



2º de Grado en Ingeniería Informática
Sistemas Operativos



TEMA 4 – PLANIFICACIÓN

Bibliografía

El contenido de este documento se ha elaborado, principalmente, a partir de las siguientes referencias bibliográficas, y con propósito meramente académico y no lucrativo:

- W. Stallings. *Sistemas operativos, 5ª edición.* Prentice Hall, Madrid, 2005.
- A. S. Tanenbaum. *Sistemas operativos modernos, 3a edición.* Prentice Hall, Madrid, 2009.
- A. Silberschatz, G. Gagne, P. B. Galvin. *Fundamentos de sistemas operativos, séptima edición.* McGraw-Hill, 2005.
- A. McIver, I. M. Flynn. *Sistemas operativos, 6ª edición.* Cengage Learning, 2011.
- J. A. Alamansa, M. A. Canto Diaz, J. M. de la Cruz García, S. Dormido Bencomo, C. Mañoso Hierro. *Sistemas operativos, teoría y problemas.* Editorial Sanz y Torres, S.L, 2002.
- F. Pérez, J. Carretero, F. García. *Problemas de sistemas operativos: de la base al diseño, 2ª edición.* McGraw-Hill, 2003.
- S. Candela, C. Rubén, A. Quesada, F. J. Santana, J. M. Santos. *Fundamentos de Sistemas Operativos, teoría y ejercicios resueltos.* Paraninfo, 2005.
- J. Aranda, M. A. Canto, J. M. de la Cruz, S. Dormido, C. Mañoso. *Sistemas Operativos: Teoría y problemas.* Sanz y Torres S.L, 2002.
- J. Carretero, F. García, P. de Miguel, F. Pérez, *Sistemas Operativos: Una visión aplicada.* Mc Graw Hill, 2001.

1 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.

La naturaleza o finalidad 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, ningún proceso debe de acaparar la CPU.
- **Productividad.** La productividad es la cantidad de trabajo desarrollada por unidad de tiempo. Este objetivo es igual a la optimización de la utilización del procesador. Lo que se pretende es maximizar la productividad.
- **Tiempo de respuesta aceptable.** En sistemas de tiempo compartido, múltiples usuarios sobre el mismo sistema, es muy importante que todos los usuarios tengan un tiempo de respuesta aceptable. También se puede aplicar a un sistema mono-usuario.
- **Costo extra.** El tiempo utilizado en la planificación debe ser el mínimo posible, ya que es tiempo inútil para el sistema. Recordemos algunos conceptos, el *dispatcher* es un módulo del SO que da el control de la CPU al proceso seleccionado por el planificador de corto plazo, que se estudiará en las siguientes secciones. El tiempo empleado por el *dispatcher* debe ser la menor posible.
- **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 o se posterga durante mucho tiempo.
- **Degradación aceptable.** El algoritmo o algoritmos de planificación de un sistema, deben de procurar una degradación del rendimiento de forma suave, a medida que sube la carga del sistema, y sin que ocurra una caída global.

2 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 o trabajos.
- **Tiempo de retorno, de residencia, o de estancia (*turnaround time*):** Es el tiempo que transcurre desde el momento en que se crea un trabajo, estado Nuevo, hasta el momento en

que se completa, estado Saliente. Incluye: el tiempo de ejecución efectiva o uso de CPU + el tiempo de espera + tiempos de E/S, etc.

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 (*waiting time*):** El problema del criterio del tiempo de retorno es que incluye muchos factores que no dependen del algoritmo de planificación, como por ejemplo que se produzcan interrupciones u operaciones de entrada/salida mientras un proceso está ejecutando, o incluso de la longitud del propio proceso. Por ello definimos el tiempo de espera, como la suma de tiempos que el proceso está esperando a ser servido, o lo que es lo mismo, la suma de tiempos de permanencia en estados distintos a de Ejecución. El objetivo es minimizar el tiempo medio de espera.
- **Tiempo de respuesta o latencia (*response time*):** Tiempo que transcurre desde que se crea o lanza un proceso hasta que se obtienen sus primeros resultados, es decir, hasta que su primera interacción se produzca. Es una medida buena desde el punto de vista del usuario, al cual le interesa empezar a obtener resultados, aunque no estén completos, lo más pronto posible.

Ejemplos: El “loading...” de un juego o la carga de un pdf de gran tamaño. En este último caso, 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.

- **Utilización de la CPU, eficacia, o tiempo de servicio (*CPU utilization*):** 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 sistemas mientras menos tiempo esté la CPU ociosa mejor).

Desde el punto de vista de un proceso es la suma de los tiempos que éste está en la CPU o el tiempo que necesita para ejecutarse por completo si no hubiera ningún proceso más en el sistema.

3 Modos de decisión

Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan los instantes de tiempo en que se ejecuta la selección de un proceso: 1) **No apropiativos o sin expulsión**, 2) **Apropiativos o con expulsión**.

3.1 No apropiativos o sin expulsión

El planificador selecciona un proceso para ejecutarlo y después deja que se ejecute en CPU hasta que el mismo proceso se bloquea o termina (normal o de manera abrupta).

