Tema 4

Introducción a los Sistemas de Tiempo Real

desarrollo de programas con tareas de tiempo real

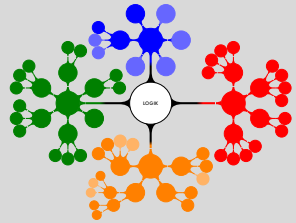
Asignatura *Sistemas Concurrentes y Distribuidos*

Fecha 22 de noviembre de 2024

Manuel I. Capel
manuelcapel@ugr.es
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada

Confusión conceptual acerca de qué es un STR

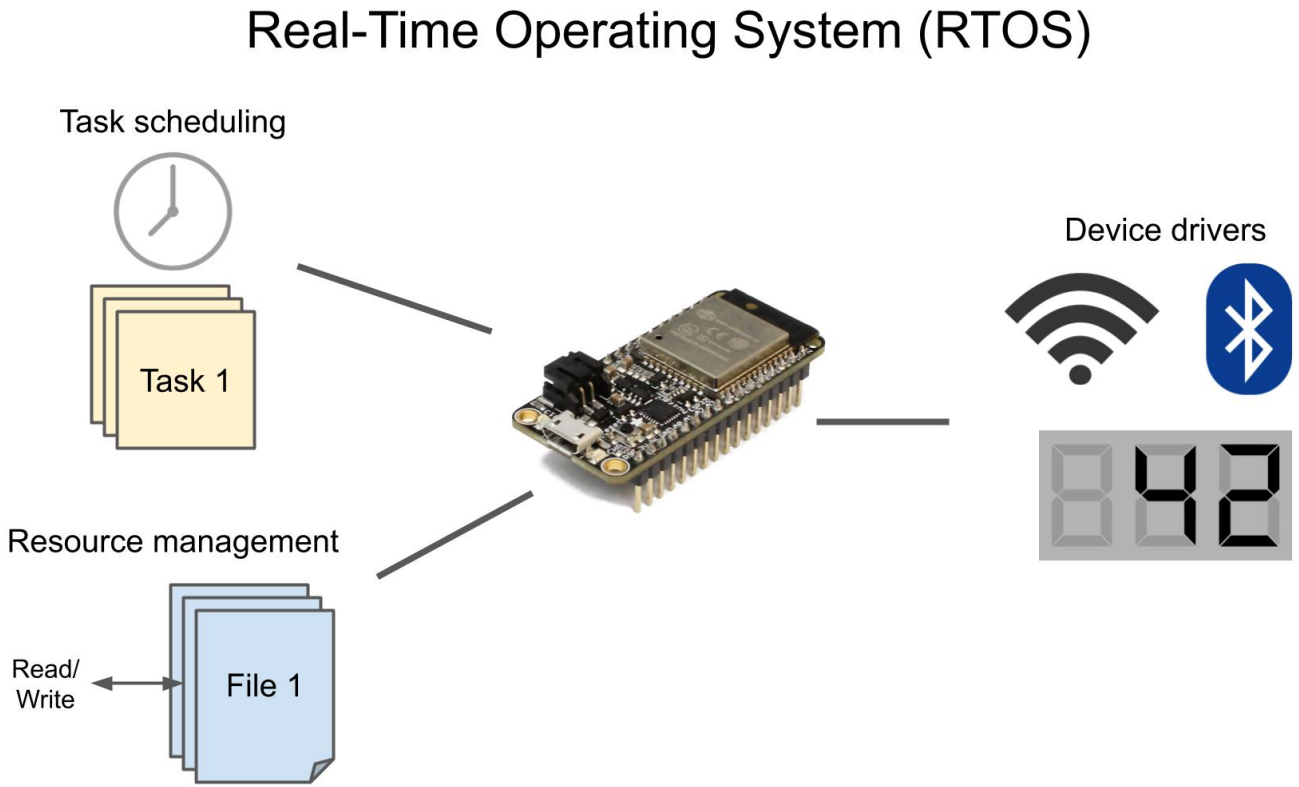
Manuel I. Capel
manuelcapel@ugr.es



- Los STR se confunden con sistemas:
  (a) *en línea*, (b)*interactivos*, (c) *rápidos*
- Propiedades definitorias de un STR :
  1. *Reactividad*
  2. *Determinismo*
  3. *Responsividad*
  4. *Confiabilidad*

Un STR es aquél en el que la respuesta correcta a un cálculo realizado por el programa no sólo depende de que efectivamente *haga lo que tenga que hacer*, sino también de cuándo esté disponible dicho resultado.

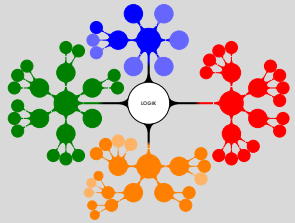
Definición en el ámbito de los Sistemas Operativos



Sistemas operativos de de tiempo real

Sistema operativo de TR es aquel que tiene la capacidad para suministrar un nivel de servicio requerido en un tiempo limitado y especificado a priori.

Manuel I. Capel
manuelcapel@ugr.es



Clasificación de las aplicaciones de tiempo real basada en varios criterios

- Atendiendo a la *criticidad* de los STR:

Denominación	Ejemplo	Complementos
1. Misión crítica	Control de aterrizaje	Tolerancia a fallos
2. Estrictos	Reservas de vuelos	Calidad de respuesta
3. Permisivos	Adquisición datos meteorológicos	Medidas de fiabilidad

Español	Inglés
Misión Crítica	Hard Real-time Task
Estrictos	Firm Real-time Task
Permisivos	Soft Real-Time Task, Systems with time criticality

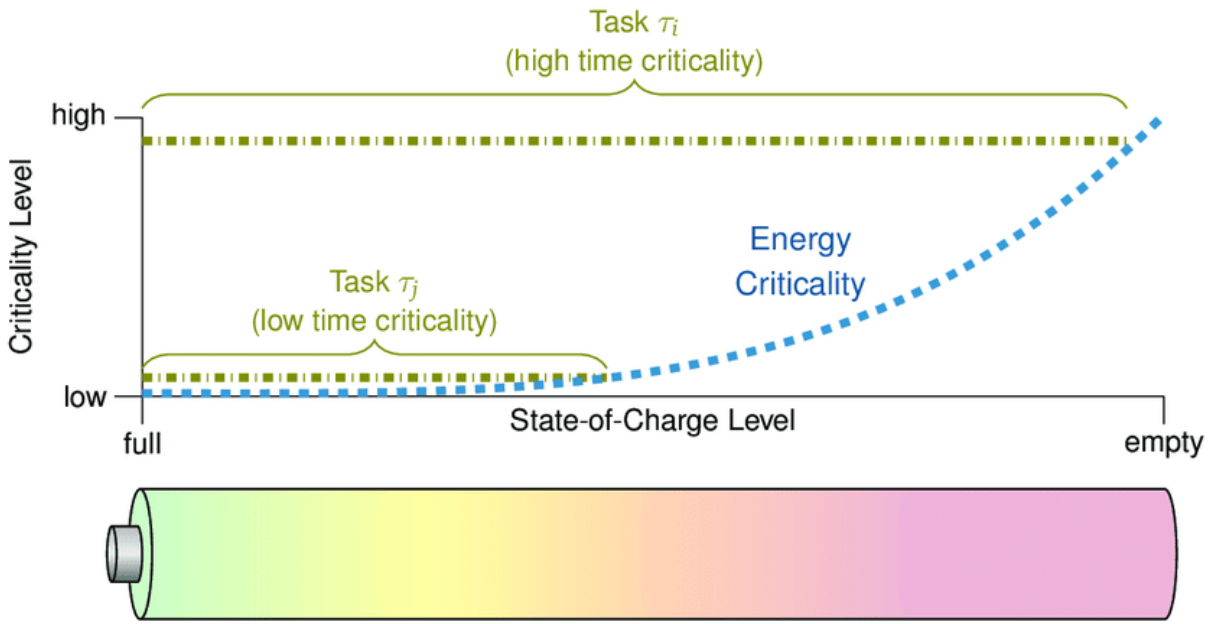
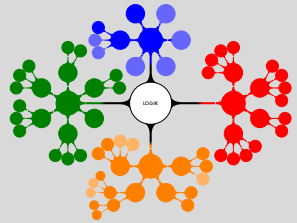


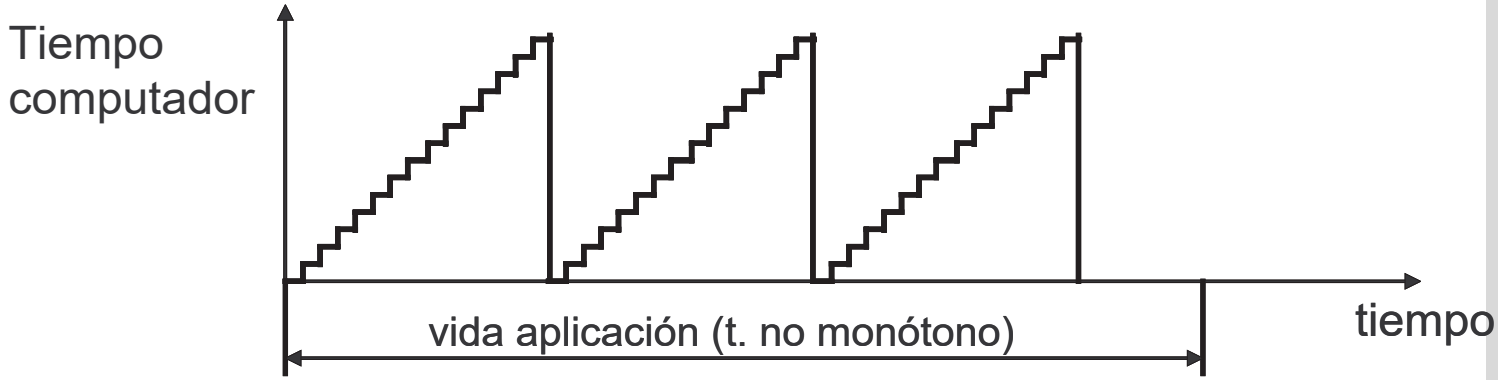
Ilustración de 2 tareas con diferentes niveles de criticidad temporal

Manuel I. Capel
manuelcapel@ugr.es



Tipos de medidas del tiempo de interés en STR

1. Tiempo absoluto:
   - Sistemas de referencia locales,
   - Astronómicos (`UT0`),
   - Atómicos (`IAT`),
   - Tiempo Coordinado Universal, desde 1972 (`UTC`),
   - Satelital (`GPS`).
2. Intervalos o tiempo relativo:

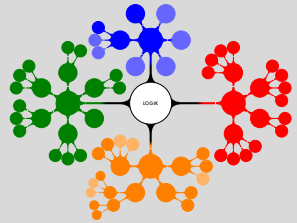


Representación del tiempo no-monótono en un reloj de tiempo real

El tiempo de GPS (GPST) es una escala de tiempo continua (sin segundos intercalares) definida por el segmento de control GPS sobre la base de un conjunto de relojes atómicos en las estaciones de monitorización y a bordo de los satélites. Comenzó a las 0h UTC (medianoche) del 5 al 6 de enero de 1980.

