Planificadores de Procesos en Sistemas Operativos

1. Introducción

Los **planificadores de procesos** (o schedulers) son componentes fundamentales del sistema operativo encargados de decidir qué proceso ejecuta la CPU en un momento dado. Su objetivo principal es maximizar la utilización del procesador mientras proporciona tiempos de respuesta razonables a los usuarios.

2. Conceptos Clave

2.1 Ciclo de Vida de un Proceso

Un proceso atraviesa varios estados durante su ejecución:

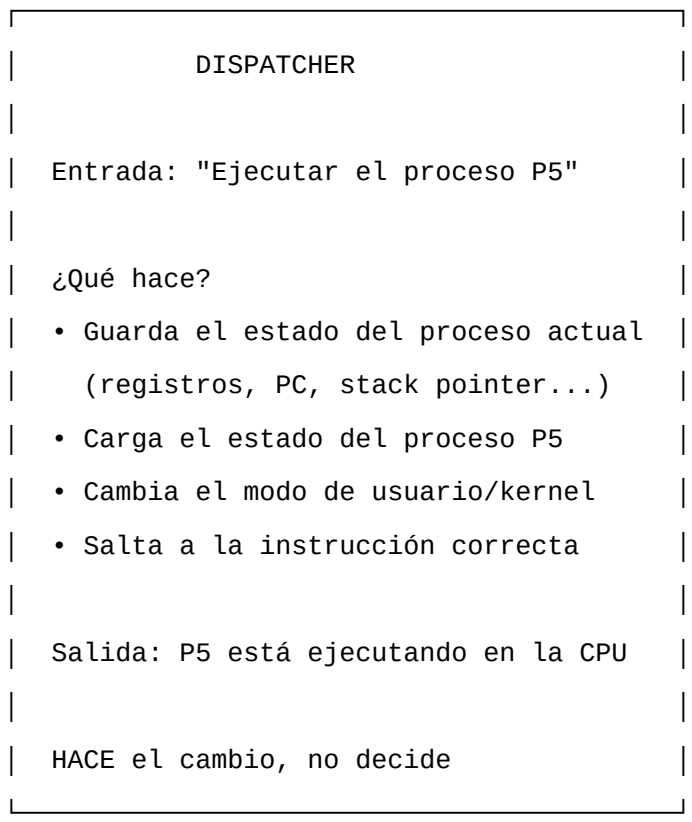
Estado	Descripción
Nuevo	El proceso está siendo creado
Listo	Esperando ser asignado a la CPU
Ejecutando	Instrucciones están siendo procesadas
Bloqueado	Esperando un evento (E/S, señal)
Terminado	Ha finalizado su ejecución

2.2 Tipos de Planificadores

1. **Planificador a Largo Plazo (Long-term scheduler / Job scheduler)**
    - Decide qué procesos se admiten al sistema
    - Controla el grado de multiprogramación
    - Se ejecuta con poca frecuencia
2. **Planificador a Mediano Plazo (Medium-term scheduler)**
    - Gestiona la suspensión y reanudación de procesos (swapping)
    - Reduce el grado de multiprogramación temporalmente
3. **Planificador a Corto Plazo (Short-term scheduler / CPU scheduler)**
    - Decide qué proceso de la cola de listos ejecutará la CPU
    - Se ejecuta muy frecuentemente (milisegundos)
    - Debe ser extremadamente rápido

2.3 Dispatcher (Despachador)

Función: Cambiar el contexto para que el proceso elegido realmente ejecute.
Es el **brazo** que ejecuta la acción. Realiza el cambio físico.



2.4 Apropiativo (Preemptive) vs. No Apropiativo (Non-preemptive)

No Apropiativo (Non-preemptive) — "Cooperativo"

Una vez que la CPU se asigna a un proceso, **este la conserva hasta que termina** o hasta que se bloquea voluntariamente (por ejemplo, esperando E/S).

El SO NO se apropia (no quita) la CPU; el proceso la conserva

Analogía: Es como una conversación donde una persona habla todo lo que quiere y solo cede la palabra cuando decide parar o cuando necesita algo externo.
Características:

- El proceso **no puede ser interrumpido** a la fuerza
- El cambio de contexto solo ocurre cuando el proceso termina o se bloquea
- Menor overhead (menos cambios de contexto)
- Riesgo: un proceso largo puede monopolizar la CPU

Ejemplos de algoritmos: FCFS, SJF no apropiativo

Apropiativo (Preemptive) — "A la fuerza"

El sistema operativo puede **quitarle la CPU a un proceso en ejecución** en cualquier momento para dársela a otro proceso, generalmente cuando expira un quantum de tiempo o llega un proceso de mayor prioridad.

