

PLANIFICACIÓN DE PROCESOS

Se tienen 3 planificaciones: largo, mediano y corto plazo. El más importante y complejo es el de **corto plazo**: acceso al cpu.

Objetivos del SO

Generales para todos los entornos.

Específicos al entorno de ejecución: batch, interactivo y sistemas de tiempo real.

Variables a planificar: tiempos de uso de recursos.

Modelos de evaluación de la planificación: Determinista, probabilístico y prioridades

Algoritmos de planificación según el uso de CPU:

No apropiativos: first come first served (FCFS), shortest job first (SJF) y prioridades

Apropiativos: shortest remaining time first (SRTF), round robin (cola circular), colas multinivel, sistemas en tiempo real (STR)

Planificación según la cantidad de procesadores:

Un procesador con hebras

multiprocesadores (nucleos de CPU)

Analogías de la computación con la vida en un país

Computación	Vida humanos
Computador	País
Sistema operativo	Estado
cpu	presidente
Procesos	personas

Operativamente las personas **somos un: BioCompuBot.**

 Biológico: nace, vive y muere;

 Computador: cerebro con memoria y pensamiento;

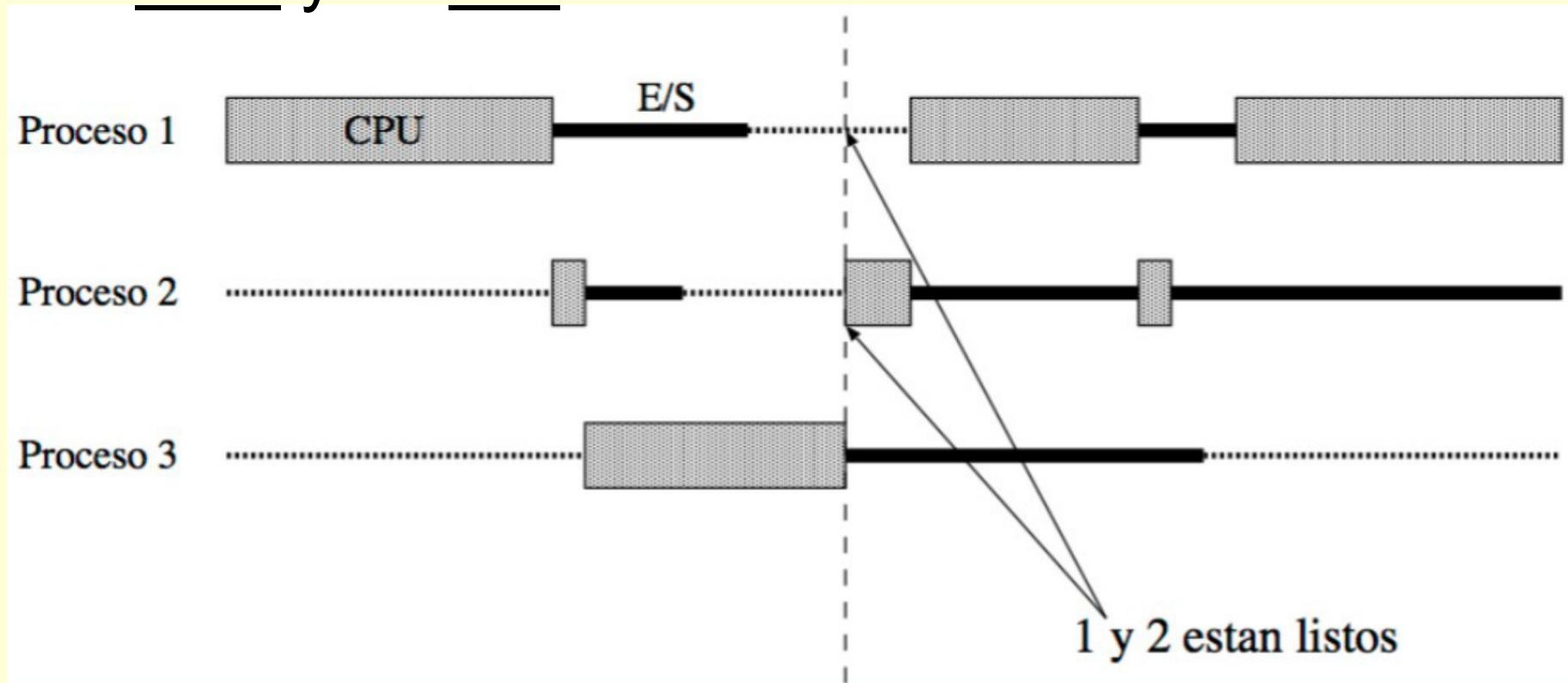
 Robot: hacemos cosas.

