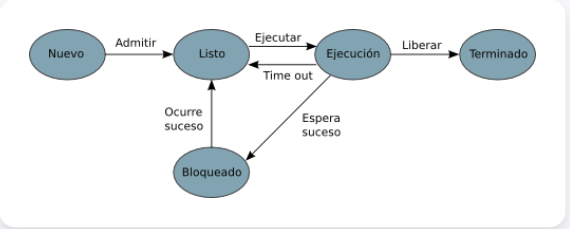
CAPÍTULO 3: ELEMENTOS DE UN SISTEMA OPERATIVO INFORMÁTICO

Gestión de Procesos

3.1 La gestión de procesos

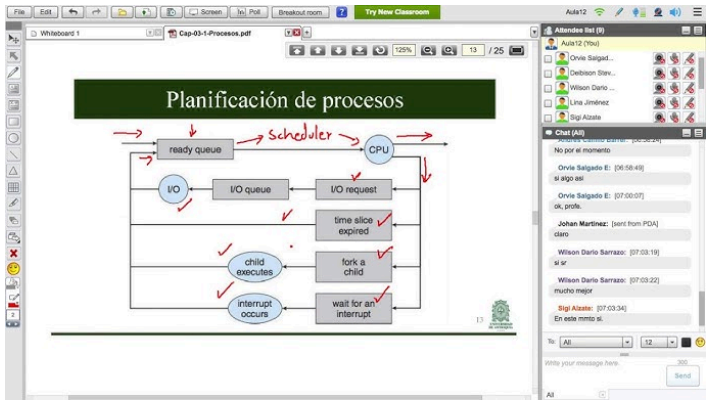
Índice de contenidos



3.1.1 La planificación de procesos

Concepto y definición

- Mecanismo para **gestionar asignación de CPU** entre procesos activos
- El **scheduler** implementa política de planificación



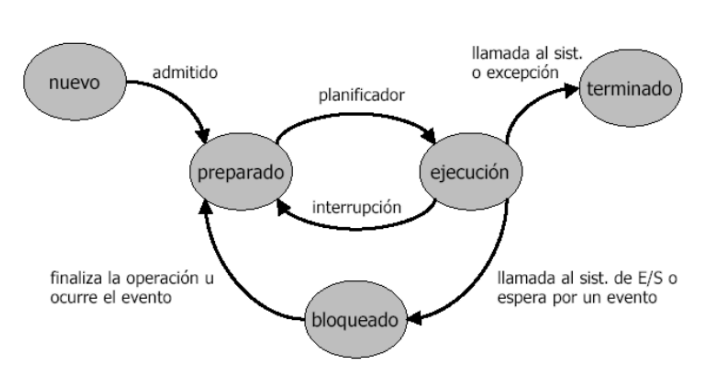
Objetivos de la planificación

- **Equidad**: Porción justa de CPU
- **Eficiencia**: Maximizar uso de CPU
- **Bajo tiempo de respuesta**: Procesos interactivos
- **Alto rendimiento**: Maximizar trabajo completado

Tipos de planificación

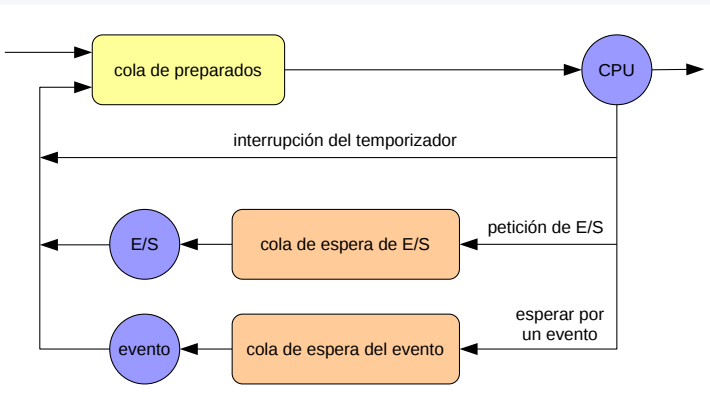
A largo plazo : Selecciona siguiente trabajo

