

Tema 2: Static Priorities Scheduling

Sistemas Empotrados 2

25/09/2025

Índice

Objetivos	1
Métricas	2
Factor de Utilización (Utilization factor)	2
Trabajo del procesador (W)	2
Tiempo de respuesta de una tarea T_i	2
Como calcularlo	2
Planificación de tareas periódicas críticas	2
Rate Monotonic (Deadline = Periodo)	2
Condiciones de garantía de deadlines	2
Deadline Monotonic (Deadline \leq Periodo)	3
Condiciones de garantía de deadlines	3
Cambios de contexto	3
Tareas periódicas críticas y no críticas	3
Comunicación entre tareas	4
Inversión de prioridad	4
Protocolos limitadores de inversión de prioridad	4
Sección crítica no preferente	4
Herencia de prioridad	4
Techo de prioridad	4
Techo de prioridad inmediato	4
Cálculo de bloqueos	4
Sección crítica no preferente	5
Protocolo de techo de prioridad (y Techo de prioridad inmediato)	5
Protocolo de herencia de prioridad	5
Más garantías de deadlines	5
Resumen	6
Cumplir deadlines	6
Cálculo de tiempo de bloqueo	7
Tareas esporádicas y aperiódicas	7
Tareas aperiódicas	7
Servidor esporádico	7

Objetivos

El objetivo principal es **analizar y asegurar el cumplimiento de plazos (deadlines) de tareas críticas del sistema.**

Para ello se presentan:

- Guías de diseño de sistemas de tiempo
 - Tareas periódicas
 - Tareas esporádicas
 - Comunicación a través de servidores
- Técnicas de priorización estática:
 - Rate Monotonic (RM): **priority to the most frequent**
 - * RMS/RMA: Rate Monotonic Scheduling / Analysis

- * Response time = period in periodic tasks.
- Deadline Monotonic (DM): **priority to the most urgent**
 - * DMS/DMA: Deadline Monotonic Scheduling / Analysis
 - * Deadline for response \leq period in periodic tasks

Métricas

Factor de Utilización (Utilization factor)

Medida de la **carga de un procesador**. El objetivo es encontrar métodos que produzcan una planificación aceptable con factores de utilización lo más altos posibles.

Utilización de una tarea T_i	Utilización del sistema
$U_i = \frac{C_i}{P_i}$	$U = \sum_{i=1}^n U_i = \sum_{i=1}^n \frac{C_i}{P_i}$

Trabajo del procesador (W)

Trabajo pedido al procesador (W) en un cierto nivel de prioridad (i):

- Peor escenario:
 - Todas las tareas empiezan en $t=0$.
 - Todas las tareas son ejecutadas con su **WCET** (Worst-Case Execution Time)
- Tareas ordenadas por prioridad.

Dado un nivel de prioridad i, el trabajo solicitado al procesador es:

$$W_i(t) = \sum_{j=1}^i \lceil \frac{t}{P_j} \rceil C_j$$

Tiempo de respuesta de una tarea T_i

Si el trabajo solicitado concuerda con el tiempo transcurrido, la CPU ha sido capaz de completar todo el trabajo, por lo tanto, la tarea T_i cumplirá el deadline de respuesta si $R_i \leq D_i$.

Como calcularlo

Se puede utilizar el siguiente método iterativo hasta que se consiga que $W_i^{k+1} = W_i^k = R_i$:

$$W_i^{k+1} = \lceil \frac{W_i^k}{P_1} \rceil \times C_1 + \cdots + \lceil \frac{W_i^k}{P_{i-1}} \rceil \times C_{i-1} + C_i$$

Planificación de tareas periódicas críticas

Planificación basada en prioridad:

- Consideraremos **prioridades estáticas** y programación preventiva.
- Las tareas se suelen priorizar por importancia.

Rate Monotonic (Deadline = Periodo)

- **Prioridad al más frecuente (RMS).**
- **Es óptima** (Liu & Layland, 1973), no hay mejor planificación. Si no se consigue ningún plan admisible, no se puede obtener de ninguna otra forma.

