

Planificación en Sistemas de Tiempo Real



Manuel Díaz

Dpto. Lenguajes y Ciencias de la Computación

Universidad de Málaga

GISUM

Planificación en Sistemas de Tiempo Real

- Introducción
- Planificación basada en Razón Monótona
- Planificación basada en Límites de Tiempo
- Interacción entre Procesos
- Un modelo de Procesos Extendido
- Planificadores. Esquemas de Implementación y Análisis.



Introducción

- **Planificación:** Asignación de recursos y tiempo a actividades de forma que se cumplan determinados requisitos de eficiencia.
- Los requisitos que guían la planificación dependen del sistema computacional:
 - Estáticos sin tiempo real: minimizar el tiempo de ejecución.
 - Dinámicos sin tiempo real: minimizar el tiempo de respuesta y maximizar el aprovechamiento de recursos.
 - Sistemas de tiempo real: **cumplir los límites temporales.**



Introducción

- Un método de planificación tiene dos aspectos:
 - Un **algoritmo de planificación**, que determina el orden de acceso de las tareas a los recursos.
 - Un **método de análisis** que permite calcular el comportamiento temporal del sistema:
 - Se pueden comprobar si los requisitos temporales están garantizados en **todos los casos posibles**.
 - En general se estudia el **peor comportamiento** posible.
 - Los algoritmos pueden ser estáticos y dinámicos.



Introducción

- Planificación Estática:
 - Los requisitos temporales se analizan antes de la ejecución y se determina el orden en que estas se ejecutarán.
 - Minimiza la sobrecarga en tiempo de ejecución.
 - Menos flexible.
- Planificación dinámica:
 - El orden de las tareas se decide durante la ejecución.
 - El análisis también se realiza durante la ejecución.



Introducción

- Nos centraremos en el método más utilizado actualmente: Prioridades Fijas y Procesos Totalmente Interrumpibles.
 - La prioridad es un parámetro relacionado con la urgencia o la importancia de la tarea.
 - Siempre se ejecuta la tarea lista con mayor prioridad.
- Es el algoritmo que mejor comportamiento tiene en sistemas de tiempo real crítico, sobre todo en casos de sobrecarga.

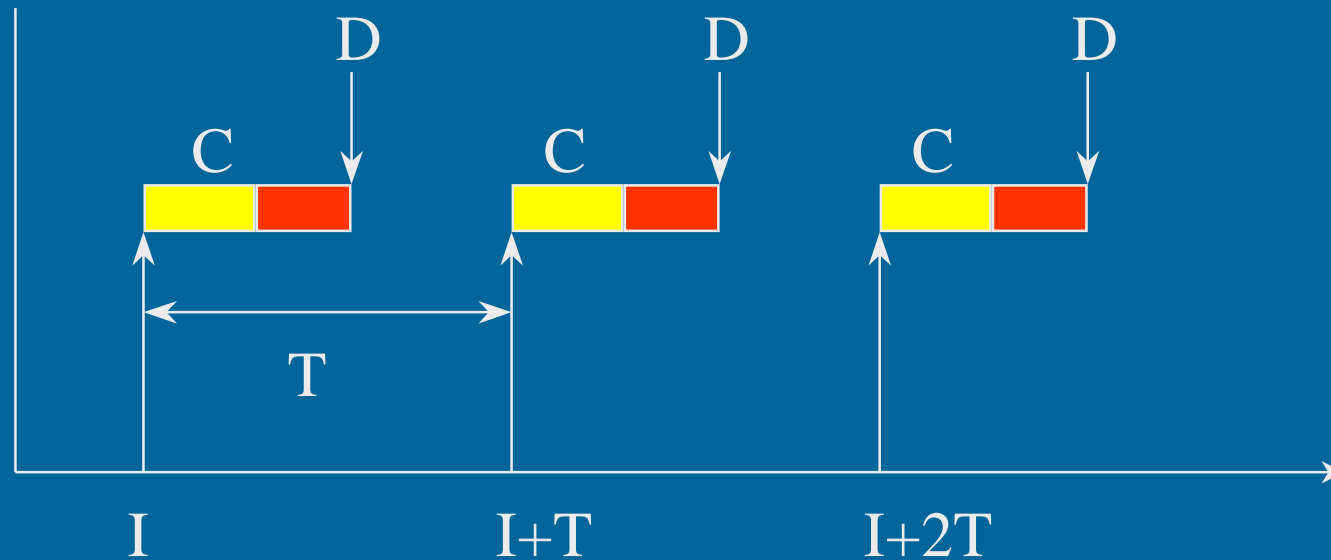


Modelo de Tareas

- Vamos a partir de un modelo simple que iremos completando.
- Características del modelo:
 - El conjunto de tareas es estático.
 - Todas las tareas son periódicas
 - Las tareas son independientes
 - Los límites de tiempo de todas las tareas son iguales a su periodo.
 - El tiempo de ejecución máximo es conocido.



Modelo de Tareas



I	Fase
T	Período
C	Peor Tiempo de Ejecución
D	Límite de Tiempo



Ejecutivos Cíclicos

- Si todas las tareas son periódicas, se puede confeccionar un plan de ejecución fijo.
- Los ejecutivos cíclicos son la aproximación más tradicional a la construcción de sistemas de tiempo real.
- No se utiliza la concurrencia. Las distintas tareas se simulan en un programa secuencial, sin soporte del lenguaje o sistema operativo.



Ejecutivos Cíclicos

- Se trata de un esquema que se repite cada:

$$T_m = \text{mcm}(T_i)$$

(ciclo principal)

- El ciclo principal se divide en ciclos secundarios, con período.

$$T_s \mid T_m = k \cdot T_s$$

- En cada ciclo secundario se ejecutan las actividades correspondientes a determinadas tareas.²¹

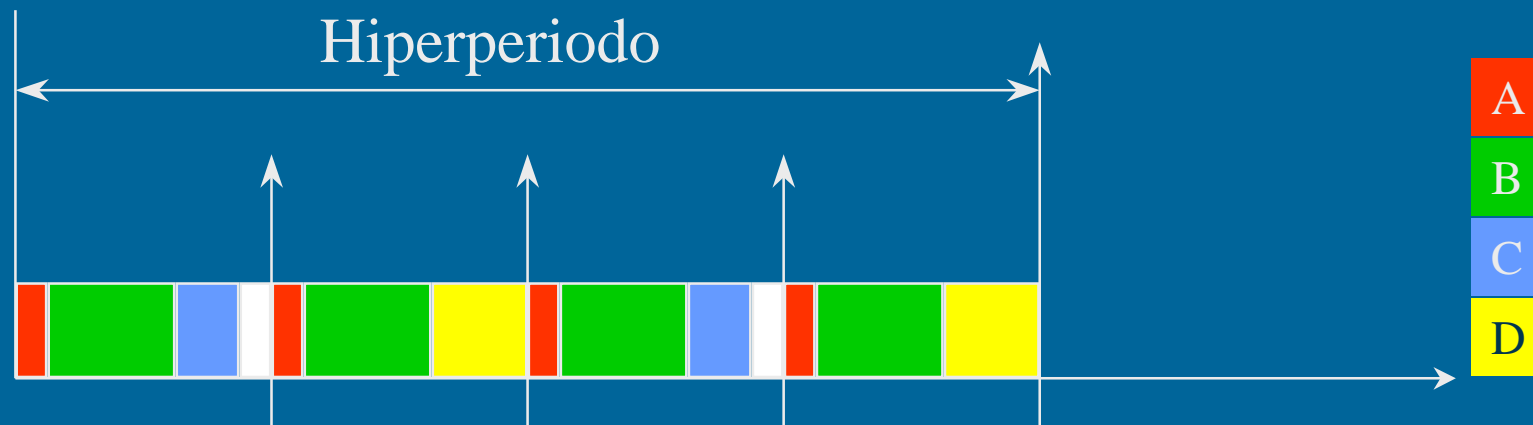


Ejecutivos Cíclicos

Tarea	T	C
A	4	0,5
B	4	2
C	8	1
D	16	3

- El ciclo principal dura 16ms
- Hay 4 ciclos secundarios
- El conjunto de tareas es planificable si

$$4C_A + 4C_B + 2C_C + C_D \leq 16$$



Ejecutivo Cíclico

```
void sistema()
{
  int ciclo=0;
  while (1) {
    Espera_Interrupcion_Timer; /* cada 4 ms*/
    switch ciclo {
      case 0: A;B;C; break;
      case 1: A;B;D1; break;
      case 2: A;B;C; break;
      case 3: A;B;D2; break;
    }
    ciclo= (ciclo +1) % 3;
  }
}
```



Ejecutivo Cíclico

- Propiedades:
 - No hay concurrencia en la ejecución
 - Los procedimientos/procesos pueden compartir datos (no hace falta exclusión mutua)
 - Los períodos deben ser armónicos
 - No hace falta analizar el comportamiento del sistema, es correcto por construcción



Ejecutivo Cíclico

- Problemas:
 - Las tareas aperiódicas son difíciles de incorporar
 - El plan cíclico es difícil de construir
 - Si los periodos son de diferentes ordenes de magnitud
 - Es necesario partir los procedimientos de forma artificial
 - En el caso general el NP-duro
 - Es poco flexible y difícil de mantener



Planificación basada en Razón Monótona

- Utilizamos la abstracción de la concurrencia
- Cada proceso tiene una prioridad fija
- Siempre se ejecuta la tarea de mayor prioridad
- Los procesos son totalmente interrumpibles
 - Si un proceso pasa al estado de *listo* y su prioridad es mayor que la del proceso en ejecución, este proceso es interrumpido por el planificador
- En principio vamos a suponer que el planificador no sobrecarga al sistema



Planificación basada en Razón Monótona

- El primer trabajo en el que se propone esta alternativa es el de Liu y Layland en 1973.

