

## ★ Algoritmos de planificación:

### *Planificación de Plazo Fijo:*

En la planificación de plazo fijo se programan ciertos trabajos para terminarse en un tiempo específico o plazo fijo. Estas tareas pueden tener un gran valor si se entregan a tiempo, y carecer de él si se entregan después del plazo. Esta planificación es compleja por varios motivos:

- El usuario debe informar por adelantado de las necesidades precisas de recursos del proceso.
- El sistema debe ejecutar el proceso en un plazo fijo sin degradar demasiado el servicio a los otros usuarios.
- El sistema debe planificar cuidadosamente sus necesidades de recursos dentro del plazo. Esto puede ser difícil por la llegada de nuevos procesos que impongan demandas imprevistas al sistema.
- Si hay muchas tareas a plazo fijo activas al mismo tiempo, la planificación puede ser tan compleja que se necesiten métodos de optimización avanzados para cumplir los plazos.
- La administración intensiva de recursos requerida por la planificación de plazo fijo puede producir un gasto extra substancial.

### *Planificación Primero en Entrar-Primero en Salir (FIFO):*

Cuando se tiene que elegir a qué proceso asignar la CPU se escoge al que llevara más tiempo listo. El proceso se mantiene en la CPU hasta que se bloquea voluntariamente. La ventaja de este algoritmo es su fácil implementación, sin embargo, no es válido para entornos interactivos ya que un proceso de mucho cálculo de CPU hace aumentar el tiempo de espera de los demás procesos. Para implementar el algoritmo (ver Figura 1) sólo se necesita mantener una lista con los procesos listos ordenada por tiempo de llegada. Cuando un proceso pasa de bloqueado a listo se sitúa el último de la cola. En a) el proceso P7 ocupa la CPU, los procesos P2, P4 y P8 se mantienen en la lista de preparados. En b) P7 se bloquea (ya sea al realizar una E/S, una operación WAIT sobre un semáforo a cero u otra causa) y P2 pasa a ocupar la CPU. En c) ocurre un evento (finalización de la operación de E/S, operación SIGNAL, ...) que desbloquea a P7, esto lo vuelve al estado listo, pasando al final de la lista de procesos listos.



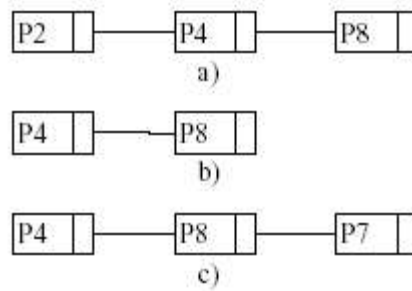


Figura 1. Lista de procesos preparados en FIFO.

Algunas de las características de este algoritmo es que es no apropiativo y justo en el sentido formal, aunque injusto en el sentido de que: los trabajos largos hacen esperar a los cortos y los trabajos sin importancia hacen esperar a los importantes. Por otro lado es predecible, pero no garantiza buenos tiempos de respuesta, por ello se emplea como esquema secundario.

#### *Planificación por Turno Rotatorio (Round Robin):*

Este es uno de los algoritmos más antiguos, sencillos y equitativos en el reparto de la CPU entre los procesos, muy válido para entornos de tiempo compartido. Cada proceso tiene asignado un intervalo de tiempo de ejecución, llamado **quantum o cuanto**. Si el proceso agota su **quantum** de tiempo, se elige a otro proceso para ocupar la CPU. Si el proceso se bloquea o termina antes de agotar su **quantum** también se alterna el uso de la CPU. El **round robin** es muy fácil de implementar. Todo lo que necesita el planificador es mantener una lista de los procesos listos, como se muestra en la Figura 2. En esta figura en a) el proceso P7 ocupa la CPU. En b) P7 se bloquea pasando P2 a ocupar la CPU. En c) P2 agota su **quantum** con lo que pasa al final de la lista y P4 ocupa la CPU.



Este algoritmo presupone la existencia de un reloj en el sistema para generar periódicamente interrupciones al expirar el *quantum* y proceder a llamar al *dispatcher*.

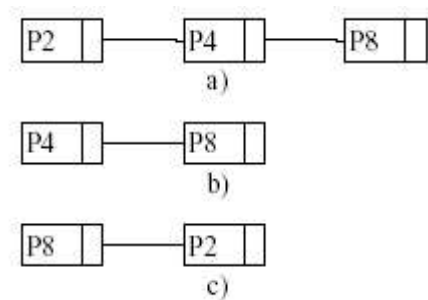


Figura 2. Lista de procesos preparados en Round-Robin.