Analogías de la computación con la vida en una persona

Computación	Vida humanos
Computador	Yo soy un BioCompuBot
Sistema operativo	Información guardada en lóbulo temporal: Conocimientos, hábitos.
cpu	Lóbulo frontal
Procesos	mis actividades

Planificación

- En un SO multiproceso varios procesos (threads) listos pueden competir por la CPU.
- Los procesos se suelen comportar alternando ráfagas de CPU y de E/S.



Planificación

El módulo de planificación a corto plazo (scheduler) decide a qué proceso preparado se le da paso a CPU.

Existen distintos algoritmos de planificación (scheduling algorithms).

- Planificación no apropiativa (non-preemptive): deja ejecutar al proceso en CPU hasta que se bloquea: (inicio E/S), espera por otro proceso o terminación voluntaria.
- Planificación apropiativa: el planificador puede desalojar al proceso en CPU durante su ejecución y cambiarlo por otro. Necesita una interrupción de reloj para poder ejecutarse en períodos regulares de tiempo (quantum).

Objetivos del planificador

Los objetivos de un planificador varían dependiendo del entorno de aplicación:

- **Entornos de proceso por lotes (batch):** ej. inventarios, cálculo de nóminas, etc. Típicamente planificación no apropiativa, o si no, apropiativa con quantum muy largo. Se denomina planificador a largo plazo (batch) y mayormente decide el orden de los trabajos y el grado de multiprogramación.
- **Entornos interactivos:** entornos gráficos, servidores, etc. Planificación apropiativa para atender distintos procesos concurrentemente (tiempo compartido). Decide también si debe realizar intercambio (swapping).
- **Entornos de tiempo real:** menos dependencia de apropiación del CPU, suelen tener holguras de tiempos y redundancia de recursos (duplicados). Los procesos ejecutan por períodos cortos de tiempo. El sistema operativo y sus procesos suelen ser más limitados y controlados ¿por qué?

Glosario

job (trabajo): es un nombre histórico de un **proceso de tipo batch**

Throughput: productividad o rendimiento =
(Numero de trabajos) / (unidad de tiempo)

Turnaround: Tiempo transcurrido desde que va a cargar a memoria hasta que termina el proceso: $t_R \equiv t_F - t_I$

Objetivos típicos del planificador

En todos los entornos:

- Justicia (fairness): que cada proceso obtenga una porción de CPU “justa” o razonable.
- Política: que se satisfaga un determinado criterio establecido (ej. prioridades).
- Equilibrio: que todas las partes del sistema estén ocupadas haciendo algo.

En sistemas batch:

- Productividad o rendimiento (throughput) = número de trabajos/unidad de tiempo. Intentamos maximizarlo.
- Tiempo de paso o de retorno (turnaround): tiempo transcurrido entre que se lanza un proceso y termina. Intentamos minimizarlo.
- Capacidad de ejecución: $\% (\text{uso CPU}) / \text{turnaround}$.

Objetivos típicos del planificador

En entornos interactivos:

- Minimizar el tiempo de respuesta: tiempo que transcurre entre que el usuario da una orden y se obtiene alguna respuesta.
- Proporcionalidad: cumplir expectativas de usuario (tareas sencillas = tiempo de respuesta corto).
- Mantener un máximo de usuarios interactivos o de clientes.

En sistemas de tiempo real:

- Fiabilidad: evitar perder datos; reaccionar en tiempo límite, etc.
- Predectibilidad: p. ej. evitar degradación de calidad para multimedia: vista y sonido

Medidas de tiempo

- Tiempo de paso o de retorno (turnaround) (t_R) = el total transcurrido desde que se inicia (T_i) hasta que finaliza (T_f).

$$t_R \stackrel{\text{def}}{=} T_f - T_i$$

Incluye:

- Tiempo de carga en memoria
- Tiempo en la cola de preparados
- Tiempo de ejecución en CPU t_{CPU}
- Tiempo en operaciones E/S (bloqueado) $t_{E/S}$
- Tiempo de espera (t_E) es el tiempo de retorno quitando CPU y E/S, $t_E \stackrel{\text{def}}{=} t_R - t_{CPU} - t_{E/S}$

