

# Arquitectura de Sistemas Operativos

---

Clase de repaso – Clases 1 a 5

Tecnicatura Universitaria en  
Programación

# Agenda Repaso 1er Parcial



Clase 1: Introducción a los Sistemas Operativos



Clase 2: Procesos



Clase 3: Procesos e Hilos



Clase 4: Concurrencia, exclusión mutua y sincronización



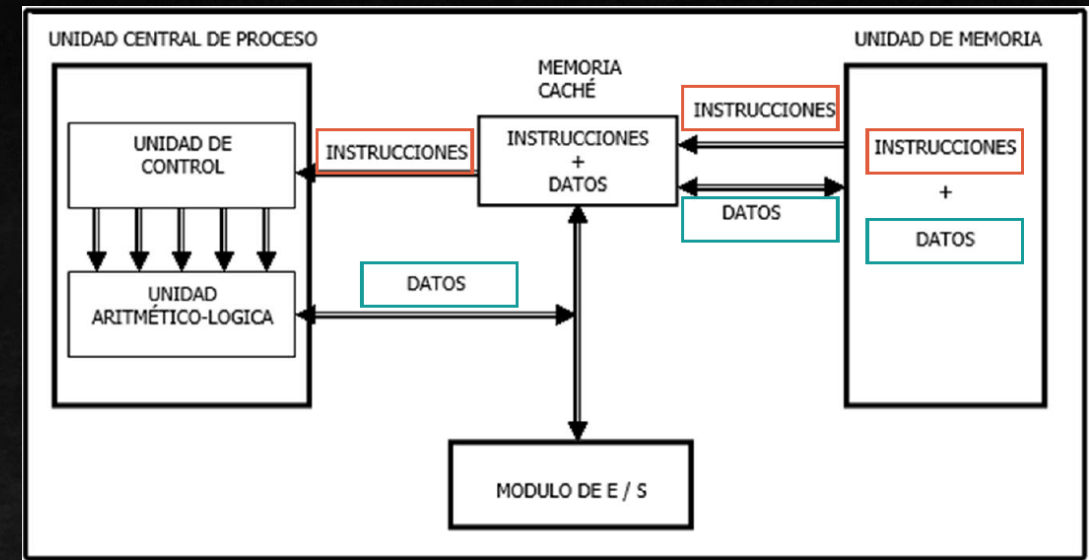
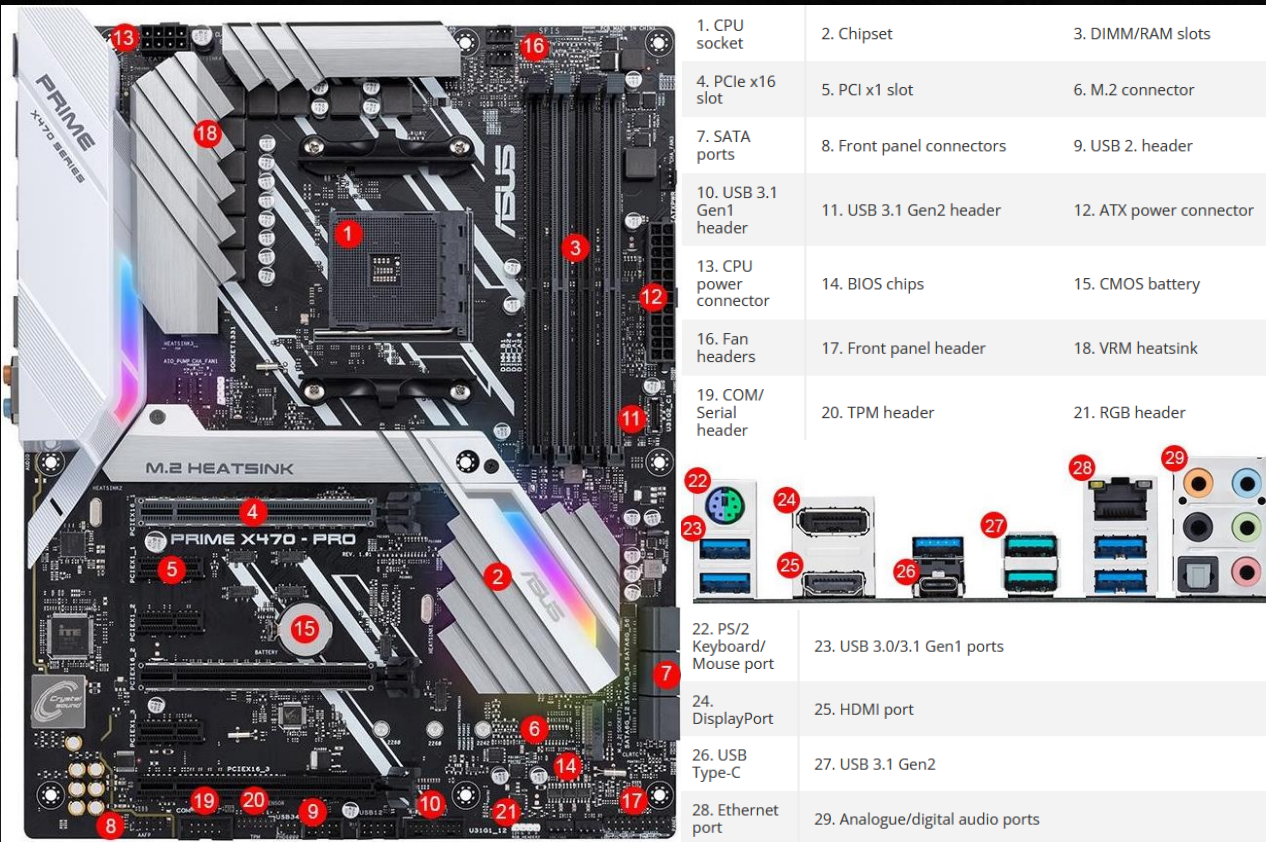
Clase 5: Fundamentos de interbloqueo

## Clase 1

- Objetivos y funciones
- Historia y evolución en el tiempo
- Conceptos básicos
- Los 5 pilares de un Sistema Operativo

# Introducción a los Sistemas Operativos

# Motherboard / Computadora



## Módulos de Entrada/Salida (E/S):

- ✓ Permiten la comunicación con el exterior (discos, teclado, pantalla, red, etc.).
- ✓ Transfieren datos entre la computadora y los dispositivos periféricos.

## Bus del Sistema:

- ✓ Es el canal de interconexión que comunica CPU, memoria y E/S.
- ✓ Transporta direcciones, datos y señales de control.

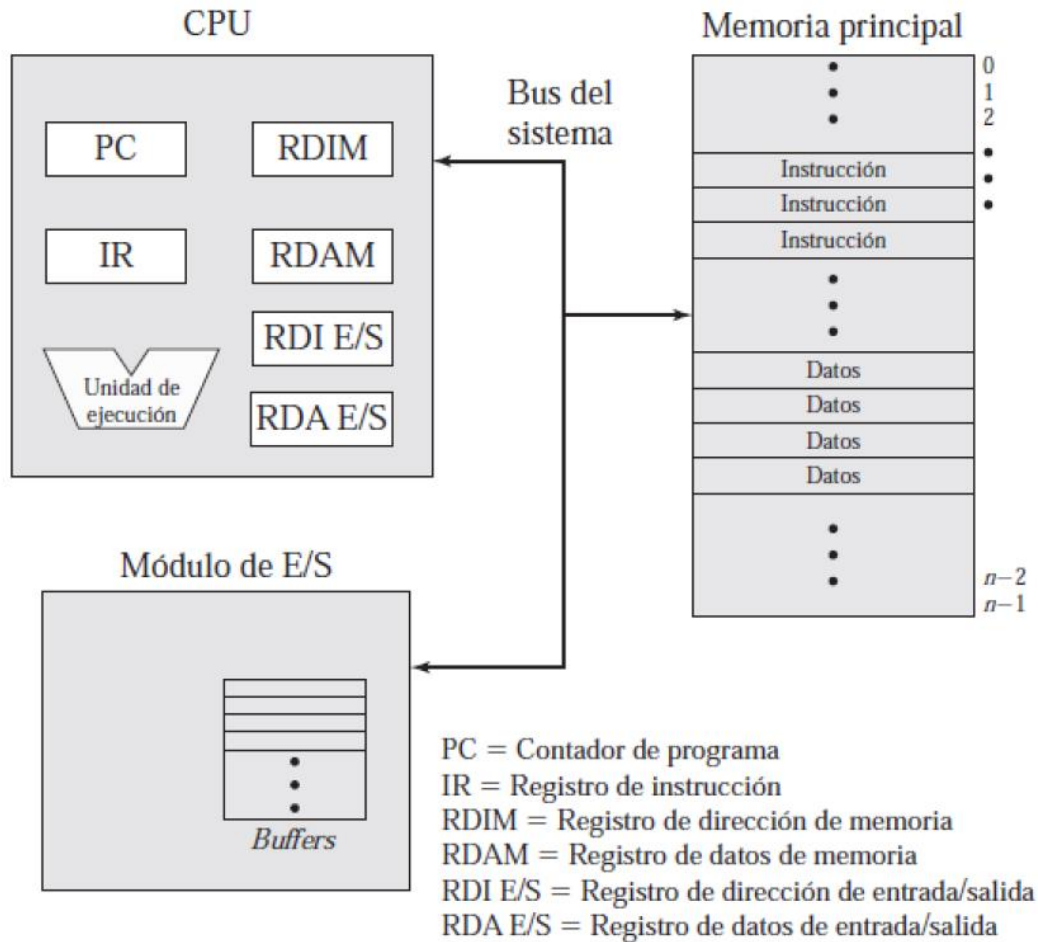
## Procesador

- ✓ Es el "cerebro" de la computadora.
- ✓ Controla la ejecución de instrucciones y realiza las operaciones de datos.
- ✓ Incluye registros para direccionamiento, datos y control.

## Memoria principal:

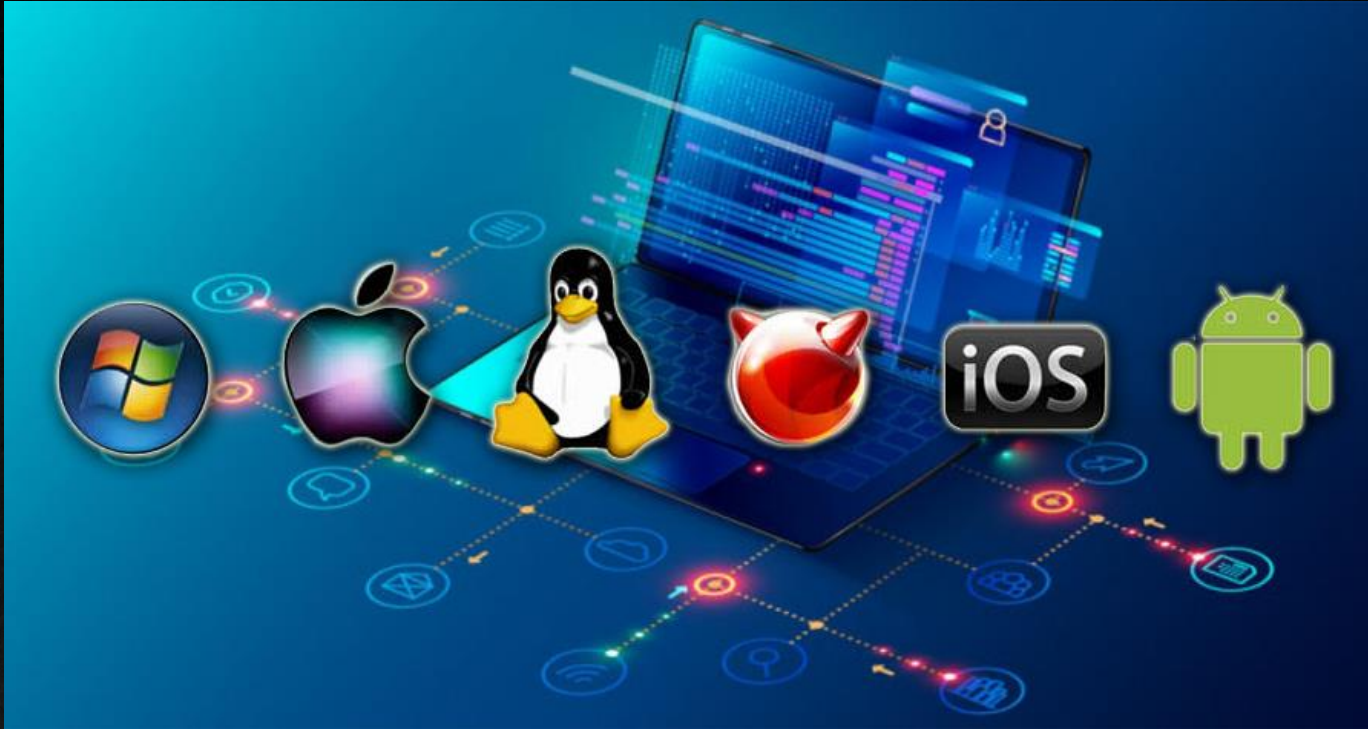
- ✓ Almacena temporalmente programas e información que la CPU necesita en ejecución.
- ✓ Es volátil: pierde el contenido al apagar el sistema.

# ¿Que es un programa?



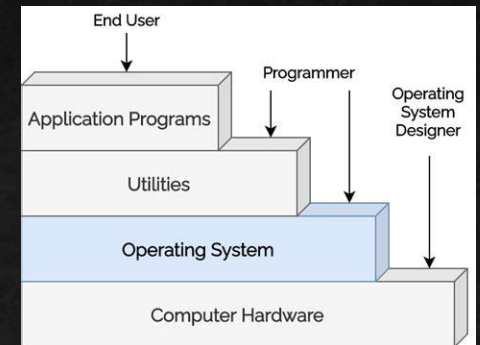
- Un **programa** es un conjunto de **instrucciones**.
- Escritas en un **lenguaje de programación**.
- La **computadora** puede **interpretar y ejecutarlo**.
- Para realizar una tarea específica o resolver un problema.

# ¿Que es un sistema operativo?



- **Objetivos:**
  - ✓ Facilidad de uso
  - ✓ Gestionar recursos
  - ✓ Capacidad de evolucionar
- **IMPORTANTE:**
  - ✓ Debe poder adaptarse a cambios y mejoras tecnológicas sin dejar de cumplir su función principal

- Un **sistema operativo** es un **programa** que controla la ejecución de aplicaciones y actúa como interfaz entre las aplicaciones y el hardware de la computadora.
- Es el software fundamental que **gestiona recursos** como procesador, memoria, discos y dispositivos de entrada/salida.
- Su misión es facilitar el uso del **hardware** y brindar servicios a programas y usuarios.
- Ejemplos: Windows, Linux, macOS, iOS, Android



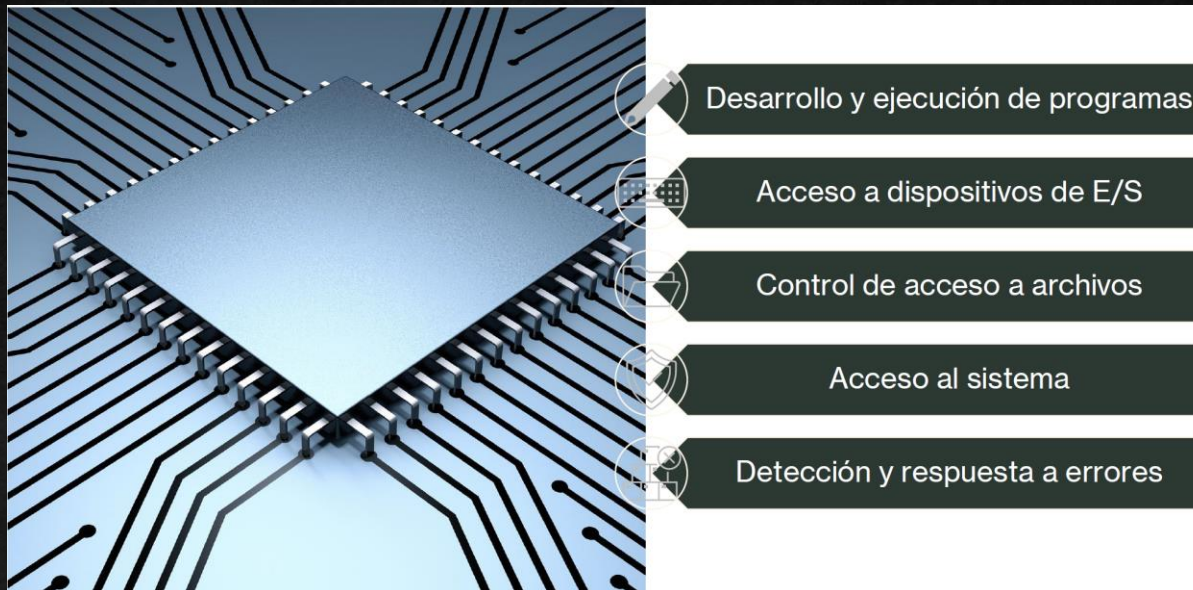
- El **hardware** y **software** utilizados para proporcionar aplicaciones a los usuarios se puede pensar como un sistema organizado **jerárquicamente en capas**, donde la capa inferior proporciona servicios a la capa superior.

