Planificación en sistemas de Tiempo Real

Aspectos básicos de la problemática de planificación para Tiempo−Real. Principales resultados de la teoría de prioridades estáticas.

José Ismael Ripoll Ripoll[1]

En las aplicaciones de tiempo real, el funcionamiento correcto NO sólo depende de la corrección de los resultados de los cálculos, sino que también depende del instante en el que éstos son obtenidos. El sistema operativo, y concretamente la política de planificación debe garantizar que todas las tareas críticas sean capaces de cumplir sus plazos de ejecución (obtener los resultados antes del plazo máximo especificado).

Índice de contenido

[1]Mailto:iripoll@disca.upv.es http://www.disca.upv.es/~iripoll

1. Tareas de Tiempo−Real

Desde el punto de vista de la planificación, el sistema operativo considera a las tareas como procesos que consumen una cierta cantidad de tiempo de procesador, y a las que hay que asignarles esa cantidad cada cierto tiempo. Tanto los datos que necesita cada tarea, como el código que ejecutan y los resultado que producen son totalmente irrelevantes para el planificador.

A partir de ahora supondremos que existen tres tipos de tareas en función de las restricciones temporales:

Tareas Periódicas:

- **Ci**: Tiempo de cómputo en el peor de los casos. (Para obtener este valor se ha de analizar el código de la tarea).
- **Pi**: Periodo de repetición. Cada Ti unidades de tiempo se ha de activar la tarea.
- **Di**: Plazo máximo de finalización (deadline). Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización u obtención de los resultados.

Tareas esporádicas:

- **Ci**: Tiempo de cómputo en el peor de los casos. (Para obtener este valor se ha de analizar el código de la tarea).
- **Pi**: Periodo mínimo entre dos peticiones consecutivas.
- **Di**: Plazo máximo de finalización (deadline). Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización de la ejecución de ésta.

Tareas Aperiódicas:

- **Ci**: Tiempo de ejecución en el peor de los casos. (Para obtener este valor se ha de analizar el código de la tarea).
- **Di**: Plazo de finalización (deadline). Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización de la ejecución de ésta. Este atributo es opcional: las tareas aperiódicas críticas si que tienen plazo de finalización, mientras que las tareas aperiódicas no críticas no tienen plazo de ejecución.

Si atendemos a las características semánticas de las tareas, podemos dividirlas en dos grupos:

Críticas: El fallo de una de estas tareas puede ser catastrófico.

Opcionales (no críticas): Se pueden utilizar para refinar el resultado dado por una tarea crítica, o para monitorizar el estado del sistema, etc.

Los estudios de planificación suponen que todas las tareas críticas son periódicas o esporádicas; y las tareas opcionales se tratan como aperiódicas, bien con plazo o sin plazo de finalización asignado. Como más adelante veremos, los tests de planificabilidad sólo son capaces de analizar a priori la correcta ejecución de las tareas periódicas o esporádicas.

2. Características de las políticas de planificación

Los objetivos que persigue toda política de planificación de tiempo real son:

- Garantizar la correcta ejecución de todas las tareas críticas.
- Ofrecer un buen tiempo de respuesta a las tareas aperiódicas sin plazo.
- Administrar el uso de recursos compartidos.
- Posibilidad de recuperación ante fallos software o hardware.
- Soportar cambios de modo, esto es, cambiar en tiempo de ejecución el conjunto de tareas. Por ejemplo: un cohete espacial tiene que realizar acciones muy distintas durante el lanzamiento, estancia en órbita y regreso; en cada fase, el conjunto de tareas que se tengan que ejecutar ha de ser distinto.

Inicialmente supondremos las siguientes simplificaciones. Conforme estudiemos con más detalle los algoritmos de planificación, iremos eliminando estas restricciones:

- Las tareas son independientes.
- No comparten recursos, ni se comunican entre ellas.
- Todas las tareas son periódicas.
- Los tiempos de cambio de contexto son despreciables.

3. Definiciones previas

Un **planificador** es un método para asignar recursos (el tiempo de procesador).

Diremos que un **conjunto de tareas** es **factible o planificable** si existe algún planificador que sea capaz de cumplir las restricciones de todas las tareas (en nuestro caso las restricciones temporales: los plazos de ejecución).