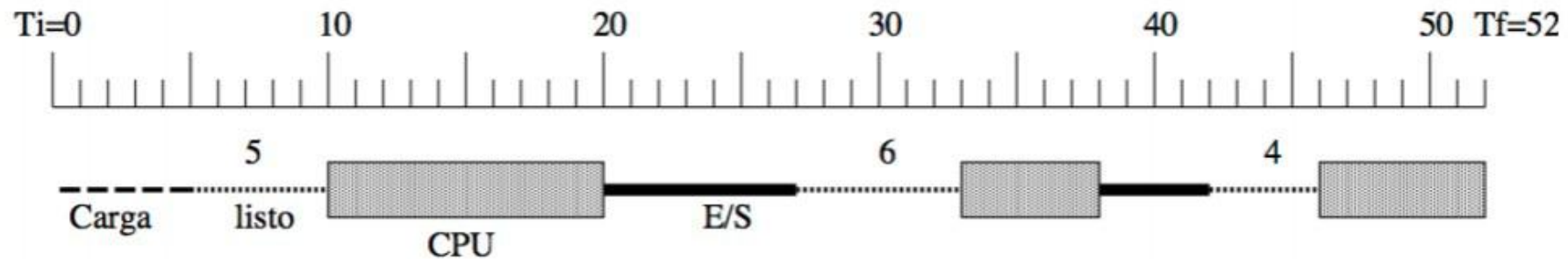
Medidas de tiempo

- Tiempo de servicio (t_s) = Es el tiempo que consumiría si fuese el único proceso existente (y no precisase carga a memoria). Es decir, el tiempo de retorno menos el tiempo de espera.
$$t_s \stackrel{\text{def}}{=} t_R - t_E = t_{CPU} + t_{E/S}$$

- Índice de servicio (i)
$$i_s \stackrel{\text{def}}{=} t_s / t_R$$

Medidas de tiempo

Un ejemplo . . .



- Tiempo de **retorno**: $t_R = T_f - T_i = 52 - 0 = 52$
- Tiempo de **CPU**: $t_{CPU} = 10 + 5 + 6 = 21$
- Tiempo de **E/S**: $t_{E/S} = 7 + 4 = 11$
- Tiempo de **servicio**: $t_S = t_{CPU} + t_{E/S} = 32$
- Tiempo de **espera**: $t_E = t_R - t_S = 52 - 32 = 20$
- **índice de servicio**: $i_S = 32/52 = 0,615$

Evaluación de la planificación

Modelos **deterministas** (utiliza indicadores medibles directamente)

- Toman una carga de trabajo concreta y evalúan los algoritmos sobre ella. Importante: seleccionar casos representativos.
- Comparan los algoritmos en función de alguna de las medidas de rendimiento (ej. tiempo medio de retorno, productividad, etc.).
- Ventajas: sencillos. Proporcionan medidas exactas.
- **Desventaja**: engañosos si la carga de trabajo no es representativa.

Evaluación de la planificación

Modelos **no deterministas** (teoría de colas)

- En muchos sistemas, los trabajos son impredecibles y no es posible usar un modelo determinista.
- Se usan distribuciones de probabilidad para modelar las ráfagas de CPU, de cola de espera para entrar a preparado, de E/S.
- A partir de esas distribuciones se pueden calcular las medias de productividad, tiempo de retorno, tiempos de espera, etc.

Evaluación de la planificación

Modelos **no deterministas** (teoría de colas)

- El computador se describe como una red de “servidores”. Cada servidor tiene una cola de trabajos en espera. La CPU es un servidor de la cola de preparados, así como el sistema de E/S lo es de la cola de E/S.
- Si conocemos los ritmos de llegada y de servicio, podemos calcular la utilización, la longitud media de cola, el tiempo de espera medio, etc. Esto se conoce como análisis de redes de colas

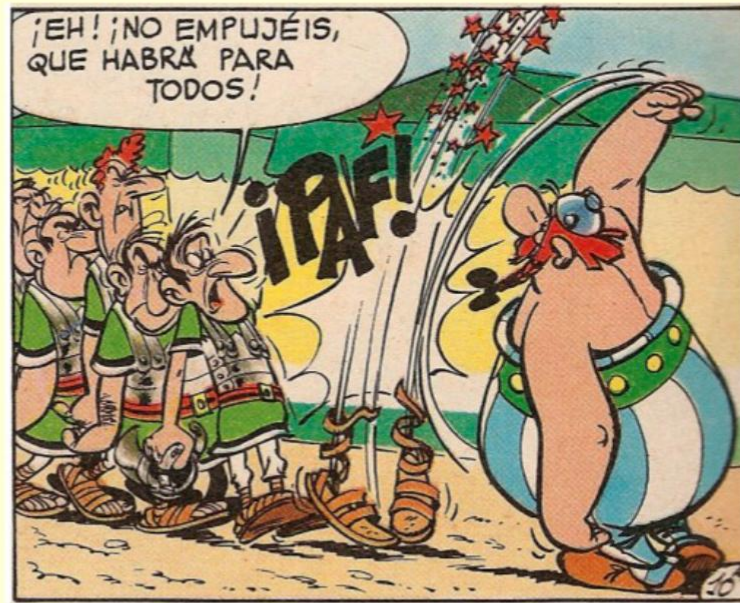
Evaluación de la planificación

Simulación: modelaje en computadora

- . Los datos de procesos y ráfagas se generan aleatoriamente o se obtienen de trazas reales.
- . Permiten una evaluación cercana a casos reales.
- . Tienen alto coste (obtención de datos, tiempo de simulación, mediciones, etc.).

