Sistemas Operativos

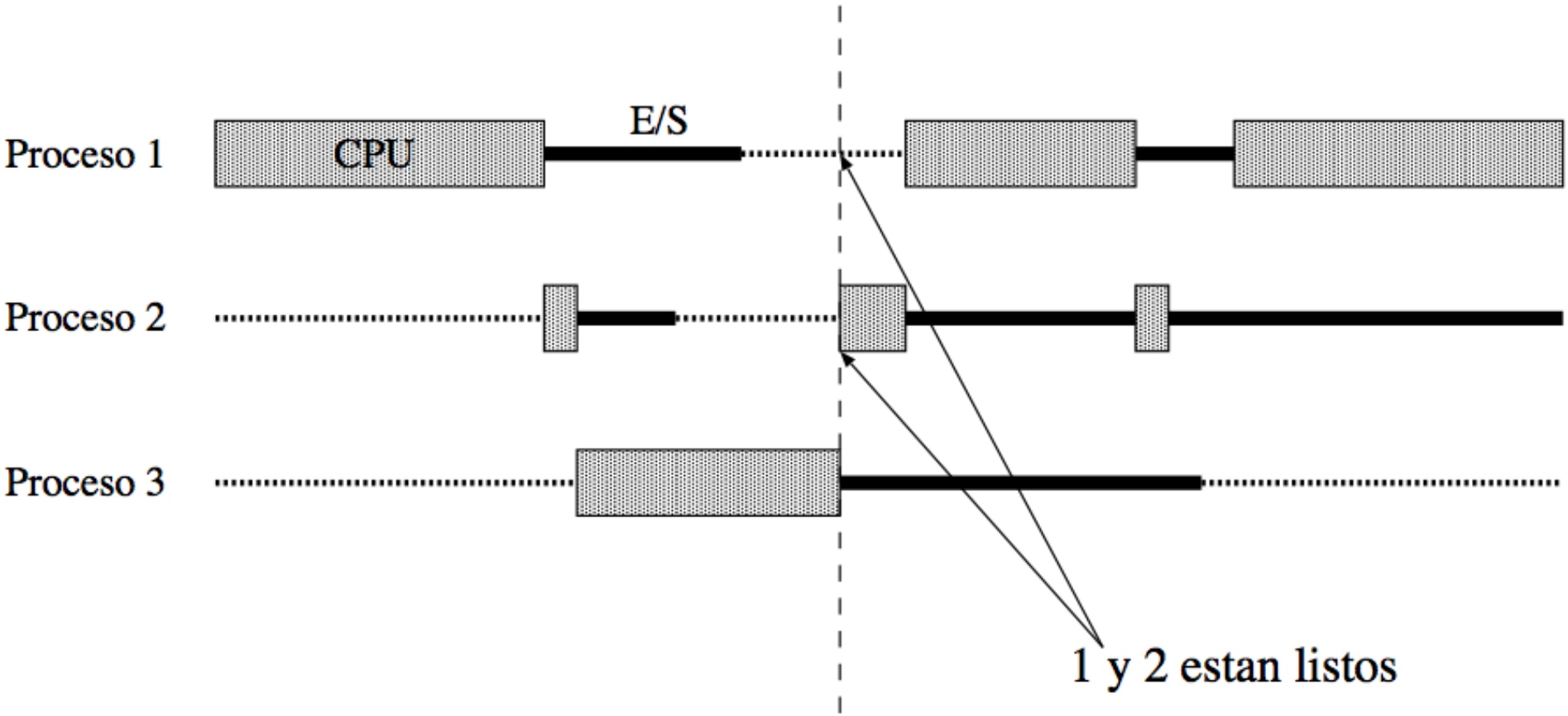
Pedro Cabalar

Depto. de Computación
Universidade da Coruña

TEMA III. PROCESOS

Planificación

- Como vimos, en un SO multiprograma varios procesos o threads listos pueden competir por la CPU.
- Los procesos se suelen comportar alternando ráfagas de CPU y de E/S.



Planificación

Definición (Planificador)

El planificador (scheduler) es la parte del SO que 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 éste para por bloqueo (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 (o de trabajos) 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). Se suele llamar planificador a medio plazo. Decide también si debe realizar intercambio (*swapping*).
- Entornos de tiempo real: menos dependencia de apropiación de CPU. Los procesos no ejecutan por largos períodos de tiempo. El sistema y sus procesos suelen ser más limitados y controlados.

Objetivos típicos del planificador

En todos los entornos:

- Justicia (*fairness*): que el 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: mantener la CPU ocupada todo el rato.

