dit
UPM

Planificación de tareas

Juan Antonio de la Puente
DIT/UPM

Objetivos

- Veremos cómo **planificar** el uso de los recursos para poder **garantizar** los **requisitos temporales**
- Un **método de planificación** tiene dos aspectos importantes:
  - Un **algoritmo de planificación** que determina el orden de acceso de las tareas a los recursos del sistema (en particular al procesador)
  - Un **método de análisis** que permite calcular el **comportamiento temporal** del sistema
    - » Así se pude comprobar si los requisitos temporales están **garantizados** en todos los casos posibles
    - » En general se estudia el **peor comportamiento** posible

Índice

Planificación de tareas

- Un método de planificación puede ser
  - **estático**: el análisis se puede hacer antes de la ejecución
  - **dinámico**: el análisis se hace durante la ejecución
- Un método estático muy utilizado es el de **planificación con prioridades fijas y desalojo**
  - La prioridad es un parámetro relacionado con la *urgencia* o la *importancia* de la tarea
  - En cada momento se ejecuta la tarea con mayor prioridad de todas las ejecutables

Modelo de tareas

Inicialmente consideraremos un **modelo simple**:

- El conjunto de tareas es **estático**
- Todas las tareas son **periódicas**
- Las tareas son **independientes** unas de otras
- Los **plazos** de respuesta de todas las tareas son iguales a los **períodos** respectivos
- El **tiempo de ejecución máximo** de cada tarea es conocido
- Las operaciones del núcleo de multiprogramación son **instantáneas**

Parámetros de planificación

N	Número de tareas
T	Período de activación
C	Tiempo de ejecución máximo
D	Plazo de respuesta
R	Tiempo de respuesta máximo
P	Prioridad

En el modelo anterior, para todas las tareas τ_i :

$$C_i \leq D_i = T_i$$

Se trata de asegurar que

$$R_i \leq D_i$$

Hiperperíodo

- En el modelo de tareas simple, el valor

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

  se denomina **hiperperíodo** del sistema

  - El comportamiento temporal se repite cada hiperperíodo

Índice

Planificación cíclica

- Si todas las tareas son periódicas, se puede confeccionar un plan de ejecución fijo
- 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

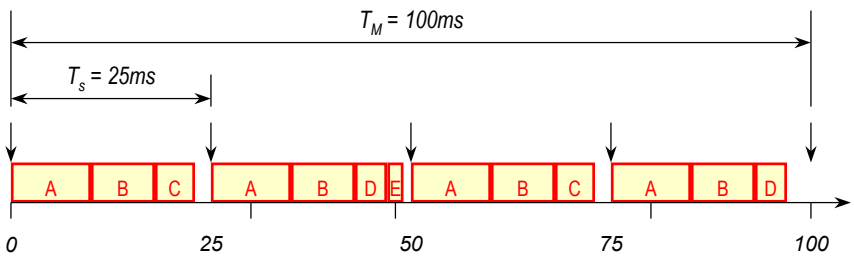
$$T_S \qquad (T_M = k \cdot T_S)$$

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

Ejemplo

Tarea	T	C
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

- El sistema es armónico
- El ciclo principal dura 100ms
- Se compone de 4 ciclos secundarios de 25ms cada uno



Ejecutivo cíclico

```
procedure Cyclic_Executive is
   type Cycle is mod 4;
   Frame : Cycle := 0;
begin
   loop
      Wait_for_Interrupt;
      case Frame is
        when 0 => A; B; C;
        when 1 => A; B; D; E;
        when 2 => A; B; C;
        when 3 => A; B; D;
      end case;
      Frame := Frame + 1;
   end loop;
end Cyclic_Executive;
```


Propiedades

- **No hay concurrencia** en la ejecución
  - Cada ciclo secundario es una secuencia de invocaciones de procedimientos
- Los procedimientos pueden **compartir datos**
  - No hace falta usar mecanismos de exclusión mutua
- Los períodos deben ser **armónicos**
  - Puede ser necesario utilizar períodos más cortos de lo necesario
- **No hace falta analizar** el comportamiento temporal
  - El sistema es correcto por construcción

