5. Planificación en Tiempo real

Contenido

5.1 Introducción

En un programa concurrente, no es necesario especificar el orden exacto con el que se ejecutan los procesos. Se utilizan primitivas de sincronización para imponer restricciones locales en el orden de ejecución, como pueden ser las primitivas de exclusión mutua, pero en este caso el comportamiento temporal del programa se caracteriza por ser no determinista. Si el programa es correcto entonces la funcionalidad de sus salidas será la misma pase lo que pase en su comportamiento interno o en los detalles de la implementación.

Por ejemplo, cinco procesos independientes se pueden ejecutar de forma no preemptiva en 120 ordenes diferentes (5!). En un sistema multiprocesador o en comportamiento preemptivo existen un número mucho mayor de posibilidades.

Mientras que las salidas del programa serán las mismas en todas las posibles combinaciones de tareas, el comportamiento temporal puede varias considerablemente. Si uno de los cinco procesos tiene un plazo de finalización breve, entonces posiblemente sólo las combinaciones en las que él se ejecute primero serán válidas podrán satisfacer los requerimientos temporales. Un sistema de tiempo real necesita restringir el indeterminismo que aparece en los sistemas de tiempo compartido, seleccionando solo el orden de ejecución de las tareas que cumplan las restricciones temporales. A este proceso se le llama planificación.

5.2 Conceptos

5.2.1 Tipos de tareas

En lo sucesivo consideraremos las tareas como una sucesión de trabajos que se repiten. Cuando la tarea comienza a ejecutarse comenzará a ejecutarse su primer trabajo y, cuando el último trabajo finalice, terminará la ejecución de la tarea.

Normalmente, no todas las tareas tienen la misma importancia a la hora de cumplir los plazos de ejecución. En un mismo sistema, simultáneamente pueden existir tareas críticas y tareas con plazos de ejecución menos críticas, o también tareas sin plazos de ejecución.

Distinguiremos entre tres tipos de tareas, según sus características temporales:

- **Tareas periódicas**. Su ejecución se realiza periódicamente. Sus trabajos entran en ejecución en un periodo constante.

  A lo largo de toda la teoría que vamos a ver, supondremos que todas las tareas con restricciones temporales son periódicas.

- **Tareas esporádicas**. Son tareas que al igual que las tareas periódicas, tienen un plazo de finalización estricto, pero no se ejecutan de forma periódica sino esporádicamente.

  Se tratarán como tareas periódicas suponiendo que existe un tiempo mínimo entre dos activaciones consecutivas. Este tiempo se tomará como el periodo "equivalente" de la tarea.

- **Tareas aperiódicas**. Al contrario que las tareas periódicas, no tienen plazo de finalización, o si lo tienen, no es de obligado cumplimiento.

Se podría hacer otra clasificación de las tareas de tiempo real en base a sus características semánticas:

- **Tareas críticas**. El fallo de una de estas tareas (por no ejecutarse a tiempo) puede ser catastrófico para el sistema.

- **Tareas opcionales**. Si se retrasa su ejecución no afecta a la seguridad del sistema. Las funciones que suelen realizar son:

  - Monitorización del sistema

  - Tareas de mantenimiento

  - Refina el resultado obtenido por tareas críticas

Las tareas opcionales se pueden dividir en dos grupos:

- **Opcionales con plazo** (hard aperiodic). Disponen de un tiempo de ejecución recomendado.

- **Opcionales sin plazo** (soft aperiodic).

Algunas políticas de planificación tienen en cuenta esta distinción e intentan cumplir los plazos de las tareas opcionales con plazo, siempre que no haya sobrecarga por parte de las tareas críticas o por el resto de tareas opcionales con plazo.

5.2.2 Parámetros

En la descripción de un sistema concurrente de tiempo real es común utilizar una serie de parámetros que afectan al comportamiento temporal.

Vamos a seguir suponiendo que un sistema de tiempo real está formado por un conjunto de tareas periódicas, o esporádicas convertidas en periódicas. En esta situación, estos parámetros son:

N: Número de tareas en el sistema

P_i: Periodo de activación

C_i: Tiempo máximo de ejecución

D_i: Plazo máximo de terminación.

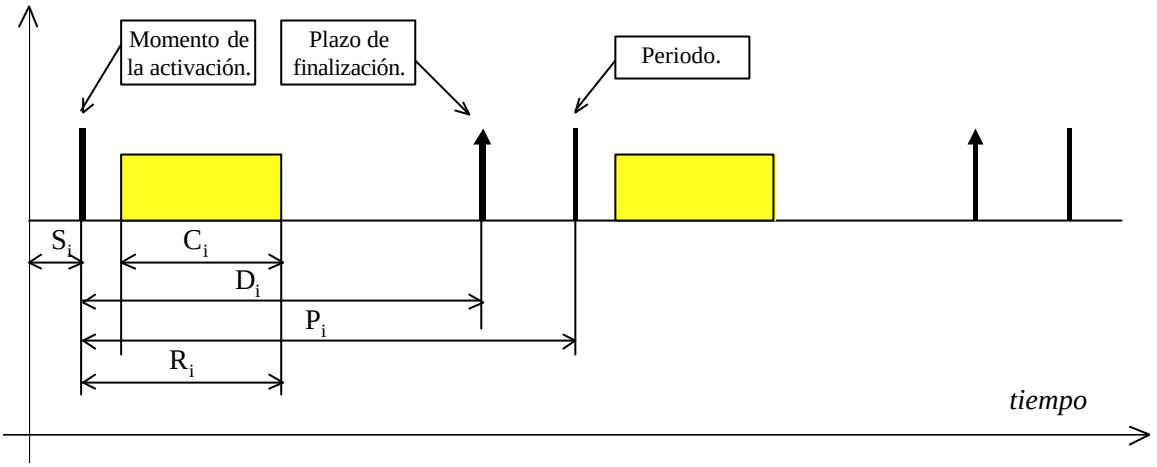
R_i: Tiempo de respuesta máximo. Es el tiempo desde que la tarea se activa hasta que termina realmente a ejecutarse.

Pr_i: Prioridad de la tarea (convenio: mayor valor, menor prioridad)

S_i: Desfase respecto al momento inicial

En las tareas esporádicas cabe señalar que el periodo P_i se entiende como el tiempo mínimo entre dos activaciones consecutivas. La carga del sistema será menos que la que da esta suposición, pero esto nos asegura que si las restricciones temporales se cumplen con este valor para su periodo, también se cumplirá cuando el tiempo entre activaciones sea mayor.

El desfase respecto al momento inicial S_i solo tiene sentido para la primera activación, en el instante inicial del sistema. Veremos más adelante que el momento inicial juega un papel importante a la hora de estudiar la planificabilidad de un sistema



5.2.3 Tiempo real crítico y opcional

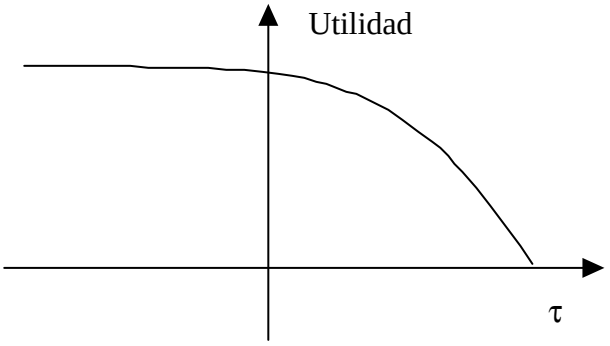
Con estas definiciones podemos profundizar más en los conceptos de tiempo real estricto y tiempo real opcional.

Vamos a definir la tardanza de una tarea como la diferencia entre el tiempo de respuesta y el plazo de ejecución de una tarea.

$$\tau_i = R_i - D_i$$

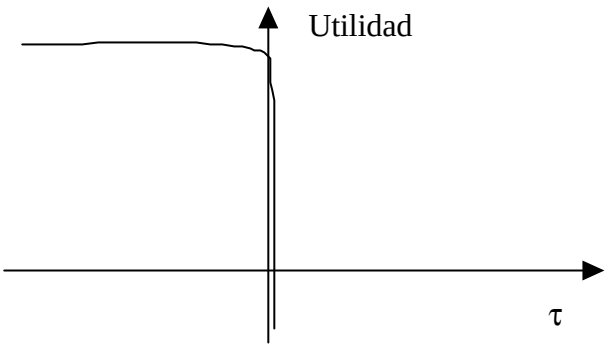
Diremos que una tarea se ejecuta de forma tardía si su tardanza se hace positiva, es decir si finaliza su ejecución después de haber vencido su plazo de ejecución.

En los sistemas de tiempo real opcional un resultado puede seguir siendo útil, aún cuando este se obtenga de forma tardía. Podríamos expresar la utilidad de un resultado en función de su tardanza, obteniendo una gráfica de la forma:



Tiempo real opcional

Esta sería una representación típica en una tarea con plazo de ejecución opcional. Sin embargo, en las tareas de tiempo real crítico la utilidad desciende rápidamente al hacerse positiva la tardanza.



Tiempo real crítico

De esta forma, una tarea de tiempo real crítico será aquella en que la utilidad se hace 0 cuando la tardanza se hace positiva, a diferencia de las tareas de tiempo real opcional en la que se mantiene mayor que cero cuando la tardanza se hace positiva.

Sin embargo, la utilidad es una magnitud difícil de medir en el resultado de una tarea y es difícil de validar.

Otra forma de representar el comportamiento de un sistema es representando la probabilidad de la obtención de una respuesta en función de la tardanza. La representación tomará la forma de una campana de Gauss alrededor del valor medio en que la tarea ofrece la respuesta.

Definida de esta forma, la utilidad se puede expresar en forma de una magnitud medible, simplemente haciendo una estadística en el funcionamiento de nuestro sistema. La utilidad se expresaría en forma de un valor de la probabilidad de que el tiempo de respuesta sea menor o igual a la cota temporal (área de la campana de Gauss a la izquierda de la cota).

Un sistema de tiempo real crítico sería aquel en que la probabilidad de que se cumplan todas las cotas temporales es 1.

Teniendo en cuenta todos estos criterios, en los sistemas de tiempo real las restricciones temporales se pueden expresar en varios términos, por ejemplo:

1. Restricciones deterministas. Por ejemplo, el resultado que proporciona una tarea se debe obtener siempre antes de 50 ms.

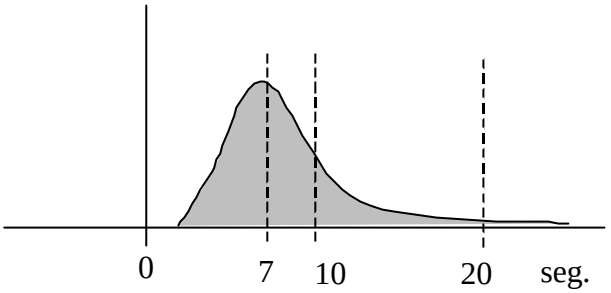
2. Restricciones probabilísticas. Por ejemplo, la probabilidad de que el tiempo de respuesta exceda de 50 ms. debe ser inferior a 0.2.

3. Restricciones es términos de una función de utilidad. Por ejemplo, la utilidad de los resultados que proporciona una tarea debe ser 0.8 o superior.

En la práctica, las restricciones temporales críticas raramente se expresan en los dos últimos términos, mientras que las restricciones temporales opcionales si suelen expresarse en términos de probabilidad o de utilidad.

Por ejemplo, supongamos un sistema telefónico en el que, tras marcar el número con el que deseamos hablar, la conexión se debe establecer en un tiempo que no sea excesivo. Este sistema no requiere un cumplimiento estricto del tiempo máximo de respuesta, sino que puede ser suficiente un comportamiento expresado en términos de probabilidades.

Así, el cliente puede quedar satisfecho si la conexión se establece antes de 10 seg. en el 95% de los casos y antes de 20 seg. en el 99.95% de las llamadas.



Probabilidad del tiempo de respuesta en un sistema telefónico

Otra característica que diferencia a los sistemas de tiempo real crítico es que estos requieren tener una seguridad en que la tardanza nunca será positiva, pues el sistema ya no sería útil. Por tanto, se necesita de una demostración que asegure que esto va a ser siempre así.

Por el contrario, los sistemas de tiempo real opcional no necesitan de una demostración tan estricta pues, por la definición anterior, puede seguir dando resultados válidos aunque se incumplan ocasionalmente los requerimientos temporales.

En lo sucesivo, consideraremos sólo sistemas de tiempos real crítico, o a lo sumo sistemas híbridos en los que conviven tareas con restricciones temporales críticas y opcionales. Por tanto, prestaremos especial atención a la demostración que asegure que las restricciones temporales críticas se van a cumplir bajo cualquier circunstancia.

5.3 Modelo de referencia de los sistemas de tiempo real

Existen multitud de factores que afectan al comportamiento temporal de un sistema. Muchos de ellos pueden ser insignificantes y otros son altamente determinantes. Por ejemplo, la temperatura puede afectar a la frecuencia del reloj de cuarzo del procesador. O el tipo de procesador RISC o CISC también puede afectar a los tiempos de ejecución. Si se tienen en cuenta todos estos factores, el análisis de los sistemas puede resultar extremadamente complicado o hasta imposible de abordar.

Para hacer un estudio general del comportamiento temporal de los sistemas interesa tener en cuenta solamente los factores más determinantes, obviando los menos significativos. De lo contrario va a resultar imposible hacer una análisis claro y sencillo del comportamiento temporal de un sistema.

Interesa pues partir de un modelo teórico lo más sencillo posible, que tenga en cuenta los elementos más determinantes, y hacer el estudio temporal sobre sistemas que contemple solamente estos factores.

Posteriormente se estudiará como afecta a nuestro modelo la eliminación de las restricciones que hemos impuesto y que no están presentes en los sistemas reales.

Seguramente, este camino llevará a mejores resultados que si se aborda inicialmente el problema en toda su complejidad.

5.3.1 Tiempo de ejecución

Un primer paso en la definición del modelo simple es la consideración del tiempo de ejecución de una tarea.

En general, el tiempo de ejecución de una tarea variará entre un valor máximo y un valor mínimo, dependiendo del código que se ejecute según los saltos condicionales que se activen o el número de iteraciones en los bucles.

Otros factores pueden alterar en gran medida el tiempo de ejecución, en general relacionados al uso de recursos que son compartidos, como por ejemplo canales de comunicación o el uso de memoria virtual.

El uso de elementos no deterministas puede hacer que la consideración del tiempo de ejecución en el peor de los casos produzca unos tiempos mucho mayores que el tiempo normal de ejecución, y que la consideración de estos tiempos haga inviable la consecución de los requerimientos temporales.

Existen dos razones que apoyan la consideración de una aproximación determinista. Muchos sistemas de tiempo real requieren un alto nivel de seguridad. En el diseño de estos sistemas es usual utilizar técnicas en las que la variación del tiempo de ejecución sea lo más pequeña posible. Así, por ejemplo, se evita la utilización de estructuras dinámicas que introducen una gran variación en la cantidad de memoria utilizada y en el tiempo para operar con estas estructuras. Trabajando con estas restricciones, el diseñador puede modelar más fácilmente los tiempos de ejecución de forma determinista.

