CAPÍTULO 5

Planificación de la CPU

Los mecanismos de planificación de la CPU son la base de los sistemas operativos multiprogramados. Mediante la conmutación de la CPU entre distintos procesos, el sistema operativo puede hacer que la computadora sea más productiva. En este capítulo, vamos a presentar los conceptos básicos sobre la planificación de la CPU y varios de los algoritmos utilizados para este fin. También consideraremos el problema de seleccionar el mejor algoritmo para un sistema particular.

En el Capítulo 4, hemos presentado el concepto de hebras en el modelo de procesos. En los sistemas operativos que las soportan, son las hebras del nivel del *kernel*, y no los procesos, las que de hecho son planificadas por el sistema operativo. Sin embargo, los términos **planificación de procesos** y **planificación de hebras** se usan indistintamente. En este capítulo, utilizaremos el término *planificación de procesos* cuando hablemos de los conceptos generales sobre planificación y *planificación de hebras* cuando hagamos referencia a conceptos específicos de las hebras.

OBJETIVOS DEL CAPÍTULO

- Presentar los mecanismos de planificación de la CPU, que constituyen los cimientos de los sistemas operativos multiprogramados.
- Describir los distintos algoritmos para la planificación de la CPU.
- Exponer los criterios de evaluación utilizados para seleccionar un algoritmo de planificación de la CPU para un determinado sistema.

5.1 Conceptos básicos

En un sistema de un único procesador, sólo puede ejecutarse un proceso cada vez; cualquier otro proceso tendrá que esperar hasta que la CPU quede libre y pueda volver a planificarse. El objetivo de la multiprogramación es tener continuamente varios procesos en ejecución, con el fin de maximizar el uso de la CPU. La idea es bastante simple: un proceso se ejecuta hasta que tenga que esperar, normalmente porque es necesario completar alguna solicitud de E/S. En un sistema informático simple, la CPU permanece entonces inactiva y todo el tiempo de espera se desperdicia; no se realiza ningún trabajo útil. Con la multiprogramación, se intenta usar ese tiempo de forma productiva. En este caso, se mantienen varios procesos en memoria a la vez. Cuando un proceso tiene que esperar, el sistema operativo retira el uso de la CPU a ese proceso y se lo cede a otro proceso. Este patrón se repite continuamente y cada vez que un proceso tiene que esperar, otro proceso puede hacer uso de la CPU.

Este tipo de planificación es una función fundamental del sistema operativo; casi todos los recursos de la computadora se planifican antes de usarlos. Por supuesto, la CPU es uno de los principales recursos de la computadora, así que su correcta planificación resulta crucial en el diseño del sistema operativo.

5.1.1 Ciclo de ráfagas de CPU y de E/S

La adecuada planificación de la CPU depende de una propiedad observada de los procesos: la ejecución de un proceso consta de un **ciclo** de ejecución en la CPU, seguido de una espera de E/S; los procesos alternan entre estos dos estados. La ejecución del proceso comienza con una **ráfaga de CPU**. Ésta va seguida de una **ráfaga de E/S**, a la cual sigue otra ráfaga de CPU, luego otra ráfaga de E/S, etc. Finalmente, la ráfaga final de CPU concluye con una solicitud al sistema para terminar la ejecución (Figura 5.1).

La duración de las ráfagas de CPU se ha medido exhaustivamente en la práctica. Aunque varían enormemente de un proceso a otro y de una computadora a otra, tienden a presentar una curva de frecuencia similar a la mostrada en la Figura 5.2. Generalmente, la curva es de tipo exponencial o hiperexponencial, con un gran número de ráfagas de CPU cortas y un número menor de ráfagas de CPU largas. Normalmente, un programa limitado por E/S presenta muchas ráfagas de CPU cortas. Esta distribución puede ser importante en la selección de un algoritmo apropiado para la planificación de la CPU.

5.1.2 Planificador de la CPU

Cuando la CPU queda inactiva, el sistema operativo debe seleccionar uno de los procesos que se encuentran en la cola de procesos preparados para ejecución. El **planificador a corto plazo** (o planificador de la CPU) lleva a cabo esa selección del proceso. El planificador elige uno de los procesos que están en memoria preparados para ejecutarse y asigna la CPU a dicho proceso.

Observe que la cola de procesos preparados no necesariamente tiene que ser una cola FIFO (first-in, first-out). Como veremos al considerar los distintos algoritmos de planificación, una cola de procesos preparados puede implementarse como una cola FIFO, una cola prioritaria, un árbol o simplemente una lista enlazada no ordenada. Sin embargo, conceptualmente, todos los proce-

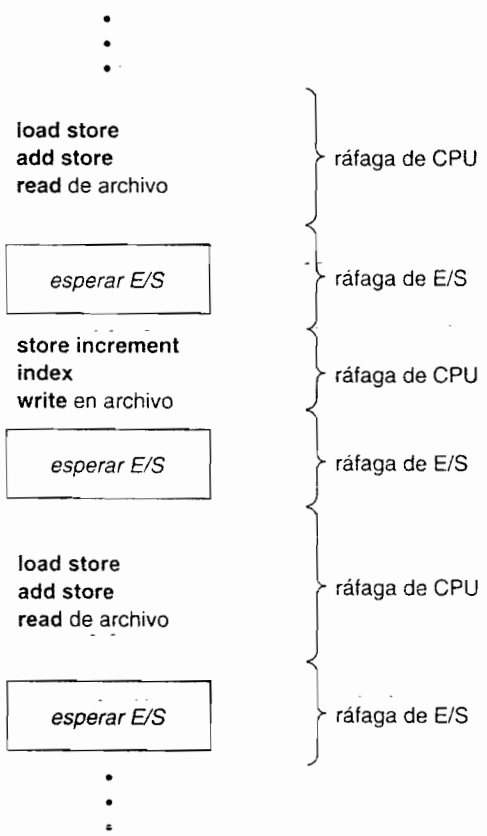


Figura 5.1 Secuencia alternante de ráfagas de CPU y de E/S.

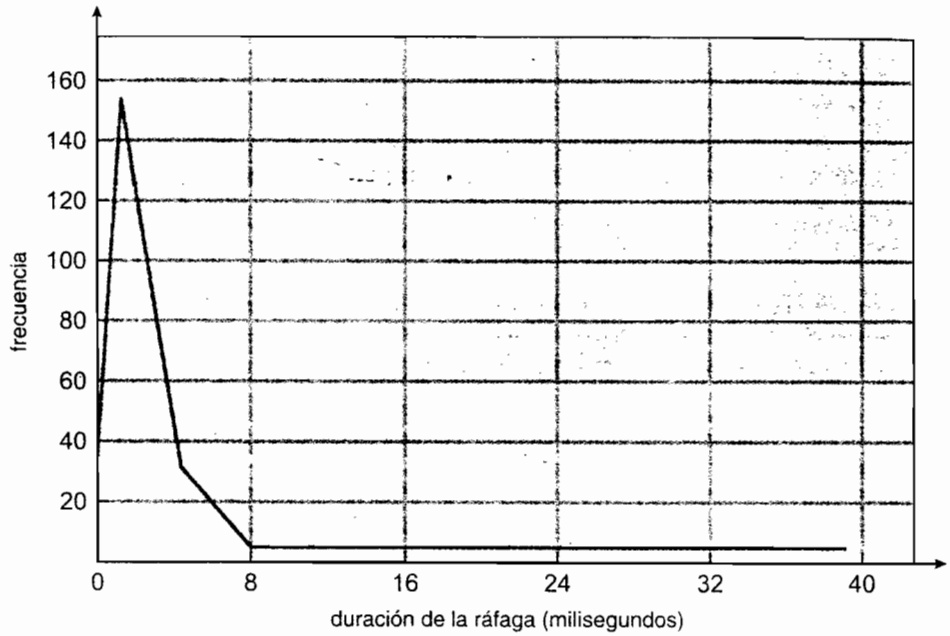


Figura 5.2 Histograma de duración de las ráfagas de CPU.