El SO se apropia (quita) la CPU del proceso

Analogía: Es como una reunión con un moderador que puede interrumpir a alguien para darle el turno a otra persona, asegurando que todos participen.
Características:

- El SO puede forzar un cambio de contexto
- Mayor overhead (más cambios de contexto)
- Mejor tiempo de respuesta para procesos interactivos
- Evita que un solo proceso monopolice el sistema

Ejemplos de algoritmos: Round Robin, SRTF, Prioridad apropiativa, Multilevel Feedback Queue

Ejemplo No apropiativo (FCFS):

P1 llega → CPU para P1 → P1 ejecuta 10 minutos

→ P2 y P3 llegan en medio → ESPERAN

→ P1 termina → recién ahí P2 ejecuta

P1 "se quedó" con la CPU porque NADIE se la pudo quitar

Ejemplo Apropiativo (Round Robin):

P1 llega → CPU para P1 → P1 ejecuta 4ms
→ ¡TIM! El SO quita la CPU (se la apropia)
→ P2 ejecuta 4ms
→ ¡TIM! El SO quita la CPU
→ P3 ejecuta 4ms…
El SO se apropia constantemente de la CPU para repartirla

Comparación Visual

NO APROPIATIVO (FCFS):

P1 ████████████████████ (24ms) → P2 ████ (3ms) → P3 ████ (3ms)

↑ P1 no puede ser interrumpido, aunque P2 y P3 sean cortos

APROPIATIVO (Round Robin, quantum=4):

P1 ████ → P2 ███ → P3 ███ → P1 ████ → P1 ████ → P1 ████ → P1 ████

↑ cada proceso recibe su turno, luego vuelve a la cola

¿Cuándo se produce la apropiación?

Evento	Descripción
Expiración del quantum	En Round Robin, cuando se acaba el tiempo asignado
Llegada de proceso prioritario	En prioridad apropiativa, un proceso más importante llega
Interrupción de E/S completada	Un proceso bloqueado se vuelve listo y tiene más prioridad
Llamada al sistema	El SO decide que otro proceso debe ejecutarse

---

Resumen Rápido

Aspecto	No Apropiativo	Apropiativo
¿Quién decide cuándo soltar la CPU?	El proceso mismo	El sistema operativo
¿Puede ser interrumpido?	No	Sí
Overhead	Menor	Mayor
Tiempo de respuesta	Peor para interactivos	Mejor
Riesgo de monopolio	Alto	Bajo
Ejemplo real	FCFS	Round Robin, Windows, Linux

---

3. Criterios de Planificación

Criterio	Descripción
Utilización de CPU	Mantener la CPU ocupada el mayor tiempo posible
Throughput (Rendimiento)	Número de procesos completados por unidad de tiempo
Tiempo de Turnaround	Tiempo total desde que un proceso llega hasta que termina
Tiempo de Espera	Tiempo que un proceso pasa en la cola de listos
Tiempo de Respuesta	Tiempo desde que se hace una solicitud hasta la primera respuesta
Justicia (Fairness)	Todos los procesos deben recibir tratamiento equitativo

4. Algoritmos de Planificación

4.1 First-Come, First-Served (FCFS)

- **Descripción**: El primer proceso en llegar es el primero en ser atendido.
- **Tipo**: No apropiativo (non-preemptive)
- **Implementación**: Cola FIFO (First In, First Out)

Ventajas:

- Simple de implementar
- Sin inanición ó starvation (Es cuando un proceso nunca o casi nunca recibe la CPU porque siempre hay otros procesos con mayor prioridad.)

Desventajas:

- Efecto convoy: procesos cortos esperan detrás de procesos largos
- Tiempo de espera promedio puede ser alto

Ejemplo:

Proceso	Tiempo de Llegada	Tiempo de CPU
P1	0	24
P2	1	3
P3	2	3

Diagrama de Gantt:

P1 (0-24) | P2 (24-27) | P3 (27-30)

P1 ████████████ → P2 ██ → P3 ██

0 24 27 30

- Tiempo de espera: P1=0, P2=23, P3=25 → Promedio = 16
- Tiempo de turnaround: P1=24, P2=26, P3=28 → Promedio = 26

4.2 Shortest-Job-First (SJF)

- **Descripción**: El proceso con el tiempo de ejecución más corto se ejecuta primero.
- **Tipo**: Puede ser apropiativo (SRTF) o no apropiativo

Ventajas:

- Minimiza el tiempo de espera promedio
- Óptimo en teoría para minimizar el tiempo de espera

Desventajas:

- Difícil conocer el tiempo de ejecución real de antemano
- Puede causar inanición de procesos largos

Variante Apropiativa: Shortest-Remaining-Time-First (SRTF)

- Si llega un proceso con tiempo de ejecución menor al tiempo restante del proceso actual, se produce un cambio de contexto. Siempre ejecuta el proceso que tenga el **menor tiempo restante** de ejecución.

Si llega un nuevo proceso con un tiempo de ejecución **menor** al tiempo que le queda al proceso actual, el SO **interrumpe** al proceso actual y ejecuta el nuevo.

