

# TEMA 4 – PLANIFICACION

## Bibliografía

Este documento se ha elaborado, principalmente, a partir de las referencias bibliográficas que se exponen a continuación, y con propósito meramente académico. El alumno debería completar el contenido de este tema con bibliografía existente en la biblioteca y mediante recursos de la Web, hasta que los conceptos que se indican en el mismo sean afianzados correctamente.

[STALLINGS] W. Stallings. Sistemas operativos. 5ª edición, Prentice Hall, Madrid, 2005.


[TANENBAUM] A. S. Tanenbaum. Sistemas operativos modernos, 3a edición. Prentice Hall, Madrid, 2009.

## Contenido

- Tipos de planificación del procesador (Capítulo 9, sección 9.1 [Stallings]).
- Objetivos de la planificación (Recursos de la Web).
- Criterios para la planificación (Recursos Web y Capítulo 9, sección 9.2 [Stallings]).
- Modos de difusión: sin expulsión o con expulsión (Capítulo 9, sección 9.2 [Stallings], Capítulo 2, sección 2.4.1 [Tanenbaum]).
- Algoritmos de planificación (Capítulo 9, sección 9.2 [Stallings]).
  - Uso de prioridades
  - FCFS
  - RR Y VRR
  - SPN
  - SRT
  - Colas multinivel con retroalimentación.
  - FSS
- Planificación tradicional en UNIX (Capítulo 9, sección 9.3 [Stallings]).
- Planificación multiprocesador (Capítulo 10, sección 10.1 [Stallings]).

## Objetivos de la planificación

Existen diferentes objetivos que se tratan de conseguir con la planificación. Muchos de ellos son contradictorios entre sí, y hay que llegar a soluciones de compromiso. Hay que indicar que la naturaleza de cada sistema determinará qué objetivos son los más importantes. Algunos de los más relevantes son:

- **Justicia.** El algoritmo de planificación debe de dar una porción de tiempo de CPU justa a todos los procesos. Nadie debe de acaparar la CPU o quedar sin servicio. 
- **Productividad.** La productividad es la cantidad de trabajo desarrollada por unidad de tiempo. El objetivo es maximizar la productividad. Este objetivo es igual a la optimización de la utilización del procesador.
- **Tiempo de respuesta aceptable.** En sistemas de tiempo compartido es muy importante que todos los usuarios tengan un tiempo de respuesta aceptable.
- **Predecible.** El algoritmo debe de obtener resultados de planificación (en cuanto a métricas) parecidos sea cual sea la carga del sistema.
- **Costo extra.** El tiempo utilizado en la planificación debe ser el mínimo posible, ya que es tiempo inútil para el sistema. ~~Prioridades.~~ En algunas ocasiones se puede establecer un sistema de prioridades en función de la importancia de dar servicio a cada trabajo para el sistema completo.
- **Ocupación de los recursos.** Se debe de maximizar el uso de los recursos del sistema. Por ejemplo, puede ser útil servir a un proceso de menor prioridad, pero que retiene un recurso crítico o muy solicitado dentro del sistema. También se puede dar más prioridad a procesos que requieran recursos poco usados.
- **Aplazamiento indefinido.** Se debe de evitar la inanición de los procesos. Esto ocurre cuando un proceso no recibe nunca servicio por parte del procesador.
- **Degradación aceptable.** El algoritmo o algoritmos de planificación deben de procurar una degradación del rendimiento del sistema forma suave, a medida que sube la carga del sistema, y sin que ocurra una caída global.

A medida que aumentan el num de procesos, el procesador debe tener la capacidad de introducir esos procesos en memoria poco a poco. Utilizando la lista de nuevos sin cargarlos en la lista de listos de memoria

## Criterios para la planificación

Los administradores de los centros grandes de cómputo se basan en una serie de métricas para verificar el desempeño de sus sistemas, las cuales también sirven para comparar el rendimiento de los diversos algoritmos de planificación:

- **Rendimiento o Productividad (throughput):** Número de trabajos por unidad de tiempo que completa el sistema. Es mejor terminar 50 trabajos por unidad de tiempo que terminar 40. Esta medida depende mucho de la "longitud" de los procesos.
- **Tiempo de retorno o de estancia (turnaround time):** Es el tiempo estadísticamente promedio desde el momento en que se ~~envía~~ <sup>crea</sup> un trabajo, hasta el momento en que se completa (finaliza). Incluye el tiempo de ejecución efectiva, el tiempo de espera, tiempos de E/S, etc. Este parámetro mide cuánto tiempo tiene que esperar el usuario promedio la salida. He aquí la regla: lo pequeño es bello. Un algoritmo de planificación que maximiza el rendimiento no necesariamente minimiza el tiempo de retorno. Por ejemplo, dada una mezcla de trabajos cortos y largos, un planificador que siempre ha ejecutado trabajos cortos y nunca ejecutó trabajos largos podría lograr un rendimiento excelente (muchos trabajos cortos por hora), pero a expensas de un terrible tiempo de retorno para los trabajos largos. Si los trabajos cortos llegaran a un tiempo bastante estable, los trabajos largos tal vez nunca se ejecutarían, con lo cual el tiempo de respuesta promedio se volvería infinito, a la vez que se lograría un rendimiento alto.
- **Tiempo de espera.** El problema del criterio del tiempo de retorno es que incluye muchos

Llamado  
LATENCIA

factores que no dependen del algoritmo de planificación. Por ello definimos el tiempo de espera, que incluye únicamente el tiempo que el proceso está esperando a ser servido, o lo que es lo mismo, el tiempo de espera hasta que se le concede el procesador. El objetivo es minimizar el tiempo medio de espera. Suma de tiempos de los procesos que estan en espera sin usar el procesador

- **Tiempo de respuesta (response time):** Tiempo que transcurre desde que se envía o se lanza un proceso hasta que se termina de ejecutar la primera instrucción (tiempo de obtención de los primeros resultados). Es una medida buena desde el punto de vista del usuario, al cual le interesa empezar a obtener resultados (aunque no esten completos) lo más pronto posible. Ejemplo: Carga de un pdf de gran tamaño: Al principio se cargan las primeras hojas, pero el documento entero tarda en cargarse o mostrarse algo más de tiempo. El objetivo es minimizar el tiempo de respuesta. Tiempo que trascurre desde que se envía o lanza un proceso hasta que se obtienen los primeros resultados

Llamado  
TIEMPO DE  
SERVICIO

- **Utilización de la CPU o eficacia:** Porcentaje de tiempo que está ocupada la CPU de forma útil. En sistemas de usuario normales no es una métrica importante. Si lo es en grandes sistemas de cómputo que deben tratar gran cantidad de datos o realizar constantemente muchas operaciones matemáticas (en estos sistema mientras menos tiempo esté la CPU ociosa mejor). Tiempo que esta ejecutando un proceso en el procesador

## Modos de decisión

Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj, o lo que es lo mismo, los instantes de tiempo en que se ejecuta la selección de un proceso: apropiativos o sin expulsión y no apropiativos o con expulsión.

No apropiativo

### **Sin expulsión o apropiativos (nonpreemptive)**

Es el propio proceso el que se bloquea, de lo contrario no se expulsaría de CPU hasta que termina

El planificador selecciona un proceso para ejecutarlo y después deja que se ejecute (estado ejecutando) hasta que el mismo proceso se bloquea (ya sea en espera de una operación de E/S o de algún otro proceso, o en espera a una petición de servicio al sistema operativo), termina, o hasta que libera la CPU en forma voluntaria.

Incluso aunque se ejecute durante horas, no se suspenderá de manera forzosa. No se toman decisiones de planificación durante las interrupciones que se produzcan. Una vez que se haya completado el procesamiento de la interrupción, se reanuda la ejecución del proceso que estaba antes de la interrupción.

El procesador sacara al proceso ejecutando cuando se termine su rodaja de tiempo y se pasara a listos hasta su nuevo turno, tambien se sacaria si necesita una interrupcion de E/S

### **Con expulsión o apropiativos (preemptive)**

El planificador selecciona un proceso para ejecutarlo y la decisión de expulsar se toma si se ha ejecutado por un máximo de tiempo fijo (rodaja de tiempo), cuando llega un nuevo proceso con mayor prioridad, cuando llega una interrupción que pasa un proceso de bloqueado a estado de listo. Si sigue en ejecución hasta el final del intervalo o rodaja de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo (si hay uno disponible).

Las políticas expulsivas tienen mayor sobrecarga que las no expulsivas (muchos cambios de contexto), pero pueden proporcionar mejor servicio a la población total de procesos, ya que previenen que cualquier proceso pueda monopolizar el procesador durante mucho tiempo y produzca inanición entros procesos. Además, el coste de la expulsión puede resultar relativamente bajo a través de la utilización de mecanismos eficientes de cambios de proceso (tanta ayuda del hardware como sea posible) y proporcionando una gran cantidad de memoria principal para dejar un alto porcentaje de programas en la memoria principal y hacer los cambios de contexto rápidamente.

A medida que se describan las políticas de planificación se utilizará el conjunto de procesos de la Tabla 9.4 como ejemplo de ejecución.

**Tabla 9.4.** Ejemplo de planificación de procesos.

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

## Tipos de planificación del procesador

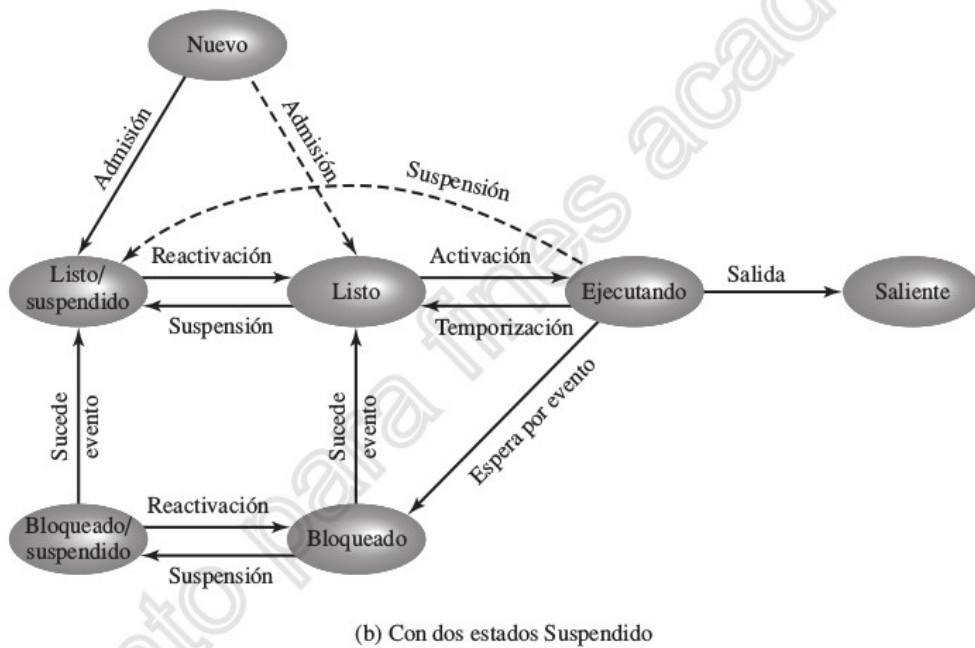
El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, el rendimiento y la eficiencia del procesador. En muchos sistemas, esta actividad de planificación se divide en tres funciones independientes: planificación a largo-, medio-, y corto-plazo. Los nombres sugieren las escalas de tiempo relativas con que se ejecutan estas funciones. La tabla 9.1 muestra los tipos de planificación.

### IMPORTANTE

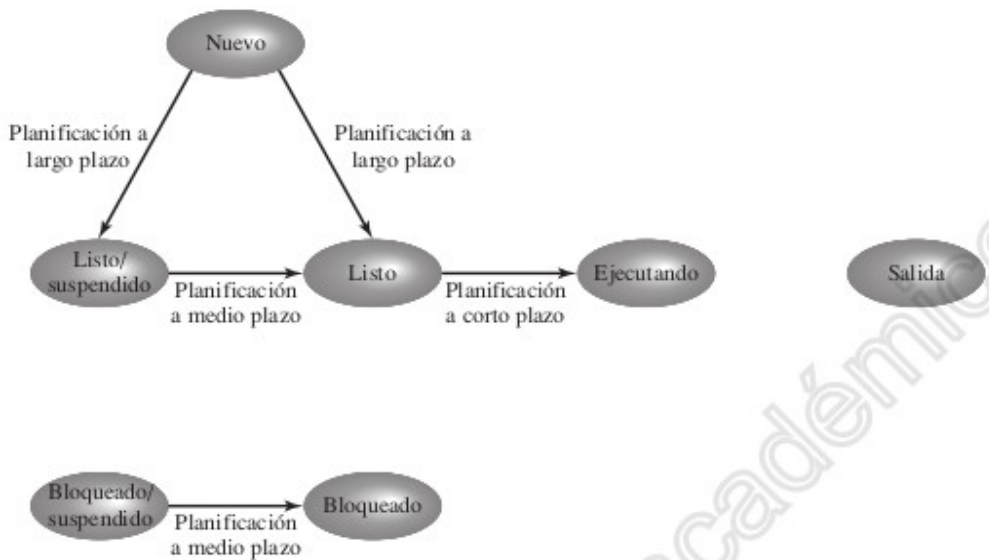
**Tabla 9.1.** Tipos de planificación.