En segundo lugar, es común que sólo una pequeña parte del sistema requiera un comportamiento temporal estricto, mientras que en el resto el cumplimiento de las restricciones temporales es opcional. En este caso, se puede realizar el diseño de tiempo real considerando exclusivamente el subsistema compuesto por las tareas de tiempo real crítico, dejando el tiempo sobrante para el resto de tareas que no son urgentes. De esta forma, se evita el sobredimensionar el sistema obligando al cumplimiento de todos los tiempos de ejecución, cuando en realidad no es necesario.

Por tanto, en nuestro modelo supondremos la utilización de un sistema determinista en el que el tiempo de ejecución en el peor de los casos de todas las tareas es conocido de antemano.

Relacionado también con la aproximación determinista, supondremos que el número de taras de tiempo real crítico en el sistema es constante y conocido. No se

considera de momento la aparición de nuevas tareas de forma esporádica ni el cambio en el funcionamiento del sistema que requiera añadir nuevas tareas o la sustitución de las actuales por otras.

5.3.2 Modelo de tareas periódicas

Dando un paso más para aumentar en determinismo en los sistemas, vamos a suponer que todas las tareas son periódicas, es decir, que sus instantes de activación se repiten en el tiempo en intervalos fijos.

Esta restricción define la carga máxima que cada tarea solicita del sistema en cada una de sus activaciones, que corresponderá a su tiempo máximo de ejecución.

Esta condición es muy importante pues sin ella no sería posible hacer ninguna previsión sobre la respuesta de una tarea, cuando el resto podrían consumir los recursos del sistema sin ninguna restricción.

Esta restricción no obliga a que todas las tareas del sistema sean periódicas con un periodo de activación rigurosamente exacto. Es admisible un cierto grado de indeterminación en el periodo, pero en este caso el valor del periodo que se considerará será el menor de los posibles.

Ampliando este criterio, también se pueden considerar como periódicas las tareas esporádicas, siempre que se pueda determinar un valor mínimo del tiempo entre dos activaciones consecutivas. Este valor mínimo será el que se toma como su periodo.

Veamos una serie de definiciones de términos aplicables en los sistemas donde todas las tareas son periódicas y que utilizaremos en el estudio de las políticas de planificación.

5.3.2.1 Sistema Síncrono

Un sistema será síncrono si todas las tareas piden ejecución al mismo tiempo la primera vez que se activan después del instante inicial.

En nuestro modelo, esto significa que todos los S_i de todas las tareas tienen el mismo valor. Lo normal en este caso es que valga 0.

5.3.2.2 Factor de Utilización

Es la fracción de la CPU que se utiliza en el sistema. En nuestro modelo simplificado se define con la expresión:

$$U = \sum_{i=1}^{N} \frac{Ci}{Pi}$$

que viene a ser la suma de las fracciones de CPU utilizado por el total de las tareas.

Para que un conjunto de tareas periódicas se puedan ejecutar en un procesador es necesario que el factor de utilización sea menor o igual que uno.

5.3.2.3 Trabajo o activación

Un trabajo o activación de una tarea corresponde a cada una de las instancias de las tareas periódicas o esporádicas. En las tareas periódicas se producirá una activación en cada periodo.

5.3.2.4 Hiperperiodo

Es el mínimo común múltiplo de los periodo de todas las tareas.

El valor del hiperperiodo es importante porque a partir de este valor el sistema se repite. Por tanto, si se determina que un sistema es planificable en la duración de un hiperperiodo, lo será también durante toda la vida del sistema

Ocurre que con un número elevado de tareas el valor del hiperperiodo se dispara (en el caso más desfavorable es el producto de todos los periodos). Hay que tener en cuenta que la probabilidad de que dos números elegidos al azar sean primos es del 60%.

5.3.3 Dependencias entre tareas

Es un sistema multitarea puede ocurrir que existan ciertas dependencias temporales en la ejecución de las distintas tareas. Pueden existir diferentes tipos de dependencias. Algunas de estas son:

1. Restricciones de precedencia. Ocurre cuando varias tareas interaccionan de forma cooperativa para realizar una determinada función, de forma que una tarea no puede continuar hasta que otra la active explícitamente.

2. Dependencia en los datos. En este caso existe una relación competitiva debido a que varias tareas utilizan unas datos que no pueden ser compartidos, es decir, que se deben ejecutar bajo exclusión mutua.

3. Dependencia temporal relativa. Varias tareas debe finalizar sus trabajos en un instante que no puede distar más de una cantidad. El plazo ahora no es absoluto sino relativo al de otra tarea. Es, por ejemplo, el caso de una videoconferencia en la que se quiere evitar que la voz acompañe al movimiento de los labios.

4. Restricciones de precedencia AND / OR. Es una relación entre tareas similar a la restricción de precedencia, pero cuando una tarea debe esperar a que finalicen varias predecesoras (AND), o sólo alguna de ellas (OR).

5. Ejecución condicional. Esta relación hace referencia a cuando una tarea se ejecuta o no dependiendo del resultado de otra.

6. Relación de cauce. Es la relación típica del tipo productor / consumidor. El consumidor debe esperar a tener datos para continuar y el productor debe detenerse si el medio de comunicación se colapsa debido a que el productor envía información más rápido de lo que la lee el consumidor.

Abordar todos los tipos de dependencias posibles entre tareas sin duda nos llevaría a un modelo de sistema extremadamente complicado para un primer análisis.

En nuestro modelo de taras sencillo vamos a obviar de entrada todos estos tipos de dependencias, de forma que consideraremos todas las tareas independientes, o al menos el conjunto de tareas con restricciones temporales críticas.

En el caso de que entre dos tareas exista una relación de precedencia, supondremos ambas tareas como periódicas independientes con el mismo periodo igual al de la tarea que precede a la segunda.

Posteriormente veremos como afecta a nuestra teoría la consideración de las dependencias en los datos entre las tareas.

5.3.4 Parámetros funcionales

Aunque la planificación y la sincronización entre tareas se hace sin tener en cuenta ciertas características funcionales, estas afectan a la forma en que se realizan.

Algunas de las características funcionales que se debe tener en cuenta son las siguientes:

5.3.4.1 Trabajos interrumpibles

La ejecución de los trabajos que realizan las tareas a menudo pueden ser interrumpidos para dar paso a ejecutar otras tareas más urgentes. Cuando finalice el trabajo de una tarea más urgente, se reanuda la ejecución de la tarea que ha sido interrumpida.

La interrupción de un trabajo se llama preempción (*preemption* en inglés). Un trabajo será interrumpible si su ejecución se puede suspender en cualquier momento para permitir la ejecución de un trabajo más prioritario.

Un trabajo será no interrumpible si se debe ejecutar sin interrupción desde que comienza hasta que termina.

Dentro de un sistema puede haber trabajos interrumpibles y trabajos que no lo son, y esto habrá que tenerlo en cuenta cuando se planifica la ejecución de los trabajos.

Cada vez que se interrumpa una tarea el sistema debe guardar su estado, para que este pueda ser restituido más adelante. Al intercambio de estados en la CPU producido por el cambio de tarea que está en ejecución se le denomina *'cambio de contexto*'. El tiempo consumido en esta operación se denomina *'tiempo de cambio de contexto*'. Este tiempo introducirá una sobrecarga en los tiempos para realizar los trabajos de las tareas que afectará a sus tiempos de respuesta.

En nuestro modelo de tareas simples consideraremos que todas las tareas son interrumpibles y que el tiempo de cambio de contexto es nulo o despreciable frente a los tiempos de ejecución de las tareas.

5.3.4.2 Importancia de los trabajos

No todos los trabajos son igual de importantes en un sistema de tiempo real. Habrá trabajos que deberán terminar antes que otros y esto lo debe tener en cuenta el planificador para decidir el orden en que se ejecutan las tareas.

La importancia de las tareas en este sentido habrá que definirla de alguna manera y habrá que indicárselo al planificador.

En los sistemas de tiempo real interesa que sean más importantes las tareas cuyo plazo de ejecución se alcance antes para adelantar su ejecución.

Una forma de marcar la importancia de las tareas es asignando una prioridad a las tareas, de forma que el planificador se encargue de ejecutar la tareas más prioritaria de entre todas las que estén activas en cada momento.

5.3.4.3 Ejecución opcional

En un sistema puede aparecer una situación de sobrecarga en la que no sea posible finalizar todas los trabajos a tiempo.

Para no tener que sobredimensionar el equipamiento utilizado, una alternativa es configurar un subconjunto de tareas como opcionales.

En el caso de que un trabajo de una tarea opcional se ejecute fuera de plazo, o no llegue a ejecutarse, producirá un funcionamiento degradado del sistema, pero seguirá ofreciendo un comportamiento satisfactorio.

Por el contrario, los trabajos obligatorios deben ejecutarse completamente dentro de sus plazos.

De esta forma, durante una sobrecarga transitoria en la que no sea posible ejecutar todos los trabajos completamente dentro de sus plazos, se deberán descartar los trabajos opcionales para que los obligatorios puedan finalizar a tiempo.

Se tendrá que utilizar un parámetro funcional que indique si un trabajo es obligatorio u opcional para que se utilice esta información en los casos de sobrecarga.

En nuestro modelo simplificado supondremos que todos los trabajos de cada tarea son obligatorios. Más adelante se verá la forma de introducir los trabajos opcionales.

5.3.5 Planificadores

En un sistema con uno o más procesadores, los trabajos deben ser asignados a los procesadores en un orden determinado para que sean ejecutados. El modulo que implementa el algoritmo que decide que trabajos ejecuta cada procesador y en que orden se llama *planificador*.

Se entiende por *planificación* a una asignación concreta de trabajos a procesadores en unos intervalos de tiempo determinados.

Un planificador debe producir planificaciones válidas, de forma que satisfagan las siguientes condiciones.

1. Cada procesador debe estar asignado como máximo a un solo trabajo en cada instante.

2. Cada trabajo está asignado como máximo a un solo procesador en cada instante.

3. La cantidad total de tiempo de procesador asignada a cada trabajo debe ser igual a su tiempo de ejecución máximo o a su actual tiempo de ejecución.

4. Se satisfacen todas las restricciones de precedencia y uso de recursos para acceder a los datos.

Con estas condiciones se da por supuesto que un trabajo no se ejecuta en paralelo en más de un procesador para aumentar su velocidad. En tal caso se supondrán trabajos distintos los que se ejecuten en procesadores distintos, aunque colaboren para realizar una función en conjunto.

Una planificación válida es una *planificación posible* si cada trabajo finaliza antes de su límite temporal, es decir, si se cumplen todas las restricciones temporales.

De la misma forma, diremos que un conjunto de tareas es *planificable* de acuerdo con un algoritmo de planificación si cuando utiliza este algoritmo el planificador siempre produce planificaciones posibles.

La forma que tenemos de medir el rendimiento de un algoritmo de planificación en las aplicaciones de tiempo real es por su habilidad de encontrar planificaciones posibles en una determinada aplicación cuando estas existen.

Así, diremos que un algoritmo de planificación es *óptimo* si utilizando este algoritmo el planificador siempre produce planificaciones posibles cuando existen en el conjunto de trabajos dados.

Y a la inversa, si un planificador óptimo falla al encontrar una planificación posible, entonces se puede concluir que el conjunto de tareas no puede ser planificada con ningún otro algoritmo de planificación.

Puede ocurrir que un planificador no consiga encontrar una planificación posible para un conjunto de tareas. Tal vez porque esta no exista.

En algunas situaciones puede ser aceptable que para algunos procesos (opcionales) no se cumplan sus restricciones temporales. Entonces habrá trabajos que finalizarán fuera de plazo, o también es posible que algunos trabajos no lleguen a ejecutarse.

En este caso, un planificador puede elegir entre minimizar la tardanza de los trabajos que se ejecutan fuera de plazo, o minimizar el número de tareas que no se ejecutan para que las que lo hacen si cumplan los plazos.

Estos dos criterios son opuestos, es decir, si un planificador decide completar más trabajos se aumentará la tardanza total y, al contrario, si se quiere disminuir la tardanza habrá que dejar de ejecutar un mayor número de trabajos.

Una medida del rendimiento que tiene en cuenta ambos criterios es la razón de invalidez, que contempla el total de los trabajos inválidos, es decir, tanto los que no se ejecutan como los que no terminan en plazo.

Por lo general, un planificador que admita trabajos opcionales debe minimizar la razón de invalidez.

5.3.6 Restricciones del modelo simple (resumen)

Para poder elaborar con facilidad una teoría que nos permita estudiar las distintas políticas de planificación, asumiremos las siguientes suposiciones:

1. Supondremos un sistema formado por un único procesador.

2. El conjunto de tareas es estático. No se destruyen ni se crean nuevas tareas, por tanto, el número de tareas permanecerá constante durante toda la vida del sistema.

3. El tiempo de ejecución máximo de los trabajos de cada tarea es conocido.

4. Todos los trabajos son interrumpibles.

5. Las operaciones del núcleo de multiprogramación son instantáneas. En concreto, se considera nulo el tiempo de cambio de contexto utilizado para dejar de ejecutar una tarea y retomar otra.

6. Todas las tareas son periódicas, o esporádicas transformadas en periódicas utilizando el tiempo mínimo entre dos activaciones consecutivas como su periodo.

7. Los plazos de finalización de todas las tareas son iguales a sus periodos respectivos.

8. Las tareas son independientes unas de otras. No se considera la existencia de secciones críticas ni de recursos compartidos que obligue a la sincronización entre tareas.

Posteriormente se modificará la teoría desarrollada sobre este modelo eliminando algunas de las restricciones, para acercar mas el modelo de tareas a los sistemas reales.

5.4 Políticas de planificación

Entenderemos por política de planificación un conjunto de aspectos que engloban el diseño e implementación de un sistema de tiempo real y que está formado por:

1. **Test de garantía** off-line. Se emplea durante la fase del diseño del sistema y se aplica sobre las tareas críticas periódicas.

2. Un **algoritmo de planificación**. Este determina que tarea se ejecutará en cada momento y en que instante se producirán los cambios de tareas. Cuando hablamos de planificadores en realidad nos estamos refiriendo al algoritmo de planificación que utiliza.

3. Los **protocolos de acceso a recursos compartidos**. Estos influyen en la planificación y el test de garantía.

4. La **forma de servir las tareas aperiódicas**. Determina la forma de realizar trabajo no crítico sin que afecte a las tareas críticas.

El algoritmo es el aspecto más determinante. Muchas veces, cuando se habla de planificador en realidad se hace referencia a la política de planificación que en que se utiliza.

En nuestro modelo simple no se incluyen los puntos 3 y 4, pero si serán necesarios a medida que eliminemos restricciones al modelo e incluyamos los elementos relacionados con estos puntos en las políticas de planificación.

Podemos distinguir dos grupos de políticas de planificación en relación a los planificadores que utilizan: los planificadores cíclicos y los planificadores por prioridades.