EJEMPLO SJF APROPIATIVO:

Procesos:

Proceso	**Llegada**	**CPU Burst**
P1	0	8
P2	1	4
P3	2	2
P4	3	1

T0-1	T1-2	T2-3	T3-4	T4-5	T5-6	T6-7	T7-8	T8-9	T9-10	T10-11	T11-12	T12-12	T13-14	T14-15
P1	P2	P3	P3	P4	P2	P2	P2	P1	P1	P1	P1	P1	P1	P1
P1 7	P1 7	P1 7	P1 7	P1 7	P1 7	P1 7	P1 7	P1 6	P1 5	P1 4	P1 3	P1 2	P1 1	P1 0
P2 4	P2 3	P2 3	P2 3	P2 3	P2 2	P2 1	P2 0	P2 0	P2 0	P2 0	P2 0	P2 0	P2 0	P2 0
	P3 2	P3 1	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0	P3 0
		P4 1	P4 1	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0	P4 0

EJECUCIÓN CON SRTF:

Tiempo 0: P1 llega, ejecuta P1 (quedan 8)
[P1]

Tiempo 1: P2 llega (burst 4). P1 quedan 7.
¿4 < 7? Sí → ¡INTERRUMPE P1! Ejecuta P2

[P1|P2]

Tiempo 2: P3 llega (burst 2). P2 quedan 3.
¿2 < 3? Sí → ¡INTERRUMPE P2! Ejecuta P3
[P1|P2|P3]

Tiempo 3: P4 llega (burst 1). P3 quedan 1.
¿1 < 1? No (empate, o P3 sigue) → P3 sigue
[P1|P2|P3|P4]

Tiempo 4: P3 termina. ¿Quién tiene menos tiempo restante?
P1=7, P2=3, P4=1 → ¡Ejecuta P4!
[P1|P2| |P4]

Tiempo 5: P4 termina. Ahora: P1=7, P2=3 → Ejecuta P2
[P1|P2| |]

Tiempo 8: P2 termina. Solo queda P1 → Ejecuta P1
[P1| | |]

Tiempo 16: P1 termina. Fin.

Llenar la siguiente tabla;

Proceso	**Llegada**	**Inicio**	**Fin**	**Espera (Inicio - Llegada)**	**Turnaround (Fin - Llegada)**
P1	0				
P2	1				
P3	2				
P4	3				

Promedios:

- **Turnaround: ?**
- **Espera: ?**

Como seria el no Apropiativo?

4.3 Priority Scheduling

- **Descripción**: Cada proceso tiene una prioridad asignada; el de mayor prioridad se ejecuta primero.
- **Tipo**: Puede ser apropiativo o no apropiativo

Problema: Inanición de procesos de baja prioridad.
Solución: Envejecimiento (Aging) - aumentar gradualmente la prioridad de los procesos que esperan mucho tiempo.

Ejemplo No Apropiativo:

Proceso	Llegada	CPU Burst	Prioridad
P1	0	6	3
P2	1	2	1
P3	2	8	4
P4	3	3	2

Proceso	Prioridad	Espera
P3	4 (alta)	4
P1	3 (media)	0
P4	2 (baja)	11
P2	1 (baja)	16

T0-1	T1-2	T2-3	T3-4	T4-5	T5-6	T6-7	T7-8	T8-9	T9-10	T10-11	T11-12	T12-13	T13-14	T14-15	T15-16	T16-17	T17-18	T18-19	T19-20
P1	P1	P1	P1	P1	P1	P3	P3	P3	P3	P3	P3	P3	P3	P4	P4	P4	P2	P2	-

TIEMPO 0:
Llega P1 (prio 3). Es el único → Ejecuta P1

TIEMPO 1:
P1 ejecuta (quedan 5)
Llega P2 (prio 1). Cola listos: [P2]

TIEMPO 2:
P1 ejecuta (quedan 4)
Llega P3 (prio 4). Cola listos: [P2, P3]

TIEMPO 3:
P1 ejecuta (quedan 3)

Llega P4 (prio 2). Cola listos: [P2, P3, P4]

TIEMPO 6:
P1 termina. ¿Quién llegó y tiene mayor prioridad?
P2=1, P3=4, P4=2 → ¡P3 (prio 4)! Ejecuta P3

TIEMPO 14:
P3 termina. ¿Quién queda?
P2=1, P4=2 → ¡P4 (prio 2)! Ejecuta P4

TIEMPO 17:
P4 termina. Solo queda P2 → Ejecuta P2

TIEMPO 19:
P2 termina. FIN

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0	6	6 - 0 = **6**	0 - 0 = **0**
P2	1	17	19	19 - 1 = **18**	17 - 1 = **16**
P3	2	6	14	14 - 2 = **12**	6 - 2 = **4**
P4	3	14	17	17 - 3 = **14**	14 - 3 = **11**

Promedios:

- **Turnaround:** (6 + 18 + 12 + 14) / 4 = **12.5**
- **Espera:** (0 + 16 + 4 + 11) / 4 = **7.75**

Como seria el apropiativo?

