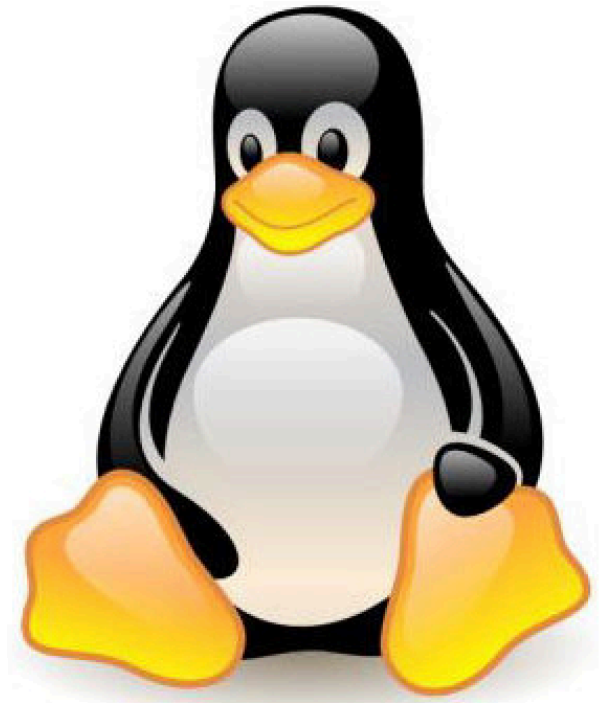
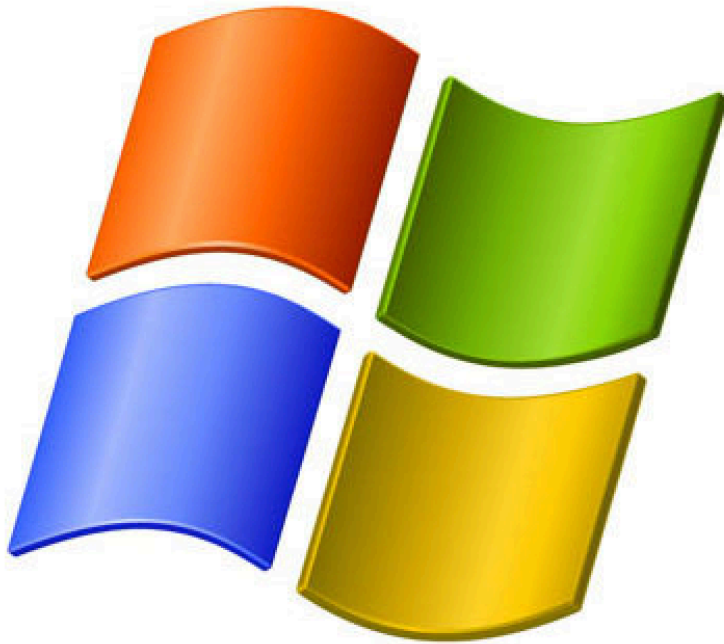


# Tema 2. Segunda parte: Planificación de procesos

## Sistemas Operativos

ETSI Informática. Universidad de Málaga



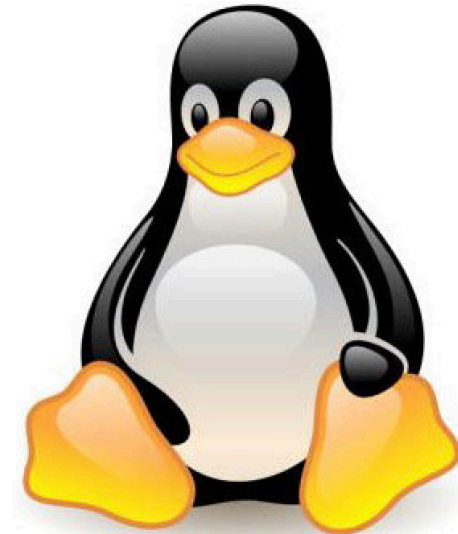
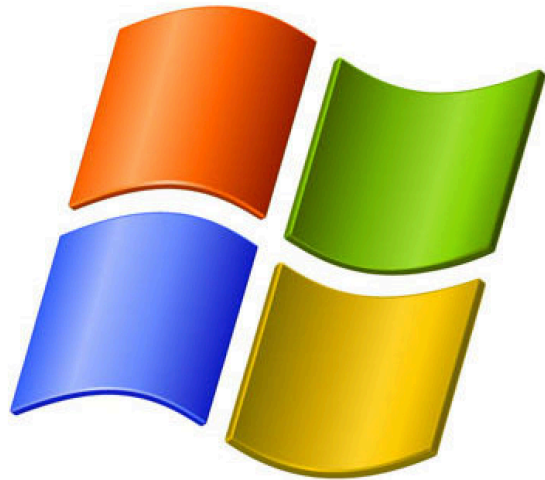
**Manuel Ujaldón**

Catedrático de Arquitectura de Computadores  
Departamento de Arquitectura de Computadores  
Universidad de Málaga

# Índice [30 diapositivas]

---

- I. Conceptos básicos [5]
- II. Algoritmos de planificación [19]
- III. Ejemplos [5]
- IV. Bibliografía [1]