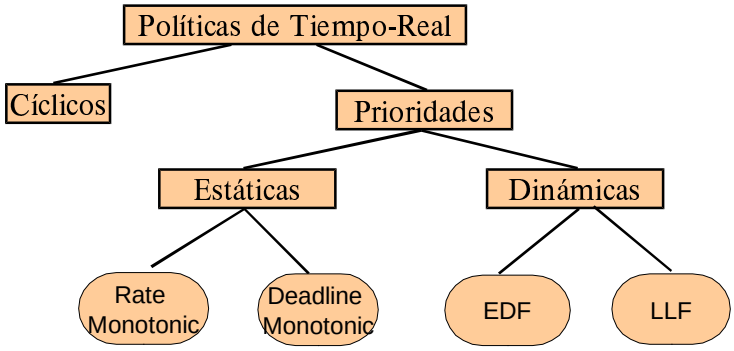
Un **planificador es óptimo** si es capaz de planificar correctamente cualquier conjunto de tareas factible.

El **factor de utilización** es $U = \sum_{i=1}^{k} \frac{C_i}{P_i}$

El **hiperperiodo** es el mínimo común múltiplo de los periodos de las tareas.

4. Clasificación de las políticas de planificación

Los primeros planificadores se diseñaban "a mano", esto es, se construía durante la fase de diseño del sistema un plan con todas las acciones que tenía que llevar a cabo el planificador durante la ejecución. Durante la ejecución, el planificador tan sólo tenía que consultar la tabla (plan). Las ordenes de planificación que estaban en la tabla se repetían constantemente. A estos planificadores se les llama **CÍCLICOS**.



El principal problema que presentan es la poca flexibilidad a la hora de modificar alguno de los parámetros de las tareas, pues ello conlleva la re−elaboración de todo el plan.

Aún hoy este tipo de planificación se utiliza en la industria.

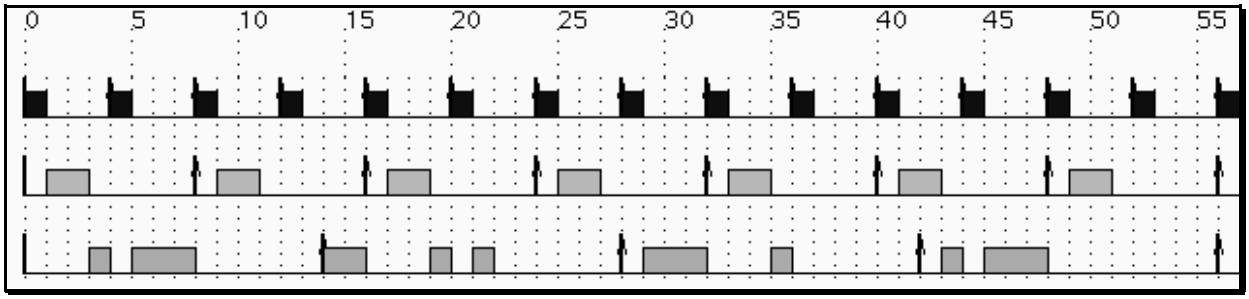
El otro gran grupo de planificadores son los basados en prioridades. Estos a su vez se dividen en prioridades estáticas y prioridades dinámicas. Nuestro estudio se centrará principalmente en los planificadores estáticos.

En los planificadores estáticos, durante la fase de diseño a cada tarea se le asigna una prioridad. Después, durante la ejecución, el algoritmo de planificación simplemente tiene que ordenar la ejecución de las tareas en función de la prioridad asignada.

5. Rate Monotonic

Durante la fase de diseño, a cada tarea se le asigna una prioridad inversamente proporcional al periodo (o plazo de finalización). Durante la fase de ejecución, el planificador selecciona aquella activación con mayor prioridad. La planificación es expulsiva.

A todas las restricciones antes mencionadas, hay que añadir que el plazo máximo de ejecución ha de ser igual al periodo de cada tarea.



Ejemplo de planificación del Rate Monotonic.

Test de garantía (factor de utilización)

Un conjunto de n tareas será planificable bajo el Rate−Monotonic si se cumple que el factor de utilización del conjunto de tareas es menor que la expresión: $n(2^{\frac{1}{n}}-1)$

n	*U(n)*
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
∞	0.693

Valores U(n)

Esto es:

$$\frac{C_1}{P_1}+\cdots+\frac{C_n}{P_n}=U\le U(n)=n(2^{\frac{1}{n}}-1)$$

Si aplicamos este test al siguiente conjunto de tareas:

T1=(1,5); T2=(2,8); T3=(3,14)

Tenemos lo siguiente:

$$\frac{1}{5}+\frac{2}{8}+\frac{3}{14}=0.2+0.25+0.214=0.664\le U(3)=3(2^{\frac{1}{3}}-1)=0.779$$

Con lo que podemos asegurar que este conjunto de tareas es planificable por el Rate−Monotonic.