Planificación a largo plazo	La decisión de añadir un proceso al conjunto de procesos a ser ejecutados
Planificación a medio plazo	La decisión de añadir un proceso al número de procesos que están parcialmente o totalmente en la memoria principal
Planificación a corto plazo	La decisión por la que un proceso disponible será ejecutado por el procesador
Planificación de la E/S	La decisión por la que un proceso que está pendiente de una petición de E/S será atendido por un dispositivo de E/S disponible

La Figura 9.1 relaciona las funciones de planificación con el diagrama de transición de estados de los procesos (anteriormente visto en la Figura 3.9b). La planificación a largo plazo se realiza cuando se crea un nuevo proceso. Hay que decidir si se añade un nuevo proceso al conjunto de los que están activos actualmente. La planificación a medio plazo es parte de la función de intercambio (swapping function). Hay que decidir si se añade un proceso a los que están al menos parcialmente en memoria y que, por tanto, están disponibles para su ejecución. La planificación a corto plazo conlleva decidir cuál de los procesos listos para ejecutar es ejecutado. La Figura 9.2 reorganiza el diagrama de transición de estados de la Figura 3.9b para que se pueda apreciar el anidamiento de las funciones de planificación.

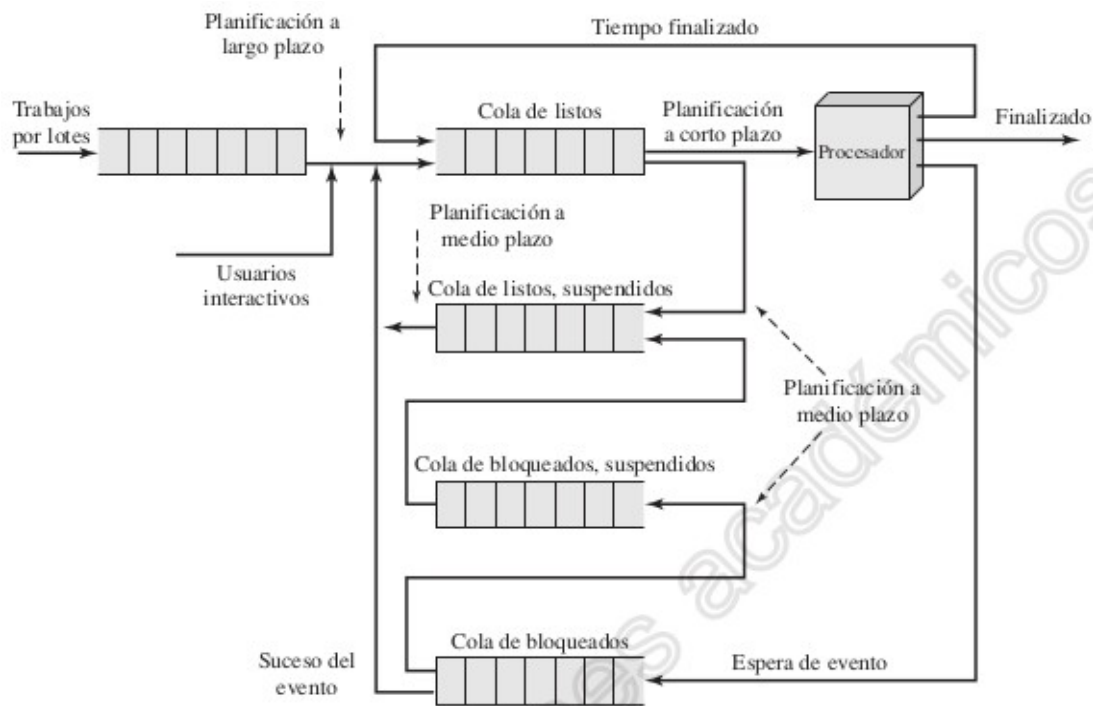


**Figura 3.9.** Diagrama de transición de estados de procesos con estado suspendidos.



**Figura 9.1.** Planificación y transiciones de estado de los procesos.

La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y qué proceso progresará. Este punto de vista se presenta en la Figura 9.3, donde se muestran las colas involucradas en los estados de transición de los procesos 1. Fundamentalmente, la planificación es un problema de manejo de colas para minimizar el retardo en la cola y para optimizar el rendimiento en un entorno de colas.



**Figura 9.3.** Diagrama de encolamiento para la planificación.

### Planificación a largo plazo

El planificador a largo plazo determina qué programas se admiten en el sistema para su procesamiento. De esta forma, se controla el grado de multiprogramación. Una vez admitido, un trabajo o programa de usuario se convierte en un proceso y se añade a la cola del planificador a corto plazo. En algunos sistemas, un proceso de reciente creación comienza en la zona de intercambio, en cuyo caso se añaden a la cola del planificador a medio plazo.

La decisión de cuándo crear un nuevo proceso se toma dependiendo del grado de multiprogramación deseado. Cuanto mayor sea el número de procesos creados, menor será el porcentaje de tiempo en que cada proceso se pueda ejecutar (es decir, más procesos compiten por la misma cantidad de tiempo de procesador). De esta forma, el planificador a largo plazo puede limitar el grado de multiprogramación a fin de proporcionar un servicio satisfactorio al actual conjunto de procesos. Cada vez que termine un trabajo, el planificador puede decidir añadir uno o más nuevos trabajos. Además, si la fracción de tiempo que el procesador está ocioso excede un determinado valor, se puede invocar al planificador a largo plazo.

La decisión de qué trabajo admitir el siguiente, puede basarse en un sencillo «primero en llegar primero en servirse», o puede ser una herramienta para gestionar el rendimiento del sistema. El criterio utilizado puede incluir la prioridad, el tiempo estimado de ejecución y los requisitos de E/S. Por ejemplo, si la información está disponible, el planificador puede intentar encontrar un compromiso entre procesos limitados por el procesador y procesos limitados por la E/S<sup>1</sup>. Además, la decisión puede ser tomada dependiendo de los recursos de E/S que vayan a ser utilizados, de forma que se intente equilibrar el uso de la E/S.

<sup>1</sup> Se dice que un proceso está limitado por el procesador si realiza mucho trabajo computacional y ocasionalmente usa los dispositivos de E/S. Se dice que un proceso está limitado por la E/S si se pasa más tiempo utilizando los dispositivos de E/S que el procesador.

## Planificación a medio plazo

La planificación a medio plazo es parte de la función de intercambio. Con frecuencia, la decisión de intercambio se basa en la necesidad de gestionar el grado de multiprogramación. En un sistema que no utiliza la memoria virtual, la gestión de la memoria es también otro aspecto a tener en cuenta. De esta forma, la decisión de meter un proceso en la memoria, tendrá en cuenta las necesidades de memoria de los procesos que están fuera de la misma.

## Planificación a corto plazo

En términos de frecuencia de ejecución, el planificador a largo plazo ejecuta con relativamente poca frecuencia y toma la decisión de grano grueso de admitir o no un nuevo proceso y qué proceso admitir. El planificador a medio plazo se ejecuta más frecuentemente para tomar decisiones de intercambio. El planificador a corto plazo, conocido también como activador, ejecuta mucho más frecuentemente y toma las decisiones de grano fino sobre qué proceso ejecutar el siguiente.

El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro. Algunos ejemplos de estos eventos son:

- Interrupciones de reloj. (Por rodaja de tiempo)
- Interrupciones de E/S. (De fuera hacia dentro)
- Llamadas al sistema. (Llamadas de E/S)
- Señales (por ejemplo, semáforos). (Como semWait)

## Algoritmos de planificación

Se habla del algoritmo de colas de prioridades.

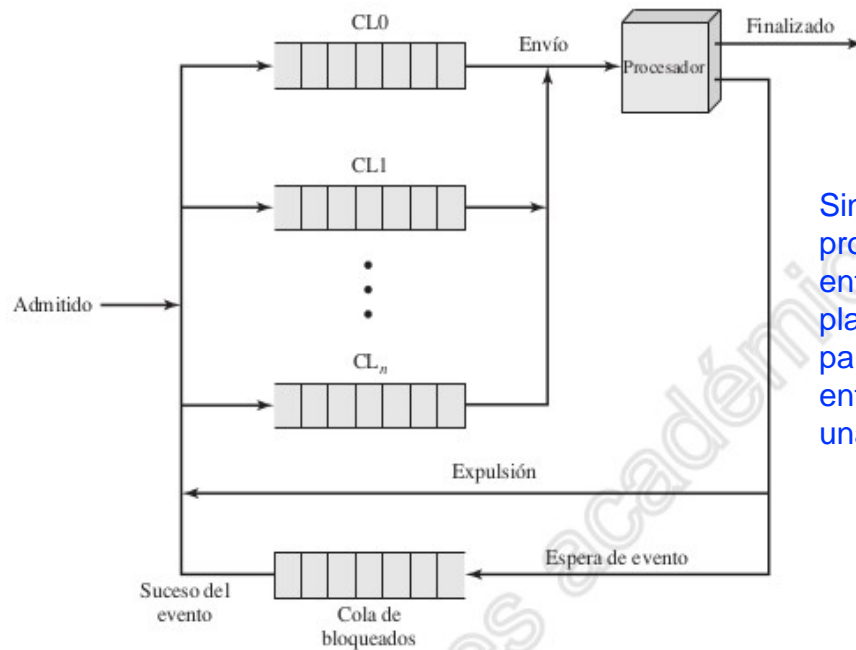
En muchos sistemas, a cada proceso se le asigna una prioridad y el planificador siempre elegirá un proceso de prioridad mayor sobre un proceso de prioridad menor. La Figura 9.4 muestra el uso de las prioridades. Por claridad, el diagrama de colas está simplificado, ignorando la existencia de múltiples colas bloqueadas y de estados suspendidos (comparar con la Figura 3.8a). En lugar de una sola cola de procesos listos para ejecutar, se proporcionan un conjunto de colas en orden descendente de prioridad: CL0, CL1, ... CLn, con la prioridad[CLi] > prioridad[CLj] para  $i < j$ <sup>2</sup>. Cuando se va a realizar una selección en la planificación, el planificador comenzará en la cola de listos con la prioridad más alta (CL0). Si hay uno o más procesos en la cola, se selecciona un proceso utilizando alguna política de planificación. Si CL0 está vacía, entonces se examina CL1, y así sucesivamente.

Un problema de los esquemas de planificación con prioridades es que los procesos con prioridad más baja pueden sufrir inanición. Esto sucederá si hay siempre un conjunto de procesos de mayor prioridad listos para ejecutar.