A corto plazo : Asigna procesador a procesos preparados



Funcionamiento del despachador

- Decide si cambiar proceso activo según política
- Salva entorno volátil del proceso actual
- Toma primer proceso de cola de preparados
- Carga entorno volátil del proceso elegido (PCB)



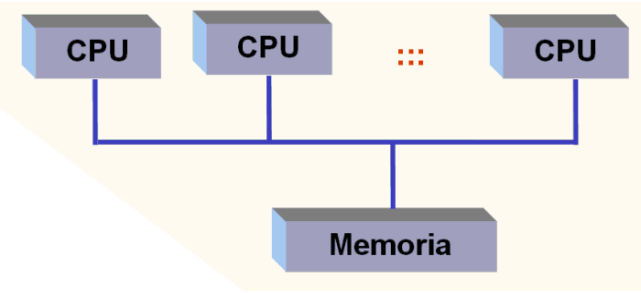
Criterios de eficiencia

- **Uso del tiempo de CPU**: Fracción de tiempo ocupada
- **Productividad**: Trabajo completado por unidad de tiempo
- **Tiempo de retorno**: Desde lanzamiento hasta finalización
- **Tiempo de espera**: Tiempo en estado de espera

3.1.2 Planificación de CPU-multiprocesamiento

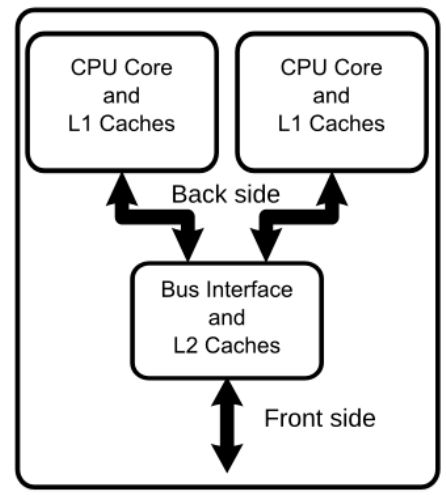
Sistemas multiprocesador

- Ordenadores con**varios procesadores**
- Capaces de ejecutar**múltiples tareas simultáneamente**
- A diferencia de sistemas convencionales (un solo procesador)



Arquitecturas de múltiples núcleos

- Procesadores modernos con**varios núcleos**
- Actúan como**procesadores independientes**
- Permiten**ejecución paralela**de procesos



Planificación en entornos multiprocesador

Balanceo de carga
Distribuir procesos entre procesadores

Sincronización
Coordinar acceso a recursos compartidos

Afinidad de procesos
Mantener proceso en mismo procesador

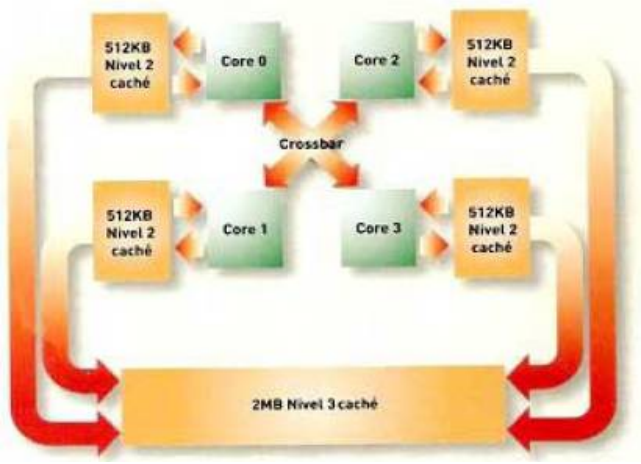
Ventajas y desafíos

Ventajas

- **Mayor rendimiento**global
- **Mayor fiabilidad**(tolerancia a fallos)
- **Más procesos**simultáneos