# El sistema operativo

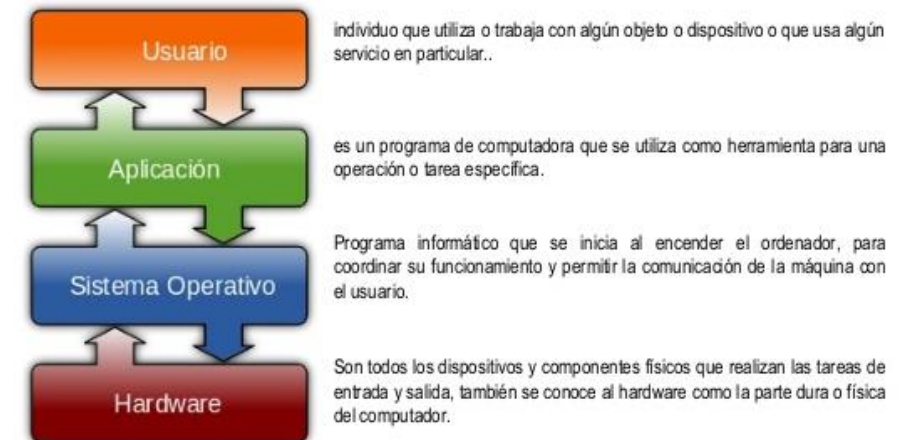
## ■ El rol clave del Sistema Operativo

Oculto los detalles del **hardware** a los **programadores** proporcionando una interfaz adecuada para su uso (bibliotecas), mientras que a las **aplicaciones** le ofrecen **servicios** y **utilidades**.

## ■ Servicios que provee



### Interacción entre el SO con el resto de las partes

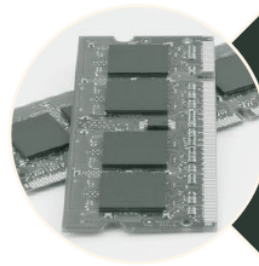
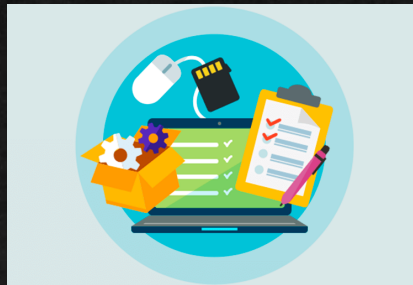


# El SO : Gestión de los Recursos

## ■ Gestión de recursos

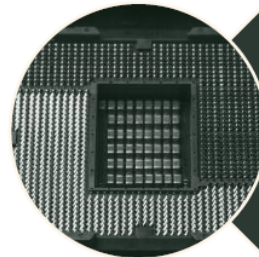
La **computadora** es un **conjunto de recursos** (ej.: procesador, memoria, disco) que se utilizan para el ingreso, **procesamiento** y **almacenamiento de datos**, así como también para llevar a cabo el **control** de estas **operaciones**.

El **sistema operativo** es el encargado de **gestionar** y **controlar** estos recursos.



### MEMORIA.

Es administrada en forma conjunta por el sistema operativo y el hardware de gestión de memoria del procesador



### PROCESADOR.

El sistema operativo dirige al procesador en el uso de los otros recursos de la computadora y administra su uso entre los distintos programas



### E/S.

El sistema operativo el que debe decidir cuando un programa puede utilizar un dispositivo de E/S (entrada / salida) a la vez que controla el acceso y uso de los archivos.

# Evolución del Sistema Operativo



# Evolución del Sistema Operativo



## ■ Procesamiento en serie (primera generación)

- ✓ No existe un sistema operativo
- ✓ SW = código máquina en tarjetas perforadas
- ✓ Los programas ejecutan uno atrás del otro
- ✓ Resultados de la ejecución por impresora o en registros
- Problemas:
  - Planificación
  - Tiempo de configuración



## ■ Sistemas en lotes simple (segunda generación)

- ✓ Surgimiento de los mainframes
- ✓ Surge el sistema operativo en lotes (monitor)
- ✓ Los programas se pasaban a cintas para ser ejecutados
- ✓ Comienzan a aparecer conceptos como:
- Protección de memoria
  - Instrucciones privilegiadas
  - Interrupciones
  - Temporizador

# Evolución del Sistema Operativo

## ■ Sistemas en lotes simple (segunda generación)

### ✚ Desde el punto de vista del SO (monitor)

- ✓ El **monitor** siempre debe estar en memoria (conjunto residente).
- ✓ Su tarea es **cargar programas** desde la cinta magnética hacia la memoria principal.
- ✓ Una vez que el programa termina, el control **vuelve al monitor** para cargar el siguiente trabajo.

### ✚ Desde el punto de vista del procesador

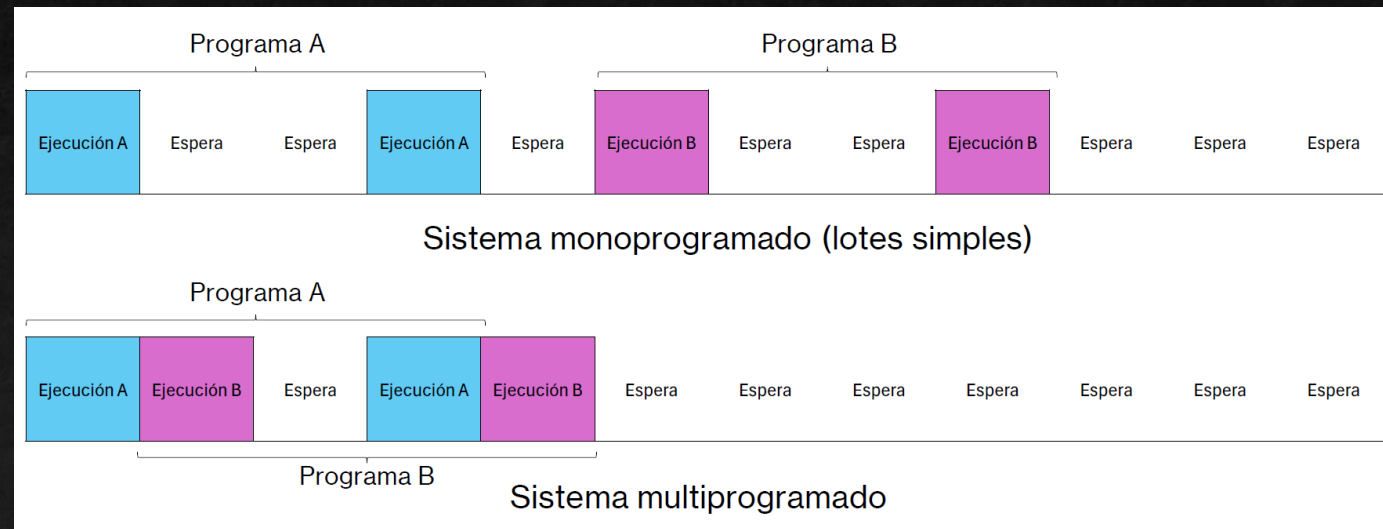
- ✓ El procesador **solo ejecuta instrucciones**, no decide qué programa corre.
- ✓ El monitor carga el programa y coloca una **instrucción de salto** para que el procesador ejecute desde allí.
- ✓ Al finalizar el programa, otra instrucción devuelve el control al monitor para que cargue el próximo.



# Evolución del Sistema Operativo

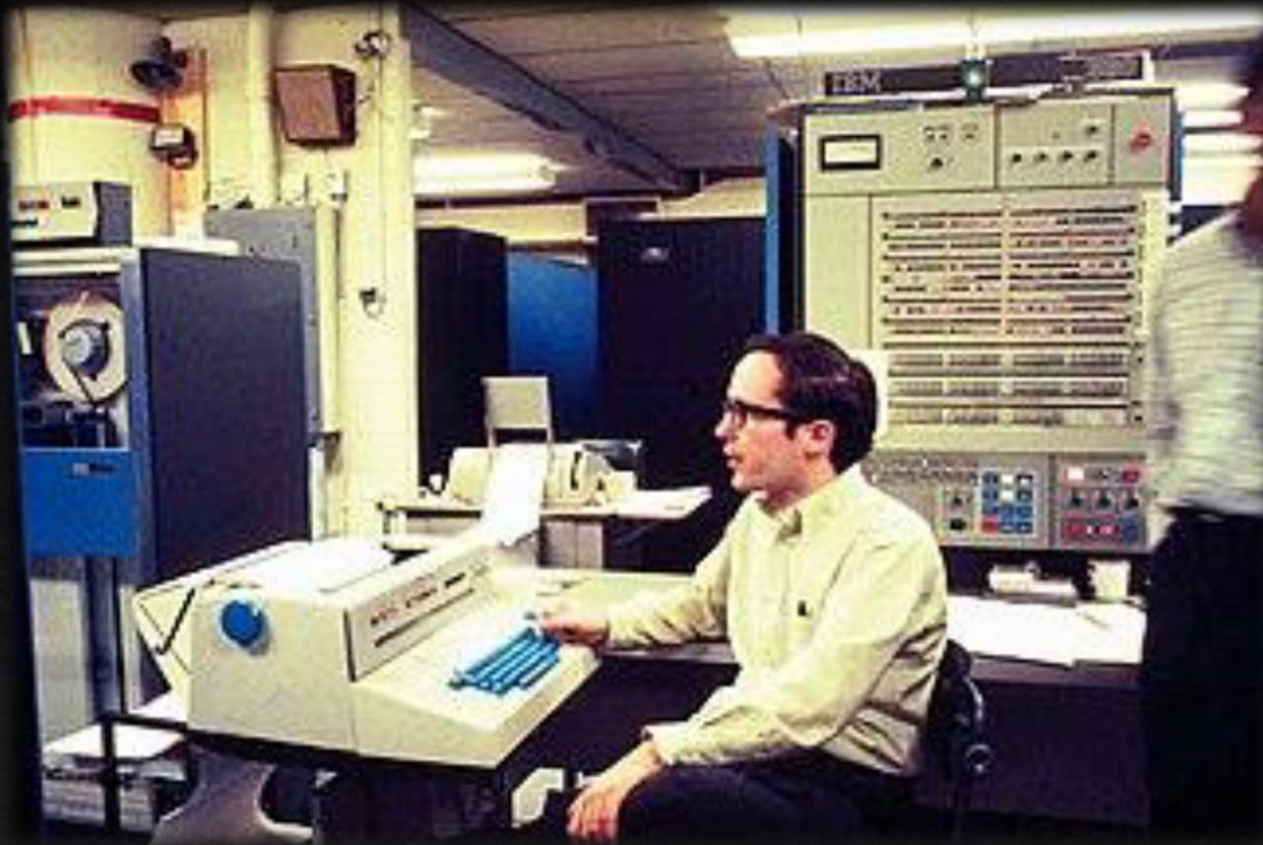
## ■ Sistema multiprogramado (tercera generación)

- ✓ Capacidad **procesar** varias tareas al mismo tiempo.
- ✓ Justificación: **aprovechar** el tiempo que el procesador se encuentra ocioso.
- ✓ Debe existir memoria para cargar más de 1 programa.
- ✓ Surgen las **interrupciones** y el **DMA**.

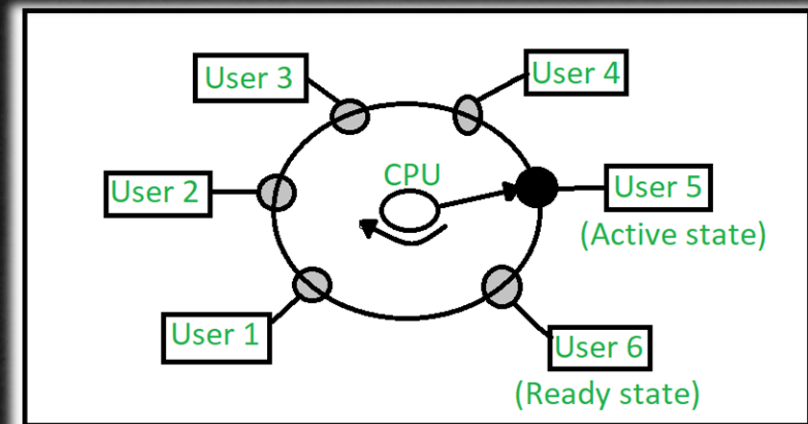


# Evolución del Sistema Operativo

## ■ Sistema tiempo compartido



- ✓ **Comparte** el uso de **procesador** entre los distintos usuarios conectados.
- ✓ Es un concepto similar a la **multiprogramación** pero que permite gestionar **múltiples usuarios** en vez de programas.
- ✓ Estos sistemas entrelazan la **ejecución** de los **programas** de usuario **utilizando** pequeños **intervalos** de tiempo.



# Conceptos Básicos

## Procesador

- Conjunto de instrucciones propio
- Registros (memoria interna)

## Ciclo de ejecución

- Pasos que realiza el procesador para ejecutar cada instrucción

## Interrupciones

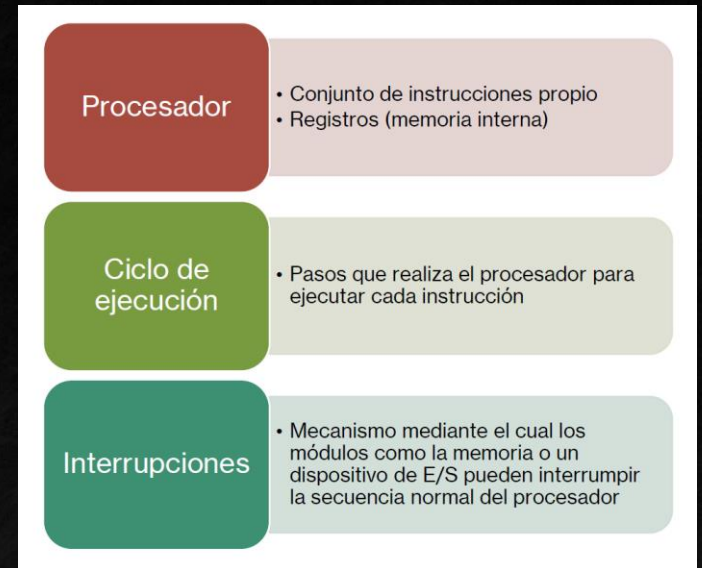
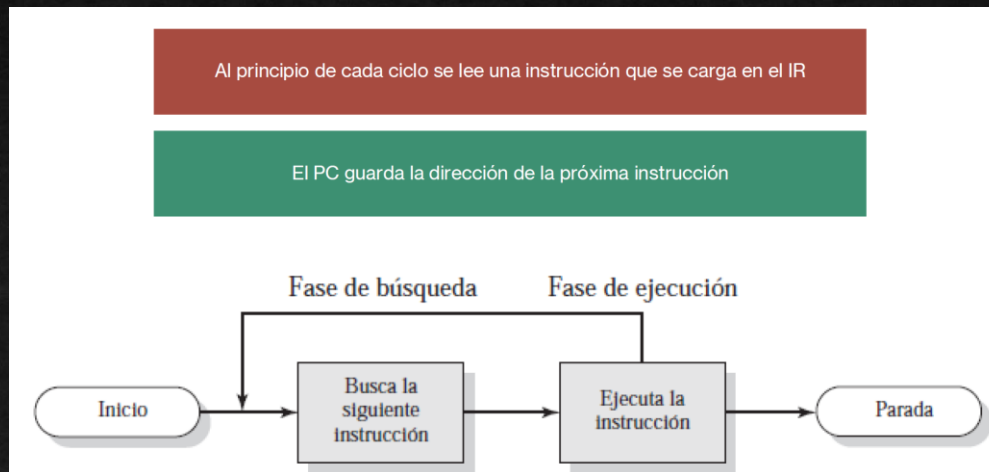
- Mecanismo mediante el cual los módulos como la memoria o un dispositivo de E/S pueden interrumpir la secuencia normal del procesador

# Conceptos Básicos

## ■ Registros del procesador

- ✓ Visibles por el usuario: son utilizados por programas de usuario
- ✓ De control y estado
  - IR (Instruction Register): guarda la instrucción actual a ejecutar.
  - PC (Program Counter): indica la dirección de la próxima instrucción.
  - PSW (Program Status Word): contiene el estado del procesador (banderas, modo, etc.).