Objetivos típicos del planificador

En entornos interactivos:

- 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=poco tiempo de respuesta).
- Mantener un máximo de usuarios interactivos o de clientes.

En sistemas de tiempo real:

- Fiabilidad: evitar perder datos; reaccionar en tiempo límite, etc.
- Predecibilidad: p. ej. evitar degradación de calidad multimedia.

Medidas de tiempo

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

$$t_R \stackrel{def}{=} Tf - Ti$$

  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, $t_E \stackrel{def}{=} t_R - t_{CPU} - t_{E/S}$.

Medidas de tiempo

- Tiempo de servicio (t_S) = Es el tiempo que consumiría si fuese el único proceso existente (y no precisase carga). Es decir, el tiempo de retorno menos el tiempo de espera.

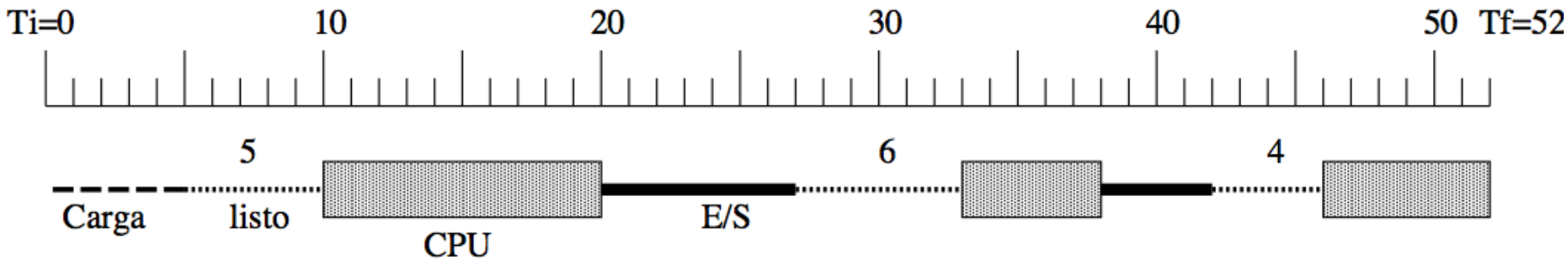
$$t_S \stackrel{def}{=} t_R - t_E = t_{CPU} + t_{E/S}$$

- Índice de servicio (i)

$$i_S \stackrel{def}{=} t_S / t_R$$

Medidas de tiempo

Un ejemplo . . .



- Tiempo de retorno: $t_R = Tf - Ti = 52 - 0 = 52$
- 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: $t_E = t_R - t_S = 52 - 32 = 20$
- Tiempo de índice de servicio: $i_S = 32/52 = 0{,}615$

Evaluación de la planificación

Modelos deterministas

- Tomamos una carga de trabajo concreta y evaluamos los algoritmos sobre ella. Importante: seleccionar casos representativos.
- Comparamos los algoritmos en función de alguna de las medidas de rendimiento (ej. tiempo medio de retorno, productividad, etc).
- Ventajas: sencilla. Proporciona medidas exactas.
- Desventaja: engañosa 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 y los tiempos de llegada de los trabajos al sistema.
- A partir de esas dos 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 sistema informático se describe como una red de “servidores”. Cada servidor tiene una cola de trabajos en espera. La CPU es un servidor de su cola de preparados, así como el sistema de E/S lo es de su cola de dispositivo.
- 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.