El UTC se obtiene a partir del Tiempo Atómico Internacional (IAT), este estándar se calcula a partir de una media ponderada de las señales de los relojes atómicos, localizados en cerca de 70 laboratorios nacionales de todo el mundo. Se retrasa con

Manuel I. Capel
manuelcapel@ugr.es



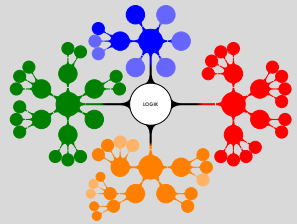
Medida del tiempo

- Concepto de *reloj de tiempo real* : $\{\texttt{oscilador}, \texttt{contador}, \texttt{software}\}$
- Características más importantes:
  - Precisión (granularidad)
  - Tiempo de desbordamiento
  - Intervalo
- Escalas temporales:
  - Tiempo monótono /No monótono
  - Absolutas/no absolutas

Precisión	Intervalo
100 ns.	Hasta 429,5 s.
1 μs.	Hasta 71,58 m.
100 μs.	Hasta 119,3 h.
1 ms.	Hasta 49,71 días
1 s.	Hasta 136,18 años

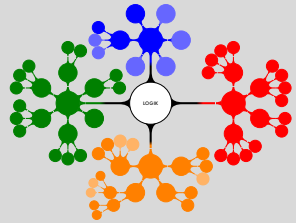
Intervalo vs.precisión para un contador de 32 bits

Manuel I. Capel
manuelcapel@ugr.es



Temporizadores

Manuel I. Capel
manuelcapel@ugr.es

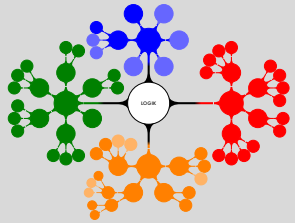


Ideas fundamentales

- Se activan con una llamada (operación) de sistema operativo, **no son instrucciones de los lenguajes de programación**
- Tipo de reloj especializado de los sistemas operativos
- Elementos para programar:
  - Tiempo de *arranque*
  - Tiempo de *parada*

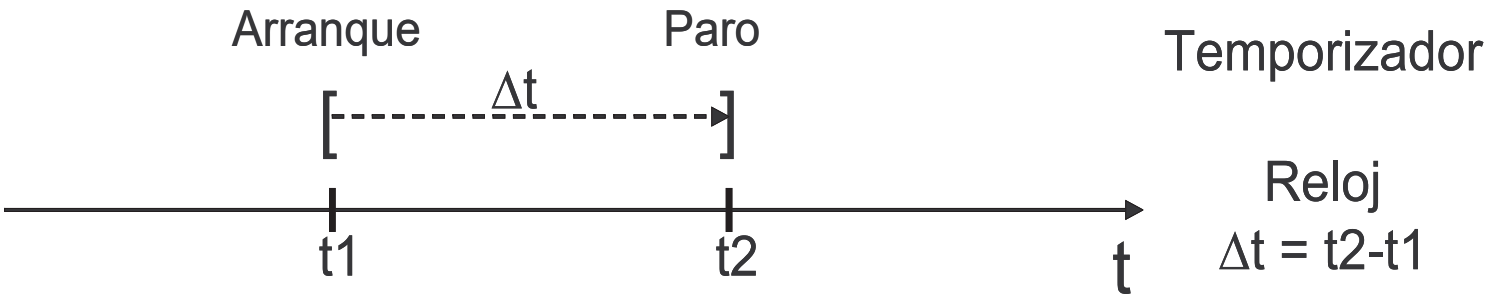
Tipos de temporizadores

Manuel I. Capel
manuelcapel@ugr.es



Características fundamentales:

- De 1 solo disparo
- Periódicos
- Pueden presentar *deriva*



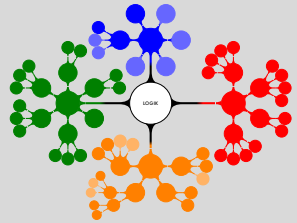
Representación del tiempo no-monótono en un reloj de tiempo real

Retardos en las tareas:

- Se pueden programar con *temporizadores* o con instrucciones de los lenguajes de programación
- Suspenden, al menos, hasta la duración especificada

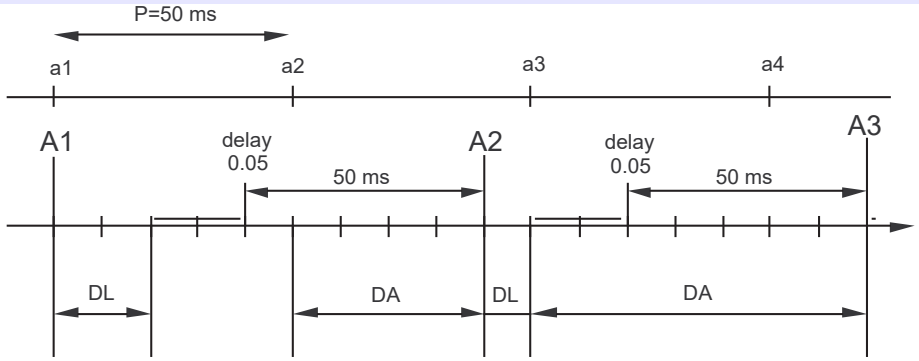
```
tarea T(milliseconds t_computo_p){
  ...
  t_computo= t_computo_p; //p.e.: 100 milisegundos
  ...

  do {

    // accion a realizar simulada
    sleep_for(T_computo);//afectada de deriva local

  while (true);
  }
```


Manuel I. Capel
manuelcapel@ugr.es

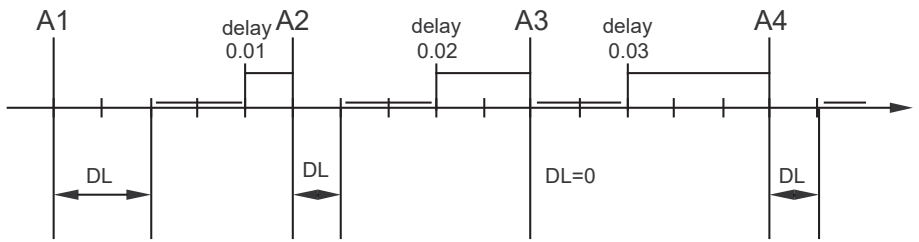


Deriva acumulativa y activación periódica de las tareas

- Problema de la *deriva acumulativa* de las tareas
- Resultaría imposible programar una tarea que se active transcurrido un periodo exacto

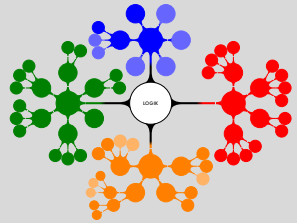


Primera aproximación



Aproximación buena

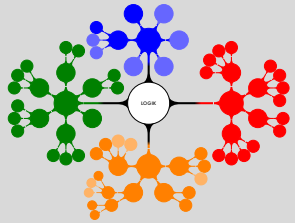
Manuel I. Capel
manuelcapel@ugr.es



Eliminación de la deriva acumulativa

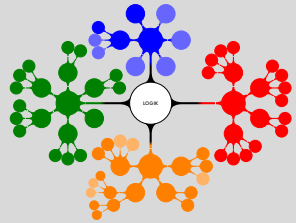
```
tarea Periodica(){
  t_ciclo= 100; //milisegundos
  siguiente_instante= clock()::now();//milisegundos
                                     //funcion del sistema
  do {
    // accion a realizar

    siguiente_instante += t_ciclo;
    sleep_until(siguiente_instante);

  } while (true);
  }
```


Manuel I. Capel
manuelcapel@ugr.es



Tareas y recursos

Manuel I. Capel
manuelcapel@ugr.es



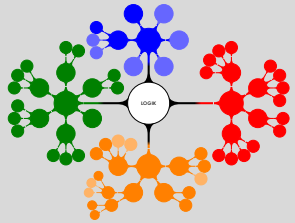
Tipos de elementos de un STR

- Tarea ($\equiv$ "proceso")
  - Sujeta a restricciones de tiempo, definidas por *atributos temporales*
  - Concepto de *activación* de una tarea de TR
  - *Instante de activación*
- Recurso (necesarios para ejecución de las tareas):
  - *Activo*: procesador, red de comunicaciones, ...
  - *Pasivo*: datos, memoria, discos, ...

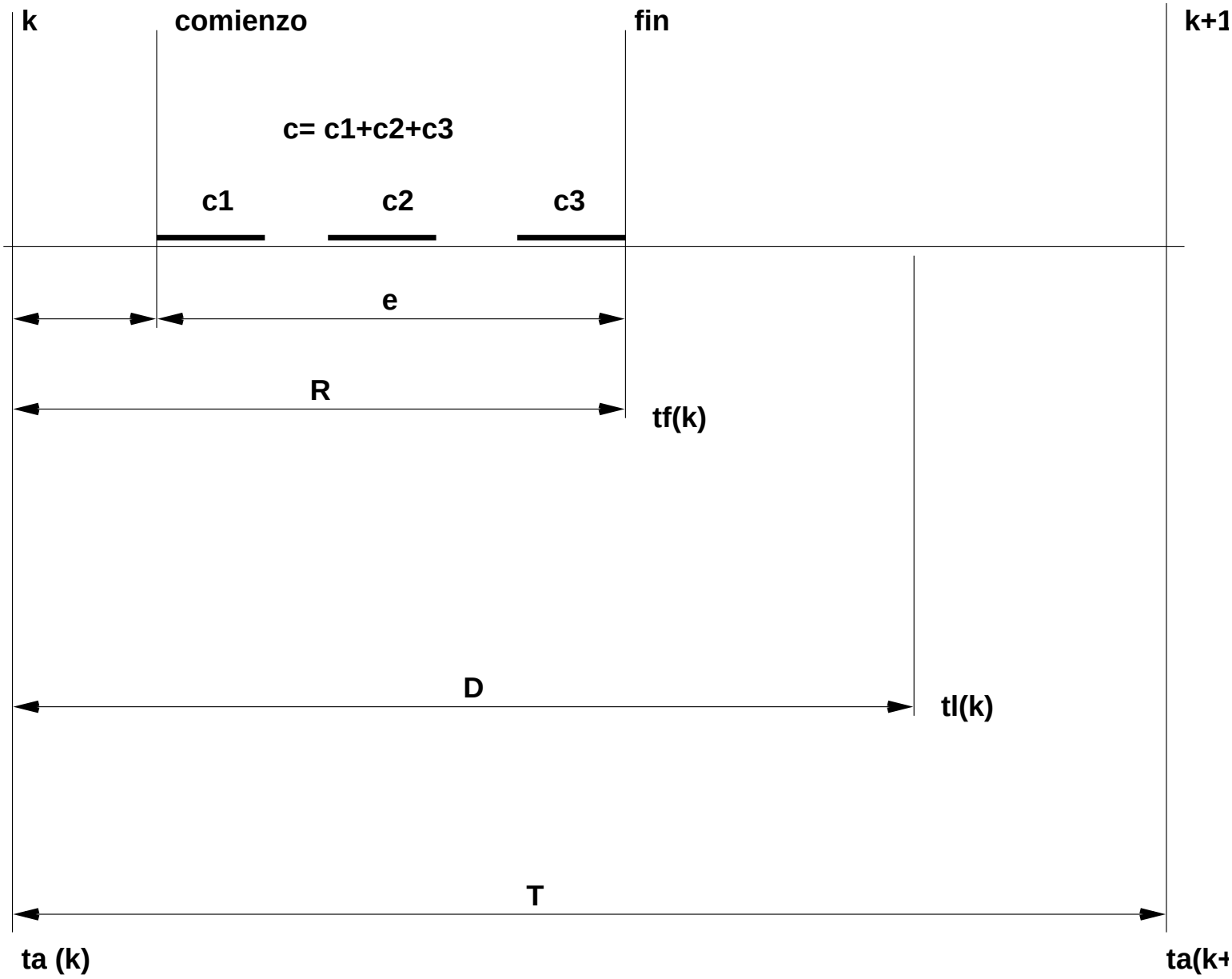
Atributos temporales principales de una tarea

- *Tiempo de cómputo máximo o de ejecución (C_i)*
- *Tiempo de respuesta (R_i)*
- *Plazo de respuesta máximo ("deadline")(D_i)*
- *Periodo (T_i)*

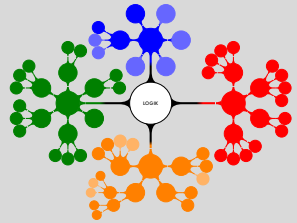
Manuel I. Capel
manuelcapel@ugr.es



Atributos temporales principales de una tarea-II

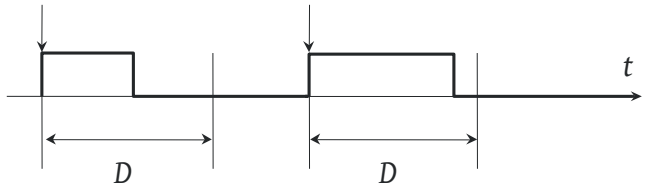


Manuel I. Capel
manuelcapel@ugr.es

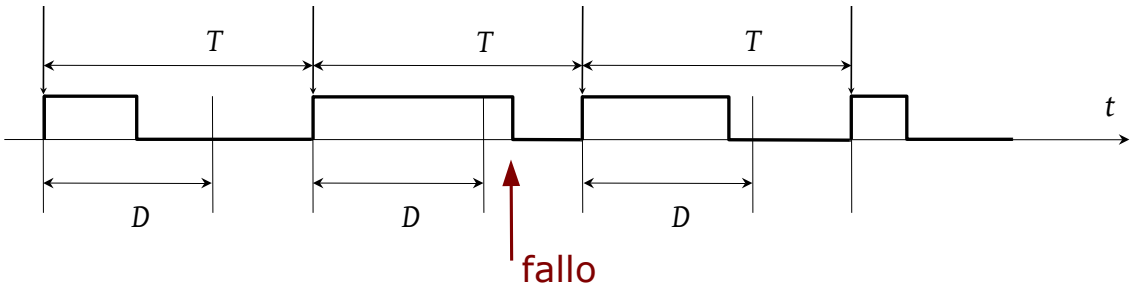


Tipos de tareas

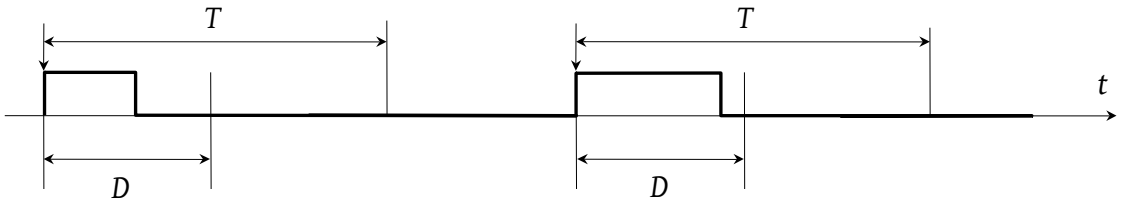
- **Aperiódica:** activación en instantes arbitrarios.