---

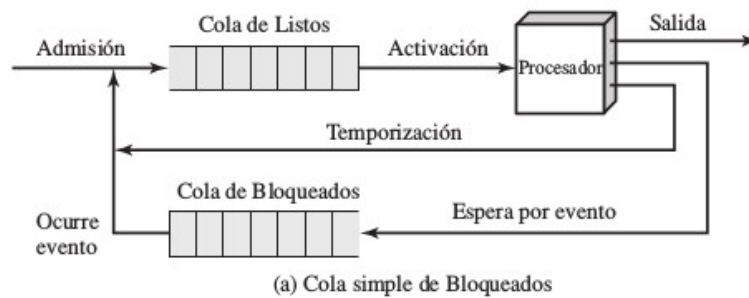
2 En UNIX y otros muchos sistemas, los valores mayores de prioridad representan procesos de prioridad más baja; a menos que se especifique, seguiremos esta convención. Algunos sistemas, tales como Windows, utilizan la convención opuesta: un número mayor significa una mayor prioridad



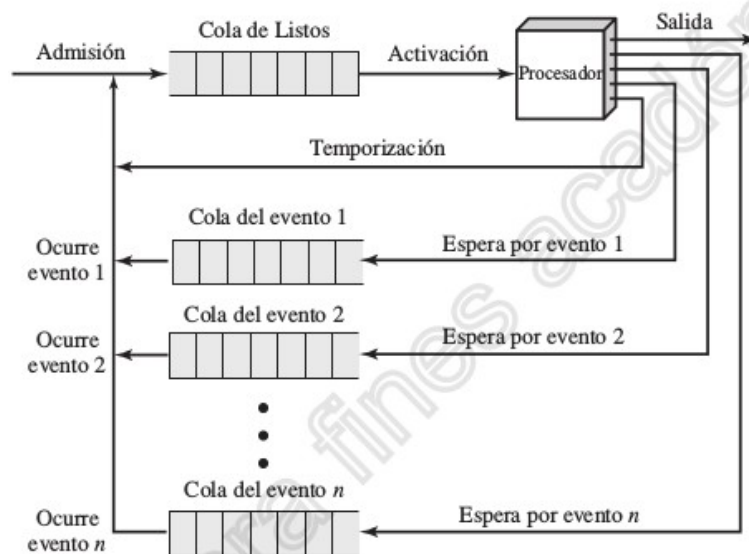


**Figura 9.4.** Encolamiento con prioridades.

Sin expulsión, los procesos se ejecutan enteros y podría planificarse con expulsión para dividir el tiempo entre los procesos de una misma cola.



(a) Cola simple de Bloqueados



(b) Múltiples colas de Bloqueados

**Figura 3.8.** Modelo de colas de la Figura 3.6.



## Primero en llegar, primero en servirse (first-come-first-served, FCFS)

La directiva de planificación más sencilla es primero en llegar primero en servirse (FCFS), también conocida como primero entra-primero-sale (FIFO) o como un sistema de colas estricto. En el momento en que un proceso pasa al estado de listo, se une a la cola de listos. Cuando el proceso actualmente en ejecución deja de ejecutar, se selecciona para ejecutar el proceso que ha estado más tiempo en la cola de listos.

FCFS funciona mucho mejor para procesos largos que para procesos cortos. Considérese el siguiente ejemplo:

Proceso	Tiempo de Llegada	Tiempo de Servicio ( $T_s$ )	Tiempo de Comienzo	Tiempo de Finalización	Tiempo de Estancia ( $T_e$ )	T. final - t. llegada
						Tiempo de estancia/residencia normalizado $T_e/T_s$
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1,99
Media					100	26

El tiempo de estancia normalizado para el proceso Y es excesivamente grande en comparación con los otros procesos: el tiempo total que está en el sistema es 100 veces el tiempo requerido para su proceso. Esto sucederá siempre que llegue un proceso corto a continuación de un proceso largo. Por otra parte, incluso en este ejemplo extremo, los procesos largos no van mal. El proceso Z tiene un tiempo de estancia de casi el doble que Y, pero su tiempo de residencia normalizado está por debajo de 2.0.

Otro problema de FCFS es que tiende a favorecer procesos limitados por el procesador sobre los procesos limitados por la E/S. Considere que hay una colección de procesos, uno de los cuales está limitado por el procesador y un número de procesos limitados por la E/S. Cuando el proceso limitado por el procesador está ejecutando, el resto de los procesos debe esperar. Alguno de estos estarán en las colas de E/S (estado bloqueado) pero se pueden mover a la cola de listos mientras que el proceso limitado por el procesador sigue ejecutando. En este punto, la mayor parte de los dispositivos de E/S pueden estar ociosos, incluso aunque exista trabajo potencial que pueden hacer. Cuando el proceso actual en ejecución deja el estado Ejecutando, los procesos listos pasarán al estado de Ejecutando y se volverán a bloquear en un evento de E/S. Si el proceso limitado por el procesador está también bloqueado, el procesador se quedará ocioso. De esta manera, FCFS puede conllevar usos ineficientes del procesador y de los dispositivos de E/S.

FCFS no es una alternativa atractiva por sí misma para un sistema uniprocador. Sin embargo, a menudo se combina con esquemas de prioridades para proporcionar una planificación eficaz. De esta forma, el planificador puede mantener varias colas, una por cada nivel de prioridad, y despachar dentro de cada cola usando primero en llegar primero en servirse. Veremos un ejemplo de este sistema más adelante, cuando hablemos de la planificación retroalimentada (feedback).

## Turno rotatorio (round robin)

no es interrupción de E/S sino que lo  
saca de CPU

Una forma directa de reducir el castigo que tienen los trabajos cortos con FCFS es la utilización de expulsión basándose en el reloj. La política más sencilla es la del turno rotatorio, también denominada planificación cíclica. Se genera una interrupción de reloj cada cierto intervalo de tiempo. Cuando sucede la interrupción, el proceso actual en ejecución se sitúa en la cola de listos, y se selecciona el siguiente trabajo según la política FCFS. Esta técnica es también conocida como

cortar el tiempo (time slicing), porque a cada proceso se le da una rodaja de tiempo antes de ser expulsado.

En quantos muy pequeños, Existen muchos cambios de BCP y sobrecarga

Con la planificación en turno rotatorio, el tema clave de diseño es la longitud del quantum de tiempo, o rodaja, a ser utilizada. Si el quantum es muy pequeño, el proceso se moverá por el sistema relativamente rápido. Por otra parte, existe una **sobrecarga de procesamiento** debido al manejo de la interrupción de reloj y por las funciones de planificación y activación. De esta forma, se deben evitar los quantums de tiempo muy pequeños. Una buena idea es que el quantum de tiempo debe ser ligeramente mayor que el tiempo requerido para una interacción o una función típica del proceso. Si es menor, muchos más procesos necesitarán, al menos, dos quantums de tiempo. La Figura 9.6 muestra el efecto que tiene en el tiempo de respuesta.

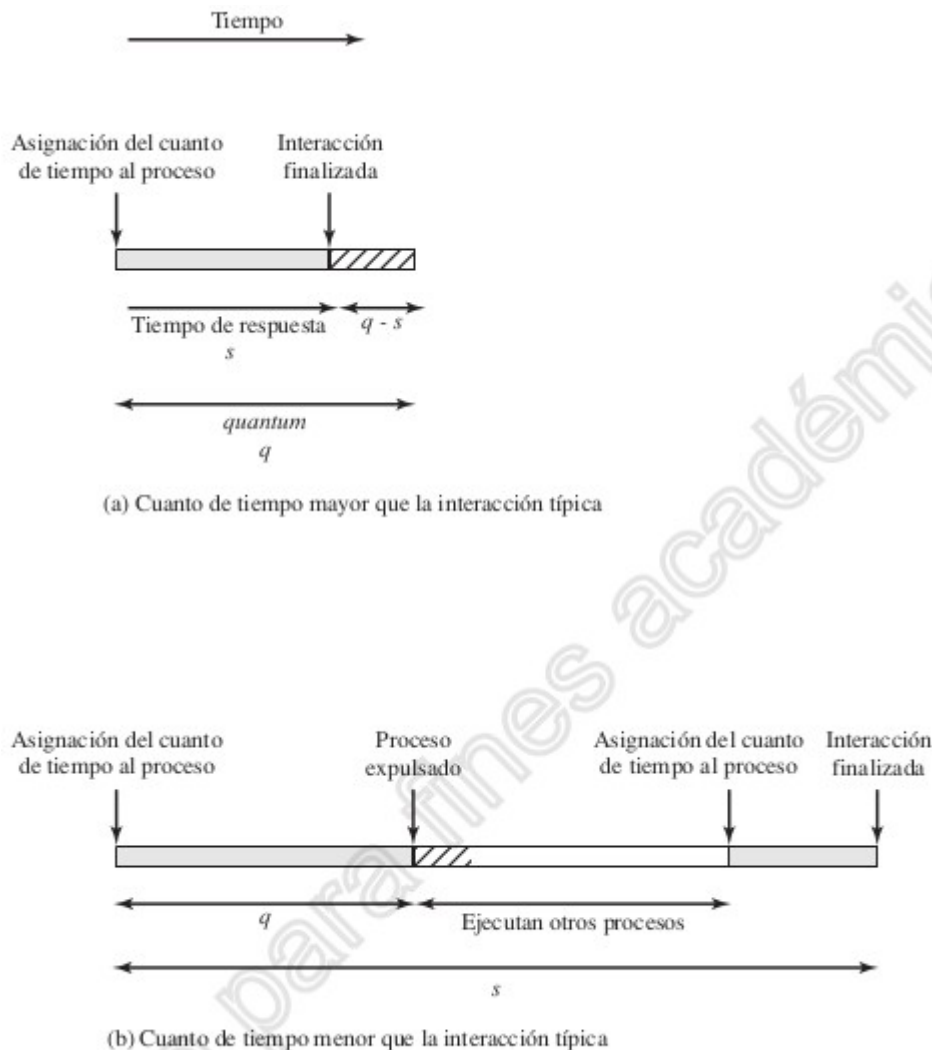


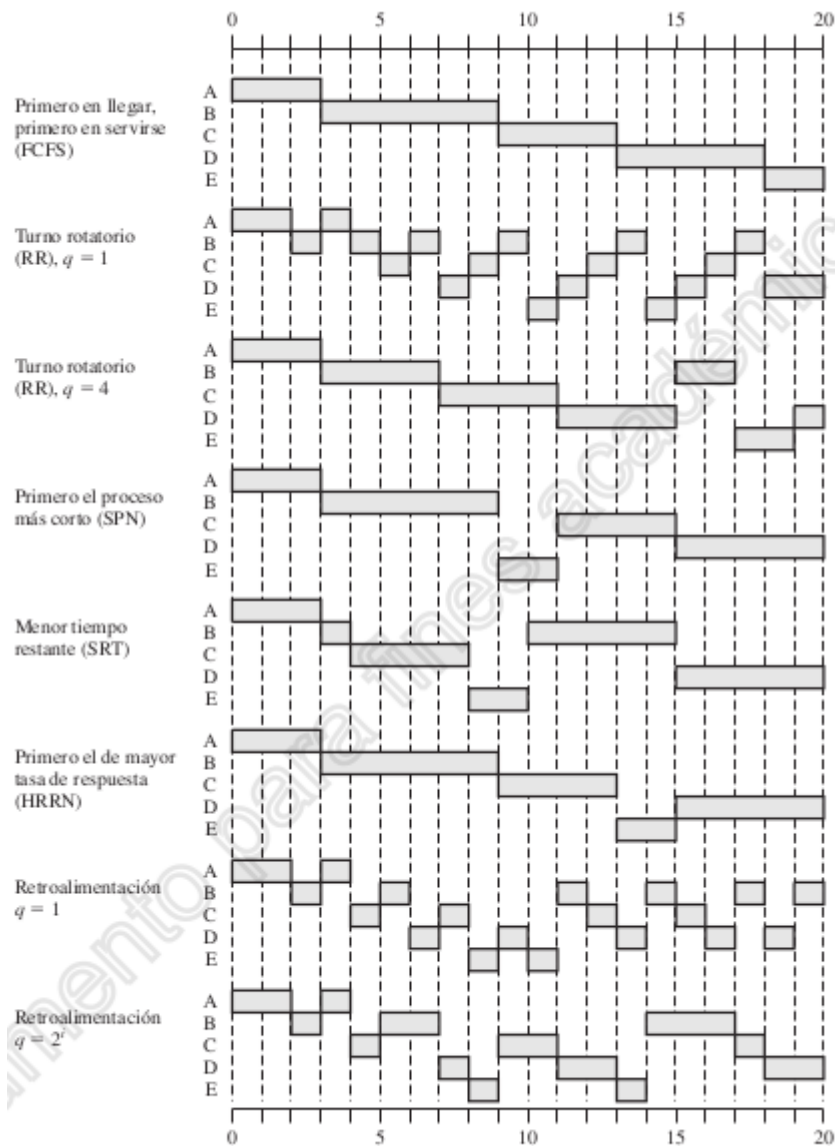
Figura 9.6. Efecto del tamaño del quantum de tiempo de expulsión.

Nótese que en el caso extremo de un quantum de tiempo mayor que el proceso más largo en ejecución, la planificación en turno rotatorio degenera en FCFS.

La Figura 9.5 y la Tabla 9.5 muestran los resultados de nuestro ejemplo utilizando quantums de tiempo  $q$  de 1 y 4 unidades de tiempo. Nótese que el proceso E, que es el trabajo más corto, obtiene mejoras significativas para un quantum de tiempo de 1.