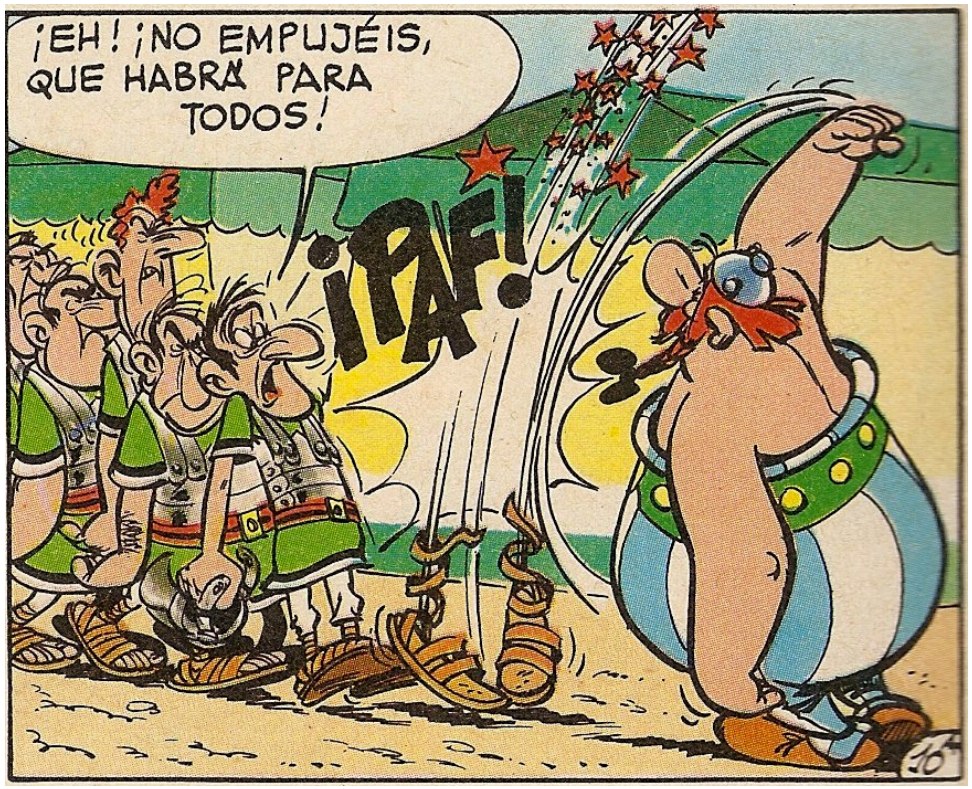
Evaluación de la planificación

Simulación

- Una tercera opción es realizar simulaciones del comportamiento del sistema.
- Los datos de procesos y ráfagas se generan aleatoriamente o se obtienen de trazas reales.
- Permiten una evaluación cercana a casos reales.
- Sin embargo 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



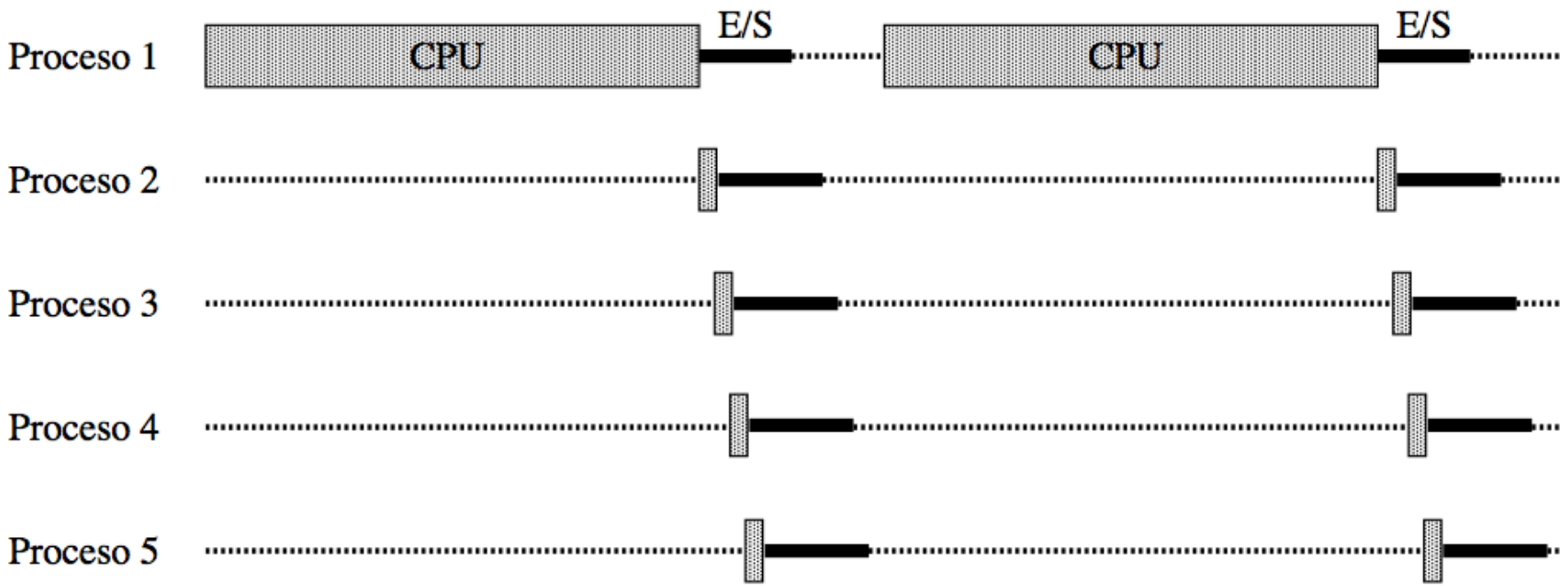
Ventajas:

- Fácil de implementar. Basta una cola FIFO.
- Es bastante justo, si entendemos que procesos con menos CPU tienen menos derecho a usarla.