Algoritmos no apropiativos

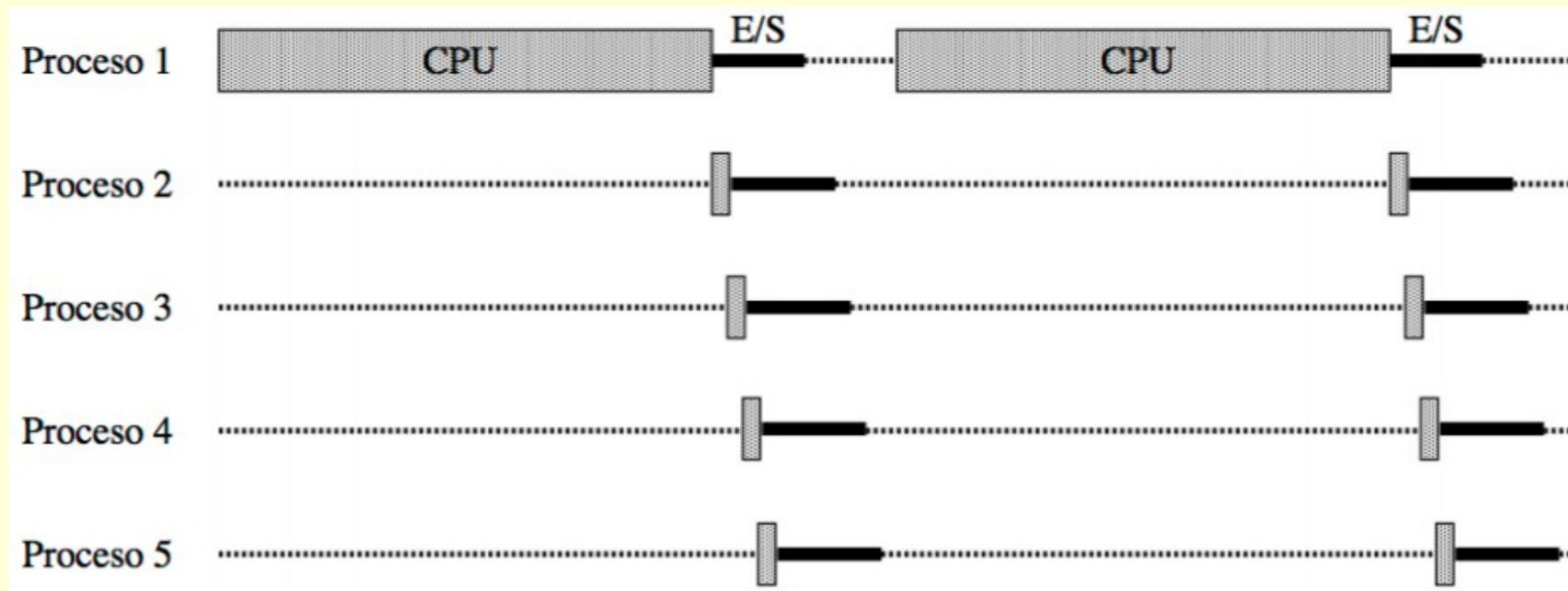
Algoritmo **First-Come-First-Served (FCFS)**: Primero en llegar, primero en ser servido



Ventajas:

- Fácil de implementar. Basta una cola FIFO.
- Es bastante justo, si entendemos que todos los procesos tienen el mismo derecho a usar la CPU.

- Desventaja: puede provocar baja productividad; efecto “convoy”.
- Ejemplo: un proceso que consume mucha CPU y muchos procesos con E/S muy frecuente.
¿por qué?



Algoritmos no apropiativos

Algoritmo **Shortest Job First (SJF)**: Primero el más corto.



- . Tiene sólo utilidad teórica, ya que precisa conocer el tiempo que va a usarse la CPU antes de usarla.
- . Óptimo para minimizar el tiempo de paso a CPU o de retorno (turnaround) con varios procesos listos en llegada simultánea.
- . A tiempos iguales de uso de CPU, se usa FCFS.

- Supongamos 4 procesos con ráfagas entrantes de 8, 4, 4 y 4 ms. Si los colocamos en orden de llegada:

8	4	4	4
---	---	---	---
- P1 tarda 8, P2 tarda $8 + 4 = 12$, P3 tarda $12 + 4 = 16$, P4 tarda $16 + 4 = 20$.
- El plazo de entrega medio es: $(8 + 12 + 16 + 20)/4 = 14$

Usando SJF tenemos

4	4	4	8
---	---	---	---

- El plazo de entrega medio es ahora: $(4 + 8 + 12 + 20)/4 = 11$
- Se puede demostrar que es el **óptimo**.