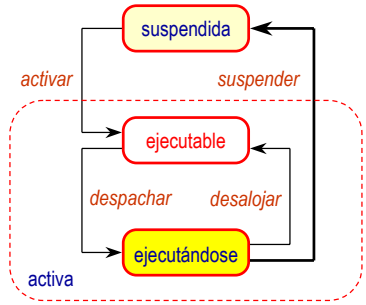
Problemas

- Las **tareas esporádicas** son difíciles de tratar
  - Se puede utilizar un *servidor de consulta*
- El plan cíclico es **difícil de construir**
  - Si los períodos son de diferentes órdenes de magnitud el número de ciclos secundarios se hace muy grande
  - Puede ser necesario partir una tarea en varios procedimientos
  - En el caso más general es NP-duro
- Es **poco flexible** y difícil de mantener
  - Cada vez que se cambia una tarea hay que rehacer toda la planificación

Índice

Planificación con prioridades fijas

- Las tareas se realizan como **procesos concurrentes**
- Una tarea puede estar en varios **estados**
- Las tareas ejecutables se **despachan** para su ejecución en orden de **prioridad**
- El despacho puede hacerse
  - **con desalojo**
  - **sin desalojo**
- En general supondremos **prioridades fijas con desalojo**



Prioridades monótonas en frecuencia

- La asignación de mayor prioridad a las tareas de menor período *(rate monotonic scheduling)* es **óptima** para el modelo de tareas simple (tareas periódicas, independientes, con plazos iguales a los períodos)

Si se pueden garantizar los plazos de un sistema de tareas con otra asignación de prioridades, se pueden garantizar con la asignación monótona en frecuencia

Factor de utilización

- La cantidad

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

es el *factor de utilización del procesador*

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

$$U \leq 1$$

Condición de garantía de los plazos basada en la utilización

- Para el modelo simple, con prioridades monótonas en frecuencia, los plazos están garantizados si

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

- La cantidad

$$U_0(N) = N \cdot \left(2^{1/N} - 1\right)$$

es la **utilización mínima garantizada** para N tareas

Utilización mínima garantizada

N	U0
1	1,000
2	0,828
3	0,779
4	0,756
5	0,743

$$\lim_{n\to\infty} U_0(N) = \log 2 \approx 0{,}693$$

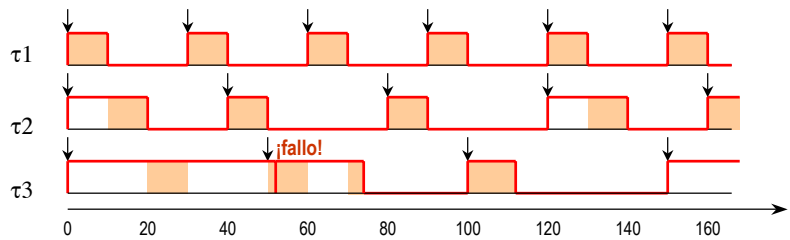
Ejemplo 1

Tarea	T	C	P	U
τ1	30	10	3	0,333
τ2	40	10	2	0,250
τ3	50	12	1	0,240
				0,823

El sistema no cumple la prueba de utilización

(U > 0,779)

La tarea 3 falla en *t* = 50



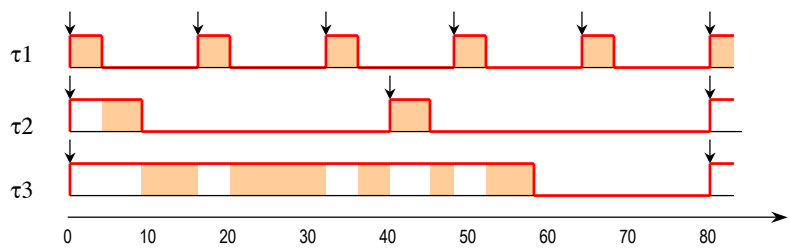
Instante crítico

- El cronograma se puede utilizar para comprobar si se cumplen los plazos
  - Hay que trazar el cronograma durante un hiperperíodo completo
  - En el caso más desfavorable, $H = O(N^N)$
- El tiempo de respuesta es máximo cuando todas las tareas se activan a la vez
  Se denomina **instante crítico**
- Si el instante inicial es crítico basta comprobar el **primer ciclo** de cada tarea

Ejemplo 2

Tarea	T	C	P	U
τ1	16	4	3	0,250
τ2	40	5	2	0,125
τ3	80	32	1	0,400
				0,775

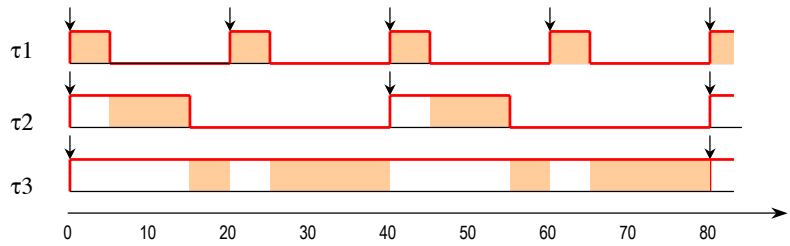
Este sistema está garantizado
(U < 0,779)