En los **planificadores cíclicos** se coloca "a mano" el orden de ejecución de los trabajos durante un hiperperiodo fuera del tiempo de ejecución. Cuando esté en funcionamiento el sistema, el software que ejecuta el planificador se encargará de activar las tareas según el plan que ya ha sido definido previamente.

En los **planificadores por prioridades** se asigna una prioridad a cada tarea en base a su importancia. En tiempo de ejecución se ejecutará siempre la tarea de más prioridad que este activa en cada instante. En este caso es el planificador el que decide en cada instante que tarea debe ejecutarse.

La asignación de prioridades puede ser:

- **Estática**. La prioridad de cada tarea permanece constante durante toda la vida del sistema.

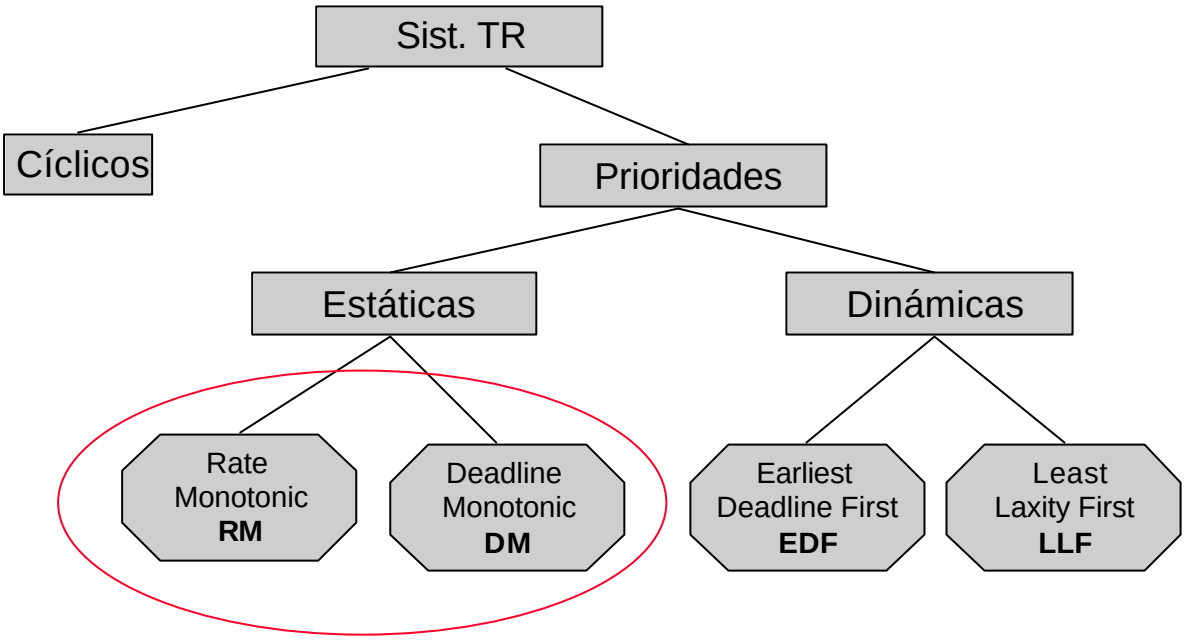
- **Dinámica**. Las prioridades de las tareas varían con el tiempo según unas determinadas reglas.

Entre los planificadores con prioridades estáticas, los más importantes son el Rate Monotonic (RM) y el Deadline Monotonic (DM).

Los planificadores con prioridades dinámicas más importantes son el Earliest Deadline First (EDF) y el Least Laxity First (LLF).

Los más utilizados son los planificadores estáticos pues, aunque los dinámicos pueden ofrecer una mejor planificación, su soporte de ejecución es más complicado y puede suponer un coste de CPU inaceptable.

En el siguiente gráfico se muestra un esquema de las diferentes políticas de planificación:



5.4.1 Planificadores cíclicos

Si todas las tareas de un sistema son cíclicas, se puede confeccionar un plan de ejecución fijo, de manera que las tareas se ejecuten en un orden tal que se cumplan sus límites temporales.

El plan de ejecución se guarda en una tabla que se consulta periódicamente. El planificador es el software encargado de consultar esta tabla y de ejecutar los trabajos que corresponden en cada momento.

5.4.2 Planificación con prioridades

La planificación con prioridades está basada en asignar una prioridad a cada tarea y decidir en cada momento que tarea se debe ejecutar en base a la tarea más prioritaria que esté pendiente de ejecución.

La prioridad de una tarea se expresa normalmente por un número entero. Nosotros utilizaremos el convenio que toma UNIX, en el que la prioridad es un número natural, disminuyendo la prioridad en orden inverso. Es decir, la mayor prioridad corresponderá al valor 0 y la prioridad será menor según aumente su valor.

Cuando hablemos de una tarea de mayor prioridad querremos decir que su valor es menor, y viceversa, a mayor valor menor será la prioridad.

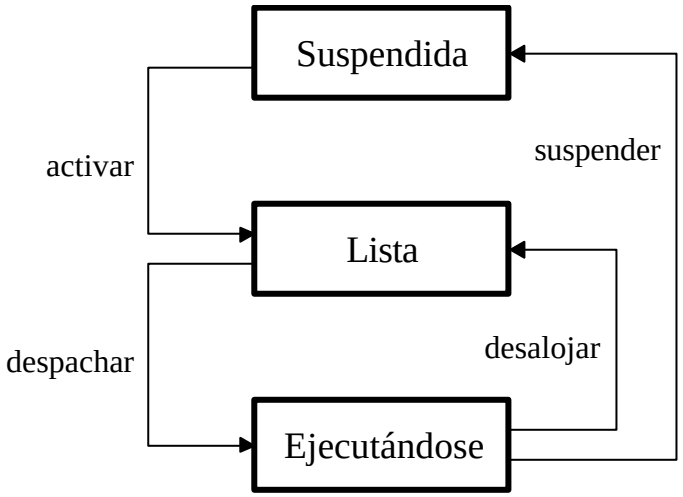
El planificador puede ser expulsivo (preemtive) o no expulsivo. Será expulsivo si una tarea de más prioridad, al activarse, puede quitarle la CPU a otra de menor prioridad. Si el planificador es no expulsivo, cuando una tarea toma la CPU no la deja hasta que termina o la cede voluntariamente.

Ya hemos dicho que la asignación de prioridades puede ser estática o dinámica. Si es estática una tarea tendrá la misma prioridad en cada activación durante toda la vida del sistema. Si la prioridad de la tarea es variable, lo normal es que la prioridad aumente según se va acercando su plazo de finalización.

En la planificación por prioridades, las tareas alternarán entre distintos estados según el control que tengan de la CPU:

- **Ejecutándose**: La tarea tiene el control de la CPU.
- **Lista**: La tarea está lista para utilizar la CPU, pero no dispone de ella porque la está utilizando otra tarea de mayor prioridad.
- **Suspendida**: La tarea a terminado de ejecutarse y está en espera de una nueva activación. También pude estar suspendida si necesita algún recurso que no está disponible, permaneciendo bloqueada hasta que quede libre. Este último caso no lo consideraremos de momento pues suponemos que las tareas son independientes.

La transición entre estos estados se realizará según las siguientes acciones:



Transición entre estados

En las políticas de planificación con prioridades, además del mecanismo de elección de tarea, deben tener en cuenta el test de garantía que nos permita conocer si nuestro sistema será planificable o no.

De momento no consideraremos los protocolos de acceso a recursos ni el servicio aperiódico, pues en nuestro modelo simple consideramos todas las tareas periódicas e independientes.

5.5 Planificadores cícliclos

5.5.1 Supuestos

Los planificadores cíclicos son aplicables únicamente cuando existe un grado alto de determinismo, excepto en unas pocas tareas aperiódicas y esporádicas que se puedan acomodar dentro de este marco determinista. Por esta razón, asumiremos un modelo restringido a tareas periódicas. Los supuestos de los que partimos son los siguientes:

1. En el sistema existen un número N de tareas periódicas que permanece constante durante toda la vida del sistema.

2. Los parámetros de todas las tareas periódicas son conocidos.

3. Cada trabajo $J_{i,k}$ está listo para ser ejecutado a partir de su tiempo de activación $r_{i,k}$.

4. Existe un único procesador en el sistema.

Los parámetros de la tarea T_i con desfase S_i, periodo P_i, tiempo de ejecución C_i y plazo de finalización D_i, los definiremos con la 4-tupla (S_i, P_i, C_i, D_i).

Por ejemplo, en la tarea definida por (1, 10, 3, 6), el primer trabajo se activará en la unidad de tiempo 1, y debe finalizar antes de la unidad 7, el segundo se activa en el instante 11 y terminará antes del 17, etc. El factor de utilización de esta tarea será 0.3.

Por defecto, el desfase se toma como cero y el plazo de finalización igual al periodo. Cuando los parámetros de una tarea toman los valores por defecto se omiten de la tupla, escribiendo por ejemplo (10, 3, 6) si el desfase es 0 y (10, 3) si, además, el plazo de finalización coincide con el periodo.

5.5.2 Planificaciones estáticas

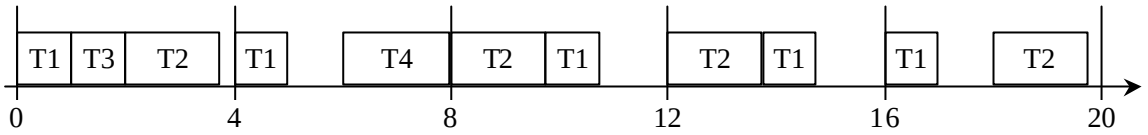
Cuando los parámetros de las tareas con plazos de finalización estrictos son conocidos antes de que comience la ejecución del sistema, la forma más directa de asegurar que cumplen sus plazos es construyendo un planificador cíclico de sus trabajos off-line. La planificación elegida define cuando se ejecuta cada uno de los trabajos.

De acuerdo con esta planificación, el tiempo de CPU asignado a cada trabajo será su tiempo máximo de ejecución, de forma que cada trabajo debe finalizar antes de su plazo de finalización.

En tiempo de ejecución, el planificador activará los trabajos de acuerdo con esta planificación. Salvo situaciones erróneas debido a que los trabajos consuma más CPU que su tiempo máximo de ejecución, todas las tareas cumplirán sus plazos de finalización. Como la planificación se calculo off-line, se pueden utilizar mecanismos todo lo complejos que sea necesario para elegir, de entre todas las planificaciones válidas, la que responda a unos determinados criterios, por ejemplo, que los trabajos finalicen los más cercano posible a sus tiempo de finalización para poder adelantar la ejecución de trabajo aperiódico.

Por ejemplo, consideremos el sistema formado por las tareas T_1 = (4, 1), T_2 = (5, 1.8), T_3 = (20, 1) y T_4 = (20, 2). El factor de utilización del sistema es 0.76 y su hiperperiodo es 20. Será pues suficiente construir la planificación durante el hiperperiodo pues, a partir de entonces, la planificación se repetirá cíclicamente para cada valor del hiperperiodo.

Una planificación posible podría ser:



Algunos intervalos, tales como (3.8, 4), (5,6) y (10.8, 12) no son utilizados por las tareas periódicas. Estos intervalos pueden ser usados para ejecutar tareas aperiódicas. Para este propósito, es más ventajoso dejar estos espacios esparcidos periódicamente durante la planificación.

Una forma de implementar el planificador es guardar la planificación calculada de antemano en una tabla. Cada entrada en la tabla (t_k, T(t_k)) corresponde a un instante en que se ha tomado una decisión de planificación y T(t_k) será el nombre de la tarea que se activa en ese instante o I, siendo I el instante en que el procesador queda ocioso.

A partir de esta tabla, el funcionamiento del planificador sería el siguiente:

1. El planificador utilizará un timer para activar cada una de las entradas de la tabla.

2. En la inicialización, el sistema debe crear todas las tareas que se van a ejecutar, reservándoles todos los recursos que vayan a necesitar.

3. Tras inicializar las tareas, se programará el timer para que active al planificador en el instante t_1 y lanzará la ejecución de la tarea T(t_0).

4. En cada interrupción del timer en el instante t_k se lanzará la tarea T(t_k) y se programará el timer para que despierte al planificador en el instante t_{k+1}.

5. Cuando se alcance el final de la tabla al final del hiperperiodo se volverá a leer la tabla desde el principio otra vez.

En este sentido, a las planificaciones estáticas periódicas se les llama también planificaciones cíclicas.

5.5.3 Planificadores cíclicos con tramas

En lugar de utilizar planificaciones cíclicas en las que los trabajos se activan en instantes arbitrarios, resulta más interesante dotar a las planificaciones de una cierta estructura.

Una estructura adecuada es que los instantes de decisión no sean arbitrarios si no que se realicen periódicamente. De esta forma, los instantes de decisión dividirán el

tiempo de ejecución en segmentos que se denomina *tramas*. Cada trama tendrá una duración *f*, que denominaremos *tamaño de la trama*.

Al principio de cada trama, además de lanzar la ejecución de las tareas que correspondan, puede interesar realizar algunas comprobaciones, como por ejemplo:

- Si cada trabajo a ejecutar en la trama ha sido activado y está listo para ejecución.

- Si se ha producido un overrun en la trama anterior.

Estos objetivos de diseño pueden hacer que unos valores para las tramas sean más adecuados que otros.

Idealmente, es deseable elegir las tramas para que cada trabajo pueda comenzar en una trama y finalizar su ejecución en la misma trama. Este objetivo se puede alcanzar si hacemos el tamaño de la trama lo bastante grande para que sea mayor que el tiempo de ejecución de todas las tareas,

$$f \geq \max(C_i),\ 1 \leq i \leq n \qquad \text{Ecu. (1)}$$

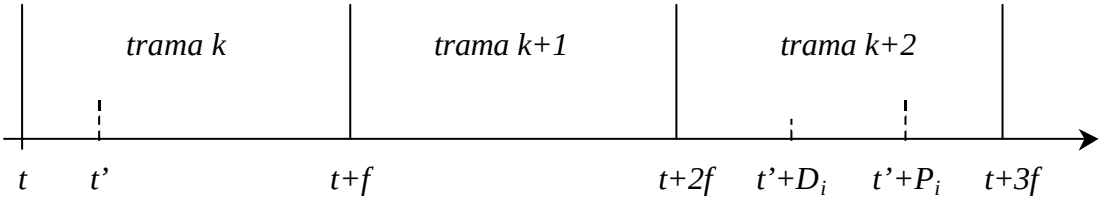
Para mantener la longitud del ciclo de ejecución lo más corta posible, interesa elegir *f* de forma que sea un divisor de H. Esto se cumple si *f* divide el periodo P_i de al menos una tarea T_i, es decir

$$P_i / f - \lfloor P_i / f \rfloor = 0 \qquad \text{Ecu. (2)}$$

para al menos un valor de *i*. Cuando esto ocurre, existe un número entero de tramas en un hiperperiodo *H*. Llamaremos *F* al número de tramas: $F = H / f$.