4.4 Round Robin (RR)

- **Descripción**: Cada proceso recibe un quantum de tiempo fijo en la CPU.
- **Tipo**: Apropiativo por diseño
- **Implementación**: Cola circular

Ventajas:

- Justo: todos los procesos reciben tiempo de CPU
- Buen tiempo de respuesta para procesos interactivos

Desventajas:

- Rendimiento depende del tamaño del quantum
    - Quantum muy grande → se comporta como FCFS
    - Quantum muy pequeño → overhead excesivo por cambios de contexto

Ejemplo (Quantum = 4):

Proceso	Tiempo de CPU
P1	24
P2	3
P3	3

Diagrama de Gantt:

P1(0-4) | P2(4-7) | P3(7-10) | P1(10-14) | P1(14-18) | P1(18-22) | P1(22-26)

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27
P1	P1	P1	P1	P2	P2	P2	P3	P3	P3	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	

4.5 Multilevel Queue (Colas Multinivel)

- **Descripción**: La cola de listos se divide en varias colas separadas por tipo de proceso.
- Cada cola tiene su propio algoritmo de planificación.
- Las colas tienen prioridad fija entre sí.

Ejemplo típico:

Cola de Sistema (prioridad más alta) → RR con quantum pequeño
Cola de Procesos Interactivos → RR
Cola de Procesos por Lotes → FCFS
Cola de Estudiantes (prioridad más baja) → FCFS

4.6 Multilevel Feedback Queue (Colas Multinivel con Retroalimentación)

- **Descripción**: Similar a colas multinivel, pero los procesos pueden moverse entre colas.
- Permite separar procesos según sus características de CPU burst.

Reglas típicas:

- Un nuevo proceso entra en la cola de mayor prioridad (Q0)
- Si no termina en su quantum, desciende a la siguiente cola
- Las colas inferiores tienen quantums más largos

- Un proceso en cola inferior puede ascender si espera mucho tiempo

Ventajas:

- Adaptativo: procesos I/O-bound permanecen en colas altas
- Procesos CPU-bound descienden a colas con quantums más largos
- Muy flexible y usado en sistemas reales

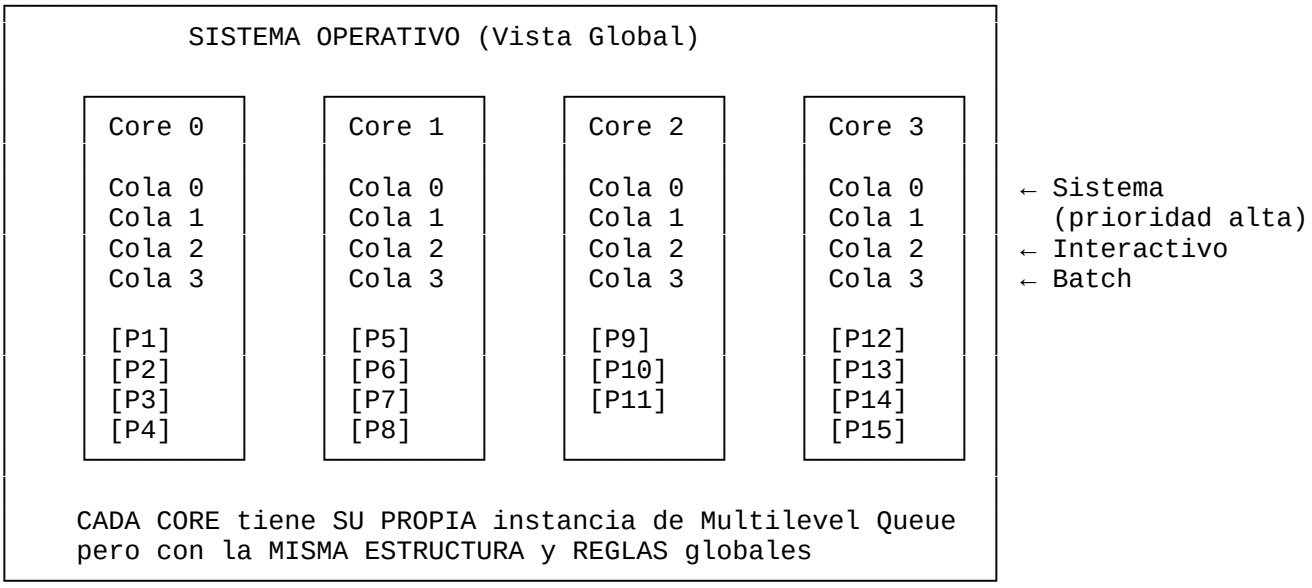
5. Comparación de Algoritmos

Algoritmo	Tipo	Inanición	Uso Típico
FCFS	No apropiativo	No	Sistemas batch simples
SJF	Ambos	Sí (no apropiativo)	Sistemas batch
SRTF	Apropiativo	Sí	Sistemas batch
Prioridad	Ambos	Sí (sin aging)	Sistemas en tiempo real
Round Robin	Apropiativo	No	Sistemas de tiempo compartido
Multilevel Queue	Ambos	Posible	Sistemas mixtos
Multilevel Feedback	Apropiativo	No (con aging)	Sistemas generales (Unix, Windows)

https://www.youtube.com/watch?v=SJ4SnE9Mc58&list=PLBTcJetyW9aJvmEEA0rkQqBeSdQ3r89fX

6. ¿Se procesa de forma diferente en cada núcleo?