Ejemplo 3

Tarea	T	C	P	U
τ1	20	5	3	0,250
τ2	40	10	2	0,250
τ3	80	40	1	0,500
				1,000

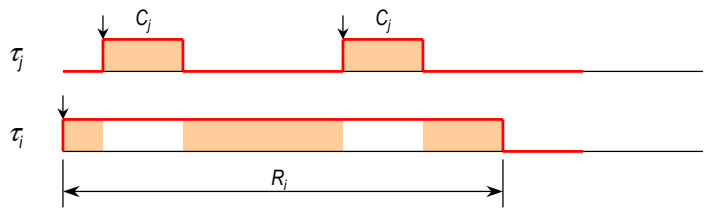
Este sistema no pasa la prueba ($U > 0{,}779$), pero se cumplen los plazos



Análisis del tiempo de respuesta

- La prueba del factor de utilización no es exacta, ni se puede generalizar a modelos de tareas más complejos
- La construcción de un cronograma es compleja, incluso considerando que el instante inicial es crítico
- Veremos una prueba basada en el cálculo del tiempo de respuesta de cada tarea

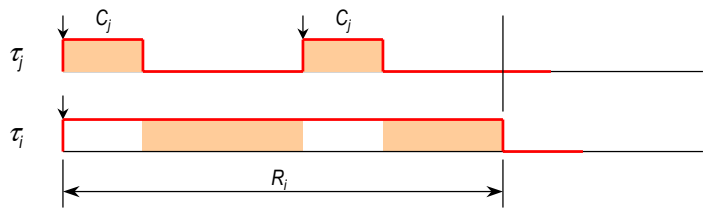
Ecuación del tiempo de respuesta



$$R_i = C_i + I_i$$

El tiempo de respuesta de una tarea es la suma de su tiempo de cómputo más la interferencia que sufre por la ejecución de tareas más prioritarias

Cálculo de la interferencia máxima



Para una tarea de prioridad superior

$$I_i^j = \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Para todas las tareas de prioridad superior

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

Cálculo del tiempo de respuesta

La ecuación del tiempo de respuesta queda así:

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

R_i es la solución mínima de la ecuación

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

- La ecuación no es continua ni lineal
- No se puede resolver analíticamente

Iteración lineal

La ecuación del tiempo de respuesta se puede resolver mediante la relación de recurrencia

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

Un valor inicial aceptable es

$$w_i^0 = C_i + \sum_{j \in hp(i)} C_j$$

Se termina cuando

a) $w^{n+1} = w^n$, o bien

b) $w^{n+1} > T_i$ (no se cumple el plazo)

Ejemplo 4

Tarea	T	C	P	R
τ1	7	3	3	3
τ2	12	3	2	6
τ3	20	5	1	20

$$\tau_1: \quad w_1^0 = 3;$$

$$\tau_2: \quad w_2^0 = 3 + 3 = 6;$$

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

$$\tau_3: w_3^0 = 5 + 3 + 3 = 11;$$

$$w_3^1 = 5 + \left\lceil \frac{11}{7} \right\rceil \cdot 3 + \left\lceil \frac{11}{12} \right\rceil \cdot 3 = 14;$$

$$w_3^2 = 5 + \left\lceil \frac{14}{7} \right\rceil \cdot 3 + \left\lceil \frac{14}{12} \right\rceil \cdot 3 = 17;$$

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

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

Todas las tareas tienen sus plazos garantizados

Tenemos una condición **suficiente** y **necesaria**

Tiempo de cómputo

Hay dos formas de obtener el valor de *C* para una tarea:

- **Medida del tiempo de ejecución**
  - no es fácil saber cuándo se mide el tiempo máximo
- **Análisis del código ejecutable**
  - » se descompone el código en un grafo de bloques secuenciales
  - » se calcula el tiempo de ejecución de cada bloque
  - » se busca el camino más largo
  - la estimación de *C* puede ser muy pesimista
  - es difícil tener en cuenta los efectos de los dispositivos de hardware (*caches, pipelines,* etc..)
  - se puede mejorar con análisis estático

Restricciones en el código

- Para poder calcular el tiempo de cómputo hay que evitar utilizar estructuras con tiempo de cómputo no acotado, como:
  - bucles no acotados
  - recursión no acotada
  - objetos dinámicos
  - tareas dinámicas