Demostración

El CPU tiene la **ráfaga** de procesos P1, P2, P3 y P4 con tiempos de ejecución a, b, c y d respectivamente. ¿Cuál es el orden óptimo de procesos para minimizar la suma de los tiempos de servicios?

Orden de Ejecución	Tiempos de servicio incluye los tiempos de espera
P1	a
P2	a + b
P3	a + b + c
P4	a + b + c + d
Tiempo Serv. Total	$4a + 3b + 2c + d$

Para minimizar la suma $4a + 3b + 2c + d$, los pesos (tiempos de ejecución) a, b, c y d deben cumplir: $a \leq b \leq c \leq d$. En este caso conviene el algoritmo Shortest Job First (SJF)

Anomalía del Algoritmo Shortest Job First (SJF)

- Si los procesos llegan en distintos instantes, puede no ser óptimo. Ejemplo: tenemos 5 procesos

Proceso	Duración CPU	instante listo
A	2	0
B	4	0; espera = 2
C	1	3; espera 6-3 = 3
D	1	3; espera 3+1 = 4
E	1	3; espera 4+1 = 5

Secuencia: A B C D E **turnaround para SJF**

$$tr = 2 + \textcolor{red}{2} + 4 + \textcolor{red}{3} + 1 + \textcolor{red}{4} + 1 + \textcolor{red}{5} + 1 = \mathbf{23} \quad // \textcolor{red}{espera}$$

turnaround para el orden B,C,D,E,A.

Secuencia: B C D E A

$$tr = 4 + \textcolor{red}{1} + 1 + \textcolor{red}{2} + 1 + \textcolor{red}{3} + 1 + \textcolor{red}{7} + 2 = \mathbf{20} < \mathbf{23} \quad // \textcolor{red}{espera}$$

Algoritmos no apropiativos

En la práctica, **SJF** se calcula estimando la siguiente ráfaga a CPU en función de las anteriores:

Datos conocidos:

τ_n = valor de la estimación de ráfaga anterior

t_n = última ráfaga real

Estimación de ráfaga siguiente:

$T(o)$ = número de partida.

$\tau(n+1)$ = valor de la estimación

$= \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$ (valor entre t_n y T_n)

con $\alpha \in [0, 1]$ factor de ajuste.

Prioridades

Definición (Prioridad) La prioridad de un proceso es un valor numérico que se usa como factor para determinar si debe entrar en CPU antes que otros.

Tipos de prioridades:

- Internas: asignadas por el SO a partir de información de los Procesos, ej: tiempo en CPU, uso de memoria, ficheros abiertos, relación entre ráfagas CPU y E/S, etc.
- Externas: asignadas al SO: privilegios del usuario o incluso por preferencias del propietario (ej: comando **nice** en UNIX).

El algoritmo SJF es un caso particular del algoritmo por prioridades, con prioridad tiempo de proceso.

Se puede usar de modo apropiativo o no apropiativo.

Prioridades

- Principal **desventaja**: inanición (starvation). Un proceso queda siempre esperando.
- Se suele resolver mediante asignación dinámica de prioridades. Dos ejemplos:
 - Usar como prioridad la fracción q/t_{CPU} donde t_{CPU} es el tiempo de CPU que falta para terminar.
 - Envejecimiento (aging): cuanto más tiempo CPU consume va disminuyendo su prioridad.

Algoritmos apropiativos

Algoritmo Shortest Remaining Time First (SRTF)

- . Es una versión apropiativa de SJF.
- . Cada vez que entran trabajos se interrumpe el actual y se compara el tiempo restante de éste con el de los entrantes.
- . Si hay un trabajo nuevo más corto que lo que le falta al actual en CPU, sale el actual y entra el nuevo.
- . De nuevo, se supone que se conocen los tiempos de uso futuro de CPU de antemano. Una versión práctica debe hacer uso de una estimación.

Algoritmos apropiativos

Algoritmo Round-Robin (RR: cola circular). Es una cola LIFO: Entra al CPU el primero, cuando sale se coloca al final de la cola y se continúa.

- Podemos traducirlo como asignación circular o por torneo.
- Cada proceso tiene un tiempo límite de uso de CPU llamado quantum q .
- Los procesos preparados se organizan en una cola LIFO.

Algoritmos apropiativos

- Algoritmo Round-Robin (RR) Si **A** está ejecutando y alcanza el quantum $\Rightarrow$ cambio de contexto: se pasa el primero de la cola a CPU y se inserta **A** al final de la cola.
- Un temporizador (interrupción de reloj) se encarga de despertar al planificador para que compruebe si debe actuar o no.