sos que se encuentran en la cola de procesos preparados se ponen en fila esperando la oportunidad de ejecutarse en la CPU. Los registros que se almacenan en las colas son, generalmente, bloques de control de proceso (PCB) que describen los procesos en cuestión.

5.1.3 Planificación apropiativa

Puede ser necesario tomar decisiones sobre planificación de la CPU en las siguientes cuatro circunstancias:

1. Cuando un proceso cambia del estado de ejecución al estado de espera (por ejemplo, como resultado de una solicitud de E/S o de una invocación de `wait` para esperar a que termine uno de los procesos hijo).
2. Cuando un proceso cambia del estado de ejecución al estado preparado (por ejemplo, cuando se produce una interrupción).
3. Cuando un proceso cambia del estado de espera al estado preparado (por ejemplo, al completarse una operación de E/S).
4. Cuando un proceso termina.

En las situaciones 1 y 4, no hay más que una opción en términos de planificación: debe seleccionarse un nuevo proceso para su ejecución (si hay algún proceso en la cola de procesos preparados). Sin embargo, en las situaciones 2 y 3 sí que existe la opción de planificar un nuevo proceso o no.

Cuando las decisiones de planificación sólo tienen lugar en las circunstancias 1 y 4, decimos que el esquema de planificación es **sin desalojo** o **cooperativo**; en caso contrario, se trata de un esquema **apropiativo**. En la planificación sin desalojo, una vez que se ha asignado la CPU a un proceso, el proceso se mantiene en la CPU hasta que ésta es liberada bien por la terminación del proceso o bien por la conmutación al estado de espera. Este método de planificación era el utilizado por Microsoft Windows 3.x; Windows 95 introdujo la planificación apropiativa y todas las versiones siguientes de los sistemas operativos Windows han usado dicho tipo de planificación. El sistema operativo Mac OS X para Macintosh utiliza la planificación apropiativa; las versiones anteriores del sistema operativo de Macintosh se basaban en la planificación cooperativa. La planificación cooperativa es el único método que se puede emplear en determinadas plataformas

hardware, dado que no requiere el hardware especial (por ejemplo, un temporizador) necesario para la planificación apropiativa.

Lamentablemente, la planificación apropiativa tiene un coste asociado con el acceso a los datos compartidos. Considere el caso de dos procesos que compartan datos y suponga que, mientras que uno está actualizando los datos, resulta desalojado con el fin de que el segundo proceso pueda ejecutarse. El segundo proceso podría intentar entonces leer los datos, que se encuentran en un estado incoherente. En tales situaciones, son necesarios nuevos mecanismos para coordinar el acceso a los datos compartidos; veremos este tema en el Capítulo 6.

La técnica de desalojo también afecta al diseño del *kernel* del sistema operativo. Durante el procesamiento de una llamada al sistema, el *kernel* puede estar ocupado realizando una actividad en nombre de un proceso. Tales actividades pueden implicar cambios importantes en los datos del *kernel* (por ejemplo, en las colas de E/S). ¿Qué ocurre si el proceso se desaloja en mitad de estos cambios y el *kernel* (o el controlador del dispositivo) necesita leer o modificar la misma estructura? El resultado será un auténtico caos. Ciertos sistemas operativos, incluyendo la mayor parte de las versiones de UNIX, resuelven este problema esperando a que se complete la llamada al sistema o a que se transfiera un bloque de E/S antes de hacer un cambio de contexto. Esta solución permite obtener una estructura del *kernel* simple, ya que el *kernel* no desalojará ningún proceso mientras que las estructuras de datos del *kernel* se encuentren en un estado incoherente. Lamentablemente, este modelo de ejecución del *kernel* no resulta adecuado para permitir la realización de cálculos en tiempo real y el multiprocesamiento. Estos problemas y sus soluciones se describen en las Secciones 5.4 y 19.5.

Dado que, por definición, las interrupciones pueden producirse en cualquier momento, y puesto que no siempre pueden ser ignoradas por el *kernel*, las secciones de código afectadas por las interrupciones deben ser resguardadas de un posible uso simultáneo. Sin embargo, el sistema operativo tiene que aceptar interrupciones casi continuamente, ya que de otra manera podrían perderse valores de entrada o sobreescribirse los valores de salida; por esto, para que no puedan acceder de forma concurrente varios procesos a estas secciones de código, lo que se hace es desactivar las interrupciones al principio de cada sección y volver a activarlas al final. Es importante observar que no son muy numerosas las secciones de código que desactivan las interrupciones y que, normalmente, esas secciones contienen pocas instrucciones.

5.1.4 Despachador

Otro componente implicado en la función de planificación de la CPU es el **despachador**. El despachador es el módulo que proporciona el control de la CPU a los procesos seleccionados por el planificador a corto plazo. Esta función implica lo siguiente:

- Cambio de contexto.
- Cambio al modo usuario.
- Salto a la posición correcta dentro del programa de usuario para reiniciar dicho programa.

El despachador debe ser lo más rápido posible, ya que se invoca en cada conmutación de proceso. El tiempo que tarda el despachador en detener un proceso e iniciar la ejecución de otro se conoce como **latencia de despacho**.

5.2 Criterios de planificación

Los diferentes algoritmos de planificación de la CPU tienen distintas propiedades, y la elección de un algoritmo en particular puede favorecer una clase de procesos sobre otros. A la hora de decidir qué algoritmo utilizar en una situación particular, debemos considerar las propiedades de los diversos algoritmos.

Se han sugerido muchos criterios para comparar los distintos algoritmos de planificación. Las características que se usen para realizar la comparación pueden afectar enormemente a la determinación de cuál es el mejor algoritmo. Los criterios son los siguientes:

- **Utilización de la CPU.** Deseamos mantener la CPU tan ocupada como sea posible. Conceptualmente, la utilización de la CPU se define en el rango comprendido entre el 0 y el 100 por cien. En un sistema real, debe variar entre el 40 por ciento (para un sistema ligeramente cargado) y el 90 por ciento (para un sistema intensamente utilizado).

- **Tasa de procesamiento.** Si la CPU está ocupada ejecutando procesos, entonces se estará llevando a cabo algún tipo de trabajo. Una medida de esa cantidad de trabajo es el número de procesos que se completan por unidad de tiempo, y dicha medida se denomina *tasa de procesamiento*. Para procesos de larga duración, este valor puede ser de un proceso por hora; para transacciones cortas, puede ser de 10 procesos por segundo.

