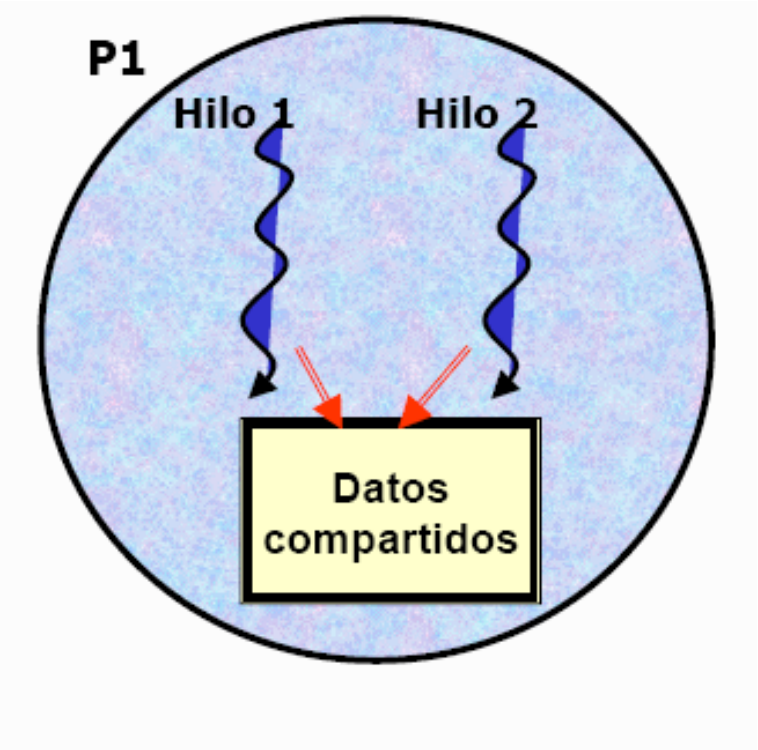
5.4 Comunicación de procesos en Java

- Para comunicar un proceso padre con un proceso hijo la clase Process tiene los siguientes métodos:
  - **getOutputStream():**se obtiene el flujo de salida (OutputStream) conectado a la entrada estándar del subproceso.
  - **getInputStream():** se obtiene el flujo de entrada (InputStream) conectado a la salida estándar del subproceso.
  - **getErrorStream()**: devuelve el flujo de salida conectado a la entrada normal del subproceso

  **Utilizar redirectErrorStream(boolean) de la clase ProcessBuilder para separarlos.

6. Condiciones de competencia y regiones críticas

- **Condición de competencia**: cuando dos procesos o más necesitan el mismo recurso.
- **Región crítica o de exclusión mutua:** conjunto de instrucciones de un proceso que deben ejecutarse de manera exclusiva porque el proceso utiliza un recurso que no debe usar ningún otro proceso hasta que termine.



6. Condiciones de competencia y regiones críticas

- Herramientas para mejorar la concurrencia y hacer la sincronización entre hilos:
  1. Región crítica
  2. Semáforos
  3. Región crítica condicional
  4. Buzones
  5. Monitores
- Veremos a lo largo del curso cómo utilizar estas herramientas desde el lenguaje Java.