Desafíos

- **Complejidad**en gestión de memoria
- **Sincronización**más compleja
- **Condiciones de carrera** difíciles de detectar



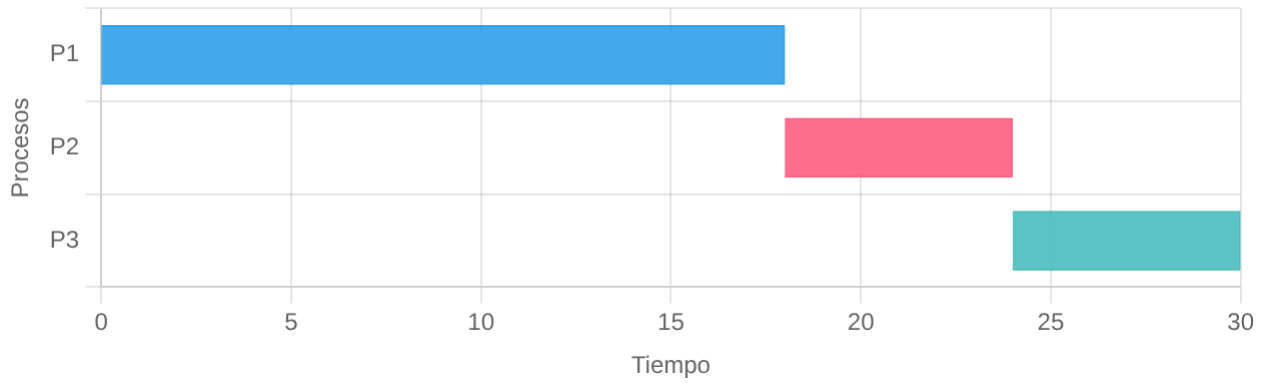
3.1.3 Algoritmo FIFO/PEPS

Concepto y Funcionamiento

- **FIFO**: First In First Out (Primero en Entrar, Primero en Salir)
- **PEPS**: Primero en Entrar, Primero en Salir
- Procesos ejecutados en **orden de llegada**
- CPU no se libera hasta que el proceso **termina**
- Implementación mediante **cola simple**

Ejemplo Práctico

Proceso	Tiempo de Llegada	Tiempo de Ejecución
P1	0	18
P2	0	6
P3	0	6



Tiempo medio de retorno: (18 + 24 + 30) / 3 = 24

Ventajas y Desventajas

Ventajas

- ✓ Implementación simple
- ✓ Justo para procesos cortos
- ✓ Sin inanición

Desventajas

- ✗ Tiempo de espera alto
- ✗ No óptimo
- ✗ Procesos largos bloquean

Características Clave

- **No expropiativo**: No interrumpe procesos en ejecución
- **Tiempo de respuesta variable** según orden de llegada
- **Fórmula tiempo de retorno**: Tiempo final - Tiempo llegada
- **Fórmula tiempo de espera**: Tiempo retorno - Tiempo ejecución
- **Eficiencia**: Depende del orden de los procesos

Aplicaciones

- Sistemas con **carga predecible**
- Sistemas de **colas de impresión**
- Gestión de **tráfico de red**
- Procesamiento por **lotes (batch)**

3.1.3 Algoritmo SJF (Shortest Job First)

Concepto y Funcionamiento

- **SJF**: Shortest Job First (Trabajo más Corto Primero)
- Asigna CPU al proceso con **menor tiempo de ejecución**
- **Óptimo** para minimizar tiempo medio de espera
- Requiere **estimación** del tiempo de ejecución