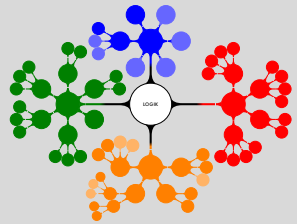
- **Periódica:** repetitiva, T es el **período** (tiempo exacto entre act.)



- **Esporádica:** repetitiva, T es intervalo mínimo entre activaciones



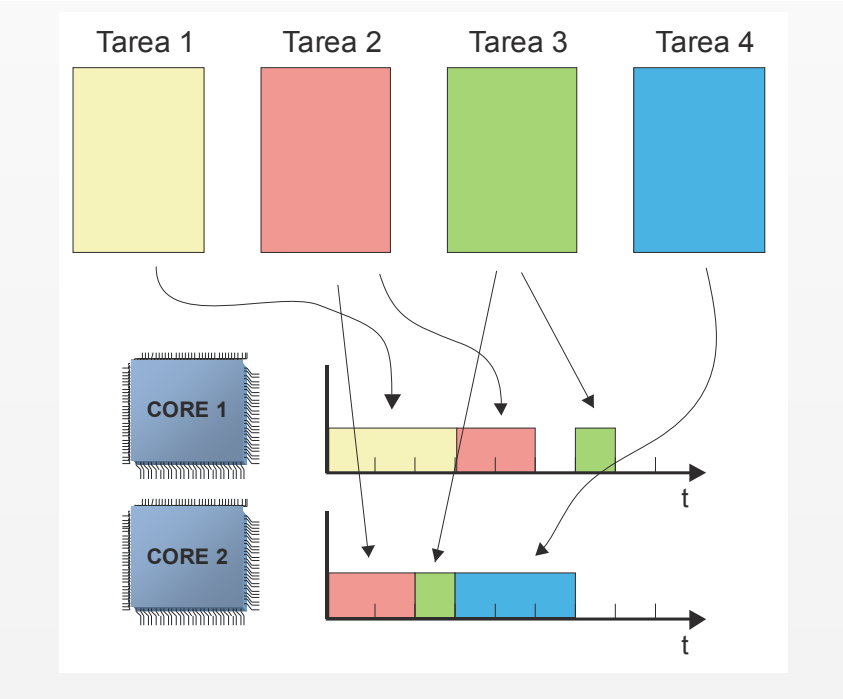
Manuel I. Capel
manuelcapel@ugr.es



Planificación de tareas

Planificación

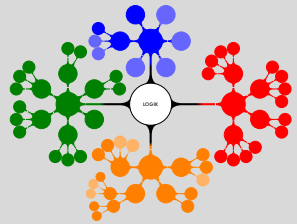
- Asignación de tareas a los recursos activos del sistema para garantizar la ejecución completa de cada tarea sujeto a un conjunto de restricciones definidas



Planificación de tareas de TR

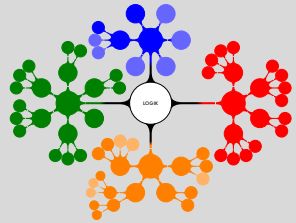
Se ha de garantizar la ejecución completa de cada tarea antes de que expire su plazo de respuesta máximo(D)

Manuel I. Capel
manuelcapel@ugr.es



Problemática de la planificación de tareas de tiempo–real

Manuel I. Capel
manuelcapel@ugr.es

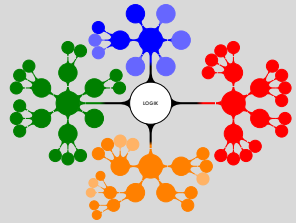


Problema central a resolver:

- **Calcular a priori si, dado un conjunto arbitrario de tareas, estas pueden ejecutarse completamente antes de que termine el plazo de respuesta máximo de cada una de ellas, en cada una de sus activaciones**
- El cálculo anterior para un conjunto de tareas, que se pueden interrumpir unas a otras varias veces durante su ejecución, no tiene una solución matemática sencilla
- Se puede “resolver"de forma *estática*
- Para resolverlo de forma *dinámica* se utiliza un *modelo simple* de tareas de TR

Planificación cíclica

Manuel I. Capel
manuelcapel@ugr.es

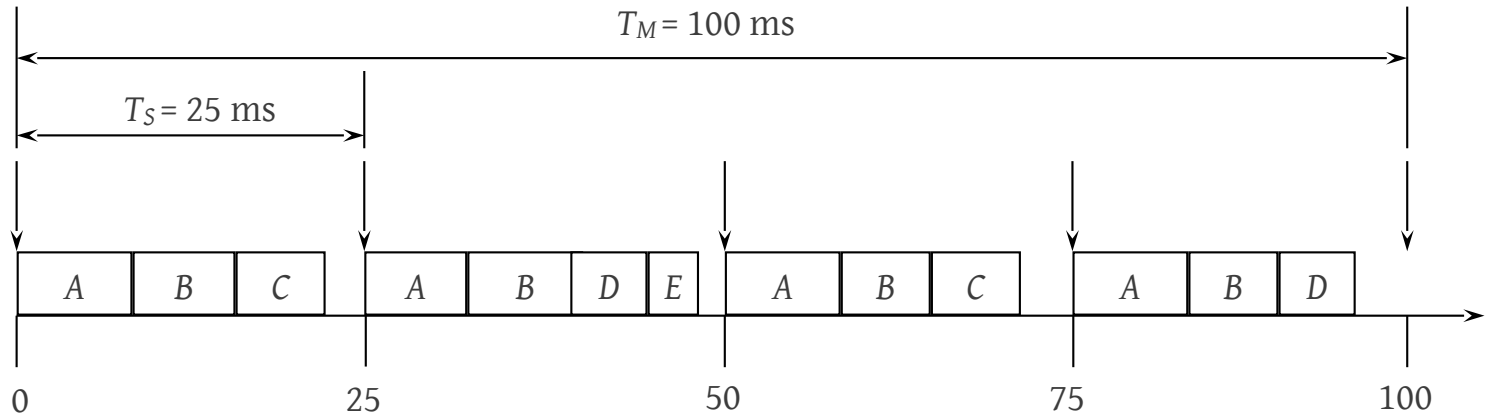


Ejecutivo cíclico

- La planificación cíclica se basa en una estructura de activación fija de las tareas periódicas
- El plan principal es un ciclo que tiene una duración constante denominado hiperperiodo (T_M):
  - $T_M = mcm(T_1, T_2, ..., T_n)$ (mínimo común múltiplo de los periodos de las tareas)
  - La unidad que cuenta el tiempo es siempre **entera** (p.e., ticks de reloj de tiempo real del computador)
  - El ciclo principal, que se repite siempre, se descompone en varios ciclos secundarios de igual duración todos ellos (T_S)

Ejecutivo cíclico

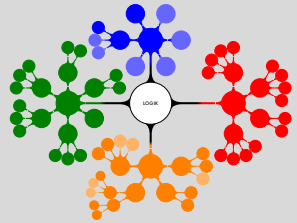
Tarea	T_i	C_i
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2



Suponemos que $D_i = T_i$ para cada tarea τ_i

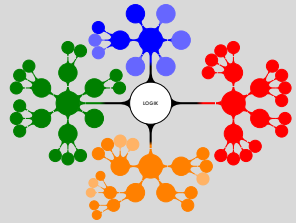
$$T_M = mcm(25, 50, 100)$$

Manuel I. Capel
manuelcapel@ugr.es



Planificación cíclica

Manuel I. Capel
manuelcapel@ugr.es



Ideas fundamentales

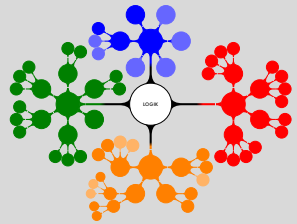
- Plan principal de ejecución de las tareas explícito y *fuera de línea* (“off line”), es decir,
  sin acceso concurrente al procesador según la prioridad de cada tarea
- Establece de una vez cómo se entrelazan las tareas en cualquier ejecución del programa, pero puede variar la ordenación relativa de las tareas en cada *ciclo secundario*
- Cada tarea se ejecuta completamente 1 sola vez dentro de cada repetición de su *periodo*
- Si una tarea se inicia dentro de una iteración del ciclo secundario, ha de terminar antes del final de dicha iteración

Implementación de la planificación cíclica

```
void EjecutivoCiclico()
{
const milliseconds tmp_secundario (25);
const int nciclos = 4 , // número de ciclos secundarios
siguiente_instante= clock::now();
int frame = 0 ; // número del siguiente ciclo secundario
  while( true ){
  for(frame= 1; i<=nciclos; frame++){ // ejecuta la tareas
       del ciclo secundario actual
    switch( frame )
    { case 0 : A(); B(); C(); break ;
      case 1 : A(); B(); D(); E(); break ;
      case 2 : A(); B(); C(); break ;
      case 3 : A(); B(); D(); break ;
    }
     siguiente_instante += tmp_secundario;
    //pasar al siguiente ciclo secundario
    sleep_until(siguiente_instante);
    // espera inicio siguiente periodo de 25 miliseg.
   }//for
  } //while
}
```

- Cada iteración del bucle `for` ejecuta un ciclo secundario
- Cada 4 iteraciones del bucle ejecutan un ciclo principal

Manuel I. Capel
manuelcapel@ugr.es



Diseño del ejecutivo cíclico

Obtención del *buen valor* de T_S:

- Restricciones a tener en cuenta:
  - El periodo del ciclo secundario será un divisor del periodo del ciclo principal:
    existirá un entero k tal que: $T_M = k \cdot T_S$
  - $T_S \geq$ el tiempo de cómputo máximo (Cw) de cualquier tarea,

$$máximo\,(C_1, C_2, ..., C_n) \leq T_S$$

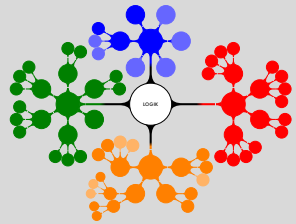
- Sugerencias para conseguirlo:
  - El ciclo secundario debe ser menor o igual que el mínimo plazo de respuesta máximo (D) de todas las tareas:

$$T_S \leq mínimo\,(D_1, D_2, ..., D_n)$$

Acotación del tamaño del ciclo secundario:

$$máximo\,(C_1, C_2, ..., C_n) \leq T_S \leq mínimo\,(D_1, D_2, ..., D_n)$$

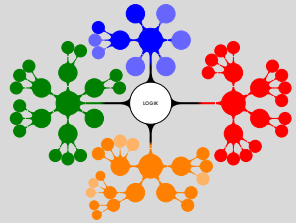
Manuel I. Capel
manuelcapel@ugr.es



Propiedades del ejecuctivo cíclico

- No hay concurrencia en la ejecución:
  - Cada ciclo secundario es una secuencia de llamadas a procedimientos (no se lanza la ejecución de las tareas)
  - No se necesita un núcleo de ejecución multitarea, ni un sistema operativo de tiempo real, etc.
- Los procedimientos acceden secuencialmente a los datos que pudieran compartirse entre las tareas:
  - No se necesitan mecanismos de exclusión mutua como los semáforos o monitores
- No hace falta analizar el comportamiento temporal del programa, es decir, se hace innecesario realizar un análisis de planificabilidad del conjunto de tareas:
  - El sistema es correcto por construcción

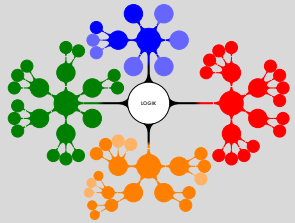
Manuel I. Capel
manuelcapel@ugr.es



Problemas del ejecutivo cíclico

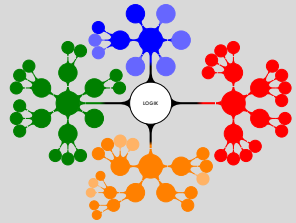
- Dificultad para incorporar tareas con periodos largos
- Las tareas esporádicas son difíciles de tratar:
  - Se podría utilizar un servidor de consulta
- El plan cíclico del proyecto es difícil de construir:
  - Si los periodos son de diferentes órdenes de magnitud el número de ciclos secundarios se hace muy grande
  - Puede ser necesario partir una tarea en varios procedimientos
- Es poco flexible y difícil de mantener:
  - Cada vez que se cambia una tarea hay que rehacer toda la planificación

Manuel I. Capel
manuelcapel@ugr.es



Esquema de planificación de tareas

Manuel I. Capel
manuelcapel@ugr.es

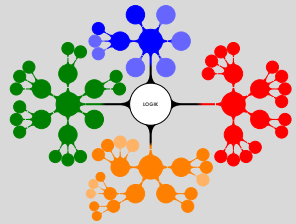


Determinación de la planificabilidad de un conjunto de tareas

- *Algoritmo de planificación*: determina el orden que acceden las tareas a los procesadores
- *Método de análisis de planificabilidad*: se basa en un test que predice si las tareas son planificables conjuntamente, sujetas a las condiciones o restricciones especificadas a priori durante:
  - todos las activaciones posibles de las tareas en su ejecución
  - la ejecución de estas suponiendo la situación más desfavorable o WCET (*Worst Case Execution Time*) en dicha ejecución para cada tarea

Modelo simple de tareas

Manuel I. Capel
manuelcapel@ugr.es

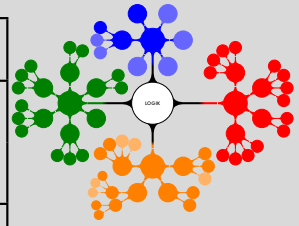


Estructura de las tareas y sus interacciones para resolver el problema de planificabilidad:

- Programa: conjunto fijo de tareas que comparten el tiempo de 1 procesador
- Tareas periódicas, con periodos conocidos
- Independientes entre sí (las tareas no se bloquean por acceder a recursos compartidos)
- Todas las tareas tienen un *plazo de respuesta máximo* (D) que coincide con su periodo (T)
- Los retrasos debidos al sistema (cambios de contexto, etc.) son ignorados
- Los *eventos* de tiempo real no se guardan Tiempo máximo de cómputo de las tareas (C_i): conocido y fijo
- No se contemplan tareas esporádicas o aperiódicas.