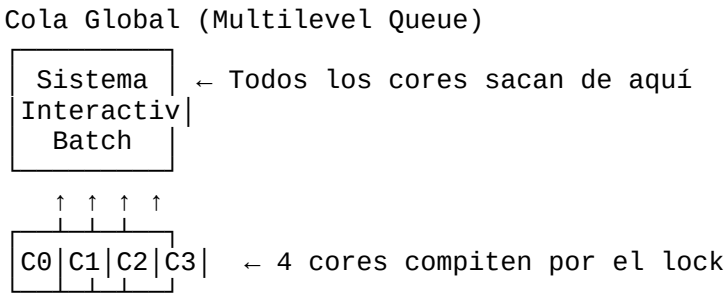
Sí y No. Depende del nivel de análisis:



Dos Formas de Implementarlo

6.1. Colas Globales (Menos común hoy)

Característica
Una sola cola compartida entre todos los cores
Cualquier core puede ejecutar cualquier proceso de la cola
Problema: Contención (varios cores compiten por la misma cola)
Problema: Falta de afinidad de caché



Recordemos que un **lock** (cerrojo/mutex) es un mecanismo de sincronización que garantiza que **solo un proceso o core acceda a un recurso compartido a la vez**.

Recurso	¿Compartido?	¿Necesita Lock?	Ejemplo de Conflicto
CPU (cola de procesos)	✅ Sí	✅ Sí	Dos cores quieren sacar el mismo proceso
RAM (memoria)	✅ Sí	✅ Sí	Dos hilos escriben en la misma dirección
Almacenamiento (disco)	✅ Sí	✅ Sí	Dos procesos escriben el mismo archivo
Red (interfaz de red)	✅ Sí	✅ Sí	Dos procesos envían paquetes simultáneamente
Impresora	✅ Sí	✅ Sí	Dos trabajos se mezclan en la misma hoja
Variables globales	✅ Sí	✅ Sí	Dos hilos incrementan el mismo contador
Estructuras del kernel	✅ Sí	✅ Sí	Tablas de procesos, colas de E/S

Ejemplo 1

Hilo A: lee variable X = 5
Hilo B: lee variable X = 5
Hilo A: X = X + 1 → escribe X = 6
Hilo B: X = X + 1 → escribe X = 6

¡RESULTADO! X = 6, pero debería ser 7
(se perdió una actualización)

SOLUCIÓN: Lock sobre la variable X

Ejemplo 2

Proceso A: abre "datos.txt" → lee → modifica → (interrumpido)
Proceso B: abre "datos.txt" → lee versión vieja → modifica → guarda
Proceso A: guarda → ¡sobrescribe los cambios de B!

SOLUCIÓN: File locking (flock, lockf en Linux)

Ejemplo 3

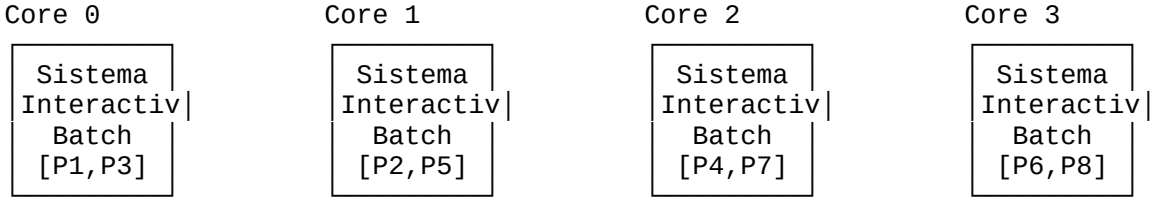
Core 0: prepara paquete TCP para IP 192.168.1.5
Core 1: prepara paquete TCP para IP 192.168.1.10
Ambos: intentan escribir en el buffer de la NIC al mismo tiempo

¡RESULTADO! Paquetes corruptos o mezclados

SOLUCIÓN: Lock en el buffer de transmisión de la NIC

6.2. Colas por Core (Lo que hacen los SO modernos)

Característica
Cada core tiene su propia cola
El SO **migra** procesos entre cores para balancear carga
Ventaja: Menor contención, mejor afinidad de caché
Complicación: Necesita balanceo de carga



Si Core 0 está sobrecargado y Core 3 libre, el SO migra P3 a Core 3

¿Se procesa "de forma diferente" en cada core?