# I. Conceptos básicos

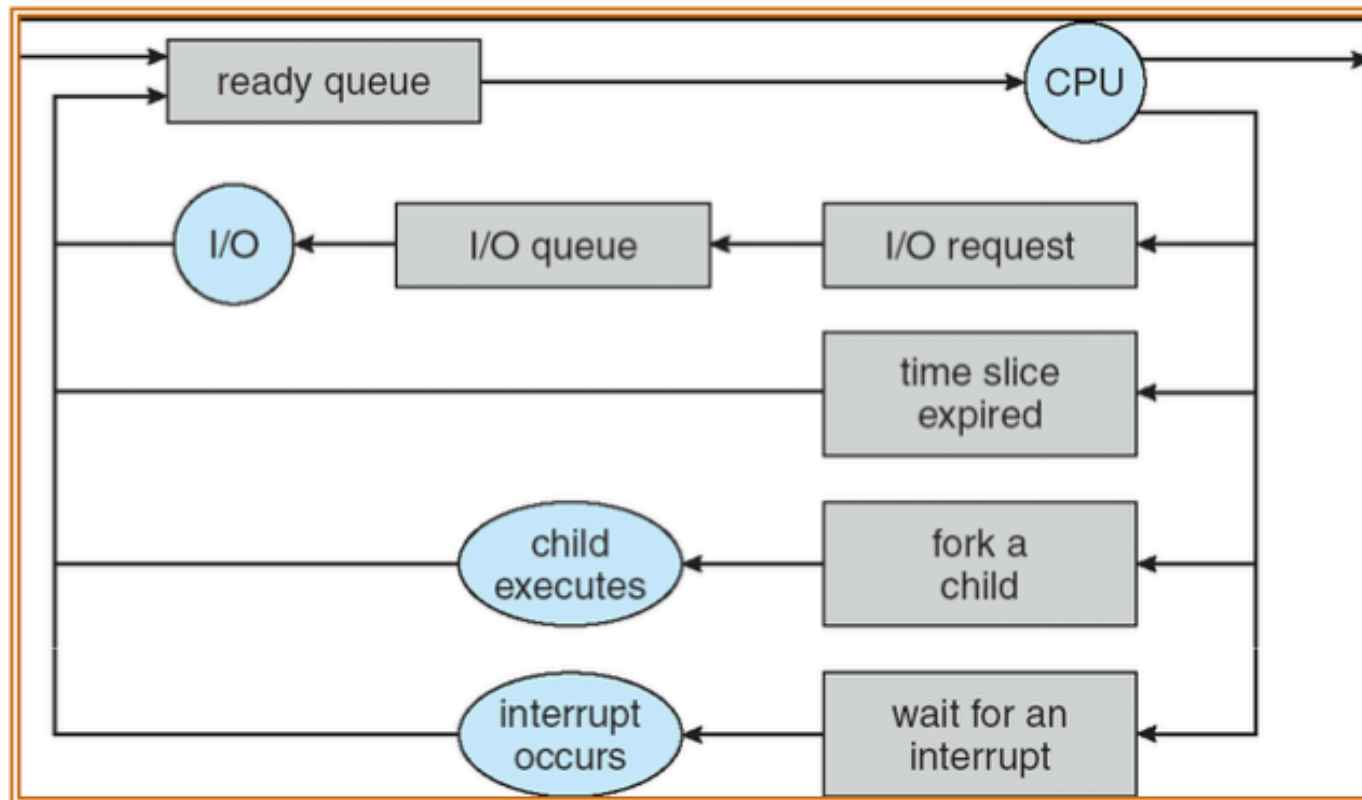
# Axiomas de trabajo

---

- Nuestro modelo de ejecución asume que los programas intercalan tiempos de computación en CPU (bursts o ráfagas) con otros en los que usan los periféricos (entrada/salida, I/O).
  - Los tiempos de CPU son más frecuentes, pero también más cortos.
  - Si predomina el tiempo de CPU, el proceso se califica como *CPU-bound* (limitado por la CPU). En el polo opuesto, tenemos *I/O-bound*.
- El planificador (*scheduler*) del S.O. se encarga de ocupar la CPU por *quantums* de tiempo fijos para todos los procesos al margen de su intensidad computacional.
- Si un proceso necesita una ráfaga de CPU mayor que el quantum que le asigna el S.O., abandonará la CPU y reanudará su uso en el siguiente quantum, así hasta terminar.

# Matrices de la planificación

- El S.O. implementa un algoritmo para elegir el siguiente proceso que ocupa la CPU (y cada recurso compartido), de entre todos los que se encuentran en la cola de listos (Ready).



# Tipos de planificadores

---

- **A corto plazo (*short-term*):** Elige el siguiente proceso que ocupa la CPU. Es el principal protagonista de este capítulo.
  - Se invoca con mucha frecuencia (cada pocos milisegundos) y debe ser ágil para que el rendimiento no decaiga.
- **A medio plazo:** Elige los procesos que se quedan en memoria y los que se alojan en disco.
  - Determina la presión que se ejerce sobre la memoria por parte del grado de multiprogramación, tratando de evitar el riesgo de *thrashing*.
- **A largo plazo (*long-term*):** Selecciona los procesos que se meten en la cola de listos (*Ready*).
  - Se invoca con poca frecuencia (cada pocos segundos), y debe controlar, además, el grado de multiprogramación vigente en cada momento (máximo número de procesos activos).

# Métricas de planificación

---

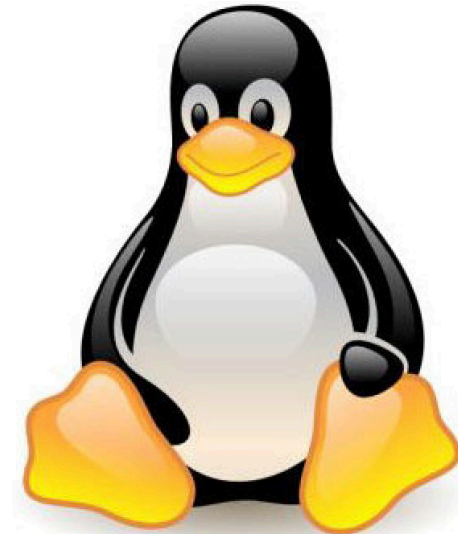
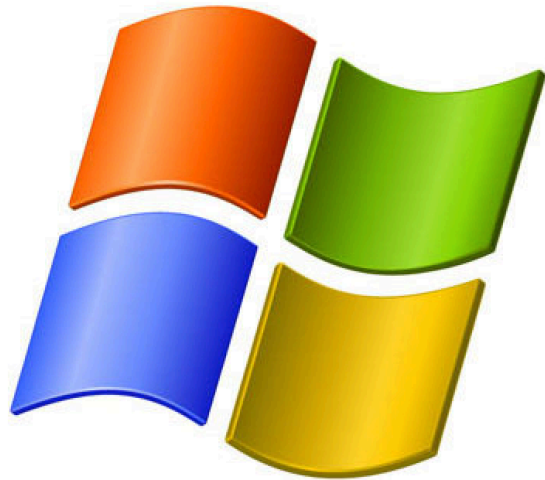
- Utilización de la CPU: Maximizar el % que está ocupada.
- *Throughput (ritmo de ejecución)*: Maximizar el número de procesos que completan su ejecución por unidad de tiempo.
- Tiempo de compleción ( $T_r$ ): Minimizar el que engloba los 4 tiempos siguientes:
  - La espera para alojarse en memoria.
  - La espera en la cola de listos (Ready).
  - El tiempo en CPU.
  - El tiempo de E/S.
- Tiempo de espera ( $T_e$ ): Sólo la espera en la cola de listos.
- Tiempo de respuesta ( $T_a$ ): Lo que tarda en responderse una petición. También se denomina latencia.