Modelo de tareas-II

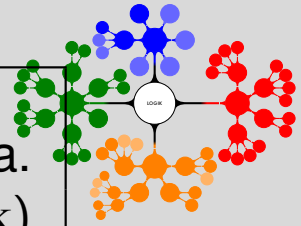
Manuel I. Capel
manuelcapel@ugr.es



Notación	Atributo temporal	Descripción
P	Prioridad	Prioridad asignada a la tarea (si fuera aplicable)
τ	Tarea	Nombre de la tarea
t_a	Tiempo de activación de la tarea (arrival time, request time, release time)	Instante en el que la tarea está para ejecución.
t_s	Tiempo de comienzo (start time)	Instante de tiempo en que comienza realmente su ejecución.
t_f	Tiempo de finalización de la tarea (finishing time)	Instante en el que la tarea finaliza su ejecución.
t_l, d	Tiempo límite (absolute deadline)	Instante de tiempo límite para la ejecución de la tarea. Es fijo: $t_l(k) = d = t_a + D$

Modelo de tareas-III

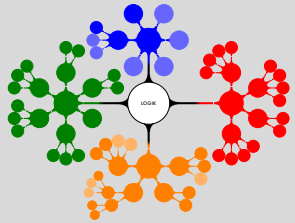
Manuel I. Capel
manuelcapel@ugr.es



T	Periodo de ejecución	Intervalo de tiempo entre dos activaciones sucesivas de una tarea periódica. Es un valor fijo, dado por $T = t_a(k+1) - t_a(k)$
J	Latencia	tarea suspendida hasta que se ejecuta: $J(k) = t_s(k) - t_a(k)$
c	Tiempo de cómputo	Tiempo de ejecución de la tarea
C	Cómputo máximo	Tiempo de ejecución de peor caso de la tarea
e	Tiempo de transcurrido	Tiempo transcurrido desde el instante de comienzo hasta la finalización de la tarea Viene dado por: $e(k) = t_f(k) - t_s(k)$
R	Tiempo de respuesta	Tiempo de la tarea hasta completar completamente su ejecución y viene dado: $R(k) = J(k) + e(k)$.

Propósito

Manuel I. Capel
manuelcapel@ugr.es



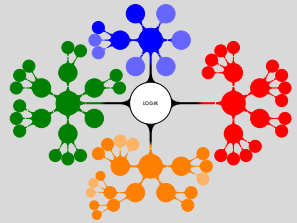
Planificación (“Task–Scheduling") de Trabajos:

Estudia un conjunto de técnicas de *Programación Entera* para obtener una asignación de recursos y tiempo a actividades de tal modo que se cumplan determinados requisitos de eficiencia.

Para conseguirlo se utiliza una heurística que intenta maximizar una función objetivo: $\mathrm{U}(\mathrm{N})$, denominada *utilización del* procesador para N tareas (de Tiempo-Real en nuestro caso).

Planificación con prioridades

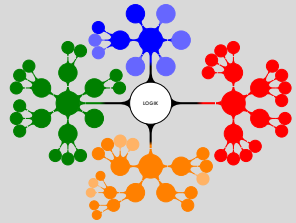
Manuel I. Capel
manuelcapel@ugr.es



- La planificación con prioridades permite solventar los problemas descritos. Cada tarea tiene asociado un valor entero positivo, llamado prioridad de la tarea:
  - Es un atributo de las tareas normalmente ligado a su importancia relativa en el conjunto de tareas
  - Por convención se asigna números enteros menores a procesos más urgentes
- La prioridad de una tarea la determina sus necesidades temporales; no es importante el rendimiento o comportamiento del sistema
- Una tarea puede estar en varios estados
- Las tareas ejecutables se despachan para su ejecución en orden de prioridad
- No confundir *urgencia* con *criticidad* de una tarea de tiempo real

Esquema de planificación de tareas

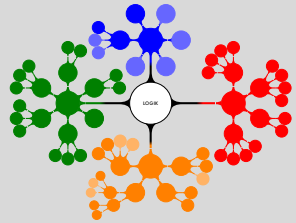
Manuel I. Capel
manuelcapel@ugr.es



1. Algoritmo para ordenar el acceso de las tareas al procesador
2. Un test para predecir el comportamiento del sistema en el *peor caso* de planificación, es decir, cuando exista mayor interferencia entre las tareas
3. Principales características de un esquema de planificación de tareas de TR:
   - Dinámico vs. estático
   - Desplazante (*preemptive*) de tareas menos prioritarias
   - Análisis de las tareas dentro de 1 “ventana temporal"
   - Concepto de **instante crítico**
4. Una posible solución al problema del análisis temporal de las tareas esporádicas y aperiódicas

Planificación con prioridades-II

Manuel I. Capel
manuelcapel@ugr.es



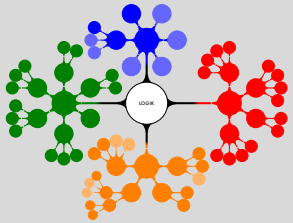
Tipos de esquemas de planificación:

- **Prioridades asignadas estáticamente a las tareas**: Cada tarea tiene asociada una prioridad fija durante toda la ejecución del sistema:
  - *Cadencia monótona* o RMS (Rate Monotonic Scheduling): prioridad a la tarea más frecuente (menor periodo de activación)
  - *Plazo de respuesta monótono* o DMS (Deadline Monotonic Scheduling):
    prioridad a la tarea más urgente, es decir, con menor plazo de respuesta máximo D_i

Modelo de tareas-IV

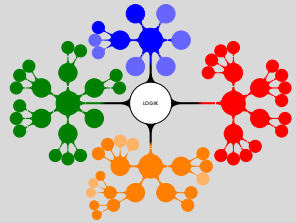
D	Plazo de respuesta máximo(relative deadline)	Define el máximo intervalo de tiempo o máximo tiempo de respuesta
ϕ	Desplazamiento o fase	Tiempo para activarse por primera vez
RJ	Fluctuación relativa o *jitter*(relative release jitter)	Máxima desviación en el tiempo de comienzo entre 2 activac. de una tarea Viene definido por: $\mathrm{RJ} = \max((\mathrm{t_s(k+1)} - t_a(k+1)) - (t_s(k) - t_a(k)))$
H	Holgura (laxity, slack time)	Tiempo de permanecer activa dentro del plazo de respuesta máximo: $\mathrm{H(k)} = \mathrm{t_l} - \mathrm{t_a(k)} - \mathrm{e(k)} = \mathrm{D} - \mathrm{e(k)}$

Manuel I. Capel
manuelcapel@ugr.es



Planificación con prioridades-III

Manuel I. Capel
manuelcapel@ugr.es

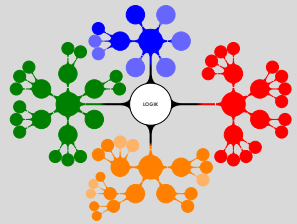


Tipos de esquemas de planificación:

- **Prioridades asignadas dinámicamente a las tareas**: Las prioridades de las tareas cambian durante la ejecución del sistema:
  - *Primero la de tiempo límite más cercano* o EDF: (Earliest Deadline First):
    prioridad a la tarea que tenga siguiente plazo de respuesta absoluto (absolute deadline) más próximo a expirar
  - Primero la de menor holgura temporal o LLF (Least Laxity First):
    prioridad a la tarea que le queda menos tiempo durante el que puede permanecer *activa* (antes de que expire su plazo de respuesta máximo)

Esquema de planificación de tareas de TR basado en el algoritmo de *cadencia monótona*

Manuel I. Capel
manuelcapel@ugr.es



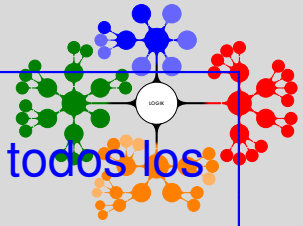
1. Algoritmo de cadencia monótona (*Rate Monotonic Scheduling*)(RMS)
   - Prioridad estáticamente asignada a cada tarea (τ_i), que es tanto mayor cuanto menor es su periodo (T_i)

$$\forall i,j:\ T_i < T_j \Rightarrow P_i > P_j$$

   - No se atiende a la *criticidad* de las tareas, sino a su *urgencia* ($= \frac{1}{T_i}$)
   - Este algoritmo es óptimo entre los algoritmos de asignación estática de prioridades si las tareas son periódicas y el plazo de respuesta coincide con el periodo ($D_i = T_i$)
2. El **test de planificabilidad** se basa en la suma del **Factor de Utilización** del procesador para cada tarea:

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

Manuel I. Capel
manuelcapel@ugr.es



1. Test de planificabilidad de un conjunto de tareas $\{\tau_1, \tau_2, \ldots \tau_N\}$:

En un sistema de N tareas periódicas, independientes, con prioridades asignadas en orden de su frecuencia (e inverso al periodo), se cumplen todos los tiempos límite de las tareas, para cualquier desfase (ϕ_i) inicial de cada tarea si el factor de*utilización del procesador* ($U(N)$) no supera la *utilización prefijada máxima* ($U_0(N)$) para ese número de tareas N:

$$U = \sum_{i=1}^{N} (\frac{C_i}{T_i}) < U_0 = \quad N \times (2^{\frac{1}{N}} - 1)$$

- Un conjunto de tareas de tiempo real pasa el test si el factor de utilización es menor o igual que el factor máximo:

$$U \leq U_0(N)$$

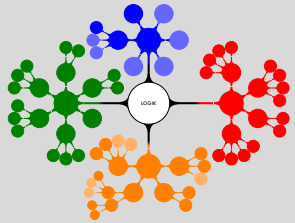
- En este caso el sistema es planificable. En caso contrario, no se puede afirmar nada (es sólo *condición suficiente*).