Test de garantía (tiempos de finalización)

Un conjunto de »n« tareas será planificable bajo cualquier asignación de prioridades si y sólo si: Cada tarea cumple su plazo de ejecución en el peor caso. Siendo el peor caso de cada tarea aquel en el que todas las tareas de prioridad superior se activan a la vez que ésta.

Si lo expresamos matemáticamente el test queda formulado como sigue:

$$\forall 1\le i\le n, W_i\le D_i$$

donde W_i representa el instante en el que finaliza la ejecución en el peor caso. Este valor se obtiene de la siguiente expresión:

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

Función techo. Devuelve el entero por exceso de su argumento.

donde hp(i) representa el conjunto de tareas con prioridad mayor que la tarea i. Tal como vemos a ambos lados de la igualdad tenemos el valor W_i , que no se puede despejar. La solución de esta expresión se obtiene de forma iterativa:

$$W_i^0 = C_i$$

$$W_i^1 = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{W_i^0}{P_j} \right\rceil C_j$$

$$W_i^1 = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{W_i^0}{P_j} \right\rceil C_j$$

$$\ldots\ldots$$

$$W_i^k = W_i^{k+1}$$

La iteración se repite hasta encontrar dos valores consecutivos (W_i^k y W_i^{k+1}) iguales. El último valor de W obtenido es el que luego emplearemos en la expresión del test de garantía. Si en alguna iteración se obtiene un valor de W mayor que el plazo máximo de ejecución de la tarea, entonces esta tarea no será planificable y por tanto el sistema tampoco.

Ejemplo de aplicación del test

Dado el siguiente conjunto de tareas: T1=(C=1,P=4); T2=(C=2,P=9); T3=(C=4,P=10).

Hay que analizar que cada una de las tareas cumple la condición que exige el test:

Primera tarea:

$$W_1^0 = C_i = 1$$

$$W_1^1 = C_1 + 0 = 1 \leq D_1 = 4$$

Segunda tarea:

$$W_2^0 = C_2 = 2$$

$$W_2^1 = C_2 + \sum_{\forall j \in 1} \left\lceil \frac{W_2^0}{P_j} \right\rceil C_j = 2 + \left\lceil \frac{2}{4} \right\rceil 1 = 3$$

$$W_2^2 = C_2 + \sum_{\forall j \in 1} \left\lceil \frac{W_2^1}{P_j} \right\rceil C_j = 2 + \left\lceil \frac{3}{4} \right\rceil 1 = 3$$

$$W_2^2 = W_2^1 = 3 \leq D_2 = 9$$

Tercera tarea:

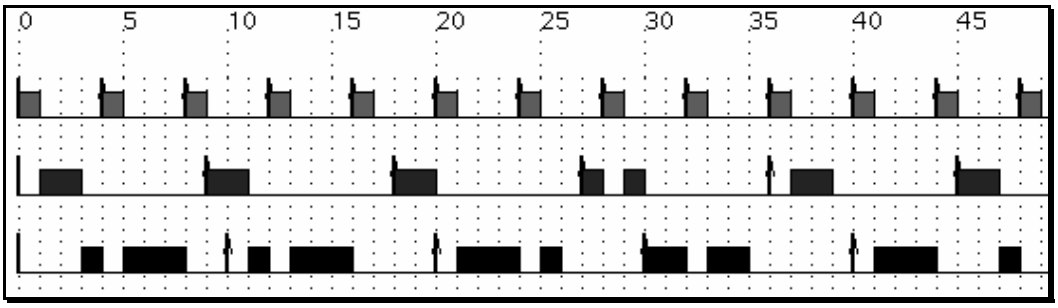
$$W_3^0 = C_3 = 4$$

$$W_3^1 = C_3 + \sum_{\forall j \in 12} \left\lceil \frac{W_3^0}{P_j} \right\rceil C_j = 4 + \left\lceil \frac{4}{4} \right\rceil 1 + \left\lceil \frac{4}{9} \right\rceil 2 = 7$$

$$W_3^2 = C_3 + \sum_{\forall j \in 12} \left\lceil \frac{W_3^0}{P_j} \right\rceil C_j = 4 + \left\lceil \frac{7}{4} \right\rceil 1 + \left\lceil \frac{7}{9} \right\rceil 2 = 8$$

$$W_3^3 = C_3 + \sum_{\forall j \in 12} \left\lceil \frac{W_3^0}{P_j} \right\rceil C_j = 4 + \left\lceil \frac{8}{4} \right\rceil 1 + \left\lceil \frac{8}{9} \right\rceil 2 = 8$$

$$W_3^3 = W_3^2 = 8 \leq D_3 = 10$$



Resultado de planificación de este conjunto de tareas.