## ■ Ciclo de ejecución



## ■ Interrupciones

- ✓ Señal que pausa la ejecución normal del procesador para atender un nuevo evento.
- ✓ Es el mecanismo que permite a la memoria o a un dispositivo de E/S interrumpir el flujo normal de ejecución del procesador.
- ✓ Tipos de interrupciones:
  - De programa
  - De E/S
  - Por temporizador
  - Por fallo de HW

# Pilares de un sistema operativo



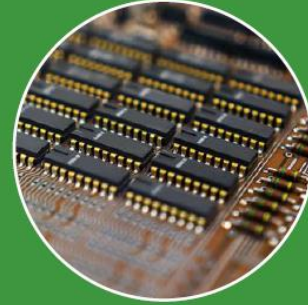
Procesos



Gestión de  
Almacenamiento



Seguridad



Estructura



Gestión de  
recursos

# Pilares de un sistema operativo

## ■ Procesos



- ✓ Es un **programa en ejecución**, considerado la **unidad básica de trabajo del sistema operativo**, con un estado y recursos asignados.
- ✓ Se compone de: **Datos + Instrucciones + Contexto de ejecución (registros, memoria, recursos asignados)**.

## ■ Gestión de Almacenamiento



- ✓ El SO tiene 5 responsabilidades esenciales en el almacenamiento:
  1. Aislamiento de procesos
  2. Asignación automática
  3. Soporte para programación modular
  4. Protección y ctrl. de acceso
  5. Almacenamiento a largo plazo
- ✓ Memoria virtual
  - Páginas
  - Page Fault

## ■ Seguridad



- ✓ Disponibilidad
- ✓ Confidencialidad
- ✓ Integridad
- ✓ Autenticidad

## ■ Estructura



- ✓ Modularidad
- ✓ Interfaces bien definidas y sencillas
- ✓ Estructura jerárquica (niveles)

## ■ Gestión de los recursos

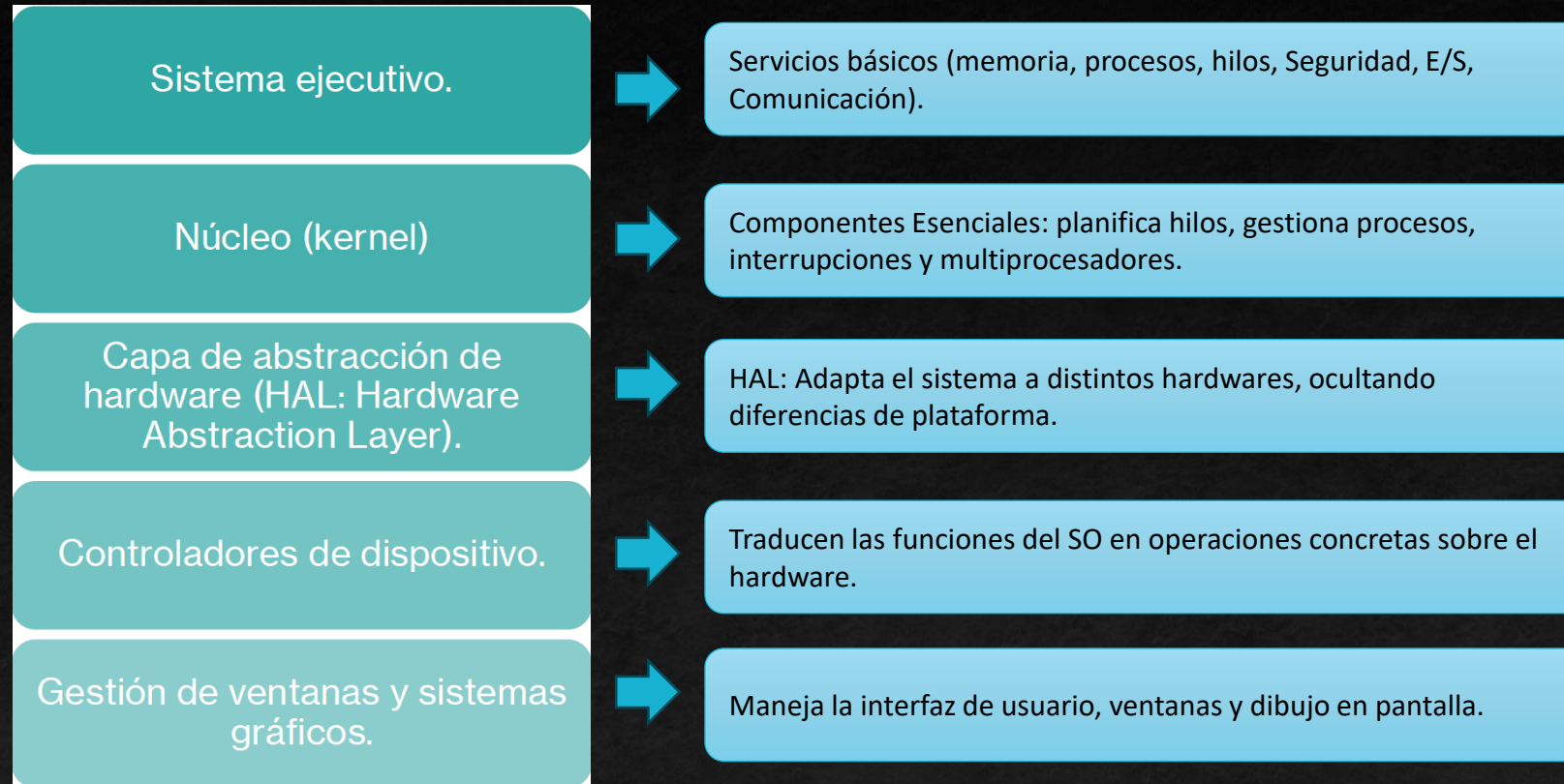


Un sistema operativo debe garantizar:

- ✓ **Equidad:** acceso justo a los recursos.
- ✓ **Prioridad:** responder según importancia del proceso/usuario.
- ✓ **Eficiencia:** aprovechar al máximo CPU, memoria y dispositivos.



# Organización del Sistema Operativo



# Tipos de Kernel (Núcleo)

Monolítico



Único proceso: todo el sistema corre en el espacio del kernel; un único bloque con todos los servicios integrados.

En capas



Estructura jerárquica en anillos; cada capa usa los servicios de la inferior mediante llamadas al sistema.

Microkernel



Solo funciones básicas en el kernel; los demás servicios se ejecutan como procesos separados.

# ¿Preguntas ?



## Clase 2

- Descripción de procesos
- Modelos de estado
- Control de procesos
- Modos de ejecución

# Procesos

# Descripción de un Proceso

## ■ Representación física de un proceso

### Glosario:

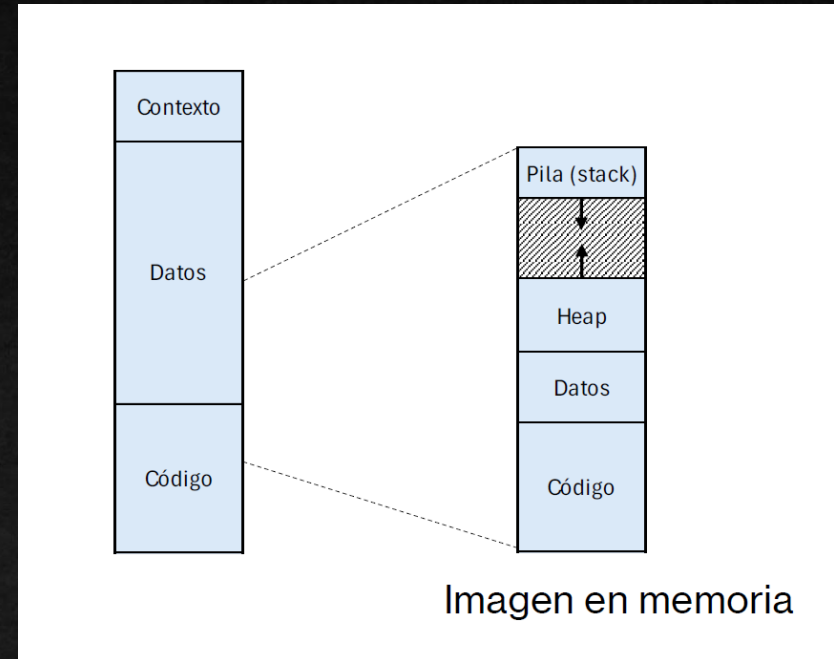
- ✓ Contexto: estado del proceso (registros, contador de programa, banderas).
- ✓ Datos: variables globales y estáticas usadas durante la ejecución.
- ✓ Código: instrucciones del programa que deben ejecutarse.
- ✓ Pila (Stack): variables locales y control de llamadas a funciones.
- ✓ Heap: memoria dinámica solicitada en tiempo de ejecución.



## ■ Proceso

Es una unidad de actividad caracterizada por:

- ✓ Hilo secuencial de ejecución
- ✓ Un estado
- ✓ Un conjunto de recursos.



# Conceptos Básicos: Procesos

## Estado del proceso

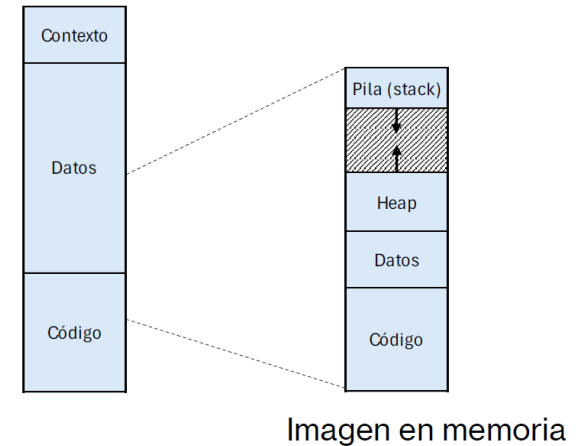
- Datos que el SO y el procesador utilizan para gestionar y ejecutar el proceso
- Compuesto por PC, IR, PSW, entre otros

## Tabla de Procesos

- Una entrada por cada proceso
- Contiene información para gestionar cada proceso

## Espacio de Memoria

- Todas las referencias a memoria son relativas al registro base
- El procesador Controla el acceso a memoria mediante un registro de base y otro de desplazamiento



## Glosario:

- ✓ PC (Program Counter): Registro que guarda la dirección de la próxima instrucción a ejecutar.
- ✓ IR (Instruction Register): Contiene la instrucción actual que está siendo ejecutada por el procesador.
- ✓ PSW (Program Status Word): Registro que almacena el estado del procesador (banderas, modo de ejecución, prioridades, etc.).
- ✓ Espacio de Memoria: Registro Base + Desplazamiento.

# Descripción de Procesos

## ■ Estructuras y tablas

tar	Ver	Buscar	Terminal	Ayuda
2237	0.0	0.4	455720	19528
2259	0.0	0.2	322356	11916
2275	0.0	0.4	483736	16072
2280	0.0	0.2	348872	8932
2286	0.0	0.3	252040	14768
2303	0.0	0.0	11416	720
2308	0.0	0.0	0	0
2309	0.0	0.0	0	0
2311	0.0	0.0	0	0
2360	0.0	0.4	497572	19476
2374	0.1	1.4	1002472	56444
2384	0.0	0.1	104440	4300
2391	0.0	0.1	124592	4184
2399	24.7	13.9	1523600	55774
2512	0.0	0.1	378072	6956
2566	0.5	0.7	697156	29852
2573	0.0	0.0	14828	1820
2574	0.0	0.1	27064	5588
2602	0.0	0.0	4448	800
2603	0.0	0.0	4340	644
2610	0.0	0.0	4448	1684
2644	0.0	0.0	4348	644
2698	0.0	0.0	22708	2756

## ■ Gestión de Procesos

### Tabla de Procesos

- Almacena una entrada por cada proceso.

¿Cómo está compuesta cada una de esas entradas?

### BCP (bloque de control de proceso) ó PCB

- Es una estructura de datos que incluye:
  - PID
  - Estado
  - Prioridad
  - Datos de contexto

### Glosario:

- ✓ PCB (Process Control Block / Bloque de Control de Proceso)  
Contiene la info necesaria gestionar el proceso.
- ✓ PID (Process Identifier / Identificador de Proceso).
- ✓ Identificador único del proceso.

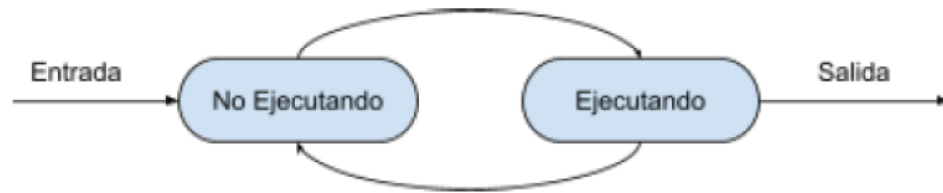
# BCP Bloque de Ctrl de Proceso

Es a través del BCP que el SO puede proporcionar multiprogramación

0	1	2	3	4	6	7	8	9	10	11	12	13	14	15
TICK	PID	PPID	USERID	STATE	SWAP FLAG	INODE INDEX	INPUT BUFFER	MODE FLAG	USER AREA SWAP STATUS	USER AREA PAGE NUMBER	KERNEL STACK POINTER (KPTR)	USER STACK POINTER (UPTR)	PTBR	PTLR

# Modelos de Estado - 2

## ■ Modelo de 2 Estados



### ¿Qué sucede al crear un proceso?

1. El SO crea el BCP y lo agrega al sistema en estado "No ejecutando"
2. Los procesos se insertan en una cola de espera
3. Cuando el procesador es interrumpido el proceso pasa a ejecutarse => también cambia de estado