Inexactitud del test de planificabilidad RM

N	límite utilización
1	100
2	82,85 %
3	78,0 %
4	75,7 %
5	74,3 %
10	71,8 %
$\rightarrow \infty$	69,3 %

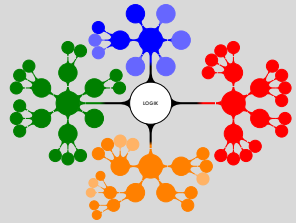
El valor del límite de utilización máxima $\mathrm{U}_0(\mathrm{N})$ del procesador depende del número de tareas N

Manuel I. Capel
manuelcapel@ugr.es



Condición suficiente de planificabilidad

Manuel I. Capel
manuelcapel@ugr.es



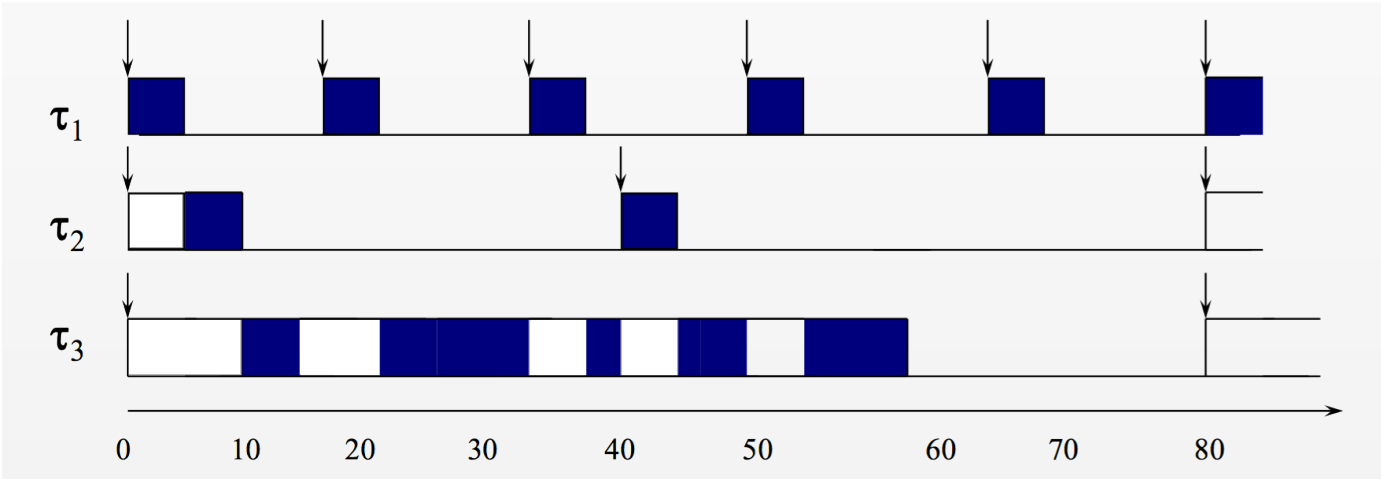
Características del test RMS

- Se dice que el test anterior es sólo una condición suficiente para determinar si un conjunto de tareas es planificable
- Se utilizan diagramas de Gantt con una ventana temporal M_i = **m.c.m.**$\{T_1\ T_2\ ...\ T_i\}$ para comprobar si un conjunto de tareas que no ha pasado el test anterior aún puede resultar planificable
- Los tests de planificabilidad basados en la utilización del procesador no proporcionan información acerca del tiempo de respuesta de la tareas

Ejemplo de RMS con el test de Liu y Layland

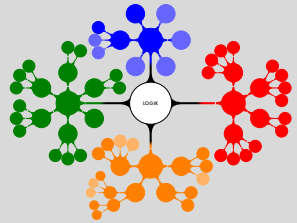
Tarea	C	D	T	C/T
τ_1	4	16	16	0,250
τ_2	5	40	40	0,125
τ_3	32	80	80	0,400

El sistema pasa el test: $U = 0,775 \leq 0,779 = U_0(3)$



(No sería necesario realizar el diagrama de Gantt en este caso)

Manuel I. Capel
manuelcapel@ugr.es



Ejemplo de RMS sin pasar el test de Liu y Layland

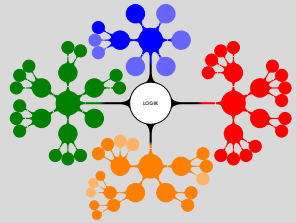
Tareas	C	D	T
Tarea 1	10	30	30
Tarea 2	10	40	40
Tarea 3	10	50	50

Calculamos U y lo comparamos con $U_0(3)$:

$$U = \sum_{i=1}^{N}(\frac{C_i}{T_i}) = \frac{10}{30} + \frac{10}{40} + \frac{10}{50} = 0,783333 \not< 0,779 = U_0(3)$$

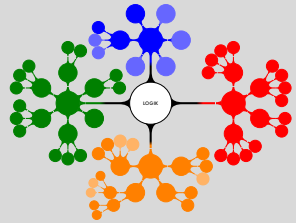
- El sistema no pasa el test
- No podemos decir si es planificable o no lo es
- Intentamos el diagrama de Gantt

Manuel I. Capel
manuelcapel@ugr.es

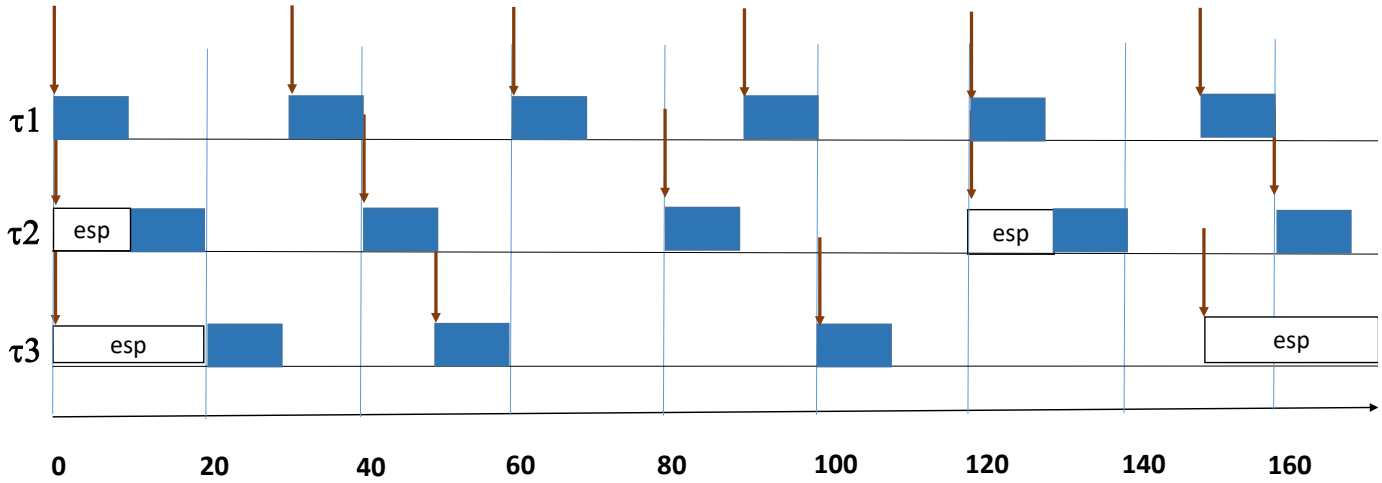


Ejemplo de RMS sin pasar el test de Liu y Layland-II

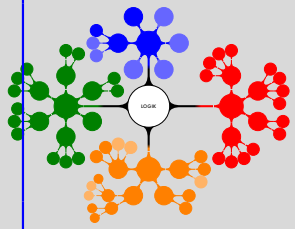
Manuel I. Capel
manuelcapel@ugr.es



- Para analizar la planificabilidad del sistema con dichas restricciones, hay que hacer el cronograma y verificar que:
  - para cada tarea i, se cumple el plazo de respuesta: R < Di
  - esto se debe verificar para un hiperperiodo (después se repite). En el ejemplo, se debería hacer el cronograma completo hasta t = 600 (= mcm{30, 40, 50}), aquí vemos una primera parte (hasta t = 160).



Manuel I. Capel
manuelcapel@ugr.es



Segundo teorema de planificabilidad de un conjunto de tareas $\{\tau_1, \tau_2, \ldots \tau_N\}$: periódicas

En un sistema de N tareas periódicas, independientes, con prioridades asignadas en orden de su frecuencia, se cumplen todos los tiempos límite de las tareas, si cuando se activan todas ellas simultáneamente, cada tarea acaba antes de que expire el tiempo límite que tiene asignado en su primera activación.

Con el test anterior se puede llegar a una utilización del procesador máxima:

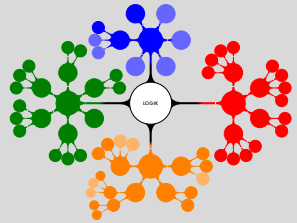
$$\Sigma_{i=1}^{N}(\frac{C_i}{T_i}) \leq 1$$

- La cota máxima que puede alcanzar U es ahora mayor:
  - puede aplicarse a más conjuntos de tareas que con la planificación RMS porque éstas pueden tener mayor factor de utilización de procesador y, sin embargo, pasan el test
- El test ahora es *exacto*: expresa una condición necesaria y suficiente para que un conjunto de tareas sea planificable

Ventajas de los esquemas de planificación estáticos

- Un esquema estático resulta ser más sencillo y eficiente de implementar
- Resulta más sencillo que diseñar un ejecutivo de tareas con tiempos límite (*plazos de respuesta absolutos*) calculados exactamente
- Permite incorporar otros factores que influyen en la planificación de un conjunto de tareas cuando sus prioridades no están asociadas a un tiempo límite
- Durante sobrecarga transitoria, un esquema de planificación estático normalmente resulta ser predicible. Lo cual no ocurre necesariamente si se utiliza un esquema dinámico (EDF)

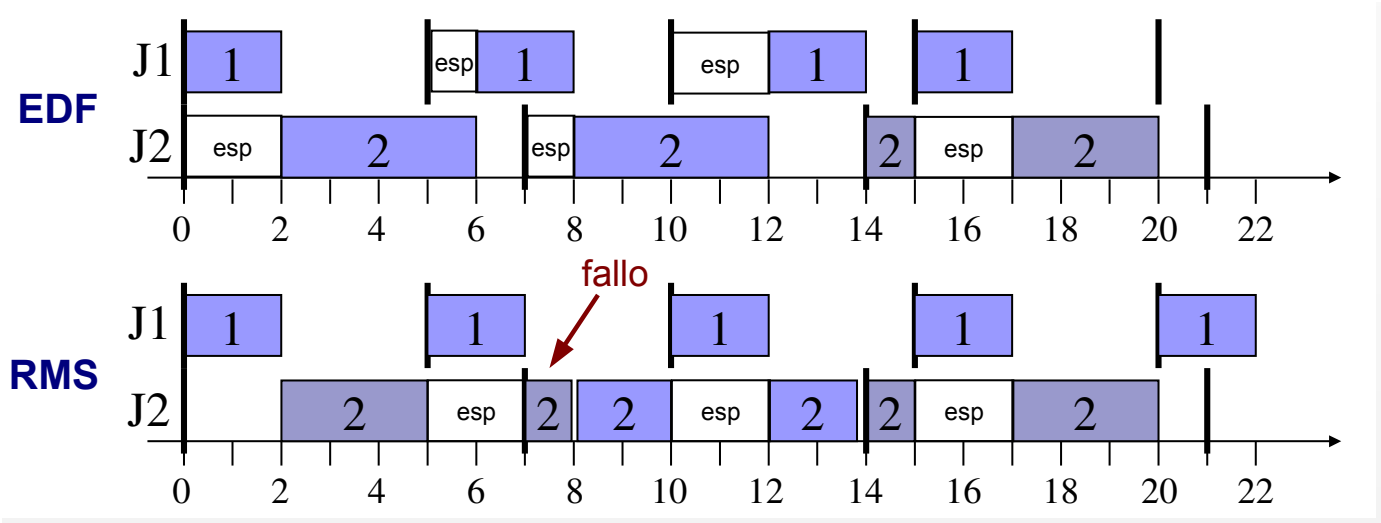
Manuel I. Capel
manuelcapel@ugr.es



Ejemplo de planificación dinámica: EDF

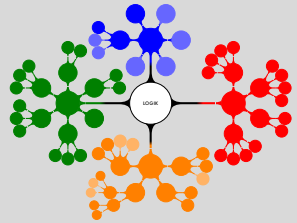
Tarea	C	D	T
χ_1	2	5	5
χ_2	4	7	7

El sistema pasa el segundo test de Liu & Layland (EDF): $U = \frac{2}{5} + \frac{4}{7} = 0{,}971 < 1{,}0$, pero no se cumple la condición que establece el segundo teorema para poder asegurar su planificabilidad



Se verifica que el sistema es planificable con EDF, pero no con RMS

Manuel I. Capel
manuelcapel@ugr.es

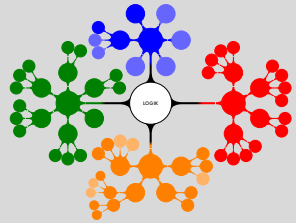


Modelo general de tareas de TR

Estructura general que se puede asumir para resolver el problema de planificabilidad:

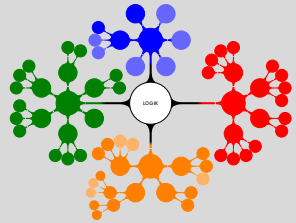
- Programa: conjunto fijo de tareas que comparten el tiempo de 1 procesador, pero pueden aparecer tareas aperiódicas o esporádicas
- Las tareas pueden bloquearse por acceder a recursos compartidos (por ejemplo, comparten semáforos en su código)
- Las tareas tienen un *plazo de respuesta máximo* (D) que, en general, no coincide con su periodo (T) Por ejemplo las tareas esporádicas suelen tener un plazo de respuesta máximo (D) muy corto
- Los retrasos debidos al sistema (cambios de contexto, etc.) son ignorados
- Los *eventos* de tiempo real no se guardan. Tiempo máximo de cómputo de las tareas (C): conocido y fijo.