- **Tiempo de ejecución.** Desde el punto de vista de un proceso individual, el criterio importante es cuánto tarda en ejecutarse dicho proceso. El intervalo que va desde el instante en que se ordena la ejecución de un proceso hasta el instante en que se completa es el *tiempo de ejecución*. Ese tiempo de ejecución es la suma de los períodos que el proceso invierte en esperar para cargarse en memoria, esperar en la cola de procesos preparados, ejecutarse en la CPU y realizar las operaciones de E/S.

- **Tiempo de espera.** El algoritmo de planificación de la CPU no afecta a la cantidad de tiempo durante la que un proceso se ejecuta o hace una operación de E/S; afecta sólo al período de tiempo que un proceso invierte en esperar en la cola de procesos preparados. El *tiempo de espera* es la suma de los períodos invertidos en esperar en la cola de procesos preparados.

- **Tiempo de respuesta.** En un sistema interactivo, el tiempo de ejecución puede no ser el mejor criterio. A menudo, un proceso puede generar parte de la salida con relativa rapidez y puede continuar calculando nuevos resultados mientras que los resultados previos se envían a la salida para ponerlos a disposición del usuario. Por tanto, otra medida es el tiempo transcurrido desde que se envía una solicitud hasta que se produce la primera respuesta. Esta medida, denominada *tiempo de respuesta*, es el tiempo que el proceso tarda en empezar a responder, no el tiempo que tarda en enviar a la salida toda la información de respuesta. Generalmente, el tiempo de respuesta está limitado por la velocidad del dispositivo de salida.

El objetivo consiste en maximizar la utilización de la CPU y la tasa de procesamiento, y minimizar el tiempo de ejecución, el tiempo de espera y el tiempo de respuesta. En la mayoría de los casos, lo que se hace es optimizar algún tipo de valor promedio. Sin embargo, en determinadas circunstancias, resulta deseable optimizar los valores máximo y mínimo en lugar del promedio. Por ejemplo, para garantizar que todos los usuarios tengan un buen servicio, podemos tratar de minimizar el tiempo de respuesta máximo.

Diversos investigadores han sugerido que, para los sistemas interactivos (como, por ejemplo, los sistemas de tiempo compartido), es más importante minimizar la *varianza* del tiempo de respuesta que minimizar el tiempo medio de respuesta. Un sistema con un tiempo de respuesta razonable y *predecible* puede considerarse más deseable que un sistema que sea más rápido por término medio, pero que resulte altamente variable. Sin embargo, no se han llevado a cabo muchas investigaciones sobre algoritmos de planificación de la CPU que minimicen la varianza.

A medida que estudiemos en la sección siguiente los diversos algoritmos de planificación de la CPU, trataremos de ilustrar su funcionamiento. Para poder ilustrar ese funcionamiento de forma precisa, sería necesario utilizar muchos procesos, compuesto cada uno de una secuencia de varios cientos de ráfagas de CPU y de E/S. No obstante, con el fin de simplificar, en nuestros ejemplos sólo vamos a considerar una ráfaga de CPU (en milisegundos) por cada proceso. La medida de comparación que hemos empleado es el tiempo medio de espera. En la Sección 5.7 se presenta un mecanismo de evaluación más elaborado.

5.3 Algoritmos de planificación

La cuestión de la planificación de la CPU aborda el problema de decidir a qué proceso de la cola de procesos preparados debe asignársele la CPU. Existen muchos algoritmos de planificación de la CPU; en esta sección, describiremos algunos de ellos.

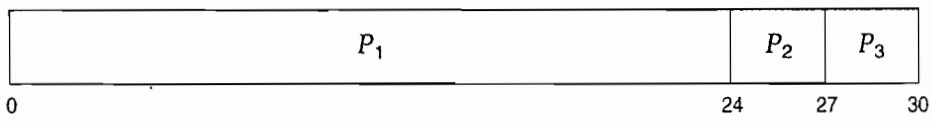
5.3.1 Planificación FCFS

El algoritmo más simple de planificación de la CPU es, con mucho, el algoritmo FCFS (first-come, first-served; primero en llegar, primero en ser servido). Con este esquema, se asigna primero la CPU al proceso que primero la solicite. La implementación de la política FCFS se gestiona fácilmente con una cola FIFO. Cuando un proceso entra en la cola de procesos preparados, su PCB se coloca al final de la cola. Cuando la CPU queda libre, se asigna al proceso que esté al principio de la cola y ese proceso que pasa a ejecutarse se elimina de la cola. El código del algoritmo de planificación FCFS es simple de escribir y fácil de comprender.

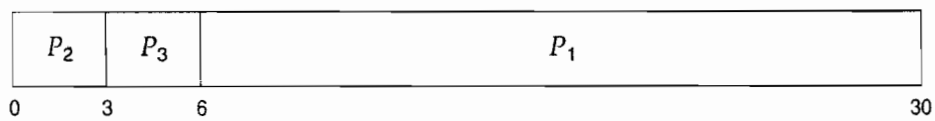
Sin embargo, el tiempo medio de espera con el algoritmo FCFS es a menudo bastante largo. Suponga que el siguiente conjunto de procesos llega en el instante 0, estando la duración de la ráfaga de CPU especificada en milisegundos:

Proceso	Tiempo de ráfaga
P_1	24
P_2	3
P_3	3

Si los procesos llegan en el orden P_1, P_2, P_3, y se sirven según el orden FCFS, obtendremos el resultado mostrado en el siguiente **diagrama de Gantt**:



El tiempo de espera es de 10 milisegundos para el proceso P_1, de 24 milisegundos para el proceso P_2 y de 27 milisegundos para el proceso P_3. Por tanto, el tiempo medio de espera es de (0 + 24 +27)/3 = 17 milisegundos. Sin embargo, si los procesos llegan en el orden P_2, P_3, P_1, los resultados serán los mostrados en el siguiente diagrama de Gantt:



Ahora el tiempo de espera es de (6 + 0 + 3)/3 = 3 milisegundos. Esta reducción es sustancial. Por tanto, el tiempo medio de espera con una política FCFS no es, generalmente, mínimo y puede variar significativamente si la duración de las ráfagas de CPU de los procesos es muy variable.

Además, considere el comportamiento del algoritmo de planificación FCFS en un caso dinámico. Suponga que tenemos un proceso limitado por CPU y muchos procesos limitados por E/S. A medida que los procesos fluyen por el sistema, puede llegarse al siguiente escenario: el proceso limitado por CPU obtendrá y mantendrá la CPU; durante ese tiempo, los demás procesos terminarán sus operaciones de E/S y pasarán a la cola de procesos preparados, esperando a acceder a la CPU. Mientras que los procesos esperan en la cola de procesos preparados, los dispositivos de E/S están inactivos. Finalmente, el proceso limitado por CPU termina su ráfaga de CPU y pasa a esperar por un dispositivo de E/S. Todos los procesos limitados por E/S, que tienen ráfagas de CPU cortas, se ejecutan rápidamente y vuelven a las colas de E/S. En este punto, la CPU permanecerá inactiva. El proceso limitado por CPU volverá en algún momento a la cola de procesos preparados

y se le asignará la CPU. De nuevo, todos los procesos limitados por E/S terminarán esperando en la cola de procesos preparados hasta que el proceso limitado por CPU haya terminado. Se produce lo que se denomina un **efecto convoy** a medida que todos los procesos restantes esperan a que ese único proceso de larga duración deje de usar la CPU. Este efecto da lugar a una utilización de la CPU y de los dispositivos menor que la que se conseguiría si se permitiera a los procesos más cortos ejecutarse primero.