# Política de planificación

---

- Es la estrategia que sigue el algoritmo de planificación para optimizar la métrica que mejor refleje los objetivos.
- En sistemas por lotes (*batch*):
  - % CPU.
  - Tiempo de compleción ( $T_r$ ).
- En sistemas de tiempo compartido:
  - Varianza del tiempo de respuesta.
  - Proporción entre la complejidad y el tiempo de respuesta.
- En sistemas de tiempo real:
  - No superar tiempos críticos.
  - Predecibilidad o consistencia.





## II. Algoritmos de planificación

# Planificador de la CPU

---

- Elige el proceso que va a ocupar la CPU de entre los que se encuentran en la cola de listos. Actúa cuando un proceso:
  - Conmuta del estado *Running* al estado *Waiting*.
  - Conmuta del estado *Waiting* al estado *Ready*.
  - Conmuta del estado *Running* al estado *Ready*.
  - Finaliza.
- Un planificador con desalojo o expropiativo (*preemptive*) permite a un proceso desalojar de la CPU de forma inmediata al que la está ocupando (por ejemplo, por ser más prioritario). El planificador sin desalojo (*non-preemptive*) esperará siempre a que salga el proceso que ocupa la CPU para relevarle.

# Síntesis de los algoritmos de planificación que estudiaremos

No desalojan la CPU (no expropiativos, <i>non-preemptive</i> )	Desalojan la CPU (expropiativos, <i>preemptive</i> )	Métrica a la que conceden más importancia
FCFS	Tiempo compartido ( <i>round-robin</i> )	Instante en que cada proceso llega al sistema
SJF	SRTF	Tiempo medio de compleción
Prioridad	Prioridad con desalojo	Respetar la prioridad

● También veremos la planificación por colas multinivel, que es un mecanismo muy empleado en los S.O. reales.

# FCFS (First Come First Served)

---

- Los procesos se ejecutan en el orden de llegada, es decir, como nos atienden en la cola de un comercio.

- Ventaja:

- Sencillo de implementar.

- Inconvenientes:

- Le cuesta maximizar el porcentaje de uso de la CPU y los dispositivos.

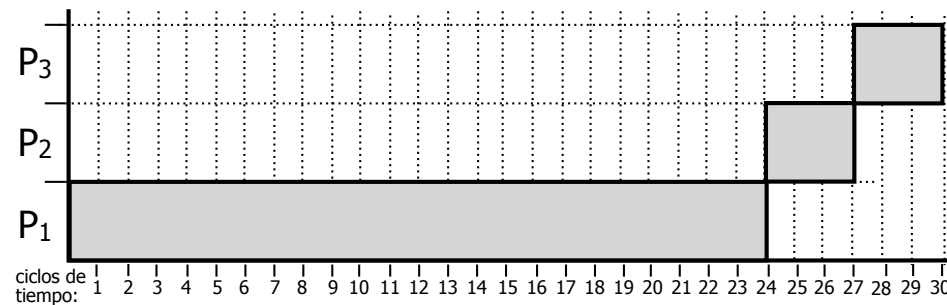
- Le cuesta minimizar los tiempos de compleción ( $T_r$ ) y espera ( $T_e$ ).

- No es buena opción en los sistemas de tiempo compartido, en los que cada usuario necesita mantener una cuota de uso de la CPU a intervalos regulares.

- Adolece del *convoy effect*: Los procesos cortos eternizan su llegada a la CPU por encontrarse ésta monopolizada por un proceso largo.

# FCFS. Ejemplo

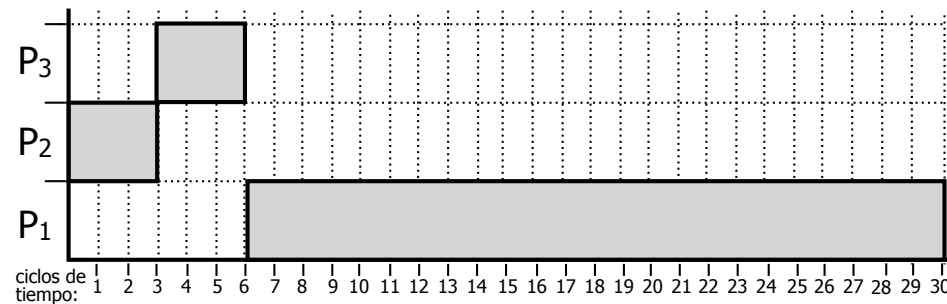
- Supongamos un sistema en el que los procesos llegan en el orden  $P_1$ ,  $P_2$  y  $P_3$ , requiriendo 24, 3 y 3 ciclos de CPU, respectivamente. Usaremos diagramas de Gantt para ilustrar la evolución temporal de los procesos. En este caso:



- El cálculo de métricas de rendimiento es el siguiente:
  - Tiempos de espera ( $T_e$ ):  $P_1 = 0$ ;  $P_2 = 24$ ;  $P_3 = 27$ . Media: 17.
  - Tiempos de compleción ( $T_r$ ):  $P_1 = 24$ ;  $P_2 = 27$ ;  $P_3 = 30$ . Media: 27.
- Se produce el *convoy-effect*: Dos procesos cortos esperan demasiado por estar precedidos de otro mucho más largo.