Manuel I. Capel
manuelcapel@ugr.es



Inversión de prioridad

Manuel I. Capel
manuelcapel@ugr.es

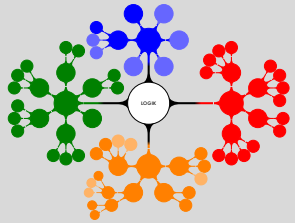


En qué consiste el problema:

- La tarea más prioritaria del conjunto espera durante un tiempo arbitrariamente largo porque se ejecutan continuamente tareas menos prioritarias, mientras la primera se mantiene bloqueada
- Si se produce la denominada *inversión de prioridad*, invalida cualquier previsión sobre la planificación del conjunto de tareas, incluso si han pasado el test RMS
- Se produce debido al esquema estático de asignación de prioridades a las tareas
- La inversión de prioridad no puede ser eliminada completamente, cuando se produce, pero sus efectos adversos sobre la planificación pueden ser minimizados

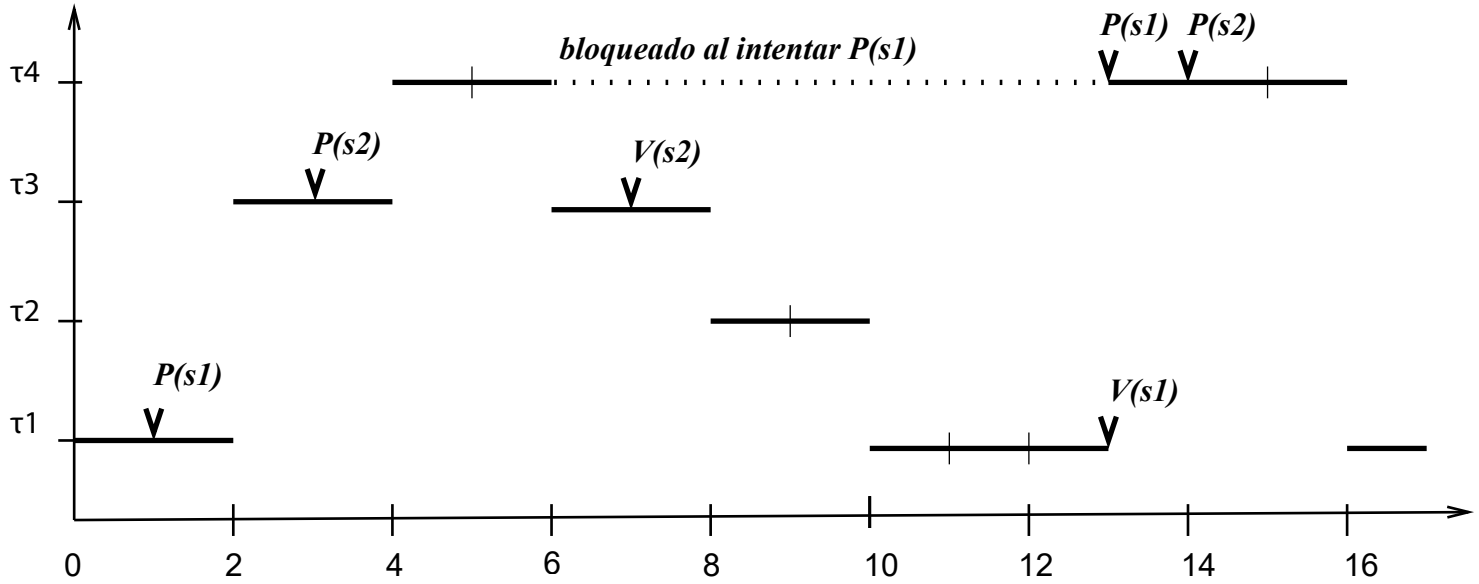
Escenario con Inversión de la prioridad de tareas

Manuel I. Capel
manuelcapel@ugr.es



Ejemplo de "fuerte" inversion de prioridad de la tarea t4

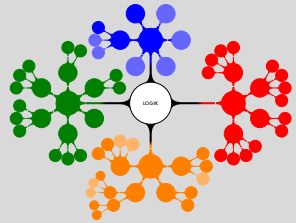
	Ci	Pi	Ii
τ4	5	1	4
τ3	4	2	2
τ2	2	3	2
τ1	6	4	0



Representación de la inversión de prioridad entre tareas periódicas con prioridades estáticas

Sección crítica no expulsable

Manuel I. Capel
manuelcapel@ugr.es



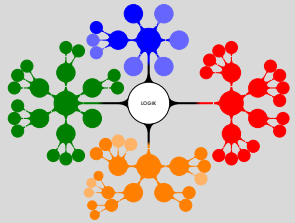
En qué consiste este protocolo:

- Las secciones críticas se ejecutan con una prioridad estática igual a la prioridad máxima del sistema

Características:

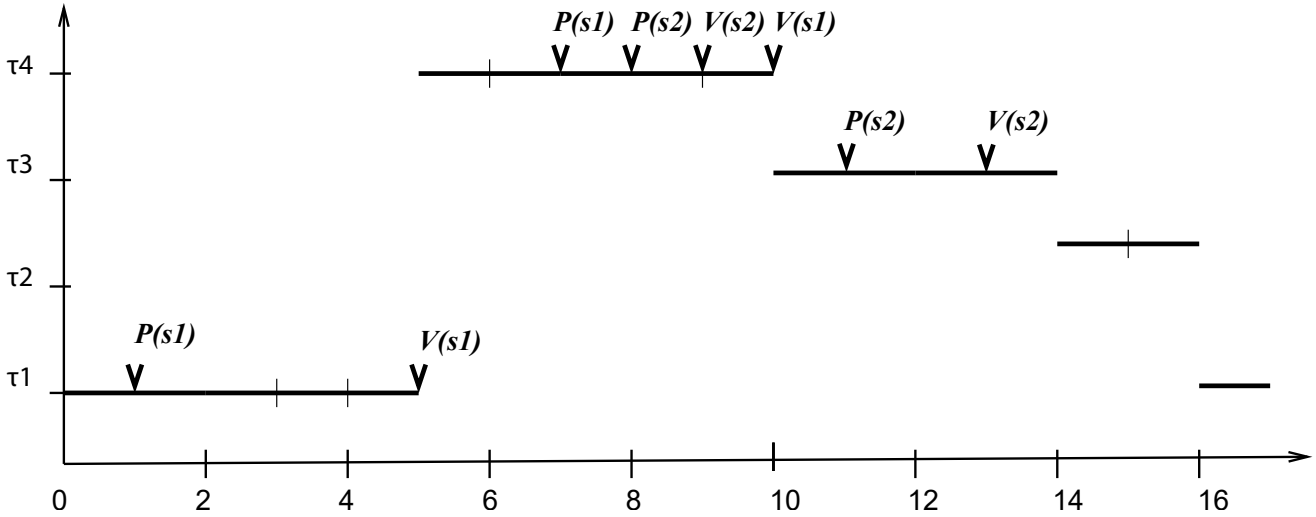
- Es el protocolo más simple para evitar la *inversión de prioridad*
- Puede inducir bloqueos excesivamente largos de las tareas más prioritarias
- Este protocolo puede llegar a interferir en (es decir, impedir) la ejecución de todas las tareas, incluso si no llegan a hacer uso de los recursos compartidos

Manuel I. Capel
manuelcapel@ugr.es



Ejemplo de seccion critica "no−expulsable"

	Ci	Pi	Ii
τ4	5	1	4
τ3	4	2	2
τ2	2	3	2
τ1	6	4	0



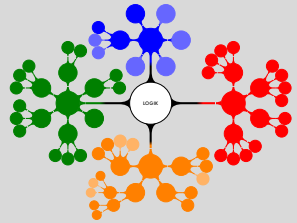
Representación de tareas que usan la sección crítica no expulsable

Nota:

prioridad del sistema= prioridad (τ_4)= 1 (máxima)

Tiempo de bloqueo de la sección crítica no expulsable

Manuel I. Capel
manuelcapel@ugr.es



- Para la tarea τ_i, su tiempo de ejecución de peor caso se incrementará en un factor constante:

$$C_i^* = C_i + B_i$$

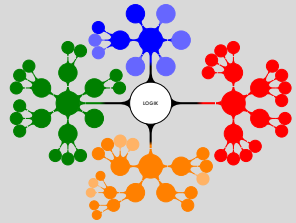
- La tarea más prioritaria τ_i puede ser bloqueada, como máximo, durante la ejecución de 1 sección crítica:

$$B_i = max_{j,\, j>i}(max_k(Dur(s_{jk}))$$

Donde s_{jk} es la sección crítica *k* ejecutada por la tarea τ_j menos prioritaria que τ_i y $Dur(s_{jk})$ es la duración de dicha sección crítica.

Herencia de prioridad

Manuel I. Capel
manuelcapel@ugr.es



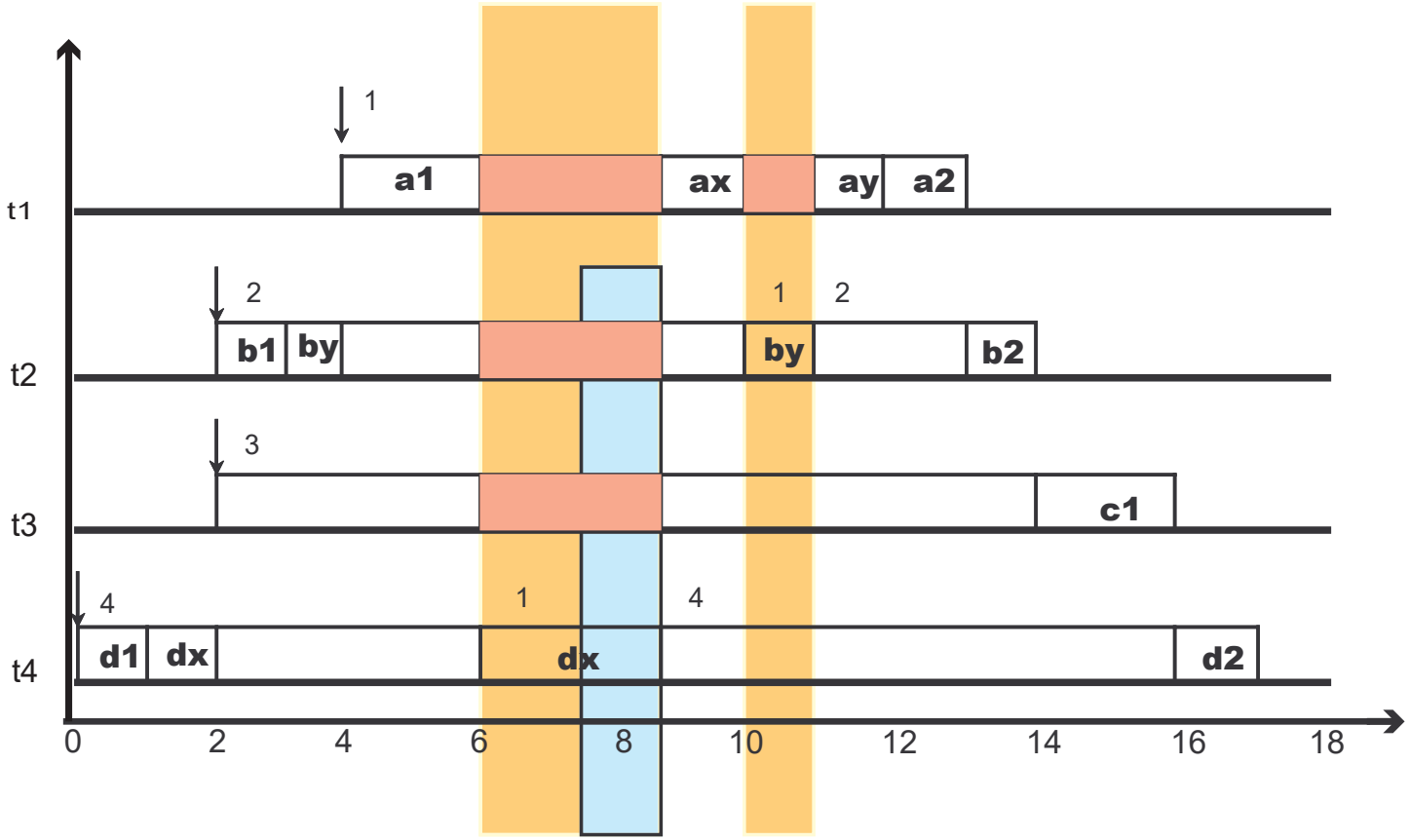
En qué consiste este protocolo:

- La prioridad efectiva de una tarea del programa será el máximo entre su prioridad por defecto y las prioridades de las demás tareas con las que comparte recursos y que tiene bloqueadas en ese momento

Características:

- Las prioridades de las tareas no se mantienen estáticas durante todo el programa
- De esta forma, se consigue minimizar el efecto de la inversión de prioridad y se optimiza el aprovechamiento del tiempo del procesador

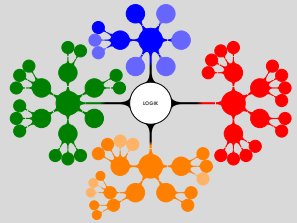
Herencia de prioridad



Tareas con el protocolo de *herencia de prioridad*
Nota: los valores de las prioridades son: $P_i = 1$ (más prioridad)...
$P_i = 4$(menos prioridad)

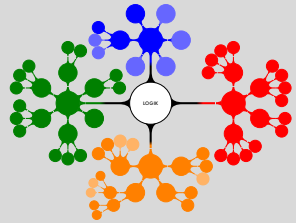
- `t4`: posee una sección crítica (**x**) de 4 unidades de tiempo
- `t1`: posee 2 secciones críticas: (**x**) e (**y**), ambas de 1 unidad temporal
- `t2` sufre un *bloqueo indirecto* y posee una sección (**y**) de 2 unidades temporales

Manuel I. Capel
manuelcapel@ugr.es



Tipos de Bloqueos con el Protocolo de *Herencia de Prioridad*

Manuel I. Capel
manuelcapel@ugr.es



Tipos de bloqueos que se pueden dar con este protocolo:

(a) Bloqueos directos,

(b) Bloqueos indirectos

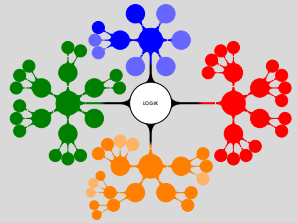
Características de este protocolo

Las tareas sólo pueden verse bloqueadas un número limitado de veces por otras menos prioritarias, en consecuencia:

1. Si una tarea tiene definidas en su código M secciones críticas, entonces el número máximo de veces que puede verse bloqueada durante su ejecución es M.
2. Si hay sólo $N < M$ tareas menos prioritarias, el máximo número de bloqueos que puede experimentar la tarea más prioritaria se reducirá a N.