Con quantum largos se asemeja al FCFS con se indicaba antes

Aquí hay una rata



PT, Reg. T. SRT

A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Figura 9.5. Comparación de políticas de planificación.

La planificación en turno rotatorio es particularmente efectiva en sistemas de tiempo compartido de propósito general o en sistemas de procesamiento transaccional. Una desventaja de la planificación en turno rotatorio es que trata de forma desigual a los procesos limitados por el procesador y a los procesos limitados por la E/S. Generalmente, un proceso limitado por la E/S tiene ráfagas de procesador más cortas (cantidad de tiempo de ejecución utilizada entre operaciones de E/S) que los procesos limitados por el procesador. Si hay una mezcla de los dos tipos de procesos, sucederá lo siguiente: un proceso limitado por la E/S utiliza el procesador durante un periodo corto y luego se bloquea; espera a que complete la operación de E/S y a continuación se une a la cola de listos. Por otra parte, un proceso limitado por el procesador generalmente utiliza la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de listos. De esta forma, los procesos limitados por el procesador tienden a recibir una rodaja no equitativa de tiempo de procesador, lo que conlleva un mal rendimiento de los procesos limitados por la E/S, uso ineficiente de los recursos de E/S y un incremento en la varianza del tiempo de respuesta.

**Tabla 9.5.** Comparación de las políticas de planificación.

	Proceso	A	B	C	D	E	Media
	Tiempo de llegada	0	2	4	6	8	
	Tiempo de servicio ( $T_s$ )	3	6	4	5	2	
FCFS	Tiempo de finalización	3	9	13	18	20	
	Tiempo de estancia ( $T_e$ )	3	7	9	12	12	8.60
	$T_e/T_s$	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$	Tiempo de finalización	4	18	17	20	15	
	Tiempo de estancia ( $T_e$ )	4	16	13	14	7	10.80
	$T_e/T_s$	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$	Tiempo de finalización	3	17	11	20	19	
	Tiempo de estancia ( $T_e$ )	3	15	7	14	11	10.00
	$T_e/T_s$	1.00	2.5	1.75	2.80	5.50	2.71
SPN	Tiempo de finalización	3	9	15	20	11	
	Tiempo de estancia ( $T_e$ )	3	7	11	14	3	7.60
	$T_e/T_s$	1.00	1.17	2.75	2.80	1.50	1.84
SRT	Tiempo de finalización	3	15	8	20	10	
	Tiempo de estancia ( $T_e$ )	3	13	4	14	2	7.20
	$T_e/T_s$	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	Tiempo de finalización	3	9	13	20	15	
	Tiempo de estancia ( $T_e$ )	3	7	9	14	7	8.00
	$T_e/T_s$	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$	Tiempo de finalización	4	20	16	19	11	
	Tiempo de estancia ( $T_e$ )	4	18	12	13	3	10.00
	$T_e/T_s$	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2$	Tiempo de finalización	4	17	18	20	14	
	Tiempo de estancia ( $T_e$ )	4	15	14	14	6	10.60
	$T_e/T_s$	1.33	2.50	3.50	2.80	3.00	2.63

Se sugiere por parte de otros autores un refinamiento de la planificación en turno rotatorio que denomina turno rotatorio virtual (virtual round robin —VRR—) y que evita esta injusticia. La Figura 9.7 muestra un esquema. Los nuevos procesos que llegan se unen a la cola de listos, que es gestionada con FCFS. Cuando expira el tiempo de ejecución de un proceso, vuelve a la cola de listos. Cuando se bloquea un proceso por E/S, se une a la cola de E/S. Hasta aquí, todo como siempre. La nueva característica es una cola auxiliar FCFS a la que se mueven los procesos después de estar bloqueados en una E/S.

Cuando se va a tomar una decisión de activación, los procesos en la cola auxiliar tienen preferencia sobre los de la cola de listos. Cuando se activa un proceso desde la cola auxiliar, éste ejecuta por un tiempo no superior a la rodaja de tiempo, menos el tiempo total que ha estado ejecutando desde que no está en la cola principal de listos. Los estudios de rendimiento realizados por los autores, indican que este enfoque es realmente superior al turno rotatorio en términos de equidad.

Si en la cola auxiliar de E/S hay procesos que han terminado su operación, tendrán prioridad sobre los procesos de listos, el planificador siempre cogerá un proceso de la cola auxiliar antes que de listos, y le dará el tiempo de ejecución que le quedara de su rodaja de tiempo anterior.

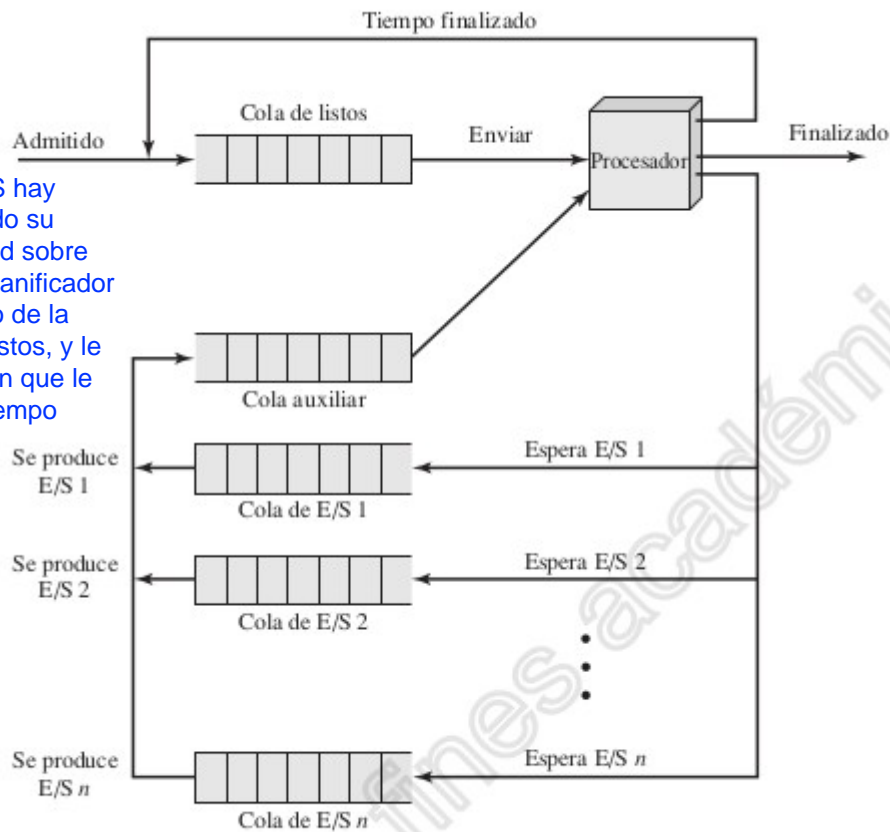


Figura 9.7. Diagrama de encolamiento para el planificador en turno rotatorio virtual.

### Primero el proceso más corto (shortest process next – SPN)

Otro enfoque para reducir el sesgo a favor de los procesos largos inherente al FCFS es la política primero el proceso más corto (SPN). Es una política no expulsiva en la que se selecciona el proceso con el tiempo de procesamiento más corto esperado. De esta forma un proceso corto se situará a la cabeza de la cola, por delante de los procesos más largos.

La Figura 9.5 y la Tabla 9.5 expuestas anteriormente muestran el resultado de nuestro ejemplo. Observe que el proceso E recibe servicio mucho antes que con FCFS. El rendimiento global también se mejora significativamente en términos del tiempo de respuesta. Sin embargo, se incrementa la variabilidad de los tiempos de respuesta, especialmente para los procesos más largos, y de esta forma se reduce la predecibilidad.

Un problema de la política SPN es la necesidad de saber, o al menos de estimar, el tiempo de procesamiento requerido por cada proceso. Para trabajos por lotes, el sistema puede requerir que el programador estime el valor y se lo proporcione al sistema operativo. Si la estimación del programador es mucho menor que el tiempo actual de ejecución, el sistema podría abortar el trabajo. En un entorno de producción, ejecutan frecuentemente los mismos trabajos y, por tanto, se pueden recoger estadísticas. Para procesos interactivos, el sistema operativo podría guardar una media del tiempo de ejecución de cada «ráfaga» de cada proceso. La forma más sencilla de cálculo podría ser la siguiente:

procesos limitados por operaciones de entrada/salida

En procesos limitados por la cpu, al ser no expulsivo se ejecutaría enteros y si están limitados por E/S pueden quedar sin ejecutarse

$$S_{n+1} = \frac{1}{n} \sum_{i=1}^n T_i \quad (9.1)$$

donde,

$T_i$  = tiempo de ejecución del procesador para la instancia  $i$ -ésima de este proceso (tiempo total de ejecución para los trabajos por lotes; tiempo de ráfaga de procesador para los trabajos interactivos)

$S_i$  = valor predicho para la instancia  $i$ -ésima

$S_1$  = valor predicho para la primera instancia; no calculado

Para evitar volver a calcular la suma completa cada vez, podemos reescribir la ecuación como:

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n \quad (9.2)$$

Observe que esta fórmula otorga el mismo peso a cada instancia. Así, nos gustaría otorgar mayor peso a las instancias más recientes, ya que hay más probabilidades de que reflejen comportamientos futuros. El promedio exponencial es una técnica común de predecir valores futuros basándose en una serie de tiempos pasados:

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n \quad (9.3)$$

donde  $\alpha$  es un factor de multiplicación constante ( $0 < \alpha < 1$ ) que determina el peso relativo dado a las observaciones más recientes en relación a las antiguas observaciones. Comparar con la Ecuación (9.2). A través del uso de un valor constante de  $\alpha$ , independientemente del número de observaciones pasadas, todos los valores pasados se consideran, pero los más distantes tienen menor peso. Para ver esto con mayor claridad, considere el siguiente desarrollo de la Ecuación (9.3):

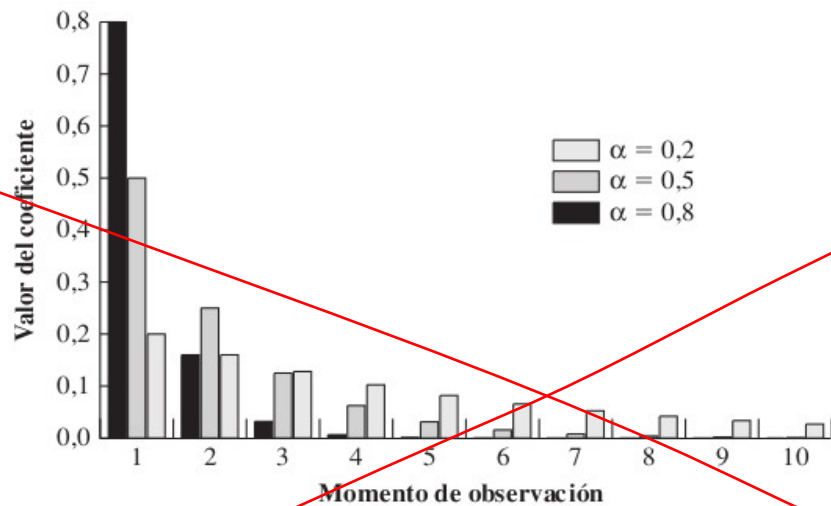
$$S_{n+1} = \alpha T_n + (1 - \alpha) \alpha T_{n-1} + \dots + (1 - \alpha)^i \alpha T_{n-i} + \dots + (1 - \alpha)^n S_1 \quad (9.4)$$

Debido a que  $\alpha$  y  $(1 - \alpha)$  son menores que 1, cada término sucesivo de la anterior ecuación es más pequeño. Por ejemplo, para  $\alpha = 0,8$ , la Ecuación (9.4) queda:

$$S_{n+1} = 0,8T_n + 0,16T_{n-1} + 0,032T_{n-2} + 0,0064T_{n-3} + \dots$$

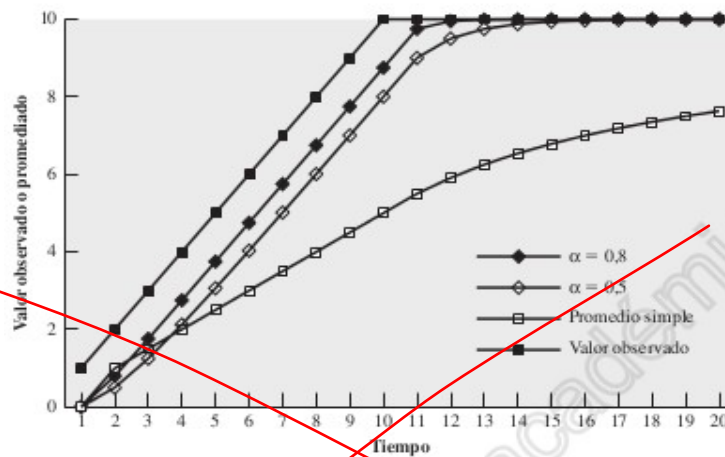
Más antigua sea la observación, menos cuenta en el promedio.