Algoritmos apropiativos

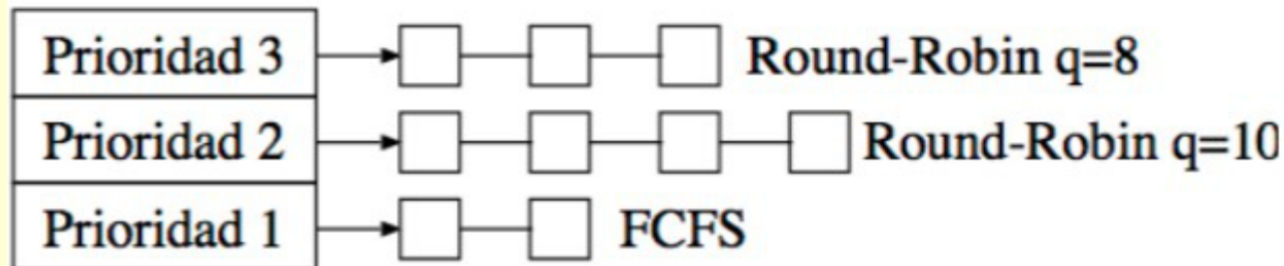
Algoritmo Round-Robin (RR)

- Ventajas: es fácil de implementar. Es el algoritmo más justo: todos los procesos tienen garantizada su porción de CPU.
- Problema: fijar el valor de q es crítico.
 - Si q muy pequeño, provoca muchos cambios de contexto. P.ej. si el cambio tarda ej. 1 ms , y las ráfagas son p.ej. de 4 ms un 20 % del tiempo se desperdicia en tareas S.O.
 - Si q demasiado grande, degenera en un FCFS.
- Empíricamente, da mejor resultado cuando un $\approx 80\%$ de las ráfagas son más cortas que q . Valor habitual $20\text{ ms} \leq q \leq 50\text{ ms}$.

Algoritmos apropiativos

Algoritmo Colas Multinivel

- . Es una elaboración de algoritmo por prioridades.
- . Tenemos una cola por cada nivel de prioridad. Cada cola puede tener su propio algoritmo de planificación.
- . Además, para evitar inanición, se suele calcular dinámicamente la prioridad permitiendo cambio de cola.



Algoritmos apropiativos

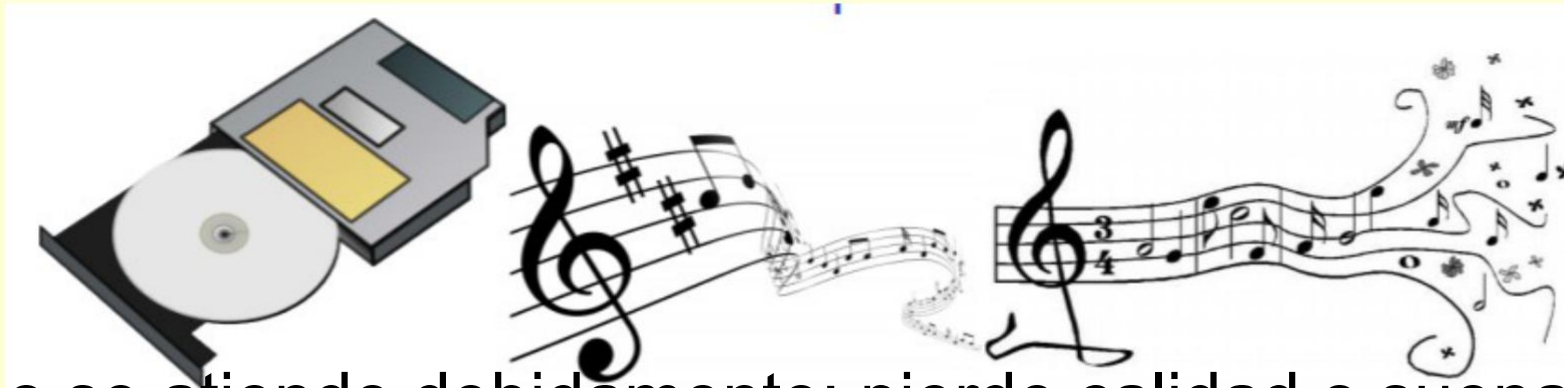
Algoritmo Colas Multinivel

Típicamente:

- Mayores prioridades para procesos del sistema, procesos foreground interactivos, o para procesos con poca CPU usando RR.
- Menor prioridad para procesos background usando FCFS.
- Si entra un trabajo con mayor prioridad, desbanca a los de menor.
- Otra opción: fraccionar el tiempo entre colas (ej. 80 % para la RR y 20 % para la cola FCFS).

Planificación en Sistemas Tiempo Real (STR)

- En STR el tiempo juega un papel crucial.
- Uno o más dispositivos físicos generan eventos y el computador debe reaccionar a ellos dentro de un tiempo limitado.
- Ejemplo: un reproductor de CD toma información del disco y debe ir la convirtiendo en sonido al ritmo preciso.



- Si no se atiende debidamente: pierde calidad o suena raro.