El algoritmo de planificación FCFS es cooperativo. Una vez que la CPU ha sido asignada a un proceso, dicho proceso conserva la CPU hasta que la libera, bien porque termina su ejecución o porque realiza una solicitud de E/S. El algoritmo FCFS resulta, por tanto, especialmente problemático en los sistemas de tiempo compartido, donde es importante que cada usuario obtenga una cuota de la CPU a intervalos regulares. Sería desastroso permitir que un proceso mantuviera la CPU durante un largo período de tiempo.

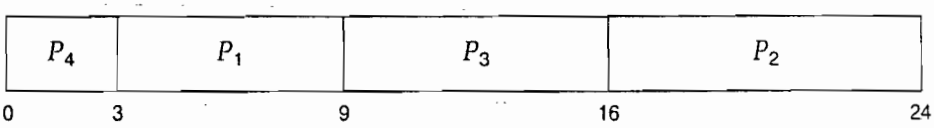
5.3.2 Planificación SJF

Otro método de planificación de la CPU es el algoritmo de planificación con selección del trabajo más corto (SJF, shortest-job-first). Este algoritmo asocia con cada proceso la duración de la siguiente ráfaga de CPU del proceso. Cuando la CPU está disponible, se asigna al proceso que tiene la siguiente ráfaga de CPU más corta. Si las siguientes ráfagas de CPU de dos procesos son iguales, se usa la planificación FCFS para romper el empate. Observe que un término más apropiado para este método de planificación sería el de *algoritmo de la siguiente ráfaga de CPU más corta*, ya que la planificación depende de la duración de la siguiente ráfaga de CPU de un proceso, en lugar de depender de su duración total. Usamos el término SJF porque casi todo el mundo y gran parte de los libros de texto emplean este término para referirse a este tipo de planificación.

Como ejemplo de planificación SJF, considere el siguiente conjunto de procesos, estando especificada la duración de la ráfaga de CPU en milisegundos:

Proceso	Tiempo de ráfaga
P_1	6
P_2	8
P_3	7
P_4	3

Usando la planificación SJF, planificaríamos estos procesos de acuerdo con el siguiente diagrama de Gantt:



El tiempo de espera es de 3 milisegundos para el proceso P_1, de 16 milisegundos para el proceso P_2, de 9 milisegundos para P_3 y de 0 milisegundos para P_4. Por tanto, el tiempo medio de espera es de (3 + 16 + 9 + 0)/4 = 7 milisegundos. Por comparación, si estuviéramos usando el esquema de planificación FCFS, el tiempo medio de espera sería de 10,25 milisegundos.

El algoritmo de planificación SJF es probablemente *óptimo*, en el sentido de que proporciona el tiempo medio de espera mínimo para un conjunto de procesos dado. Anteponer un proceso corto a uno largo disminuye el tiempo de espera del proceso corto en mayor medida de lo que incrementa el tiempo de espera del proceso largo. Consecuentemente, el tiempo *medio* de espera disminuye.

La dificultad real del algoritmo SJF es conocer la duración de la siguiente solicitud de CPU. En una planificación a largo plazo de trabajos en un sistema de procesamiento por lotes, podemos usar como duración el límite de tiempo del proceso que el usuario especifique en el momento de enviar el trabajo. Con este mecanismo, los usuarios están motivados para estimar el límite de tiempo del proceso de forma precisa, dado que un valor menor puede significar una respuesta más

rápida. (Un valor demasiado bajo producirá un error de límite de tiempo excedido y será necesario reenviar el proceso.) La planificación SJF se usa frecuentemente como mecanismo de planificación a largo plazo.

Aunque el algoritmo SJF es óptimo, no se puede implementar en el nivel de la planificación de la CPU a corto plazo, ya que no hay forma de conocer la duración de la siguiente ráfaga de CPU. Un método consiste en intentar aproximar la planificación SJF: podemos no *conocer* la duración de la siguiente ráfaga de CPU, pero podemos *predecir* su valor, por el procedimiento de confiar en que la siguiente ráfaga de CPU será similar en duración a las anteriores. De este modo, calculando una aproximación de la duración de la siguiente ráfaga de CPU, podemos tomar el proceso que tenga la ráfaga de CPU predicha más corta.

Generalmente, la siguiente ráfaga de CPU se predice como la media exponencial de las duraciones medidas de las anteriores ráfagas de CPU. Sea t_n la duración de la n-ésima ráfaga de CPU y sea τ_{n+1} el valor predicho para la siguiente ráfaga de CPU. Entonces, para α, $0 \leq \alpha \leq 1$, se define

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

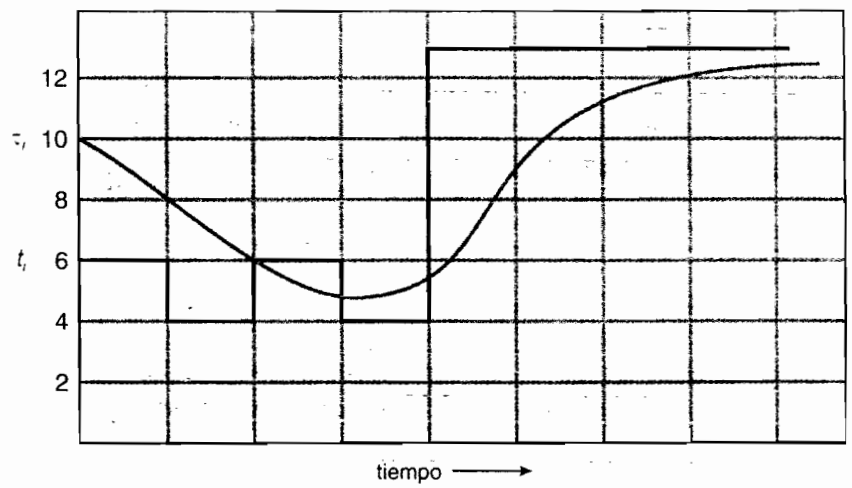
Esta fórmula define un **promedio exponencial**. El valor de t_n contiene la información más reciente; τ_n almacena el historial pasado. El parámetro α controla el peso relativo del historial reciente y pasado de nuestra predicción. Si $\alpha = 0$, entonces $\tau_{n+1} = \tau_n$, y el historial reciente no tiene ningún efecto (se supone que las condiciones actuales van a ser transitorias); si $\alpha = 1$, entonces $\tau_{n+1} = t_n$, y sólo la ráfaga de CPU más reciente importa (se supone que el historial es obsoleto y, por tanto, irrelevante). Frecuentemente, $\alpha = \frac{1}{2}$, en cuyo caso, el historial reciente y pasado tienen el mismo peso. El valor inicial τ_0 puede definirse como una constante o como un promedio global para todo el sistema. La Figura 5.3 muestra un promedio exponencial con $\alpha = \frac{1}{2}$ y $\tau_0 = 10$.