Índice

Tareas aperiódicas

- Las tareas aperiódicas pueden ser
  - **críticas *(hard)*** o **esporádicas**
    - » tienen un plazo de respuesta crítico (no puede fallar)
    - » se caracterizan por una separación mínima entre dos sucesos de activación consecutivos
  - **acríticas *(soft)***
    - » pueden responder tarde ocasionalmente
    - » se caracterizan por un esquema de activación estocástico (por ejemplo, Poisson)
- Deben garantizarse
  - los plazos de todas las tareas en condiciones “normales”
  - los plazos de las tareas críticas en las peores condiciones

Tareas esporádicas

- Para incluir tareas esporádicas hace falta modificar el modelo simple de tareas:
  - El parámetro ***T*** representa la **separación** mínima entre dos sucesos de activación consecutivos
  - Suponemos que en el peor caso la activación es pseudoperiódica (con período *T)*
  - El plazo de respuesta puede ser menor que el período ($D \leq T$)
- El análisis de tiempo de respuesta sigue siendo válido
- Funciona bien con cualquier orden de prioridad

Prioridades monónotas en plazos

Cuando los plazos son menores o iguales que los períodos, la asignación de mayor prioridad a las tareas de menor plazo de respuesta *(deadline monotonic scheduling)* es **óptima**

- El tiempo de respuesta se calcula de la misma forma que con la asignación monótona en frecuencia
  - se termina cuando $w_i^{n+1} = w_i^n$,
  - o cuando $w_i^{n+1} > D_i$

Ejemplo 5

Tarea	T	D	C	P	R
τ1	20	5	3	4	3
τ2	15	7	3	3	6
τ3	10	10	4	2	10
τ4	20	20	3	1	20

Con prioridades monónotas en frecuencia los plazos no están garantizados:

Tarea	T	D	C	P	R
τ3	10	10	4	4	4
τ2	15	7	3	3	7
τ1	20	5	3	2	10
τ4	20	20	3	1	20

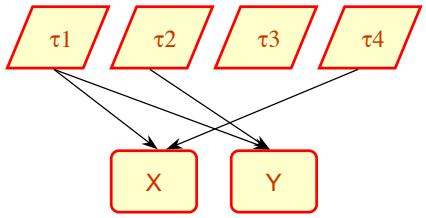
Índice

- Introducción
- Planificación con ejecutivos cíclicos
- **Planificación con prioridades**
  - Modelo de tareas básico
  - Tareas esporádicas y aperiódicas
  - **Interacción entre tareas y bloqueos**
  - Modelo de tareas generalizado
- Realización de sistemas con prioridades
- Otros métodos de planificación

Interacción entre tareas

- En la mayoría de los sistemas de interés práctico las tareas interaccionan mediante
  - datos comunes (protegidos)
  - citas o mensajes
- En todos estos casos puede ocurrir que una tarea tenga que esperar un suceso de otra menos prioritaria
- Esta situación se denomina **bloqueo**, y produce una **inversión de prioridad** indeseable
- La inversión de prioridad no se puede eliminar completamente, pero es posible limitar su duración

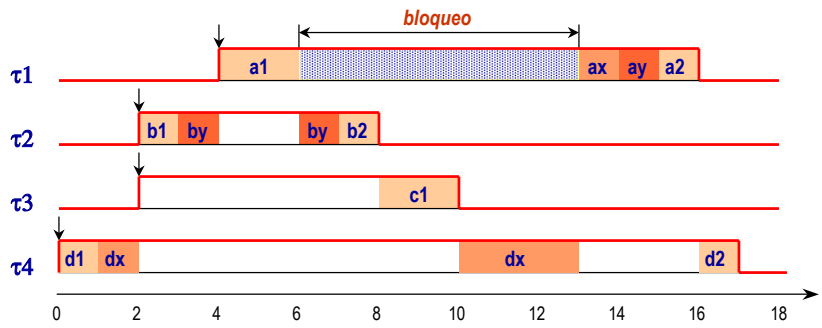
Ejemplo 6



Tarea	P	ta	Acciones
τ1	4	4	a1; ax; ay; a2
τ2	3	2	b1; by; b2
τ3	2	2	c1
τ4	1	0	d1; dx; d2

Acción	P	C	usa
a1	4	2	
ax	4	1	X
ay	4	1	Y
a2	4	1	
b1	3	1	
by	3	2	Y
b2	3	1	
c1	2	2	
d1	1	1	
dx	1	4	X
d2	1	1	