### Tamaño del Cuanto

La determinación del tamaño del cuanto es vital para la operación efectiva de un sistema de cómputo. Si el cuanto de tiempo es muy grande, cada proceso tendrá el tiempo necesario para terminar, de manera que el esquema de planificación por turno degenera en uno de *primero-en-entrar-primero-en-salir*. Si el cuanto es muy pequeño, el gasto extra por cambio de proceso se convierte en el factor dominante y el rendimiento del sistema se degradará hasta el punto en que la mayor parte del tiempo se invierte en la conmutación del procesador, con muy poco o ningún tiempo para ejecutar los programas de los usuarios.

El tamaño del cuanto debe fijarse en el tamaño lo bastante grande como para que la mayoría de las peticiones interactivas requieran menos tiempo que la duración del cuanto.

Por ejemplo, supongamos que el cambio de proceso tarda 5 mseg., y la duración del *quantum* es de 20 mseg.. Con estos parámetros, se utiliza un mínimo del 20% del tiempo de la CPU en la ejecución del sistema operativo. Para incrementar la utilización de la CPU por parte de los procesos de usuario podríamos establecer un *quantum* de 500 mseg., el tiempo desperdiciado con este parámetro sería del 1%.

Pero consideremos lo que ocurriría si diez usuarios interactivos oprimieran la tecla *enter* casi al mismo tiempo. Diez procesos se colocarían en la lista de procesos listos. Si la CPU está inactiva, el primero de los procesos comenzaría de inmediato, el segundo comenzaría medio segundo después, etc. Partiendo de la hipótesis de que todos los procesos agoten su *quantum*, el último proceso deberá de esperar 4'5 seg. para poder ejecutarse. Esperar 4'5 seg. para la ejecución de una orden sencilla como *dir* parece excesivo.



En conclusión, un *quantum* corto disminuye el rendimiento de la CPU, mientras que un *quantum* muy largo empobrece los tiempos de respuesta y degenera en el algoritmo FIFO. La solución es adoptar un término medio como 100 mseg.

#### *Planificación por Prioridad al más corto (SJF):*

Al igual que en el algoritmo FIFO las ráfagas se ejecutan sin interrupción, por tanto, sólo es útil para entornos *batch*. Su característica es que cuando se activa el planificador, éste elige la ráfaga de menor duración. Es decir, introduce una noción de prioridad entre ráfagas. Hay que recordar que en los entornos *batch* se pueden hacer estimaciones del tiempo de ejecución de los procesos. La ventaja que presenta este algoritmo sobre el algoritmo FIFO es que minimiza el tiempo de finalización promedio.

Se puede demostrar que este algoritmo es el óptimo. Para ello, consideremos el caso de cuatro ráfagas, con tiempos de ejecución de  $a$ ,  $b$ ,  $c$  y  $d$ . La primera ráfaga termina en el tiempo  $a$ , la segunda termina en el tiempo  $a+b$ , etc. El tiempo promedio de finalización es  $(4a+3b+2c+d)/4$ . Es evidente que  $a$  contribuye más al promedio que los demás tiempos, por lo que debe ser la ráfaga más corta,  $b$  la siguiente, y así sucesivamente. El mismo razonamiento se aplica a un número arbitrario de ráfagas. No obstante, este algoritmo sólo es óptimo cuando se tienen simultáneamente todas las ráfagas.

#### *Planificación por Prioridad al Tiempo Restante más Corto (SRTF):*

Es similar al anterior, con la diferencia de que si un nuevo proceso pasa a listo se activa el dispatcher para ver si es más corto que lo que queda por ejecutar del proceso en ejecución. Si es así, el proceso en ejecución pasa a listo y su tiempo de estimación se decrementa con el tiempo que ha estado ejecutándose.

En SRTF se penaliza a las ráfagas largas (como en SJF). Un punto débil de este algoritmo se evidencia cuando una ráfaga muy corta suspende a otra un poco más larga, siendo más larga la ejecución en este orden al ser preciso un cambio adicional de proceso y la ejecución del código del planificador.