Para comprender el comportamiento del promedio exponencial, podemos desarrollar la fórmula para τ_{n+1} sustituyendo el valor de τ_n y de los términos sucesivos, obteniendo

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \cdots + (1-\alpha)^j \alpha t_{n-j} + \cdots + (1-\alpha)^{n+1} \tau_0$$

Dado que tanto α como $1-\alpha$ son menores o iguales a 1, cada término sucesivo tiene menor peso que su predecesor.

El algoritmo SJF puede ser cooperativo o apropiativo. La necesidad de elegir surge cuando un proceso llega a la cola de procesos preparados mientras que un proceso anterior está todavía en



ráfaga de CPU (t_i)		6	4	6	4	13	13	13	...
"invitado" (τ_i)	10	8	6	6	5	9	11	12	...

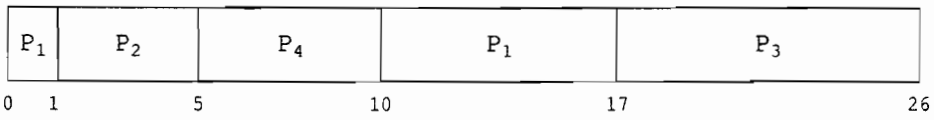
Figura 5.3 Predicción de la duración de la siguiente ráfaga de CPU.

ejecución. La siguiente ráfaga de CPU del proceso que acaba de llegar puede ser más corta que lo que quede del proceso actualmente en ejecución. Un algoritmo SJF apropiativo detendrá el proceso actualmente en ejecución, mientras que un algoritmo sin desalojo permitirá que dicho proceso termine su ráfaga de CPU. La planificación SJF apropiativa a veces se denomina **planificación con selección del proceso con tiempo restante más corto.**

Como ejemplo, considere los cuatro procesos siguientes, estando especificada la duración de la ráfaga de CPU en milisegundos:

Proceso	Tiempo de llegada	Tiempo de ráfaga
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Si los procesos llegan a la cola de procesos preparados en los instantes que se muestran y necesitan los tiempos de ráfaga indicados, entonces la planificación SJF apropiativa es la que se muestra en el siguiente diagrama de Gantt:



El proceso P_1 se inicia en el instante 0, dado que es el único proceso que hay en la cola. El proceso P_2 llega en el instante 1. El tiempo que le queda al proceso P_1 (7 milisegundos) es mayor que el tiempo requerido por el proceso P_2 (4 milisegundos), por lo que el proceso P_1 se desaloja y se planifica el proceso P_2. El tiempo medio de espera en este ejemplo es de $((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 = 6{,}5$ milisegundos. La planificación SJF cooperativa proporcionaría un tiempo medio de espera de 7,75 milisegundos.

5.3.3 Planificación por prioridades

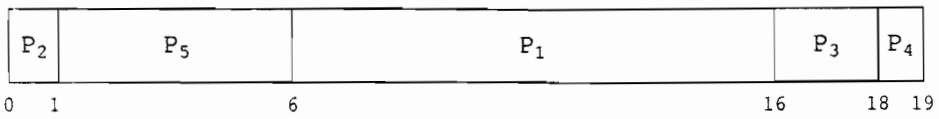
El algoritmo SJF es un caso especial del **algoritmo de planificación por prioridades** general. A cada proceso se le asocia una prioridad y la CPU se asigna al proceso que tenga la prioridad más alta. Los procesos con la misma prioridad se planifican en orden FCFS. Un algoritmo SJF es simplemente un algoritmo por prioridades donde la prioridad (p) es el inverso de la siguiente ráfaga de CPU (predicha). Cuanto más larga sea la ráfaga de CPU, menor será la prioridad y viceversa.

Observe que al hablar de planificación pensamos en términos de *alta* prioridad y *baja* prioridad. Generalmente, las prioridades se indican mediante un rango de números fijo, como por ejemplo de 0 a 7, o de 0 a 4095. Sin embargo, no existe un acuerdo general sobre si 0 es la prioridad más alta o la más baja. Algunos sistemas usan los números bajos para representar una prioridad baja; otros, emplean números bajos para especificar una prioridad alta; esta diferencia puede llevar a confusión. En este texto, vamos a asumir que los números bajos representan una alta prioridad.

Por ejemplo, considere el siguiente conjunto de procesos y suponga que llegan en el instante 0 en este orden: $P_1, P_2, \ldots, P_5$, estando la duración de las ráfagas de CPU especificada en milisegundos:

Proceso	Tiempo de ráfaga	Prioridad
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Usando la planificación por prioridades, vamos a planificar estos procesos de acuerdo con el siguiente diagrama de Gantt:



El tiempo medio de espera es de 8,2 milisegundos.

Las prioridades pueden definirse interna o externamente. Las prioridades definidas internamente utilizan algún valor mensurable para calcular la prioridad de un proceso. Por ejemplo, para calcular las prioridades se han usado en diversos sistemas magnitudes tales como los límites de tiempo, los requisitos de memoria, el número de archivos abiertos y la relación entre la ráfaga de E/S promedio y la ráfaga de CPU promedio. Las prioridades definidas externamente se establecen en función de criterios externos al sistema operativo, como por ejemplo la importancia del proceso, el coste monetario de uso de la computadora, el departamento que patrocina el trabajo y otros factores, a menudo de carácter político.

La planificación por prioridades puede ser apropiativa o cooperativa. Cuando un proceso llega a la cola de procesos preparados, su prioridad se compara con la prioridad del proceso actualmente en ejecución. Un algoritmo de planificación por prioridades apropiativo, expulsará de la CPU al proceso actual si la prioridad del proceso que acaba de llegar es mayor. Un algoritmo de planificación por prioridades cooperativo simplemente pondrá el nuevo proceso al principio de la cola de procesos preparados.

Un problema importante de los algoritmos de planificación por prioridades es el **bloqueo indefinido** o la **muerte por inanición.** Un proceso que está preparado para ejecutarse pero está esperando a acceder a la CPU puede considerarse bloqueado; un algoritmo de planificación por prioridades puede dejar a algunos procesos de baja prioridad esperando indefinidamente. En un sistema informático con una carga de trabajo grande, un flujo estable de procesos de alta prioridad puede impedir que un proceso de baja prioridad llegue a la CPU. Generalmente, ocurrirá una de dos cosas: o bien el proceso se ejecutará finalmente (a las 2 de la madrugada del domingo, cuando el sistema finalmente tenga una menor carga de trabajo) o bien el sistema informático terminará fallando y se perderán todos los procesos con baja prioridad no terminados. Se rumorea que, en 1973, en el MIT, cuando apagaron el IBM 7094, encontraron un proceso de baja prioridad que había sido enviado en 1967 y todavía no había sido ejecutado.