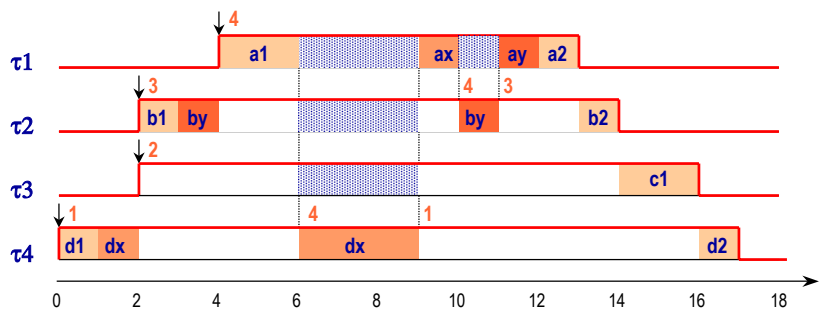
Ejemplo 6 : inversión de prioridad



Herencia de prioridad

- Una forma de reducir la duración de los bloqueos es variar dinámicamente la prioridad de las tareas
- Cuando una tarea está bloqueando a otra más prioritaria, **hereda** la prioridad de ésta
- La prioridad dinámica de una tarea es el máximo de
  - su prioridad básica
  - las prioridades de todas las tareas bloqueadas por ella
- La herencia de prioridad es **transitiva**

Ejemplo 6 : herencia de prioridad



Duración máxima del bloqueo

- Con el protocolo de herencia de prioridad, una tarea se pude bloquear como máximo
  - una vez por cada recurso
  - una vez por cada tarea de prioridad inferior
- La duración total máxima de los bloqueos es

$$B_i = \sum_{j \in lp(i),k} C_{j,k}$$

donde

- K es el número de recursos compartidos
- $C_{j,k}$ es el tiempo durante el cual la tarea j accede al recurso k (= 0 si no usa el recurso)

Ejemplo 6 : cálculo del bloqueo

$$B_1 = C_{2,Y} + C_{4,X} = 2 + 4 = 6$$
$$B_2 = C_{4,X} = 4$$
$$B_3 = C_{4,X} = 4$$
$$B_4 = 0$$

- Una tarea puede bloquearse por recursos a los que no accede (por ejemplo, $\tau 2$)
- Una tarea puede sufrir bloqueo aunque no acceda a recursos compartidos (por ejemplo, $\tau 3$)
- La tarea de menor prioridad ($\tau 4$) no sufre bloqueo

Tiempo de respuesta con bloqueos

- Cuando hay bloqueos, la ecuación del tiempo de respuesta queda así:

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

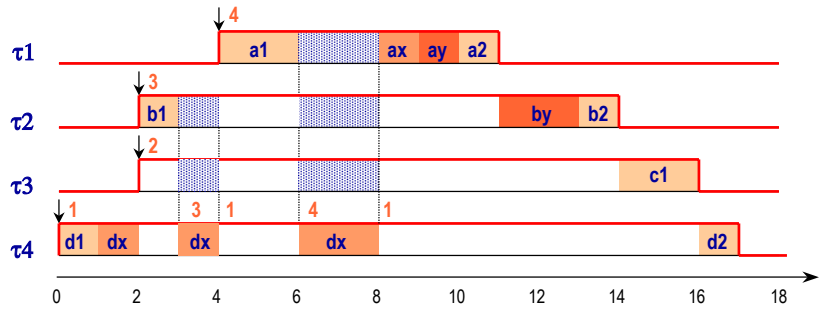
La solución se obtiene mediante la relación de recurrencia

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

Protocolos de techo de prioridad

- El **techo de prioridad** de un recurso es la máxima prioridad de las tareas que lo usan
- El **protocolo del techo de prioridad** consiste en :
  - la prioridad dinámica de una tarea es el máximo de su prioridad básica y las prioridades de las tareas a las que bloquea
  - una tarea sólo puede usar un recurso si su prioridad dinámica es mayor que el techo de todos los recursos en uso por otras tareas

Ejemplo 6 : techo de prioridad



Propiedades

- Cuando se usa el protocolo del techo de prioridad en un sistema monoprocesador,
  - Cada tarea se puede bloquear una vez, como máximo, en cada ciclo
  - No puede haber interbloqueos
  - No puede haber bloqueos encadenados
  - Se consigue la exclusión mutua sin mecanismos de protección adicionales
- La duración máxima del bloqueo es ahora