En la Figura 9.8 se muestra el tamaño del coeficiente como una función de su posición en el desarrollo. Mayor sea el valor de  $\alpha$ , mayor será el peso dado a las observaciones más recientes. Para  $\alpha = 0,8$ , prácticamente se da todo el peso a las cuatro observaciones más recientes, mientras que para  $\alpha = 0,2$ , el promedio se extiende más o menos sobre las ocho últimas observaciones. La ventaja de utilizar un valor de  $\alpha$  cercano a 1 es que el promedio reflejará un cambio rápido en las cantidades observadas. La desventaja es que si hay un cambio brusco en los valores observados y luego vuelven a su valor medio anterior, el uso de un valor alto de  $\alpha$  puede dar lugar a oscilaciones en el promedio.

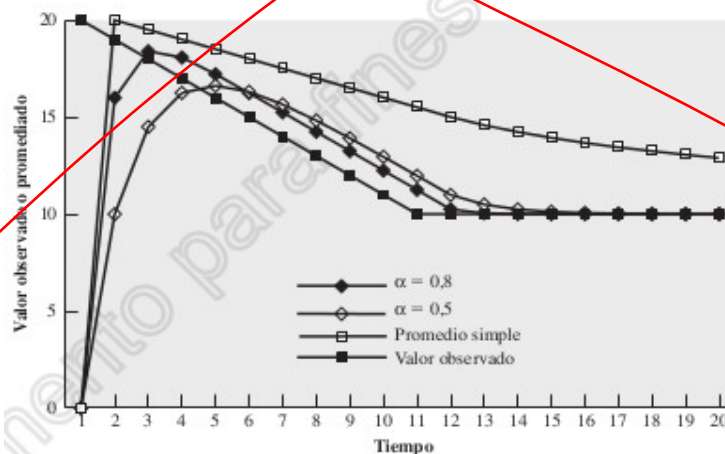


**Figura 9.8.** Coeficientes exponenciales suaves.

La Figura 9.9 compara el promedio simple con el exponencial (para dos valores diferentes de  $\alpha$ ). En la Figura 9.9a, el valor observado comienza en 1, crece gradualmente hasta el valor de 10, y luego se mantiene allí. En la Figura 9.9b, el valor observado comienza en 20, baja gradualmente hasta 10, y luego se mantiene allí. En ambos casos, comenzamos con un valor estimado de  $S_1 = 0$ . Esto da mayor prioridad a los nuevos procesos. Observar que el promedio exponencial reacciona a los cambios de comportamiento de los procesos más rápido que el promedio simple, y que los valores mayores de  $\alpha$  generan reacciones más rápidas a los cambios en los valores observados.



(a) Función creciente



(b) Función decreciente

**Figura 9.9.** Uso del promedio exponencial.



Un riesgo con SPN es la posibilidad de inanición para los procesos más largos, si hay una llegada constante de procesos más cortos. Por otra parte, ~~aunque SPN reduce la predisposición a favor de los trabajos más largos, no es deseable para un entorno de tiempo compartido o de procesamiento transaccional por la carencia de expulsión.~~ Volviendo al peor caso descrito con FCFS, los procesos W, X, Y y Z seguirán ejecutando en el mismo orden, penalizando fuertemente al proceso corto Y.

luego en determinadas situaciones, incluso un proceso corto puede ser desfavorecido debido a la carencia de expulsión de esta política



### **Menor tiempo restante (shortest remaining time – SRT)**

La política del menor tiempo restante (SRT) es una versión expulsiva de SPN. En este caso, el planificador siempre escoge el proceso que tiene el menor tiempo de proceso restante esperado. Cuando un nuevo proceso se une a la cola de listos, podría tener un tiempo restante menor que el proceso actualmente en ejecución. Por tanto, el planificador podría expulsar al proceso actual cuando llega un nuevo proceso. Al igual que con SPN, el planificador debe tener una estimación del tiempo de proceso para realizar la función seleccionada, y existe riesgo de inanición para los procesos más largos.

SRT no tiene el sesgo en favor de los procesos largos que se puede encontrar en FCFS. A diferencia del turno rotatorio, no se generan interrupciones adicionales, reduciéndose la sobrecarga. Por otra parte, se deben almacenar los tiempos de servicio transcurridos, generando sobrecarga. SRT debería mejorar los tiempos de estancia de SPN, porque a un trabajo corto se le da preferencia sobre un trabajo más largo en ejecución.

Obsérvese que en nuestro ejemplo (Tabla 9.5), los tres procesos más cortos reciben servicio de forma inmediata, dando un tiempo de estancia normalizado para cada uno de ellos de 1,0.

### **Retroalimentación (feedback)**

Si no es posible averiguar el tiempo de servicio de varios procesos, SPN y SRT no se pueden utilizar. Otra forma de establecer una preferencia para los trabajos más cortos es penalizar a los trabajos que han estado ejecutando más tiempo. En otras palabras, si no podemos basarnos en el tiempo de ejecución restante, nos podemos basar en el tiempo de ejecución utilizado hasta el momento.

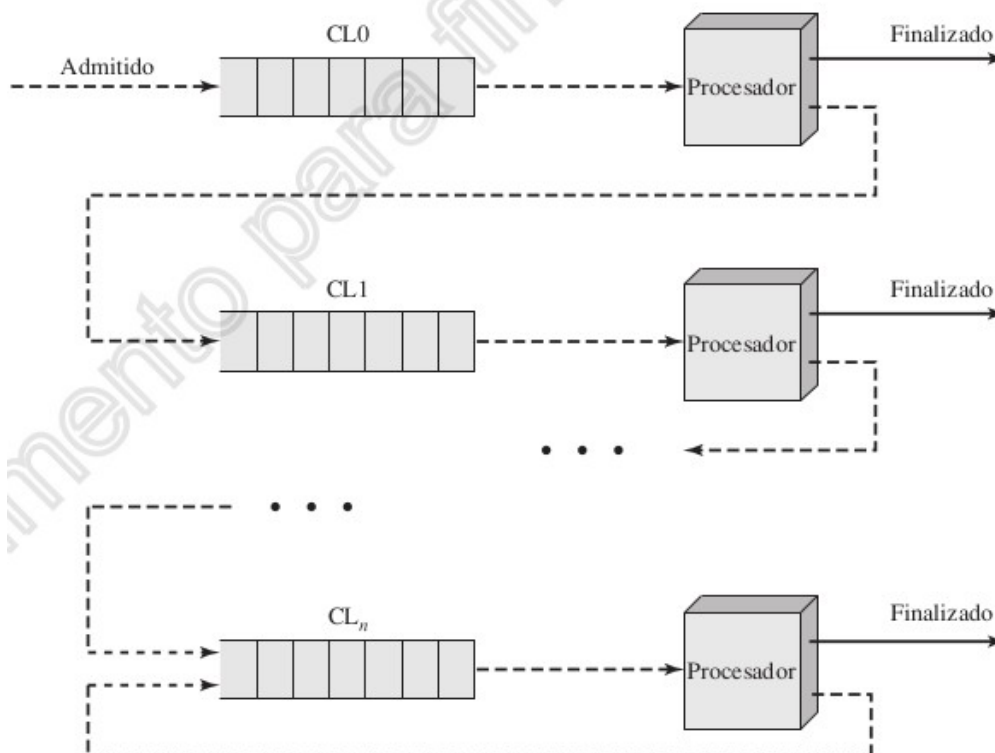
La forma de hacerlo es la siguiente. La planificación se realiza con expulsión (por rodajas de tiempo), y se utiliza un mecanismo de prioridades dinámico. Cuando un proceso entra en el sistema, se sitúa en CL0 (véase Figura 9.4). Después de su primera expulsión, cuando vuelve al estado de Listo, se sitúa en CL1. Cada vez que es expulsado, se sitúa en la siguiente cola de menor prioridad. Un proceso corto se completará de forma rápida, sin llegar a migrar muy lejos en la jerarquía de colas de listos. Un proceso más largo irá degradándose gradualmente. De esta forma, se favorece a los procesos nuevos más cortos sobre los más viejos y largos. Dentro de cada cola, excepto en la cola de menor prioridad, se utiliza un mecanismo FCFS. Una vez en la cola de menor prioridad, un proceso no puede descender más, por lo que es devuelto a esta cola repetidas veces hasta que se consigue completar. De esta forma, esta cola se trata con una política de turno rotatorio.

La Figura 9.10 ilustra el mecanismo de planificación con retroalimentación mostrando el camino que seguirá un proceso a lo largo de las colas<sup>3</sup>. Este enfoque es conocido como retroalimentación multinivel, ya que el sistema operativo adjudica el procesador a un proceso y, cuando el proceso se bloquea o es expulsado, lo vuelve a situar en una de las varias colas de prioridad.



Hay una serie de variaciones de este esquema. Una versión sencilla consiste en realizar la expulsión de la misma manera que la política rotatoria: con intervalos periódicos. Nuestro ejemplo muestra esta situación (Figura 9.5 y Tabla 9.5) para una rodaja de una unidad de tiempo. Observar que en este caso, el comportamiento es similar a un turno rotatorio con rodaja de tiempo 1.

<sup>3</sup> Se utilizan las líneas de tiempo para enfatizar que se trata de un diagrama de secuencias de tiempo y no una representación estática de posibles transacciones, como en la Figura 9.4.



**Figura 9.10.** Planificación retroalimentada.

Un problema que presenta el esquema simple antes contado es que el tiempo de estancia para procesos más largos se puede alargar de forma alarmante. De hecho, puede ocurrir inanición si están entrando nuevos trabajos frecuentemente en el sistema. Para compensar este problema, podemos variar los tiempos de expulsión de cada cola: un proceso de la cola CL0 tiene una rodaja de una unidad de tiempo; un proceso de la cola CL1 tiene una rodaja de dos unidades de tiempo, y así sucesivamente. En general, a un proceso de la cola CL<sub>i</sub> se le permite ejecutar  $2^i$  unidades de tiempo antes de ser expulsado. Este esquema se muestra en nuestro ejemplo de la Figura 9.5 y Tabla 9.5.

Incluso asignando rodajas de tiempo mayores a las colas de menor prioridad, un proceso grande puede sufrir inanición. Una posible solución es promover a los procesos a colas de mayor prioridad después de que pasen un determinado tiempo esperando servicio en su cola actual.

## **La planificación contribución justa (Fair-share scheduling - FSS)**

Todos los algoritmos de planificación discutidos hasta el momento tratan a la colección de procesos listos como un simple conjunto de procesos de los cuales seleccionar el siguiente a ejecutar. Este conjunto de procesos se podría romper con el uso de prioridades, pero es homogéneo.

Sin embargo, en un sistema multiusuario, si las aplicaciones o trabajos de usuario se pueden organizar como múltiples procesos (o hilos), hay una estructura en la colección de procesos que no se reconoce en los planificadores tradicionales. Desde el punto de vista del usuario, la preocupación no es cómo ejecuta un simple proceso, sino cómo ejecutan su conjunto de procesos, que forman una aplicación. De esta forma, sería deseable tomar decisiones de planificación basándose en estos conjuntos de procesos. Este enfoque se conoce normalmente como planificación contribución justa.

Además, el concepto se puede extender a un grupo de usuarios, incluso si cada usuario se representa con un solo proceso. Por ejemplo, en un sistema de tiempo compartido, podríamos querer considerar a todos los usuarios de un determinado departamento como miembros del mismo grupo.

Se podrían tomar las decisiones de planificación intentando dar a cada grupo un servicio similar. De esta forma, si muchos usuarios de un departamento se meten en el sistema, nos gustaría ver cómo el tiempo de respuesta se degrada sólo para los miembros de este departamento, más que para los usuarios de otros departamentos.