Si utilizar el primer test propuesto (basado en el factor de utilización) veremos que NO cumple el test, y por tanto no sabríamos si el conjunto de tareas es planificable o no. El primero es un test necesario pero no suficiente. Mientras que este nuevo test es necesario y suficiente.

6. Deadline Monotonic

Esta política de planificación es en principio idéntica al RM pero eliminando una restricción: las tareas pueden tener un plazo de ejecución menor o igual a su periodo. Las prioridades se asignan de forma inversamente proporcional al plazo máximo de ejecución. Se puede ver que el RM es un caso particular del DM en el que todas las tareas tienen plazo de ejecución igual al su periodo.

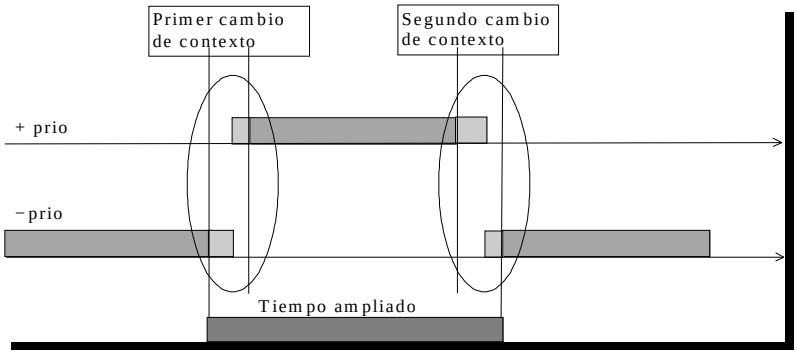
El test de planificabilidad (garantía) es el test de *tiempos de finalización*. Recordemos que el enunciado de éste dice que es valido para cualquier asignación de prioridades.

Un aspecto importante del DM es que **es óptimo entre los planificadores basados en prioridades estáticas**[2]. En otras palabras, si el sistema no es planificable, entonces no existe otra asignación de prioridades que haga al sistema planificable.

7. Cambios de contexto

Hasta ahora habíamos supuesto que los cambios de contexto no consumían tiempo. Pero la realidad es muy distinta. En los actuales procesadores RISC, que cuentan con un gran número de registros, el tiempo de salvar todos estos registros en memoria principal y recargar los registros de otra tarea para continuar con su ejecución no es despreciable.

Si estudiamos en detalle cómo se producen los cambios de contexto veremos que podremos acotar el tiempo que cualquier tarea pierde en estos menesteres.



Cada vez que una tarea de más prioridad se activa (suponemos que en esos momentos está en ejecución otra de menor prioridad) ha de expulsar a la menos prioritaria (lo que implica suspender la menos prioritaria y activar la nueva tarea); y al finalizar la ejecución hay que reponer la ejecución de la tarea interrumpida (eliminando la más prioritaria y reponiendo la tarea anterior). Es importante notar que no se producen más cambios de contexto (si suponemos que no hay más activaciones).

[2] Lo que no significa que sea capaz de planificar todos los conjuntos de tareas factibles.

Los dos cambios de contexto han sido forzados por la tarea más prioritaria, por lo que podemos considerar este tiempo como tiempo de cómputo de ésta. Así, la tarea menos prioritaria puede no tener en cuenta los tiempos de cambio de contexto producidos por tareas más prioritarias.

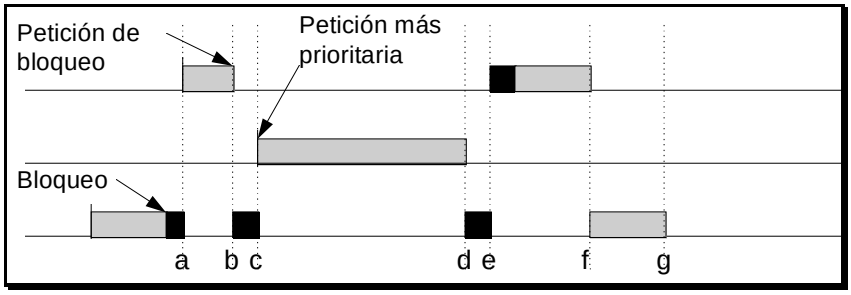
El test de planificabilidad no se ve alterado por los cambios de contexto si modificamos los tiempos de cómputo de todas las tareas con dos veces el tiempo de cambio de contexto.