Con este modelo, incluso aunque un proceso se ejecute durante horas, no se suspenderá de manera forzosa, no existe rodaja de tiempo. Por tanto, no se toman decisiones de planificación durante una interrupción, sino que una vez que se haya completado el procesamiento de la interrupción, se reanuda la ejecución del mismo proceso. El planificador volverá a planificar cuando el proceso termine.

3.2 Apropiativos o con expulsión

El planificador selecciona un proceso para ejecutarlo y la decisión de expulsarlo de la CPU se toma si suceden alguno de estos ejemplos:

- Se ha ejecutado por un máximo de tiempo fijo (rodaja de tiempo). Si sigue en ejecución hasta el final del intervalo o rodaja de tiempo, el proceso se pasa a la lista de Listos y el planificador selecciona otro proceso para ejecutar.
- Llega un nuevo proceso con mayor prioridad.
- Llega una interrupción que pasa un proceso de mayor prioridad de la lista de Bloqueados a la lista de Listos.
- Invocación de una operación de E/S.

Las políticas expulsivas tienen mayor sobrecarga que las no expulsivas, ya que hay muchos más 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 de procesos.

4 Tipos de planificación del procesador

El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador 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**.

- 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 o memoria virtual (*swapping*). En ocasiones hay que decidir si se añade un proceso a la memoria principal o por el contrario, por razones de rendimiento, se añade a memoria de disco o *swap*.
- La planificación a corto plazo conlleva decidir cuál de los procesos Listos para ejecutar es el siguiente a introducir en la CPU.

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, por ejemplo en el caso de que el sistema disponga de poca memoria principal, se encuentre muy llena, o se necesite pasar de estado Nuevo a estado Listo un proceso que ocupe mucha RAM, pasando para ello otros procesos de la lista de Listo o Bloqueados a Swap.
- El planificador a corto plazo se ejecuta mucho más frecuentemente y toma las decisiones de grano fino sobre qué proceso ejecutar el siguiente.

La planificación afecta al rendimiento del sistema porque determina qué proceso esperará y qué proceso progresará. 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.

4.1 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 programa de usuario, éste se convierte en un proceso y se añade a la cola del planificador a corto plazo. En algunos sistemas, si están muy saturados en cuanto al uso de la memoria RAM, un proceso de reciente creación (estado Nuevo) comienza cargándose en la memoria *swap* o virtual, en cuyo caso se añade a la cola del planificador a medio plazo.

La decisión de cuándo pasar a Listo un nuevo proceso se toma dependiendo del grado de multiprogramación deseado. Cuanto mayor sea el número de procesos creados y cargados en RAM, menor será el porcentaje de tiempo en que cada proceso se pueda ejecutar, es decir, más procesos compiten por la CPU. Así, el planificador a largo plazo puede limitar el grado de multiprogramación a fin de proporcionar un servicio satisfactorio al actual conjunto de procesos cargados en RAM, de forma que cada vez que termine un proceso, el planificador puede decidir si añadir o no, uno o más trabajos Nuevos a la lista de Listos.

Por otro lado, si la fracción de tiempo que el procesador está ocioso excede un determinado valor, se puede invocar al planificador a largo plazo para traer procesos desde estado Nuevo a estado Listo.

4.2 Planificación a corto plazo

El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el Bloqueo del proceso actual o su retirada de la CPU a la lista de Listos. En este caso hay que expulsar al proceso actualmente en ejecución en favor de otro. Ténganse como por ejemplo las siguientes circunstancias en las que se puede producir esa expulsión:

- Se produce una interrupciones de reloj por rodaja o ráfaga de tiempo.
- Se produce una interrupción de E/S por parte de un evento que puede interrumpir la ejecución del actual para tratar la interrupción.
- Al pasar un proceso de estar en estado Bloqueado a Listo, porque por ejemplo se finalizó una operación de E/S que solicitó.
- En determinadas llamadas al sistema que provoquen una E/S, como por ejemplo *open()*, *read()*, *close()*.
- Ante determinadas señales o mecanismos de exclusión mutua como los semáforos, los cuales pueden bloquear el proceso actual.
- Cuando un proceso se bloquea a espera de que termine un hijo o un hilo.
- Cuando un proceso finaliza su ejecución, ya que hay que dejarlo de tener en cuenta en la planificación.

En general, cualquier tipo de cambio de estados en un proceso conlleva la invocación del planificador.

La decisión de qué trabajo admitir el siguiente puede basarse en encontrar un compromiso entre **procesos limitados por el procesador** y **procesos limitados por la E/S**. 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:

- Se dice que un **proceso está limitado por el procesador** si realiza mucho trabajo computacional y solo muy ocasionalmente o incluso nunca, 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.

4.3 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).

En el momento en que un proceso pasa al estado de listo, se une a una única 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.

Tenga en cuenta que en este algoritmo de planificación no existen las rodajas de tiempo, es un modelo de decisión no expulsivo, por lo que un proceso se puede ejecutar entero o hasta que, por

ejemplo, realice un operación de E/S por la que tenga que esperar. En caso de que un proceso se Bloquee, pasaría a la lista de Bloqueados en espera de un evento. Cuando ocurra el evento pasará a la lista de Listos.