El término contribución justa indica la filosofía que hay detrás de este planificador. A cada usuario se le asigna una prima de algún tipo que define la participación del usuario en los recursos del sistema ~~como una fracción del uso total de dichos recursos. En particular, a cada usuario se le asigna una rodaja del procesador.~~ Este esquema debería operar, más o menos, de forma lineal. Así, si un usuario A tiene el doble de prima que un usuario B, en una ejecución larga, el usuario A debería ser capaz de hacer el doble de trabajo que el usuario B. El objetivo del planificador contribución justa es controlar el uso para dar menores recursos a usuarios que se han excedido de su contribución justa y mayores recursos a los que no han llegado.

Se han realizado una serie de propuestas para los planificadores contribución justa por diversos autores. En esta sección se describe el esquema propuesto por HENR en el año 84, que ha sido implementado en diversos sistemas UNIX. A este sistema se le denomina FSS (Fair Share Scheduler). FSS considera el histórico de ejecución de un grupo de procesos relacionados, junto con el histórico de ejecución de cada uno de los procesos, para tomar decisiones de planificación. El sistema divide a la comunidad de usuarios en un conjunto de grupos y destina una fracción del procesador a cada grupo. De esta forma, podría haber 4 grupos, cada uno con un 25% del procesador. De hecho, a cada grupo se le proporciona un sistema virtual que ejecuta más lento que un sistema completo.

y con una única cola de listos

La planificación se realiza en base a la prioridad y tiene en cuenta la prioridad del proceso, su uso reciente de procesador y el uso reciente de procesador del grupo al que pertenece el proceso. Mayor sea el valor numérico de la prioridad, menor será la prioridad. La siguiente fórmula se aplica al proceso  $j$  del grupo  $k$ :

$$\begin{aligned}
 CPU_j(i) &= \frac{CPU_j(i-1)}{2} \\
 GCPU_k(i) &= \frac{GCPU_k(i-1)}{2} \\
 P_j(i) &= Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}
 \end{aligned}$$

a mayor valor de  $W$  (peso del proceso) sería menor valor de prioridad  $P$  (menor valor pero mas prioridad) y se ejecutara antes

donde

$CPU_j(i)$  = Medida de utilización del procesador por el proceso  $j$  en el intervalo  $i$ .

$GCPU_k(i)$  = Medida de utilización del procesador del grupo  $k$  en el intervalo  $i$ .

$P_j(i)$  = Prioridad del proceso  $j$  al comienzo del intervalo  $i$ ; valores más pequeños equivalen a prioridades más altas.

$Base_j$  = Prioridad base del proceso  $j$ .

$W_k$  = Prima asignada al grupo  $k$ , con la restricción que  $0 < W_k \leq 1$  y  $\sum_k W_k = 1$

A cada proceso se le asigna una prioridad base. La prioridad de un proceso disminuye a medida que el proceso utiliza el procesador y a medida que el grupo al que pertenece el proceso utiliza el procesador. En el caso de la utilización del grupo, se normaliza la media dividiendo por el peso de ese grupo. Mayor sea el peso asignado al grupo, su utilización afectará menos a su prioridad.

En la figura, se planifica primero al proceso A. Al final del primer segundo se expulsa. Los procesos B y C tienen ahora la mayor prioridad, y se planifica al proceso B. Al final de la segunda unidad de tiempo, el proceso A tiene la mayor prioridad. Fijarse cómo se repite el patrón: el núcleo planifica los procesos en orden: A, B, A, C, A, B y así sucesivamente. De esta forma, el 50% del procesador se destina al proceso A, que constituye un grupo, y el otro 50% a los procesos B y C que constituyen otro grupo.

El rectángulo coloreado representa el proceso en ejecución

**Figura 9.16.** Ejemplo de planificación contribución justa – tres procesos, dos grupos.

## Comparación de rendimiento

Claramente, el rendimiento de las políticas de planificación es un factor crítico en su elección. Sin embargo, es imposible hacer una comparación definitiva, porque el rendimiento relativo dependerá de multitud de factores, que incluyen la distribución de probabilidad de los tiempos de servicio de varios procesos, la eficiencia de la planificación y del mecanismo del cambio de contexto, la naturaleza de las demandas de E/S y el rendimiento de los subsistemas de E/S. Por tanto, dependiendo de estos factores, lo ideal sería que el sistema aplicase una política de planificación u otra. Por otro lado, no es lo mismo un sistema de propósito general que sistemas de cómputo de propósito específico donde sabemos que tipo de procesos va a haber en el sistema y para que se va a utilizar fundamentalmente. En éste último caso la elección de una política de planificación es algo más sencillo y concreto. Consulte la bibliografía básica y la Web si está interesado en estudiar la política de planificación utilizada en los diferentes sistemas operativos existentes en el mercado.

A continuación se mostrará como ejemplo la planificación llevada a cabo en un sistema UNIX tradicional.

## Planificación UNIX tradicional

En esta sección examinamos la planificación tradicional UNIX, que se utiliza tanto en UNIX SVR3 como en 4.3 BSD UNIX. Estos sistemas están orientados a entornos interactivos de tiempo compartido. El algoritmo de planificación está diseñado para proporcionar buenos tiempos de respuesta a usuarios interactivos a la vez que asegura que los trabajos de fondo (background) de baja prioridad no sufran inanición. Aunque este sistema ha sido reemplazado en los sistemas UNIX modernos, merece la pena examinarlo porque es representativo de los algoritmos prácticos de planificación de tiempo compartido.

El planificador UNIX tradicional emplea una retroalimentación multinivel y utiliza planificación de turno rotatorio en cada una de las colas de prioridad. El sistema hace uso de expulsión de 1 segundo, es decir, si un proceso no se bloquea o se completa en un segundo, es expulsado. La prioridad está basada en el tipo de proceso y el histórico de ejecución. Se aplican las siguientes fórmulas:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

pretende poner a los procesos en un rango de prioridades para optimizar el uso de la cpu

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + ajuste_j$$

donde

$CPU_j(i)$  = Medida de utilización del procesador por el proceso  $j$  en el intervalo  $i$ .

$P_j(i)$  = Prioridad del proceso  $j$  al comienzo del intervalo  $i$ ; valores más pequeños equivalen a prioridades más altas.

$Base_j$  = Prioridad base del proceso  $j$ .

$ajuste_j$  = Factor de ajuste controlable por el usuario.

La prioridad de cada proceso se recalcula una vez por segundo, momento en el que se realiza una nueva decisión de planificación. El propósito de la prioridad base es dividir todos los procesos en bandas fijas de niveles de prioridad. Los componentes CPU y ajuste se limitan para impedir que los

procesos migren de su banda asignada (asignada por el nivel de prioridad base). Estas bandas se utilizan para optimizar el acceso a dispositivos de bloques (por ejemplo, disco) y para permitir al sistema operativo responder rápidamente a llamadas al sistema. En orden decreciente de prioridad las bandas son:

- Intercambiador (swapper).
- Control de dispositivos de bloque de E/S.
- Manipulación de ficheros.
- Control de dispositivos orientados a carácter de E/S.
- Procesos de usuario.

Esta jerarquía debería proporcionar el uso más eficiente de los dispositivos de E/S. Dentro de la banda de procesos de usuario, el uso del histórico de ejecución tiende a penalizar los procesos limitados por el procesador a costa de los procesos limitados por la E/S. De nuevo, esto debería mejorar la eficiencia. Junto con el esquema expulsivo cíclico, la estrategia de planificación está bien diseñada para satisfacer los requerimientos del tiempo compartido de propósito general.

En la Figura 9.17 se muestra un ejemplo de planificación de procesos. Los procesos A, B y C se crean al mismo tiempo con una prioridad base de 60 (ignoraremos el valor ajuste). El reloj interrumpe al sistema 60 veces por segundo e incrementa un contador para el proceso en ejecución. El ejemplo supone que ninguno de los procesos se bloquea y que ningún otro proceso está listo para ejecutar. Compárese con la Figura 9.16.

Tiempo	Proceso A		Proceso B		Proceso C	
	Prioridad	Contador CPU	Prioridad	Contador CPU	Prioridad	Contador CPU
0	60	0	60	0	60	0
		1				
		2				
		•				
		•				
		60				
1	75	30	60	0	60	0
				1		
				2		
				•		
				•		
				60		
2	67	15	75	30	60	0
						1
						2
						•
						•
						60
3	63	7	67	15	75	30
		8				
		9				
		•				
		•				
		67				
4	76	33	63	7	67	15
				8		
				9		
				•		
				•		
				67		
5	68	16	76	33	63	7

El rectángulo coloreado representa el proceso en ejecución

**Figura 9.17.** Ejemplo de planificación tradicional de procesos UNIX.



## Planificación multiprocesador

Cuando un sistema computador contiene más de un procesador, el diseño de la función de planificación plantea varias cuestiones nuevas. Se comienza con un breve resumen de los multiprocesadores y luego se ve que las consideraciones son bastante diferentes cuando la planificación se hace a nivel de proceso y cuando se hace nivel de hilo.

Podemos clasificar los sistemas multiprocesador como sigue:

- **Débilmente acoplado o multiprocesador distribuido, o cluster.** Consiste en una colección de sistemas relativamente autónomos; cada procesador tiene su propia memoria principal y canales de E/S.
- **Procesadores de funcionalidad especializada.** Un ejemplo es un procesador de E/S. En este caso, hay un procesador de propósito general maestro y procesadores especializados que son controlados por el procesador maestro y que le proporcionan servicios.
- **Procesamiento fuertemente acoplado.** Consiste en un conjunto de procesadores que comparten la memoria principal y están bajo el control integrado de un único sistema operativo.

Lo que nos concierne en esta sección es la última categoría y específicamente los aspectos relacionados con la planificación.

## Granularidad

Una buena manera de caracterizar los multiprocesadores y de situarlos en contexto respecto de otras arquitecturas, es considerar la granularidad de sincronización, o frecuencia de sincronización entre los procesos del sistema. Podemos distinguir cinco categorías de paralelismo que difieren en el grado de granularidad. Éstas se resumen en la Tabla 10.1.

Tabla 10.1. Granularidad de sincronización y procesos.

Tamaño del Grano	Descripción	Intervalo de sincronización (Instrucciones)
Fino	Paralelismo inherente en un único flujo de instrucciones	<20
Medio	Procesamiento paralelo o multitarea dentro de una única aplicación	20-200
Grueso	Multiprocesamiento de procesos concurrentes en un entorno multiprogramado	200-2000
Muy grueso	Procesamiento distribuido entre nodos de una red para conformar un único entorno de computación	2000-1M
Independiente	Múltiples procesos no relacionados	(N/D)

Comunicación casi dependiente

Comunicación muy alta entre hilos

Comunicación excasa entre hilos

- **Paralelismo Independiente:** Con paralelismo independiente, no hay sincronización explícita entre procesos. Cada uno representa un trabajo o aplicación independiente y separada. Un uso típico de este tipo de paralelismo se da en los sistemas de tiempo compartido. Cada usuario desarrolla su propio trabajo con una aplicación particular, como tratamiento de textos o una hoja de cálculo. El multiprocesador proporciona el mismo servicio que un monoprocesador multiprogramado. Dado que hay más de un procesador disponible, el tiempo de respuesta medio de los usuarios será menor. Es posible conseguir una mejora de prestaciones similar proporcionando a cada usuario un computador personal o estación de trabajo. Si se va a compartir cualquier información o archivo, los sistemas individuales deben de estar conectados entre sí a través de una red, formando un sistema distribuido. Por otro lado, un único sistema multiprocesador compartido es, en muchos casos, más eficaz en coste que un sistema distribuido, si consideramos la economía de escala en discos y otros periféricos.



- **Paralelismo de grano grueso y muy grueso:** Con el paralelismo de grano grueso y muy grueso, hay sincronización entre procesos, pero a un nivel muy burdo. Este tipo de situación se trata sencillamente como un conjunto de procesos concurrentes ejecutando en un monoprocesador multiprogramado y puede proporcionarse en un multiprocesador con poco o ningún cambio en el software de usuario.

En general, cualquier colección de procesos concurrentes que necesitan comunicarse o sincronizarse pueden beneficiarse del uso de una arquitectura multiprocesador. En el caso de que la interacción entre procesos sea muy poco frecuente, un sistema distribuido puede proporcionar un buen soporte. Sin embargo, si la interacción es algo más frecuente, entonces la sobrecarga de comunicaciones a través de la red puede mermar la potencial mejora de velocidad. En este caso, la organización multiprocesador proporciona un soporte más eficaz.

- **Paralelismo de grano medio:** Mientras que los paralelismos independiente, grueso y muy grueso pueden proporcionarse en un monoprocesador multiprogramado o en un multiprocesador con poco o ningún impacto en la función de planificación, cuando se trata de la planificación de hilos es necesario reexaminar la planificación.