A la tarea menos prioritaria si bien no puede provocar expulsiones a otras tareas, también hay que añadirle los tiempos de cambio de contexto pues en realidad si que expulsa a otra tarea: la tarea idle. Ésta es una tarea comodín utilizada en casi todas las implementaciones de sistemas operativos para simplificar el código del planificador.

8. Uso de recursos

Por uso de recursos entenderemos tanto la sincronización entre tareas, como la comunicación (bloqueo directo) o el uso de recursos comunes (bloqueo indirecto). En otras palabras, cualquier método por el cual una tarea se puede bloquear a la espera que otra tarea realice alguna operación.

El uso de recursos introduce el problema de la **inversión de prioridad**. La inversión de prioridad consiste en que una tarea con una prioridad intermedia puede "colarse" a una tarea más prioritaria si ésta esta bloqueada a la espera de un recurso que tiene ocupado otra tarea de baja prioridad. En el siguiente



Inversión de prioridad

esquema se puede apreciar este efecto:

Los rectángulos negros representan intervalos de tiempo en los que la tarea tiene bloqueado el recurso. Desde el instante "c" hasta el "d" la tarea intermedia está ejecutándose (y por tanto retrasando a la tarea más prioritaria).

Es importante destacar que el bloqueo de una tarea más prioritaria por parte de otra menos prioritaria porque ésta tiene un recurso, no es el problema de inversión de prioridad.

Los métodos propuestos para solucionar este problema se pueden clasificar en tres grandes grupos:

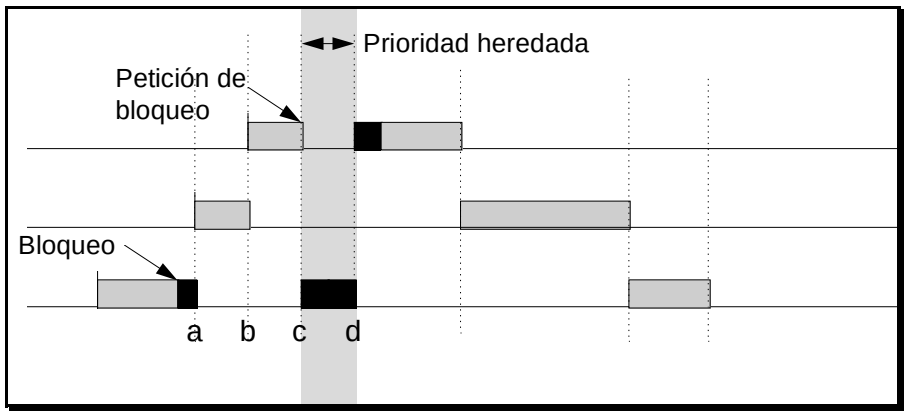
- EVITAR: Ordenar la ejecución de todas las tareas (off−line) con el fin de evitar que se produzcan estas situaciones. Esto se puede hacer fácilmente si tenemos un planificador cíclico.
- IGNORAR: Tratamos a todas las secciones críticas como bloques in−interrumpibles; y hacemos que todas las secciones críticas sean lo más breves posible.
- MINIMIZAR: Modificar la prioridad de las tareas que entren en una sección crítica, normalmente elevándoles la prioridad para que finalicen lo antes posible.
- Priority Inheritance Protocol (PIP) (herencia de prioridades)
- Priority Ceiling Protocol (PCP) (techo de prioridades)
- Semaphore Inheritance (PSP)

Los protocolos que tienen como objetivo minimizar la duración en la que se sufre inversión de prioridad.

Priority inheritance protocol

Cuando una tarea T_h intenta entrar en una sección crítica que está bloqueada (otra tarea T_l está dentro de ésta), la tarea bloqueante T_l hereda la prioridad de la tarea más prioritaria que quiere entrar Th. La

herencia de prioridad sólo se produce cuando una tarea menos prioritaria bloquea a una más prioritaria. La entrada en una sección crítica NO implica herencia de prioridad.