FCFS funciona mucho mejor para procesos largos que para procesos cortos. Considérese el siguiente ejemplo de procesos limitados por el procesador, donde el *Tiempo de Estancia* = *Tiempo Finalización-Tiempo Llegada* y el *Tiempo de Estancia Normalizado* = $(\text{Tiempo de estancia o retorno} / \text{Tiempo de CPU o de servicio})$:

Proceso	Tiempo de Llegada	Tiempo de Servicio (T_s)	Tiempo de Comienzo	Tiempo de Finalización	Tiempo de Estancia (T_r)	T_r/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 valor de 100 significa que el tiempo total que el proceso está en el sistema es 100 veces mayor que el tiempo requerido para su procesamiento. Esto sucederá siempre que llegue un proceso corto a continuación de un proceso largo (Y llega en el instante 2 y X en el instante 1, pero X es un proceso largo que va a retrasar a Y, que es corto).

Por otra parte, los procesos largos no van mal, por ejemplo el proceso Z tiene un tiempo de estancia de casi el doble que Y (199), pero su tiempo de estancia normalizado está por debajo de 2.0.

En la Figura 4 se muestra otro ejemplo del algoritmo de planificación FCFS. t es el tiempo de servicio, T el tiempo de estancia, P es el tiempo de estancia normalizado, y E se refiere a tiempo de espera (Tiempo de inicio – Tiempo de llegada). Los rectángulos negros se corresponden con el tiempo de espera E .

Proceso	Tiempo de Llegada	t	Inicio	Fin	T	E	P
A	0	3	0	3	3	0	1
B	1	5	3	8	7	2	1.4
C	3	2	8	10	7	5	3.5
D	9	5	10	15	6	1	1.2
E	12	5	15	20	8	3	1.6
Promedio		4			6.2	2.2	1.74

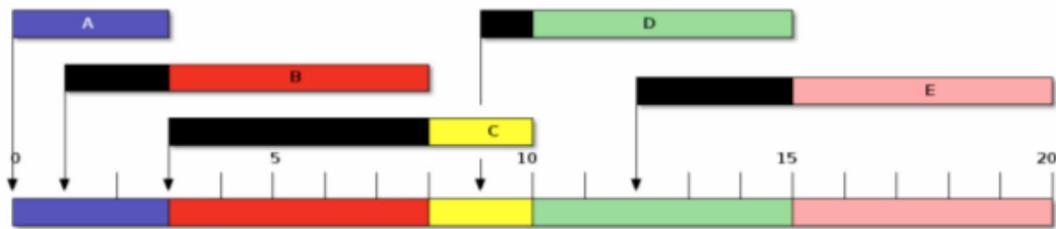


Figura 4: Primero llegado, primero servido (FCFS)

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. Si un proceso limitado por el procesador llega primero y 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, ya que hasta que no termine de ejecutarse el proceso actual no se servirá el resto. 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**. Sin embargo, a menudo se combina con otras políticas para proporcionar una planificación eficaz.

4.4 Turno rotatorio (round robin)

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. Si antes de finalizar su rodaja de tiempo un proceso invoca a una operación de E/S que requiere sacarlo de la CPU o invoca algún tipo de rutina que lo Bloquee, el proceso no cumpliría la rodaja entera.

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 (*dispatcher*). De esta forma, se deben evitar los *quantums* de tiempo muy pequeños.

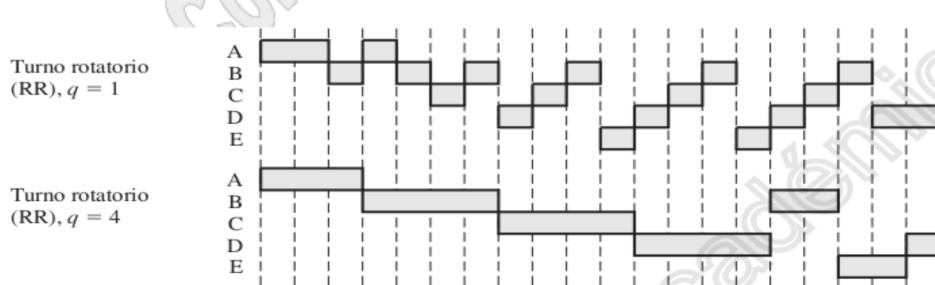
Una buena idea es que el *quantum* de tiempo debe ser ligeramente mayor que el tiempo requerido hasta que se producen los primeros resultados de un proceso, por ejemplo, la carga de las primera páginas de un fichero pdf. Si es menor, muchos más procesos necesitarán, al menos, dos *quantums* de tiempo para ejecutarse, aunque esto es algo muy relativo (depende de la longitud de proceso y de posibles eventos de espera).

Nótese que en el caso extremo de un *quantum* de tiempo mayor que el proceso más largo en el sistema, la planificación en turno rotatorio degenera en FCFS, pudiendo provocar inanición, favoreciendo a los procesos largos y perjudicando a los cortos.

La siguiente Figura (puede empezar a numerar por instante cero) muestra los resultados de un ejemplo de procesos limitados por el procesador, utilizando *quantums* de tiempo $q = 1$ y $q = 4$. Nótese que el proceso E, que es el trabajo más corto, obtiene mejoras significativas para un *quantum* de tiempo de $q = 1$.