# Modelos de Estado - 5

## ■ Modelo de 5 Estados

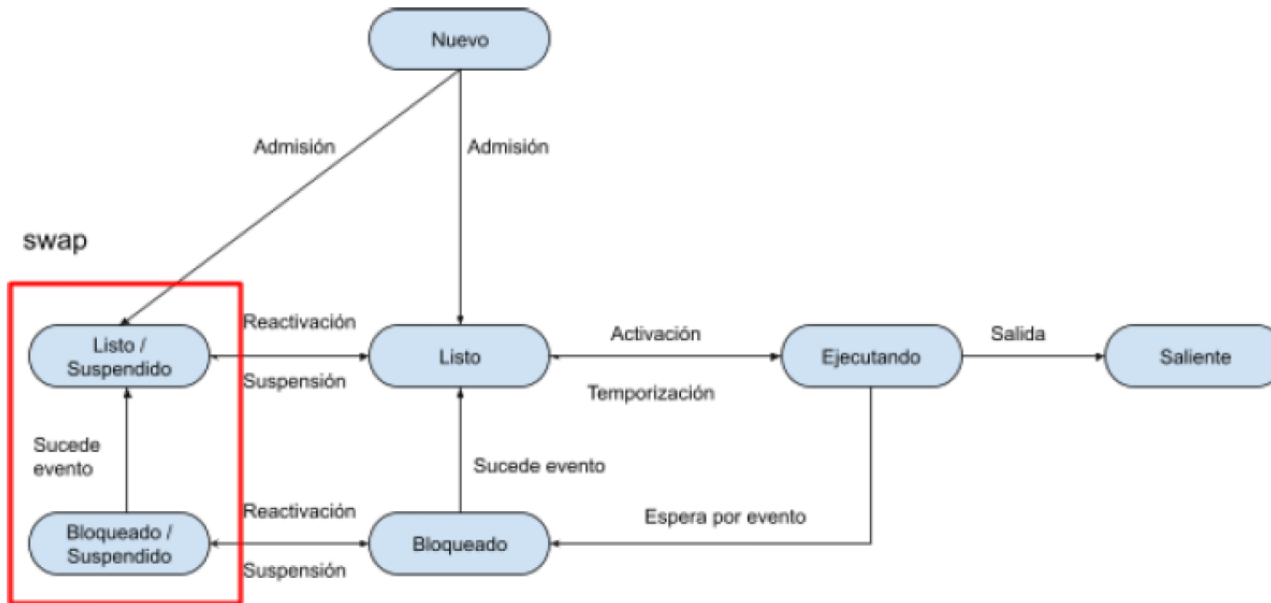


### ¿Qué sucede internamente?

1. Al recibir un pedido de creación de proceso se crean todas las estructuras (PCB) y se asigna el PID
- **Nota:** Cuando un proceso se encuentra en estado Nuevo, el programa permanece en almacenamiento secundario

# Modelos de Estado - 7

## ■ Modelo de 7 Estados

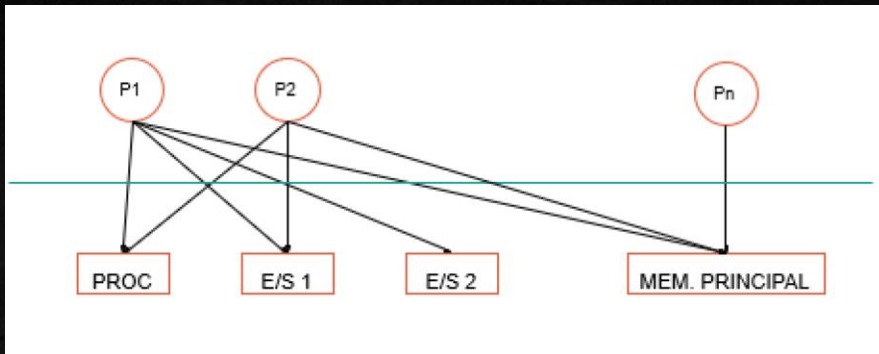


### ¿Qué es el swapping?

- Es una operación que implica mover una parte o todo el proceso de memoria principal a disco.
- Se ejecuta cuando ninguno de los procesos que se encuentra en memoria principal se encuentra en estado Listo
- Se utiliza para obtener espacio para admitir un nuevo proceso en el sistema.

# Control de Procesos – Modos de Ejecución

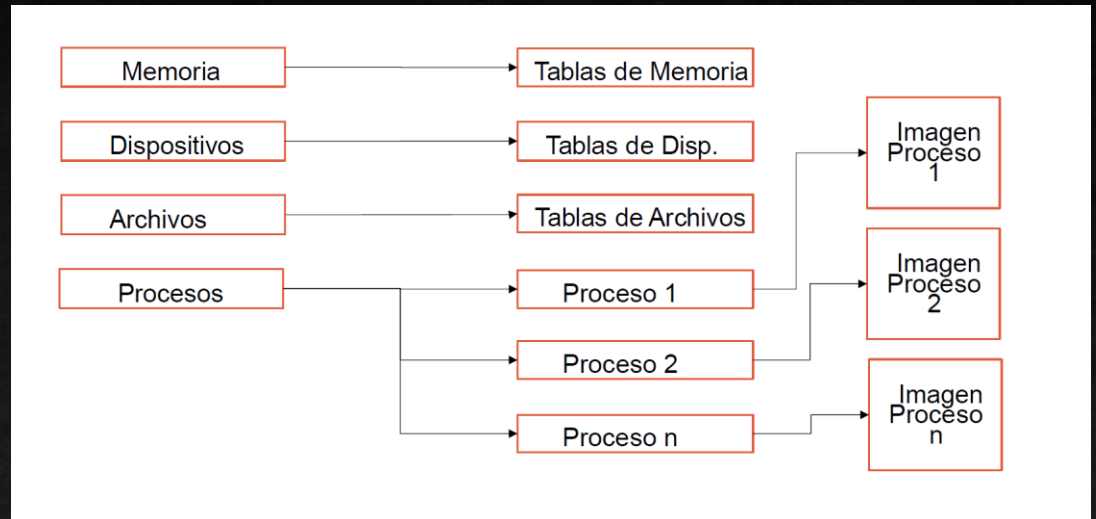
## ■ El SO Gestiona los Recursos



## ■ Sobre los recursos

- ✓ El SO gestiona recursos: asigna CPU, memoria, archivos y dispositivos a los procesos.

## ■ Estructuras de Control



## ■ Sobre las estructuras de control

- ✓ Estructuras de control: **tablas internas** que organizan la información de memoria, dispositivos, archivos y procesos.
- ✓ **Imagen de proceso**: cada proceso tiene su propia representación en memoria con sus **datos y contexto**.

# Modos de Ejecución de un Proceso

## Usuario

- Existe para proteger al sistema
- Las referencias a memoria son relativas al BR
- Sólo puede realizar syscall

## Kernel (Núcleo)

- Control absoluto del sistema
- Las referencias a memoria son a direcciones físicas.

El modo de ejecución se almacena en el PSW

### Glosario:

- ✓ **PSW (Program Status Word)** o registro de estado del procesador CPU.
- ✓ **BR (Base Register / Registro Base):** Almacena la dirección inicial de la memoria asignada a un proceso. Todas las referencias a memoria del proceso son relativas a este registro, lo que permite protección y aislamiento entre procesos.
- ✓ **Syscall (System Call):** Mecanismo que permite a un programa en modo usuario pedir un servicio al sistema operativo.

# Funciones en Modo Núcleo - Kernel

## Procesos

- Creación y terminación
- Planificación y activación
- Intercambio
- Sincronización y Comunicación entre procesos

## E/S

- Gestión de buffers
- Reserva de dispositivos para los procesos

## Modo Núcleo

## Memoria

- Reserva
- Swapping
- gestión de páginas

## Soporte

- Interrupciones
- Auditoría
- Monitoreo

# Cambio de contexto – Obtener el control del proceso

Un cambio de contexto o *context switch* ocurre cada vez que el sistema operativo obtiene el control sobre el proceso que está actualmente en ejecución

## Causas de context switch

### Interrupción

- Reloj
- E/S

### Excepción

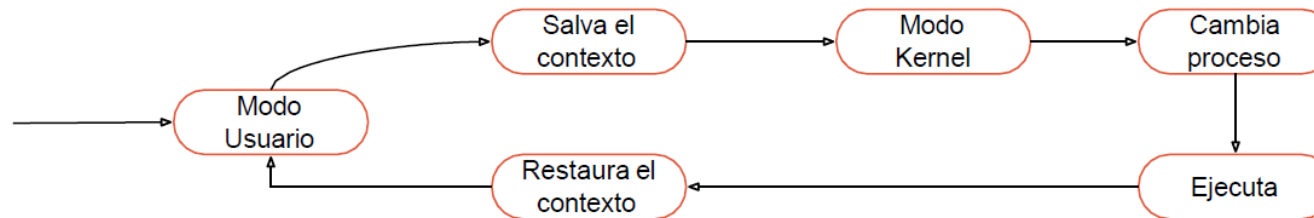
- Faults
- Traps
- Aborts

### Syscall

- Abrir/Cerrar archivo
- Leer/Escribir archivo
- Crear procesos/hilos

# Cambio de Modo

Sólo ocurre si ante un evento que implique un cambio de contexto, el proceso en ejecución era un proceso de usuario



# Modos de Ejecución del Sistema Operativo



## Núcleo sin Procesos

- El SO es una única pieza de sw que ejecuta por fuera de todo proceso



## Núcleo en procesos de usuario

- El SO se ejecuta virtualmente en el contexto del proceso de usuario



## Basado en procesos

- El SO es una colección de procesos separada de los procesos de usuario



# Modos de Ejecución del Sistema Operativo



## Núcleo sin Procesos

- El SO es una única pieza de sw que ejecuta por fuera de todo proceso

### ■ Núcleo Sin Procesos

- ✓ El sistema operativo tiene su propia pila y región de memoria independientes.
- ✓ El concepto de proceso se aplica únicamente a los programas de usuario, no al núcleo.



## Núcleo en procesos de usuario

- El SO se ejecuta virtualmente en el contexto del proceso de usuario

### ■ Núcleo en Procesos de usuario

- ✓ El SO ofrece rutinas que el usuario invoca para distintas funciones.
- ✓ Estas rutinas se ejecutan dentro del proceso de usuario.
- ✓ No hay un cambio de proceso completo, se evita el doble cambio.
- ✓ SO ejecuta rutinas en el contexto del usuario, sin cambio de proceso completo.



## Basado en procesos

- El SO es una colección de procesos separada de los procesos de usuario

### ■ Basado en Procesos

- ✓ El sistema operativo es una colección de procesos independientes.
- ✓ Permite ejecutar en paralelo procesos de usuario y de sistema.
- ✓ Ventaja: mayor eficiencia y paralelismo en la ejecución.

# ¿Preguntas ?



## Clase 3

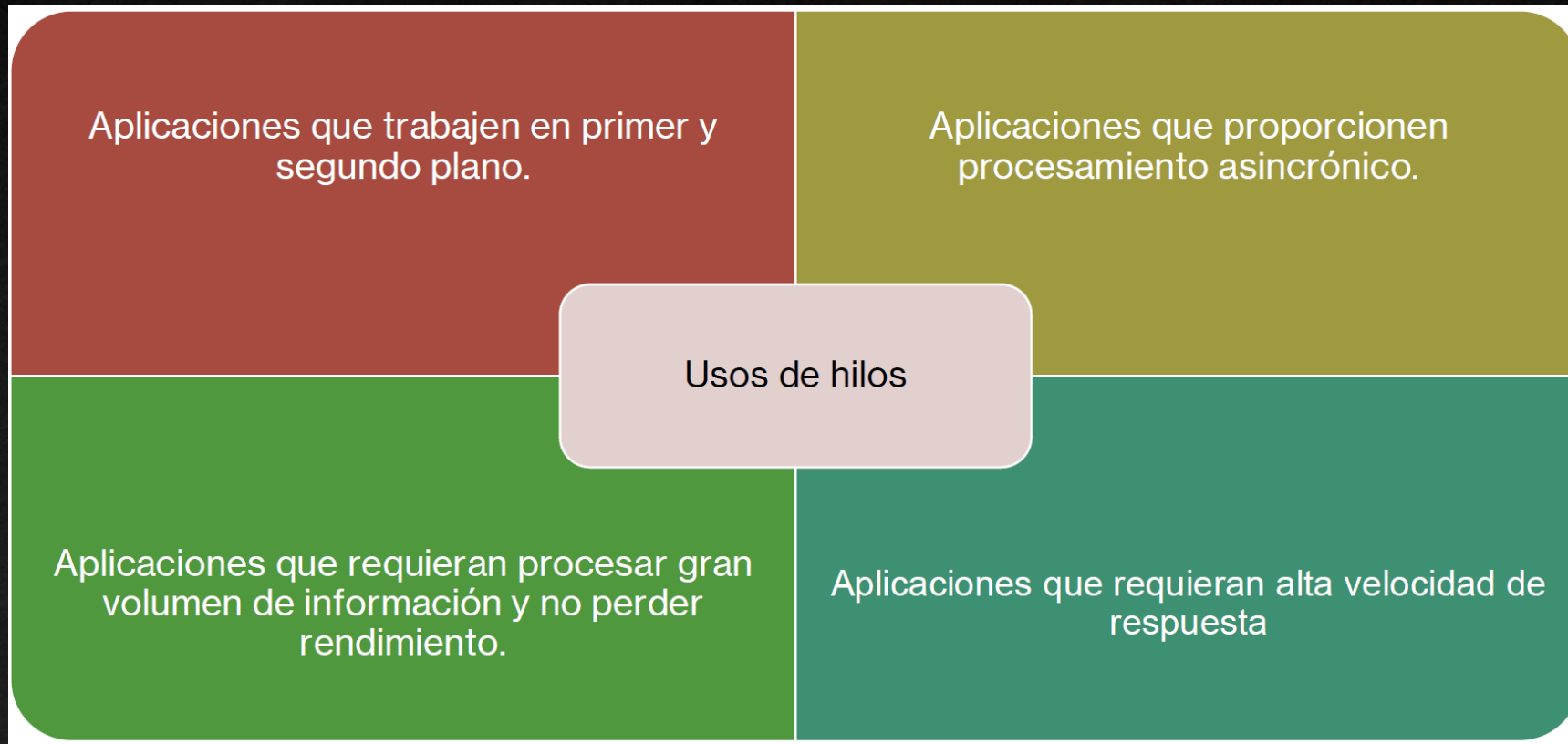
- Conceptos Básicos
- Procesos vs. Hilos
- Planificación de Hilos
- ULT (User Level Thread)
- KLT (Kernel Level Thread)

# Procesos e Hilos

# Conceptos Básicos - Definiciones

## ■ Hilo

- ✓ “Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea.” (Wikipedia).



# Diferentes enfoques



## ■ Enfoque tradicional

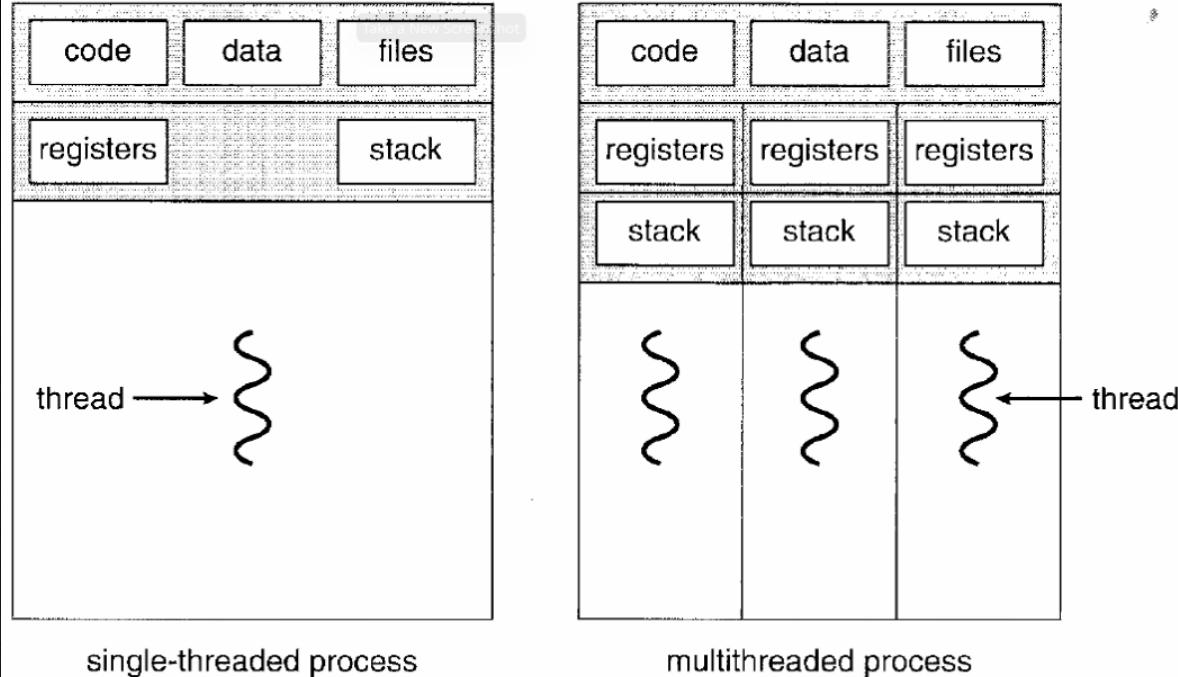
- ✓ El proceso era visto como una unidad **indivisible**: poseía recursos (espacio de direcciones, código, datos, pila, registros) y además la planificación de **CPU** recaía directamente en él.
- ✓ Esto llevaba a **procesos más pesados** y **cambios de contexto más costosos**.