Por otro lado, para que el planificador pueda determinar cuando un trabajo finaliza antes de su plazo de ejecución, el tamaño de la trama debe ser lo suficientemente pequeño para que, entre el instante de activación de cualquier trabajo y su plazo de finalización, exista al menos una trama completa.

Supongamos una tarea definida por T_i, = (P_i, C_i, D_i,), en la que queremos que se cumpla la condición anterior.



Si *t'* es el instante de activación de la tarea, supongamos que se activa en la trama *k*, que comienza en el instante $t \leq t'$.

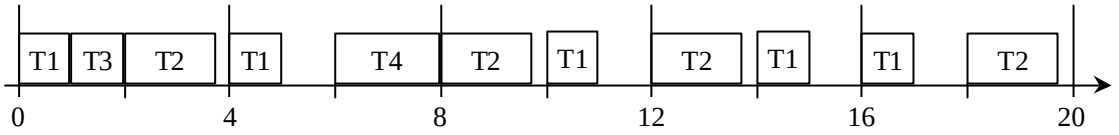
En el caso que $t = t'$ bastaría con que *f* fuera menor que D_i. Consideremos el caso más desfavorable en que $t' > t$. Nos interesa que, al menos, la trama (k+1) esté completa en el intervalo comprendido entre *t'* y $t'+D_i$. Si esto ocurre, entonces $t+2f \leq t'+D_i$, o lo

que es lo mismo $2f - (t' - t)$ £ D_i. Como $(t' - t)$ es mayor o igual que el $mcd(P_i, f)$, entonces la condición se cumple si es cierta la siguiente desigualdad:

$$2f - mcd(P_i, f) \leq D_i \qquad \text{Ecu. (3)}$$

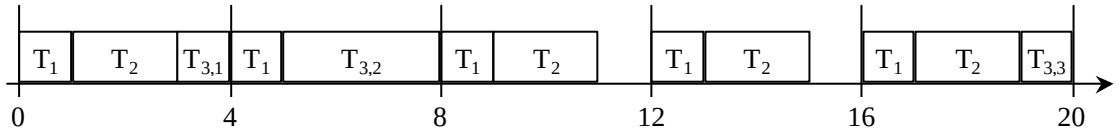
Esta inecuación se debe cumplir para todos los valores de $i = 1, 2, ..., n$.

En nuestro ejemplo, por la restricción (1) $f \geq 2$. Como el hiperperiodo es 20, los valores posibles para el tamaño de trama según (2) son 2, 4, 5, 10 y 20. No obstante, solo el valor 2 satisface la restricción (3), por tanto, este debe ser el valor del tamaño de trama más adecuado para construir el ejecutivo cíclico.



Sin embargo, los parámetros de algunos sistemas de tareas no pueden cumplir simultáneamente las tres restricciones para el valor del tamaño de trama. Por ejemplo, en el sistema T = {(4, 1), (5, 2, 7), (20, 5)}. Para que se cumpla (1) $f \geq 5$, pero según (3) $f \leq 4$. En esta situación, no hay más remedio que dividir cada trabajo de las tarea con tiempo de ejecución mayor que f en porciones o rodajas (slices) con tiempos de ejecución menores en las que su ejecución sea indivisible. De esta forma, podemos reducir el tamaño mínimo de f que impone la condición (1) tanto como sea necesario.

En el caso anterior, podemos dividir T_3 = (20, 5) en las tres subtareas (20, 1), (20, 3) y (20, 1). El sistema resultante estará compuesto por 5 tareas en las que se puede tomar f = 4. Las subtareas resultante de subdividir a T_3 se denominan $T_{3,1}$, $T_{3,2}$, y $T_{3,3}$.



Para satisfacer la condición (1) bastaría con haber subdividido T_3 en dos subtareas del tipo (20,2) y (20, 3) pero a la vista del planificador no queda ninguna trama en la que planificar la duración 2. De hecho, para poder planificar la de duración 3 es necesario que el plazo de finalización de T_2 sea 7, pues su segundo trabajo termina en el instante 11.

A la vista de este ejemplo, vemos que existen tres decisiones de diseño en el planificador:

1) Elegir el tamaño de la trama

2) Dividir los trabajos en rodajas

3) Situar las rodajas en las tramas

Por lo general, estas decisiones no se pueden tomar de forma independiente.

Para no penalizar el rendimiento, interesa subdividir los trabajos en el menor número de rodajas posibles, sin embargo, esto no siempre es factible. Si las rodajas son

grandes, es posible que no exista un planificador válido al no poder encajar las rodajas en las tramas. Y al contrario, si las rodajas son pequeñas es mas fácil encontrar un planificador válido.

Además, si las tareas no son divisibles, cuanto menor tengan que ser las rodajas más fácil será encontrar problemas debido a las zonas no divisibles de los trabajos.

5.5.4 Algoritmo para la construcción de planificadores estáticos

El problema general de elegir una longitud de trama lo suficientemente pequeña en un conjunto de tareas periódicas, segmentando las tareas si es necesario, y planificar los trabajos para que se cumplan las restricciones temporales y se respeten las secciones no interrumpibles, es un problema NP-duro.

Nosotros vamos a considerar el caso en el que no existen secciones no interrumpibles en las tareas, y veremos un algoritmo en que el tiempo necesario para alcanzar una solución depende de una función polinómica.

Se trata de un algoritmo iterativo que permite encontrar una planificación cíclica válida si ésta existe. El algoritmo se llama "*iterative network-flow algorithm*" o abreviado algoritmo INF.

Las premisas clave en este algoritmo es que las tareas pueden ser interrumpidas en cualquier instante y son independientes unas de otras.

Antes de aplicar el algoritmo INF, debemos encontrar todos los tamaños de trama posibles en el sistema. Las tramas deben cumplir las condiciones Ecu. (2) y Ecu. (3), pero no es necesario que cumpla la Ecu. (1).

Por ejemplo, los tamaños posibles para las tareas $T_1 = (4,1)$, $T_2 = (5, 2, 7)$ y $T_3 = (20,5)$ son 2 y 4, que satisfacen Ecu. (2) y Ecu. (3), pero no Ecu. (1).

El algoritmo INF trata de encontrar una planificación cíclica válida iterativamente para cada valor del tamaño de trama, comenzando por el mayor. Si el algoritmo falla para todos los valores posibles, entonces no existirá ninguna planificación válida que satisfaga las restricciones en el tamaño de trama aunque se dividan las tareas en subtareas.

Gráfico de red de flujo

En lugar de tener en cuenta las tareas, nombraremos los trabajos que hay que planificar en un hiperperiodo, compuesto por F tramas. Las restricciones a tener en cuenta para planificar los trabajos se representan en el gráfico de red de flujo del sistema.

El gráfico contiene los siguientes vértices y flechas; y cada flecha tendrá asociada una capacidad que será un número no negativo.

1. Existe un *vértice de trabajo* J_i que representa a cada trabajo J_i, para $i = 1, 2, .. N$.

2. Existe un *vértice de trama* llamado j que representa a cada trama del hiperperiodo, para $j = 1, 2, .. F$.

3. Existen dos vértices especiales llamados *fuente* y *sumidero*.

4. Existe una flecha (J_i, j) desde un vértice de trabajo J_i a un vértice de trama j, si el trabajo J_i puede ser planificado en la trama j. La capacidad de la flecha es el tamaño de trama f.

5. Existe una flecha desde el vértice fuente a cada uno de los vértices de trabajo J_i, y su capacidad es el tiempo de ejecución C_i del trabajo.

6. Existe una flecha desde cada vértice de trama j hacia el vértice sumidero, y la capacidad de esta flecha es f.

El flujo de una flecha es un número no negativo que satisface las siguientes restricciones:

a) No puede ser mayor que la capacidad del eje.

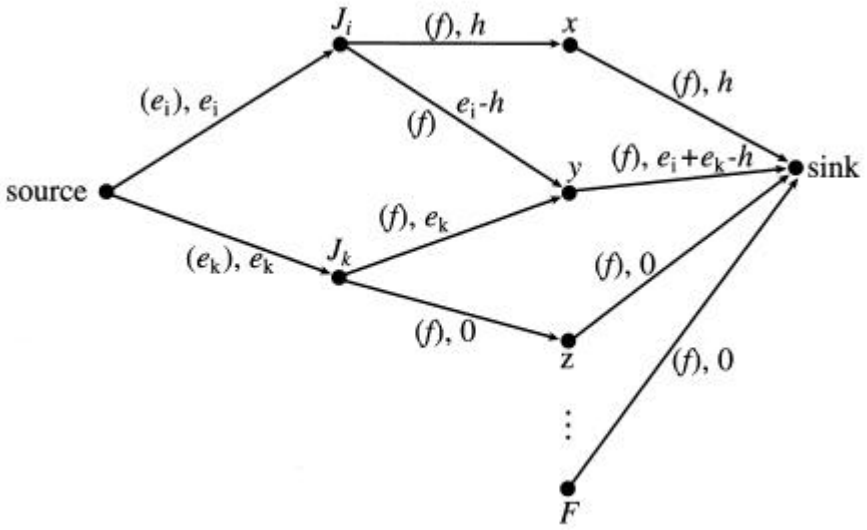
b) Con la excepción de los vértices fuente y sumidero, la suma de los flujos de las flechas que entran en cada vértice debe ser igual a la suma de los flujos en las flechas que salen del mismo vértice.

c) El flujo de los ejes (J_i, j) será la cantidad de tiempo que el trabajo J_i se ejecuta en la trama j. La suma de los flujos de todas las flechas que salen del vértice J_i debe ser C_i si el trabajo se puede planificar por completo. Por otro lado, la suma del flujo de todas las flechas que llegan a la trama j será como máximo f, que es la cantidad máximo de trabajo que se puede ejecutar en una trama.

d) En los flechas que parten del vértice fuente y llegan a un vértice de trabajo, el flujo es la porción del trabajo planificado.

e) En las flechas que parten de un vértice de trama y llegan al vértice sumidero, el flujo es la porción de trama rellenada por trabajos.

Un ejemplo sencillo en que solo se representan los trabajos J_i y J_k y tres tramas x, y y z sería el de la siguiente figura:



Este gráfico indica que el trabajo J_i se planifica en las tramas *x* e *y*, y el trabajo J_k se planifica en las tramas *y* y *z*.

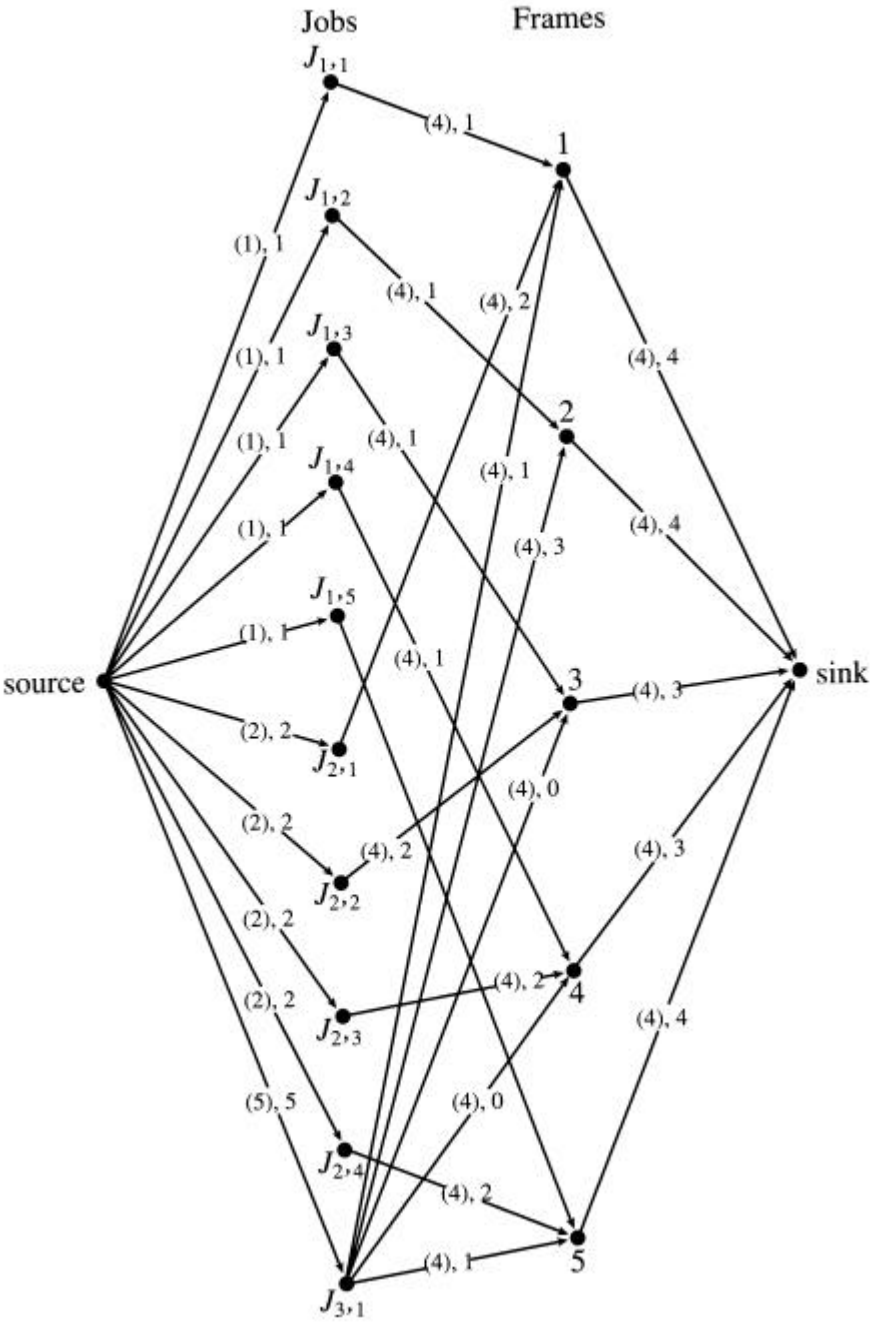
El flujo total del gráfico se define como la suma de los flujos que parten del vértice fuente, que será igual a la suma de los flujos que llegan al vértice sumidero. El flujo máximo será la suma de los tiempos de ejecución de todos los trabajos y se alcanzará cuando todos los trabajos puedan ser planificados en las tramas.

A la hora de planificar trabajos en tramas se deben tener en cuenta las siguientes restricciones:

1. Un trabajo no se puede planificar en una trama que comienza antes de su instante de activación.

2. La finalización de un trabajo en una trama no debe superar su límite de ejecución.

Teniendo en cuenta esto podemos aplicar el algoritmo en nuestro ejemplo de tres tareas usando el valor 4 para el tamaño de la trama.



El flujo total del gráfico es 18, que coincide con la suma de los tiempos de ejecución de todos los trabajos en el hiperperiodo, por tanto, representa una planificación válida.

Podría ocurrir que el flujo total fuera menor que la suma de los tiempos de ejecución. En ese caso habría que seguir el procedimiento iterativo para valores menores del tamaño de trama.