Planificación en Sistemas Tiempo Real (STR)

- . Tiempo real estricto: el plazo de tiempo límite es obligatorio.
- . Tiempo real no estricto: perder un plazo límite es indeseable, pero a veces tolerable.
- . Un programa se suele dividir en distintos procesos cortos y predecibles cuya duración se conoce de antemano (repetitivo).
- . El planificador debe organizar los procesos de modo que se cumplan los plazos límite.
- . En un STR podemos tener eventos:
 - . Periódicos: suceden a intervalos regulares
 - . Aperiódicos: suceden de forma impredecible.

Planificación en Sistemas Tiempo Real (STR)

- Un STR puede tener que responder a varios flujos (streams) de eventos periódicos. Si cada evento requiere mucho tiempo, puede incluso ser inmanejable.
- Supongamos $1, \dots, m$ flujos de eventos periódicos, y en cada flujo i :

C_i = Tiempo CPU para atender el evento, ejm: 30 min

P_i = Período de tiempo de ocurrencia del evento: 60 min

Fracción de utilización = $C_i/P_i = 0.5 \leq 1$;

Definición: Un STR con m flujos es Planificable si satisface:

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

Planificación en Sistemas Tiempo Real

- Ejemplo: 3 flujos con periodos en ms, $P1 = 100$, $P2 = 200$ y $P3 = 500$ y consumos de CPU de $C1 = 50$, $C2 = 30$ y $C3 = 100$.
- La suma da $0.5 + 0.15 + 0.2 = 0.85 < 1$.
- Si añadimos un cuarto flujo con $P4 = 1000$ ¿cuánto podría valer $C4$ como máximo para seguir siendo planificable?

Se debe cumplir: $\text{Suma}(C_i/P_i) \leq 1$,

$$\text{Suma}(C_i/P_i) = C1/P1 + C2/P2 + C3/P3 + C4/P4 \leq 1$$

$$\text{Suma}(C_i/P_i) = .85 + C4/1000 \leq 1$$

Para que $C4$ sea máximo, se debe cumplir:

$$.85 + C4/1000 = 1$$

$$C4 = (.15) * 1000 = 150.$$

Planificación con Hebras

- La hebra (thread) se puede definir como la unidad básica de utilización de CPU.



Ariadna y Teseo (Nicolò Bambini)

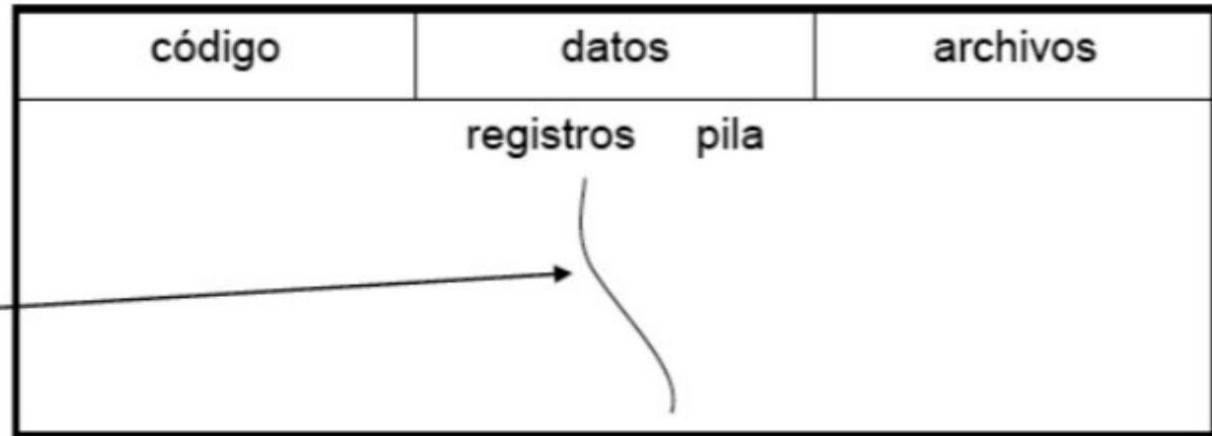
- Un proceso tiene como mínimo una hebra. Si tiene varias, puede realizar varias tareas concurrentemente.