## ■ Enfoque moderno

- ✓ Separamos propiedad de recursos (el proceso) de la **planificación** (los hilos).
- ✓ Un **proceso** puede tener **múltiples hilos**, cada uno ejecutándose de manera concurrente dentro del mismo espacio de direcciones.
- ✓ Ventajas: mayor **eficiencia**, mejor aprovechamiento del **multiprocesador** y menor costo en los cambios de contexto.



# Multihilo



Es la capacidad de un sistema operativo de dar soporte a **múltiples hilos** de ejecución **en un solo proceso**.

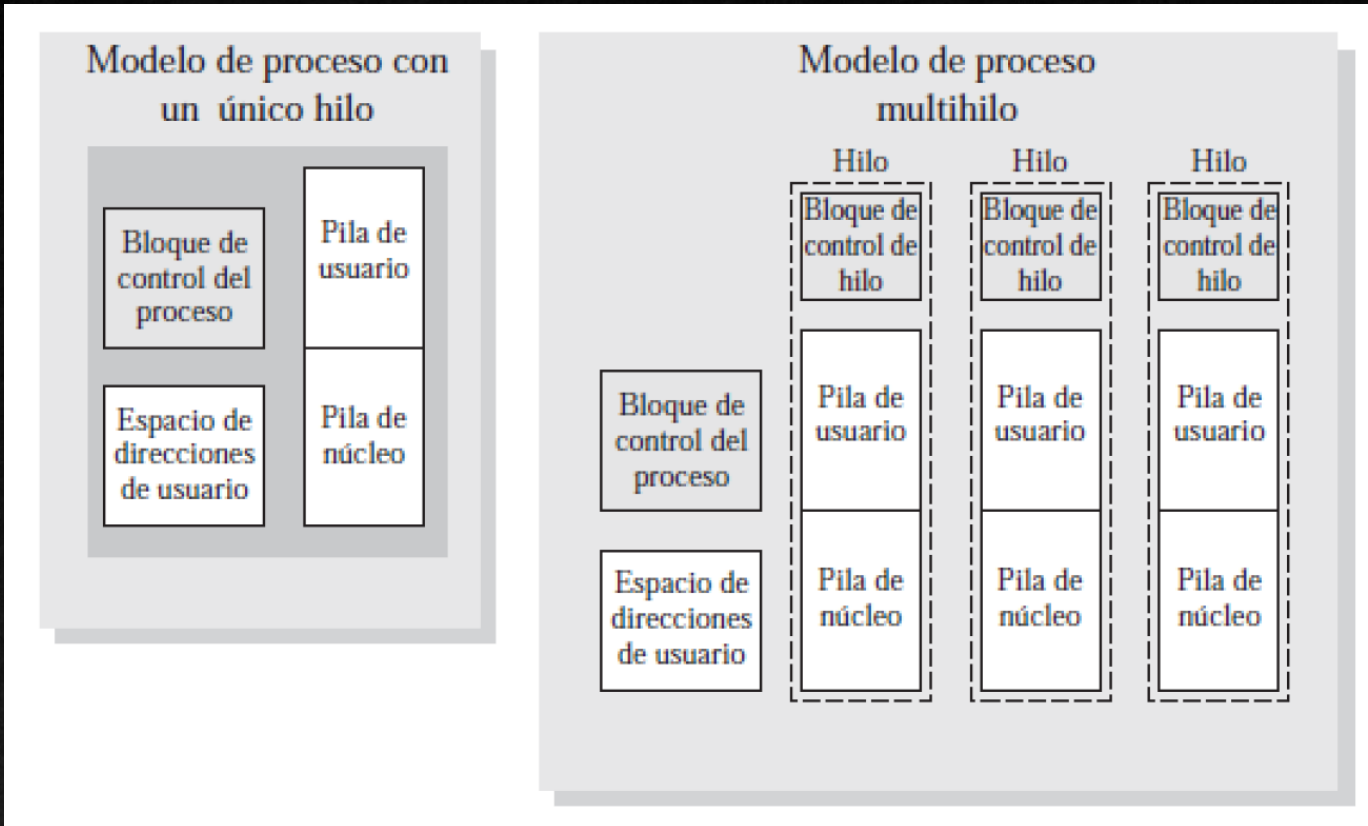
Cada hilo cuenta con:

- Código
- Registros (estado de ejecución, pc, etc)
- Una pila de ejecución
- Espacio de almacenamiento para variables locales.
- Acceso a memoria y recursos de su proceso.

El acceso a memoria y recursos asignados a un proceso es compartido con todos sus hilos.

# Procesos vs Hilos

## Diferencias



## ■ Proceso con único hilo

- ✓ Cada proceso posee **un solo flujo de ejecución**.
- ✓ El cambio de proceso implica guardar y restaurar todo el **contexto**, lo que es **costoso en tiempo**.
- ✓ Adecuado para **sistemas simples**, pero **poco eficiente** en tareas concurrentes.

## ■ Proceso multihilo

- ✓ Un proceso puede contener **varios hilos**, todos **compartiendo el mismo espacio de direcciones y recursos**.
- ✓ Cada **hilo** tiene su propio **contador de programa, registros y pila**, pero comparte código y datos con el resto.
- ✓ Ventajas:
  - Concurrencia dentro de un mismo proceso.
  - **Mejor uso del CPU y multiprocesadores.**
  - Cambio de hilo más rápido que cambio de proceso.

## 👉 En resumen:

- ✓ Proceso = contenedor de recursos.
- ✓ Hilo = unidad de ejecución.
- ✓ El multihilo permite más eficiencia, escalabilidad y rapidez en sistemas modernos.

# Ventajas de utilizar hilos



## ■ Creación (rápida):

- ✓ Crear un hilo es como abrir una pestaña nueva en el navegador; crear un proceso es como abrir otro navegador completo.

## ■ Finalización (menos costosa):

- ✓ Cerrar una pestaña del navegador es más rápido que cerrar todo el programa.

## ■ Cambio (más ágil):

- ✓ Cambiar entre pestañas dentro de un mismo navegador es más rápido que pasar de un navegador a otro distinto.

## ■ Comunicación (directa):

- ✓ Dos pestañas dentro del mismo navegador pueden compartir información (ej. cookies o historial) fácilmente; en cambio, dos navegadores distintos requieren más pasos para compartir.

# Planificación

## ■ Definición:

- ✓ En un sistema multihilo, la planificación se realiza a nivel de hilos.

## Estados de los hilos:

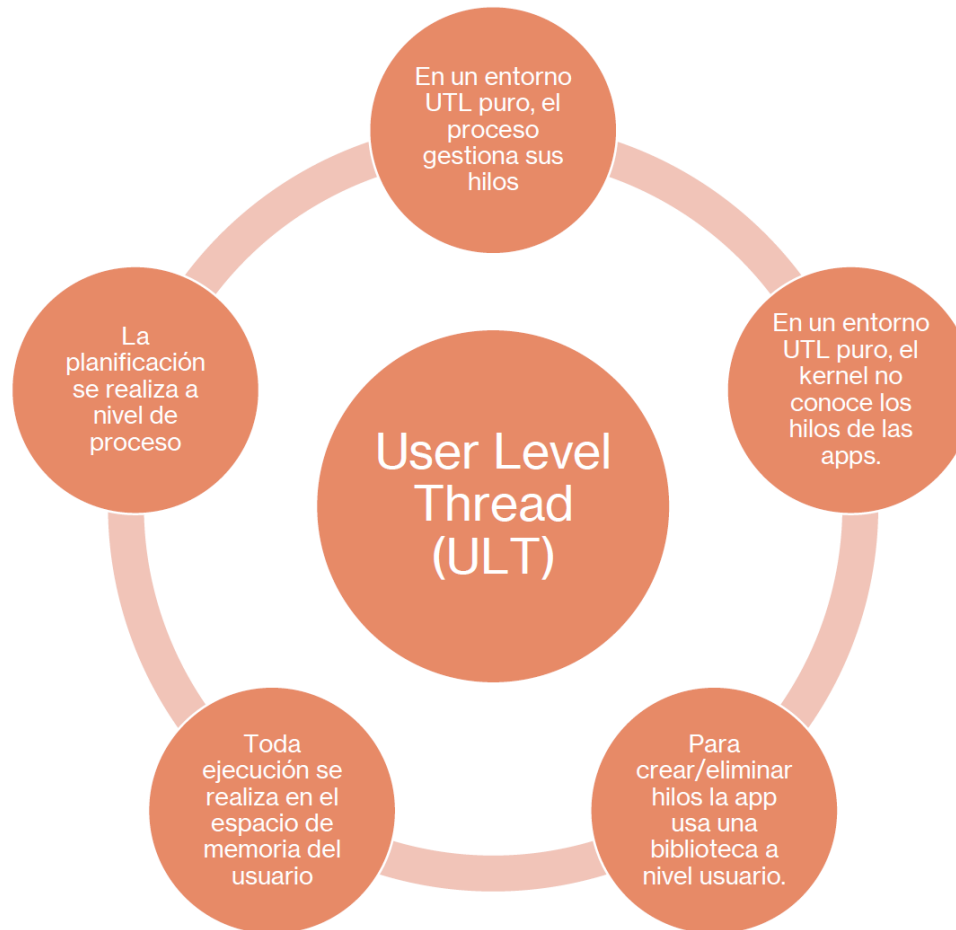
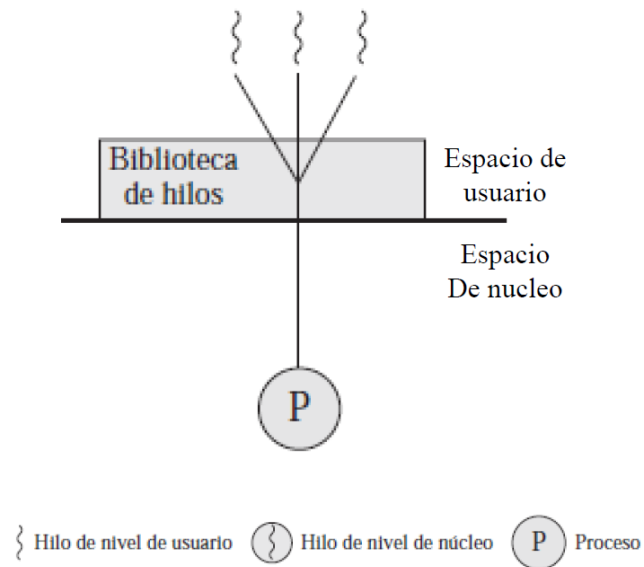


## ■ Consideraciones

- ✓ **Suspender** un proceso implica que todos sus **hilos** sean **suspendidos**
- ✓ **Finalizar** un proceso implica que todos sus **hilos** sean **finalizados**
- ✓ No tiene sentido manejar estados como "Suspendido", ya que son conceptos que se aplican a nivel de proceso.

# User Level Thread – Hilo Nivel Usuario

## ULT



## Ventajas

Un cambio de hilo no implica un doble cambio de modo

La aplicación puede especificar la planificación.

## Desventajas

Un hilo puede bloquear todo el proceso.

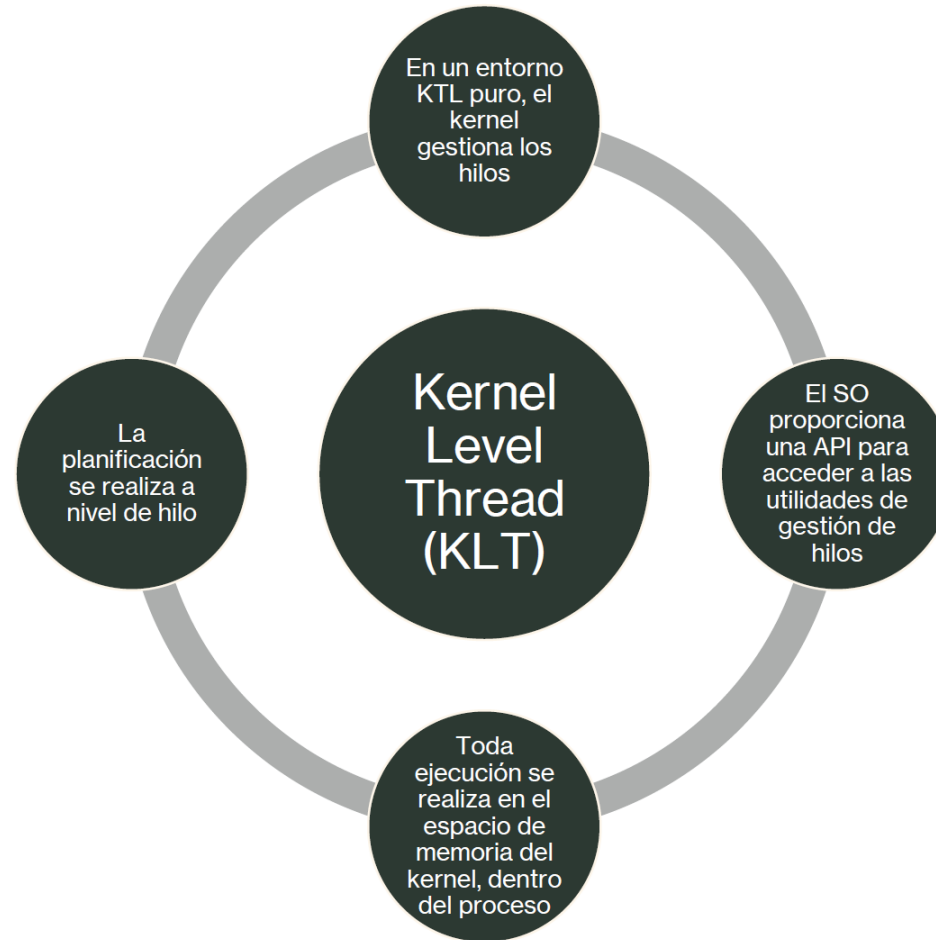
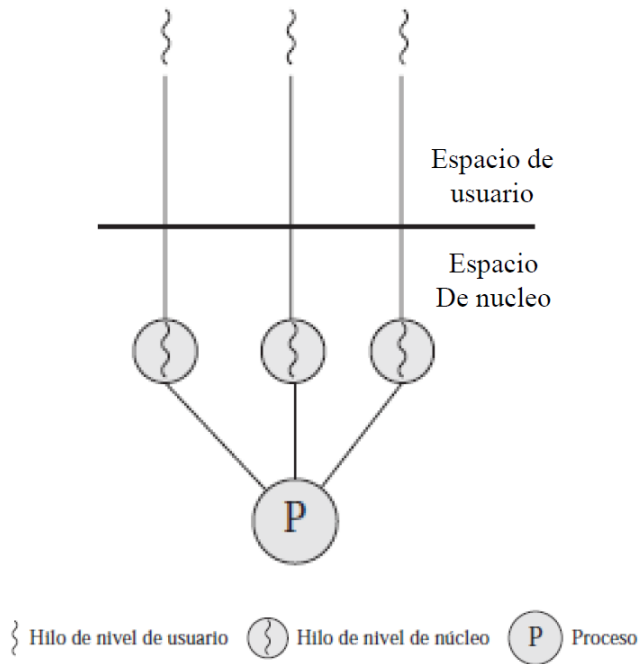
No se beneficia del multiprocesamiento.

- **El kernel no puede administrar los hilos directamente**, ya que no tiene conocimiento de ellos.

👉 Esto significa que si un hilo queda bloqueado (por E/S, por ejemplo), **todo el proceso se bloquea**, aunque otros hilos estén listos para ejecutarse.

# Kernel Level Thread – Hilo Nivel Núcleo

## KLT



## Ventajas

El kernel puede planificar múltiples hilos en distintos procesadores.

Si un hilo se bloquea, no bloquea todo el proceso.