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



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 que los procesos limitados por el procesador, ya que antes de que se acabe su *quantum*, posiblemente invocarán una operación de E/S que los lleve al estado Bloqueado, quedando a la espera de que termine la operación de E/S que

hayan invocado. Por el contrario, un proceso limitado por el procesador generalmente utiliza la rodaja de tiempo completa mientras ejecuta e inmediatamente vuelve a la cola de Listos.

En la Figura 5 se muestra otro ejemplo del algoritmo de planificación de turno rotatorio $q=1$:

Proceso	Tiempo de Llegada	t	Inicio	Fin	T	E	P
A	0	3	0	6	6	3	2.0
B	1	5	1	11	10	5	2.0
C	3	2	4	8	5	3	2.5
D	9	5	9	18	9	4	1.8
E	12	5	12	20	8	3	1.6
Promedio		4			7.6	3.6	1.98

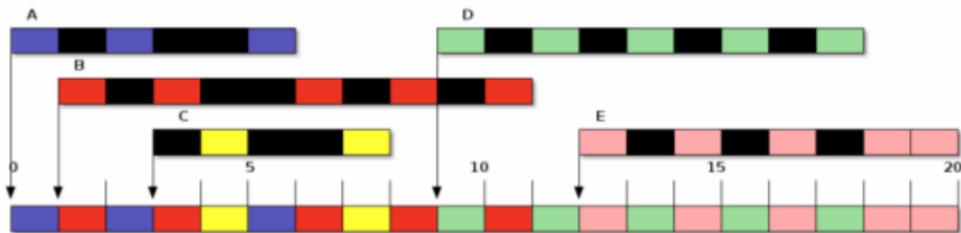


Figura 5: Ronda (Round Robin)

4.4.1 Turno rotatorio virtual

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, es decir, termina su rodaja de tiempo (decisión expulsiva), vuelve a la cola de Listos. Cuando se bloquea un proceso por E/S, se une a la cola de Bloqueados por ese tipo de E/S. Hasta aquí, todo como siempre. La nueva característica es una cola auxiliar FCFS a la que se mueven los procesos Bloqueados en una E/S.

Cuando se va a tomar un proceso por parte del planificador a corto plazo, 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 lo que resta de su rodaja de tiempo.

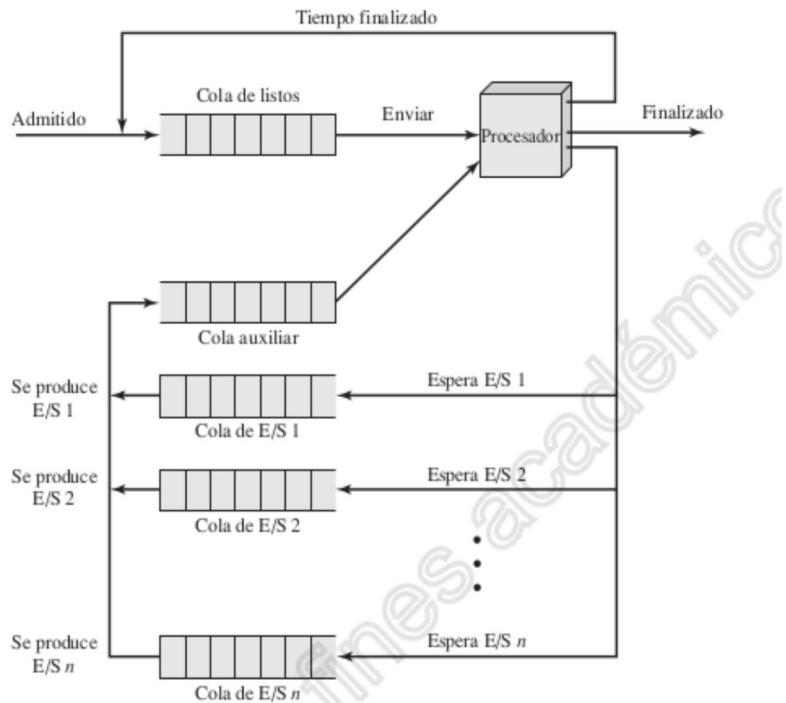


Figura 9.7. Diagrama de encolamiento para el planificador en turno rotatorio virtual.

4.5 Planificación mediante colas de prioridades multinivel

En muchos sistemas, a cada proceso se le asigna una prioridad y se le asigna en una cola multinivel, de forma que el planificador siempre elegirá un proceso de prioridad mayor sobre un proceso de prioridad menor.

La prioridad la establece el sistema inicialmente en función de determinados parámetros como el propietario del proceso, el tipo del proceso, su tamaño, los recursos que requiere, etc.

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. En lugar de una sola cola de procesos listos para ejecutar, se proporcionan un conjunto de colas en orden descendente de prioridad: CL_0, CL_1, \dots, CL_n , con la $prioridad[CL_i] > prioridad[CL_j]$, para $i < j$ ¹.

Cuando se va a realizar una selección en la planificación, el planificador comenzará en la cola de listos con la prioridad más alta, CL_0 . Si hay uno o más procesos en la cola, se selecciona un proceso utilizando alguna política de planificación, por ejemplo una política FIFO. Si CL_0 está vacía, entonces se examina CL_1 , y así sucesivamente.