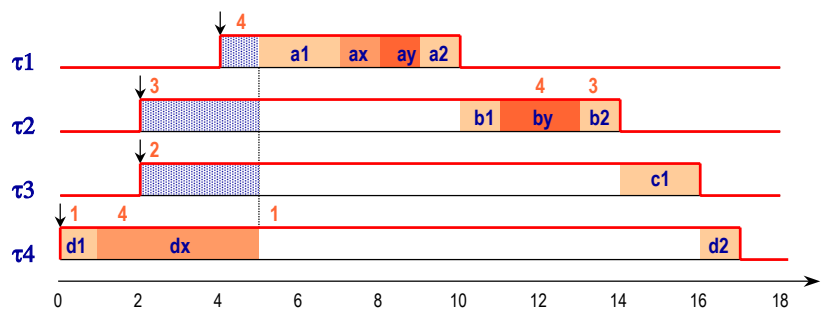
$$B_i = \max_{j \in lp(i), k \in lc(i)} C_{j,k}$$

lc(i) es el conjunto de recursos que pueden bloquear *i*

Protocolo del techo de prioridad inmediato

- Con este protocolo, una tarea que accede a un recurso hereda inmediatamente el techo de prioridad del recurso
  - la prioridad dinámica de una tarea es el máximo de su prioridad básica y los techos de prioridad de los recursos que usa
- Las propiedades son las mismas que las del protocolo del techo de prioridad, y además,
  - si una tarea se bloquea, lo hace al principio del ciclo
- Es más fácil de implementar que el protocolo básico
- Es más eficiente (menos cambios de contexto)

Ejemplo 6 : techo de prioridad inmediato



Índice

- Introducción
- Planificación con ejecutivos cíclicos
- **Planificación con prioridades**
  - Modelo de tareas básico
  - Tareas esporádicas y aperiódicas
  - Interacción entre tareas y bloqueos
  - **Modelo de tareas generalizado**
- Realización de sistemas con prioridades
- Otros métodos de planificación

Modelo de tareas generalizado

- El modelo de tareas básico que hemos visto incluye
  - tareas periódicas y esporádicas
  - plazos menores o iguales que los períodos
  - interacción mediante secciones críticas
- El análisis del tiempo de respuesta se puede extender con
  - planificación cooperativa
  - variación *(jitter)* en el esquema de activación
  - plazos arbitrarios (también mayores que los períodos)

Planificación cooperativa

- Es un método de despacho más eficiente que el desalojo inmediato
  - Las tareas se pueden bloquear por acceso al núcleo
  - Con el protocolo del techo de prioridad inmediato los bloqueos no se acumulan
  - Si B_{MAX} es el bloqueo máximo por recursos, se divide el código de las tareas en **bloques no desalojables** de duración B_{MAX}
  - Al final de cada bloque se ofrece un cambio de contexto (si hay alguna tarea más prioritaria)
  - Las secciones críticas se meten dentro de un solo bloque

Ventajas

- Ventajas
  - Mejora la planificabilidad
  - Se pueden reducir los tiempos de cómputo
  - Los tiempos de cómputo de los bloques son más fáciles de calcular
- Problemas
  - el coste de ofrecer un cambio de contexto puede ser alto (e innecesario)

Análisis del tiempo de respuesta

- El último bloque de una tarea no puede sufrir interferencia
- La ecuación queda así:

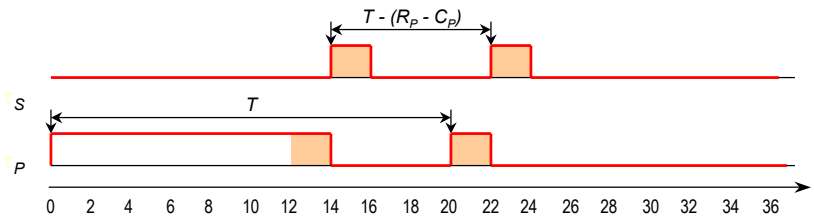
$$w_i^{n+1} = B_{MAX} + C_i - F_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil \cdot C_j$$

$$R_i = w_i^n + F_i, \text{para } w_i^{n+1} = w_i^n$$

donde F_i es el tiempo de cómputo del último bloque

Activación irregular

- Puede haber variaciones en los esquemas de activación periódicos y esporádicos
- Por ejemplo, sea una tarea esporádica activada por un suceso producido por una tarea periódica situada en otro procesador



Tareas esporádicas con *jitter*

- La variación en los instantes de activación con respecto al esquema ideal se denomina *jitter*
- El tiempo de respuesta de la tarea *i* se obtiene con