Las rutinas del kernel (como manejo de excepciones) pueden ser multihilo

## Desventajas

El cambio de un hilo a otro produce un doble cambio de modo.

👉 El kernel gestiona los hilos:

Soporta multiprocesamiento y evita bloqueos de todo el proceso, aunque cada cambio de hilo requiere pasar por el kernel.

# ¿Preguntas ?



## Clase 4

- Principios de concurrencia
- Interacción entre procesos
- Exclusión mutua por hardware
- Exclusión mutua por software

# Concurrencia, exclusión mutua y sincronización

# Principios de concurrencia

## ■ Acerca de la Concurrencia

- ✓ La concurrencia es fundamental en áreas como la **multiprogramación**, el **multiprocesamiento** y el **procesamiento distribuido**.
- ✓ La concurrencia abarca **aspectos** como la **comunicación** entre **procesos**, la compartición o **competencia** por los **recursos**, la **sincronización** de actividades y la reserva de **tiempo** del **procesador**.

👉 Concurrencia ejemplo:

Varios procesos progresan de manera simultánea compartiendo recursos, y el SO coordina para evitar conflictos.

## ■ Que es la Concurrencia

- ✓ La concurrencia analiza situaciones en las que **dos o más procesos** se ejecuten de forma tal que requieran **acceder a un recurso compartido al mismo tiempo**.
- ✓ El sistema operativo debe proporcionar mecanismos para mantener la **estabilidad del sistema** y **coordinar** las **ejecuciones** de los **procesos** involucrados.



# Multiprogramación vs. Multiprocesamiento

## ¿Presentan problemas distintos?

### Recursos Globales

- El orden de ejecución es crítico

### Asignar recursos

- La gestión de recursos es compleja para el SO y puede conducir a interbloqueo

### Gestión de errores

- Errores no determinísticos y no reproducibles

👉 En resumen:

- ✓ **Multiprogramación:** Alternancia en un solo CPU (conurrencia lógica).
- ✓ **Multiprocesamiento:** Ejecución paralela en varios CPUs (conurrencia física).

## ■ Multiprogramación

- ✓ **Único procesador** ejecuta varios programas alternadamente.
- ✓ **El SO organiza y decide** que **procesos** se **ejecuta** en cada momento.
- ✓ La **CPU** debe estar **ocupada siempre**, mientras otros procesos esperan por E/S.

## ■ Multiprocesamiento

- ✓ **Más procesadores** trabajando en paralelo sobre **distintos procesos o hilos**.
- ✓ **Ejecución concurrente real**, aumentando **rendimiento** y **capacidad de respuesta**.
- ✓ Mayor **coordinación del SO** para la **asignación de recursos y sincronización**.



# Velocidad relativa de ejecución – Recursos

- **La velocidad – es impredecible**

No se puede predecir la **velocidad** relativa de ejecución de un **proceso**, debido a que **depende de la actividad de otros procesos**, la **gestión de las interrupciones** y las **políticas de planificación del sistema operativo**.

- **¿Cómo solucionamos estos problemas?**

Hay que **proteger** los **recursos compartidos** y la única manera de hacerlo es **controlar el código** que accede a los mismos.



## Definición

- **Condición de Carrera**

El resultado final de un recurso compartido depende de la **coordinación de los procesos**, es decir del **orden de ejecución**.



# Interacción de procesos

## ¿Cómo interactúan los procesos?



### No se perciben entre si

- Son procesos independientes que no se pretende que trabajen juntos y que no realizan intercambio de información. El SO debe ocuparse de la competencia por los recursos.

Procesos independientes



### Se perciben indirectamente entre si

- Son procesos, que no tienen conocimiento explícito del resto de los procesos a través de su PID pero saben que comparten el acceso a un objeto común, por ejemplo, un dispositivo de E/S, memoria compartida, etc

Se perciben indirectamente



### Se perciben directamente entre si

- Son procesos que se comunican entre sí ya que conocen el PID del otro y son diseñados para trabajar en conjunto. La comunicación proporciona una manera de sincronizar las actividades que realizan

Se perciben directamente



# Conclusiones de la interacción entre procesos



# Interacción entre procesos

## La exclusión mutua genera...

### Deadlock

- Recursos bloqueados hacen que se bloqueen los procesos

### Starvation

Procesos que no terminan de ejecutar por falta de recursos

#### 👉 En resumen:

- ✓ Deadlock = Interbloqueo (procesos atrapados esperando recursos).
- ✓ Starvation = Inanición (un proceso queda esperando indefinidamente por falta de recursos).

# Exclusión mutua – Soporte por Hardware

## Técnicas



### Semáforos

- Son estructuras que permiten sincronizar el acceso a un recurso crítico mediante el uso de señales



### Monitores

- Es un módulo de software desarrollado en un lenguaje de programación (alto nivel) que proporciona funcionalidad equivalente a un semáforo



### Mensajes

- Los mensajes son utilizados por procesos cooperantes para comunicarse y sincronizarse entre ellos



# Exclusión mutua – Soporte por Hardware – Semáforos

## Propiedades de los semáforos

### Tipos de semáforos

- “Comunes” o con contador
- Binarios o mutex (valores posibles 0 y 1)

### Operaciones

- Un semáforo se define como una variable con un valor entero
- semSignal y semWait para s comunes
- semSignalB y semWaitB para s binarios

```
struct semaphore {  
    int cuenta;  
    queueType cola;  
}  
void semWait(semaphore s)  
{  
    s.cuenta--;  
    if (s.cuenta < 0)  
    {  
        poner este proceso en s.cola;  
        bloquear este proceso;  
    }  
}  
void semSignal(semaphore s)  
{  
    s.cuenta++;  
    if (s.cuenta <= 0)  
    {  
        extraer un proceso P de s.cola;  
        poner el proceso P en la lista de listos;  
    }  
}
```

- Un semáforo puede ser inicializado a un valor no negativo
- La operación semWait decrementa el valor del semáforo. Si pasa a ser negativo, el proceso que ejecuta semWait se bloquea
- La operación semSignal incrementa el valor del semáforo. Si pasa a ser negativo o cero, se desbloquea uno de los procesos bloqueados con semWait

### ■ Idea clave:

- ✓ semWait (P): resta 1 al semáforo → si el valor es negativo, el proceso se bloquea.
- ✓ semSignal (V): suma 1 al semáforo → si el valor  $\leq 0$ , desbloquea a un proceso de la cola.
- ✓ Garantía: solo un proceso a la vez accede a la sección crítica → se evita la condición de carrera.

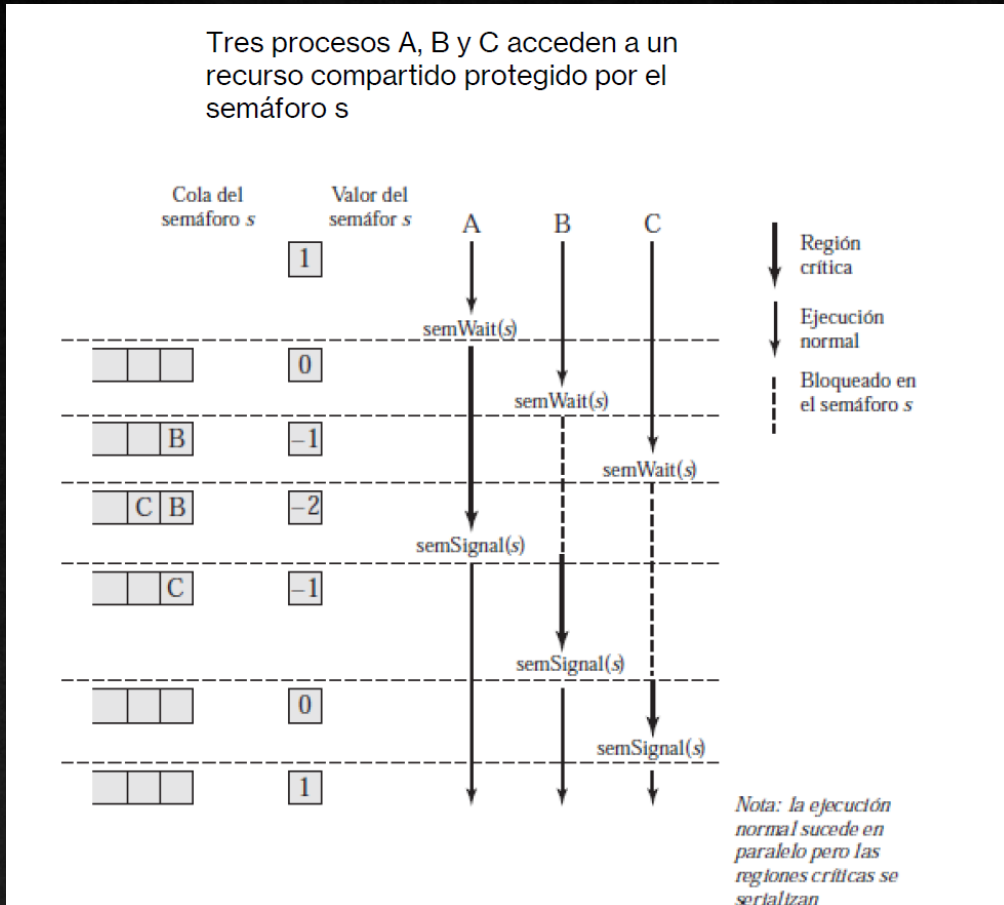
### ■ En resumen:

El semáforo actúa como un guardia que permite o no entrar a la sección crítica. Si alguien más ya está adentro, los demás esperan en fila hasta que el recurso se libere.



# Exclusión mutua – Soporte por Hardware – Semáforos

**Ejemplo de un semáforo** – Solo un proceso ingresa ... los demás esperan ...



## ■ Objetivo:

- ✓ Hay tres procesos (A, B y C) que quieren acceder a un recurso compartido.
- ✓ Ese recurso está protegido por un semáforo s inicializado en 1 (es decir, solo un proceso puede entrar a la sección crítica a la vez).

## ■ Idea clave:

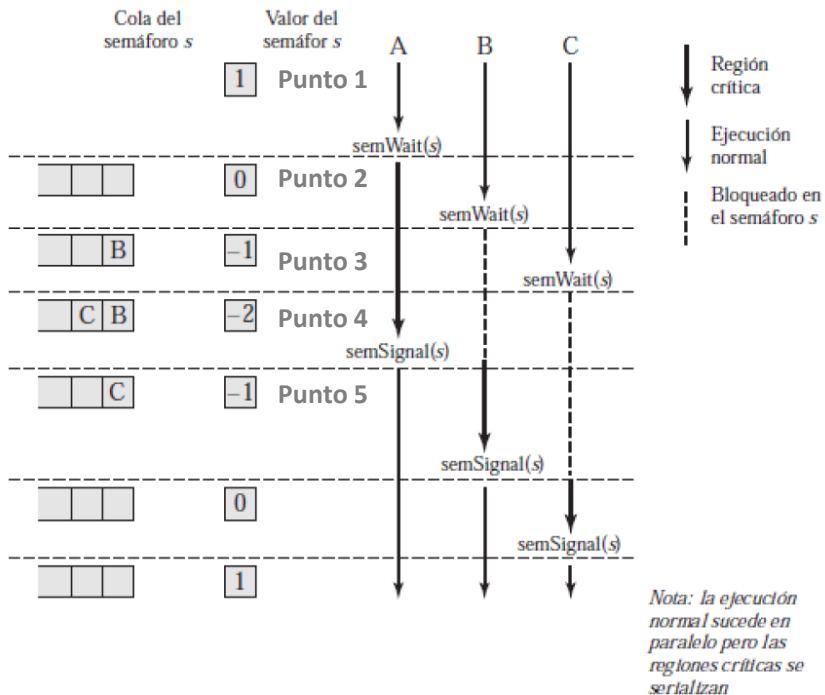
- ✓ semWait (P): resta 1 al semáforo → si el valor es negativo, el proceso se bloquea.
- ✓ semSignal (V): suma 1 al semáforo → si el valor  $\leq 0$ , desbloquea a un proceso de la cola.
- ✓ Garantía: solo un proceso a la vez accede a la sección crítica → se evita la condición de carrera.



# Exclusión mutua – Soporte por Hardware – Semáforos

## Ejemplo de un semáforo

Tres procesos A, B y C acceden a un recurso compartido protegido por el semáforo s

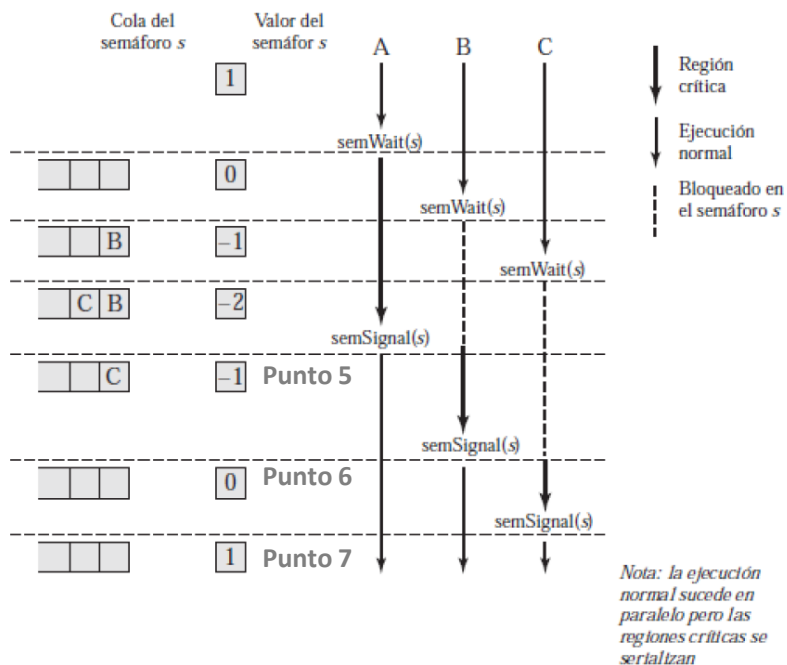


- 1. Inicio:  
 $s = 1 \rightarrow$  recurso libre.  
La cola del semáforo está vacía.
- 2. Proceso A hace `semWait(s)`:  
 $s$  pasa a 0  $\rightarrow$  A entra a la sección crítica.  
Cola vacía, porque no hay procesos esperando todavía.
- 3. Proceso B ingresa con `semWait(s)`:  
 $s$  se decrementa a -1.  
Como el valor es negativo, B se bloquea y pasa a la cola del semáforo.
- 4. Proceso C también hace `semWait(s)`:  
 $s$  se decrementa a -2.  
C también se bloquea y se encola detrás de B.
- 5. Proceso A termina su trabajo y hace `semSignal(s)`:  
 $s$  aumenta a -1.  
Se desbloquea el primer proceso en la cola (B)  $\rightarrow$  B pasa a la sección crítica.  
C sigue en espera.

# Exclusión mutua – Soporte por Hardware – Semáforos

## Ejemplo de un semáforo

Tres procesos A, B y C acceden a un recurso compartido protegido por el semáforo s



- 5. Proceso A termina su trabajo y hace semSignal(s):  
s aumenta a -1.  
Se desbloquea el primer proceso en la cola (B) → B pasa a la sección crítica.  
C sigue en espera.
- 6. Cuando B hace semSignal(s):  
s sube a 0.  
Se desbloquea C → ahora C entra a la sección crítica.
- 7. Finalmente, C hace semSignal(s):  
s vuelve a 1.  
Cola vacía → el recurso queda nuevamente libre.