Variantes

No Expropiativo

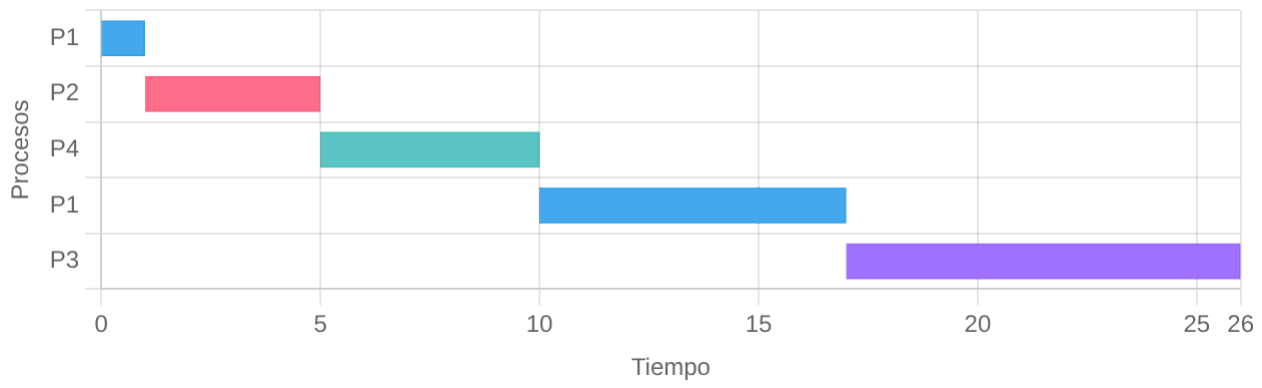
- Proceso mantiene CPU hasta **terminar**
- Menor sobrecarga del sistema
- Puede causar mayor tiempo de espera

Expropiativo

- Si llega proceso **más corto**, se interrumpe actual
- Mejor tiempo de respuesta
- Mayor sobrecarga por cambios de contexto

Ejemplo Práctico

Proceso	Tiempo de Llegada	Tiempo de Ejecución
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Tiempo medio de espera: (0 + 7 + 11 + 5) / 4 = 5.75

Ventajas y Desventajas

Ventajas

- ✓ Mínimo tiempo medio de espera
- ✓ Alto rendimiento
- ✓ Óptimo teóricamente

Desventajas

- ✗ Inanición de procesos largos
- ✗ Requiere predecir tiempos
- ✗ Complejidad de implementación

Características Clave

- **Estimación de tiempos**: Basada en historial
- **Envejecimiento**: Técnica para evitar inanición
- **Eficiencia**: Mejor que FIFO en la mayoría de casos
- **Predictibilidad**: Difícil en entornos dinámicos

Aplicaciones

- Sistemas de **impresión** (trabajos cortos primero)
- Procesamiento por **lotes** con tiempos conocidos
- Sistemas **empotrados** con tareas predecibles
- Planificación de **trabajos batch** en mainframes

3.1.3 Algoritmo Round Robin

Concepto y Funcionamiento

- Planificación por **rondas** o turnos
- Cada proceso tiene un **quantum** de tiempo asignado
- Si no termina, pasa al **final de la cola**
- Implementado mediante una **cola circular**