Condiciones de garantía de deadlines

- **Teorema 1 (Liu & Layland):** en un sistema de n tareas periódicas independientes con prioridades asignadas en orden de frecuencia, todos los deadlines se cumplen para cualquier configuración inicial si:

$$U = \sum_{i=1}^n \frac{C_i}{P_i} \leq n \times (2^{1/n} - 1) = U_0(n)$$

Donde $U_0(n)$ es la utilización mínima garantizada para n tareas:

- $U_0(1) = 1.000$
- $U_0(2) = 0.828$
- $U_0(3) = 0.779$
- $U_0(4) = 0.757$
- $U_0(5) = 0.743$

Para infinitas tareas, $0.693 = \ln(2)$.

- **Teorema 2 (Liu & Layland):** en un sistema de n tareas periódicas independientes con prioridades estáticas, todos los deadlines se cumplen para cualquier lag en el inicio de las tareas si cuando todas son activadas en el mismo momento todas cumplen con su deadline.

Deadline Monotonic (Deadline \leq Periodo)

- **Prioridad al más urgente (tiempo de respuesta más pequeño)** también es **óptimo** (Leung & Whitehead).

Condiciones de garantía de deadlines

- **Condición de garantía 1:** esta es una condición suficiente pero no necesaria.
 - En un sistema de n tareas periodicas independientes con prioridades estáticas, todos los deadlines se cumplen para cualquier inicio si:
$$C_i + \sum_{j=1}^{i-1} \lceil \frac{D_i}{P_j} \rceil \times C_j \leq D_i$$
- **Condición de garantía 2:** $t_{fi} < D_i$, donde t_{fi} es el tiempo de finalización de la tarea i.
- **Teorema 3 (Lehoczky, Sha & Ding):** en un sistema de n tareas periodicas independientes con prioridades estáticas asignadas al más urgente, todos los deadlines se cumplen para cualquier setup inicial de las tareas si:

$$W_i(t) = \sum_{j=1}^i (C_j \times \lceil \frac{t}{P_j} \rceil) \leq t$$

Donde t representa cada uno de los “**Scheduling Points**” (para una tarea T_i son todos los instantes límites (fin de periodos) de tareas de mayor o igual prioridad a T_i que ocurren antes del final de su deadline).

Cambios de contexto

En el peor de los casos un cambio de contexto conlleva:

- C_{sa} : guardar el contexto de la tarea en ejecución.
- C_{sb} : planificar y recuperar el contexto de la nueva tarea.

Cada vez que una tarea se antepone a una tarea de menor prioridad, **se llevan a cabo dos cambios de contexto**, esto implica que se debe añadir $2 * C_s$ al tiempo de ejecución de cada tarea para su análisis.

Tareas periódicas críticas y no críticas

Cuando una tarea es **no crítica** podemos admitir fallos ocasionales en sus tiempos de respuesta. El objetivo es garantizar:

- Se cumplen los deadlines para todas las tareas (con los tiempos de cómputo medios).
- Las tareas críticas cumplen deadlines con los tiempos de cómputo máximos.

En el caso de que las tareas críticas no son las más frecuentes, se puede hacer una transformación del periodo (acortar periodo de dicha tarea o alargar el periodo de las menos críticas).

Comunicación entre tareas

Simplificaremos los problemas de modo que:

- Las comunicaciones entre procesos son hacia variables compartidas protegidas por secciones críticas.
- Las primitivas del monitor no pueden ser bloqueadas (evitar bloqueos).

Inversión de prioridad

A veces, **una tarea puede ser retrasada por la ejecución de otra tarea de menor prioridad** (esto puede ocurrir si hay secciones críticas activas cuando se activa una tarea de mayor prioridad). Esto ocurre cuando hay interacción entre tareas. Este retraso puede no estar limitado y por eso es necesario **crear protocolos para limitarlo**.

Protocolos limitadores de inversión de prioridad

Sección crítica no preferente

Un servidor ejecutando un servicio no puede perder el procesador.