Por ejemplo, en las tareas T_1 = (4,3) y T_2 = (6, 1'5), los valores para el tamaño de trama son 2 y 4. En la primera iteración se llega a que el flujo total es 11'5, mientras que la suma de los tiempos de ejecución es 12. Por tanto, no existe una planificación válida para el tamaño de trama 4. Se debe seguir entonces con la segunda iteración para obtener una planificación válida.

5.5.5 Propiedades e inconvenientes

Los ejecutivos cíclicos tienen las siguientes propiedades:

- Son conceptualmente sencillos y fáciles de implementar. Basta con generar una tabla en la que se guardan que trabajos se deben activar en cada una de las tramas.

- Son altamente deterministas. Si está bien construido y se parte de datos correctos, ofrece buenas garantías de que se van a respetar los plazos temporales.

- Las restricciones de precedencia entre tareas y las secciones no interrumpibles se pueden tener en cuenta al generar el propio planificador.

- Los sistemas son relativamente fáciles de validar, verificar y certificar. Se puede detectar fácilmente si los trabajos no terminan antes de su plazo de finalización al principio de cada trama.

Los problemas que presenta son los siguientes:

- Es poco flexible y difícil de mantener. Cada vez que se modifica una tarea y varia su tiempo máximo de ejecución hay que rehacer toda la planificación. Por tanto, son adecuados para sistemas que raramente se modifican una vez se ha finalizado su construcción (por ejemplo, controladores empotrados sencillos).

- Las tareas esporádicas son difíciles de tratar, pues no se pueden introducir en la tabla de secuencia de tareas.

- El plan cíclico puede ser difícil de construir.

  ⇒ Si los periodos son de distintos órdenes de magnitud, el número de ciclos secundarios se hace muy grande.

  ⇒ Puede ser necesario dividir una tarea en varios procedimientos.

  ⇒ En el caso más general es NP-duro.

Debido a estos inconvenientes resulta ser aplicable en muy pocos casos, y por ello es poco utilizado, salvo en sistemas muy sencillos. En los sistemas complejos son mucho más los problemas que las ventajas que ofrece.

5.6 Planificadores con prioridades estáticas

5.6.1 Planificador Rate Monotonic (RM)

El planificador de prioridades monótonas en frecuencia (Rate Monotonic) será un planificador con prioridades con las siguientes características:

- Todas las tareas son periódicas e independientes unas de otras.

- El plazo de finalización de cada tarea se tomará igual a su periodo: ($D_i = P_i$).

- La asignación de prioridades se hará de forma inversa al periodo, es decir, una tarea de menor periodo que otra tendrá mayor prioridad (solo tiene importancia el valor relativo de la prioridad, el absoluto es indiferente).

- El planificador será expulsivo, es decir, si al activarse una tarea la CPU la está utilizando otra de menor prioridad (mayor valor), la de menor prioridad pasará al estado de 'lista' y tomará la CPU la tarea que se acaba de activar.

- El tiempo de cambio de contexto se considera despreciable.

- El conjunto de tareas es síncrono: ($S_i = 0, \forall i$).

El planificador RM no es un planificador óptimo en el caso general, pero si que lo es cuando las tareas son periódicas simples.

Definición:

Un conjunto de N tareas es periódico simple cuando para cualquier par de tareas T_i, T_k con $P_i < P_k$, entonces P_k es un número entero de veces P_i.

Teorema:

Un sistema de tareas periódico simple, independientes e interrumpibles, cuyos plazos de finalización son mayores o iguales que sus periodos, es planificable en un procesador de acuerdo con el algoritmo RM, si y solo si el factor de utilización total es menor o igual que uno.

Demostración:

La condición necesaria es trivial, pues un conjunto de tareas con factor total de utilización superior a uno no es planificable.

Para demostrar la condición suficiente, vamos a suponer que las tareas están en fase y que el procesador nunca está ocioso antes de que la tarea T_i incumpla su plazo en el instante t (de estar ocioso ese tiempo lo debería haber ocupado la tarea T_i pues está activa desde el mismo instante que las tareas de prioridad superior que habrán dejado el hueco).

Como los plazos de finalización coinciden con sus periodos, entonces t será un múltiplo entero de P_i.

Al ser el sistema periódico simple, t es también un múltiplo entero de P_k para cada tarea T_k de mayor prioridad que T_i. Por tanto, el tiempo de CPU usado antes de t es:

$$\sum_{k=1}^{i} \frac{C_k}{P_k} \bullet t$$

Que es *t* veces el factor de utilización de las tareas con mayor prioridad que T_i:

$$U_i = \sum_{k=1}^{i} U_k$$

Como T_i incumple su plazo en *t*, esto implica que el tiempo de CPU requerido excede de *t*, o sea:

$$t \bullet U_i > t \Rightarrow U_i > 1$$

Como partimos de la suposición contraria, se deduce que las tareas no incumplen su plazo en ningún momento y que, por tanto, el sistema es planificable.

Cuando estudiemos el planificador DM, veremos como caso particular que RM es óptimo entre cualquier otro planificador que utilice prioridades fijas.

Para el caso más general en el que el conjunto de tareas no es periódico simple, el test de garantía de esta política de planificación viene dado por el siguiente teorema:

Teorema:

Un conjunto de N tareas será planificable bajo la política de planificación Rate Monotonic si se cumple la siguiente desigualdad:

$$U = \sum_{i=1}^{N} \frac{Ci}{Ti} < N\left(2^{1/N} - 1\right)$$

Veamos un ejemplo:

	Ci	Pi
Task 1	1	4
Task 2	2	8
Task 3	3	12

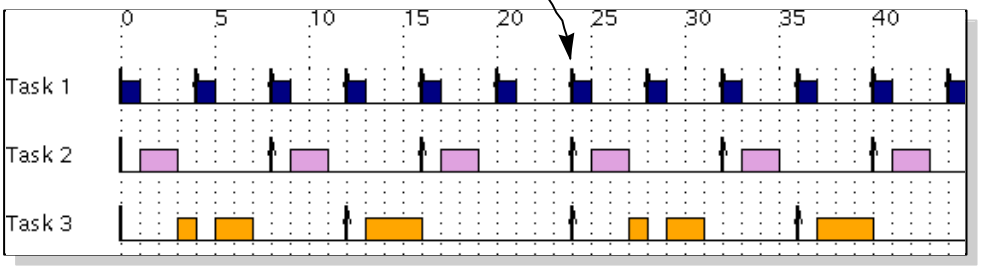
Factor de utilización

$$\frac{1}{4}+\frac{2}{8}+\frac{3}{12}=0.75$$

Limite de Utilización

$$3(2^{\frac{1}{3}}-1)=0.779$$

$$mcm(4,8,12)=24$$

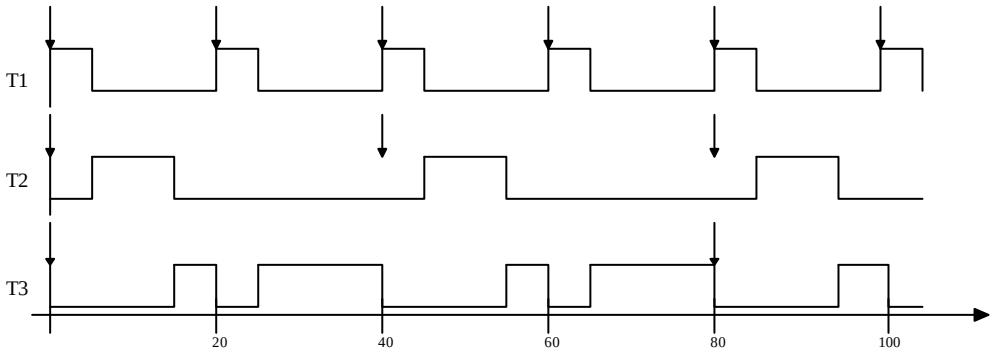


La desigualdad anterior fue probada por Liu and Layland en 1973. Este teorema da una condición sobre el factor de utilización del sistema, que depende exclusivamente del número de tareas. Nos garantiza que si se cumple la condición anterior, el sistema será planificable.

Ahora bien, podría ocurrir que el sistema fuera planificable y la desigualdad anterior fuera falsa. Se trata pues de una condición suficiente, pero no necesaria. Un ejemplo sería el siguiente:

Tareas	P	C	Pr	U
T1	20	5	1	0,250
T2	40	10	2	0,250
T3	80	40	3	0,500

Conjunto de tareas planificable con U=1



La expresión que da el límite del factor de utilización es una sucesión decrecientes de números reales que desde 1 (N = 1) hasta ln(2) = 0,693 (lim $N \rightarrow \infty$).

Una prueba sencilla para asegurar que el sistema es planificable es comprobar que el factor de utilización es menor que ln(2).

5.6.2 Análisis del tiempo de respuesta

La utilización del test de garantía basado en el factor de utilización tiene dos inconvenientes importantes:

- Es inexacto. Da una condición suficiente, pero no necesaria.
- No se puede aplicar a un modelo de procesos más general, en el que las prioridades no se asignen según indica el RM.

Vamos a ver ahora un test de garantía diferente.

Este test consta de dos fases. En la primera se utiliza un aproximación analítica para predecir el caso más desfavorable del tiempo de respuesta en cada tarea. Luego se comparan estos valores con los plazos de ejecución de cada tarea. Este proceso requiere que se analice cada tarea de forma independiente.

Para el proceso de más prioridad, el caso más desfavorable en el tiempo de respuesta corresponde a su tiempo de ejecución(esto es, $R_i = C_i$), pues ningún otro proceso lo puede retardar. El resto de procesos sufrirán una interferencia producida únicamente por los procesos de mayor prioridad. De forma general, para una tarea T_i su tiempo de respuesta será:

$$R_i = C_i + I_i$$

I_i es el valor máximo de la interferencia a la puede verse afectada la tarea T_i. Para el proceso de mayor prioridad ya hemos visto que este valor es cero. Para el resto de procesos el valor máximo se dará cuando todas las tareas de mayor prioridad se activen al mismo tiempo que la tarea T_i (este es el instante crítico).

Con esto, podríamos formular el siguiente teorema:

Teorema:

Un conjunto de N tareas será planificable bajo cualquier asignación de prioridades, si y solo si, cada una de las tareas cumple su plazo de finalización en el peor de los casos.

El tiempo de finalización da cada tarea en el peor de los casos ocurre cuando todas las tareas de prioridad superior se inician a la vez que esta.

Un instante crítico (el peor caso) válido para todas las tareas será el momento en que todas las tares piden ejecutarse a la vez. En un sistema síncrono esto ocurre, al menos, en el instante inicial.

Consideremos una tarea T_j de mayor prioridad que la tarea T_i. En el intervalo $[0, R_i)$ se activará un número determinado de veces (una por lo menos). El número de veces que se activará será:

$$numero\_de\_activaciones = \left\lceil \frac{R_i}{P_j} \right\rceil$$

La función techo ($\lceil \rceil$) da el número entero más pequeño superior al resultado de la fracción sobre la que actúa. Por ejemplo, de 1/3 será 1 y de 5/6 será 2. La definición para los valores negativos no afecta en nuestro caso.

Cada activación de la tarea T_j producirá una interferencia sobre la tarea T_i de valor:

$$interferencia\_maxima = \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

El total de la interferencia a la que se ve sometida la tarea T_i por todas las de mayor prioridad será:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

donde $hp(i)$ es el conjunto de tareas de mayor prioridad que la tarea T_i.

Sustituyendo este valor en la expresión del tiempo de respuesta tendremos:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

Aunque la formulación de la ecuación de la interferencia es exacta, el valor de la interferencia sigue siendo desconocido pues está en función de R_i que es justo lo que queremos calcular.

La ecuación final tiene en ambos lados el valor R_i, pero es difícil de despejar debido al operador techo. En general existirán más de un valor que den solución a esta ecuación, pero a nosotros sólo nos interesará el valor más pequeño. Todos serán válidos para el tiempo de respuesta en el peor de los casos de la tarea T_i, pero para que se cumplan los plazos bastará con que uno sea menor o igual que D_i, y si hay alguno siempre será el menor.

Una forma sencilla de resolver la ecuación es por medio de un mecanismo iterativo. Suponga mos la sucesión:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

La sucesión $\{w_i^0 = C_i, w_i^1, w_i^2, \ldots w_i^n, \ldots\}$ es monótonamente no decreciente. Esto es evidente ya que la fracción del sumatorio primero será 0 y después del

numerador tomará valores que nunca serán más pequeños que el anterior, debido al signo '+' de la expresión.

Cuando se obtenga que $w_i^n = w_i^{n+1}$ se habrá encontrado una solución para la ecuación. Si $w_i^n < D_i$ entonces w_i^n será la solución más pequeña, y habremos encontrado la solución que buscábamos.

Si la ecuación no tiene solución, entonces la sucesión aumentará indefinidamente (esto ocurrirá para las tareas de baja prioridad si el total del factor de utilización del procesador supera el 100%). Si se alcanza un valor superior al periodo de la tarea, se puede asumir que no cumplirá su límite temporal.

Si obtenemos todos los tiempos de respuesta de cada tarea menores que sus correspondientes periodos, entonces el sistema será planificable.

Veamos un ejemplo de cálculo del tiempo de respuesta.

Tareas	P	C	Pr	U
T1	7	3	1	0,429
T2	12	3	2	0,250
T3	20	5	3	0,250

Tabla de tareas

En este caso, el factor de utilización es U = 0,929, que es superior a la condición de planificabilidad basada en la utilización de Liu and Layland para el RM, que da un resultado de 0,780 con N=3.

Veamos que el sistema si es planificable utilizando el método del cálculo del tiempo de respuesta.

Los distintos valores que obtenemos para las w_i^n son:

Para la tarea 1, por ser la mas prioritaria, su tiempo de respuesta será su tiempo de proceso.

Para la tarea 2 tenemos:

$$w_2^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6 = w_2^2$$

Y para la tarea 3 obtenemos la secuencia de valores:

$$w_3^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_3^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_3^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_3^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_3^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

Con lo que finalizamos el cálculo del valor de R_3 al repetirse los valores.

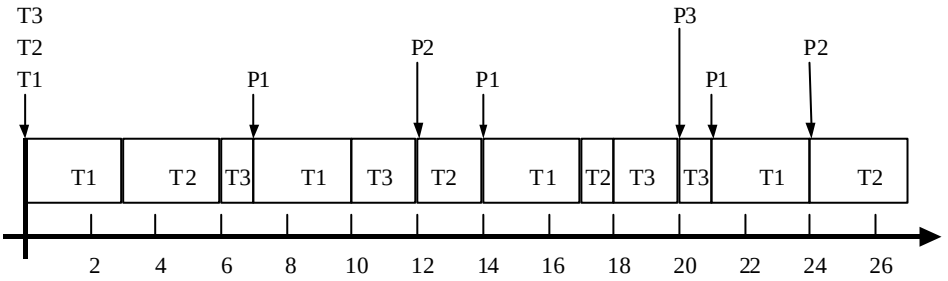
Con ello ya podemos comprobar que, efectivamente, se cumplen las restricciones temporales:

Tareas	P	R
T1	7	3
T2	12	6
T3	20	20

Tiempos de respuesta

cumpliéndose los plazos para todas las tareas.