Un proceso se expulsa de la CPU si se cumple su rodaja de tiempo asignada o si invoca a una operación que lo lleve a estado Bloqueado. Cuando se produce su expulsión sale de la cola CL_i , de forma que cuando vuelve al estado de Listo, se sitúa en CL_i nuevamente. Con esta política, habrá

1 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

tantas colas en el sistema como prioridades, a no ser que se agrupen rangos de prioridades en colas.

Si la prioridad de un proceso no cambia durante su ejecución y ésta se usa para planificar a corto plazo, se obtiene una política de **prioridades estáticas**.

Un problema de las prioridades estáticas es que algunos parámetros, fundamentalmente los referidos al consumo de recursos, y en particular al tiempo de CPU y duración de sus intervalos, no se conocen *a priori*, por lo que la prioridad puede no ser indicativa de comportamiento del programa. Esto es así porque la prioridad se asigna al crearse el proceso, según del tipo que sea.

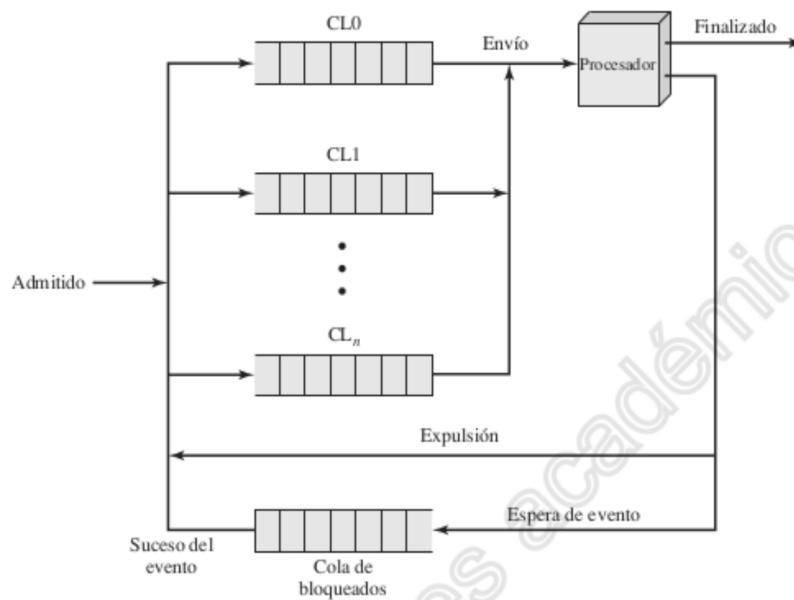


Figura 9.4. Encolamiento con prioridades.

Otro problema de las colas multinivel con prioridades estáticas es que un proceso con baja prioridad puede llegar a inanición si hay continuamente procesos con mayor prioridad listos para ejecutarse.

4.5.1 Colas de prioridades multinivel retroalimentada (feedback)

Una variación de las colas con prioridades multinivel es la **retroalimentación multinivel o feedback**. En esta política de planificación hay fijado un número máximo de colas con prioridad. Cuando un proceso llega al sistema se sitúa en la primera cola, CL0.

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, por ejemplo una política FIFO. Si CL0 está vacía, entonces se examina CL1, y así sucesivamente.

Un proceso se expulsa de la CPU si se cumple su rodaja de tiempo asignada o si invoca a una operación que lo lleve a estado Bloqueado. A diferencia de las colas con prioridades básicas, si un proceso se expulsa de una cola CL_i , automáticamente pasará a la siguiente cola con menor prioridad

(en sistema POSIX se indica con un número mayor). Cuando un proceso cambia su nivel de prioridad a lo largo de su estancia en el sistema se dice que se tienen **prioridades dinámicas**.

Cuando un proceso llega a la cola máxima o de menor prioridad no puede descender más, por lo que es devuelto a esta cola repetidas veces hasta que se consigue completar. De esta forma, la cola con menor prioridad se trata con una política de turno rotatorio o *Round Robin*.