Quantum

Definición

- Unidad fija de **tiempo de CPU**
- Típicamente entre **10-100ms**

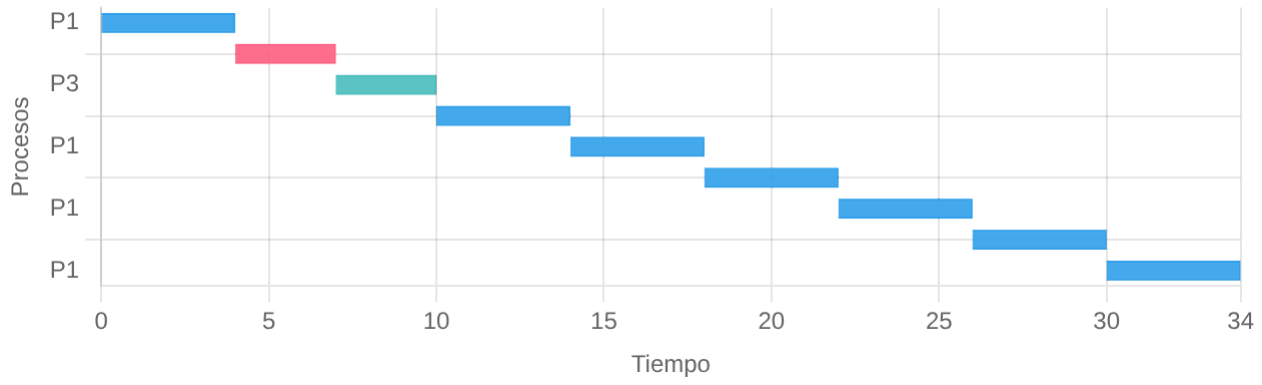
Elección adecuada

- **Quantum pequeño**: Mayor sobrecarga
- **Quantum grande**: Se aproxima a FIFO
- Ideal: **80% de procesos** terminan antes de quantum

Ejemplo Práctico

Proceso	Tiempo de Ejecución
P1	24
P2	3
P3	3

Quantum = 4 unidades de tiempo



Tiempo medio de espera: (7 + 4 + 7) / 3 = 6

Ventajas y Desventajas

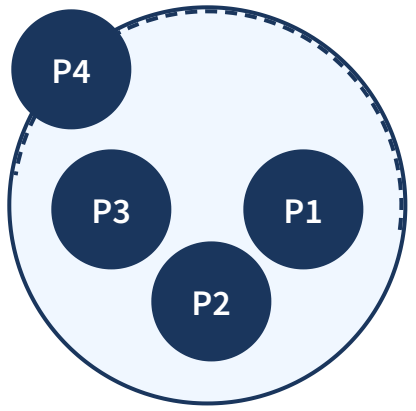
Ventajas

- ✓ Equitativo
- ✓ Sin inanición
- ✓ Respuesta predecible

Desventajas

- ✗ Sobrecarga por cambios
- ✗ Rendimiento menor que SJF
- ✗ Depende del quantum

Funcionamiento



- **Cambio de contexto** cuando expira quantum
- Proceso vuelve al **final de la cola**
- Proceso **termina** cuando completa su tiempo

Aplicaciones

- Sistemas **interactivos** (tiempo compartido)
- Servidores con **múltiples usuarios**
- Sistemas **operativos modernos**
- Planificación de **procesos en red**

3.1.3 Planificación por prioridades

Concepto y Funcionamiento

- Asigna **prioridad** a cada proceso
- CPU al proceso con **prioridad más alta**
- Puede ser **expropiativo** o no
- Implementado mediante **colas multinivel**

Tipos de Prioridades

Estáticas

- Asignadas **antes de ejecución**
- No cambian durante la ejecución
- Basadas en características del proceso