# FCFS. Ejemplo (cont.)

- Si los procesos llegan en el orden  $P_2$ ,  $P_3$  y  $P_1$ , el diagrama de Gantt cambia por completo:



- Y las métricas revelan un resultado mucho mejor, al desaparecer el *convoy effect*:
  - T. de espera ( $T_e$ ):  $P_1 = 6$ ;  $P_2 = 0$ ;  $P_3 = 3$ . Media: 3. (antes 17)
  - T. de compleción ( $T_r$ ):  $P_1 = 30$ ;  $P_2 = 3$ ;  $P_3 = 6$ . Media: 13 (antes 27).

# Tiempo compartido

---

- Reparte el tiempo de la CPU en intervalos o *quantums* de unos 5 msg. que asigna de forma rotatoria entre los procesos que compiten por el uso de la CPU en cada momento.
- El rendimiento depende del quantum:
  - Si es grande, converge al algoritmo FCFS y es menos justo.
  - Si es pequeño, penaliza más la sobrecarga del cambio de contexto.
  - Mejor pequeño si hay muchos procesos y grande si hay pocos.
- Al repartir el tiempo democráticamente entre todos los procesos, los más cortos salen ganando en el reparto.
- Es el algoritmo que más se aplica en los sistemas reales, no sólo en los S.O.

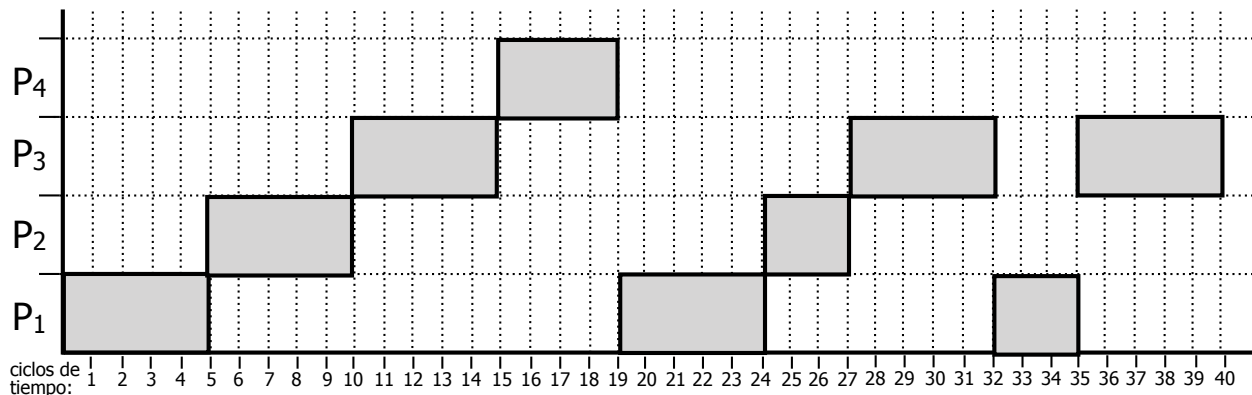
# Tiempo compartido. Ejemplo

- Sea un *quantum*  $q=5$  ciclos y los siguientes procesos:

Proceso	Ráfaga de CPU (en ciclos)
P <sub>1</sub>	13
P <sub>2</sub>	8
P <sub>3</sub>	15
P <sub>4</sub>	4

- Tiempos de espera ( $T_e$ ):
  - $P_1 = (19-5)+(32-24) = 22$
  - $P_2 = (24-10)+(5-0) = 19$
  - $P_3 = 10+12+3 = 25$
  - $P_4 = 15$
  - $T_e \text{ medio: } 82/4 = 20.25$
- $T_r = (35+27+40+19)/4 = 30.25$

- Diagrama de Gantt:





# ¿Qué pasaría si conociéramos el futuro?

---

- Asignaríamos la CPU a los procesos que les queda menos para acabar, minimizando el tiempo medio de compleción.
- En caso de desconocer este dato futuro, podemos estimarlo desde el pasado:
  - Si el proceso viene siendo intensivo en CPU, lo seguirá siendo.
  - Se aproxima así la próxima ráfaga de CPU de un proceso en función de sus últimas ráfagas, bien calculando la media de las N últimas ráfagas, o mejor aún, dando más importancia a las ráfagas más recientes en una media ponderada.
- Esa idea la aplican los algoritmos SJF y SRTF, que usan esta próxima ráfaga, ya sea conocida o estimada, para decidir el proceso que ocupa la CPU en cada momento. Pasamos a verlos a continuación.

# Shortest Job First (SJF)

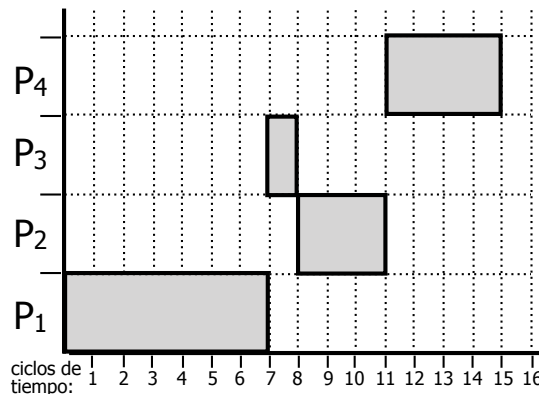
---

- Asocia a cada proceso la longitud de su próxima ráfaga de CPU, ocupando la CPU el que tenga la ráfaga más corta.
- Es un algoritmo sin desalojo, es decir, el proceso que solicita el uso de la CPU esperará a que deje de usarla el que la tiene asignada en ese momento.
- **Ventaja:**
  - Reduce el tiempo medio de compleción.
- **Inconveniente:**
  - Perjudica a los procesos intensivos en el uso de la CPU, que apenas avanzan en abundancia de procesos cortos.