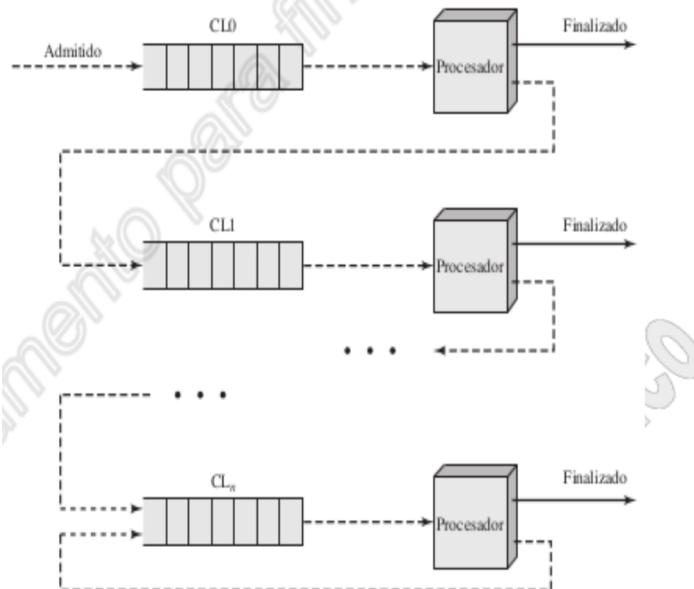


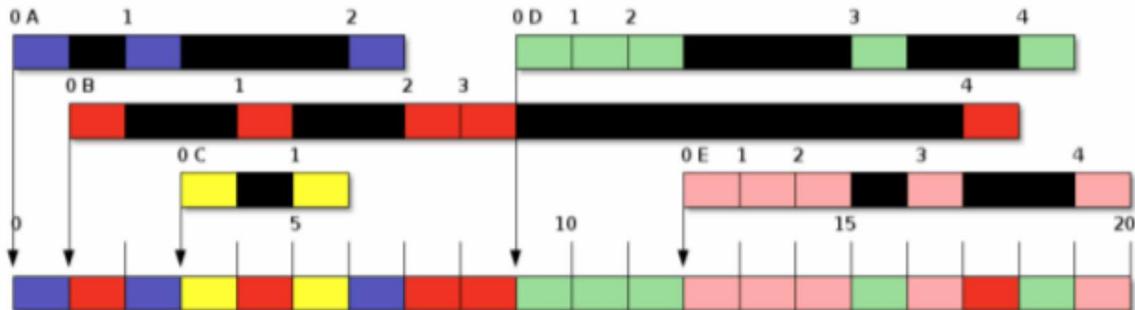
Figura 9.10. Planificación retroalimentada.

Una versión sencilla de la política de colas multinivel con retroalimentación consiste en realizar la expulsión con una rodaja o *quantum* $q=1$. Un problema que presenta el esquema de colas multinivel con $q=1$ 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, ya que el planificador siempre empieza la asignación de procesos de CPU por la cola CL0.

Para compensar el problema del quantum $q = 1$, se podrían variar los tiempos de expulsión de cada cola, de manera que 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 *CLi* se le permite ejecutar 2^i ($q=2^i$) unidades de tiempo antes de ser expulsado.

También se puede usar la política *feedback* sin tener una última cola *Round Robin*, de forma que se tengan tantas colas como se necesiten, al igual que con las colas de prioridades básicas. La siguiente Figura muestra el resultado de un ejemplo con procesos limitados por el procesador de la política de colas multinivel con retroalimentación con *quantum* fijo $q = 1$ y sin cola final *Round Robin*.

Proceso	Tiempo de Llegada	<i>t</i>	Inicio	Fin	<i>T</i>	<i>E</i>	<i>P</i>
A	0	3	0	7	7	3	2.3
B	1	5	1	18	17	12	3.4
C	3	2	3	6	3	1	1.5
D	9	5	9	19	10	5	2.0
E	12	5	12	20	8	3	1.6
Promedio		4		9	5	2.16	



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

Otro enfoque para reducir el sesgo a favor de los procesos largos en FCFS es la política **primero el proceso más corto (SPN)**, también conocida como **Shortest job next (SJN)** y **Shortest Job First (SJF)**.

SPN es una política no expulsiva en la que se selecciona el proceso con el tiempo de procesamiento más corto esperado, es decir, el proceso que necesite menos CPU o tiempo de servicio para ejecutarse por completo y salir del sistema.

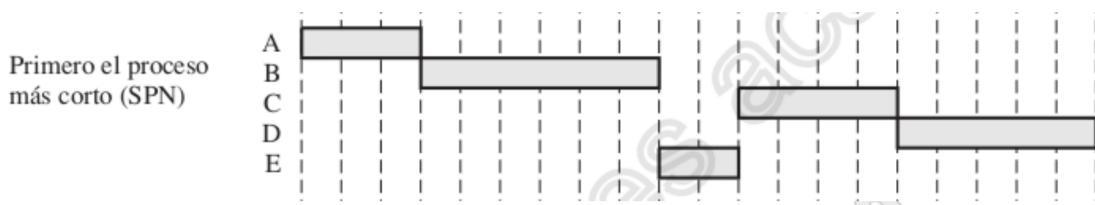
De esta forma un proceso corto se situará a la cabeza de la cola de Listos, por delante de los procesos más largos. Una vez dentro de la CPU un proceso se ejecuta entero si no se Bloquea, es decir, aunque lleguen nuevos procesos a la cola de Listos, los tiempos de cada proceso no se recalculan. Esto se hace cuando el proceso actual termine, para determinar cuál es el proceso más corto de los que hay en ese momento en la lista de Listos.

En caso de que un proceso se Bloquee, pasaría a la lista de Bloqueados en espera de un evento, al igual que ocurre con la política FCFS. Cuando ocurra el evento pasará a la lista de Listos con un tiempo de servicio igual al restante que le quede por cumplir.

La siguiente Figura muestra el resultado de un ejemplo con procesos limitados por el procesador. Observe que el proceso E recibe servicio mucho antes que con FCFS. El tiempo de estancia normalizado también se mejoran significativamente.