El comportamiento del sistema se puede representar en un diagrama de Gantt de la siguiente forma:



En este ejemplo vemos que en el primer ciclo de la tarea T3 la utilización de la CPU es del 100% (caso más desfavorable). Es otros ciclos ya quedarán algunos huecos hasta producirse un aprovechamiento medio en el hiperperiodo igual al calculado (0,929)

El procedimiento descrito para obtener los tiempos de respuesta resulta un tanto engorroso para hacerlo a mano, sobre todo si el número de tareas es grande, pero se puede automatizar fácilmente.

5.6.3 Planificador Deadline Monotonic (DM)

Esta política de planificación es idéntica a la Rate Monotic, pero ahora los plazos de finalización pueden ser distintos a los periodos (menores o iguales):

$$D_i = P_i$$

La asignación de prioridades se hace en este caso en orden inverso al plazo de finalización, de forma que la tarea con plazo de finalización más breve tenga mayor prioridad.

Como test de garantía para la política de planificación Deadline Monotic podemos tomar el resultado obtenido en el análisis del tiempo de respuesta, pero comparando con el plazo máximo de finalización de cada tarea en lugar de su periodo.

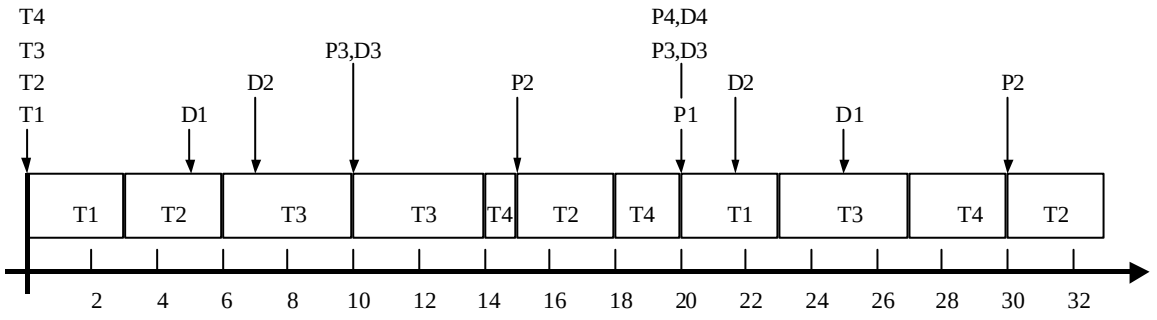
Veamos un ejemplo de asignación de prioridades con el planificador DM.

Tareas	P	D	C	Pr	R
T1	20	5	3	1	3
T2	15	7	3	2	6
T3	10	10	4	3	10
T4	20	20	3	4	20

Tabla de tareas. U=0.90

Los tiempos de respuesta han sido calculados con el procedimiento descrito anteriormente. Vemos que en todos los casos éstos son inferiores a los plazos de finalización, por tanto, el sistema es planificable bajo el DM.

El comportamiento del sistema se puede representar en un diagrama de Gantt de la siguiente forma:



Este conjunto de tareas no es planificable utilizando el RM. Pues si se disminuyen los periodos para que sean iguales a los plazos de finalización, la utilización de la CPU se hace superior al 100% (U=1.58).

El planificador DM, al igual que el RM, no es óptimo en el caso más general, pero si que es óptimo de entre los planificadores con prioridades fijas.

Teorema:

Un sistema de tareas independientes e interrumpibles, que están en fase y sus plazos de finalización son menores o iguales que sus respectivos periodos, pueden ser planificadas en un único procesador bajo la política DM sí y solo sí pueden ser planificadas con cualquier otro planificador de prioridades fijas.

Demostración:

Este teorema es cierto porque es posible transformar cualquier planificación válida realizada por un planificador de prioridades fijas en una planificación válida compatible con el planificador DM.

Para ver esto, se revisan todas las tareas por orden creciente a su plazo de finalización, empezando por la de menor plazo de finalización. Cuando encontremos dos tareas T_i y T_{i+1} tales que $D_i < D_{i+1}$ pero que T_i tiene menor prioridad que T_{i+1} con este planificador, intercambiamos las prioridades de las tareas para que sean conformes a DM y modificamos la planificación de acuerdo a las nuevas prioridades intercambiando las tareas. Este intercambio es posible hacerlo sin perder los plazos de finalización porque las tareas están en fase.

Al terminar de revisar las tareas habremos transformado el planificador en un DM y hemos visto que las planificaciones que haría el DM siguen siendo válidas.

En algunos sistemas, los plazos de finalización de cada tarea T_i son proporcionales a sus periodos: $D_i = \delta P_i$, $\delta > 0$. En estos sistemas, la asignación de prioridades que realiza DM es la misma que RM, por tanto, RM es un caso particular de DM. Esto nos permite formular el siguiente corolario.

Corolario:

El algoritmo de planificación RM es óptimo entre los algoritmos de planificación de prioridades fijas cuando los plazos de finalización de las tareas son proporcionales a sus periodos.

5.6.4 Tiempo de ejecución en el peor de los casos

En todas las políticas de planificación descritas, se asume que se conoce el tiempo de ejecución en el peor de los casos C_i para todas las tareas. Este tiempo es el máximo tiempo de CPU que puede necesitar una tarea cuando es activada.

La estimación del tiempo de ejecución en el peor de los casos se puede obtener utilizando métodos de medida o de análisis. El problema que aparece en los métodos de medida es que resulta difícil de asegurar que la ejecución que se está midiendo corresponde realmente al peor de los casos. La desventaja de los métodos de análisis es que se debe disponer de un modelo del procesador que se utiliza (incluyendo cachés, pipelines, estados de espera de la memoria, etc.)

La mayoría de las técnicas de análisis llevan con sigo dos actividades distintas:

1. La primera descompone el código de las tareas y lo representa en un grafo de bloques básicos. Estos bloques básicos representan segmentos de código que se ejecutan linealmente (sin bucles ni ejecución condicional)

2. La segunda actividad del análisis tiene en cuenta el código máquina correspondiente a cada bloque básico y utiliza el modelo del procesador para estimar su tiempo de ejecución en el peor de los casos.

Una vez se conocen los tiempos de todos los bloques, el grafo se puede simplificar. Por ejemplo, la ejecución condicional entre dos bloques básicos pueden reducirse a uno único, tomando el mayor de los dos tiempos. Los bucles se pueden simplificar en un único bloque si se conoce un límite para el número de iteraciones.

Se pueden utilizar técnicas de reducción de grafos más sofisticadas si se dispone de la suficiente información sobre la semántica de los bloques. Para ver un ejemplo de este caso, consideremos el siguiente código:

```
for i in 1..10 loop
      if Cond then
            -- Bloque basico de coste 100
      else
            -- Bloque basico de coste 10
      end if
end loop
```

Si no se dispone de información más detallada, el coste total de esta construcción será de 10 x 100 más el coste que introduce la propia realización del bucle, digamos por ejemplo 1005. Es posible, no obstante, deducir por medio de un análisis del código, que la condición que produce la ejecución del bloque de coste 100, puede darse como mucho en tres ocasiones. En tal caso, obtendríamos un valor del costo menos pesimista que sería igual a 375.

Si se pretende realizar sobre un código un análisis del coste de ejecución en el peor de los casos, se deben tener en cuenta una serie de restricciones. Por ejemplo, todos los bucles y procedimientos recursivos deben estar acotados. De lo contrario sería imposible deducir de antemano cuando finalizaría el código de la ejecución.

Por otra parte, se debe realizar un análisis del código generado por el compilador para la generación del código en cada una de las construcciones, y los procedimientos de optimización que pudiera utilizar.

5.6.5 Mejoras del modelo simple

En nuestro modelo simple de tareas introducíamos unas restricciones muy fuertes para poder realizar el análisis que hemos hecho. Aún con la ventaja del planificador DM en el que los plazos de finalización pueden ser inferiores al periodo, interesa aproximarnos más a los casos reales. Vamos a ver como afecta eliminar las siguientes restricciones en el test de planificabilidad que hemos estudiado:

- Las operaciones del núcleo de multiprogramación son instantáneas. En concreto, se considera nulo el tiempo de cambio de contexto utilizado para dejar de ejecutar una tarea y retomar otra.

- Las tareas son independientes unas de otras. No se considera la existencia de secciones críticas ni de recursos compartidos que obligue a la sincronización entre tareas.

- Todas las tareas son periódicas, o esporádicas transformadas en periódicas utilizando el tiempo mínimo entre dos activaciones consecutivas como su periodo.

5.6.5.1 Cambios de contexto

El tiempo de cambio de contexto C_S consiste en tres operaciones:

1. Decidir que tarea es la siguiente que se va a ejecutar

2. Salvar el estado del proceso en ejecución (save)

3. Restaurar el estado del nuevo proceso (restart)

Cada expulsión de una tarea por la llegada de otra más prioritaria implica dos cambios de contexto, el primero al activar la nueva tarea y el segundo al volver a la que se ha interrumpido.

Este tiempo se puede incluir en la formulación de nuestro planificador añadiendo al tiempo de computo de la tarea más prioritaria dos veces el tiempo de cambio de contexto. Aunque el segundo cambio de contexto implicaría solo a la tarea menos prioritaria que es interrumpida, es más seguro incluirlo en la más prioritaria pues es difícil saber con seguridad cuantas veces es interrumpida.

$$C'_i = C_i + 2C_S$$

Esto habría que hacerlo en todas las tareas excepto en la menos prioritaria.

Los tiempos de respuesta se calcularían utilizando el nuevo tiempo de cómputo y se compararían los resultados así obtenidos con los plazos de finalización de cada tarea.

Vemos pues que considerar el tiempo de cambio de contexto no varia la formulación de nuestro planificador ni del test de garantía.

5.6.5.2 Gestión de recursos

Queremos eliminar ahora la hipótesis de que las tareas son independientes unas de otras. Esto engloba los siguientes aspectos:

- La sincronización entre tareas.

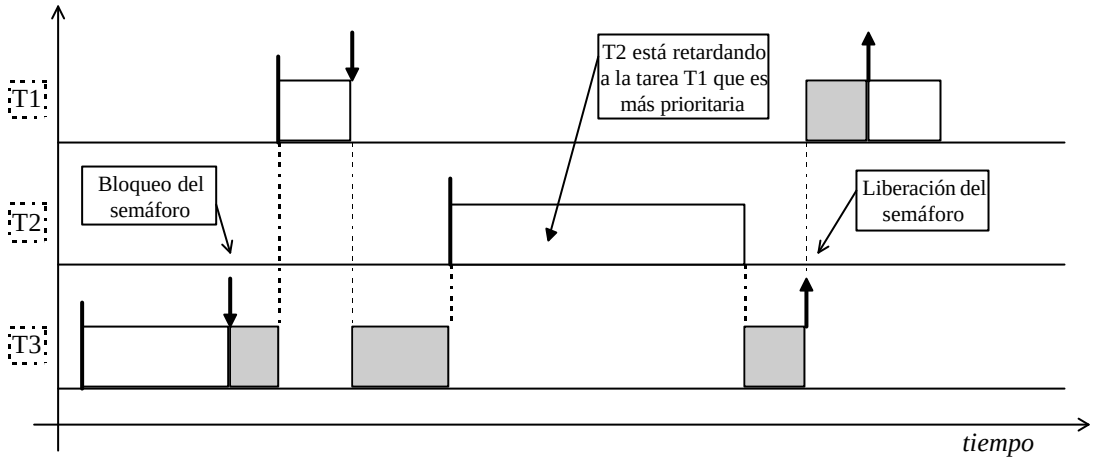
- La comunicación. Espera de la llegada de información.

- La utilización de recursos comunes (periféricos o zonas de exclusión mutua).

Nosotros vamos a ver solamente la utilización de semáforos o monitores para resolver el uso de secciones críticas (zonas de exclusión mutua).

En las aplicaciones de tiempo real no se pueden utilizar directamente los servicios de sincronización por semáforos que ofrecen la mayoría de los sistemas operativos pues hay que evitar la inversión de prioridad.

La inversión de prioridad se produce cuando una tarea más prioritaria queda bloqueada por la ejecución de una tarea de prioridad inferior. Esto puede ocurrir en el caso de dos tareas de distinta prioridad que utilicen un mismo semáforo y aparezca una tercera de prioridad intermedia que interrumpa a la de menor prioridad cuando está bloqueando a la de mayor prioridad por medio del semáforo.



La inversión de prioridad no es admisible en un sistema de tiempo real pues lo hace indeterminista. En nuestro ejemplo, no se puede determinar de antemano cuanto tiempo está T3 bloqueando a T1 al utilizar el semáforo de acceso a la sección crítica, pues puede ser interrumpida por T2.

Existen varios mecanismos para evitar la inversión de prioridad. Los más conocidos son:

- **Ordenar la ejecución de las tareas** (off-line). Se establece precedencia entre las tareas. Esto se hace modificando las prioridades, pero en ocasiones esto no es viable pues puede hacer el sistema no planificable.

- **Sección crítica no interrumpible**. Esto se puede utilizar con efectividad cuando los tiempos de utilización de las secciones críticas son muy pequeños.

- **Sección crítica interrumpible**. Se modifican la prioridad de las tareas que entran en las secciones críticas. Básicamente son protocolos de herencia de prioridad. Los que vamos a ver son:

    * Priority Inheritance Protocol

    * Priority Ceiling Protocol

    * Priority Semafore Protocol

Los protocolos de herencia de prioridad utilizan semáforos para acceder a las secciones criticas y consisten en modificar de alguna manera las prioridades de las tareas para evitar la inversión de prioridad.

5.6.5.2.1 Priority Inheritance Protocol

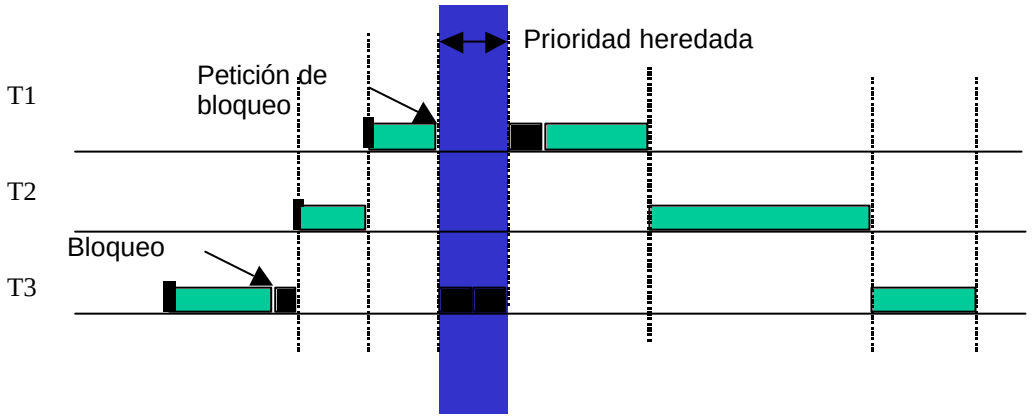
Este protocolo consiste en modificar las prioridades de las tareas de la siguiente forma:

1) Cuando una tarea T_h intenta acceder a una sección crítica que esté bloqueada por otra tarea T_l, la tarea bloqueante T_l hereda la prioridad de la tarea más prioritaria que quiere acceder T_h.