Scheduling Algorithms for Multiprogramming in a Hard Real-time

- El término **Razón Monótona** viene dado por la forma en la que se asignan las prioridades a las tareas:
- Las prioridades se asignan mediante una **función monótona de la razón de los procesos** (periódicos)
- En principio, supondremos también $D_i = T_i$.



Planificación basada en Razón Monótona

- **Teorema 1:** El mayor tiempo de respuesta para cualquier tarea T_i se da para la primera instancia de T_i cuando $I_1=I_2=...=I_n$.
 - El caso en que esto ocurre se denomina instante crítico
 - Representa el peor caso posible para la planificación, por lo que nos sirve para establecer un criterio de planificabilidad.



Planificación basada en Razón Monótona

- **Teorema 2:** Un conjunto de tareas es planificable por un determinado algoritmo si dicho algoritmo cumple los límites de tiempo para la primera instancia de cada tarea si todas las tareas comienzan en un instante crítico.
- **Algoritmo de Razón Monótona (ARM)**

$\text{prioridad}(P_i) > \text{prioridad}(P_j)$ si $T_i < T_j$



Planificación basada en Razón Monótona

- **Teorema 3.** ARM es óptimo.
 - Si algún algoritmo puede planificar un conjunto de tareas periódicas con las características consideradas, entonces dicho conjunto de tareas también se puede planificar mediante ARM.
 - Hay que hacer notar que el algoritmo sólo hace referencia a los períodos de las tareas y no a los tiempos de computación.



Planificación basada en Razón Monótona

- El siguiente paso es establecer un criterio para saber cuando un conjunto de tareas es planificable.
- **Definición 1.** Utilización del procesador

$$U = \sum_{i=1}^N \frac{C_i}{T_i}$$

- Es una medida de la carga del procesador para un conjunto de tareas
- En un sistema monoprocesador

$$U \leq 1$$



Planificación basada en Razón Monótona

- **Teorema 4.** Un conjunto de tareas periódicas P_1, \dots, P_n con $D_i = T_i$ es planificable por el algoritmo ARM si

$$U = \sum_{i=1}^N \frac{C_i}{T_i} \leq N \cdot \left| 2^{\frac{1}{N}} - 1 \right|$$

- Este valor nos da una cota de utilización del procesador por debajo de la cual cualquier conjunto de tareas es planificable



Planificación basada en Razón Monótona

- Es una condición suficiente, pero no necesaria.
- La cota de utilización es muy pesimista.

$$\lim_{N \rightarrow \infty} N \cdot \left| 2^{\frac{1}{N}} - 1 \right| = \log 2$$

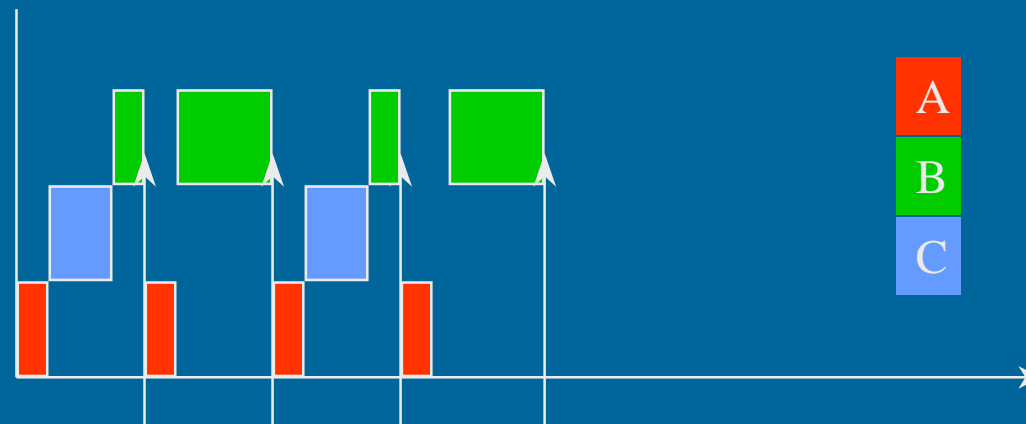
- Otros algoritmos, como primero el más urgente dan una cota muy superior de utilización.



Ejemplo

Tarea	T	C	P	U
A	4	1	3	0.25
B	8	2	2	0.25
C	16	8	1	0.5
				1

- No se cumple la prueba de utilización ($U > 0,779$)
- El conjunto de tareas es planificable



Análisis de Razón Monótona

- En la práctica, sin embargo, este 30% de diferencia en la utilización del procesador no es tan importante.
 - Los algoritmos dinámicos presentan más sobrecarga.
 - Empíricamente se comprueba que el algoritmo se comporta bien con utilizaciones cercanas al 90%.
 - El tiempo no aprovechado puede ser utilizado por otras tareas que no sean de tiempo real crítico.
 - El algoritmo se comporta mejor en situaciones de sobrecarga (prioridades fijas).



Análisis de Razón Monótona

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - Mejorar el modelo de tareas incluyendo:
 - Tareas Aperiódicas
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles)



Análisis de Razón Monótona

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - Mejorar el modelo de tareas incluyendo:
 - Tareas Aperiódicas
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles)



Análisis del Tiempo de Respuesta

- Los tests basados en utilización tienen dos fallos importantes :
 - No son exactos
 - No son aplicables a modelos de procesos más generales.
- Vamos a ver un nuevo tipo de test que se divide en dos fases :
 - Calcular analíticamente el peor tiempo de ejecución de cada proceso.
 - Comparar estos tiempos de ejecución con los límites de tiempo.



Análisis del Tiempo de Respuesta

- Para el proceso de mayor prioridad, su peor tiempo de respuesta será igual al tiempo de computación ($R=C$).
- Para el resto de los procesos este tiempo será mayor y vendrá dado por el tiempo de computación más la máxima interferencia (I) de los procesos de mayor prioridad.

$$R_i = C_i + I_i$$



Análisis del Tiempo de Respuesta

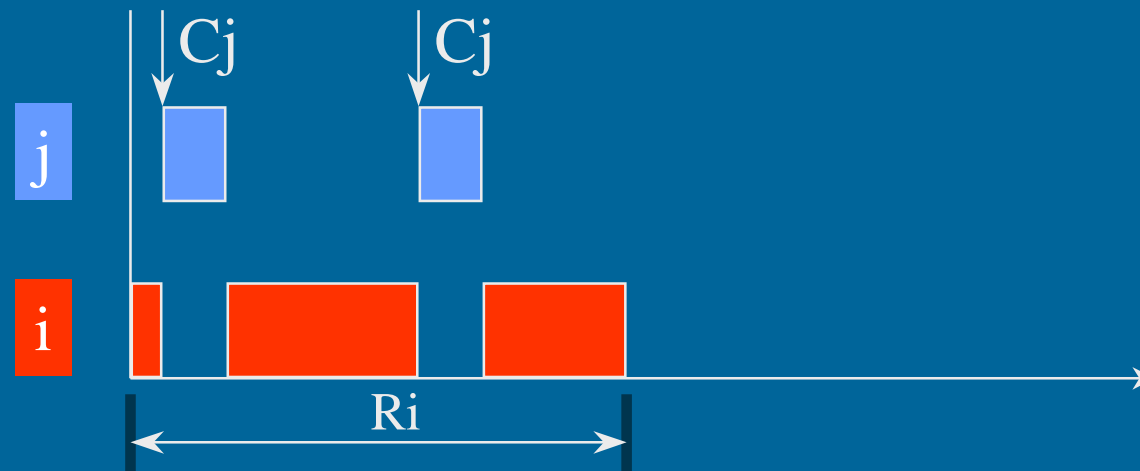
- La mayor interferencia ocurre si todos los procesos de mayor prioridad son planificados al mismo tiempo que el proceso i (instante crítico).
- Supondremos que todos los procesos empiezan su ejecución en el instante crítico, que consideramos 0.
- Sea un proceso j de mayor prioridad que i . En el intervalo $[0, R_i)$ dicho proceso tendrá que ejecutarse un cierto número de veces (dependiendo del periodo).



Análisis del Tiempo de Respuesta

- El número de veces que se ejecuta será:

$$NEjecuciones = \frac{R_i}{T_j}$$



Análisis del Tiempo de Respuesta

- Máxima interferencia del proceso j sobre i :

$$MaxInter = \frac{R_i}{T_j} \cdot C_j$$

- Esta interferencia ocurrirá para todos los procesos de mayor prioridad que i :

$$I_i = \sum_{j \in hp(i)} \frac{R_i}{T_j} \cdot C_j$$

$hp(i) = \{\text{procesos con mayor prioridad que } i\}$



Análisis del Tiempo de Respuesta

- Sustituyendo en la expresión del tiempo de respuesta:

$$R_i = C_i + \sum_{j \in hp(i)} \frac{R_j}{T_j} \cdot C_j$$

- Para calcular R_i tenemos que resolver la ecuación
 - Difícil por métodos analíticos (parte entera).
 - La podemos resolver por recurrencia, hasta alcanzar un punto fijo.



Análisis del Tiempo de Respuesta

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \frac{w_i^n}{T_j} \cdot C_j$$

- La ecuación tiene varias soluciones, de las cuales la menor representa el peor tiempo de respuesta del proceso i .
- La sucesión $w_i^0, w_i^1, \dots, w_i^n, \dots$ es **monótona no decreciente** con lo que cuando $w_i^0 = w_i^{n+1}$, encontramos la solución que buscamos.



Análisis del Tiempo de Respuesta

- Si la sucesión **no converge** podemos parar de calcular cuando $w_i^n > T_i$, ya que el proceso **no cumplirá el tiempo límite de ejecución**.
- De la misma forma **si converge** y el **valor obtenido es menor que el tiempo de ejecución** sabemos que la tarea es **planificable**.
- Para realizar los cálculos, el **valor inicial** se puede tomar como C_i , ya que sabemos que el peor tiempo de ejecución siempre será mayor.