Una solución al problema del bloqueo indefinido de los procesos de baja prioridad consiste en aplicar mecanismos de **envejecimiento**. Esta técnica consiste en aumentar gradualmente la prioridad de los procesos que estén esperando en el sistema durante mucho tiempo. Por ejemplo, si el rango de prioridades va desde 127 (baja) a 0 (alta), podríamos restar 1 a la prioridad de un proceso en espera cada 15 minutos. Finalmente, incluso un proceso con una prioridad inicial de 127 llegaría a tener la prioridad más alta del sistema y podría ejecutarse. De hecho, un proceso con prioridad 127 no tardaría más de 32 horas en convertirse en un proceso con prioridad 0.

5.3.4 Planificación por turnos

El algoritmo de planificación por turnos (RR, round robin) está diseñado especialmente para los sistemas de tiempo compartido. Es similar a la planificación FCFS, pero se añade la técnica de desalojo para conmutar entre procesos. En este tipo de sistema se define una pequeña unidad de tiempo, denominada **cuanto de tiempo**, o franja temporal. Generalmente, el cuanto de tiempo se encuentra en el rango comprendido entre 10 y 100 milisegundos. La cola de procesos preparados se trata como una cola circular. El planificador de la CPU recorre la cola de procesos preparados, asignando la CPU a cada proceso durante un intervalo de tiempo de hasta 1 cuanto de tiempo.

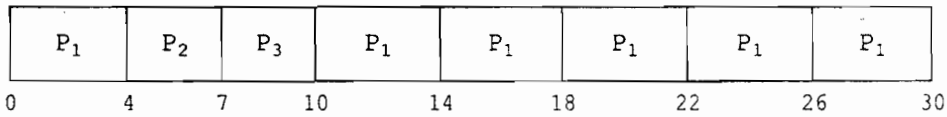
Para implementar la planificación por turnos, mantenemos la cola de procesos preparados como una cola FIFO de procesos. Los procesos nuevos se añaden al final de la cola de procesos preparados. El planificador de la CPU toma el primer proceso de la cola de procesos preparados, configura un temporizador para que interrumpa pasado 1 cuanto de tiempo y despacha el proceso.

Puede ocurrir una de dos cosas. El proceso puede tener una ráfaga de CPU cuya duración sea menor que 1 cuanto de tiempo; en este caso, el propio proceso liberará voluntariamente la CPU. El planificador continuará entonces con el siguiente proceso de la cola de procesos preparados. En caso contrario, si la ráfaga de CPU del proceso actualmente en ejecución tiene una duración mayor que 1 cuanto de tiempo, se producirá un fin de cuenta del temporizador y éste enviará una interrupción al sistema operativo; entonces se ejecutará un cambio de contexto y el proceso se colocará al **final** de la cola de procesos preparados. El planificador de la CPU seleccionará a continuación el siguiente proceso de la cola de procesos preparados.

El tiempo medio de espera en los sistemas por turnos es, con frecuencia, largo. Considere el siguiente conjunto de procesos que llegan en el instante 0, estando especificada la duración de las ráfagas de CPU en milisegundos:

Proceso	Tiempo de ráfaga
P_1	24
P_2	3
P_3	3

Si usamos un cuanto de tiempo de 4 milisegundos, entonces el proceso P_1 obtiene los 4 primeros milisegundos. Dado que necesita otros 20 milisegundos, es desalojado después del primer cuanto de tiempo, y la CPU se concede al siguiente proceso de la cola, el proceso P_2. Puesto que el proceso P_2 no necesita 4 milisegundos, termina antes de que caduque su cuanto de tiempo. Entonces la CPU se proporciona al siguiente proceso, el proceso P_3. Una vez que cada proceso ha recibido 1 cuanto de tiempo, la CPU es devuelta al proceso P_1 para un cuanto de tiempo adicional. La planificación por turnos resultante es:



El tiempo medio de espera es de 17/3 = 5,66 milisegundos.

En el algoritmo de planificación por turnos, a ningún proceso se le asigna la CPU por más de 1 cuanto de tiempo en cada turno (a menos que sea el único proceso ejecutable). Si una ráfaga de CPU de un proceso excede 1 cuanto de tiempo, dicho proceso se *desaloja* y se coloca de nuevo en la cola de procesos preparados. El algoritmo de planificación por turnos incluye, por tanto, un mecanismo de desalojo.

Si hay n procesos en la cola de procesos preparados y el cuanto de tiempo es q, cada proceso obtiene $1/n$ del tiempo de CPU en partes de como máximo q unidades de tiempo. Cada proceso no tiene que esperar más de $(n - 1) \times q$ unidades de tiempo hasta obtener su siguiente cuanto de tiempo. Por ejemplo, con cinco procesos y un cuanto de tiempo de 20 milisegundos, cada proceso obtendrá 20 milisegundos cada 100 milisegundos.

El rendimiento del algoritmo de planificación por turnos depende enormemente del tamaño del cuanto de tiempo. Por un lado, si el cuanto de tiempo es extremadamente largo, la planificación por turnos es igual que la FCFS. Si el cuanto de tiempo es muy pequeño (por ejemplo, 1 milisegundo), el método por turnos se denomina **compartición del procesador** y (en teoría) crea la apariencia de que cada uno de los n procesos tiene su propio procesador ejecutándose a $1/n$ de la velocidad del procesador real. Este método se usaba en las máquinas de Control Data Corporation (CDC) para implementar diez procesadores periféricos con sólo un conjunto de hardware y diez conjuntos de registros. El hardware ejecuta una instrucción para un conjunto de registros y luego pasa al siguiente; este ciclo se repite, dando lugar en la práctica a diez procesadores lentos, en lugar de uno rápido. Realmente, dado que el procesador era mucho más rápido que la memoria y cada instrucción hacía referencia a memoria, los procesadores no eran mucho más lentos que lo que hubieran sido diez procesadores reales.

En software, también necesitamos considerar el efecto del cambio de contexto en el rendimiento de la planificación por turnos. Suponga que sólo tenemos un proceso de 10 unidades de tiem-

po. Si el cuanto tiene 12 unidades de tiempo, el proceso termina en menos de 1 cuanto de tiempo, sin requerir ninguna carga de trabajo adicional de cambio de contexto. Sin embargo, si el cuanto de tiempo dura 6 unidades, el proceso requerirá 2 cuantos, requiriendo un cambio de contexto. Si el cuanto de tiempo es de 1 unidad, entonces se producirán nueve cambios de contexto, ralentizando correspondientemente la ejecución del proceso (Figura 5.4).

Por tanto, conviene que el cuanto de tiempo sea grande con respecto al tiempo requerido por un cambio de contexto. Si el tiempo de cambio de contexto es, aproximadamente, el 10 por ciento del cuanto de tiempo, entonces un 10 por ciento del tiempo de CPU se invertirá en cambios de contexto. En la práctica, los sistemas más modernos emplean cuantos de tiempo en el rango de 10 a 100 milisegundos y el tiempo requerido para un cambio de contexto es, normalmente, menor que 10 microsegundos; por tanto, el tiempo de cambio de contexto es una fracción pequeña del cuanto de tiempo.