Dado que varios hilos de una aplicación interactúan muy frecuentemente, las decisiones de planificación concernientes a un hilo pueden afectar a las prestaciones de la aplicación completa.

- **Paralelismo de grano fino:** El paralelismo de grano fino representa un uso mucho más complejo del paralelismo del que se encuentra en el uso de hilos. Aunque se ha realizado mucho trabajo sobre aplicaciones altamente paralelas, esta es, a día de hoy, un área especializada y fragmentada con muchas propuestas diferentes.

## Aspectos de diseño

En un multiprocesador la planificación involucra tres aspectos interrelacionados:

- La asignación de procesos a procesadores.
- El uso de la multiprogramación en cada procesador individual.
- La activación del proceso, propiamente dicha.

Al considerar estos tres aspectos, es importante tener en mente que la propuesta elegida dependerá, en general, del grado de granularidad de la aplicación y del número de procesadores disponibles.

### Asignación de procesos a procesadores

Si se asume que la arquitectura del multiprocesador es uniforme, en el sentido de que ningún procesador tiene una ventaja física particular con respecto al acceso a memoria principal o a dispositivos de E/S, entonces el enfoque más simple de la planificación consiste en tratar cada proceso como un recurso colectivo y asignar procesos a procesadores pordemanda. Surge la cuestión de si la asignación debería ser estática o dinámica.

Si un proceso se vincula permanentemente a un procesador desde su activación hasta que concluye, entonces se mantiene una cola a corto plazo dedicada por cada procesador. Una ventaja de esta estrategia es que puede haber menos sobrecarga en la función de planificación, dado que la asignación a un procesador se realiza una vez y para siempre. Asimismo, el uso de procesadores dedicados permite una estrategia conocida como planificación de grupo o pandilla, como se verá más adelante.

Una desventaja de la asignación estática es que un procesador puede estar ocioso, con su cola vacía, mientras otro procesador tiene trabajo acumulado. Para evitar esta situación, puede utilizarse una cola común. Todos los procesos van a una cola global y son planificados sobre cualquier procesador

disponible. Así, a lo largo de la vida de un proceso, puede ser ejecutado en diferentes procesadores en diferentes momentos. En una arquitectura de memoria compartida fuertemente acoplada, la información de contexto de todos los procesos estará disponible para todos los procesadores, y por lo tanto el coste de planificación de un proceso será independiente de la identidad del procesador sobre cual se planifica. Otra opción más, es el balance dinámico de carga, en el que los hilos se mueven de una cola de un procesador a la cola de otro procesador; Linux utiliza este enfoque.

Independientemente de cómo los procesos se vinculan a los procesadores, se necesita alguna manera de asignar procesos a procesadores. Se han venido usando dos enfoques: maestro/esclavo y camaradas. Con la arquitectura maestro/esclavo, ciertas funciones clave del núcleo del sistema operativo ejecutan siempre en un procesador concreto. Los otros procesadores sólo pueden ejecutar programas de usuario. El maestro es responsable de la planificación de trabajos. Una vez que el proceso está activo, si el esclavo necesita servicio (por ejemplo, una llamada de E/S), debe enviar una solicitud al maestro y esperar a que el servicio se realice. Este enfoque es bastante simple y sólo requiere mejorar un poco un sistema operativo monoprocesador multiprogramado. La resolución de conflictos se simplifica porque un procesador tiene todo el control de la memoria y de los recursos de E/S. Hay dos desventajas en este enfoque: (1) un fallo en el maestro hace que falle el sistema completo, y (2) el maestro puede llegar a ser un cuello de botella para el rendimiento del sistema.

En la arquitectura camaradas, el núcleo puede ser ejecutado en cualquier procesador, y cada procesador se auto-planifica desde la colección de procesos disponibles. Este enfoque complica el sistema operativo. El sistema operativo debe asegurar que dos procesadores no escogen el mismo proceso y que los procesos no se extravían por ninguna razón. Deben emplearse técnicas para resolver y sincronizar la competencia en la demanda de recursos.

Por supuesto, hay una gama de opciones entre estos dos extremos. Un enfoque es tener un subconjunto de los procesadores dedicado a procesar el núcleo en vez de uno solo. Otro enfoque es simplemente gestionar las diferentes necesidades entre procesos del núcleo y otros procesos sobre la base de la prioridad y la historia de ejecución.

## ***El uso de la multiprogramación en procesadores individuales***

Cuando cada proceso se asocia estáticamente a un procesador para todo su tiempo de vida, surge una nueva pregunta: ¿debería ese procesador ser multiprogramado? La primera reacción del lector puede ser preguntarse el porqué de esta cuestión; parece particularmente ineficiente vincular un procesador con un único proceso cuando este proceso puede quedar frecuentemente bloqueado esperando por E/S o por otras consideraciones de concurrencia/sincronización.

En los multiprocesadores tradicionales, que tratan con sincronizaciones de grano grueso o independientes (véase la Tabla 10.1), está claro que cada procesador individual debe ser capaz de cambiar entre varios procesos para conseguir una alta utilización y por tanto un mejor rendimiento. No obstante, para aplicaciones de grano medio ejecutando en un multiprocesador con muchos procesadores, la situación está menos clara. Cuando hay muchos procesadores disponibles, conseguir que cada procesador esté ocupado tanto tiempo como sea posible deja de ser lo más importante. En cambio, la preocupación es proporcionar el mejor rendimiento, medio, de las aplicaciones. Una aplicación que consista en varios hilos, puede ejecutar con dificultad a menos que todos sus hilos estén dispuestos a ejecutar simultáneamente.

## ***Activación de procesos***

El último aspecto de diseño relacionado con la planificación multiprocesador, es la elección real del proceso a ejecutar. Hemos visto que en un monoprocesador multiprogramado, el uso de prioridades o de sofisticados algoritmos de planificación basados en el uso pasado, pueden mejorar el rendimiento frente a la ingenua estrategia FCFS (primero en llegar, primero en ser servido). Cuando consideramos multiprocesadores, estas complejidades pueden ser innecesarias o incluso

contraproducentes, y un enfoque más simple puede ser más eficaz con menos sobrecarga. En el caso de la planificación de hilos, entran en juego nuevos aspectos que pueden ser más importantes que las prioridades o las historias de ejecución.

## Planificación de hilos

En los sistemas multiprocesador más tradicionales, los procesos no se vinculan a los procesadores sino que hay una única cola para todos los procesadores o, si se utiliza algún tipo de esquema basado en prioridades, hay múltiples colas basadas en prioridad, alimentando a un único colectivo de procesadores. En cualquier caso, el sistema puede verse como una arquitectura de colas multiservidor. Pero, ¿cómo se planifican los hilos?

Como hemos visto, con los hilos, el concepto de ejecución se separa del resto de la definición de un proceso. Una aplicación puede ser implementada como un conjunto de hilos, que cooperan y ejecutan de forma concurrente en el mismo espacio de direcciones. En un monoprocesador, los hilos pueden usarse como una ayuda a la estructuración de programas y para solapar E/S con el procesamiento. Dada la mínima penalización por realizar un cambio de hilo comparado con un cambio de proceso, los beneficios se obtienen con poco coste.

Sin embargo, el poder completo de los hilos se vuelve evidente en un sistema multiprocesador. En este entorno, los hilos pueden explotar paralelismo real dentro de una aplicación. Si los hilos de una aplicación están ejecutando simultáneamente en procesadores separados, es posible una mejora drástica de sus prestaciones. Sin embargo, puede demostrarse que para aplicaciones que necesitan una interacción significativa entre hilos (paralelismo de grano medio), pequeñas diferencias en la gestión y planificación de hilos pueden dar lugar a un impacto significativo en las prestaciones.

Entre las muchas propuestas para la planificación multiprocesador de hilos y la asignación a procesadores, destacan cuatro enfoques general:

- **Compartición de carga.** Los procesos no se asignan a un procesador particular. Se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de la cola. El término *compartición de carga* se utiliza para distinguir esta estrategia de los esquemas de balanceo de carga en los que los trabajos se asignan de una manera más permanente<sup>4</sup>.
- **Planificación en pandilla.** Un conjunto de hilos relacionados que se planifica para ejecutar sobre un conjunto de procesadores al mismo tiempo, en una relación uno-a-uno.
- **Asignación de procesador dedicado.** Esto es lo opuesto al enfoque de compartición de carga y proporciona una planificación implícita definida por la asignación de hilos a procesadores. Cada proceso ocupa un número de procesadores igual al número de hilos en el programa, durante toda la ejecución del programa. Cuando el programa termina, los procesadores regresan al parque general para la posible asignación a otro programa.
- **Planificación dinámica.** El número de hilos de un proceso puede cambiar durante el curso de su ejecución.

---

4 Cierta literatura sobre este tema se refiere a este enfoque como auto-planificación, porque cada procesador se planifica a sí mismo sin considerar a los otros procesadores. Sin embargo, este término también se utiliza en la literatura para referirse a programas escritos en lenguajes que permiten que el programador especifique la planificación

## Compartición de carga

La compartición de carga es posiblemente el enfoque más simple y que se surge más directamente de un entorno monoprocesador. Tiene algunas ventajas:

- La carga se distribuye uniformemente entre los procesadores, asegurando que un procesador no queda ocioso mientras haya trabajo pendiente.
- No se precisa un planificador centralizado; cuando un procesador queda disponible, la rutina de planificación del sistema operativo se ejecuta en dicho procesador para seleccionar el siguiente hilo.
- La cola global puede organizarse y ser accesible usando cualquiera de los esquemas expuestos anteriormente, incluyendo esquemas basados en prioridad y esquemas que consideran la historia de ejecución o anticipan demandas de procesamiento.

Hay tres versiones diferentes de compartición de carga:

- Primero en llegar, primero en ser servido (FCFS). Cuando llega un trabajo, cada uno de sus hilos se disponen consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, coge el siguiente hilo listo, que ejecuta hasta que se completa o se bloquea.
- Menor número de hilos primero. La cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan de acuerdo con qué trabajo llega primero. Al igual que con FCFS, el hilo planificado ejecuta hasta que se completa o se bloquea.
- Menor número de hilos primero con expulsión. Se le da mayor prioridad a los trabajos con el menor número de hilos no planificados. Si llega un trabajo con menor número de hilos que un trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

Usando modelos de simulación, los autores informan que, sobre un amplio rango de características de trabajo, FCFS es superior a las otras dos políticas en la lista precedente. Es más, los autores muestran que cierta forma de planificación en pandilla, expuesta en la siguiente subsección, es generalmente superior a la compartición de carga.

La compartición de carga tiene varias desventajas:

- La cola central ocupa una región de memoria a la que debe accederse de manera que se cumpla la exclusión mutua. De manera que puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo. Cuando hay sólo un pequeño número de procesadores, es difícil que esto se convierta en un problema apreciable. Sin embargo, cuando el multiprocesador consiste en docenas o quizás cientos de procesadores, la posibilidad de que esto sea un cuello de botella es real.
- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador está equipado con una cache local, ésta se volverá menos eficaz.
- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez. Si se necesita un alto grado de coordinación entre los hilos de un programa, los cambios de proceso necesarios pueden comprometer seriamente el rendimiento.

A pesar de las potenciales desventajas, este es uno de los esquemas más utilizados en los multiprocesadores actuales.

## Planificación en pandilla

FIFO con rodaja de tiempo

El concepto de planificar un conjunto de procesos simultáneamente sobre un conjunto de procesadores es anterior al uso de hilos. Con el concepto de de planificación en grupo se obtienen los siguientes beneficios:

- Si se ejecutan en paralelo procesos estrechamente relacionados, puede reducirse el bloqueo por sincronización, pueden necesitarse menos cambios de proceso y las prestaciones aumentarán.
- La sobrecarga de planificación puede reducirse dado que una decisión única afecta a varios procesadores y procesos a la vez.

La planificación en pandilla es para aplicaciones paralelas de grano medio o fino cuyo rendimiento se ve degradado de forma importante cuando cualquier parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. También es beneficiosa para cualquier aplicación paralela incluso para aquellas de rendimiento no tan sensible. La necesidad de planificación en pandilla está ampliamente reconocida, y existen implementaciones en variedad de sistemas operativos multiprocesador.