Aspecto	¿Diferente?	Explicación
Estructura de colas	**NO**	Todos los cores usan la misma jerarquía de colas
Reglas de planificación	**NO**	Mismo algoritmo (mismas prioridades, mismos quantums)
Contenido de las colas	**SÍ**	Cada core tiene procesos diferentes en sus colas
Velocidad de ejecución	**SÍ**	Cada core puede estar más o menos ocupado
Migración entre cores	**SÍ**	El SO puede mover procesos según carga

6.3 Analogía

Imagina un **banco con varias cajas**:

Elemento	Equivalencia
Cajas del banco	Cores del procesador
Filas por tipo de cliente	Colas Multinivel (VIP, Empresas, Personas)
Estructura de filas	**Igual en todas las cajas**
Clientes en cada fila	**Diferentes en cada caja**
Gerente que reasigna clientes	Sistema Operativo (balanceo de carga)

Todas las cajas tienen las mismas reglas (VIP primero, luego empresas, luego personas), pero **el contenido de cada fila es diferente** en cada caja.

6.4 En los Sistemas Operativos Reales

Linux (CFS)

- Cada core tiene su propio **RB-Tree** (árbol de procesos listos)
- El balanceo de carga ocurre periódicamente
- Intenta mantener procesos en el mismo core (**afinidad**) para reutilizar caché

https://www.youtube.com/watch?v=Lp_YeKzN4h4

Windows

- Cada core tiene colas de prioridad independientes
- Usa **procesadores preferidos** y **afinidad de máscara**

- Puede forzar migración si un core está sobrecargado

¿Se procesa de forma diferente en cada core? NO

- **La estructura y las reglas**: **NO**, son idénticas (mismo Multilevel Queue)
- **Los procesos específicos y la carga**: **SÍ**, cada core maneja procesos diferentes
- **El SO puede intervenir**: **SÍ**, migrando procesos para balancear

La Multilevel Queue es una **política global** del SO, pero se **replica en cada core** con contenido local.

6.5 Que es una máscara de Bits (Bit Mask)?

Una **máscara** es un patrón de bits que se usa para **seleccionar, habilitar o deshabilitar** ciertos elementos. Es como un filtro.

Ejemplo con 4 cores:

Core:	3	2	1	0	
Bit:	1	1	1	1	← Todos habilitados (máscara = 1111 = 15 en decimal)

Core:	3	2	1	0	
Bit:	0	0	1	1	← Solo cores 0 y 1 (máscara = 0011 = 3 en decimal)

Core:	3	2	1	0	
Bit:	1	0	0	0	← Solo core 3 (máscara = 1000 = 8 en decimal)

"Afinidad de Máscara" en Windows

En el contexto del planificador de Windows, significa:

> **Restringir en qué cores (procesadores) puede ejecutarse un proceso**, usando una máscara de bits.

Ejemplo práctico:

Proceso: Editor de video (pesado)
Máscara de afinidad: 1111 (todos los cores)
→ Puede usar todos los cores disponibles

Proceso: Servicio en segundo plano (ligero)
Máscara de afinidad: 0001 (solo core 0)
→ Solo puede ejecutarse en el core 0

Proceso: Juego (optimizado)

Máscara de afinidad: 1100 (cores 2 y 3)
→ Solo puede usar cores 2 y 3

¿Por qué se usa?

Razón	Explicación
Rendimiento de caché	Si un proceso siempre usa el mismo core, sus datos permanecen en la caché de ese core
Aislamiento	Separar procesos críticos de procesos pesados
Licencias	Algunas licencias de software limitan cuántos cores puede usar
Compatibilidad	Programas antiguos que no funcionan bien con múltiples cores

7. Planificación en Sistemas Reales

7.1 Linux (CFS - Completely Fair Scheduler)

- Introducido en el kernel 2.6.23
- Basado en "tiempo virtual" (vruntime)
- Asigna CPU de manera proporcional al peso de cada proceso
- Usa un árbol rojo-negro para gestionar procesos

7.2 Windows

- Usa un planificador basado en prioridades con quantum variable
- 32 niveles de prioridad
- Prioridades dinámicas que se ajustan según el comportamiento del proceso

---

8. Métricas y Fórmulas

Tiempo de Espera (Waiting Time)

$$WT = \text{Tiempo de finalizacio´n} - \text{Tiempo de llegada} - \text{Tiempo de CPU}$$

Tiempo de Turnaround

$$TAT = \text{Tiempo de finalizacio´n} - \text{Tiempo de llegada}$$

Throughput

$$\text{Throughput} = \frac{\text{Tiempo total}}{\text{Nu´mero de procesos completados}}$$

---

9. Ejercicios Prácticos

Ejercicio 1

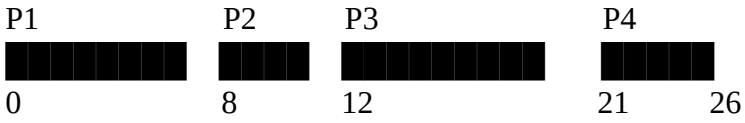
Datos del Ejercicio

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

1. FCFS (First-Come, First-Served) — No Apropiativo

Regla: El primero en llegar, el primero en ejecutar. Nadie interrumpe.