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



Con esta política es necesario contar con información por anticipado acerca del tiempo de CPU que requieren los procesos que forman la lista de Listos. Ahora bien, es muy difícil contar con esta información antes de ejecutar el proceso, con lo que normalmente se estima mediante:

- Modelos matemáticos que asignen un tiempo de servicio a un proceso dependiendo de su tipo, carga del sistema, recursos que necesite, etc.
- Mediante la caracterización de las necesidades del proceso, es decir, ver si durante su historia de ejecución de un proceso, este ha sido tendiente a manejar ráfagas limitadas por entrada-salida o limitadas por procesador, y cuál es su tendencia actual.

Al trabajar con estimaciones en lugar de duraciones reales, el éxito de la planificación dependerá de lo adecuado que sea el modelo matemático de estimación.

De forma general, el riesgo con SPN es la posibilidad de inanición para los procesos más largos si hay una llegada constante de procesos más cortos, aunque hay casos, como el ejemplo descrito de los procesos W, X, Y y Z, en los que se penaliza fuertemente al proceso corto. Concretamente se penaliza a Y debido a la carencia de expulsión, ya que cuando Y llega en el instante 2, el proceso X que es largo ya está ejecutando.

En la Figura 8 se dispone otro ejemplo más de este esquema de planificación:

Proceso	Tiempo de Llegada	t	Inicio	Fin	T	E	P
A		0	3	3	3	0	1.0
B		1	5	10	9	4	1.8
C		3	2	5	2	0	1.0
D		9	5	15	6	1	1.2
E		12	5	20	8	3	1.6
Promedio		4		5.6	1.6	1.32	

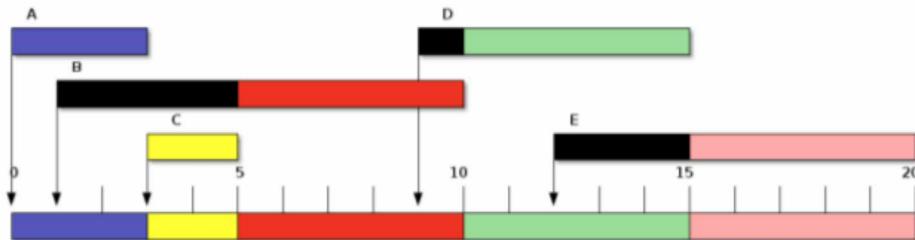


Figura 8: El proceso más corto a continuación (SPN)

4.7 Menor tiempo restante (shortest remaining time – SRT)

SPN puede también implementarse con una decisión expropiativa. Dicha política de planificación se conoce como menor tiempo restante (SRT), y expulsa procesos sin tener en cuenta su rodaja de tiempo.

De esta forma, si un proceso se une a la lista de procesos Listos con menor CPU restante o estimada, se le quita la CPU al proceso actual para asignarla al nuevo proceso más corto. Así, al igual que con SPN, el planificador debe tener una estimación del tiempo de servicio o CPU de los procesos que pasan a la lista de Listos por primera vez.

Si los procesos que van llegando a la lista de Listos no tienen tiempos restantes menores que el proceso que actualmente está en ejecución éste último no se expulsa, ya que no hay asignación de *quantum* o rodaja de tiempo.

La diferencia entonces respecto a SPN es que, mientras que en SPN una vez que termina el proceso que actualmente está ejecutando, es cuando se comprueba la lista de Listos para encontrar el proceso más corto, en SRT la lista de Listos se comprueba cada vez que llegan procesos a ésta.

En el caso de que dos o más procesos tengan el mismo tiempo de servicio restante se aplica una política FIFO, es decir, se le da prioridad al proceso que haya llegado antes a la lista de Listos.

En SRT existe riesgo de inanición para los procesos más largos. A diferencia del turno rotatorio, no se generan interrupciones adicionales por finalización de rodaja de tiempo, reduciéndose la sobrecarga en ese aspecto si no llegan procesos a la lista de Listos con menor tiempo de servicio. Por otra parte, se deben almacenar los tiempos de servicio transcurridos,

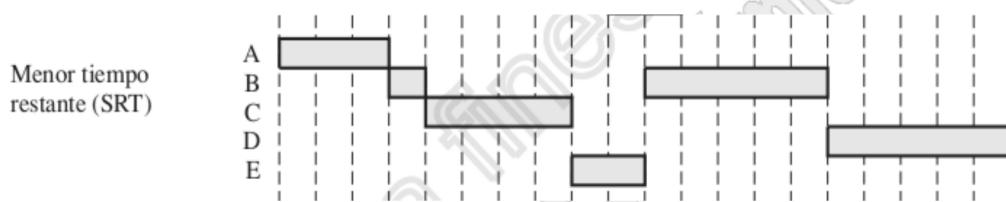
generando sobrecarga, además hay que interrumpir el proceso actual en ejecución para calcular el tiempo restante de los nuevos procesos que llegan a la cola de Listos.

Las políticas SPN y SRT no se pueden implantar en sistemas en los que no se conoce *a priori* el tiempo de CPU que requieren los procesos situados en la lista de Listos.

La siguiente Figura muestra el resultado de un ejemplo de procesos limitados por el procesador. Fíjese en el final de la gráfica, tanto B como D tienen el mismo tiempo restante, pero como B entró en la lista de Listos antes que D, se le da la prioridad a B:

Tabla 9.4. Ejemplo de planificación de procesos.