Inversión de prioridad Hay inversión de prioridad incluso si la tarea de mayor prioridad es independiente y no solicita ningún servicio.

Herencia de prioridad

Los clientes tienen prioridades estáticas y los servidores dinámicas. Se basa en las siguientes reglas:

- Cuando un servidor está ofreciendo un servicio o bloqueando a un cliente, hereda la prioridad del cliente más prioritario.
- Las llamadas pendientes son aceptadas en orden de prioridad de cliente.
- Protocolo a nivel de Kernel sobre los MUTEX que protege el servidor.

Inversión de prioridad Solo ocurre cuando una tarea solicita un servicio de un servidor que está procesando la solicitud de otra tarea de menor prioridad. Puede haber más de un bloqueo por inversión de prioridad durante una ejecución.

Techo de prioridad

Es un protocolo de herencia de prioridad con las siguientes reglas adicionales:

- El techo de prioridad de un servidor se asigna al de mayor prioridad de entre los clientes potenciales.
- Un cliente intentando llamar a un servidor es bloqueado si cualquier otro servidor con un techo de prioridad mayor o igual a la prioridad del cliente ya está ejecutando un servicio para otro cliente.

Características

- Bloqueo único: en la ejecución de una tarea, la tarea solo puede ser bloqueada por inversión de prioridad durante la ejecución de un servicio.

Techo de prioridad inmediato

Con este protocolo, una tarea que accede a un recurso hereda inmediatamente el techo de prioridad del servidor.

- Las propiedades son iguales que el techo de prioridad simple.
- Más fácil de implementar.
- Más eficiente (menos cambios de contexto).
- Más bloqueos que en techo de prioridad simple.
 - Caso peor idéntico.
 - Peor rendimiento medio.

Si el protocolo no ha sido implementado en el kernel, puede ser usado asignando una prioridad a cada servidor.

Cálculo de bloqueos

Las inversiones de prioridad pueden ser tenidas en cuenta en los análisis añadiendo un término:

- **Blocking time** b_i

- Se trata del blocking time del peor caso.
- Cada protocolo ofrece diferentes tiempos de bloqueo.
- Interferencia de otras tareas
 - Tareas de mayor prioridad (**expulsiones**).
 - Tareas de menor prioridad (bloqueos).

$$W_i(t) = \lceil \frac{t}{P_1} \rceil C_1 + \cdots + \lceil \frac{t}{P_{i-1}} \rceil C_{i-1} + C_i + b_i$$

Sección crítica no preferente

Una tarea puede ser bloqueada como máximo por un servicio.

Cálculo

$$B_i = \max_{j>i} \left(\max_k (D(s_{jk})) \right)$$

- s_{jk} servicio k usado por la tarea j.
- $D(s_{jk})$ longitud del servicio.

Protocolo de techo de prioridad (y Techo de prioridad inmediato)

El valor máximo del blocking time de una tarea T_i , b_i , es igual a la duración del servicio solicitado más largo. Un cliente puede ser bloqueado por una tarea de menor prioridad incluso si no se llama al mismo servidor. El cliente de menor prioridad no puede ser bloqueado por clientes de menor prioridad $B = 0$.

Cálculo

$$B_i = \max_{j,k} \{ D_{j,k} \mid Pr_j < Pr_i, Ceiling(S_k) \geq Pr_i \}$$

Protocolo de herencia de prioridad

Una tarea T puede ser bloqueada como máximo por $\min n, m$:

- n es el número de tareas de menor prioridad a T que pueden bloquearle.
- m es el número de servidores diferentes que pueden bloquear a T.

Cálculo Para calcular B_i es necesario establecer que servicios pueden estar ejecutándose cuando T se activa y cuales le pueden bloquear (directa o indirectamente).