Análisis del Tiempo de Respuesta

Tarea	T	C	P
T1	7	3	3
T2	12	3	2
T3	20	5	1

- La tarea T1 es la más prioritaria, luego ninguna tarea la interfiere

$$R_1 = w_1^0 = C_1 = 3$$

- Tarea T2:

$$w_2^0 = C_2 = 3$$

$$w_2^1 = 3 + \frac{3}{7} \cdot 3 = 6$$

$$w_2^2 = w_2^1 = 6$$

$$R_2 = 6$$



Análisis del Tiempo de Respuesta

- Tarea T3:

$$w_3^0 = C_3 = 3$$

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

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

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

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

$$w_3^5 = w_3^4 = 6, R_3 = 20$$



Análisis de Razón Monótona

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - **Mejorar el modelo de tareas incluyendo:**
 - **Tareas Aperiódicas**
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles)



Análisis de Razón Monótona

Tareas Aperiódicas

- Las tareas aperiódicas pueden ser:
 - **Críticas**
 - tienen un plazo de respuesta que no se puede incumplir.
 - normalmente se especifica una separación mínima entre dos activaciones consecutivas
 - **Acríticas**
 - se pueden responder tarde sin problemas graves
 - se caracterizan por un esquema de activación estocástico



Análisis de Razón Monótona

Tareas Aperiódicas

- La aproximación más simple consiste en convertir las tareas aperiódicas en periódicas: **Muestreo o Polling**.
- Una tarea periódica se encarga de comprobar si un determinado evento aperiódico y lo trata. Una vez ha finalizado no hace nada hasta el próximo periodo.
- El tiempo de ejecución de la tarea periódica será el tiempo que se tarda en tratar el evento y el periodo el intervalo de muestreo.



Análisis de Razón Monótona

Tareas Aperiódicas

- Sin embargo existen dos problemas:
 - Si ocurren muchos eventos durante un periodo de muestreo el tiempo de ejecución de la tarea puede variar mucho.
 - Si un evento ocurre inmediatamente después de un muestreo no es tratado hasta el siguiente periodo.
- La primera solución propuesta a estos problemas fue la del **Servidor Aperiódico**.
- Se trata de una tarea conceptual (puede no existir como tal) a la que se le asigna un tiempo de ejecución **supuesto** y un periodo de **reposición**.



Análisis de Razón Monótona

Tareas Aperiódicas

- El servidor atenderá tareas aperiódicas durante su tiempo de ejecución asignado y con la prioridad dada por el algoritmo de razón monótona en función del periodo que se le asigna.
- Si se siguen produciendo eventos aperiódicos después del hueco reservado al servidor, éste seguirá atendiendo los eventos pero a una prioridad más baja que la de todas las tareas periódicas.



Planificación basada en Límites de Tiempo

- Hasta ahora hemos supuesto que las tareas tenían un límite de tiempo de ejecución igual a su periodo.
- Esto es restrictivo y a veces no es aplicable, especialmente en el caso de las tareas aperiódicas, ya que el límite de tiempo suele ser mucho menor que el determinado por la máxima frecuencia con la que la tarea puede activarse.



Planificación basada en Límites de Tiempo

- Una forma de resolver estos problemas es mediante la utilización de un nuevo algoritmo para asignar las prioridades, con un fundamento similar al ARM.

Asignación de prioridades monótona en límites de tiempo o
Deadline Monotonic Scheduling

- Según esta formulación las prioridades se asignan a los procesos en función de los **límites de tiempo** (deadlines) en lugar de en función de sus periodos, es decir:

$$D_i < D_j \quad P_i > P_j$$



Planificación basada en Límites de Tiempo

- Se puede demostrar fácilmente que este algoritmo para signación de prioridades es también óptimo.
- Además el test basado en el peor tiempo de ejecución sigue siendo valido, salvo que en lugar del periodo de las tareas hay que tener en cuenta su deadline

$$R_i < D_i$$

- Sirve para todos los procesos (periódicos y aperiódicos).



Planificación basada en Límites de Tiempo

- El siguiente ejemplo es planificable con prioridades monótonas en límite de tiempo y no lo es con prioridades monótonas en frecuencia.

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

Tarea	T	D	C	P	R
T3	10	10	4	4	4
T2	15	7	3	3	7
T1	20	5	3	2	10
T4	20	20	3	1	20



Análisis de Razón Monótona

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - **Mejorar el modelo de tareas incluyendo:**
 - Tareas Aperiódicas
 - **Comunicación y Sincronización entre tareas**
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles)



Interacción entre Procesos

- Uno de los inconvenientes más importantes de los dos modelos anteriores es la suposición de que las tareas no suspenden durante su ejecución (es decir no se comunican, ni se sincronizan).
- Si eliminamos esta suposición y seguimos aplicando el algoritmo de la misma forma aparece uno de los problemas más importantes dentro de la planificación de procesos de tiempo real:

La Inversión de Prioridades.



Interacción entre Procesos

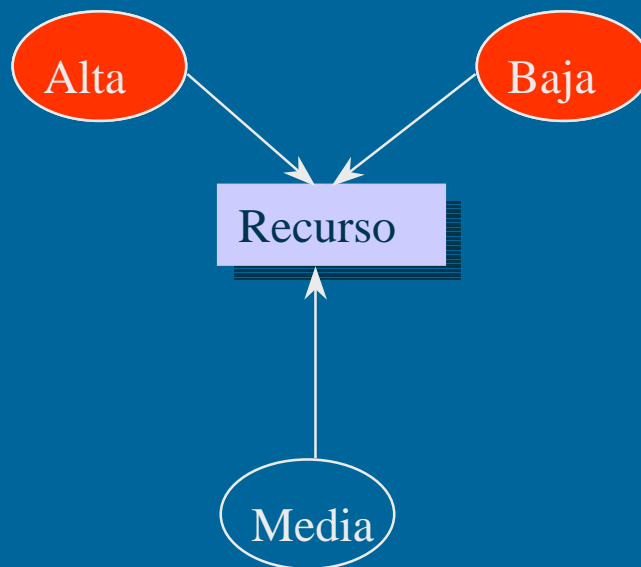
Inversión de Prioridades

- La inversión de prioridades aparece cuando una tarea de mayor prioridad debe esperar por un recurso que está siendo utilizado por otra tarea menos prioritaria.
- No se puede eliminar totalmente, pero es posible limitar su duración.
- Con los algoritmos empleados hasta ahora la **duración** puede ser **ilimitada**.



Interacción entre Procesos

Inversión de Prioridades



- La tarea de baja prioridad empieza a ejecutarse y bloquea el semáforo.
- La tarea de prioridad alta interrumpe a la tarea de prioridad baja en la mitad de la sección crítica e intenta acceder al mismo recurso quedandose bloqueada.
- Mientras que la tarea de alta prioridad está esperando a que termine la tarea de baja prioridad la tarea de prioridad media la puede interrumpir tantas veces como quiera.



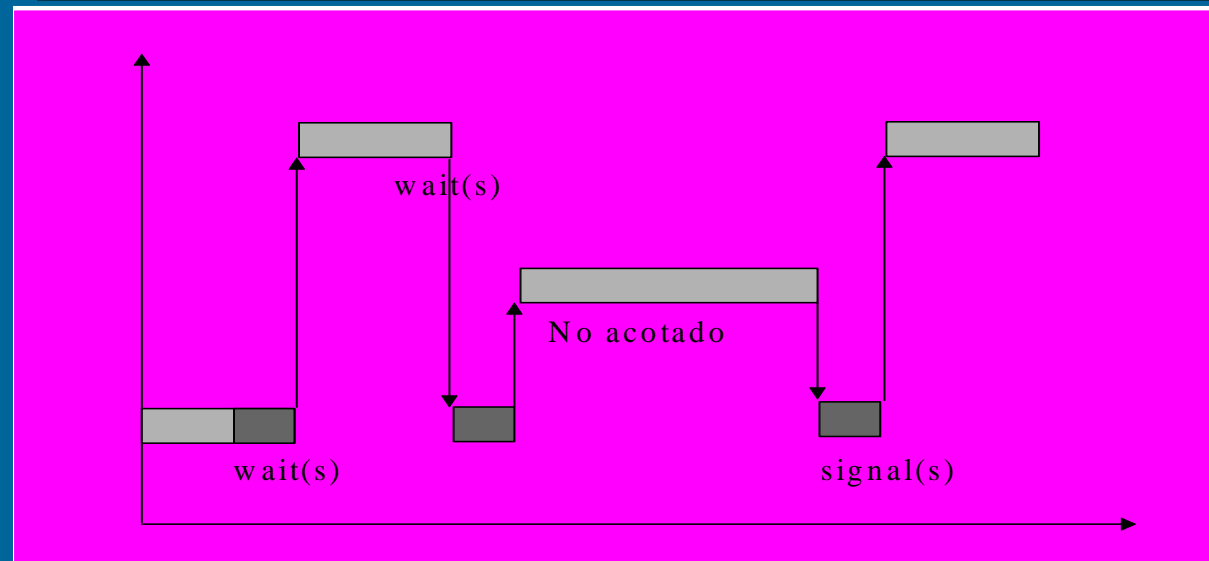
Interacción entre Procesos

Inversión de Prioridades

```
process alta;  
begin  
.....  
wait(S) ;  
(*usar recurso*)  
signal(S) ;  
.....  
end;
```

```
process media;  
begin  
.....  
(*tiempo ilimitado*)  
.....  
end ;
```

```
process baja;  
begin  
.....  
wait(s);  
(*usar recurso*)  
signal(S) ;  
.....  
end;
```



Interacción entre Procesos

Herencia de Prioridad

- Cuando una tarea T bloquea a otra tarea de mayor prioridad, T se ejecuta con la mayor de las prioridades de las tareas que está bloqueando en ese momento.
- Cuando la tarea finaliza la ejecución de su sección crítica recupera su prioridad inicial.
- La herencia de prioridades es transitiva, es decir si $T1$, $T2$ y $T3$ son tres tareas en orden descendente de prioridad, entonces si $T3$ bloquea a $T2$ y $T2$ bloquea a $T1$, $T3$ hereda la prioridad de $T1$.