Algoritmos no apropiativos

Algoritmo First-Come-First-Served (FCFS)

- Desventaja: puede provocar baja productividad; efecto "convoy".
- Ejemplo: un proceso limitado por CPU y muchos procesos con E/S muy frecuente.



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.
- Es el óptimo para minimizar el tiempo de paso o deretorno (*turnaround*) con varios procesos listos en llegada simultánea.
- A tiempos iguales, se usa FCFS.

Algoritmos no apropiativos

Algoritmo Shortest Job First (SJF): Un ejemplo.

- 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 probar que es el óptimo. Ejemplo: con tiempos a, b, c, d la media es $(4 \cdot a + 3 \cdot b + 2 \cdot c + d)/4$. Claramente, a mejor si es el más corto, etc.

Algoritmos no apropiativos

Algoritmo Shortest Job First (SJF)

- Si los procesos llegan en distintos instantes, deja de ser óptimo. Ejemplo para hacer en tutorías: tenemos 5 procesos

Proceso	Duración CPU	instante listo
A	2	0
B	4	0
C	1	3
D	1	3
E	1	3

- Calcula turnaround para SJF y para el orden B,C,D,E,A.

Algoritmos no apropiativos

Algoritmo Shortest Process Next

- En la práctica SJF se modifica usando una estimación de la siguiente ráfaga de CPU en función de las anteriores

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

donde:

$$\begin{aligned} \tau_{n+1} &= \text{valor de la estimación} \\ t_n &= \text{última ráfaga} \\ \tau_n &= \text{valor anterior de la estimación} \\ \alpha &\in [0, 1] \text{ factor de ajuste} \end{aligned}$$

Prioridades

- El algoritmo SJF es un caso particular de algoritmo por prioridades.
- Se puede usar de modo apropiativos o no apropiativo.

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 otro(s).

Tipos de prioridades:

- Internas: asignadas por el S.O. 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 por S.O. (privilegios del usuario) o incluso por preferencias del propietario (ej: comando `nice` en UNIX).

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} fue la última ráfaga.
  - 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, echamos el actual y metemos 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)

- 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 FIFO.

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

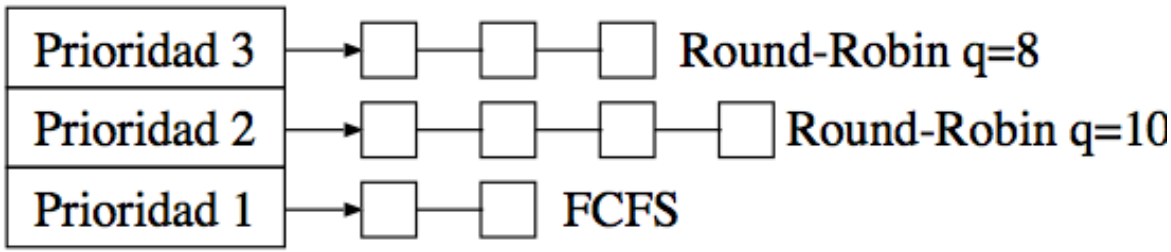
Algoritmo Round-Robin (RR)

- 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 $20\ ms \leq q \leq 50\ ms$.

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 procesos con poca CPU. Usamos RR.
- Menor prioridad para procesos background. Usamos 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

- En sistemas de tiempo real (STR) el tiempo juega un papel crucial.
- Uno o más dispositivos físicos generan estímulos y el ordenador debe reaccionar a ellos dentro de un tiempo limitado.
- Ejemplo: un reproductor de CD toma información del disco y debe irla convirtiendo en sonido al ritmo preciso.

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

Planificación en Sistemas Tiempo Real

- 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.
- 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

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

$$P_i = \text{Período de tiempo con que sucede un evento}$$
$$C_i = \text{Tiempo CPU que cuesta atender un evento}$$

Definición (STR Planificable)

Decimos que 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 $P_1 = 100$, $P_2 = 200$ y $P_3 = 500$ y consumos de CPU de $C_1 = 50$, $C_2 = 30$ y $C_3 = 100$ todo en *ms*.
- La suma da $0,5 + 0,15 + 0,2 < 1$.
- Si añadimos un cuarto flujo con $P_4 = 1000$ ¿cuánto podría valer C_4 como máximo para seguir siendo planificable?