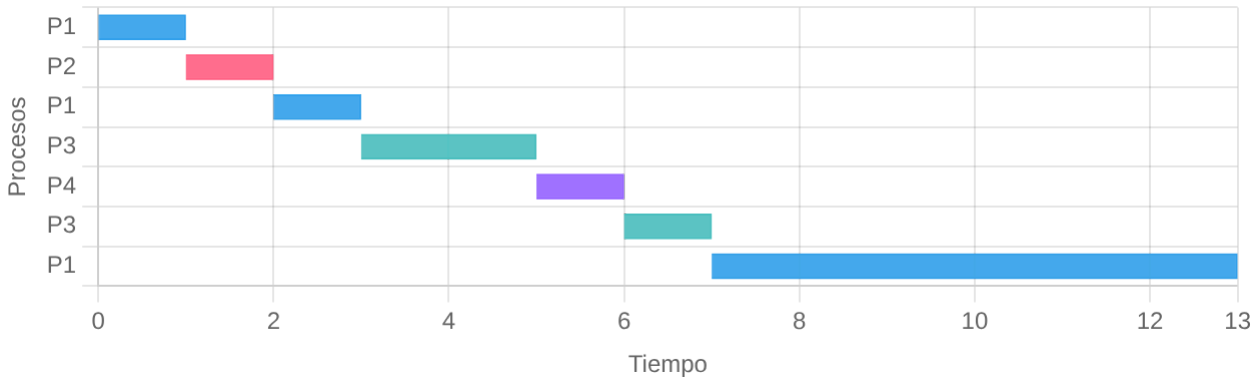
Dinámicas

- Pueden **cambiar durante ejecución**
- Se ajustan según comportamiento
- Evitan inanición mediante envejecimiento

Ejemplo Práctico

Proceso	Tiempo de Llegada	Tiempo de Ejecución	Prioridad
P1	0	10	3
P2	1	1	1
P3	2	2	4
P4	3	1	2

Expropiativo: mayor prioridad interrumpe



Funcionamiento

No Expropiativo

- Proceso mantiene CPU hasta **terminar**
- Solo se selecciona nuevo cuando termina actual
- Menor sobrecarga del sistema

Expropiativo

- Si llega proceso **mayor prioridad**, interrumpe
- Proceso actual pasa a estado listo
- Mayor tiempo de respuesta para críticos

Ventajas y Desventajas

Ventajas

- ✓ Control flexible
- ✓ Respuesta adecuada a críticos
- ✓ Combinable con otros algoritmos

Desventajas

- ✗ Inanición de baja prioridad
- ✗ Complejidad de implementación
- ✗ Difícil asignación óptima

Niveles de Prioridad

- **Alta**: Procesos críticos del sistema
- **Media**: Procesos interactivos
- **Baja**: Procesos en segundo plano
- **Envejecimiento**: Aumenta prioridad con tiempo de espera
- **Colas multinivel**: Cada nivel con su algoritmo

3.1.3 Algoritmos híbridos

Concepto y Funcionamiento

- Combinación de **varios algoritmos** de planificación
- Aprovecha **ventajas** de cada algoritmo
- Implementado mediante **colas multinivel**
- Cada nivel puede usar un **algoritmo diferente**

Estructura Típica

4 **Prioridad máxima**
Round Robin

3 **Alta prioridad**
Round Robin

2 **Media prioridad**
SJF

1 **Baja prioridad**
FIFO

Combinaciones Comunes

Prioridades + Round Robin = Híbrido 1

SJF + FIFO = Híbrido 2

Retroalimentación: Procesos pueden moverse entre colas

Adaptativo: Ajusta parámetros según carga

Ventajas y Desventajas

Ventajas	Desventajas
✓ Flexibilidad	✗ Complejidad
✓ Mejor rendimiento	✗ Sobrecarga
✓ Adaptabilidad	✗ Difícil configuración
✓ Evita inanición	✗ Coste implementación

Características Clave

- **Colas multinivel** con retroalimentación
- **Migración** de procesos entre colas
- **Parámetros ajustables** por nivel
- **Monitoreo** para optimización
- **Envejecimiento** para evitar inanición

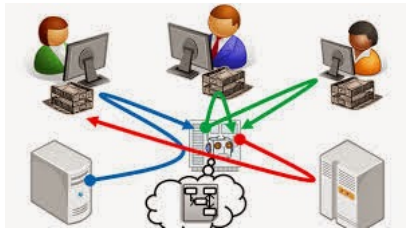
Aplicaciones

- **Sistemas operativos modernos**
- **Servidores** con cargas variables
- **Sistemas empotrados** de tiempo real
- **Entornos virtualizados**

3.1.4 Sincronización de procesos

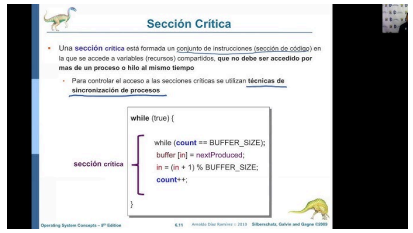
Procesos independientes vs cooperantes

Independientes	Cooperantes
Ejecución sin interferencias	Comparten estado
Deterministas	No deterministas
Reproducibles	Irreproducibles
Ej: Cálculo de primos	Ej: Escritura en pantalla