Una manera obvia en que la planificación en pandilla mejora las prestaciones de una aplicación individual es porque se minimizan los cambios de proceso. Suponga que un hilo de un proceso está ejecutando y alcanza un punto en el que debe sincronizarse con otro hilo del mismo proceso. Si este otro hilo no está ejecutando, pero está listo para ejecutar, el primer hilo quedará colgado hasta que suceda un cambio de proceso en otro procesador que tome el hilo necesario. En una aplicación con coordinación estrecha entre sus hilos, estos cambios reducirán dramáticamente el rendimiento. La planificación simultánea de hilos cooperantes puede también reducir el tiempo de ubicación de recursos. Por ejemplo, múltiples hilos planificados en pandilla pueden acceder a un fichero sin la sobrecarga adicional de bloquearse durante una operación de posicionamiento y lectura/escritura.

El uso de la planificación en pandilla crea un requisito para la ubicación de procesador. Una posibilidad es la siguiente: suponga que tenemos N procesadores y M aplicaciones, cada una de las cuales tiene N hilos o menos. Entonces a cada aplicación puede dársele  $1/M$  del tiempo disponible en los N procesadores, usando porciones de tiempo. No que esta estrategia puede ser ineficiente. Considere un ejemplo en el que hay dos aplicaciones, una con cuatro hilos y otra con un hilo. Utilizando una asignación de tiempo uniforme se desperdicia el 37,5% del recurso de procesamiento, dado que cuando ejecuta la aplicación de un único hilo, tres de los procesadores quedan ociosos (véase Figura 10.2). Si hay varias aplicaciones de un único hilo, todas ellas podrían ser encajadas juntas para mejorar la utilización de los procesadores. Si esa opción no está disponible, una alternativa a la planificación uniforme es la planificación ponderada por el número de hilos. Así, a la aplicación de cuatro hilos podría dársele  $4/5$  del tiempo y a cada aplicación de un hilo dársele solamente un quinto del tiempo, reduciéndose el desperdicio de procesador a un 15%.

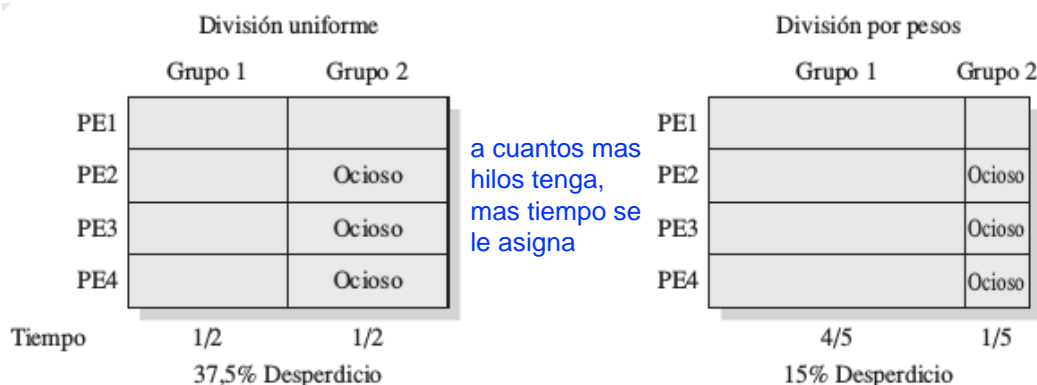


Figura 10.2. Ejemplo de planificación de grupos con cuatro y un hilos [FEIT90b].

## Asignación de procesador dedicado

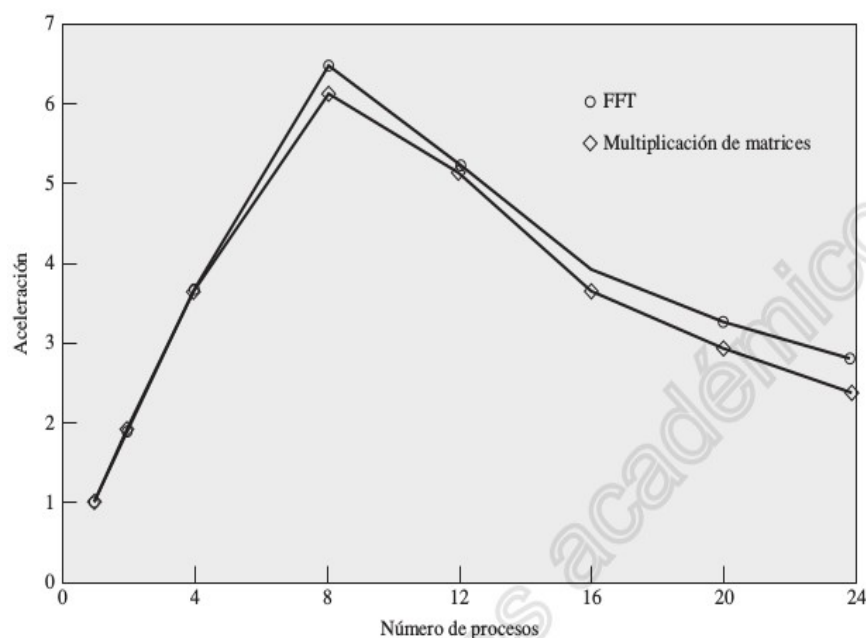
Con gran numero de procesadores

Una forma extrema de la planificación en pandilla es dedicar un grupo de procesadores a una aplicación durante toda la duración de la aplicación. Esto es, cuando se planifica una aplicación dada, cada uno de sus hilos se asigna a un procesador que permanece dedicado al hilo hasta que la aplicación concluye.

Este enfoque puede parecer un desperdicio extremo del tiempo de procesador. Si un hilo de una aplicación se bloquea esperando por E/S o por sincronización con otro hilo, entonces el procesador de ese hilo se queda ocioso: no hay multiprogramación de los procesadores. Pueden realizarse dos observaciones en defensa de esta estrategia:

1. En un sistema altamente paralelo, con decenas o cientos de procesadores, cada uno de los cuales representa una pequeña fracción del coste del sistema, la utilización del procesador deja de ser tan importante como medida de la eficacia o rendimiento.
2. Evitar totalmente el cambio de proceso durante la vida de un programa debe llevar a una sustancial mejora de velocidad de ese programa.

Algunos autores informan de los análisis que sustentan la segunda afirmación. La Figura 10.3 muestra los resultados de un experimento que ejecuta dos aplicaciones, una multiplicación de matrices y el cálculo de una transformada rápida de Fourier (FFT), en un sistema con 16 procesadores. Cada aplicación descompone su problema en varias tareas, que se proyectan en hilos que dicha aplicación ejecuta. Los programas fueron escritos de manera tal que se pueda variar el número de hilos que usan. En esencia, cada aplicación define e introduce en una cola un cierto número de tareas. La aplicación extrae las tareas de la cola y las proyecta sobre los hilos disponibles. Si hay menos hilos que tareas, las tareas restantes permanecen enconadas hasta que son extraídas por los hilos cuando completan la tarea asignada. Claramente, no todas las aplicaciones pueden ser estructuradas de este modo, pero muchos problemas numéricos y otras aplicaciones varias sí pueden tratarse de esta forma.



Cuando el numero de procesos excede de 16 (8 cada aplicscion) igual al numero de procesadores, el rendimiento descende porque es necesario cargar bcp, accesos a cache.... que reducen el rendimiento

Figura 10.3. Aceleración de la aplicación en función del número de procesos [TUCK89].

La Figura 10.3 muestra el incremento de velocidad de estas aplicaciones según el número de hilos que ejecutan las tareas de cada aplicación varía de 1 a 24. Por ejemplo, se observa que cuando ambas aplicaciones se arrancan simultáneamente con 24 hilos cada una, la aceleración obtenida, comparada con el uso de un único hilo por aplicación, es de 2,8 para la multiplicación de matrices y



de 2,4 para la FFT. La figura muestra que el rendimiento de ambas aplicaciones empeora considerablemente cuando el número de hilos en cada aplicación excede 8 y, por tanto, el número total de procesos en el sistema excede el número de procesadores. Es más, a mayor número de hilos peor rendimiento se obtiene, dado que la frecuencia de expulsión de hilos y de re-planificación es mayor. Esta excesiva expulsión provoca ineficiencia en muchos sentidos: tiempo gastado esperando a que un hilo suspendido abandone una sección crítica, tiempo gastado en cambios de proceso e ineficiente comportamiento de la cache.

Los autores concluyen que una estrategia eficaz es limitar el número de hilos activos al número de procesadores en el sistema. Si la mayoría de las aplicaciones son de un hilo único o pueden ser estructuradas como una cola de tareas, se conseguirá un uso eficaz y razonablemente eficiente de los recursos procesador.

~~Tanto la asignación de procesador dedicado como la planificación en pandilla atacan el problema de la planificación tratando el aspecto de la ubicación del procesador. Puede observarse que el problema de la ubicación del procesador en un multiprocesador se parece más al problema de la ubicación de la memoria en un monoprocesador que al problema de la planificación en un monoprocesador. Ahora la pregunta es cuántos procesadores asignar a un programa en un momento dado, que es análoga a cuántos marcos de página asignar a un proceso en dicho momento. Se define el término **conjunto residente de actividad** como el mínimo número de actividades (hilos) que deben ser planificados simultáneamente sobre procesadores para que la aplicación tenga un progreso aceptable. Al igual que con los esquemas de la gestión de la memoria, no planificar todos los elementos de un conjunto residente de actividad puede provocar trasiego de procesador. Esto sucede cuando la planificación de hilos cuyos servicios se necesitan, provoca la expulsión de otros hilos cuyos servicios serán necesitados pronto. De igual modo, la fragmentación de procesador se refiere a una situación en la cual quedan algunos procesadores sobrantes mientras otros están ubicados, y los procesadores sobrantes son o bien insuficientes en número, o bien no están organizados adecuadamente para dar soporte a los requisitos de las aplicaciones pendientes. La planificación en pandilla y la ubicación de procesador dedicado tienen como objetivo evitar estos problemas.~~

## ***Planificación dinámica***

Para algunas aplicaciones, es posible proporcionar, en el lenguaje o en el sistema, herramientas que permitan que el número de hilos del proceso pueda ser alterado dinámicamente. Esto permitiría que el sistema operativo ajustase la carga para mejorar la utilización.

~~Algunos autores proponen un enfoque en el cual tanto el sistema operativo como la aplicación están involucrados en tomar las decisiones de planificación. El sistema operativo es el responsable de particionar los procesadores entre los trabajos. Cada trabajo utiliza los procesadores actualmente en su partición para ejecutar algún subconjunto de sus tareas ejecutables proyectando estas tareas sobre hilos. La decisión apropiada acerca del subconjunto a ejecutar, así como qué hilos suspender cuando el proceso sea expulsado, se le deja a las aplicaciones individuales (quizás a través de un conjunto de rutinas de biblioteca). Este enfoque puede no ser adecuado para todas las aplicaciones. Sin embargo, algunas aplicaciones pueden ser consideradas como un único hilo mientras otras pueden ser programadas para sacar ventaja de esta particular característica del sistema operativo.~~

~~En este enfoque, la responsabilidad de planificación del sistema operativo está limitada fundamentalmente a la ubicación del procesador y se realiza de acuerdo a la siguiente política. Cuando un trabajo solicita uno o más procesadores (bien cuando el trabajo llega por primera vez o porque sus requisitos cambian),~~

~~1. Si hay procesadores ociosos, utilizarlos para satisfacer la solicitud.~~

~~2. En otro caso, si el trabajo que realiza la solicitud acaba de llegar, ubicarlo en un único procesador quitándoselo a cualquier trabajo que actualmente tenga más de un procesador.~~



~~3. Si no puede satisfacerse cualquier parte de la solicitud, mantenerla pendiente hasta que un procesador pase a estar disponible, o el trabajo rescinda la solicitud (por ejemplo, si dejan de ser necesarios los procesadores extra).~~

~~Cuando se libere uno o más procesadores (incluyendo la terminación de un trabajo),~~

~~4. Examinar la cola actual de solicitudes de procesador no satisfechas. Asignar un único procesador a cada trabajo en la lista que no tenga actualmente procesadores (por ejemplo, a todos los recién llegados en espera). Luego volver a examinar la lista, volviendo a asignar el resto de los procesadores siguiendo la estrategia FCFS.~~

~~Para las aplicaciones que pueden aprovechar las ventajas de la planificación dinámica, este enfoque es superior a la planificación en pandilla o a la asignación de procesador dedicado. No obstante, la sobrecarga de este enfoque puede invalidar esta aparente ventaja en rendimiento. Es necesario experimentar con el sistema real para probar la ventaja de la planificación dinámica.~~