Interacción entre Procesos

Herencia de Prioridad

- Con este protocolo un proceso puede bloquear como máximo:
 - una vez por cada sección crítica
 - una vez por cada tarea de prioridad inferior
- La duración total máxima de los bloqueos es:

$$B_i = \sum_{j \in lp(i), k=1}^K \text{uso}(k, i) \cdot CS_{j,k}$$

- K es el número de recursos del sistema.
- uso(k,i) indica si el proceso i utiliza el recurso k.
- CS_{j,k} es el tiempo de ejecución de las regiones críticas de j cuando accede el recurso k.



Interacción entre Procesos

Herencia de Prioridad

- Este algoritmo presenta sin embargo dos problemas:
 - El tiempo de bloqueo de una tarea de alta prioridad, aunque está acotado puede ser muy largo si se produce una cadena de accesos a semáforos.
 - No evita los interbloqueos

```
process A;          process B;
begin              begin
.....            ....
wait(S1);          wait(S2);
wait(S2);          wait(S1);
(*usar recurso*)  (*usar recurso*)
signal(S2);        signal(S1);
signal(S1);        signal(S2);
...                ...
end;                end;
```

- $Pri(A) > Pri(B)$
- B empieza a ejecutarse y bloquea s1
- A comienza su ejecución y bloquea s2
- A intenta bloquear s1 y suspende
- B continua su ejecución y hereda $pri(A)$
- B bloquea s2 y produce un interbloqueo



Interacción entre Procesos

Techos de Prioridad

- El protocolo de techo de prioridad soluciona los dos problemas anteriores.
 - La idea de este protocolo es asegurar que cuando una tarea interrumpe una sección crítica y ejecuta otra, la prioridad a la que se ejecuta esta nueva sección crítica será mayor que la de todas las prioridades heredadas por todas las secciones críticas interrumpidas.
 - Se asigna un límite de prioridad a cada recurso, que es igual a la prioridad del proceso con mayor prioridad que puede utilizar el recurso (Techo de Prioridad).



Interacción entre Procesos

Techos de Prioridad

- El protocolo consiste en:
 - Cuando una tarea intenta ejecutar una de sus secciones críticas, ésta suspenderá a menos que su prioridad sea mayor que el techo de prioridad de todos los recursos bloqueados en ese momento.
 - Si la tarea no puede acceder a la sección crítica, la tarea que tiene el recurso con mayor límite de prioridad bloqueado, hereda la prioridad de dicha tarea.



Interacción entre Procesos

Techos de Prioridad

Tarea	T	C	D	P
A	50	10	5	3
B	500	500	250	2
C	3000	3000	1000	1

Semáforo	Techo Pr.
s1	3
s2	2
s3	2

```

process alta;
begin
.....
wait(S1);
.....
signal(S);
.....
end;

```

```

process media;
begin
.....
wait(s2);
wait(s3);
.....
signal(s3);
signal(s2);
.....
end;

```

```

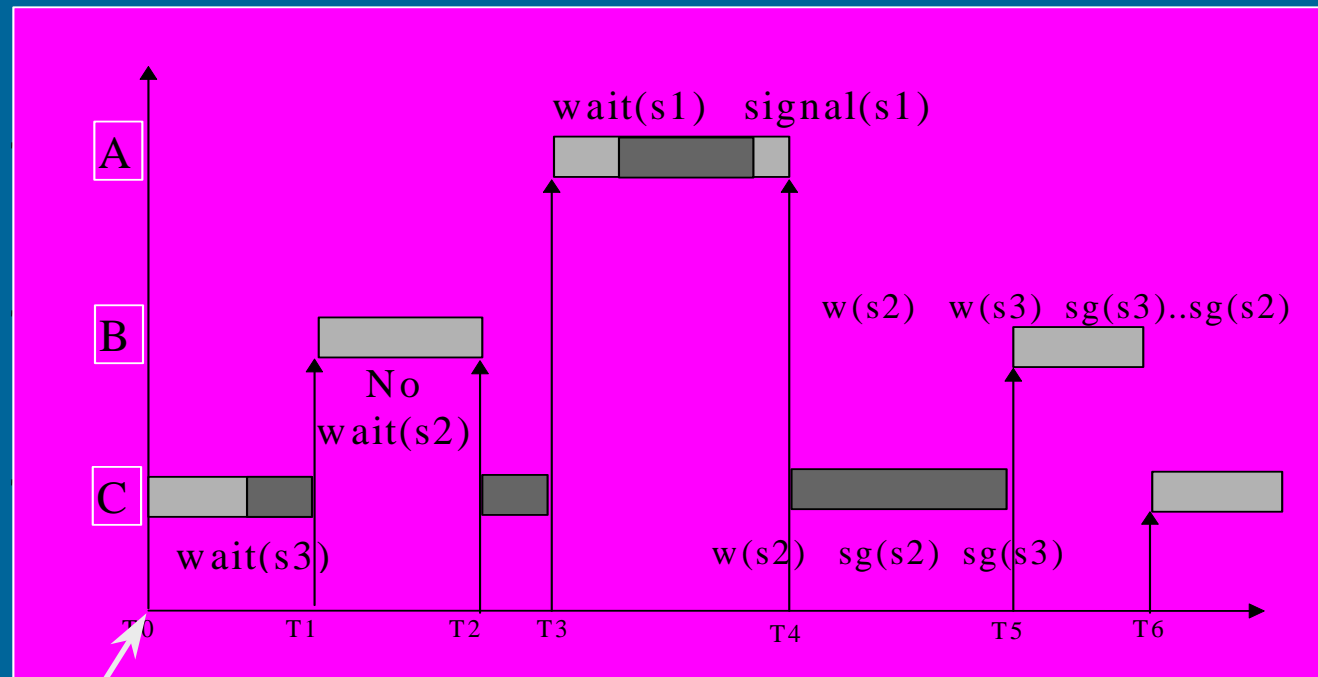
process baja;
begin
.....
wait(s3);
wait(s2);
.....
signal(s2);
signal(s3);
.....
end;

```



Interacción entre Procesos

Techos de Prioridad

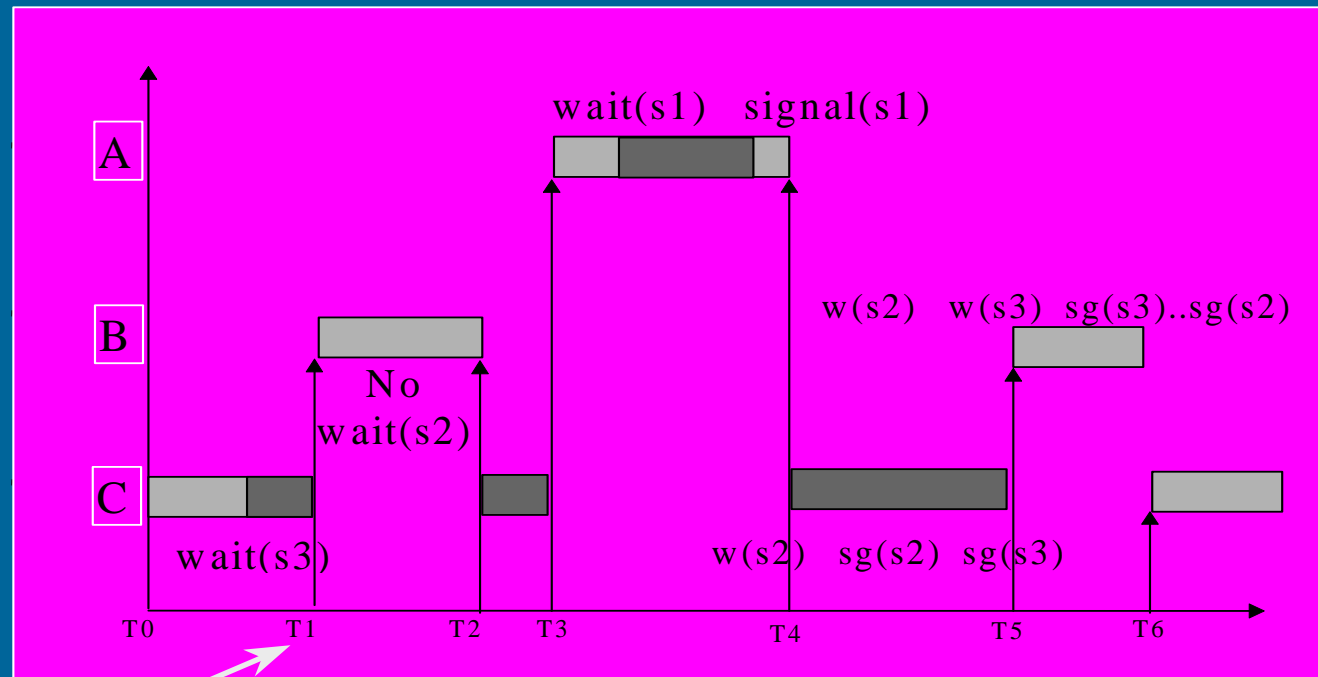


T0. C C comienza su ejecución e intenta bloquear s3, como la prioridad de C es mayor que los techos de los semáforos bloqueados por otros procesos (no hay ninguno) consigue bloquear s3