Herencia de prioridades

Las ventajas que tiene este algoritmo son:

- Una tarea de alta prioridad sólo puede ser bloqueada por una tarea de baja prioridad si ésta está dentro de una sección crítica.
- Una tarea menos prioritaria puede bloquear a otra más prioritaria como mucho durante una sola sección crítica.
- La sección crítica más larga determina el máximo tiempo que una tarea puede bloquear a otras de mayor prioridad.
- El tiempo que una tarea puede estar bloqueada está dado por: *min(n,m)*, donde "*n*" es el número de tareas de menor prioridad, y "*m*" es el número de regiones críticas (semáforos) usados por éstas.

Problemas:

- No evita el interbloqueo.
- Tiempo de bloqueo excesivo.

Priority ceiling protocol

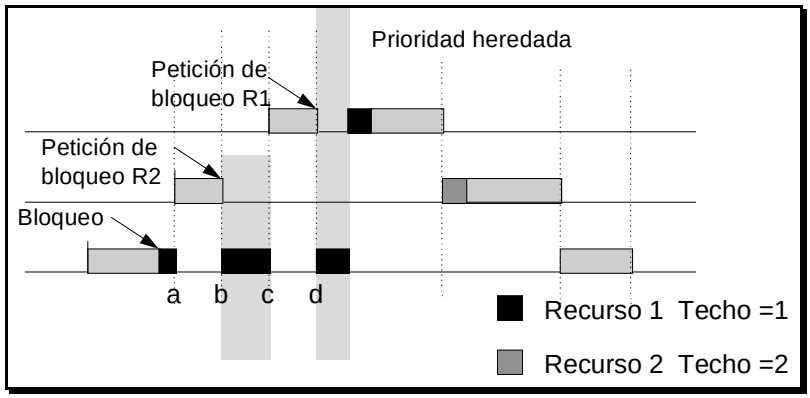
Está basado en la herencia de prioridades, pero tratando de subsanar los problemas del anterior. Las reglas de este protocolo son:

- A cada semáforo se le asigna una prioridad igual a la de la tarea más prioritaria que lo puede bloquear (techo de prioridad).
- Una tarea puede bloquear un recurso si su prioridad es estrictamente mayor que el techo de prioridad de todos los semáforos que en ese momento estén cerrados.
- Una tarea mantiene su prioridad mientras no bloquee a otra tarea más prioritaria, en cuyo caso hereda la prioridad máxima de entre las que bloquea.
- Al abandonar la región crítica recupera su prioridad inicial.

Este protocolo tiene las siguientes propiedades:

- Impide el interbloqueo.
- No existen bloqueos encadenados.
- El máximo tiempo de espera esta acotado por la sección crítica más larga de las tareas menos prioritarias.

Por otra parte es un algoritmo pesimista, en el sentido de que puede bloquear a tareas sin necesidad. Otro problema es su dificultad de implementación.



Ejemplo de funcionamiento del techo de prioridades.

Immediate ceiling priority inheritance

Es una versión simplificada del techo de prioridades original, pero que conservando el mismo tiempo de bloqueo máximo es más fácil de implementar en sistemas operativos reales. El protocolo es el siguiente:

- Cada proceso tiene una prioridad estática fija.
- Cada recurso tiene una prioridad llamado techo de prioridad fijo igual a la prioridad más alta de entre los procesos que lo utilizan.
- Un proceso tiene una prioridad dinámica igual al máximo entre su prioridad fija y el techo de todos los recursos que tiene bloqueados.

Este protocolo ha sido incluido en el estándar POSIX 1000.1c (PThreads) bajo el nombre de Priority Protected Protocol.

Planificabilidad con recursos

Un conjunto de "n" tareas será planificable bajo cualquier asignación de prioridades, empleando un protocolo de acceso a recursos compartidos, si se cumple la siguiente desigualdad:

$$\forall 1 \le i \le n, W_i \le D_i$$

Donde el valor de W_i es igual al anteriormente estudiado pero al que se le ha añadido una constante, B_i, que representa el tiempo de bloqueo debido al uso de recursos.

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

Este tiempo de bloqueo se obtiene de forma distinta en función del protocolo de gestión de recursos que se utilice.

El cálculo de Bi cuando se utiliza el protocolo de herencia de prioridades simple es el siguiente:

$B_i = \sum_{l=1}^{k} \left(usage(l,i) CS(k) \right)$ donde "k" es el número de secciones críticas diferentes. *usage* una función que se evalúa a uno si el recurso "*l*" es utilizado por el menos un proceso con una prioridad menor que la de "*i*" y otro con prioridad mayor o igual que "*i*". En otro caso el valor es cero. *CS(k)* es el máximo tiempo que un proceso puede estar dentro de la sección crítica "*k*".