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

$$R_i = w_i^n, \text{ para } w_i^{n+1} = w_i^n$$

donde J_j es el *jitter* máximo de la tarea *j*

Tareas periódicas con *jitter*

- Las tareas periódicas no suelen tener *jitter*
- Puede haberlo si la planificación se hace con una granularidad apreciable
- El tiempo de respuesta se calcula con

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

$$R_i = w_i^n + J_i, \text{ para } w_i^{n+1} = w_i^n$$

Plazos arbitrarios

- Si el plazo es mayor que el período, puede haber varias activaciones pendientes en un ciclo
- Para analizar esta situación construimos una sucesión de **ventanas** $w_i(q)$, donde $q + 1$ es el número de activaciones de τ_i.
- Para cada ventana se obtiene un valor de R_i con

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

$$R_i(q) = w_i^n(q) - q \cdot T_i$$

Tiempo de respuesta máximo

- Se calcula R_i(q) para $q = 0, 1, \ldots, q_{max}$, donde q_{max} es el menor valor que cumple

$$R_i(q_{max}) \leq T_i$$

En esta situación ya no hay solape con la siguiente activación

- El tiempo de respuesta en el peor caso es

$$R_i = \max_{q=0,..q_{max}} R_i(q)$$

Caso general

- Si consideramos plazos arbitrarios y *jitter*, el tiempo de respuesta se obtiene con

$$w_i^{n+1}(q) = (q + 1) \cdot C_i + B_i + \sum_{i \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil \cdot C_j$$

$$R_i(q) = w_i^n(q) - q \cdot T_i + J_i$$

Asignación de prioridades

- Con el modelo general que hemos, la ordenación de prioridades monótona en frecuencia (RMS) o en plazos **no es óptima**
- Se aplica el siguiente **teorema**:

> *Si una tarea τ cumple su plazo con la prioridad más baja del sistema, y existe un orden de prioridades admisible para todo el sistema, entonces hay un orden de prioridades que garantiza todas las tareas en el cual τ tiene la prioridad más baja*

Algoritmo

```
procedure Assign_Priority (System: in out Task_Set;
                           N      :        Natural;
                           OK     :    out Boolean) is
begin
  for k in 1 .. N loop
    for next in k ..N loop
      Swap (System, k, next);
      Test (System, k, OK);
      exit when OK;
    end loop;
    exit when not OK; -- no hay ninguna tarea garantizada
  end loop;
end Assign_Priority;
```

`System(1)` tiene la prioridad mínima y `System(N)` la máxima

`Test` comprueba el plazo de la tarea `System (k)`

Índice

Planificación de tareas en Ada

- En el anexo de tiempo real se define un modelo de planificación con prioridades y desalojo
- La prioridad de una tarea es de un subtipo definido en el paquete *System*

```
subtype Any_Priority is Integer range intervalo;
subtype Priority is Any_Priority
  range Any_Priority'First .. valor;
subtype Interrupt_Priority is Any_Priority
  range Priority'Last + 1 .. Any_Priority'Last;
```

  - Los valores mayores denotan prioridades más altas

Prioridad básica

- La prioridad básica de una tarea o un tipo tarea se especifica con un pragma *Priority*

```
task Controller is
   pragma Priority (10);
end Controller;
```

```
task type Server (Task_Priority : System.Priority) is
   entry Service_1 (...);
   ...
   pragma Priority (Task_Priority);
end Server;
```

Techo de prioridad inmediato

- Se puede usar el protocolo del techo de prioridad inmediato para acceder a los objetos protegidos
- Hay que poner un pragma

```
pragma Locking_Policy (Ceiling_Locking);
```

  - El techo de prioridad de un objeto o un tipo protegido se especifica mediante un pragma *Priority*

```
protected type Resource(Ceiling : System.Priority)is
   ...
   pragma Priority (Ceiling);
end Resource;
```


Prioridad activa

- La prioridad activa de una tarea es el máximo de su prioridad básica y la prioridad heredada de otras
- Una tarea puede heredar prioridad de varias formas :
  - cuando ejecuta una operación protegida hereda el techo de prioridad del objeto
  - cuando ejecuta una instrucción *accept* hereda la prioridad de la tarea que llama al punto de entrada
  - durante su activación hereda la prioridad de su progenitor

Protocolo de despacho

- Se puede especificar un protocolo de despacho mediante un pragma:

```
pragma Task_Dispatching_Policy (FIFO_within_Priority);
```

- Hay una cola conceptual por cada nivel de prioridad
- El núcleo de ejecución despacha la primera tarea de la cola de mayor prioridad que no esté vacía
- Cuando una tarea se activa se coloca al final de la cola correspondiente a su prioridad activa
- Las tareas desalojadas se colocan al principio de su cola