Cálculo del Factor de Bloqueo

Manuel I. Capel
manuelcapel@ugr.es



El factor de bloqueo vendrá dado como el mínimo entre 2 términos:

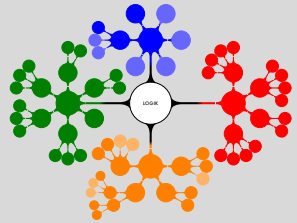
1. B_i^l: bloqueo debido a tareas τ_j menos prioritarias, que acceden a secciones críticas *k* compartidas con tareas más prioritarias [1]:
   - $B_i^l = \sum_{j=i+1}^{n} \max_k[Dur_{j,k} : Limite(S_k) \geq P_i]$
2. B_i^s: bloqueo debido a todas las secciones críticas a las que accede la tarea τ_i,
   - $B_i^s = \sum_{k=1}^{m} \max_{j>i}[Dur_{j,k} : Limite(S_k) \geq P_i]$

   Factor de bloqueo de la tarea τ_i como: $B_i = Min(B_i^l, B_i^s)$.

[1] aunque no necesariamente la comparten con la tarea τ_i que estamos considerando, ya que su prioridad se podría elevar indirectamente.

Protocolos de Techo de Prioridad

Manuel I. Capel
manuelcapel@ugr.es



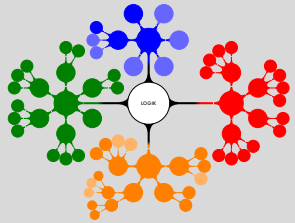
Techo de prioridad de un recurso: es la prioridad de la tarea más prioritaria que puede bloquear al recurso

- Estos protocolos garantizan que una tarea prioritaria sólo puede ser bloqueada como máximo 1 vez por otras de menor prioridad
- Se previenen los interbloqueos
- También los bloqueos transitivos
- Se asegura el acceso en exclusión mutua a los recursos compartidos

Protocolo de techo de prioridad inmediato (PPP)

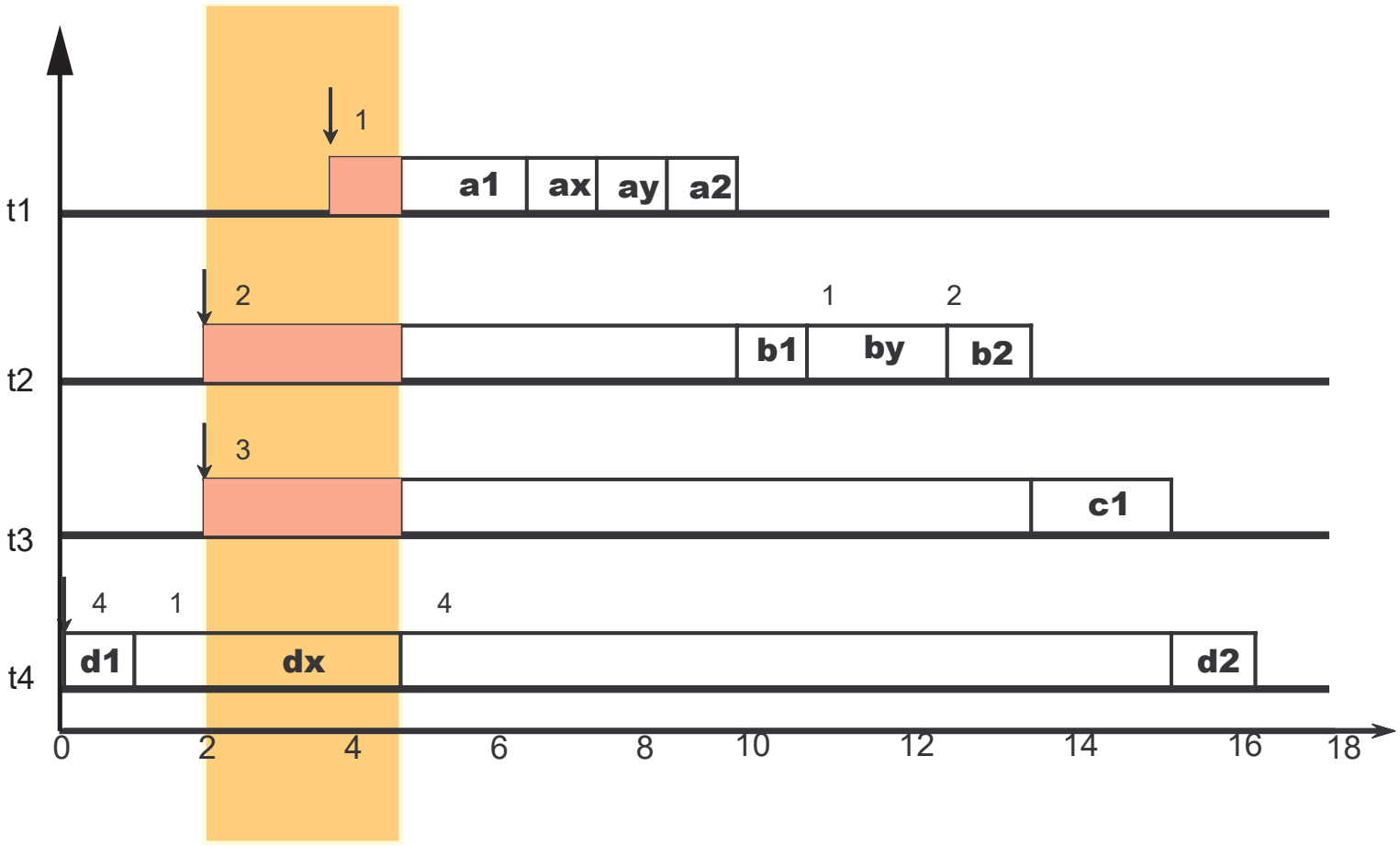
1. Cada tarea tiene una prioridad estática asignada por defecto.
2. Cada recurso tiene un valor de techo de prioridad definido.
3. Una tarea tiene una prioridad dinámica que será el máximo entre su propia prioridad estática inicial y los valores de los techos de prioridad de los recursos que mantenga bloqueados.

Manuel I. Capel
manuelcapel@ugr.es



Protocolo de Techo de prioridad

Manuel I. Capel
manuelcapel@ugr.es

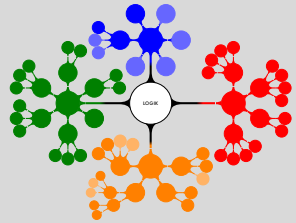


Tareas con el protocolo de *techo de prioridad inmediato*

- `t4`: posee una sección crítica (**x**) de 4 unidades de tiempo

Tiempo de bloqueo

Manuel I. Capel
manuelcapel@ugr.es



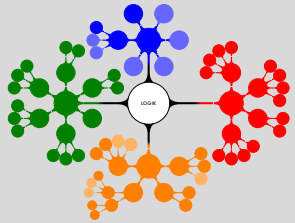
- El valor máximo del tiempo de bloqueo de una tarea es igual a la duración de la sección crítica más larga a las que acceden las tareas de prioridad inferior y que posea un techo de prioridad no inferior a la prioridad de la tarea en cuestión:

$$B_i = Max_{\{j,k\}} \{Dur_{j,k} \mid prio(\tau_j) < prio(\tau_i), techo\_prioridad(S_k) \geq prio(\tau_i)\}$$

- Con el protocolo PPP, una tarea puede verse bloqueada por otra menos prioritaria, aunque no accedan a recursos comunes

Asignación de Prioridades a las Tareas Aperiódicas

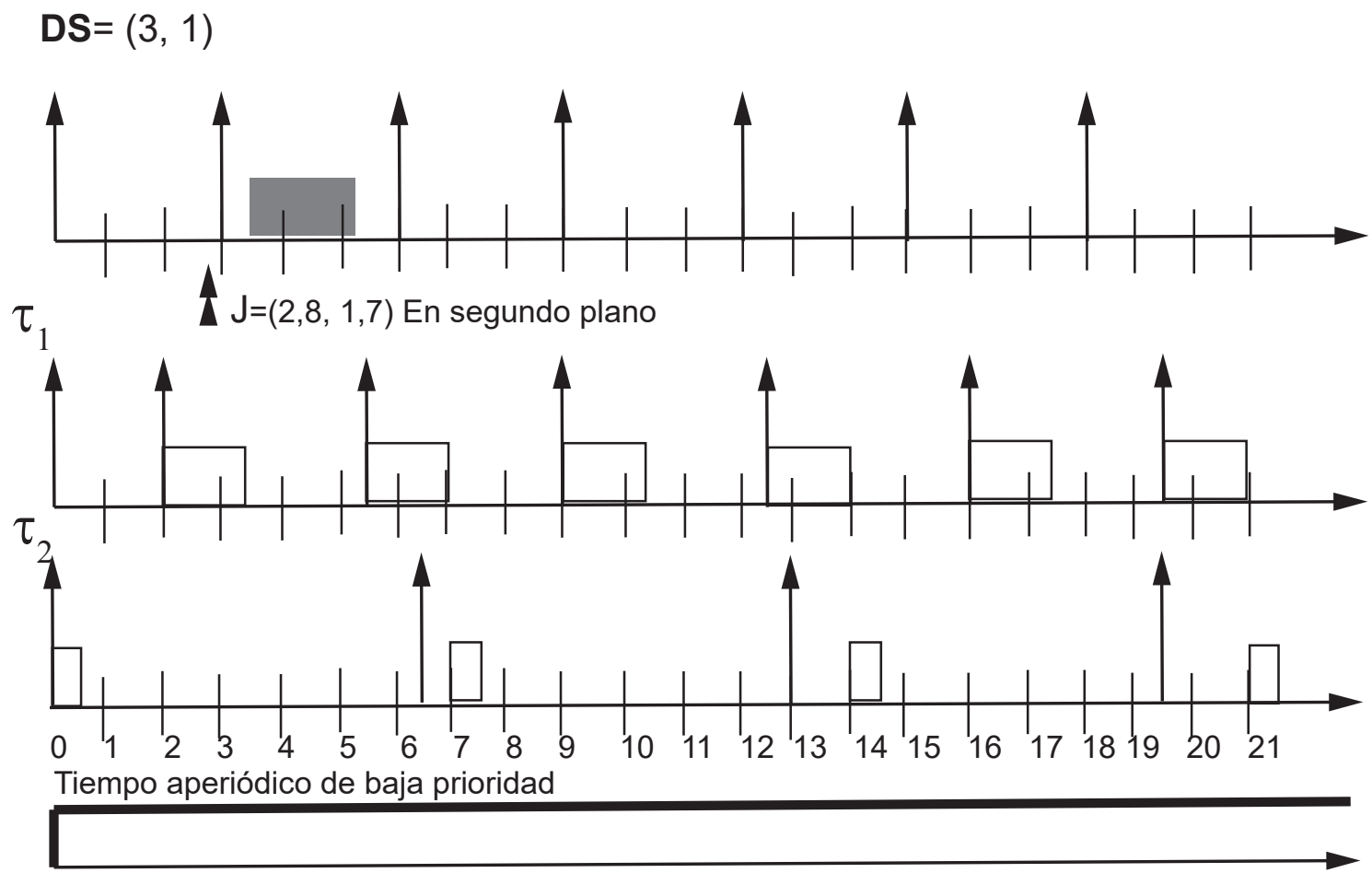
Manuel I. Capel
manuelcapel@ugr.es



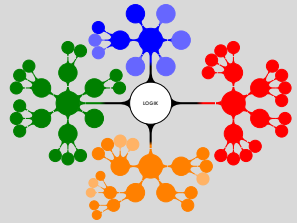
- A las tareas aperiódicas en una aplicación de tiempo real no se les puede asignar una prioridad inferior a la de las tareas de *misión crítica*
- Utilización de un *servidor aperiódico*, que puede ser una tarea real o conceptual, que permita ejecutar a los procesos aperiódicos tan pronto como sea posible