Planificación con Hebras

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

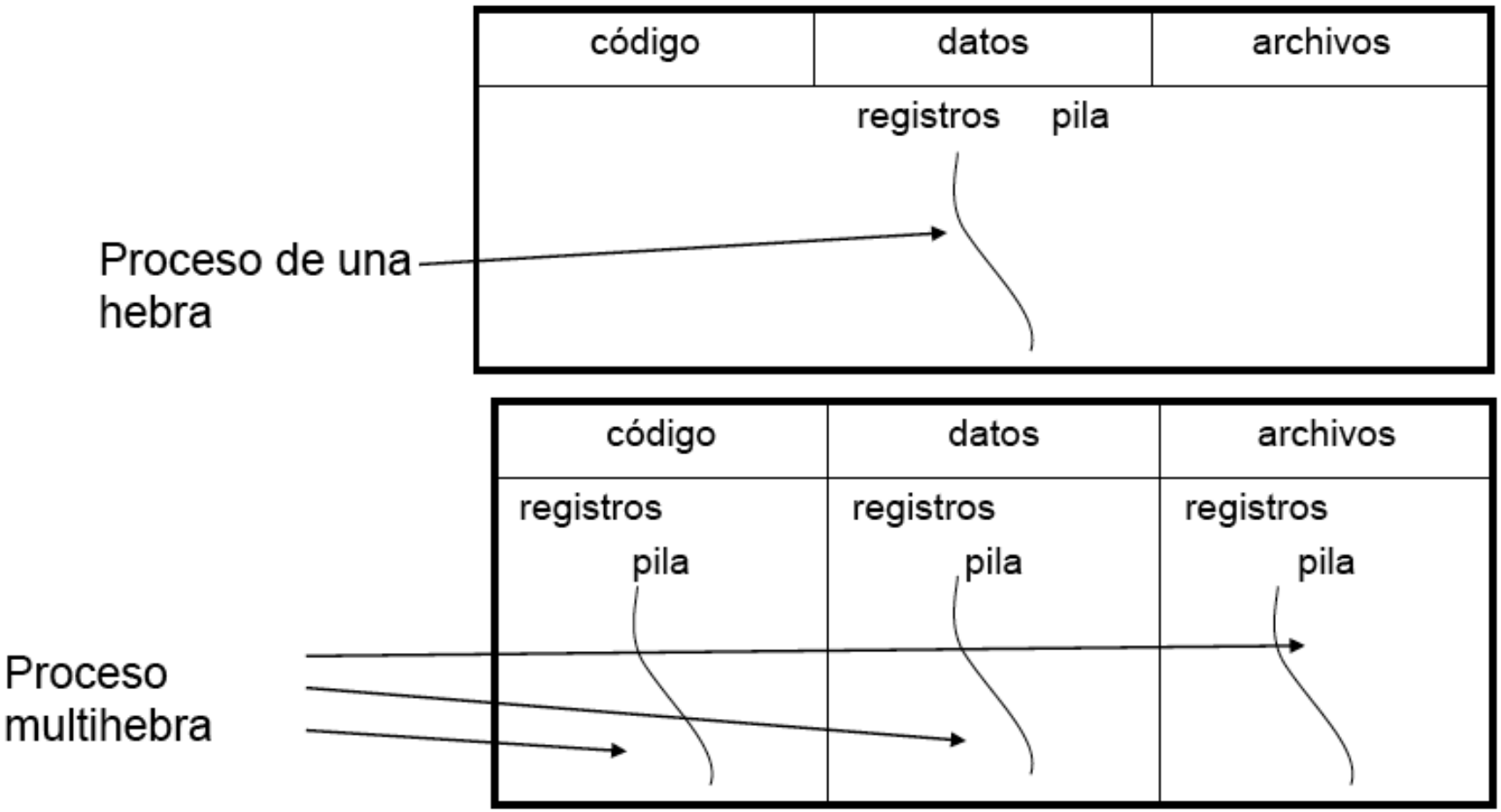
Ariadna y Teseo (Nicolò Bambini)

- 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, contador de programa, registros, pila.

Planificación con Hebras



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; 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 Multirocesadores

- 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).
- Típicamente son además sistemas multi-hebra.
- Multiprocesamiento simétrico: un procesador toma la decisiones y el resto se limita a ejecutar los procesos.
- Multiprocesamiento asimétrico: cada procesador tiene su propia planificación. Si comparten la cola de preparados, hay que controlar que cada proceso entre en un único procesador.

Planificación con Multirocesadores

- 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:
  1. Primero asignan un grupo de hebras a cada procesador
  2. 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 Multirocesadores

- Equilibrado 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.