Interacción entre Procesos

Techos de Prioridad

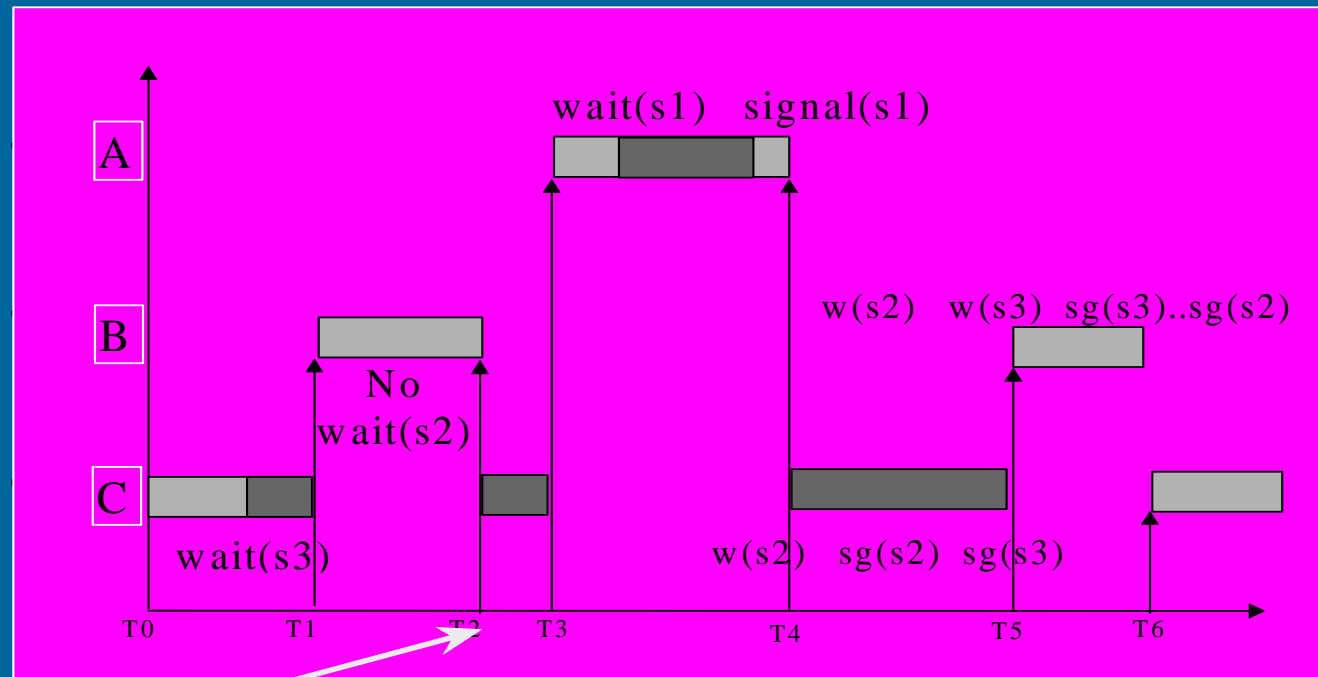


T1. B comienza su ejecución y como tiene mayor prioridad interrumpe a C



Interacción entre Procesos

Techos de Prioridad

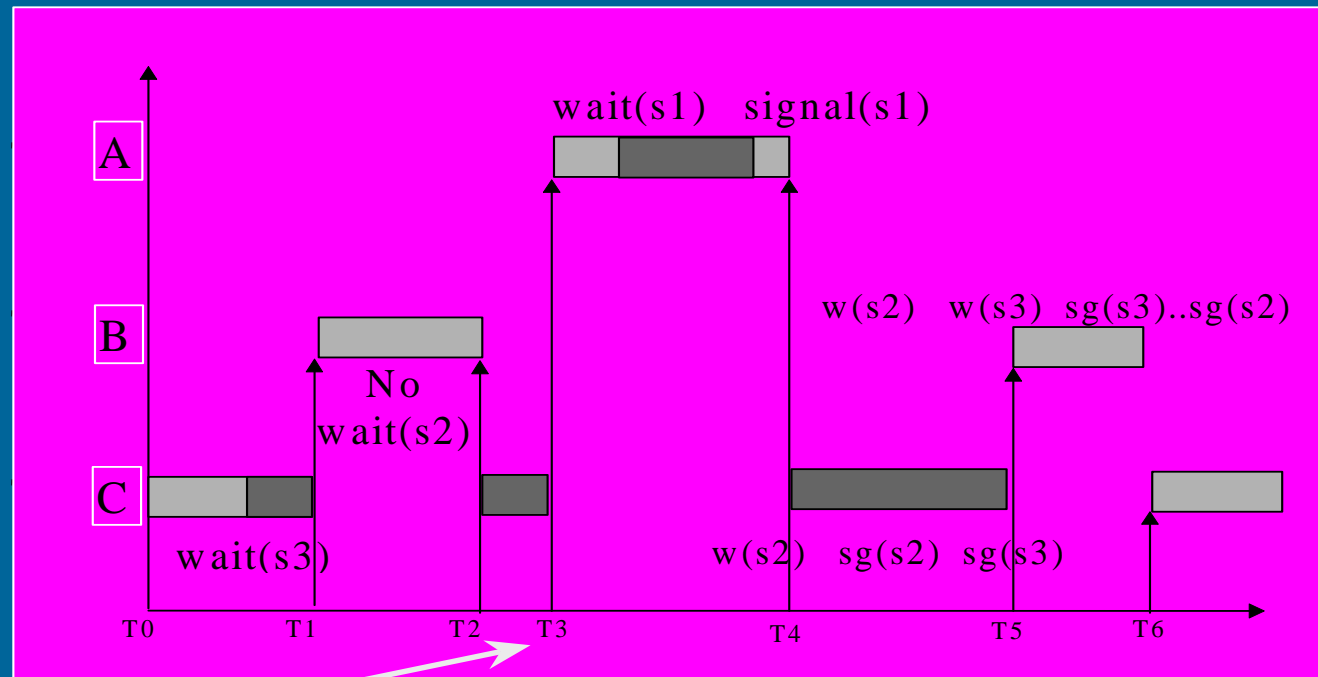


T2. B intenta bloquear s2. Hay un semáforo bloqueado por otra tarea con techo 2 (s3). Como la prioridad de B no es mayor que dicho techo bloquea sin acceder al semáforo y C hereda su prioridad



Interacción entre Procesos

Techos de Prioridad

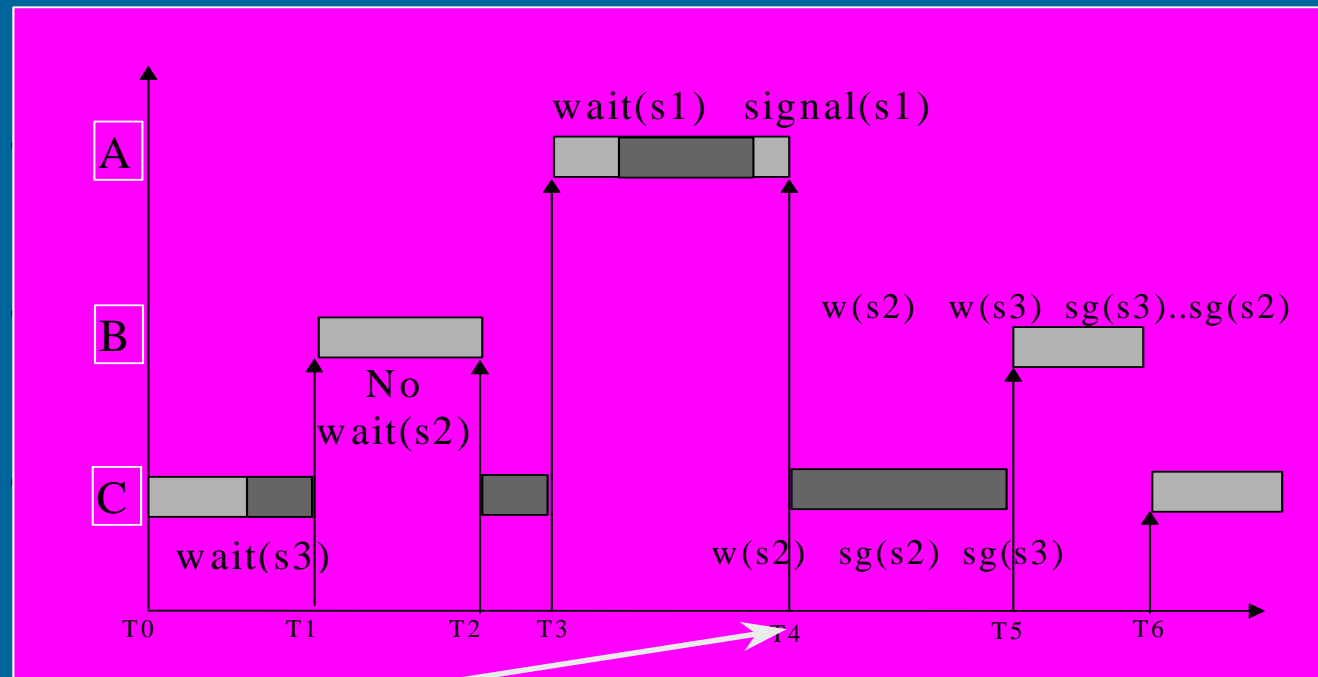


T_3 . A comienza su ejecución e interrumpe a C (tiene mayor prioridad que la que C ha heredado de B). Después A intenta bloquear s_1 y lo consigue ya que el único semáforo bloqueado en el sistema es s_3 y tiene techo 2