*Planificación a la Tasa de Respuesta más Alta:*

Esta planificación corrige algunas deficiencias de SJF, particularmente, el retraso excesivo de trabajos largos, y el favoritismo excesivo por los trabajos cortos. HRN es una disciplina de planificación no apropiativa en la cual la prioridad de cada proceso no sólo se calcula en función del tiempo de servicio, sino también del tiempo que ha esperado para ser atendido. Cuando un trabajo obtiene el procesador, se ejecuta hasta terminar. Las prioridades dinámicas en HRN se calculan de acuerdo con la siguiente expresión:

$$\text{Prioridad} = \frac{\text{tiempo de espera} + \text{tiempo de servicio}}{\text{tiempo de servicio}}$$

Como el tiempo de servicio aparece en el denominador, los procesos cortos tendrán preferencia. Pero como el tiempo de espera aparece en el numerador, los procesos largos que han esperado también tendrán un trato favorable.

*Planificación por el Comportamiento:*

Con este tipo de planificación se pretende garantizar a los procesos de usuario cierta prestación del sistema y tratar de cumplirla. Si en un sistema tenemos  $n$  procesos de usuario lo normal será garantizar a cada uno de ellos al menos  $1/n$  de la potencia del procesador. Para ello necesitamos del tiempo consumido por el procesador y el tiempo que lleva el proceso en el sistema. La cantidad de procesador que tiene derecho a consumir el proceso será el cociente entre el tiempo que lleva en el sistema entre el número de procesos que hay en el sistema. A esa cantidad se le puede asociar una prioridad que vendrá dada como el cociente entre tiempo de procesador que ha consumido y el tiempo que se le prometió (el tiempo que tiene derecho a consumir). De tal modo que si esa proporción es de 0.5 significa que tan sólo ha consumido la mitad del tiempo prometido pero si es de 2 quiere decir que ha consumido más de lo debido, justamente el doble. En sistemas de tiempo real se puede adoptar una variante de este algoritmo en el que se otorgue mayor prioridad al proceso cuyo riesgo de no cumplir el plazo sea mayor.



*Colas Retroalimentación de Niveles Múltiples:*

Cuando un proceso obtiene la CPU, sobre todo cuando todavía no ha tenido oportunidad de establecer un patrón de comportamiento, el planificador no tiene idea de la cantidad de tiempo de CPU que necesitará el proceso. Los procesos limitados por la E/S normalmente usan la CPU sólo un momento antes de generar una solicitud de E/S; los procesos limitados por la CPU pueden usar el procesador durante horas si está disponible de forma no apropiativa.

Un mecanismo de planificación debe:

- Favorecer a los trabajos cortos.
- Favorecer a los trabajos limitados por la E/S para lograr un mejor aprovechamiento de los dispositivos de E/S, puesto que debe favorecer el paralelismo.
- Determinar la naturaleza de un trabajo lo más pronto posible, y planificarlo de acuerdo con su naturaleza.

Las colas de retroalimentación de niveles múltiples (Figura 3) ofrecen una estructura que cumple con estos objetivos. Un proceso nuevo entra en la red de colas al final de la primera cola. Se desplaza en esa cola mediante Round Robin hasta que obtiene la CPU. Si el trabajo termina o cede la CPU para esperar la terminación de una operación de E/S o de algún evento, el trabajo abandona la red de colas. Si el cuanto expira antes de que el proceso ceda voluntariamente la CPU, el proceso se colocará al final de la cola del siguiente nivel. El proceso será atendido otra vez cuando llegue a la cabeza de esa cola si está vacía la primera. Mientras el proceso utilice todo el cuanto proporcionado en cada nivel, continuará desplazándose al final de la siguiente cola inferior. Por lo general, existe una cola en el nivel más bajo en la cual el proceso circula por turno rotatorio hasta que termina.

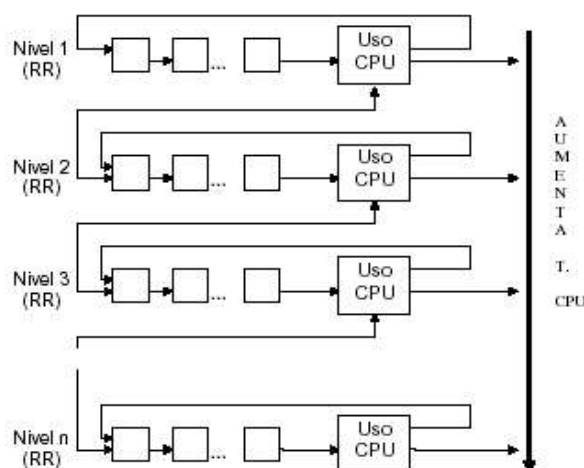


Figura 3. Colas de Retroalimentación de Niveles Múltiples.