El tiempo de ejecución también depende del valor del cuanto de tiempo. Como podemos ver en la Figura 5.5, el tiempo medio de ejecución de un conjunto de procesos no necesariamente mejora cuando se incrementa el cuanto de tiempo. En general, el tiempo medio de ejecución puede mejorarse si la mayor parte de los procesos termina su siguiente ráfaga de CPU en un solo cuanto de tiempo. Por ejemplo, dados tres procesos con una duración, cada uno de ellos, de 10 unidades de tiempo y un cuanto igual a 1 unidad de tiempo, el tiempo medio de ejecución será de 29 unidades. Sin embargo, si el cuanto de tiempo es 10, el tiempo medio de ejecución cae a 20. Si se añade el tiempo de cambio de contexto, el tiempo medio de ejecución aumenta al disminuir el cuanto de tiempo, dado que serán necesarios más cambios de contexto.

Aunque el cuanto de tiempo debe ser grande comparado con el tiempo de cambio de contexto, no debe ser demasiado grande. Si el cuanto de tiempo es demasiado largo, la planificación por turnos degenera en el método FCFS. Una regla práctica es que el 80 por ciento de las ráfagas de CPU deben ser más cortas que el cuanto de tiempo.

5.3.5 Planificación mediante colas multinivel

Otra clase de algoritmos de planificación es la que se ha desarrollado para aquellas situaciones en las que los procesos pueden clasificarse fácilmente en grupos diferentes. Por ejemplo, una clasificación habitual consiste en diferenciar entre procesos **de primer plano** (interactivos) y procesos **de segundo plano** (por lotes). Estos dos tipos de procesos tienen requisitos diferentes de tiempo de respuesta y, por tanto, pueden tener distintas necesidades de planificación. Además, los procesos de primer plano pueden tener prioridad (definida externamente) sobre los procesos de segundo plano.

Un **algoritmo de planificación mediante colas multinivel** divide la cola de procesos preparados en varias colas distintas (Figura 5.6). Los procesos se asignan permanentemente a una cola, generalmente en función de alguna propiedad del proceso, como por ejemplo el tamaño de memoria, la prioridad del proceso o el tipo de proceso. Cada cola tiene su propio algoritmo de planificación. Por ejemplo, pueden emplearse colas distintas para los procesos de primer plano y de

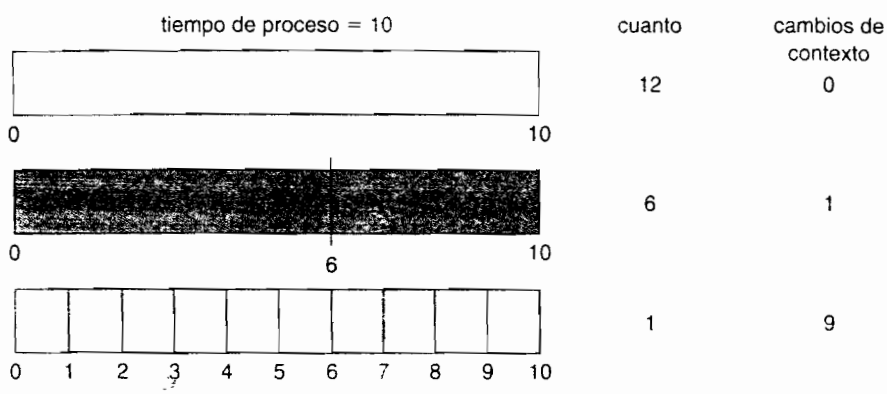
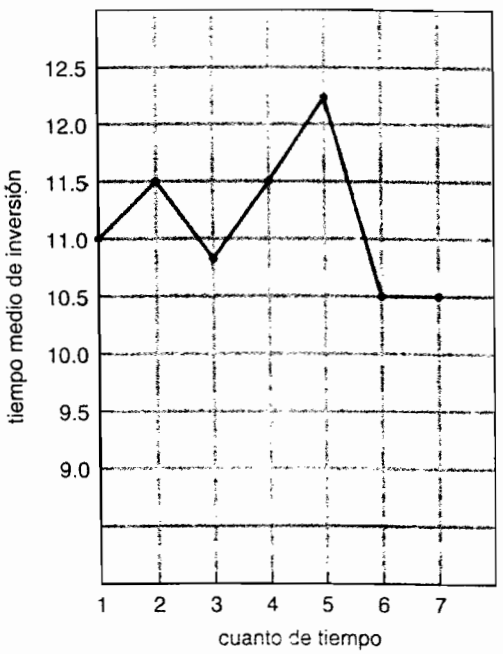


Figura 5.4 La forma en que un cuanto de tiempo muy pequeño incrementa los cambios de contexto.



proceso	tiempo
P_1	6
P_2	3
P_3	1
P_4	7

Figura 5.5 La forma en que varía el tiempo de ejecución con el cuanto de tiempo.

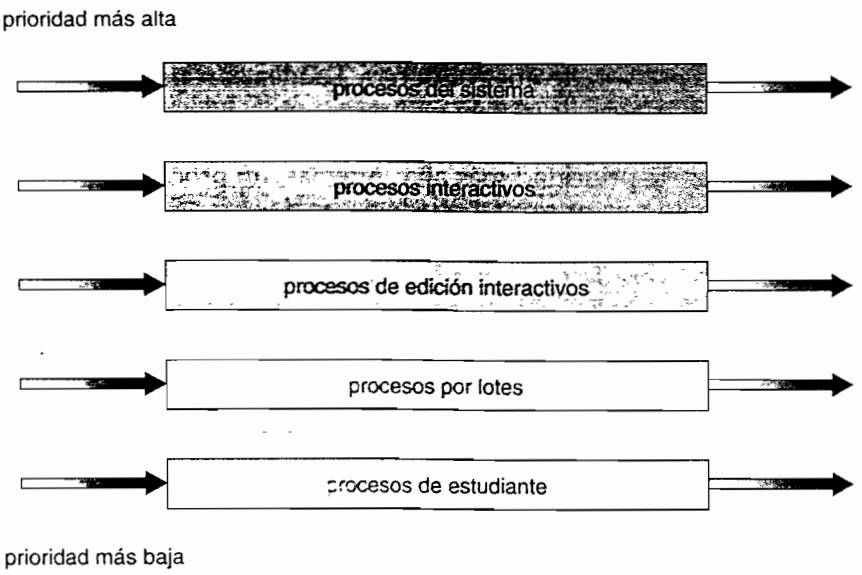


Figura 5.6 Planificación de colas multinivel.

segundo plano. La cola de primer plano puede planificarse mediante un algoritmo por turnos, mientras que para la cola de segundo plano puede emplearse un algoritmo FCFS.

Además, debe definirse una planificación entre las colas, la cual suele implementarse como una planificación apropiativa y prioridad fija. Por ejemplo, la cola de procesos de primer plano puede tener prioridad absoluta sobre la cola de procesos de segundo plano.

Veamos un ejemplo de algoritmo de planificación mediante colas multinivel con las cinco colas que se enumeran a continuación, según su orden de prioridad:

1. Procesos del sistema.
2. Procesos interactivos.

3. Procesos de edición interactivos.
4. Procesos por lotes.
5. Procesos de estudiantes.

Cada cola tiene prioridad absoluta sobre las colas de prioridad más baja. Por ejemplo, ningún proceso de la cola por lotes podrá ejecutarse hasta que se hayan vaciado completamente las colas de los procesos del sistema, los procesos interactivos y los procesos de edición interactivos. Si un proceso de edición interactivo llega a la cola de procesos preparados mientras se está ejecutando un proceso por lotes, el proceso por lotes será desalojado.