# SJF. Ejemplo

Proceso	Instante de llegada a la cola de procesos Ready	Ráfaga de CPU
P <sub>1</sub>	Ciclo 0	7 ciclos
P <sub>2</sub>	Ciclo 2	3 ciclos
P <sub>3</sub>	Ciclo 4	1 ciclos
P <sub>4</sub>	Ciclo 5	4 ciclos

## Diagrama de Gantt:



$T_e = [0 + (8-2) + (7-4) + (11-5)] / 4 = 3.75$  (tiempo de espera)

$T_r = [(7-0)+(11-2)+(8-4)+(15-5)] / 4 = 7.5$  (tiempo de compleción)

# Shortest Remaining Time First (SRTF)

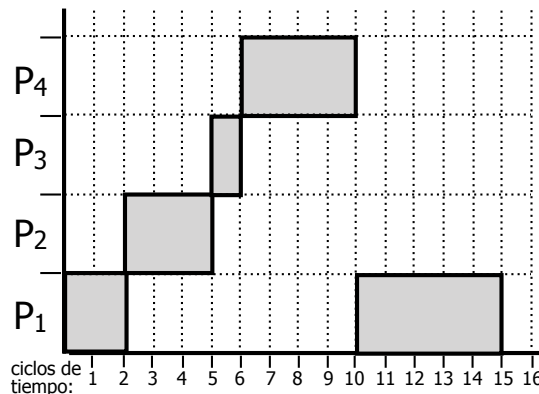
---

- Es la versión con desalojo (*preemptive*) del algoritmo SJF: En cuanto llegue un proceso con ráfaga de CPU más corta que el que tiene la CPU, se apropiará de ella.
- Ventaja:
  - Minimiza el tiempo medio de respuesta.
- Inconveniente:
  - Al ser más agresivo, acentúa el problema del SJF: En abundancia de procesos cortos, los largos apenas progresan.

# SRTF. Ejemplo

Proceso	Ciclo de llegada	Ráfaga de CPU (en ciclos)
P <sub>1</sub>	0	7
P <sub>2</sub>	2	3
P <sub>3</sub>	4	1
P <sub>4</sub>	5	4

## Diagrama de Gantt:



$T_e = [(10-2) + 0 + (5-4) + (6-5)] / 4 = 2.5$  (tiempo de espera)

$T_r = [(15-0) + (5-2) + (6-4) + (10-5)] / 4 = 6.25$  (t. de compleción)

# Planificación con prioridad

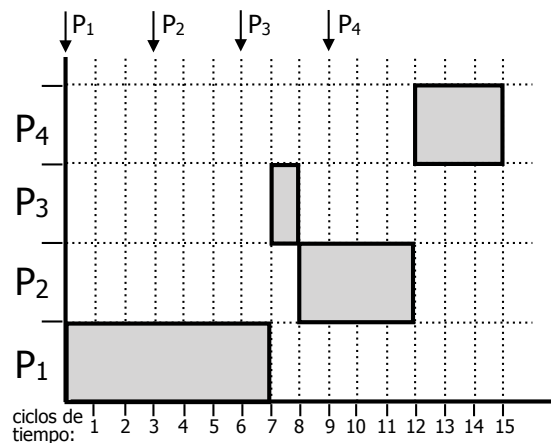
---

- Asignamos a cada proceso un número entero positivo, más bajo a mayor prioridad, para otorgar la CPU al proceso más prioritario en cada momento:
  - Algoritmo sin desalojo: La cola Ready se ordena según prioridades.
  - Algoritmo con desalojo: Cuando un proceso llega a la cola Ready, compara su prioridad con el proceso en ejecución y ocupa la CPU si tiene una prioridad mayor.
- SJF y SRTF son ejemplos sin/con desalojo si se toma el tiempo de la próxima ráfaga como número de prioridad.
- Riesgo: Relegar demasiado a procesos de baja prioridad.
- Solución: Implementar un mecanismo de “envejecimiento” (*aging*) que suba la prioridad con el tiempo de espera.

# Algoritmos con prioridad. Ejemplo

Proceso	Ciclo de llegada	Ráfaga de CPU	Prioridad
P <sub>1</sub>	0	7	3
P <sub>2</sub>	3	4	2
P <sub>3</sub>	6	1	1
P <sub>4</sub>	9	3	4

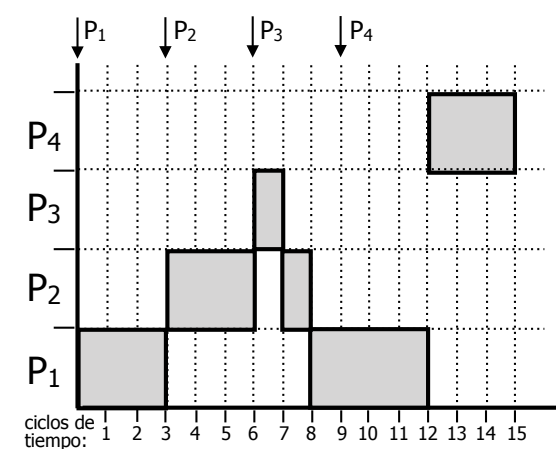
● D. de Gantt sin desalojo:



●  $T_e = [0 + (8-3) + (7-6) + (12-9)] / 4 = 2.25$

●  $T_r = [(7-0) + (12-3) + (8-6) + (15-9)] / 4 = 6$

D. de Gantt con desalojo:



$T_e = (5 + 1 + 0 + 3) / 4 = 2.25$

$T_r = [(12-0) + (8-3) + (7-6) + (15-9)] / 4 = 6$

# Planificación por colas multinivel

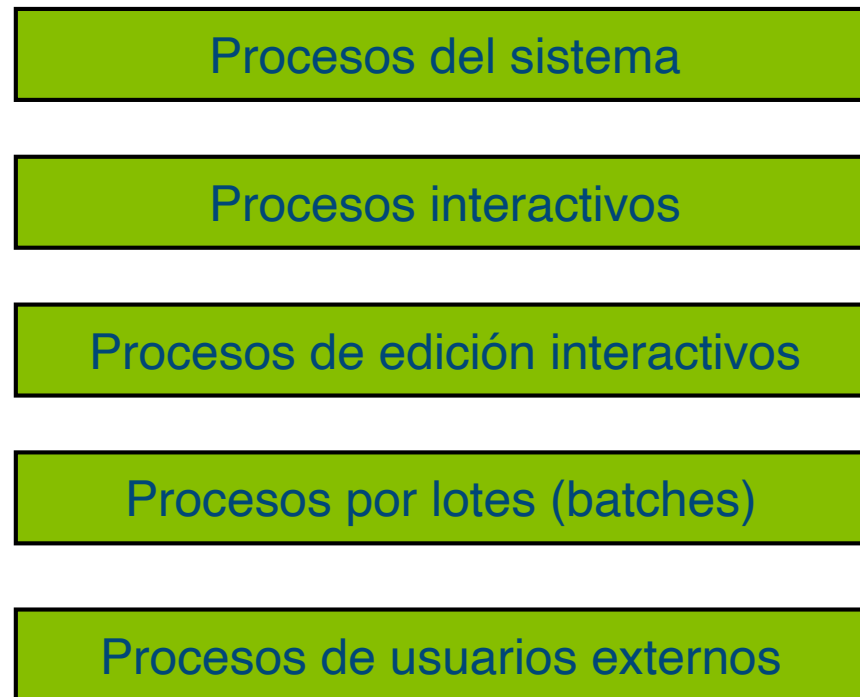
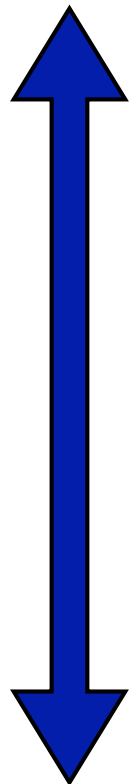
---

- Consiste en escindir la cola de procesos *Ready* en varias de distinta prioridad, y establecer un algoritmo distinto en cada cola. Ejemplo:
  - Procesos interactivos en una cola en primer plano (*foreground*), que asigne la CPU mediante *round-robin*.
  - Procesos *batch* en otra cola menos prioritaria (*background*), que seleccione los procesos mediante FCFS.
- Para prevenir la espera indefinida en los procesos menos prioritarios, podemos establecer:
  - Reglas para que un proceso migre de una cola a otra (como *aging*).
  - Un porcentaje de uso de la CPU en cada cola (por ejemplo, 80% para los procesos interactivos y 20% para los procesos *batch*).



# Ejemplo de sistema con 5 colas

Mayor prioridad.

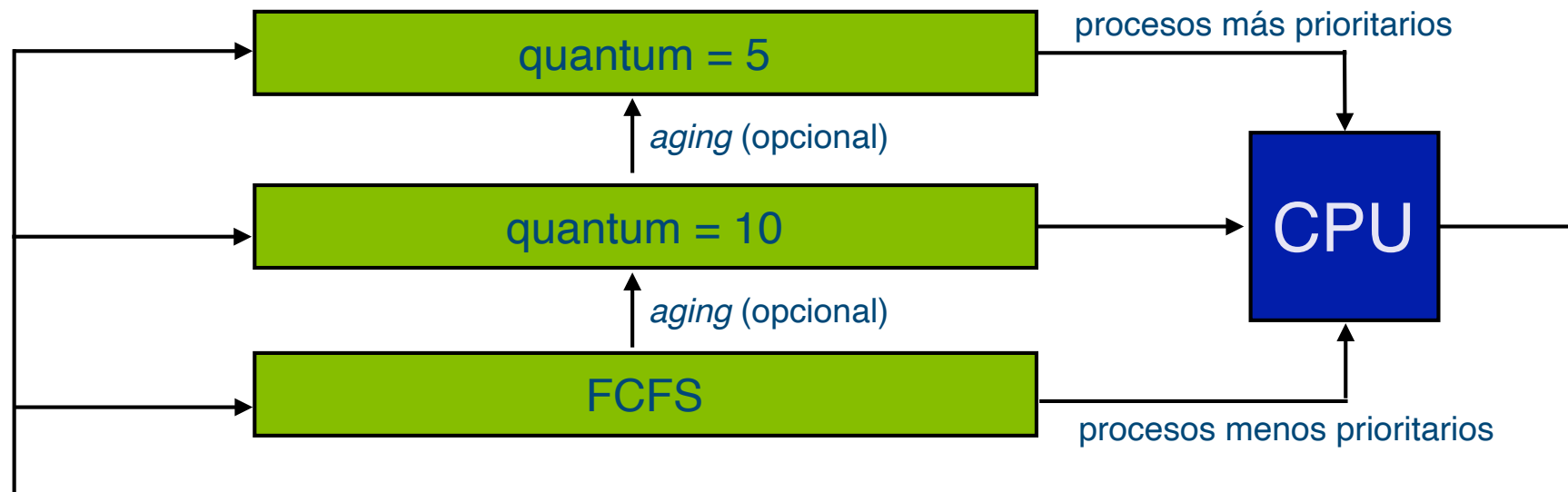


El planificador de la CPU sólo elige de una cola inferior si todas las superiores están vacías

Menor prioridad.

# Ejemplo de sistemas con 3 colas multinivel

- Cada cola se gestiona mediante una política propia.



Retorno a la cola correspondiente cuando finaliza su quantum o realiza E/S:

- Si realiza E/S, puede promocionar a una cola superior.
- Si completa su quantum, puede promocionar a una cola inferior.