Diagrama de Gantt:



Cálculos:

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0	8	8 - 0 = 8	0 - 0 = 0
P2	1	8	12	12 - 1 = 11	8 - 1 = 7
P3	2	12	21	21 - 2 = 19	12 - 2 = 10
P4	3	21	26	26 - 3 = 23	21 - 3 = 18

Promedios:
- Turnaround promedio: (8 + 11 + 19 + 23) / 4 = 15.25
- Espera promedio: (0 + 7 + 10 + 18) / 4 = 8.75

Problema: Efecto convoy. P1 es largo y hace esperar a todos los demás.

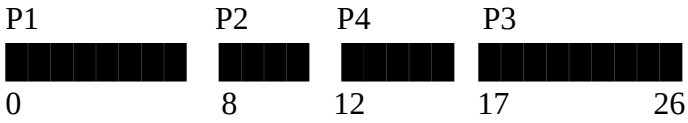
2. SJF (Shortest-Job-First) — No Apropiativo

Regla: De los procesos que han llegado, ejecuta el de menor burst. No se interrumpe.

En cada momento de decisión:

Tiempo 0: Llega P1 (burst 8). Es el único → Ejecuta P1
Tiempo 8: P1 termina. ¿Quién llegó? P2(4), P3(9), P4(5)
    Menor burst: P2(4) → Ejecuta P2
Tiempo 12: P2 termina. ¿Quién queda? P3(9), P4(5)
    Menor burst: P4(5) → Ejecuta P4
Tiempo 17: P4 termina. Solo queda P3 → Ejecuta P3

Diagrama de Gantt:



Cálculos:

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0	8	8 - 0 = 8	0 - 0 = 0
P2	1	8	12	12 - 1 = 11	8 - 1 = 7
P3	2	17	26	26 - 2 = 24	17 - 2 = 15
P4	3	12	17	17 - 3 = 14	12 - 3 = 9

Promedios:
- Turnaround promedio: (8 + 11 + 24 + 14) / 4 = 14.25
- Espera promedio: (0 + 7 + 15 + 9) / 4 = 7.75

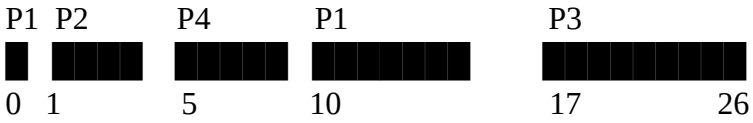
Nota: Mejor que FCFS, pero P3 (el más largo) sufre mucho.

3. SRTF (Shortest-Remaining-Time-First) — Apropiativo

Regla: Siempre ejecuta el proceso con menor tiempo restante. Puede interrumpir.

Tiempo 0: Llega P1 (restan 8). Ejecuta P1
Tiempo 1: Llega P2 (burst 4). P1 restan 7. ¿4 < 7? Sí → ¡Interrumpe P1! Ejecuta P2
Tiempo 2: Llega P3 (burst 9). P2 restan 3. ¿9 < 3? No → Sigue P2
Tiempo 3: Llega P4 (burst 5). P2 restan 2. ¿5 < 2? No → Sigue P2
Tiempo 5: P2 termina. ¿Quién tiene menos restante?
P1 restan 7, P3 restan 9, P4 restan 5
Menor: P4(5) → Ejecuta P4
Tiempo 10: P4 termina. ¿Quién tiene menos restante?
P1 restan 7, P3 restan 9
Menor: P1(7) → Ejecuta P1
Tiempo 17: P1 termina. Solo queda P3 → Ejecuta P3

Diagrama de Gantt:



Cálculos:

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0 , 10	17	17 - 0 = 17	(1-0) + (10-1 (1 que ya atendió)) = 10
P2	1	1	5	5 - 1 = 4	1 - 1 = 0
P3	2	17	26	26 - 2 = 24	17 - 2 = 15
P4	3	5	10	10 - 3 = 7	5 - 3 = 2

Promedios:
- Turnaround promedio: (17 + 4 + 24 + 7) / 4 = 13.0
- Espera promedio: (10 + 0 + 15 + 2) / 4 = 6.75

Mejor resultado para tiempo de espera promedio. Pero muchos cambios de contexto.

4. Round Robin (RR) — Quantum = 2

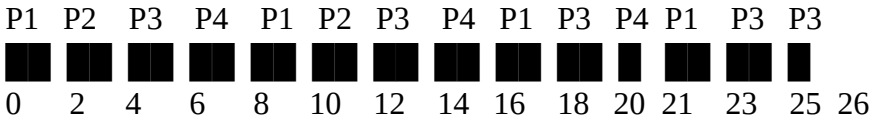
Regla: Cada proceso ejecuta máximo 2ms, luego va al final de la cola.