# ¿Preguntas ?



## Clase 5

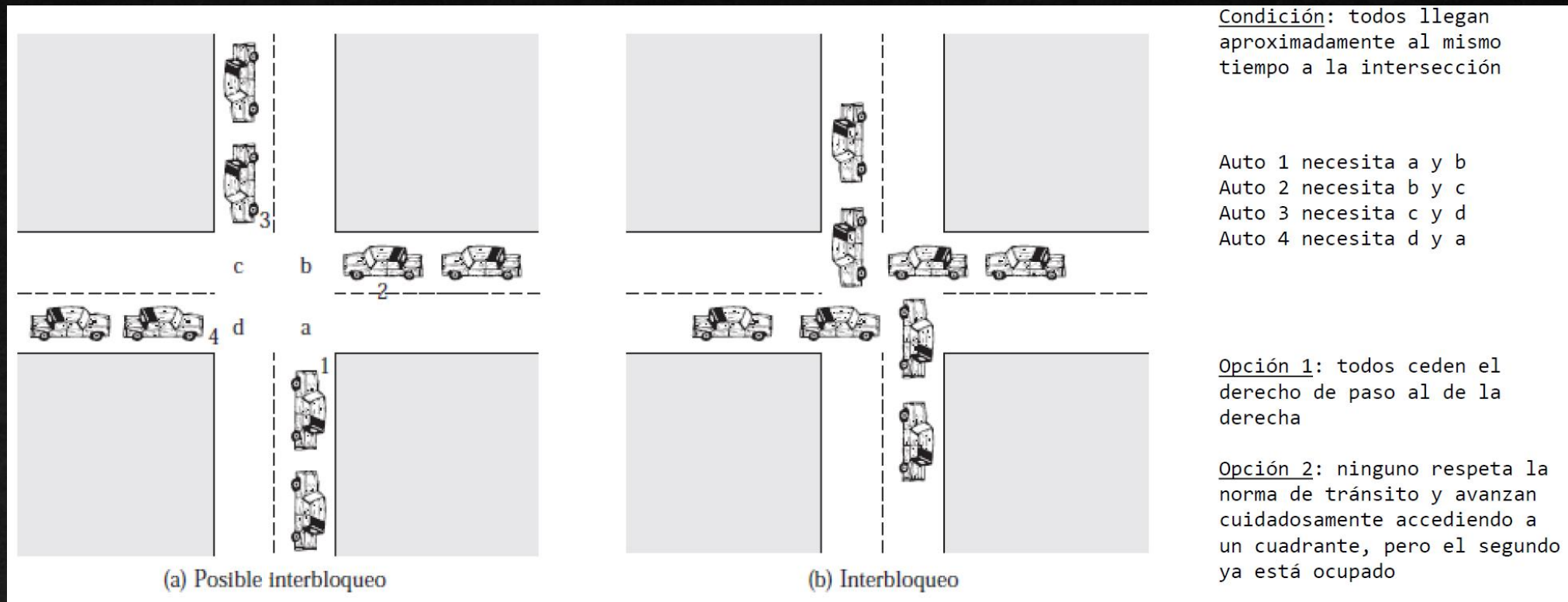
- Definiciones y condiciones para el interbloqueo
- Prevención del interbloqueo
- Predicción del interbloqueo
- Detección del interbloqueo

# Fundamentos de Interbloqueo

# Definiciones y condiciones para el interbloqueo

## ■ Interbloqueo = Deadlock

- ✓ También llamado Deadlock, es el bloqueo permanente de un conjunto de procesos que compiten por recursos del sistema o se comunican entre sí.



# Interbloqueo – Tipos de recursos – Reutilizables y Consumibles

## ■ ¿Por qué ocurre el interbloqueo?

- ✓ La ocurrencia o aparición de un **deadlock** depende tanto de la dinámica de ejecución como de los detalles de implementación de los programas.
- ✓ Todos los interbloqueos involucran necesidades (recursos) que afectan a 2 o más procesos.

### Reutilizables

- Sólo lo puede utilizar **de forma segura** un proceso en cada momento
- No se destruye con su uso
- Son el procesador, disco, memoria, archivos, etc.
- Existe un deadlock si cada proceso retiene un recurso y solicita otro

### Consumibles

- Puede crearse (producirse) y eliminarse (consumirse) infinitas veces
- Se destruye con su uso
- Son Interrupciones, señales, mensajes, etc.
- Existe deadlock si el proceso se bloquea a la espera de un mensaje.
- Otras causas involucran combinaciones de eventos

# Tipos de recursos – Reutilizables

Memoria disponible: 200 Kbytes

Proceso A	Proceso B	
solicita 80 Kbytes	solicita 70 Kbytes	Memoria libre 50 Kbytes
•	•	
•	•	
•	•	
solicita otros 60 Kbytes	solicita otros 80 Kbytes	Interbloqueo

## ■ Caso 1 – Recurso Reutilizable

- ✓ Qué muestra la imagen: Dos procesos se bloquean porque cada uno retiene un recurso distinto (ej., memoria, archivo) y están esperando el recurso que el otro tiene.
  - ✓ Por qué se produce el interbloqueo: Ambos procesos ya tienen un recurso asignado. Cada uno necesita un segundo recurso que está en poder del otro. Ninguno puede liberar lo que tiene hasta obtener lo que le falta → se genera una espera circular.
- 👉 Resultado: ninguno de los procesos avanza, todos quedan bloqueados indefinidamente.

# Tipos de recursos – Consumibles

Proceso A	Proceso B
receive(mensaje_de_B)	receive(mensaje_de_A)
.	.
send(B, mensaje)	send(A, mensaje)

## ■ Caso 2 – Recurso Consumible

- ✓ Qué muestra la imagen: Dos procesos se comunican a través de mensajes. El Proceso A espera un mensaje de B. El Proceso B espera un mensaje de A.
- ✓ Por qué se produce el interbloqueo: Como los mensajes son recursos consumibles. A espera algo que B no envía, y B espera algo que A no envía, interbloqueo.
- 👉 Resultado: ambos procesos quedan en espera permanente, ya que el recurso nunca aparecerá.

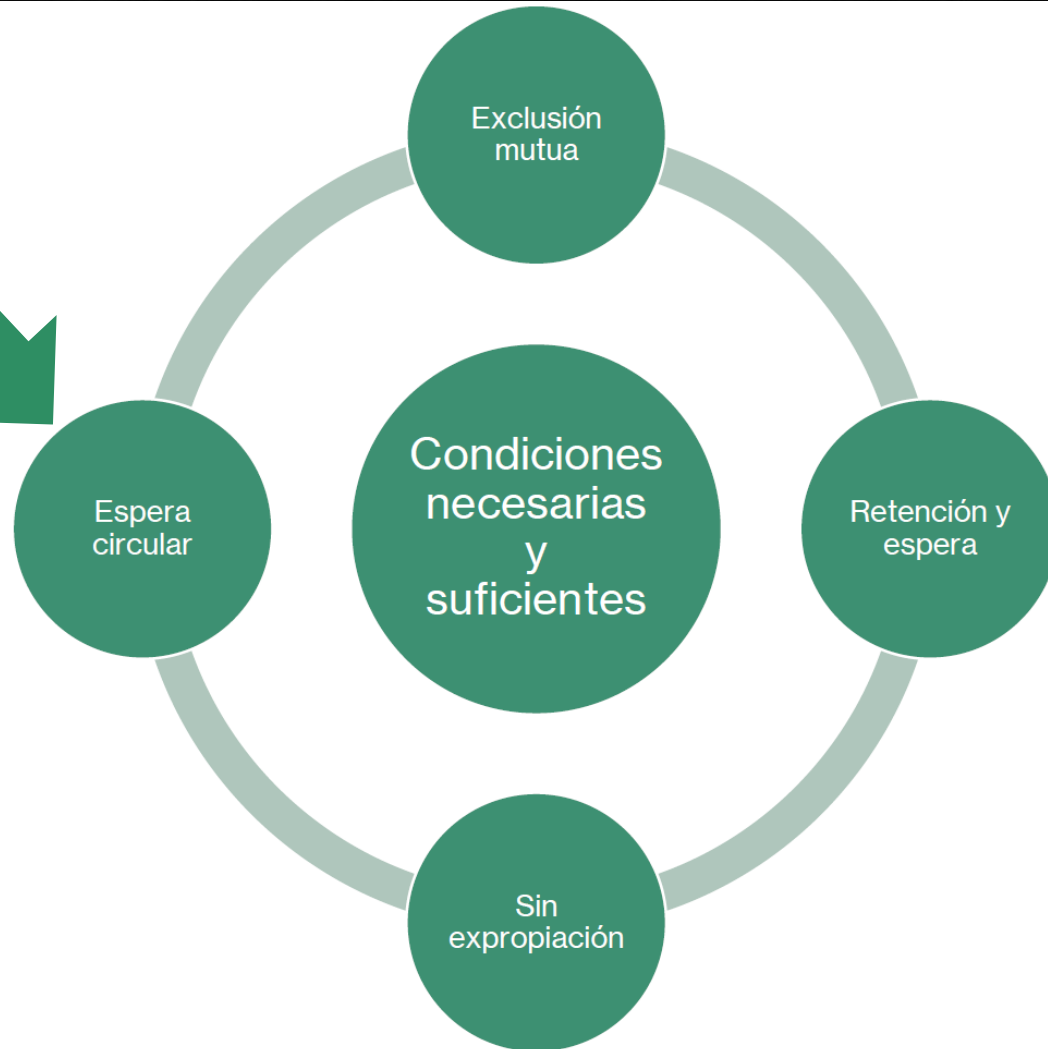
# Condiciones de Interbloqueo – Posibilidad



El hecho de que se cumplan estas 3 condiciones (al mismo tiempo), no asegura que exista interbloqueo entre procesos.

**Se dice que existe la posibilidad de que ocurra un interbloqueo**

# Condiciones necesarias y suficientes - INTERBLOQUEO

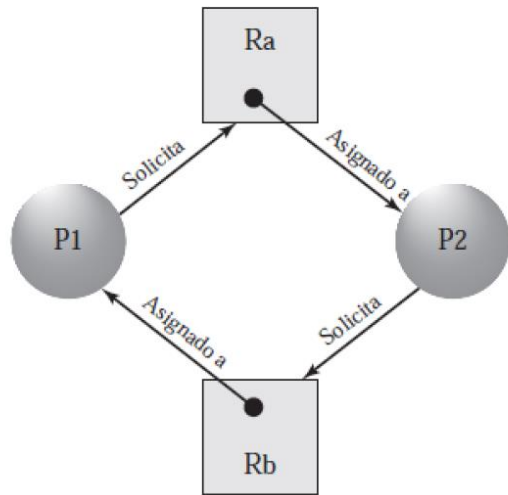


## Espera circular

Existe una lista "cerrada" de procesos, de forma tal que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.

Una **espera circular irresoluble** es lo que se denomina interbloqueo

# Espera Circular – 2 Procesos y 2 Recursos



R: Recurso

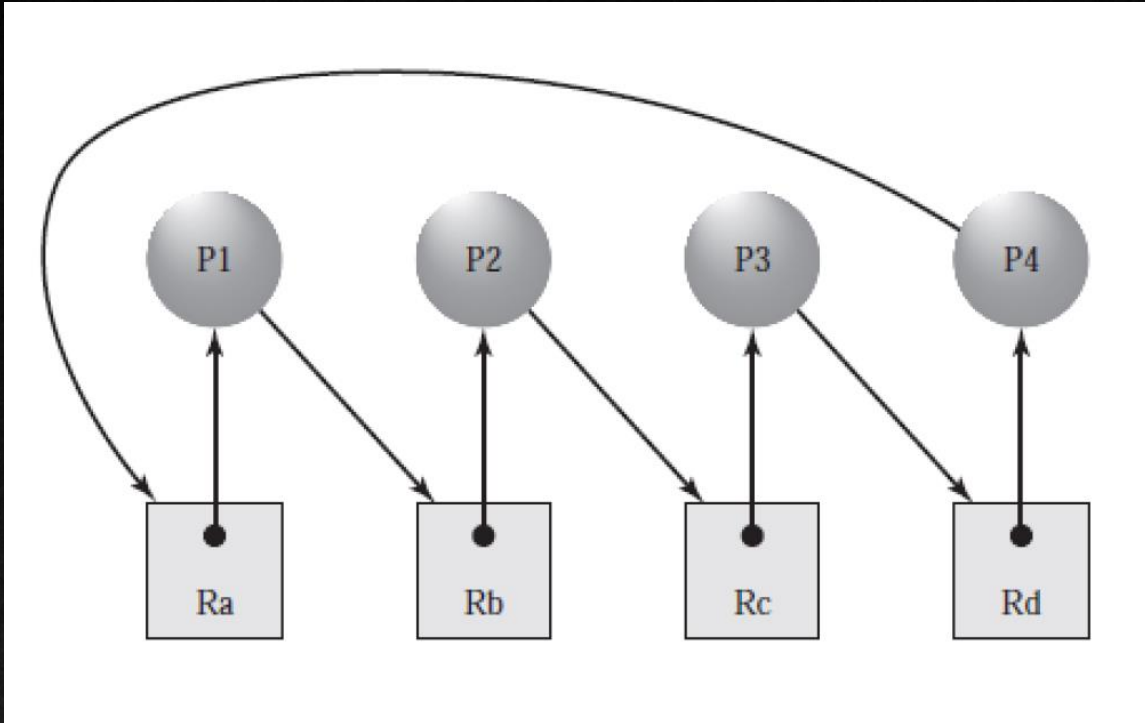
P: Proceso

## ■ Situación:

- ✓ El proceso P1 tiene asignado el recurso Rb y solicita el recurso Ra.
- ✓ El proceso P2 tiene asignado el recurso Ra y solicita el recurso Rb.
- ✓ Problema: Ninguno puede continuar porque cada uno está esperando un recurso que está en uso por el otro.
- ✓ Esto genera una espera circular de dos procesos.

👉 Conclusión: Es un interbloqueo clásico, ya que no hay forma de que se liberen los recursos sin que alguno de los procesos quede permanentemente bloqueado.

# Espera Circular – 4 Procesos y 4 Recursos



## ■ Situación:

- ✓ El proceso P1 tiene asignado Ra y solicita Rb.
- ✓ El proceso P2 tiene asignado Rb y solicita Rc.
- ✓ El proceso P3 tiene asignado Rc y solicita Rd.
- ✓ El proceso P4 tiene asignado Rd y solicita Ra.
- ✓ Problema: Cada proceso está esperando un recurso que posee otro proceso de la “cadena cerrada”. Se forma un ciclo circular de espera de recursos entre los cuatro procesos.
- ✓ 🖐 Conclusión: Este escenario extiende el mismo concepto de la imagen anterior, pero con más procesos involucrados. La espera circular sigue siendo irresoluble sin intervención externa, y por lo tanto constituye un interbloqueo.

# Prevención del interbloqueo – Metodos

**Indirectos:** evitar alguna de las 3 condiciones necesarias

Exclusión mutua: no puede eliminarse. El SO debe proporcionarla

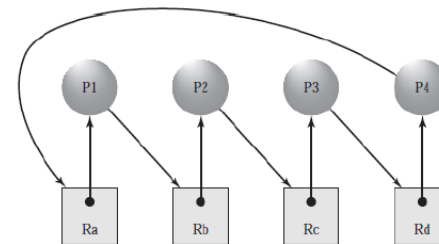
Retención y espera: solicitar todos los recursos al mismo tiempo

Sin expropiación: el proceso debe liberar los recursos y solicitarlos otra vez. El SO debe ser capaz de quitar los recursos a un proceso

**Directos:** evitar la espera circular

Se define una jerarquía de recursos

Si un proceso solicita un recurso de tipo A, luego sólo podrá solicitar un recurso de tipo B



# Predicción del interbloqueo – Estrategia - Acciones

- Estrategia de predicción
  - ✓ **Decidir dinámicamente** si la petición de un recurso podrá causar un interbloqueo.
  - ✓ La predicción del interbloqueo requiere el conocimiento de las futuras solicitudes de recursos del proceso.
- Acciones (dos posibilidades)
  - ✓ **Denegar** el inicio de un proceso si la solicitud de recursos puede conducir a un **deadlock**.
  - ✓ **No asignar** un recurso a un proceso si esta asignación puede conducir a un deadlock (algoritmo del banquero).



# Denegación del inicio del proceso

- $R_j$ : Recursos totales del sistema
- $D_j$ : Recursos disponibles del sistema
- $N_{ij}$ : necesidades del proceso  $i$  con respecto al recurso  $j$
- $A_{ij}$ : asignación actual al proceso  $i$  con respecto al recurso  $j$

1.  $R_j = D_j + \sum_{i=1}^n A_{ij}$ , para todo  $j$

Todos los recursos están disponibles o asignados.