Algoritmo para calcularlo:

- $Ceiling(S_k)$: máxima prioridad de los clientes del servicio k.
- $D_{j,k}$: duración de un servicio solicitado por la tarea T_j al servidor S_k .

$$B_i^l = \sum_{j=i+1}^n \max_k [D_{j,k} : Ceiling(S_k) \geq P_i],$$

$$B_i^s = \sum_{k=1}^m \max_{j>1} [D_{j,k} : Ceiling(S_k) \geq P_i],$$

$$B_i = \min (B_i^l, B_i^s).$$

Más garantías de deadlines

- **Teorema 4 (Sha, Rajkumar & Lehoczky)**: en un sistema de n tareas periódicas con prioridades asignadas por frecuencia, que se comunican a través de servidores, todas las respuestas cumplen con su deadline para cada escenario si:

$$\forall i, 1 \leq i \leq n, \quad \frac{C_1}{P_i} + \cdots + \frac{C_i}{P_i} + \frac{b_i}{P_i} \leq i \left(2^{1/i} - 1 \right)$$

- Corollary (Sha, Rajkumar & Lehoczky): lo mismo es cierto si:

$$\sum_{i=1}^n \frac{C_i}{P_i} + \max_{i=1..n-1} \left(\frac{b_i}{P_i} \right) \leq n \left(2^{1/n} - 1 \right) = U_0(n)$$

Condición suficiente.

- Teorema 5 (Sha, Rajkumar & Lehoczky): en un sistema de n tareas periódicas asignadas en orden de frecuencia, que se comunican a través de servidores, todas las respuestas cumplen con su deadline para cada escenario si:

$$\forall i, 1 \leq i \leq n, \quad \min_{t \in S_i} \left(\sum_{j=1}^i \frac{C_j}{t} \left\lceil \frac{t}{P_j} \right\rceil \right) + \frac{b_i}{t} \leq 1$$

- Test de tiempo de completitud: el tiempo de completitud de una tarea T_i (t_{fi}) es:

$$t_{fi} = \min_{t>0} \left(\frac{t}{P_1} C_1 + \dots + \frac{t}{P_{i-1}} C_{i-1} + C_i + b_i = t \right)$$

Resumen

Cumplir deadlines

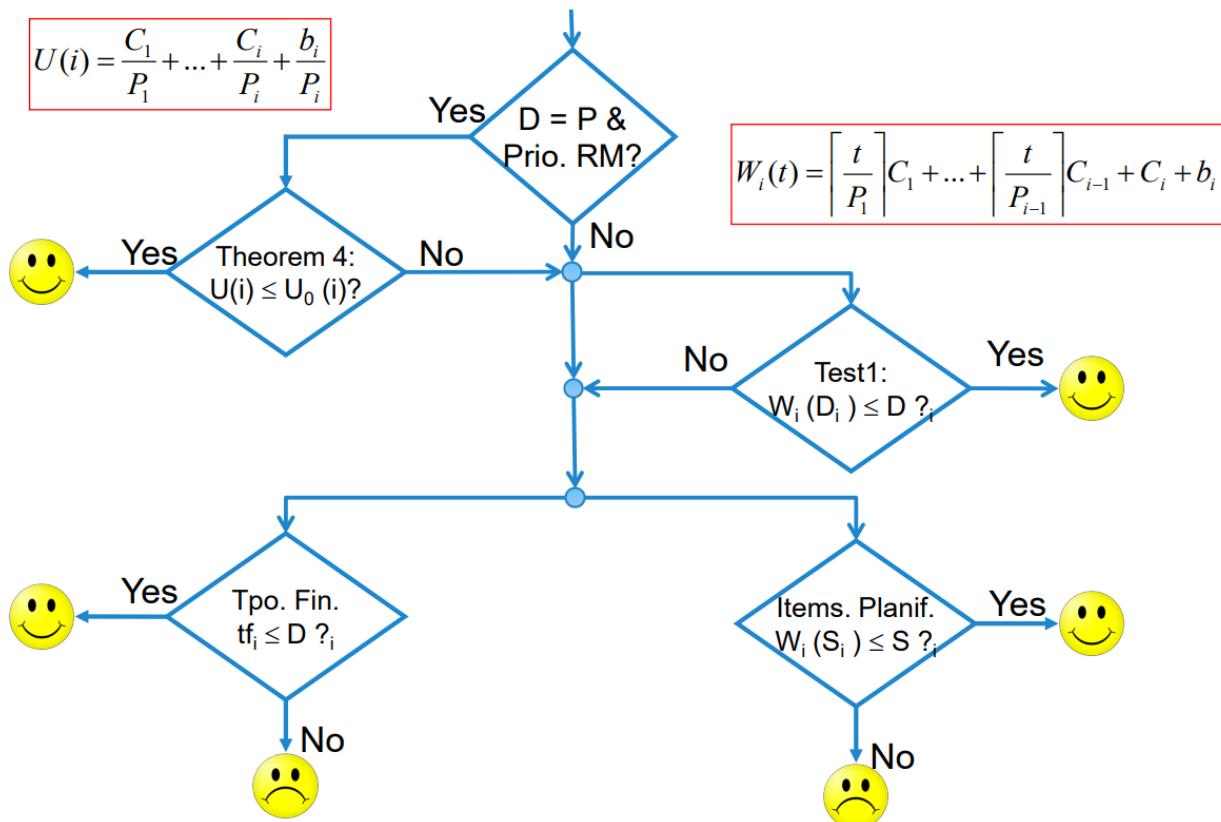
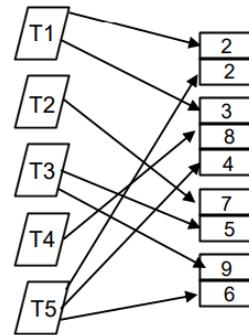