Interacción entre Procesos

Techos de Prioridad

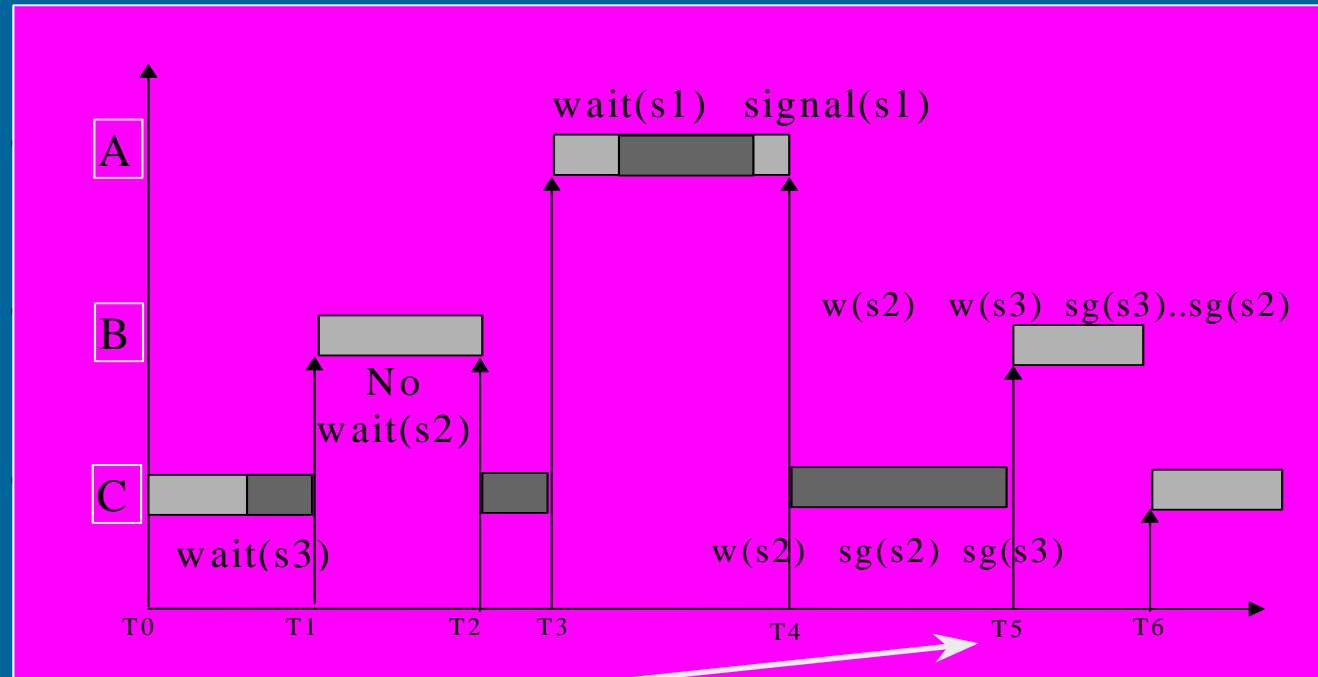


T4. A acaba su ejecución tras hacer un signal de s1. C continua (B está bloqueada) e intenta hacer `wait(s2)`. La prioridad de C es mayor que el techo de los semáforos bloqueados por otras tareas (no hay) y por tanto lo consigue



Interacción entre Procesos

Techos de Prioridad

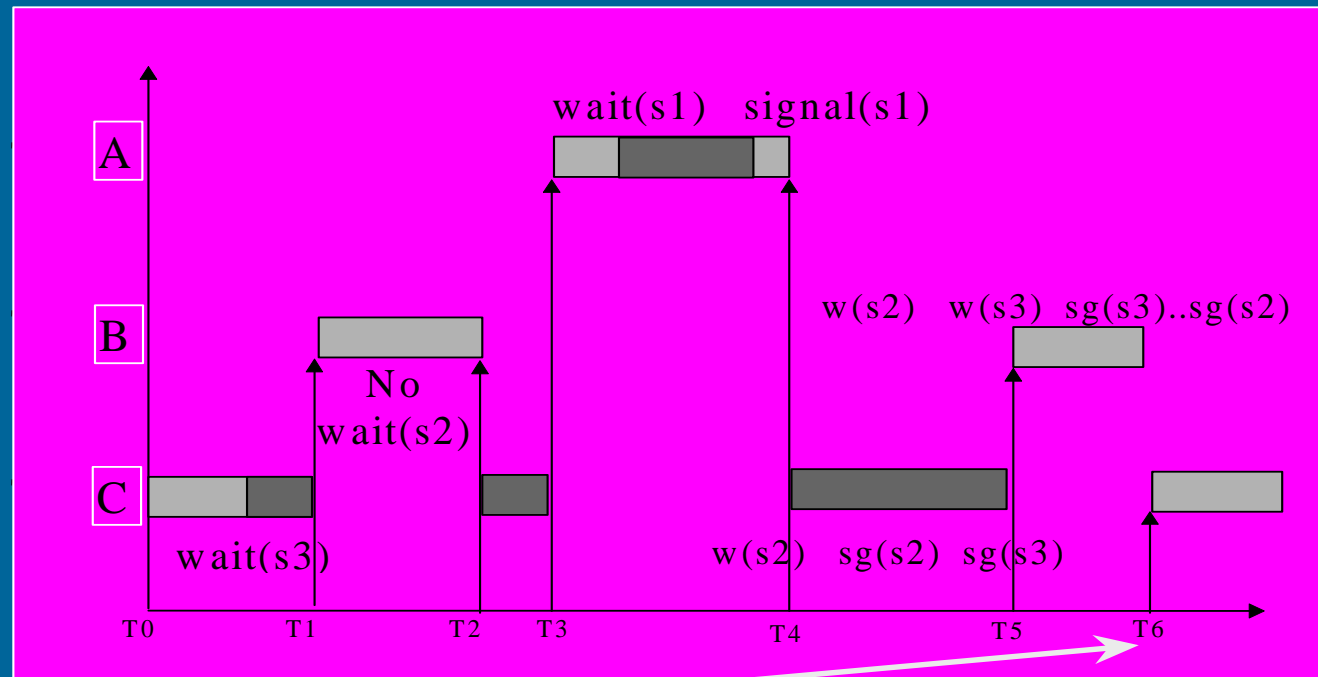


T5 C hace signal de s3 y s2 y en ese momento es interrumpida por B (pierde la prioridad heredada). B se ejecuta bloqueando los semáforos (ya no se lo impide ningún techo) y finaliza su ejecución



Interacción entre Procesos

Techos de Prioridad



T6 C acaba su ejecución



Interacción entre Procesos

Techos de Prioridad

- Utilizando este algoritmo el tiempo que una tarea de menor prioridad puede bloquear a otra de mayor prioridad viene dado por:

$$B_i = \max_{\forall k \in lp(i), \forall r \in usa(k) | techo(r) \geq pri(i)} C_{k,r}$$

- Seleccionamos los recursos que pueden bloquear los proceso de menor prioridad que i
- De entre estos seleccionamos aquellos cuyo techo sea mayor que la prioridad de i
- Para los procesos seleccionados calculamos el peor tiempo de sus secciones críticas ($CS_{i,j}$) y nos quedamos con el máximo



Interacción entre Procesos

Techos de Prioridad

- Análisis de planificabilidad:
 - Factor de utilización. Test de Razón Monótona Extendido

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} \leq 1 \cdot \left| 2^{\frac{1}{1}} - 1 \right|$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} \leq 2 \cdot \left(2^{\frac{1}{2}} - 1 \right)$$

.....

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \dots + \frac{C_k}{T_k} + \frac{B_k}{T_k} \leq k \cdot \left(2^{\frac{1}{k}} - 1 \right)$$

.....

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \dots + \frac{C_n}{T_n} + \frac{B_n}{T_n} \leq n \cdot \left(2^{\frac{1}{n}} - 1 \right)$$



Interacción entre Procesos

Techos de Prioridad

- Análisis de planificabilidad:
 - Tiempo de Respuesta

$$R_i = C_i + B_i + I_i$$

$$B_i = \max_{\forall k \in p(i), \forall r \in \text{usa}(k) | \text{techo}(r) \geq \text{pri}(i)} C_{k,r}$$

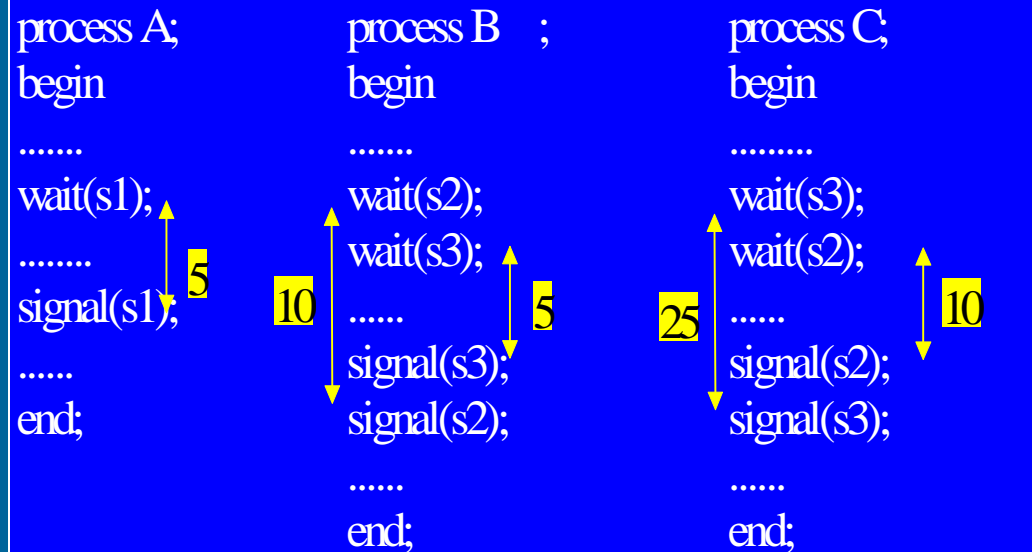
$$I_i = \sum_{j \in hp(i)} \frac{R_j}{T_j} \cdot C_j$$