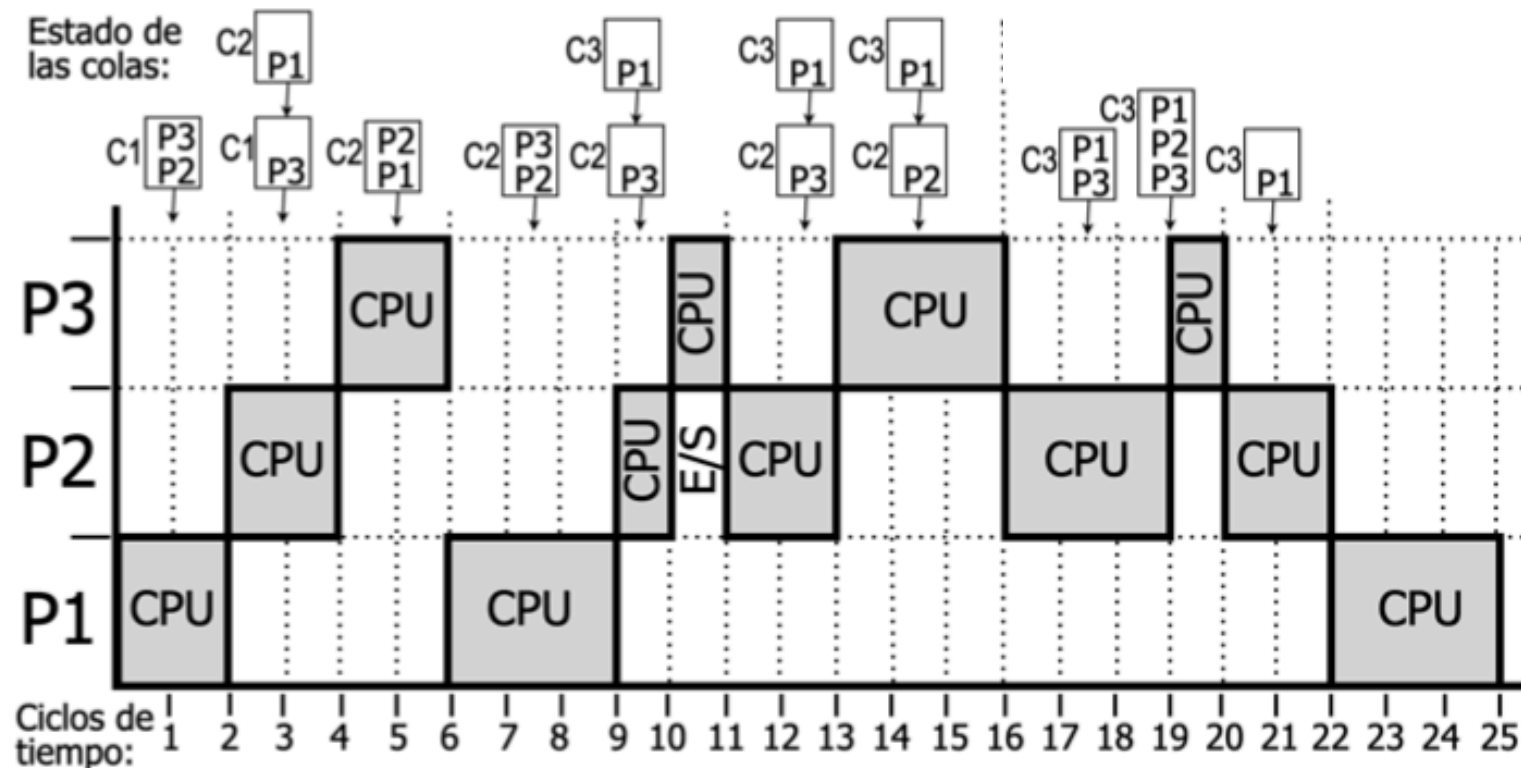
- Todos los procesos empiezan en la cola superior, y van migrando entre ellas según su actividad.
- El objetivo es proporcionar justicia y eficiencia a la vez.

# Ejercicio 2 de la relación de problemas

Proceso	CPU	E/S	CPU
P <sub>1</sub>	8		
P <sub>2</sub>	3	1	7
P <sub>3</sub>	7		

● Planificador con desalojo con tres colas C1, C2 y C3 de mayor a menor prioridad. Los procesos comienzan en C1, donde se adopta tiempo compartido con quantum de 2 ciclos. En C2, tiempo compartido con quantum de 3 ciclos. En C3, se usa SRTF.

- Si un proceso agota su quantum, baja de nivel. Si no, sube de nivel.
- Si es desalojado por prioridad mantiene ésta, y pasa al final de la cola en la que partió, recibiendo un quantum completo.
- Si un proceso reanuda su ejecución tras un bloqueo, también recibe un quantum completo.

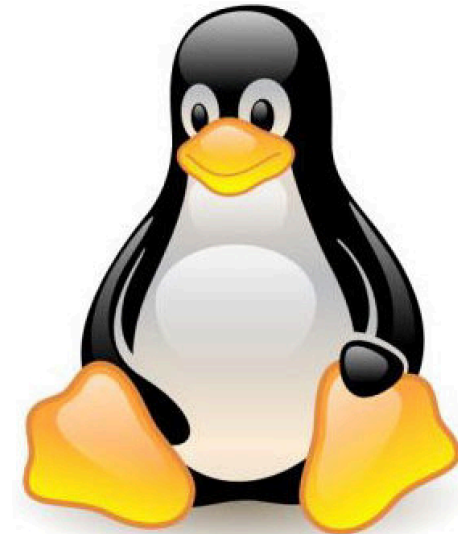
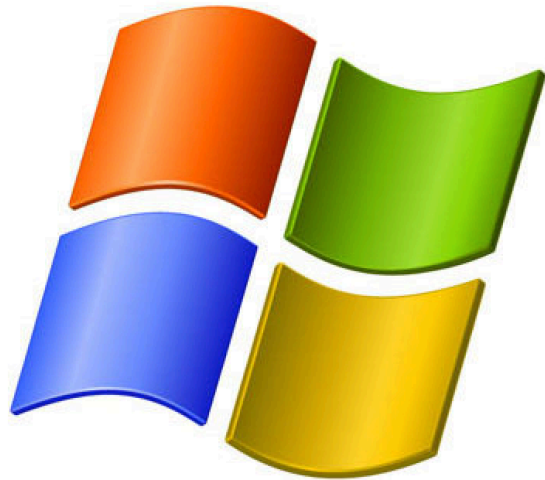


# Planificador para múltiples CPUs

---

## ● Resulta más complejo de administrar:

- Si todas las CPUs son gemelas (*SMP, Symmetric Multi-Processor*), que es lo habitual, las colas Ready admiten dos implementaciones:
  - Todos los procesos en una cola común (como en las cajas de Carrefour).
  - Cada CPU tiene su cola propia (como en las cajas de Mercadona).
- Si las CPUs son distintas, la más potente suele ocuparse de mantener las estructuras de datos del sistema, aliviando la necesidad de compartir información entre todas ellas.
- Cada proceso debe tener afinidad por la CPU en la que comienza a ejecutarse, reanudando su ejecución en ella, aunque se pueden establecer mecanismos de migración:
  - Las CPUs más sobrecargadas pueden delegar procesos.
  - Las CPUs más aliviadas pueden tomar procesos de las anteriores.