Para el PCP, y para el techo de prioridad inmediato, B_i es igual a la sección crítica más larga de entre las que están guardadas por un semáforo cuyo techo de prioridad es mayor que la prioridad de la tarea i:

$B_i = max_{1 \le l \le n} \left(usage(l,i) CS(k) \right)$

9.Tareas Aperiódicas

Hasta ahora nos hemos centrado en el estudio de las tareas periódicas. Pasamos a ver la teoría de planificación para tareas aperiódicas. Nos centraremos en el estudio de tareas aperiódicas sin plazo de ejecución. El objetivo que se pretende es el conseguir el **menor tiempo de respuesta posible**, evidentemente sin que por ello se pierda ningún plazo de ejecución de ninguna tarea crítica (periódica).

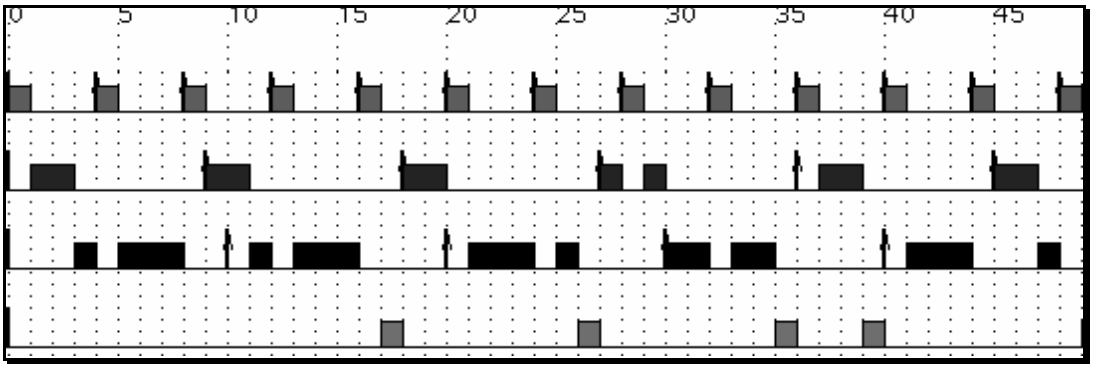
Podemos hacer la siguiente clasificación de algoritmos de planificación (sólo estudiaremos los marcados con un asterisco):

- Servidor en background (segundo plano) *
- Bandwidth preserving (conservación del tiempo de servicio)
  - Polling (por consulta) *
  - Priority exchange (intercambio de prioridades)
  - Deferrable server (servidor aplazable) *
- Sporadic server (servidor esporádico)
- Slack Stealing (extracción de holgura)
  - Estático
  - Dinámico

Background

Es el más sencillo de implementar. Cuando llega un petición aperiódica no es atendida inmediatamente, sino que se deja suspendida hasta que no haya activaciones de tareas periódicas pendientes de ejecutar, entonces se atiende a todas las peticiones de trabajo aperiódico que hay esperando. Básicamente consiste en sustituir el proceso idle del sistema por el servidor aperiódico. Otra forma de implementarlo consiste en añadir al sistema una tarea con periodo, plazo de ejecución y tiempo de cómputo infinitos y la prioridad más baja.

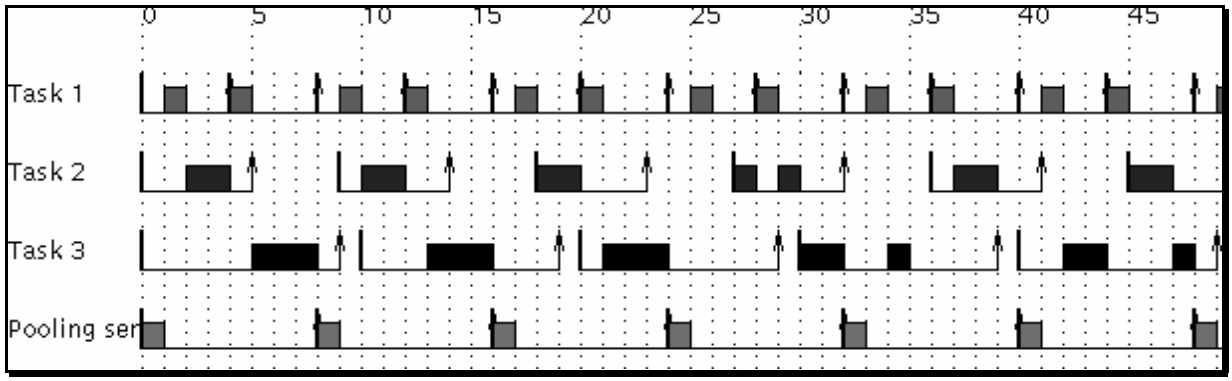
Este método si bien es muy sencillo de implementar, ofrece un mal tiempo de respuesta, concretamente ofrece el peor tiempo de respuesta posible de entre los planificadores que no dejan el procesador ocioso cuando hay trabajo que servir.