Planificación con Hebras

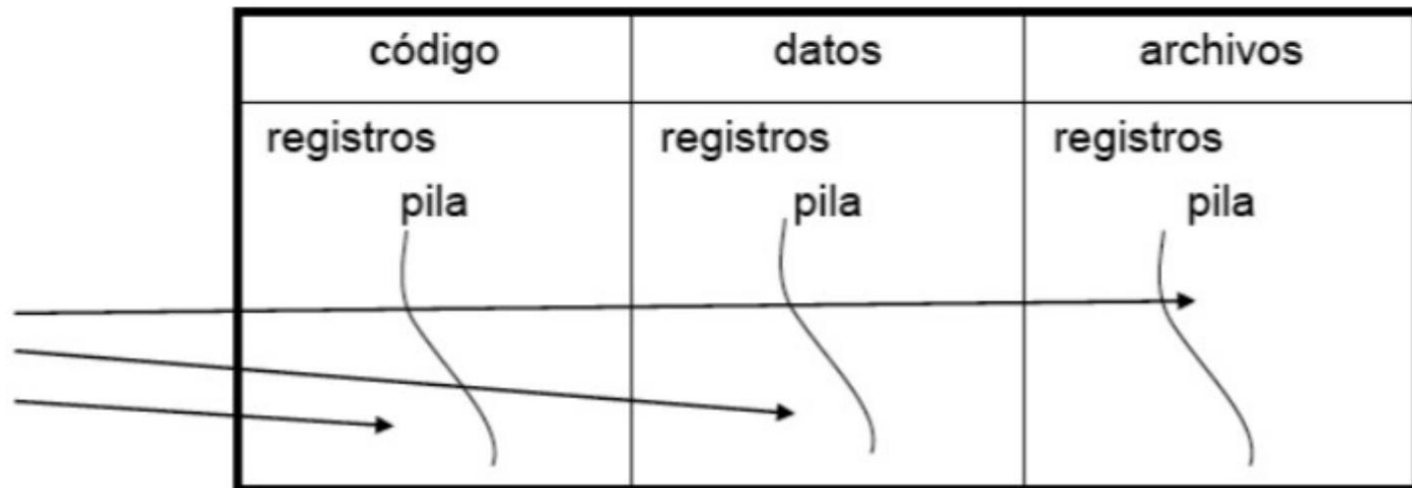
- . Las hebras de un proceso comparten: segmento de código, segmento de datos, recursos (archivos abiertos, señales, etc).
- . Por cada hebra tenemos: identificador, registros, pila.

Planificación con Hebras

Proceso de una
hebra



Proceso
multihebra



Planificación con Hebras

Ventajas:

- . Mayor capacidad de respuesta: si una hebra se bloquea, las demás pueden seguir ejecutándose.
- . Puede haber varias hebras compartiendo los mismos recursos (memoria, ficheros, etc).
- . Menos costoso que crear procesos, el cambio de contexto también es más ligero.
- . Pueden aprovechar arquitecturas multiprocesador.

Planificación con Hebras

- La planificación se separa a dos niveles: procesos y hebras.
- Un planificador de procesos elige el proceso.
Después un planificador de hebra escoge la hebra.
- No existe apropiación entre hebras. Si la hebra agota el quantum del proceso, se salta a otro proceso.
Cuando vuelva, seguirá con la misma hebra.
- Si la hebra no agota el quantum, el planificador de hebra puede saltar a otra hebra del mismo proceso.

Planificación con Multiprocesadores

- Cuando tenemos varios procesadores la planificación se complica.
- Asignar trabajos de distintas duraciones a varios procesadores de forma óptima es un problema combinatorio (complejidad NP: nondeterministic polynomial time).
- Cada procesador suele procesar procesos multi-hebra.

Planificación con Multiprocesadores

El sistema de multiprocesamiento tiene más de un procesador y pueden ejecutar múltiples procesos simultáneamente.

Existen dos tipos de multiprocesamiento: el Simétrico y Asimétrico:
<https://pc-solucion.es/2018/04/18/diferencias-entre-multiprocesamiento-simetrico-y-asimetrico/>

- En el **multiprocesamiento simétrico**, los procesadores **comparten la misma memoria**.
- En el **multiprocesamiento asimétrico** hay un **procesador maestro que controla la estructura de datos** del sistema.

La diferencia principal entre el Multiprocesamiento Simétrico y Asimétrico es que en el Simétrico todos los procesadores son gestionados por el SO; pero en el Asimétrico sólo el maestro es gestionado por el SO y los otros lo asisten en las tareas asignadas

Planificación con Multiprocesadores

- Aunque tengamos varios núcleos idénticos, para una hebra dada, no todos son igual de interesantes: Si una hebra **A** lleva más tiempo en CPU 1, la caché 1 estará llena de datos de **A**. A esto se le llama afinidad.
- Los algoritmos de afinidad funcionan a dos niveles:
 - . Primero asignan un grupo de hebras a cada procesador
 - . Después hacen planificación interna en cada CPU
- Ventaja: máxima afinidad de la caché. Posible problema: dejar CPUs ociosas (se reconsidera la asignación de hebras).

Planificación con Multiprocesadores

- . Equilibrio de carga: busca mantener equilibrada la actividad de las distintas CPUs.
- . Migración imperativa: cada determinado tiempo se comprueba la carga entre procesadores y se impone una migración entre dos de ellos, si es necesario.
- . Migración solicitada: un procesador inactivo extrae proceso a otro.