En muchos esquemas de retroalimentación de niveles múltiples, el cuanto asignado a un proceso cuando pasa a una cola de nivel inferior alcanza un valor mayor. De esta forma, cuanto más tiempo se encuentre un proceso en la red de colas más grande será el cuanto asignado cada vez que obtenga la CPU, pero tal vez no obtenga la CPU muy a menudo, porque los procesos de las colas de nivel superior tienen mayor prioridad. Un proceso situado en una cola no puede ejecutarse a menos que estén vacías las colas de nivel superior. Un proceso en ejecución será desposeído por un proceso que llegue a una cola superior. Considérese ahora cómo responde un mecanismo de este tipo a diferentes tipos de procesos. El mecanismo debe favorecer a los procesos limitados por la E/S para lograr un buen aprovechamiento de los dispositivos y una respuesta buena para los usuarios interactivos; y de hecho lo hace, porque los procesos limitados por la E/S entrarán en la red con prioridad alta y se les asignará rápidamente la CPU. El tamaño del cuanto de la primera cola se elegirá lo suficientemente grande para que la gran mayoría de los trabajos limitados por la E/S generen una petición de E/S antes de que expire el primer cuanto. Cuando el proceso solicita E/S, abandona la red y ha obtenido un tratamiento favorable, tal como se deseaba.

Ahora considérese una tarea limitada por la CPU que necesita mucho tiempo de procesador. Esa tarea entra en la cola más alta de la red con prioridad alta. Recibe rápidamente su primera asignación de la CPU, pero su cuanto expira y el proceso se coloca en la cola del siguiente nivel inferior. En ese momento, el proceso tiene una prioridad menor que la de los procesos que llegan al sistema, en particular los trabajos limitados por la E/S, que obtienen primero la CPU. El proceso limitado por la CPU acaba recibiendo ésta, obtiene un cuanto mayor que en la cola más alta y vuelve a utilizar la totalidad de su cuanto. Luego es situado al final de la siguiente cola inferior. El proceso sigue desplazándose a colas inferiores, espera más entre divisiones de tiempo y utiliza todo su cuanto cada vez que obtiene la CPU (a menos que sea arrebatada por un proceso entrante). En algún momento, el proceso limitado por la CPU llega a la cola de nivel inferior, en donde entrará en una planificación por turno hasta terminar.

Las colas de retroalimentación de niveles múltiples son ideales para separar procesos en categorías basadas en su necesidad de la CPU. En un sistema de tiempo compartido, cada vez que un proceso abandona la red de colas puede “marcarse” con la identidad de la última cola en donde estuvo, y cuando el proceso entra de nuevo en la red de colas, puede enviarse directamente a la cola en la cual terminó su operación por última vez. En este caso, el planificador está usando un razonamiento heurístico, según el cual el comportamiento anterior del proceso es un buen indicador de su comportamiento en un futuro inmediato. De esta forma, un proceso limitado por la CPU que vuelve a la red de colas no se coloca en las colas de nivel alto donde interferiría con el servicio a los procesos cortos de prioridad alta o con los limitados por la E/S.



Si los procesos se colocan siempre dentro de la red en la cola que ocuparon la última vez, será imposible que el sistema responda a cambios de un proceso, por ejemplo, de estar limitado por la CPU, a estar limitado por la E/S. El problema puede resolverse marcando al proceso también con su duración dentro de la red la última vez que estuvo en ella. Así, cuando el proceso entra de nuevo en la red puede colocarse en la cola correcta. Entonces, si el proceso entra en una fase nueva en la cual deja de estar limitado por la CPU y empieza a estar limitado por la E/S, el proceso experimentará en principio un tratamiento lento mientras el sistema determina que la naturaleza del proceso está cambiando. Pero el mecanismo de planificación responderá con rapidez a este cambio. Otra forma de hacer que el sistema responda a los cambios de comportamiento de los procesos es permitir que un proceso ascienda un nivel en la red de colas cada vez que abandona voluntariamente la CPU antes de que expire su cuanto.

El mecanismo de colas de retroalimentación de niveles múltiples es un buen ejemplo de *mecanismo adaptativo*, que responde a los cambios en el comportamiento del sistema que controla. Los mecanismos adaptativos implican, en general, una carga extra mayor que los no adaptativos, pero la sensibilidad ante los cambios en el sistema da como resultado una mejor capacidad de respuesta, y justifica el aumento en el gasto extra. Una variante común del mecanismo de colas de retroalimentación de niveles múltiples consiste en hacer que un proceso circule por turno varias veces en cada cola antes de pasar a la siguiente cola inferior. El número de ciclos en cada cola crece por lo regular cuando el proceso pasa a la siguiente cola inferior.