Mejor servicio posible con el servidor en background.

Polling

Se añade una nueva tarea periódica a la tareas críticas originales. Cada vez que se activa el servidor, consulta si hay trabajo aperiódico pendiente y lo sirve. Estará sirviendo trabajo aperiódico mientras le quede tiempo de cómputo o haya trabajo que servir. Cuando acaba de servir, **se suspende hasta que nuevamente se active en el siguiente periodo**. La tarea que realiza el pooling tiene asignados un periodo, un tiempo de cómputo máximo y una prioridad. Estos tres valores son arbitrarios, siempre y cuando el conjunto de todas las tareas sea planificable. La prioridad no se obtiene a partir del periodo, sino que se puede asignar la prioridad que más convenga. Se puede utilizar el segundo test de garantía para analizar la planificabilidad del conjunto de tareas.



El servidor por Pooling tiene la prioridad más alta, tiempo de computo unidad y periodo 8.

Se puede utilizar tanto con el RM como con el DM.

No es una cuestión trivial elegir el tiempo de cómputo y el periodo para el servidor aperiódico.

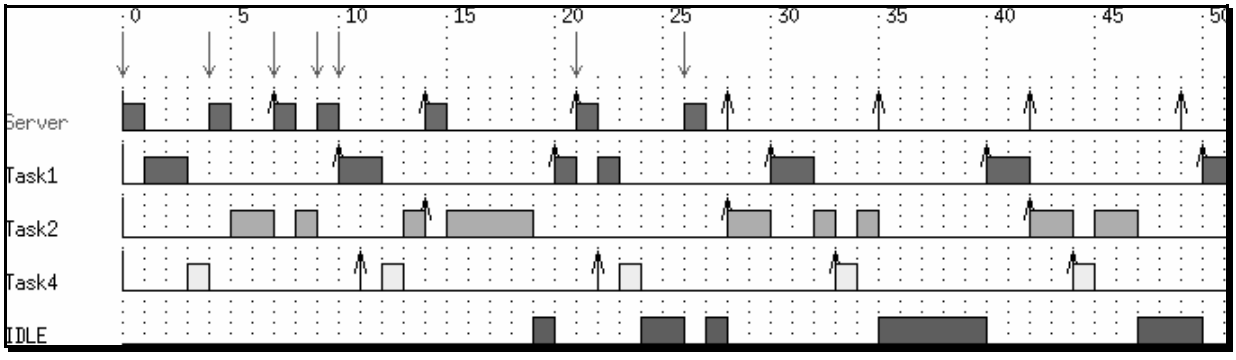
Deferrable server

Este servidor sólo puede ser empleado juntamente con el RM, pues supone que para todas las tareas $D_i=P_i$.

El servidor diferido se comporta como una tarea periódica de máxima prioridad:

- Mientras le quede tiempo de cómputo disponible, el servidor atenderá las peticiones periódicas inmediatamente (pues tiene la máxima prioridad).
- La capacidad del servidor se repone completamente al inicio de cada periodo.

Es importante hacer notar que a diferencia de el servidor por pooling, que sólo podía servir trabajo al inicio de cada periodo, el deferrable server puede ejecutar tareas aperiódicas en cualquier instante (siempre y cuando disponga de tiempo). Esta pequeña diferencia hace que no pueda ser considerada como una tarea "normal" a la hora de realizar el test de planificabilidad. Se ha tenido que desarrollar un test especifico para el deferrable server. El test nos dice cuál es el factor de utilización máximo que puede utilizar el servidor: $DS=\frac{C_{DS}}{P_{DS}}\leq \ln\frac{U+2}{2U+1}$



El servidor tiene dos unidades de tiempo de cómputo, y periodo 7.