Otra posibilidad consiste en repartir el tiempo entre las colas. En este caso, cada cola obtiene una cierta porción del tiempo de CPU, con la que puede entonces planificar sus distintos procesos. Por ejemplo, en el caso de la colas de procesos de primer plano y segundo plano, la cola de primer plano puede disponer del 80 por ciento del tiempo de CPU para planificar por turnos sus procesos, mientras que la cola de procesos de segundo plano recibe el 20 por ciento del tiempo de CPU para gestionar sus procesos mediante el método FCFS.

5.3.6 Planificación mediante colas multinivel realimentadas

Normalmente, cuando se usa el algoritmo de planificación mediante colas multinivel, los procesos se asignan de forma permanente a una cola cuando entran en el sistema. Por ejemplo, si hay colas diferentes para los procesos de primer y segundo plano, los procesos no se mueven de una cola a otra, dado que no pueden cambiar su naturaleza de proceso de primer o segundo plano. Esta configuración presenta la ventaja de una baja carga de trabajo de planificación, pero resulta poco flexible.

Por el contrario, el **algoritmo de planificación mediante colas multinivel realimentadas** permite mover un proceso de una cola a otra. La idea es separar los procesos en función de las características de sus ráfagas de CPU. Si un proceso utiliza demasiado tiempo de CPU, se pasa a una cola de prioridad más baja. Este esquema deja los procesos limitados por E/S y los procesos interactivos en las colas de prioridad más alta. Además, un proceso que esté esperando demasiado tiempo en una cola de baja prioridad puede pasarse a una cola de prioridad más alta. Este mecanismo de envejecimiento evita el bloqueo indefinido.

Por ejemplo, considere un planificador de colas multinivel realimentadas con tres colas, numeradas de 0 a 2 (Figura 5.7). En primer lugar, el planificador ejecuta todos los procesos de la cola 0. Sólo cuando la cola 0 esté vacía ejecutará procesos de la cola 1. De forma similar, los procesos de la cola 2 solo se ejecutarán si las colas 0 y 1 están vacías. Un proceso que llegue a la cola 1 desalojará a un proceso de la cola 2 y ese proceso de la cola 1 será, a su vez, desalojado por un proceso que llegue a la cola 0.

Un proceso que entre en la cola de procesos preparados se coloca en la cola 0 y a cada uno de los procesos de esa cola se le proporciona un cuanto de tiempo de 8 milisegundos. Si el proceso no termina en ese tiempo, se pasa al final de la cola 1. Si la cola 0 está vacía, al proceso que se encuentra al principio de la cola 1 se le asigna un cuanto de 16 milisegundos. Si no se completa en ese tiempo, se lo desaloja y se lo incluye en la cola 2. Los procesos de la cola 2 se ejecutan basándose en una planificación FCFS, pero sólo cuando las colas 0 y 1 están vacías.

Este algoritmo de planificación proporciona la prioridad más alta a todo proceso que tenga una ráfaga de CPU de 8 milisegundos o menos. Tales procesos acceden rápidamente a la CPU, concluyen su ráfaga de CPU y pasan a su siguiente ráfaga de E/S. Los procesos que necesitan más de 8 milisegundos y menos de 24 milisegundos también son servidor rápidamente, aunque con una prioridad más baja que los procesos más cortos. Los procesos largos terminan yendo automáticamente a la cola 2 y se sirven, siguiendo el orden FCFS, con los ciclos de CPU no utilizados por las colas 0 y 1.

En general, un planificador mediante colas multinivel realimentadas se define mediante los parámetros siguientes:

- El número de colas.

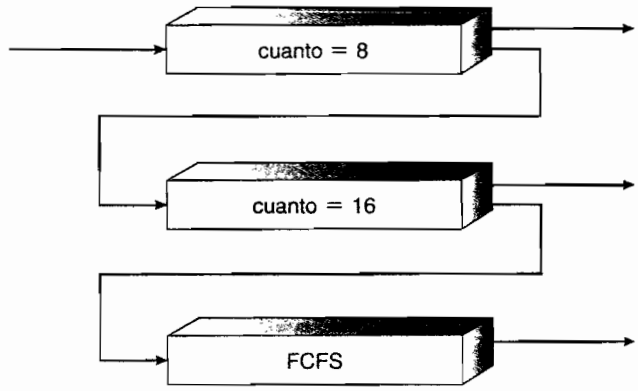


Figura 5.7 Colas multinivel realimentadas.

- El algoritmo de planificación de cada cola.
- El método usado para determinar cuándo pasar un proceso a una cola de prioridad más alta.
- El método usado para determinar cuándo pasar un proceso a una cola de prioridad más baja.
- El método usado para determinar en qué cola se introducirá un proceso cuando haya que darle servicio.

La definición del planificador mediante colas multinivel realimentadas le convierte en el algoritmo de planificación de la CPU más general. Puede configurarse este algoritmo para adaptarlo a cualquier sistema específico que se quiera diseñar. Lamentablemente, también es el algoritmo más complejo, puesto que definir el mejor planificador requiere disponer de algún mecanismo para seleccionar los valores de todos los parámetros.

5.4 Planificación de sistemas multiprocesador

Hasta el momento, nuestra exposición se ha centrado en los problemas de planificación de la CPU en un sistema con un solo procesador. Si hay disponibles múltiples CPU, se puede **compartir la carga**; sin embargo, el problema de la planificación se hace más complejo. Se han probado muchas posibilidades; y como ya hemos visto para el caso de la planificación de la CPU con un único procesador, no existe una solución única. Aquí vamos a presentar diversas cuestiones acerca de la planificación multiprocesador. Vamos a concentrarnos en los sistemas en los que los procesadores son idénticos, **homogéneos** en cuanto a su funcionalidad; en este tipo de sistemas, podemos usar cualquiera de los procesadores disponibles para ejecutar cualquier proceso de la cola. (Sin embargo, observe que incluso con múltiples procesadores homogéneos, en ocasiones hay limitaciones que afectan a la planificación; considere un sistema con un dispositivo de E/S conectado a un bus privado de un procesador. Los procesos que deseen emplear ese dispositivo deberán planificarse para ser ejecutados en dicho procesador.)

5.4.1 Métodos de planificación en los sistemas multiprocesador

Un método para planificar las CPU en un sistema multiprocesador consiste en que todas las decisiones sobre la planificación, el procesamiento de E/S y otras actividades del sistema sean gestionadas por un mismo procesador, el servidor maestro. Los demás procesadores sólo ejecutan código de usuario. Este **multiprocesamiento asimétrico** resulta simple, porque sólo hay un procesador que accede a las estructuras de datos del sistema, reduciendo la necesidad de compartir datos.

Un segundo método utiliza el **multiprocesamiento simétrico** (SMP), en el que cada uno de los procesadores se auto-planifica. Todos los procesos pueden estar en una cola común de procesos