Interacción entre Procesos

Techos de Prioridad

Para calcular BA necesitamos tener en cuenta las tareas de menor prioridad (B y C) y los semáforos que éstas usan (s2 y s3)

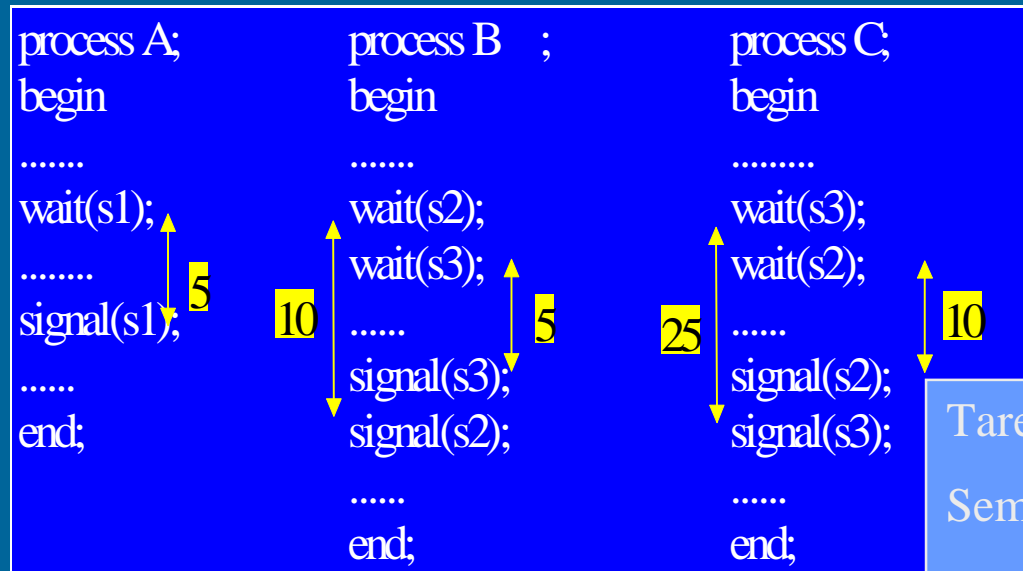


Estos semáforos tienen un techo menor que la prioridad de A, luego nunca pueden bloquear a A ($BA = 0$)



Interacción entre Procesos

Techos de Prioridad



Tareas de menor prioridad = { C }

Semáforos = { s2, s3 }

Sem. con techo > pri(B) = { s2, s3 }

Máximo Tiempo Bloqueo (s2) = 10ms

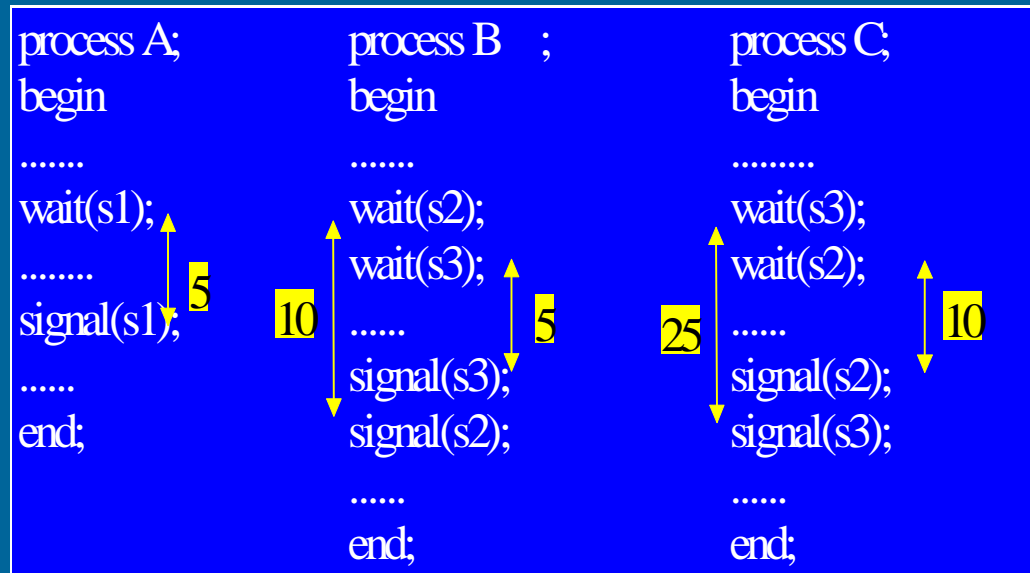
Máximo Tiempo Bloqueo (s3) = 25ms

BB=25ms



Interacción entre Procesos

Techos de Prioridad



Para Bc no hay tareas de menor prioridad, con lo que el tiempo de bloqueo a tener en cuenta es nulo



Interacción entre Procesos

Techos de Prioridad

- **Ventajas:**
 - Cada tarea se puede bloquear una vez como máximo en cada ciclo
 - No puede haber interbloqueos
 - Se consigue la exclusión mutua sin mecanismos de protección adicionales.
- **Inconvenientes:**
 - Complejidad
 - Sobrecarga del planificador



Interacción entre Procesos

Techo de Prioridad Inmediato

- Con este protocolo, una tarea que accede a un recurso hereda inmediatamente el techo de prioridad del recurso.
- Las propiedades son las mismas
- Es más fácil de implementar que el protocolo básico
- Es más eficiente (menos cambios de contexto).



Modelo Extendido de Procesos

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - Mejorar el modelo de tareas incluyendo:
 - Tareas Aperiódicas
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles,...)



Protocolo de Cambio de Modo

- Objetivos:
 - **Compatibilidad**
 - **Mantenibilidad**: debe permitir añadir nuevas tareas sin afectar a las ya existentes.
 - Las tareas deben cumplir sus límites antes, durante y después del protocolo
 - **Prestaciones**: el protocolo debe ser tan eficiente como los protocolos de cambio de modo para los ejecutivos cíclicos.



Protocolo de Cambio de Modo

- El incorporar o borrar nuevas tareas puede llevar al cambio de los techos de prioridad de los recursos:
 - Si es necesario aumentar el techo de un recurso no bloqueado, se modifica directamente y de forma indivisible
 - Si el recurso está bloqueado hay que esperar a que sea desbloqueado
 - Si es necesario bajar el techo de un recurso, hay que esperar a que todas las tareas que lo comparten y que tengan prioridades mayores sean eliminadas
 - Si la prioridad de una tarea es mayor que los techos de los recursos bloqueados que ella también puede bloquear, los techos deben ser elevados antes de añadir la nueva tarea



Protocolo de Cambio de Modo

- Una tarea puede ser eliminada en un hiperperiodo si aun no ha empezado su ejecución, en caso contrario hay que esperar al siguiente.
- El tiempo de procesador libre puede ser utilizado por otras tareas inmediatamente (en el mismo hiperperiodo).
- Una tarea puede ser añadida solo si queda capacidad de proceso disponible.
- El retraso del protocolo está acotado por el mayor de dos términos:
 - El mayor periodo de todas las tareas que tienen que ser eliminadas.
 - El menor periodo asociado al recurso con el menor límite de prioridad que tiene que ser modificado.



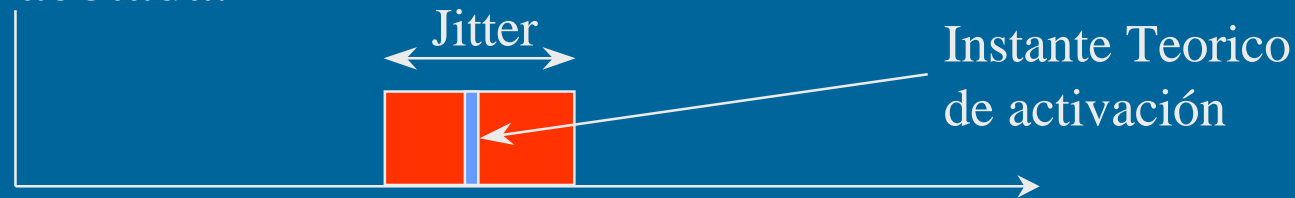
Modelo Extendido de Procesos

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - Mejorar el modelo de tareas incluyendo:
 - Tareas Aperiódicas
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - **Variación en el esquema de activación**
 - Sobrecargas del planificador
 - Otras limitaciones del entorno de ejecución (procesos no totalmente interrumpibles,...)



Variación en la Activación (Jitter)

- Hasta ahora hemos supuesto que una tarea empezaba su ejecución en el instante exacto asignado por el algoritmo de planificación.
 - Normalmente existen variaciones debidas a como se implementan los algoritmos de planificación (manejo de las colas de prioridad,...).
 - A esta variación la denominamos **Jitter** y supondremos que está acotada.



Variación en la Activación (Jitter)

- Esta variación puede ser tomada en cuenta al calcular el peor tiempo de respuesta.
- Si J_i es la variación máxima para el proceso i , la ecuación del tiempo de respuesta sería:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \frac{R_j + J_j}{T_j} \cdot C_j$$



Sobrecarga por Cambios de Contexto

- En el modelo hemos supuesto que los cambios de contexto son despreciables.
- En algunos procesadores, un cambio de contexto puede ser bastante costoso y es necesario tenerlo en cuenta.
- Cada interrupción de una tarea por otra conlleva dos cambios de contexto:
 - Uno para introducir la tarea de mayor prioridad
 - Otro para devolver la tarea de menor prioridad a ejecución cuando la de mayor prioridad acabe.