Cola inicial (por orden de llegada): [P1, P2, P3, P4]

Tiempo 0: Ejecuta P1 (2ms) → quedan 6. Cola: [P2, P3, P4, P1]
Tiempo 2: Ejecuta P2 (2ms) → quedan 2. Cola: [P3, P4, P1, P2]
Tiempo 4: Ejecuta P3 (2ms) → quedan 7. Cola: [P4, P1, P2, P3]
Tiempo 6: Ejecuta P4 (2ms) → quedan 3. Cola: [P1, P2, P3, P4]
Tiempo 8: Ejecuta P1 (2ms) → quedan 4. Cola: [P2, P3, P4, P1]
Tiempo 10: Ejecuta P2 (2ms) → quedan 0. ¡TERMINA! Cola: [P3, P4, P1]
Tiempo 12: Ejecuta P3 (2ms) → quedan 5. Cola: [P4, P1, P3]
Tiempo 14: Ejecuta P4 (2ms) → quedan 1. Cola: [P1, P3, P4]
Tiempo 16: Ejecuta P1 (2ms) → quedan 2. Cola: [P3, P4, P1]
Tiempo 18: Ejecuta P3 (2ms) → quedan 3. Cola: [P4, P1, P3]
Tiempo 20: Ejecuta P4 (1ms) → quedan 0. ¡TERMINA! Cola: [P1, P3]

Tiempo 21: Ejecuta P1 (2ms) → quedan 0. ¡TERMINA! Cola: [P3]
Tiempo 23: Ejecuta P3 (2ms) → quedan 1. Cola: [P3]
Tiempo 25: Ejecuta P3 (1ms) → quedan 0. ¡TERMINA!

Diagrama de Gantt:



Cálculos:

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0 , 8, 16, 21	23	23 - 0 = 23	(8-2)+(16-10)+(21-18) = 15
P2	1	2, 10	12	12 - 1 = 11	(10-4) = 6
P3	2	4, 12, 18, 23, 25	26	26 - 2 = 24	(12-6)+(18-14)+(23-20)+(25-24) = 14
P4	3	6, 14, 20	21	21 - 3 = 18	(14-8)+(20-16) = 10

Promedios:
- Turnaround promedio: (23 + 11 + 24 + 18) / 4 = 19.0
- Espera promedio: (15 + 6 + 14 + 10) / 4 = 11.25

Nota: Peor tiempo promedio, pero mejor tiempo de respuesta para procesos interactivos. Todos reciben atención rápido.

5. Priority Scheduling (Prioridad) — No Apropiativo

Regla: Mayor número = mayor prioridad.

Proceso	Llegada	Burst	Prioridad
P1	0	8	2
P2	1	4	1
P3	2	9	4

P4	3	5	3

En cada momento de decisión:

Tiempo 0: Solo P1 llegó (prio 2) → Ejecuta P1
Tiempo 8: P1 termina. ¿Quién llegó? P2(1), P3(4), P4(3)
    Mayor prioridad: P3(4) → Ejecuta P3
Tiempo 17: P3 termina. ¿Quién queda? P2(1), P4(3)
    Mayor prioridad: P4(3) → Ejecuta P4
Tiempo 22: P4 termina. Solo queda P2 → Ejecuta P2

Diagrama de Gantt:

Cálculos:

Proceso	**Llegada**	**Inicio**	**Fin**	**Turnaround (Fin - Llegada)**	**Espera (Inicio - Llegada)**
P1	0	0	8	8 - 0 = 8	0 - 0 = 0
P2	1	22	26	26 - 1 = 25	22 - 1 = 21
P3	2	8	17	17 - 2 = 15	8 - 2 = 6
P4	3	17	22	22 - 3 = 19	17 - 3 = 14

Promedios:
- Turnaround promedio: (8 + 25 + 15 + 19) / 4 = 16.75
- Espera promedio: (0 + 21 + 6 + 14) / 4 = 10.25

Problema: P2 (prioridad baja) sufre inanición extrema.

Comparación Final

Planificador	**Turnaround Promedio**	**Espera Promedio**	**Tipo**	**Observación**
FCFS	15.25	8.75	No apropiativo	Simple, pero efecto convoy
SJF	14.25	7.75	No apropiativo	Mejor que FCFS

SRTF	13.0	6.75	Apropiativo	Óptimo
RR (q=2)	19.0	11.25	Apropiativo	Justo, buen tiempo de respuesta
Priority	16.75	10.25	No apropiativo	P2 sufre inanición

Ejercicio

Resuelve con Round Robin (quantum = 3) y SJF apropiativo:

Proceso	**Llegada**	**Burst**
P1	0	5
P2	2	3
P3	4	2
P4	5	4

Ejercicio

Dado los siguientes procesos, calcule el tiempo de espera promedio y turnaround promedio para FCFS, SJF (no apropiativo) y Round Robin (quantum=2):

Proceso	**Llegada**	**CPU Burst**
P1	0	8
P2	1	4
P3	2	9
P4	3	5

---

9. Referencias

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Pearson.
- Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.