### III. Ejemplos

# El planificador de Linux

- Selecciona tanto procesos como hilos (tasks).
- Es un planificador con desalojo y prioridades.
- Nuestro punto de partida será el kernel 2.6, que adopta un planificador constante,  $O(1)$ , con dos colas multinivel:
  - Procesos en tiempo real: sin realimentación, con prioridades estáticas entre 1 y 99 (que siempre prevalecen frente a las normales).
  - Procesos normales: con realimentación y prioridades dinámicas entre 100 y 139, que tratan de favorecer a los procesos con más E/S.
    - Por defecto, los procesos comienzan con prioridad 120.
    - Se les sube la prioridad (hasta -5) según se prodiguen en operaciones de E/S.
    - Se les baja la prioridad (hasta +5) si son intensivos en CPU (agotan quantum).
  - Para cada prioridad, la cola se escinde en dos:
    - Active/Expired, con los procesos que no/sí agotaron su quantum.
    - El siguiente proceso a ejecutar se toma de la cola Active.

# Gestión de prioridades en el planificador $O(1)$

- Cada proceso comienza con la prioridad heredada de su proceso padre o una prioridad base que puede asignarle el usuario por medio de la llamada `nice()`.
- Los procesos con igual prioridad (valor *nice*) usan tiempo compartido en su cola, donde el quantum se calcula así:
  - Si la prioridad está entre 100 y 119,  $\text{quantum} = (140 - \text{prioridad}) \times 20$ .
  - Si la prioridad está entre 120 y 139,  $\text{quantum} = (140 - \text{prioridad}) \times 5$ .

## ● Ejemplos:

	Prioridad	Valor nice	Quantum
Máxima	100	-20	800 msg.
Alta	110	-10	600 msg.
Por defecto	120	0	100 msg.
Baja	130	10	50 msg.
Mínima	139	19	5 msg.



# El planificador mejorado

## CFS (Completely Fair Scheduler) de Linux

- Se introduce a partir del núcleo 2.6.23 (Octubre de 2007).
- La idea es compartir la CPU de forma proporcional a *nice*, asignando un % a cada proceso en lugar de un quantum.
  - Se define la **latencia objetivo** como el lapso mínimo de tiempo en el que rotarán todos los procesos (por defecto, 20 msg.).
  - Si hay N procesos compitiendo por la CPU, el quantum será  $20/N$ .
  - El valor nice se usa para ponderar este quantum medio, de forma que a cada proceso se le asignará un tiempo diferente de la CPU.

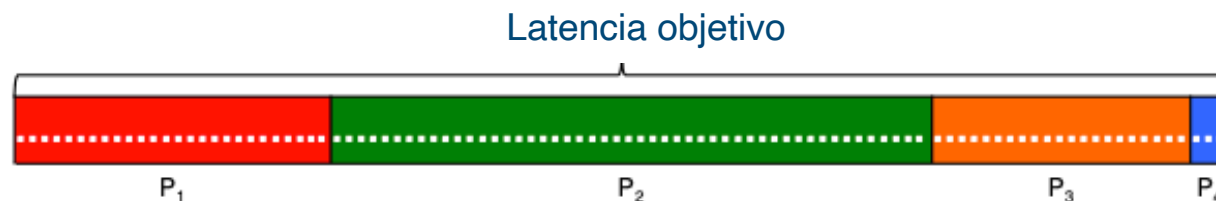
Nice	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11
Weight	88761	71755	56483	46273	36291	29154	23254	18705	14949	11916
Nice	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Weight	9548	7620	6100	4904	3906	3121	2501	1991	1586	1277
Nice	0	1	2	3	4	5	6	7	8	9
Weight	1024	820	655	526	423	335	272	215	172	137
Nice	10	11	12	13	14	15	16	17	18	19
Weight	110	87	70	56	45	36	29	23	18	15



# Ejemplo del planificador CFS

	P1	P2	P3	P4
Valor nice	0	-3	2	10
Peso (weight)	1024	1991	820	110
Peso ponderado	26 %	50 %	21 %	3 %

Y la línea de tiempo del planificador que refleja el uso de la CPU de forma similar a nuestros diagramas de Gantt quedaría de la siguiente forma:



# El planificador de Windows 7 y Windows 10

---

- Se implementa con desalojo y prioridades, teniendo al hilo como unidad de planificación.
  - Clase estática para las 16 prioridades más altas (procesos en tiempo real), que sólo puede lanzar el administrador.
  - Clase variable para las 16 prioridades más bajas (aplicaciones de usuario), que pueden subir su prioridad según su interactividad.
- Cada prioridad tiene su cola, que se gestiona por tiempo compartido con quantum fijo. Reglas:
  - Cada vez que se usa un quantum completo, se baja 1 la prioridad.
  - Cuando un proceso se bloquea por E/S, al desbloquearse se tiene en cuenta el tiempo de bloqueo: Teclado y ratón suben 6. HD sube 1.
  - Si un proceso es desalojado por otro de prioridad mayor, se coloca el primero de su cola.

# Bibliografía

---

- Se recomienda la lectura del libro que colocamos en la portada de la asignatura en el Campus Virtual UMA:
  - Operating Systems Concepts, 8th Edition, de los autores A. Silberschatz, G. Gagne, P.B. Galvin, publicado por Wiley en 2008.
- Sus contenidos están disponibles para los estudiantes de la UMA a través del siguiente enlace Web:
  - <https://www.oreilly.com/library/view/operating-system-concepts/9780470128725/?ar>
- Este segundo capítulo está cubierto en la segunda parte del libro, concretamente desde los siguientes capítulos:
  - Chapter 3: Processes
  - Chapter 4: Threads
  - Chapter 5: CPU Scheduling