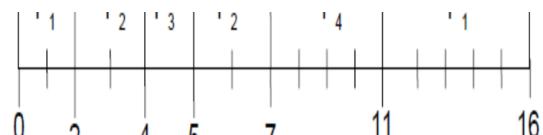
Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



En la siguiente Figura se muestra otro ejemplo más de esta política de planificación:

Shortest Job First (SJF) – Expropiativo

Proceso	Tiempo de arribo	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1



4.8 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. En la política FSS, conocida como planificación de contribución justa, se toman decisiones de planificación basándose en conjuntos de procesos.

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.

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, además de utilizar *quantum* de tiempo. El sistema divide a los procesos en un conjunto de grupos y destina una fracción del procesador a cada grupo.

La planificación, en una única cola de Listos, se realiza en base a: 1) una prioridad asignada a cada proceso, 2) al uso reciente de procesador por parte del proceso y 3) el uso reciente de procesador del grupo al que pertenece el proceso. En el esquema mostrado en esta sección, supondremos que cuanto mayor sea el valor numérico de la prioridad, menor será la prioridad (sistemas POSIX).

Dicho esto, en FSS, la siguiente fórmula se aplica al proceso j de un grupo de procesos k :

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

donde:

- $CPU_j(i)$ = Medida de utilización del procesador por el proceso j en el intervalo i . Depende de esa misma medida justo en un intervalo anterior. La primera vez que un proceso llega al sistema se le asigna un valor de CPU de cero, es decir, $CPU_j(0) = 0$.
- $GCPU_k(i)$ = Medida de utilización del procesador del grupo k en el intervalo i . Al igual que antes depende de esa misma medida en un intervalo anterior. La primera vez que un proceso llega al sistema se le asigna como valor de GCPU el que tengan los procesos del grupo al que pertenece. Se asigna 0 si el grupo lo forma ese único proceso.
- $P_j(i)$ = Prioridad del proceso j al comienzo del intervalo I .
- $Base_j$ = Prioridad base del proceso j . Se asigna por el sistema en el momento en que se crea un proceso según su tipo. La primera vez $P_j(0) = Base_j$, es decir, no se tienen en cuenta los

dos últimos sumandos de la expresión de prioridad.

- W_k = Prima asignada al grupo k . Ese valor hace que se dé más o menos importancia a un grupo de procesos con respecto a los demás. Se debe cumplir que

$$0 < W_k \leq 1 \quad y \quad \sum_k W_k = 1$$

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 W_k de ese grupo. Mientras mayor sea el peso asignado al grupo, su utilización de CPU afectará menos a su prioridad (observe el último sumando de la expresión).

La Figura 9.16 es un ejemplo en el que el proceso A está en un grupo y los procesos B y C están en un segundo grupo, donde se asume que:

- Cada grupo tiene una prima de $W_k = 0,5$.
- Todos los procesos están limitados por el procesador y están normalmente listos para ejecutar (no van a realizar operaciones de entrada salida ni nada que los bloquee).
- Todos los procesos tienen una prioridad base de 60.
- Se asigna un *quantum* de tiempo de 1 segundo.
- La utilización de procesador por parte de un proceso se mide así: Se actualizan unas variables contador para la ejecución actual 60 veces por segundo, de forma que en cada actualización se incrementa el campo del uso del procesador del proceso actualmente en ejecución, así como el campo del grupo correspondiente.
- Las prioridades se recalcularán finalizado el *quantum* de tiempo. En las divisiones que se realizan para calcular las prioridades se redondea para obtener prioridades enteras.
 - Observe por ejemplo la prioridad 90 del proceso A en el instante 1, calculada de esta manera:

$$CPUa(1) = CPUa(0)/2 = 60/2 = 30$$

$$GCPUs(1) = GCPUs(0)/2 = 60/2 = 30$$

$$Pa(1) = \text{base} + CPUa(1)/2 + GCPUs(1)/(4*0.5) = 60 + (30/2) + (30/(4*0.5)) = 90$$

- Observe la prioridad 74 del proceso A en el instante 2, que se calcula de esta manera:

$$CPUa(2) = CPUa(1)/2 = 30/15 = 15$$

$$GCPUs(2) = GCPUs(1)/2 = 30/2 = 15$$

$$Pa(2) = \text{base} + CPUa(2)/2 + GCPUs(2)/(4*0.5) = 60 + (15/2) + (15/(4*0.5)) = 74,$$

cuando debería dar 75, ya que $15/2 = 7.5$ pero se redondea al término inferior, es decir, a 7.

En la Figura 9.16, se planifica primero al proceso A, y 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. Fíjese 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.

Proceso A			Proceso B			Proceso C		
Tiempo	Contador	Contador	Contador	Contador	Contador	Contador	Contador	Contador
	CPU proceso	CPU grupo	proceso	CPU grupo	proceso	CPU grupo	CPU grupo	CPU grupo
0	60	0	0	60	0	0	60	0
1	90	30	30	60	0	0	60	0
2	74	15	15	90	30	30	75	0
3	96	37	37	74	15	15	67	0
4	78	18	18	81	7	37	93	30
5	98	39	39	70	3	18	76	15
	Grupo 1			Grupo 2				

El rectángulo coloreado representa el proceso en ejecución

5 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 qué tipo de procesos va a haber en el sistema y para qué 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.