Otros mecanismos de planificación

- Prioridades dinámicas (que se pueden cambiar por el programa)
- Colas ordenadas por prioridades en puntos de entrada

```
pragma Queuing_Policy (Priority_Queuing);
```

- Identificadores y atributos de tareas
- Control síncrono y asíncrono de tareas
- Aborto inmediato
- Restricciones para facilitar el análisis de tiempos
- Requisitos sobre documentación de tiempos

Planificación en POSIX

- POSIX.1b define un modelo de planificación con prioridades y desalojo
- Se puede especificar el protocolo de despacho *FIFO* (como en Ada)
- Se puede especificar que todos los *threads* de un sistema se planifiquen globalmente *(system contention)*
- La prioridad básica de una tarea se puede cambiar dinámicamente

Ámbito de planificación

- El ámbito de la planificación se define con

```
#include <pthread.h>
int pthread_attr_setscope (pthread_attr_t *attr,
                           int contentionscope);
```

  - También hay que especificar si los atributos de planificación son específicos de cada *thread*:

```
int pthread_attr_setinheritsched (
                           pthread_attr_t *attr,
                           int inheritsched);
```

  - Normalmente se pone
    - PTHREAD_SCOPE_SYSTEM para *contentionscope*
    - PTHREAD_EXPLICIT_SCHED para *inheritsched*

Política de planificación

- La política de planificación dentro del mismo nivel de prioridad se define con

```
int pthread_attr_setschedpolicy (
                        pthread_attr_t *attr,
                        int policy);
```

  - El parámetro *policy* puede ser
    - SCHED_FIFO (orden de llegada)
    - SCHED_RR (turno circular con rodajas de tiempo)
    - SCHED_OTHER (dependiente de la implementación)

  En general, hay que poner SCHED_FIFO

Prioridades

- La prioridad de un *thread* es un parámetro de planificación:

```
#include <sched.h>

struct sched_param {
  int sched_priority;
  ...
}
```

  y se especifica con

```
int pthread_attr_setschedparam (
                    const pthread_attr_t *attr,
                    const struct sched_param *param);
```

  Los atributos se usan para crear el *thread*

Protocolo de acceso a *mutex*

- Se puede especificar un protocolo de acceso con

```
#include <pthread.h>

int pthread_mutexattr_setprotocol (
                      pthread_mutex_attr_t *attr,
                      int *protocol);
```

El protocolo puede ser

- PTHREAD_PRIO_INHERIT (herencia de prioridad)
- PTHREAD_PRIO_PROTECT (techo de prioridad inmediato)
- PTHREAD_PRIO_NONE (no hay herencia de prioridad)

Techo de prioridad

- Se especifica con

```
int pthread_mutexattr_setprioceiling (
                    pthread_mutexattr_t *attr,
                    int prioceiling);
```

Los atributos se usan para iniciar el *mutex*

Índice

- Introducción
- Planificación con ejecutivos cíclicos
- Planificación con prioridades
  - Modelo de tareas básico
  - Tareas esporádicas y aperiódicas
  - Interacción entre tareas y bloqueos
  - Modelo de tareas generalizado
- Realización de sistemas con prioridades
- **Otros métodos de planificación**

Otros métodos de planificación

- Se pueden conseguir mejores resultados realizando la planificación **dinámicamente** en función de los requisitos temporales y de los recursos disponibles
- Dos métodos dinámicos interesantes son:
  - **Primero el más urgente** *(earliest deadline first, EDS)*
  - **Primero el menos holgado** *(least slack first, LSF)*
- Ambos son óptimos para tareas independientes
  - se garantizan los plazos hasta el 100% de utilización
  - se comportan bien cuando hay muchas tareas esporádicas
- Problemas:
  - el comportamiento en caso de sobrecarga es imprevisible
  - no está bien resuelta la interacción entre tareas

Resumen

- Un método de planificación tiene dos partes:
  - un **algoritmo de reparto** de recursos
  - un **método de análisis** del comportamiento temporal
- El método clásico se basa en un **ejecutivo cíclico**
  - el análisis es inmediato (por construcción)
  - es poco flexible y de bajo nivel
- Un método mejor se basa en el uso de **prioridades fijas**
  - las prioridades se asignan por orden de períodos, plazos o de forma arbitraria
  - se puede analizar el tiempo de respuesta de las tareas
  - se pueden analizar tareas con interacción si se usa un protocolo de herencia o techo de prioridad