2) La herencia de prioridad solo se produce cuando una tarea de más prioridad queda bloqueada. No se produce si la tarea T_l que queda bloqueada tiene menor prioridad que la propia T_h.

3) Al liberar el semáforo de la sección crítica, la tarea T_l recupera su prioridad propia.

Un ejemplo gráfico de herencia de prioridad seria el siguiente:



Cuando la tarea T1 solicita el bloqueo del recurso, la tarea queda suspendida porque lo está usando T3. Esto hace que T3 tome la prioridad de T1 y, por tanto, quien pasa a ejecutarse es T3 y no T2, para evitar que T2 retrase la finalización de T1.

Este protocolo presenta las siguientes propiedades:

- Una tarea de alta prioridad sólo puede ser bloqueada por una de baja prioridad si esta última está dentro de una zona crítica cuando se activa la primera.

- Una tarea de menos prioridad puede bloquear a otra de más prioridad, como mucho durante una zona crítica (en las posteriores ya habrá terminado la tarea de más prioridad).

- La sección crítica de mayor duración determina el tiempo máximo que una tarea puede bloquear a otra de mayor prioridad.

- El número de veces que una tarea puede quedar bloqueada viene dado por el min(n, m), donde “n” es el número de tareas de menor prioridad y “m” el número de regiones críticas (o semáforos) usados por estas (suponiendo que no están anidados).

Además de estas propiedades, presenta los siguientes inconvenientes:

– No evita el interbloqueo

– El tiempo de bloqueo puede ser excesivo

5.6.5.2.2 Priority Ceiling Protocol

El algoritmo que controla el funcionamiento de este protocolo es el siguiente:

1) A cada semáforo se le asigna una prioridad igual a la de la tarea más prioritaria que le puede bloquear.

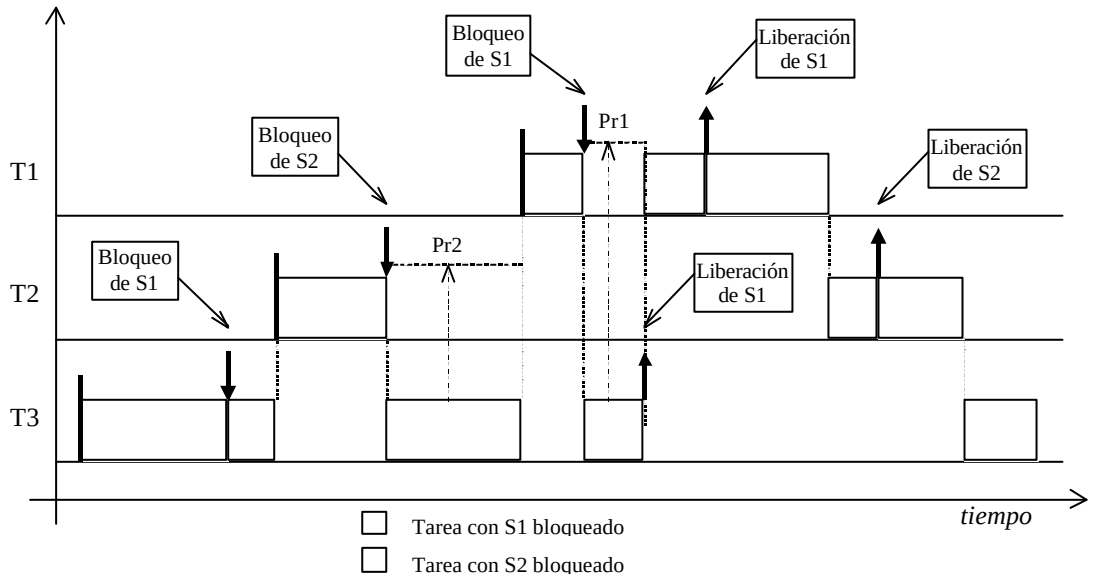
2) Una tarea puede cerrar un semáforo si su prioridad es estrictamente mayor que la prioridad de todos los semáforos que en este momento estén cerrados (no se le sube la prioridad)

3) Una tarea mantiene su prioridad mientras no bloquee a otras tareas más prioritarias, en cuyo caso hereda la prioridad máxima de entre las que bloquea.

4) Al abandonar la tarea una sección crítica recupera su prioridad.

Veamos un ejemplo con tres tareas y dos recursos. El recurso 1 (R1) lo utilizan T1 y T3, mientras que el recurso 2 (R2) lo utiliza la tarea T2. Como en los ejemplos anteriores, la tarea T1 es la más prioritaria.

El recurso R1 tiene la prioridad de T1 y el R2 la prioridad de T2.



Cuando T2 intenta cerrar el semáforo S2, queda bloqueada sin llegar a utilizarlo por el punto 2) pues tiene menor prioridad que S1. En el mismo instante se sube la prioridad de T3 a la de T2 pues la está bloqueando por el punto 3).

Cuando T1 pide el semáforo S1, queda bloqueada porque T3 lo está ocupando. Entonces T3 toma la prioridad de T1 por el punto 3).

El interbloqueo que se produce cuando dos tareas que utilizan dos recursos, los intentan cerrar en orden inverso, no puede producirse pues el techo de ambos sería el mismo y la condición 2) es “estrictamente mayor”. En el ejemplo anterior, si T3 cerrara S2 después de S1, y T2 cerrara S1 después de S2, T2 quedaría bloqueada sin cerrar a S2 pues S1 ya se está usando y su techo seria igual que su prioridad (si no estuviera T1). En tal caso T3 nunca se quedaría bloqueada en S2 debido a T2.

Las propiedades de este protocolo son las siguientes:

- Impide el interbloqueo, aunque se puede evitar con otros algoritmos
- No existen bloqueos encadenados
- El máximo tiempo de espera está acotado por la sección crítica más larga de las tareas menos prioritarias

Y los problemas que presenta son:

- Es un algoritmo pesimista, es decir, puede bloquear tareas intermedias sin necesidad.
- Es complejo de implementar. Por este motivo no es muy utilizado.

5.6.5.2.3 Priority Semafore Protocol

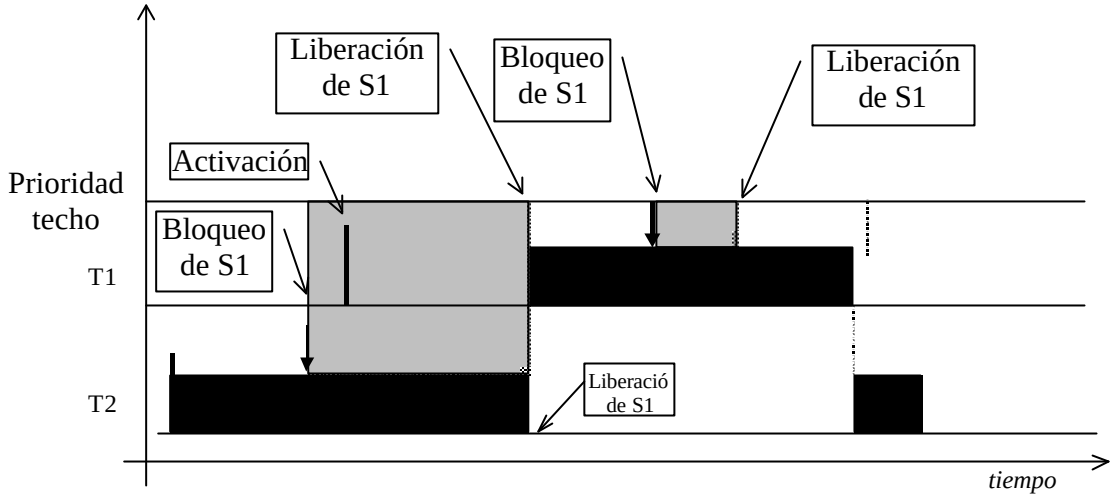
También se llama Inmediate Ceiling Protocol (ICP) o Priority Protected Protocol (PPP) en POSIX.

Este protocolo es una versión simplificada del PCP. Es más fácil de implementar y más eficiente.

El algoritmo es el siguiente:

1) A cada semáforo se le asigna una prioridad igual a la de la tarea más prioritaria que le puede bloquear.
2) Cuando una tarea entra en la sección crítica, hereda la prioridad que se le ha asignado a su semáforo.

Un ejemplo de dos tareas que usan el mismo semáforo es el siguiente:



El funcionamiento es similar a hacer las secciones críticas no interrumpibles, excepto para las tareas más prioritarias que no utilizan recursos.

El problema que presenta es que interfiere más al funcionamiento de otras tareas que el PCP.

El máximo tiempo de espera de una tarea está acotado también por la sección crítica más larga de las tareas de menor prioridad.

5.6.5.2.4 Test de garantía

Si realmente queremos introducir estos protocolos en el planificador, hemos de tener en cuenta los efecto del uso de recursos sobre el test de garantía.

Veíamos que la condición de planificabilidad para tareas independientes era:

$$\forall \mathrm{i},\ 1 \le \mathrm{i} \le \mathrm{n} \qquad R_i \le \mathrm{D_i}$$

para el planificador DM. En el RM $\mathrm{D_i} = \mathrm{P_i}$

En la expresión anterior teníamos que:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

La modificación en el test de planificabilidad consiste en evaluar un factor de interferencia B_i que corresponderá al tiempo máximo que cada tarea puede estar bloqueada por otras tareas que utilizan sus mismos recursos.

El test de planificabilidad afectará a los valores R_i y será ahora:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + B_i$$

El factor de interferencia dependerá del protocolo que se esté utilizando para el uso de los semáforos.

Por ejemplo, en el protocolo de techo de prioridad (PCP), el factor de interferencia corresponde al máximo tiempo de espera que veíamos que estaba acotado por la sección crítica más larga de las tareas menos prioritarias.

En este caso el test de planificabilidad deja de ser una condición necesaria y suficiente pues B_i es una estimación del peor de los casos. Podría no ser una condición necesaria pues aunque el test nos dijera que no es planificable, en la realidad los B_i podrían ser menores que los estimados y el sistema si ser planificable. De todas formas, siempre será una condición suficiente.

5.6.5.3 Servicio aperiódico

Se llama servicio aperiódico al conjunto de tareas que no son críticas, es decir, que no tienen plazos de finalización estrictos.

En un sistema de tiempo real es muy usual la existencia de este tipo de tareas, que también pueden tener su importancia. Por tanto, no se trata de relegarlas al último lugar pase lo que pase, sino intentar servirlas lo mejor posible ofreciendo el menor tiempo de respuesta posible, manteniendo siempre las garantías de ejecución para las tareas críticas aún en situaciones de sobrecarga del sistema.

Las tareas acríticas también se llaman aperiódicas, de ahí que las posibles formas de planificar estas tareas se llama servicio aperiódico.

Existen varios tipos de servicios aperiódicos:

- **Background**. Las tareas aperiódicas se ejecutan cuando el procesador esta ocioso, y no se altera la ejecución de las tareas periódicas. Es sencillo de implementar, pero ofrece un tiempo de respuesta pobre.

- **Bandwidth Preserving**. En este servidor se retrasa la ejecución de las tareas críticas sin dejar de cumplir sus plazos. Los mecanismos para estos son, de peor a mejor respuesta:
  - Polling
  - Priority exchange
  - Deferable server
  - Sporadic server

- **Slack Stealing**. Son mecanismos de extracción de holgura. Hay de dos tipos:
  - Estático
  - Dinámico

Este último tipo de servidor es optimo.

5.6.5.3.1 Background

Cuando no hay trabajo periódico que ejecutar se sirven las tareas aperiódicas. Ofrece mal tiempo de respuesta pero es muy simple de implementar.

Se implementa utilizando una tarea periódica con tiempo de computación y periodos grandes, dándole la prioridad más baja del sistema.

Se le llama servidor porque esta tarea va ejecutando los cuerpos de las distintas activaciones de las tareas aperiódicas. Cuando se activa una tarea aperiódica esta se pondrá en una cola de la cual estará leyendo el servidor aperiódico atendiéndolas en orden de llegada.

5.6.5.3.2 Polling

En este servidor se añade una tarea periódica a las tareas críticas (periódicas) originales. Durante el tiempo de cómputo de esta nueva tareas servidora estará activo el trabajo aperiódico (si lo hay).

La prioridad que le corresponda será la que se determine (en el planificador RM dependerá de su periodo).

Se le asigna también un tiempo de cómputo, que será el tiempo que dispondrá para ejecutar el trabajo aperiódico en cada activación suya. Si una tarea aperiódica necesitara más tiempo que el *Ci* que le corresponde en un periodo, se ejecutará en varias activaciones del servidor.

El tiempo de respuesta en este caso es algo mejor que el servidor por background, pues ahora el servidor no tiene la prioridad más baja, y se puede ejecutar aunque haya trabajo periódico de tareas críticas. De todas formas la ejecución de las tareas críticas dentro de sus plazos queda garantizado por el test de planificabilidad. Para que este se cumpla se habrá asignado mayor o menor periodo y más o menos tiempo de computo al servidor.

Tiene la desventaja que si al ejecutarse el servidor no hay trabajo aperiódico pendiente, este tiempo se pierde retardando de igual manera a las tareas periódicas de menos prioridad.

5.6.5.3.3 Priority exchange

Con este servidor se pretende evitar el inconveniente del anterior de forma que no se pierdan los tiempos en que no hay trabajo aperiódico.

También se añade una tarea periódica con una cierta prioridad intermedia con las siguientes consideraciones.

- Si cuando el servidor está activo hay tareas aperiódicas pendientes, estas se sirven hasta que se consume todo el tiempo del servidor.

- Si, por el contrario, no hay trabajo que servir, el servidor intercambia su prioridad con la tarea periódica más prioritaria que esté lista en ese momento (de las menos prioritarias que el), y sólo durante el tiempo de cómputo del servidor.

- La capacidad del servidor se repone completamente al inicio de cada periodo, pues conforme pasa el tiempo la prioridad del servidor va disminuyendo y se va gastando su tiempo de cómputo

La mejor forma de implementar el servidor, con periodos grandes o pequeños, depende del resto de tareas y de las características deseables del servidor.

La implementación del servidor presenta el inconveniente que hay que modificar el planificador para que soporte el intercambio de prioridad.

5.6.5.3.4 Deferable server

También añade una nueva tarea periódica con prioridad intermedia con las siguientes consideraciones:

- Si mientras el servidor está activo hay tareas aperiódicas pendientes estas se sirven hasta que se completa todo el tiempo del servidor. Para ello se está llevando la cuenta de su C_i hasta que se agota.