Sección crítica

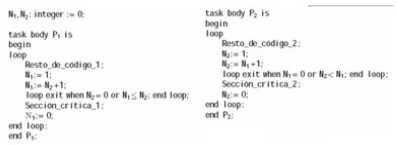
- Sección de código con **operaciones críticas**
- Solo **un proceso** puede ejecutarla
- Debe ejecutarse en **exclusión mutua**
- Un proceso no puede dormirse en ella



Ejemplo práctico

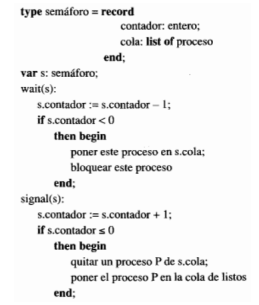
Problema de la nevera

Persona A	Persona B
09:50 Llega a casa	10:10 Llega a casa
10:00 Mira nevera: no hay leche	10:20 Mira nevera: no hay leche
10:10 Va a la tienda	10:30 Va a la tienda
10:40 Llega con leche	11:00 Llega y ¡ya hay leche!



Mecanismos de sincronización

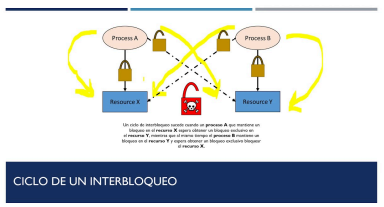
- **Semáforos**: Variables enteras para control de acceso
- **Monitores**: Estructuras de alto nivel con operaciones
- **Mensajes**: Comunicación entre procesos
- **Señales**: Notificaciones entre procesos



3.1.5 Bloqueos

Definición y causas

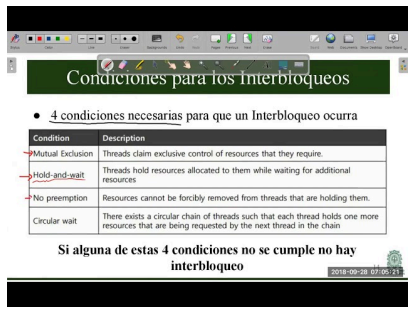
- **Deadlock**o "abrazo mortal"
- Dos o más procesos **esperando recursos**
- Cada proceso **otro** recursos espera que **libere**
- Ciclo de espera**permanente**



Condiciones necesarias

Para que ocurra un deadlock:

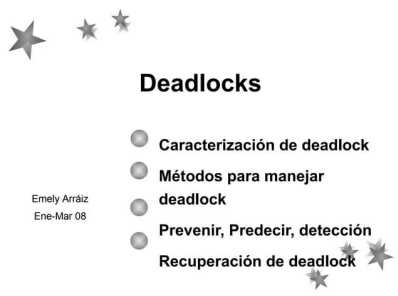
1. **Exclusión mutua** : Recursos no compartibles
2. **Retención y espera**: Procesos mantienen recursos mientras solicitan otros
3. **No expulsión** : Recursos no pueden ser retirados forzosamente
4. **Espera circular** : Cadena circular de procesos esperando recursos



Estrategias de manejo

- **Prevención**
- **Evitación**
- **Detección**
- **Recuperación**

- **Prevención**: Evitar una o más condiciones necesarias
- **Evitación**: Algoritmo del banquero de Dijkstra
- **Ignorar**: Considerar bloqueos como eventos raros



Detección y recuperación

- **Detección**:
  - Grafo de asignación de recursos
  - Búsqueda de ciclos en el grafo
- **Recuperación**:
  - Terminar procesos involucrados
  - Preemptar recursos
  - Puntos de control y recuperación