2.  $N_{ij} \leq R_j$ , para todo  $i, j$

Ningún proceso puede necesitar más de la cantidad total de recursos existentes en el sistema.

3.  $A_{ij} \leq N_{ij}$ , para todo  $i, j$

Ningún proceso tiene asignados más recursos de cualquier tipo que sus necesidades originales de ese recurso.

Se inicia un nuevo proceso  $P_{n+1}$  sólo si:

$$R_j \geq N_{(n+1)j} + \sum_{i=1}^n N_{ij} \quad \text{para todo } j$$

# Algoritmo del banquero – 2 Estados : Seguro / No Seguro

- El sistema puede estar en 2 estados: SEGURO y NO SEGURO
- El estado del sistema está representado por la actual asignación de recursos.
- Un estado **SEGURO** es aquel donde existe al menos una secuencia de asignación que **no conduce a un deadlock (interbloqueo)**.
- Elementos:
  - ✓ Vector de recursos totales
  - ✓ Vector de recursos disponibles
  - ✓ Matriz de solicitudes (necesidad de recursos de cada proceso)
  - ✓ Matriz de asignación



# Algoritmo del banquero – Ejemplo

## Ejemplo 1 (a)

$$N_{ij} - A_{ij} \leq D_j \text{ para todo } j$$

Estado inicial

R: cant. de recursos totales

$R = \{9, 3, 6\}$   
 $A = \{9, 2, 5\}$   
 $D = \{0, 1, 1\}$

Disponibles:  
 $D = R - A$

Recursos

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	6	1	3	6	1	2	0	0	1
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0
	16	6	11	9	2	5			

$A = \{9, 2, 5\}$

Procesos

- ✓ Tenemos R totales= {9, 3, 6}, es decir:
- ✓ 9 instancias de R1, 3 de R2 y 6 de R3.
- ✓ La asignación actual a los procesos consume parte de esos recursos.
- ✓ Los recursos disponibles (D) se calculan como:
- ✓  $D = R - \text{Asignación total} = \{0, 1, 1\}$
- ✓ En la tabla de la derecha aparece la necesidad restante de cada proceso.
- ✓ Este es el punto de partida: vemos qué procesos pueden ejecutarse con lo que hay disponible.

👉 ATENCION:

Para el Proceso P2,  $N - A \leq \{0, 0, 1\}$

# Algoritmo del banquero – Ejemplo

## Ejemplo 1 (b)

$$N_{ij} - A_{ij} \leq D_j \text{ para todo } j$$

Estado inicial

Necesidad

Asignación

Necesidad - Asignación

$R = \{9, 3, 6\}$

$D = \{0, 1, 1\}$

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

$$\{0, 1, 1\} + \{6, 1, 2\} = D = \{6, 2, 3\}$$

Le asigna el recurso R3 que necesita P2 para que este finalice y libere los recursos

- ✓ Con los recursos disponibles:  
 $D = \{0, 1, 1\}$ , vemos qué proceso puede terminar.
- ✓ P2 necesita  $\{0, 0, 1\}$ ,  
y como coincide con lo disponible,  
se le puede asignar R3.
- ✓ P2 finaliza, libera lo que tenía asignado  $\{6, 1, 2\}$  y estos recursos vuelven a la disponibilidad.
- ✓ Ahora el sistema queda en un estado seguro, porque hay un camino posible.

# Algoritmo del banquero – Ejemplo

## Ejemplo 1 (c)

Finalizó P2 y liberó los recursos que se asignan a P1

$R = \{9, 3, 6\}$

$D = \{6, 2, 3\}$

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

Le asigna los recursos que necesita P1 para que este finalice y los libere.

- ✓ Al terminar P2, los recursos disponibles pasan a  $D = \{6, 2, 3\}$ .
- ✓ Con esta disponibilidad, ahora P1 puede ejecutar: su necesidad restante es  $\{2, 2, 2\}$ , que sí se puede cubrir.
- ✓ El sistema decide asignarle esos recursos a P1 para que termine.
- ✓ Una vez que P1 finaliza, liberará sus asignaciones.
- ✓  $\{2, 2, 2\} \leq \{6, 2, 3\} = D$ .

$$N_{ij} - A_{ij} \leq D_j, \text{ para todo } j$$

# Algoritmo del banquero – Ejemplo

## Ejemplo 1 (d)

P3 ejecuta hasta completarse

$R = \{9, 3, 6\}$

$D = \{7, 2, 3\}$

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

Aquí tiene la alternativa de elegir primero P3 y luego P4, o viceversa.

- ✓ Después de P1, tenemos más recursos disponibles.
- ✓ Ahora el sistema tiene dos opciones:
  - Ejecutar P3 primero, o
  - Ejecutar P4 primero.
- ✓ En ambos casos el estado sigue siendo seguro porque los recursos alcanzan.
- ✓ Esta flexibilidad muestra que no estamos en riesgo de interbloqueo, modo seguro.
- ✓  $\{1, 0, 3\} \leq \{7, 2, 3\} = D$ .

$$N_{ij} - A_{ij} \leq D_j, \text{ para todo } j$$

# Algoritmo del banquero – Ejemplo

## Ejemplo 1 (e)

Luego de finalizar P3

$R = \{9, 3, 6\}$

$D = \{9, 3, 4\}$

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0
P4	4	2	2	0	0	2	4	2	0

- ✓ Al ejecutarse P3, libera sus recursos y deja  $D = \{9, 3, 4\}$ .
- ✓ Finalmente, P4 recibe lo que necesita, ejecuta y libera.
- ✓ Todos los procesos pudieron completarse, confirmando que la secuencia es segura.
- ✓  $\{4, 2, 0\} \leq \{9, 3, 4\} = D$ .

$$N_{ij} - A_{ij} \leq D_j \text{ para todo } j$$

# Algoritmo del banquero – Resumen

👉 En resumen:

- ✓ El algoritmo del banquero revisa, en cada paso, si algún proceso puede finalizar con los recursos disponibles.
- ✓ Si es posible, se le asignan, ejecuta y libera.
- ✓ Así, se evita caer en interbloqueo porque siempre se garantiza que exista al menos un camino seguro de ejecución.



# Detección del interbloqueo

- Algoritmo de detección del interbloqueo
  - ✓ La **comprobación** de si hay interbloqueo se puede hacer una vez **por cada petición de recurso** o, con menos frecuencia, dependiendo de la probabilidad de que **ocurra un interbloqueo**.
  - ✓ Comprobación por cada petición de recurso:
    - Ventaja: detección temprana y el algoritmo es relativamente sencillo debido a que está basado en cambios graduales del estado del sistema.
    - Desventaja: estas comprobaciones frecuentes consumen un considerable tiempo del procesador.

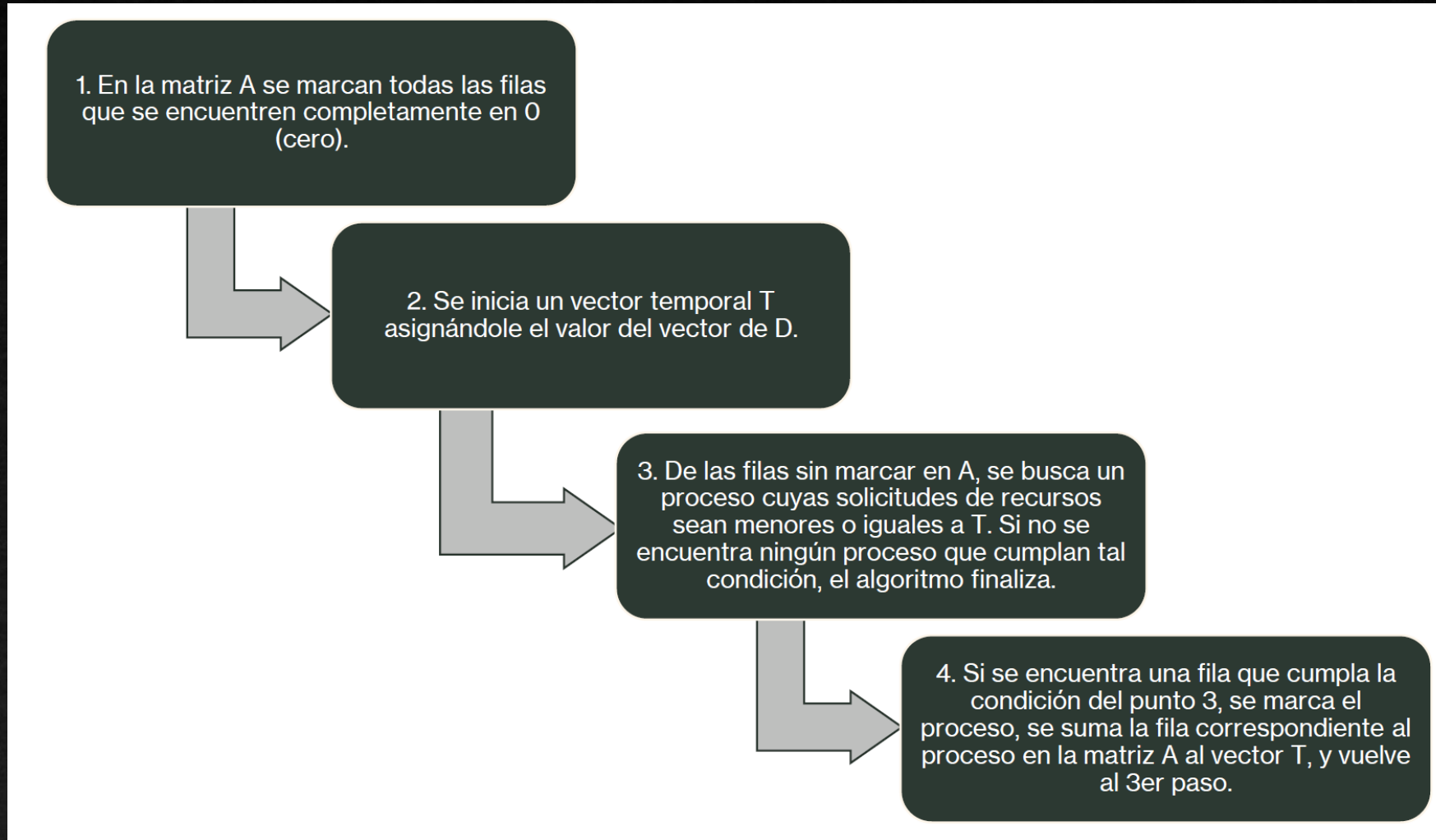
```
(typeOfFID == "BLANK#") string4replac
umOfdot == -1): value = fl
e("czFieldID",str(key)) tempString
e("czData",str(int(value*pow(10,14-
e + tempString elif(typeOfFID == "
fer tempString = tempString.replace
e + tempString elif(typeOfFID == "
e("czDataType","Buffer") tempString
line in searchlines: if "<Name valu
= searchObj1.group(1) if "</Messag
cName+"\t"+opaqueV+"\t"+onlyFilenam
= "" opaqueV = "" if not os.path.e
shutil if os.path.exists("Input4RTA
) for line in content: searchObj =
] = searchObj.group(2) for filename
(\w+)\.', str(fName), re.M|re.I) if
```

# Detección del interbloqueo - Ejemplo

Estado inicial

		Solicitudes					Asignación				
		R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
$R = \{2, 1, 1, 2, 1\}$ $D = \{0, 0, 0, 0, 1\}$	P1	0	1	0	0	1	1	0	1	1	0
	P2	0	0	1	0	1	1	1	0	0	0
	P3	0	0	0	0	1	0	0	0	1	0
	P4	1	0	1	0	1	0	0	0	0	0

# Algoritmo de detección



# Aplicación del algoritmo de detección en el ejemplo

1. Marca en la matriz A, a P4 porque no tiene recursos asignados
2. Establece  $T = \{0, 0, 0, 0, 1\}$
3. Recorre S buscando una fila que sea menor o igual a T. El proceso P3 satisface dicha condición y lo marca.
4. Actualiza T, sumando el valor de la fila de Asignaciones correspondiente al índice de la fila encontrada en S. En este caso  $D = \{0, 0, 0, 0, 1\} + \{0, 0, 0, 1, 0\} \Rightarrow D = \{0, 0, 0, 1, 1\}$
5. Ya ningún proceso cumple con la condición del punto 3, con lo cual el algoritmo concluye sin marcar P1 y P2, indicando que estos procesos están en un interbloqueo.

Solicitudes						Asignación					
						R1	R2	R3	R4	R5	
$R = \{2, 1, 1, 2, 1\}$	P1	0	1	0	0	1	1	0	1	0	
	P2	0	0	1	0	1	1	0	0	0	
$D = \{0, 0, 0, 0, 1\}$	P3	0	0	0	0	1	0	0	0	1	
	P4	1	0	1	0	1	0	0	0	0	

$D = \{0, 0, 0, 1, 1\}$

## ■ Situación:

- ✓ El vector de recursos disponibles es  $D = \{0, 0, 0, 0, 1\}$ .
- ✓ Se marca inicialmente a P4, porque no tiene recursos asignados.
- ✓ Luego se evalúa P3, que solicita  $\{0, 0, 0, 0, 1\}$ , lo cual sí puede ser satisfecho con los recursos disponibles.
- ✓ Por eso, se marca P3, se liberan sus asignaciones  $\{0, 0, 0, 1, 0\}$  y D pasa a ser  $\{0, 0, 0, 1, 1\}$ .

# Aplicación del algoritmo de detección en el ejemplo

1. Marca en la matriz A, a P4 porque no tiene recursos asignados
2. Establece  $T = \{0, 0, 0, 0, 1\}$
3. Recorre S buscando una fila que sea menor o igual a T. El proceso P3 satisface dicha condición y lo marca.
4. Actualiza T, sumando el valor de la fila de Asignaciones correspondiente al índice de la fila encontrada en S. En este caso  $D = \{0, 0, 0, 0, 1\} + \{0, 0, 0, 1, 0\} \Rightarrow D = \{0, 0, 0, 1, 1\}$
5. Ya ningún proceso cumple con la condición del punto 3, con lo cual el algoritmo concluye sin marcar P1 y P2, indicando que estos procesos están en un interbloqueo.

Solicitudes						Asignación					
	R1	R2	R3	R4	R5		R1	R2	R3	R4	R5
$R = \{2, 1, 1, 2, 1\}$	P1	0	1	0	0	1	1	0	1	1	0
$D = \{0, 0, 0, 1, 1\}$	P2	0	0	1	0	1	1	1	0	0	0
	P3	0	0	0	0	1	0	0	0	1	0
	P4	1	0	1	0	1	0	0	0	0	0

$D = \{0, 0, 0, 1, 1\}$

- ¿Qué ocurre con P1 y P2?
- ✓ P1 solicita  $\{0, 1, 0, 0, 1\} \rightarrow$  necesita 1 unidad de R2, pero  $D = \{0, 0, 0, 1, 1\}$ , es decir no hay disponibilidad de R2.
- ✓ P2 solicita  $\{0, 0, 1, 0, 1\} \rightarrow$  necesita 1 unidad de R3, pero en D tampoco hay disponibilidad de R3.
- ✓ Por lo tanto, ni P1 ni P2 pueden continuar, y el algoritmo de detección concluye que están en interbloqueo.

# Recuperación del interbloqueo

## Estrategias de recuperación

1. Abortar todos los procesos involucrados en el interbloqueo (solución mas adoptada por los SO).

✓ **Abortar** todos los procesos

2. Retroceder cada proceso en interbloqueo a algún punto de control previamente definido, y reiniciar todos los procesos.

✓ **Retroceder** cada proceso en interbloqueo

3. Abortar sucesivamente los procesos en el interbloqueo hasta que este deje de existir.

✓ **Abortar** sucesivamente

4. Expropiar sucesivamente los recursos hasta que el interbloqueo deje de existir. Un proceso al que se le ha expropiado un recurso debe retroceder a un punto anterior a la adquisición de ese recurso.

✓ **Expropiar** sucesivamente

Criterios de selección:

- la menor cantidad de tiempo de procesador consumida hasta ahora
- la menor cantidad de salida producida hasta ahora
- el mayor tiempo restante estimado
- el menor número total de recursos asignados hasta ahora
- la menor prioridad

# ¿Preguntas ?





Muchas gracias