- La capacidad del servidor se repone completamente al inicio de cada periodo.

La capacidad de este servidor es inferior al caso anterior, pero resulta más sencillo de implementar.

El motivo por el que disminuye la capacidad es porque se incumple una suposición básica en el test de planificabilidad que es que una tarea no puede suspenderse a si misma (en nuestro caso, la tarea del servidor se suspende si no hay trabajo aperiódico hasta que llega). El tal caso podría variar la situación de carga a puntos que no corresponden al comienzo del periodo.

El tiempo de cómputo que ofrece este servidor es bastante bajo.

5.6.5.3.5 Sporadic server

Se añade una nueva tarea periódica con las siguientes características:

- Conserva su capacidad al igual que el Deferable Server

- El mecanismo de relleno viene dado por el instante en que se hace y la cantidad de tiempo que se rellena:

    * Instante de relleno: Cuando el servidor se activa, y su capacidad es mayor que cero, se programa un instante de relleno igual al instante actual más el periodo del servidor.

    * Cantidad a rellenar: Se determina cuando el servidor pasa de activo a ocioso, o cuando se ha agotado la capacidad del servidor. En este momento se decide que la cantidad a rellenar es igual a la cantidad consumida desde la última vez que pasó de ocioso a activo.

- El servidor se planifica como una tarea periódica normal.

5.6.5.3.6 Slack stealing

El mecanismo del servidor consiste en averiguar cuando es lo más tarde que yo puedo ejecutar el trabajo periódico, de manera que se sigan respetando los plazos. Al retrasar el trabajo periódico se adelanta la ejecución del trabajo aperiódico.

Un planificador normal intenta ejecutar el trabajo urgente (periódico) lo antes posible, dejando el tiempo sobrante (holgura) después de ejecutar las tareas. Con este servidor es al revés, el tiempo sobrante de las tareas se reserva antes y se les deja holgura cero entre el tiempo de respuesta y el plazo de finalización.

Ahora no se añade una tarea nueva. Lo que se hace es ejecutar el trabajo aperiódico en los tiempos de cómputo que han quedado libres.

La acción de robar holgura con pocas tareas no es complicado, pero con muchas tareas resulta muy difícil.

En el caso del **planificador estático** se calculan los tiempos de activación de todas las tareas hasta el hiperperiodo. Durante la ejecución, cuando se activa una tarea aperiódica, se consulta la tabla y se decide si se va a ejecutar.

Lo complicado en este caso es consultar la tabla pues, si no hay tareas aperiódicas se debe intentar adelantar la ejecución del trabajo periódico, y las previsiones hechas con el planificador estáticos hay que rehacerlas sobre la marcha.

En el **planificador dinámico** se intenta evitar la construcción de la tabla. Se basa en calcular en el momento que llega trabajo aperiódico el trozo de tabla que afecta en ese momento y decidir si se va a ejecutar.

El servidor dinámico es óptimo, pero los tiempos de cómputo son elevados.

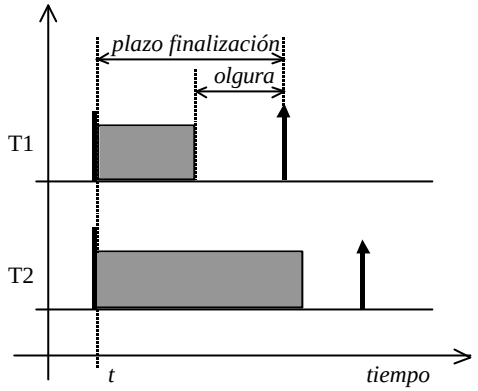
5.7 Planificadores con prioridades dinámicas (EDF)

Los planificadores dinámicos son, al igual que los estáticos, planificadores basados en prioridades, pero se caracterizan por que las prioridades no son estáticas durante toda la vida del sistema sino que varían en el tiempo siguiendo un algoritmo que define el planificador.

Básicamente se utilizan dos planificadores dinámicos, estos son:

- **EDF** (Earliest Deadline First). Se asignan las prioridades dando preferencia a las tareas con plazo de finalización más próximo.

- **LLF** (Least Laxity First). Se asignan las prioridades dando preferencia a las tareas con menor holgura.



En el ejemplo del dibujo, en el tiempo *t*, según el planificador EDF T1 tendría más prioridad que T2, pero según el LLF, T2 tendría más prioridad que T1

El inconveniente de los planificadores dinámicos es que requieren un elevado tiempo de cálculo durante la ejecución del sistema. Por este motivo, el único que resulta realmente útil es el EDF. Nosotros vamos a ver solamente el planificador EDF.

Un planificador dinámico puede expresarse definiendo la tarea en ejecución T_x en cada momento de la siguiente forma:

$$T_x \text{ Ejecutandose} \Leftrightarrow \forall\, x,i \in \text{ Activas } f\,(T_x,\, t) \geq f\,(T_i,\, t)$$

Si la función *f* no depende de *t*, se tratará un planificador estático.

En el planificador EDF la función *f* es tal que da mayor prioridad a las tareas con distancia absoluta al plazo de ejecución más pequeña.

La prioridad de cada tarea se asigna en el momento de la activación. La implementación no tiene un excesivo coste, no mucho más que pare el DM.

Este planificador presenta los siguientes inconvenientes:

- Es poco estable ante sobrecargas transitorias. Si una tarea que toma la ejecución consume mucha CPU, puede afectar a tareas más importantes.

- Es difícil de llevar a la práctica. Pocos Sistemas Operativos comerciales lo implementan.

La ventajas que ofrece son:

- Es óptimo. Consigue la máxima utilización del procesador. Si el no consigue planificar un conjunto de tareas, tampoco lo conseguirá otro planificador.

- Precisa de un tiempo de cálculo (overhead) no excesivo.

Actualmente se suele utilizar es Sistemas de Tiempo Real no estricto (soft real time), sobre todo por su novedad y su poca estabilidad.

5.7.1 Planificador óptimo

Los planificadores EDF y LLF son planificadores óptimos.

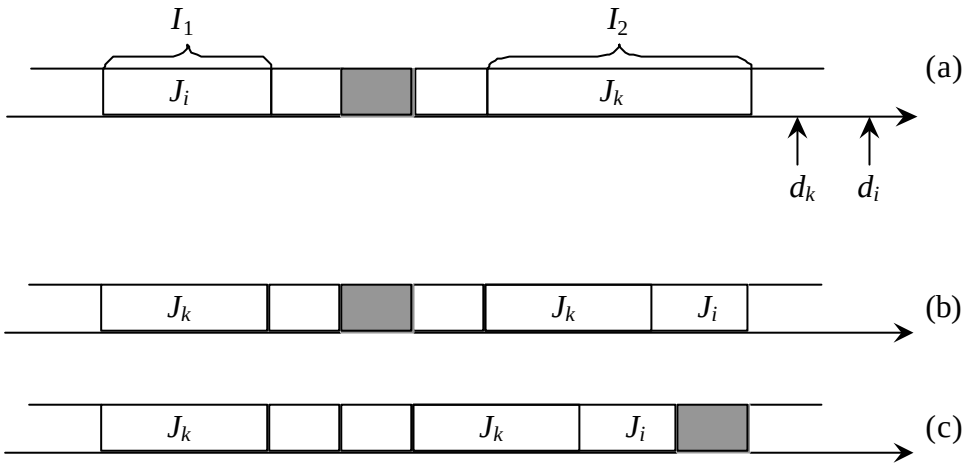
Veámoslo con el siguiente teorema para el planificador EDF:

Teorema

El planificador EDF puede producir una planificación válida en un solo procesador de un conjunto de trabajos **J** independientes con expulsión permitida y con plazos de ejecución arbitrarios, si y solo si **J** tiene planificaciones válidas.

Demostración

La prueba se basa en comprobar que cualquier planificador válido para **J** se puede transformar en un planificador EDF. Para ver esto, supongamos que dos trabajos J_i y J_k son planificados en los intervalos I_1 y I_2 respectivamente. Además, el plazo de ejecución d_i para J_i es posterior al plazo d_k para J_k. pero I_1 es anterior a I_2 (parte (a) de la figura).



En esta situación pueden ocurrir dos casos:

1. Que el instante de activación de J_k sea posterior al final de I_1. En este caso J_k no puede ejecutarse antes de J_i y, por tanto, el planificador cumple las reglas del EDF.

2. Que el instante de activación de J_k sea anterior al final de I_1. Este es el único caso que debemos considerar.

Sin perder generalidad, podemos asumir que la activación de J_k se produce entes del comienzo de I_1 (de no ser así, bastaría con considerar I_1 a partir del momento de la activación).

Para transformar este planificador, conmutamos J_i y J_k. En concreto, si I_1 es más corto que I_2, tal como se muestra en la figura, se mueve la porción de J_k que cabe en I_1 sobre I_1, y J_i se mueve por completo al final de I_2, tal como se muestra en la parte (b) de la figura. Tras este cambio, el sistema sigue siendo planificable.

Se puede hacer un cambio similar si I_1 fuera mayor que I_2. Ahora moveríamos por completo el trabajo J_k al principio de I_1 y pasaríamos a I_2 solo la parte de J_i que cupiese.

El resultado de este cambio es un planificador que se ejecuta con las condiciones del EDF.

Pero el planificador no es todavía un EDF si quedan porciones de tiempo en el que el procesador no está ocupado habiendo trabajos activados. De ser así, faltaría adelantar la ejecución de los trabajos activos para ocupar los tiempos no ocupados, pasando éstos después de ejecutar los trabajos activos (parte (c) de la figura).

De esta forma, el planificador obtenidos es del tipo EDF.

Como estas transformaciones siempre son posibles, queda demostrado que un planificador EDF siempre puede producir panificaciones válidas en el caso de que estas existan.

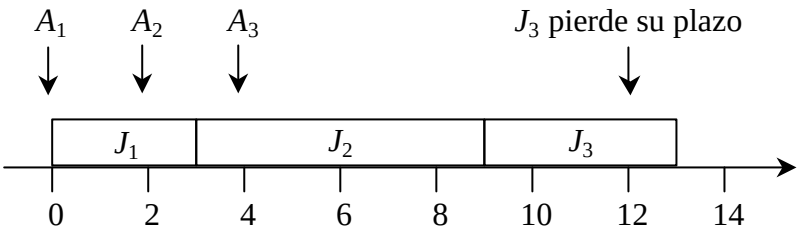
Para el caso del planificador LLF se puede realizar una demostración similar. Por tanto, el LLF también es un planificador óptimo.

La desventaja del LLF frente al EDF es que en el primero se necesita conocer el tiempo de ejecución de los trabajos para determinar su prioridad, cosa que normalmente resulta difícil de precisar con exactitud.

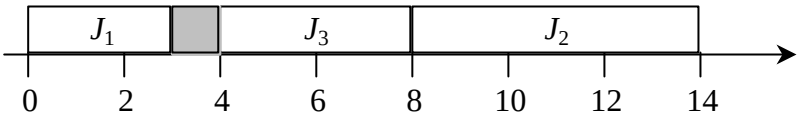
Para la demostración del teorema hemos partido de la hipótesis que el planificador es expulsivo, es decir, que los trabajos son interrumpibles en cualquier momento. Veamos un contraejemplo que nos muestra que esto debe ser así:

Consideremos tres trabajos J_1, J_2, y J_3 con los instantes de activación 0, 2 y 4 respectivamente; sus tiempos de ejecución son 3, 6 y 4; y sus límites de ejecución 10, 14 y 12.

En la parte (a) de la figura vemos que este conjunto de tareas no es planificable bajo el criterio de EDF pues J_3 pierde su plazo de ejecución al no poderse interrumpir J_2. Sin embargo, si existe una planificación válida tal como se muestra en la parte (b) de la figura.



(a)



(b)

5.7.2 Test de planificabilidad

Con el planificador EDF, en el caso de que los plazos de finalización correspondan al periodo de cada tarea, el sistema será planificable, sí y solo sí

$$D = \sum_{i=1}^{N} \frac{Ci}{Di} \leq 1 \qquad (D = \text{Densidad}) \qquad D_i = P_i :\Rightarrow D = U$$

El límite de planificabilidad en este caso es del 100% de utilización del procesador.

En el caso que los plazos de finalización sean menores que los periodos, la condición es solo suficiente.

El que no es una condición necesaria se ve en el contraejemplo {(2, 0’6, 1), (5, 2’3)}.

5.7.3 Gestión de recursos

Los protocolos son versiones adaptadas de los planificadores estáticos. En este caso se llama Stack Resource Policy (SRP).

Vamos a tener en cuenta las siguientes consideraciones:

- Supondremos en todo momento que $D_i = P_i$.
- A cada tarea se le asigna una prioridad, igual a la asignada por DM, estática.
- El techo de prioridad de cada semáforo es el máximo de las prioridades de las tareas que pueden utilizar el recurso.
- Se define el techo del sistema como el máximo entre la prioridad de la tarea (la asignada estáticamente) activa en ese momento y el techo de los semáforos en uso.

En base a esto, el protocolo de utilización de los semáforos será el siguiente:

- Sólo se permitirá iniciar una nueva activación de una tarea si la prioridad de esta es estrictamente mayor que el techo del sistema, independientemente de que use recursos.

- Una vez una tarea comienza su ejecución, todas las peticiones de recursos se concederán.

Como se puede suponer, este algoritmo es muy restrictivo. Retarda la ejecución de las tareas antes de que usen los semáforos y afecta incluso a las tareas que no usan recursos.

Las propiedades de este protocolo son las siguientes:

- Evita los interbloqueos

- El tiempo máximo de bloqueo soportado por T_i es el producido por la sección crítica más larga de entre las tareas con plazo de finalización relativo mayor que T_i.

- El SRP puede ser utilizado con prioridades dinámicas o estáticas

- El máximo número de cambios de contexto desde que una tarea se activa hasta que finaliza es de 2.

El test de planificabilidad se modifica de la siguiente forma:

$$\underset{1<k<n}{\forall}\left(\sum_{i=1}^{k}\frac{Ci}{Di}\right)+\frac{B_k}{D_k}\leq 1$$

donde B_k es la duración de las sección crítica más larga de entre las tareas con D_i menor o igual que D_k.

5.7.4 Servicio aperiódico

Casi todos los algoritmos para servicio aperiódico en planificadores estáticos están también para dinámicos.

Por ejemplo, la versión del Priority Exchange se llama Dynamic Priority Exchange, y se definiría con las siguientes reglas:

- El servicio aperiódico lo realiza una tarea.

- Al servidor se le asigna una capacidad y un periodo.

- Al inicio de cada periodo, la capacidad del servidor se repone.

- Cada activación de cada tarea periódica puede hacer también de servidor aperiódico. Inicialmente, estos servidores tienen capacidad cero (esto es porque no podemos intercambiar prioridades).

- Cuando un servidor con capacidad disponible está ocioso, su capacidad se transfiere a la del servidor que en ese momento está utilizando el procesador.

- Cuando llega una tarea aperiódica, ésta se ejecuta durante el tiempo de servicio que tiene cada activación.