Servicio de aperiódicas con tiempo en segundo plano

- El tiempo aperiódico se incrementa a intervalos regulares
- Tiene la prioridad más baja
- Cuando no hay peticiones aperiódicas se perderá

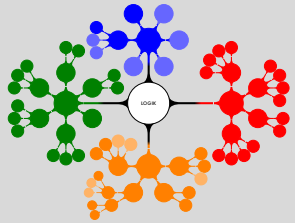


Manuel I. Capel
manuelcapel@ugr.es



Servicio de Aperiódicas con Sondeos

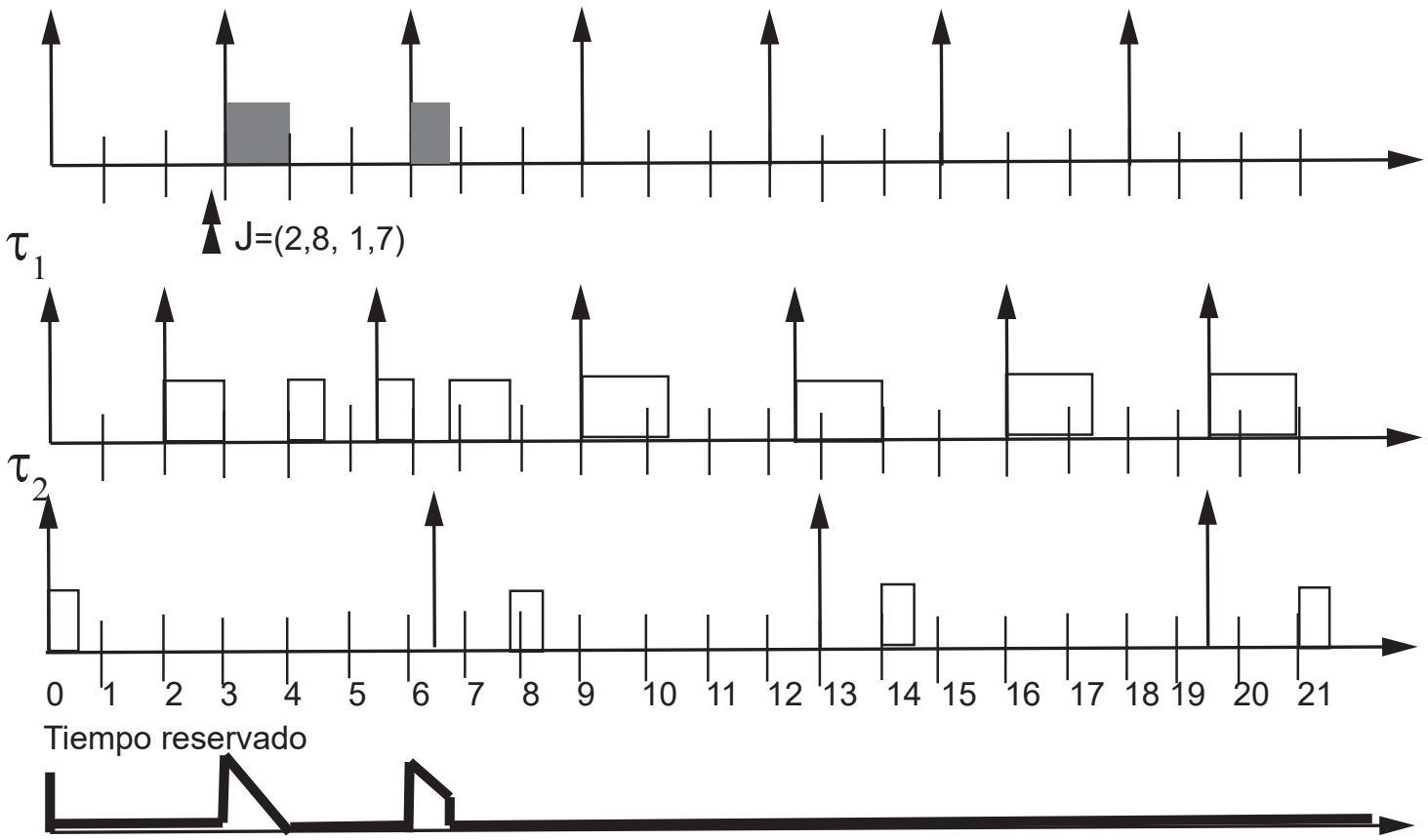
Manuel I. Capel
manuelcapel@ugr.es



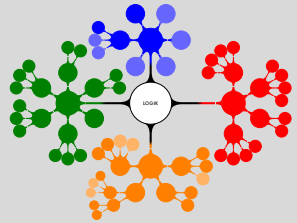
- Se añade una tarea *sondeante* a las tareas periódicas, con periodo T_s y tiempo de ejecución de peor caso: C_s
- Se le asigna la prioridad (que convenga)
- Se aplica el test de planificabilidad de cadencia monótona incluyendo la tarea sondeante:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} + \frac{C_s}{T_s} \leq (N+1)[2^{\frac{1}{(N+1)}} - 1]$$

Tareas Aperiódicas y Tarea Sondeante



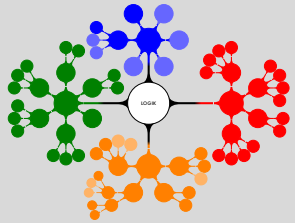
Manuel I. Capel
manuelcapel@ugr.es



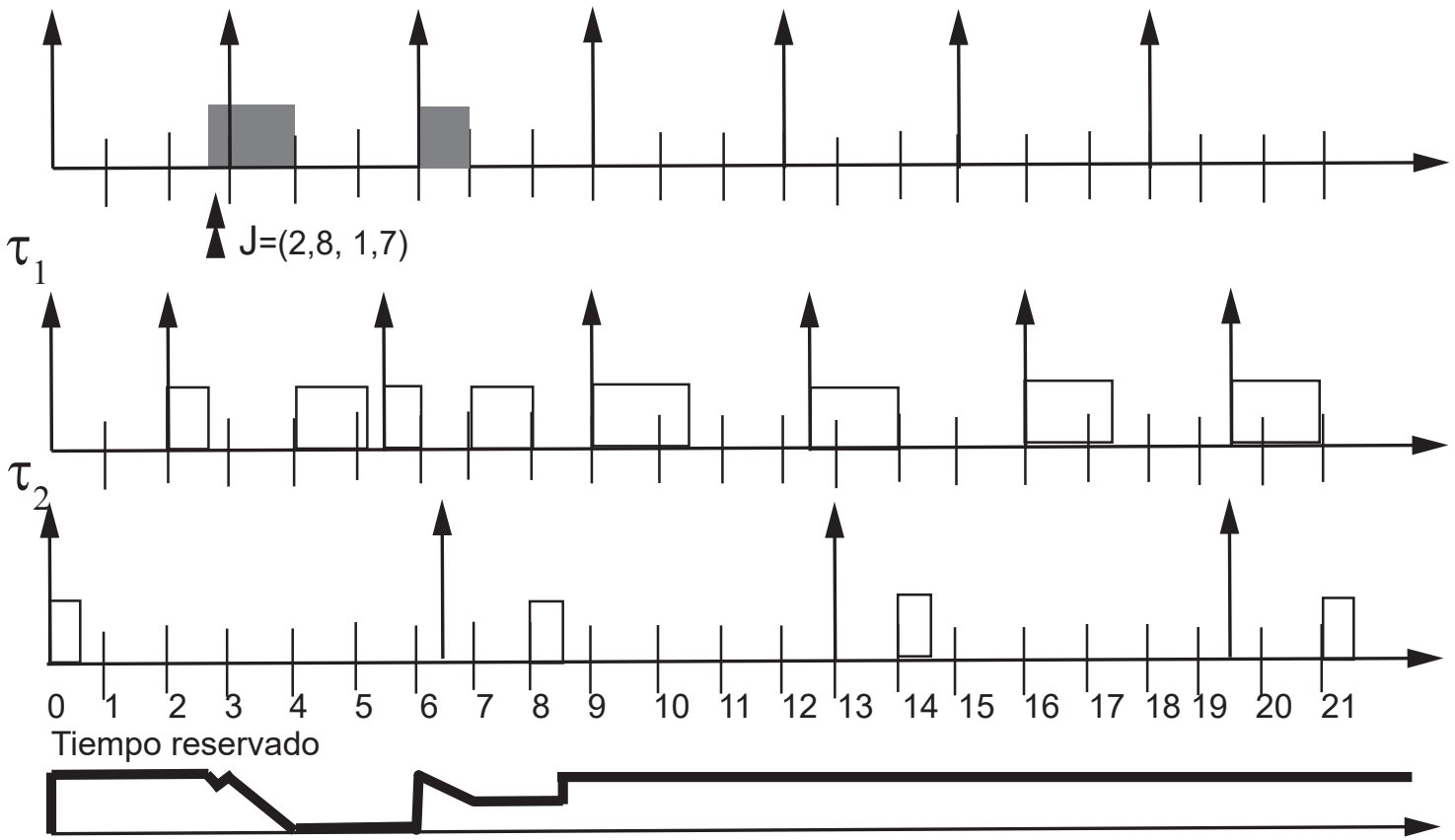
Servidor Diferido

- Preserva el tiempo aperiódico, incluso en ausencia temporal de peticiones de este tipo
- Se asigna un tamaño de servidor C_s que se gasta sólo en atender peticiones periódicas
- El tamaño del servidor se rellena hasta su valor máximo en cada periodo del servidor T_s
- El valor inicial de C_s se determina realizando un análisis previo de planificabilidad del conjunto de tareas
- Se atienden las peticiones aperiódicas a un nivel de prioridad alto siempre que el tiempo de servidor no se haya agotado

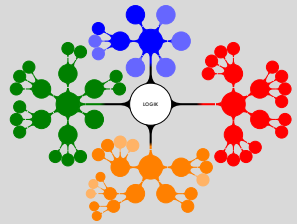
Manuel I. Capel
manuelcapel@ugr.es



Peticiones aperiódicas y Servidor Diferido

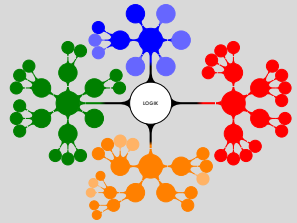


Manuel I. Capel
manuelcapel@ugr.es



Análisis de Planificabilidad con el *Servidor Diferido*

Manuel I. Capel
manuelcapel@ugr.es



- Conjunto de *N* tareas periódicas, $\tau_1 \ldots \tau_N$ y un *servidor diferido* con prioridad más alta.
- Se calcula el *menor límite superior de utilización*:

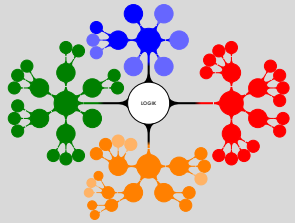
$$U_{mls} = U_s + N\left[\left(\frac{U_s + 2}{2U_s + 1}\right)^{\frac{1}{N}} - 1\right]$$

- Para $N \to \infty$, en el peor caso:

$$\lim_{N \to \infty} U_{mls} = U_s + \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$$

Análisis de Planificabilidad con el *Servidor Diferido*

Manuel I. Capel
manuelcapel@ugr.es



- Dado un conjunto de *N* tareas periódicas y un *servidor diferido* con límites de utilización U_p y U_s, respectivamente, la planificabilidad del conjunto de tareas periódicas está garantizada, con el algoritmo de cadencia monótona, si $U_p + U_s \leq U_{mls}$;

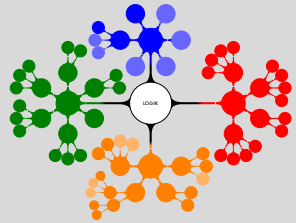
- De forma equivalente, si se cumple la desigualdad siguiente:

$$U_p \leq \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$$

.

Bibliografía

Manuel I. Capel
manuelcapel@ugr.es



Para más información, ejercicios, bibliografía adicional:

- “Programación Concurrente y de Tiempo Real”: M.I.Capel (2022), Garceta (Madrid). Sistemas de Tiempo Real (capítulo 4)
- “Sistemas de Tiempo Real”: Burns. Addison-Wesley (2003)
- “Hard real-time computing systems: predictable algorithms and applications”: Butazzo. Springer-Verlag (2005)
- “Concurrent and real-time programming in Java”: Wellings. John Wiley (2004)