Figure 1: Diagrama de actuación para verificar cumplimiento de deadlines

Summary: Blocking Times

T_{Pr}	5	5	4	3
Pr_{Pr}	S ₁	S ₂	S ₃	S ₄
5	T ₁	2	3	
4	T ₂			7
3	T ₃		5	9
2	T ₄		8	
1	T ₅	2	4	6



S.C. not expellable

T: $Pr < Pr_i$

S: All

No. of blockages: 1

Prior Ceiling / T.P.imm.

T: $Pr < Pr_i$

S: Ceiling $\geq Pr_i$

No. of blockages: 1

Priority Inheritance

T: $Pr < Pr_i$

S: Ceiling $\geq Pr_i$

No. of blockages:

- 1 per Task
- 1 per Server
(pessimistic estimate)

Figure 2: Resumen de actuación para calcular tiempos de bloqueo

Tareas esporádicas y aperiódicas

Las tareas que son ejecutadas en respuesta a un evento son **NO periodicas** y hay de dos tipos:

- **Tareas esprádicas:** tienen tiempo de respuesta crítico.
- **Tareas aperiódicas:** no tienen requerimientos críticos. Generalmente se desea que tenga un comportamiento estadístico adecuado.

Tareas aperiódicas

Pueden ser ejecutadas de varias formas:

- Como tareas de “background”: solución simple pero tiempo de respuesta muy elevado.
- Procesamiento directo de eventos: buen tiempo de respuesta pero puede perjudicar a tareas de menor prioridad.
- Servidores aperiódicos: son tareas que ejecutan las actividades aperiódicas en momentos oportunos de tal forma que no molesten a las tareas críticas. Hay de varios tipos:
 - **Servidor de muestreo:** tarea periódica que en cada activación ejecuta las actividades aperiódicas que sea necesario. Los tiempos de computo deben ser limitados.
 - **Servidor esprádico:** se basa en la idea de reservar tiempo para atender eventos aperiódicos. Si hay tiempo reservado el evento se atiende inmediatamente.

Servidor esporádico

Hay varias maneras de implementarlo dependiendo de la política de “backfill”:

- **Periodo de backfill:** igual a la media de la tasa de llegada de eventos.
- **Execution capacity:** tiempo de procesado de un evento en el peor de los casos.

- **Priority assignment:** en el peor de los casos el servidor esporádico se comporta como una tarea periódica con tiempo de periodo S y tiempo de respuesta igual al periodo.

Es necesario asumir una separación entre eventos, s_i . Se puede lograr expulsando los eventos que no cumplan con la mínima separación posible.