Sobrecarga por Cambios de Contexto

- Si conocemos el tiempo necesario para un cambio de contexto (C_{sw}), la ecuación del tiempo de respuesta puede ser modificada:

$$R_i = C_i + 2 \cdot C_{sw} + B_i + \sum_{j \in hp(i)} \frac{R_j + J_j}{T_j} \cdot (C_j + 2 \cdot C_{sw})$$

- Hemos incluido el tiempo del cambio de contexto en el tiempo de computo de cada tarea.



Modelo Extendido de Procesos

- Para que el algoritmo sea realmente útil necesitamos:
 - Calcular con más precisión la utilización o el tiempo de respuesta de las tareas.
 - Mejorar el modelo de tareas incluyendo:
 - Tareas Aperiódicas
 - Comunicación y Sincronización entre tareas
 - Cambios de prioridades durante la ejecución (cambios de modo)
 - Variación en el esquema de activación
 - Sobrecargas del planificador
 - **Otras limitaciones del entorno de ejecución** (procesos no totalmente interrumpibles,...)



Modelo de Procesos Extendido

Procesos no Totalmente Interrumpibles

- Otra de las suposiciones que hemos hecho y que en ocasiones no se cumple totalmente es la de que los procesos pueden ser interrumpidos en cualquier instante.
- Normalmente, incluso en los entornos de ejecución definidos como totalmente interrumpibles (preemptive), existen bloques que no pueden ser interrumpidos (por ejemplo un cambio de contexto).
- Esto puede ser soventado incluyendo en el factor de bloqueo B_i los bloques no interrumpibles



Planificadores

Esquemas de Implementación y Análisis

- Vamos a ver dos esquemas de implementación de planificadores con las características expuestas en el modelo:
 - Planificación basada en eventos (Event Scheduling)
 - Planificación dirigida por ticks (Tick driven Scheduling)
- Para ambos esquemas veremos como incorporar en el cálculo del tiempo de respuesta las distintas sobrecargas del planificador.



Planificación basada en Eventos

- El planificador dispone de dos colas de procesos:
 - Cola de ejecución, ordenada por prioridades
 - Cola de tiempos, ordenada por los tiempos en los que las tareas tienen que ser despertadas
- Suponemos que existe una llamada en el sistema, **delay_until**, que hace que el planificador pase una tarea de la cola de ejecución a la cola de retrasos.



Planificación basada en Eventos

- El funcionamiento del planificador es el siguiente:
 - Cuando el planificador **suspende a una tarea en la cola de tiempos**, debe programar una **interrupción de un timer** con el menor tiempo de los procesos en la cola de tiempos.
 - Cuando la **interrupción ocurre**, todas las tareas en la cola de tiempos con tiempos menores o iguales al momento de la interrupción son cambiadas a la cola de ejecución.
 - El planificador programa la **próxima interrupción**, con el menor tiempo de los procesos de la cola de tiempos.
 - El planificador **ejecuta la tarea de mayor prioridad** de la cola de ejecución.



Planificación basada en Eventos

Análisis de Tiempo de Respuesta

- Como hemos visto, a sobrecarga del planificador puede tenerse en cuenta incluyéndola en el factor B_i del cálculo del tiempo de respuesta.
- Sin embargo, así no tenemos en cuenta el tiempo de proceso de las interrupciones del timer.
- Una forma simple de tener en cuenta este tiempo es suponer que existe una **tarea ficticia** por cada tarea real del sistema.



Planificación basada en Eventos

Análisis de Tiempo de Respuesta

- La tarea ficticia tendrá las siguientes características:
 - Muy alta prioridad
 - El mismo periodo que la tarea real que modela
 - El peor tiempo de ejecución correspondiente al peor tiempo del manejador de interrupciones del timer.
- Con estas consideraciones el tiempo de respuesta sería:

$$R_i = C_i + 2 \cdot C_{sw} + B_i + \sum_{j \in hp(i)} \frac{R_j + J_j}{T_j} \cdot (C_j + 2 \cdot C_{sw}) + \sum_{j \in todas_las_tareas} \left\lceil \frac{R_j + J_j}{T_j} \right\rceil \cdot C_{timer}$$



Planificación basada en Eventos

Análisis de Tiempo de Respuesta

- Existen algunos problemas con este tipo de planificación:
 - El hardware del timer debe ser bastante sofisticado (el timer tiene que poder guardar tiempos absolutos largos)
 - En la mayoría de los sistemas los timers no son así y pueden desbordarse
 - Los desbordamientos tienen que ser detectados
 - Normalmente el desbordamiento se notifica mediante una interrupción, que hay que manejar, incrementando la sobrecarga.
 - El tiempo del próximo evento puede ser menor que el tiempo requerido para pasar a la nueva tarea y la interrupción se puede perder.



Planificación Dirigida por Ticks

- Es una forma más simple de implementar un planificador con prioridades fijas, pero menos eficiente.
- Al igual que en el caso anterior suponemos una primitiva **delay_until** que los procesos utilizan para indicar un retraso al planificador.
- También como en el caso anterior, el planificador siempre mantiene en ejecución la tarea con mayor prioridad.



Planificación Dirigida por Ticks

- La interrupción del timer en este caso se programa para que se produzca en intervalos regulares, con un periodo que llamaremos **Tick**.
- Cada vez que el timer testea la interrupción comprueba si existe alguna tarea con el tiempo expirado en la cola de tiempos.
- Elimina los problemas anteriores siempre que el planificador tenga un tiempo máximo de ejecución menor que **Tick**.



Planificación Dirigida por Ticks

- El análisis de tiempos efectuado para el planificador anterior no es valido:
 - El planificador comprueba si la tarea está lista cada periodo
 - Una tarea se puede ver retrasada un periodo de muestreo completo.
- ¿Cómo podemos modelar este retraso?
 - Lo podemos modelar como una variación en el esquema de activación (Jitter).
 - Las tareas pueden sufrir un retraso en su activación entre 0 y Tick.



Planificación Dirigida por Ticks

- También tenemos que tener en cuenta el tiempo consumido por el planificador en cada interrupción del timer.
- La primera opción para incluir el tiempo sería:
 - Suponer el planificador una tarea periódica de máxima prioridad.
 - Calcular el peor tiempo de ejecución del planificador
 - Ocurriría cuando tuvieramos que mover todas las tareas de una cola a la otra.



Planificación Dirigida por Ticks

- Es una visión muy pesimista, ya que esto ocurre muy raras veces.
- Otra opción:
 - Crear un conjunto de tareas ficticias, como en el caso anterior, que representen el coste asociado a la manipulación de las colas.
 - Crear una sola tarea periódica con periodo Tick para modelar el coste del manejo del interrupción del timer.



Planificación Dirigida por Ticks

- El tiempo de Jitter hay que añadirlo al que ya pudieran tener las tareas por otras causas.
- La expresión resultante sería:

$$R_i = C_i + 2 \cdot C_{sw} + B_i + \sum_{j \in hp(i)} \frac{R_j + J_j + Tick}{T_j} \cdot (C_j + 2 \cdot C_{sw}) + \sum_{j \in todas_las_tareas} \left\lceil \frac{R_j + J_j + Tick}{T_j} \right\rceil \cdot C_{cola} + \left\lceil \frac{R_i}{Tick} \right\rceil \cdot C_{Tick}$$



Comparación

- El segundo modelo es mucho más simple.
- No requiere hardware “especial”.
- No hay que tener en cuenta los problemas de granularidad y de desbordamiento del timer.
- Sin embargo, es mucho más ineficiente
 - Introduce retrasos en todas las tareas
 - El retraso se puede minimizar si se escoge adecuadamente el valor de Tick
 - Si Tick es muy largo el valor del Jitter puede ser muy alto.
 - Si Tick es muy pequeño la sobrecarga por el manejo de la interrupción puede ser muy alta.



Conclusiones

- Hemos visto un conjunto de herramientas que nos pueden ayudar a diseñar sistemas con restricciones de tiempo.
- El tiempo no es el único factor a tener en cuenta en el diseño, pero en ocasiones puede dominar totalmente las características de este.
- ¿Donde se enmarcan las técnicas que hemos visto dentro del ciclo de vida del desarrollo de un sistema?



Conclusiones

- Lo ideal sería incorporar el análisis temporal lo antes posible en el diseño, a partir de los requisitos temporales de la especificación. Problemas:
 - El análisis es dependiente del entorno de ejecución:
 - Hardware, que normalmente se diseña al mismo tiempo
 - Sistema Operativo o entorno de ejecución
 -
 - No es posible conocer algunos de los parámetros necesarios para el análisis (peor tiempo de ejecución, recursos compartidos, prioridades,...)



Conclusiones

- Siempre se puede utilizar para **analizar a posteriori** las características temporales del sistema una vez construido
 - Los errores encontrados pueden llevar a cambios profundos en el diseño del sistema, e incluso en el entorno de ejecución o el hardware.
- Como mínimo, hay que intentar que el sistema cumpla los requisitos del modelo para poder analizarlo.



Conclusiones

- Lo ideal es incorporar los requisitos del modelo al diseño desde la primeras fases y utilizar estimaciones de los tiempos de ejecución siempre que sea posible.
- Para estimar los tiempos de ejecución hay dos alternativas:
 - Construir un modelo estadístico
 - Desarrollar prototipos a partir de los cuales se puedan realizar estimaciones
- También es necesario conocer:
 - Las secciones críticas
 - Tiempo de ejecución máximo
 - Quien las comparte



Conclusiones

- Herramientas:
 - TimeWiz
 - MIMOSA
 - Scheduler 1-2-3 (CMU), Software Engineer Notebook (CMU más reciente).
- Algunas ayudan a tomar decisiones de diseño, analizando distintas soluciones posibles y sugiriendo las más apropiadas.

