5. SISTEMAS DE TIEMPO REAL

5.1. REQUISITOS DE TIEMPO REAL

Algunas acciones tienen que ser ejecutadas dentro de un intervalo de tiempo especificado. Diferentes niveles de criticalidad:

- *Hard RT requirements* (Tiempo real estricto): Tiene que cumplirse siempre. Un fallo puede poner en peligro la integridad del sistema.
- *Soft RT requirements* (flexible): Puede fallar ocasionalmente.
- *Firm RT requirements*: Si no se cumple, el resultado es inútil.

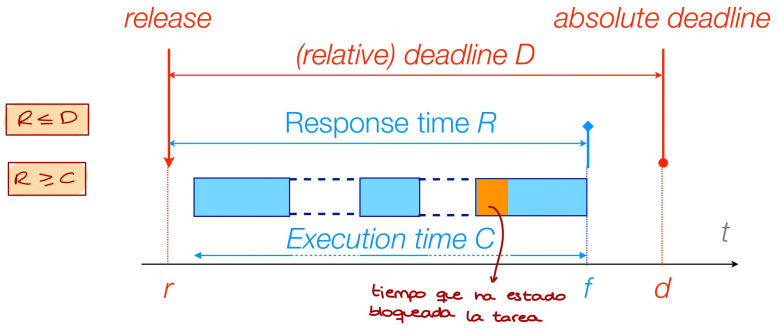
5.2. MODELO DE TAREA

Un sistema de tiempo real está compuesto por una serie de tareas. Cada tarea ejecuta una secuencia de trabajos, que se lanzan repetidamente de acuerdo con un patrón de activación (periódico, aperiódico, esporádico, random, etc.).

Cada trabajo tiene un deadline (*D*), de modo que tiene que ser ejecutado entre su reléase time y su deadline. Además, consume una cierta cantidad (*C*) de tiempo de procesador, posiblemente dividida entre diferentes intervalos de ejecución.

Tiempo de respuesta (*R*): Tiempo en el que se completa el trabajo, relativo al tiempo en el que se ha lanzado.

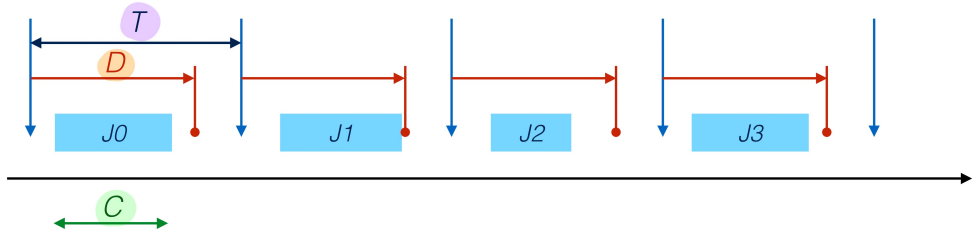
Principales requisitos de tiempo real de cada tarea: $R \leq D$. $C \leq R$ is implied.



5.3. TAREAS PERIÓDICAS Y ESPORÁDICAS

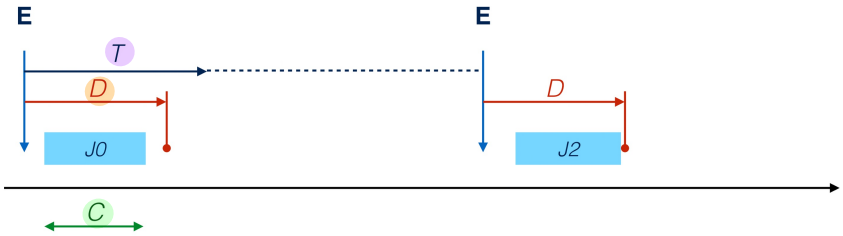
Las tareas de los sistemas de tiempo real pueden tener diferentes **patrones de ejecución**:

- Las **tareas periódicas** se ejecutan cada *T* segundos (*T* es el periodo de la tarea). Se caracterizan por *T* (periodo), *C* (tiempo de computación) y *D* (deadline).



- Las **tareas aperiódicas** se ejecutan cuando un evento *E* ocurre (*E* es el evento de activación). Ej: pasar a modo seguro. Están impulsadas por eventos, pero no tienen requisitos de tiempo real (no hay separación mínima entre ocurrencias del evento, no hay deadline, aunque a menudo se impone el requisito de que los trabajos deben completarse lo más pronto posible).

- Las **tareas esporádicas** son tareas aperiódicas que tienen una mínima separación entre dos ocurrencias sucesivas del evento de activación. La mínima separación se denota por *T*. Se caracterizan por *T*, *C*, D.



5.4. SCHEDULING DEL PROCESADOR

Si el número de tareas concurrentes es mayor que el número de procesadores, se debe tomar una decisión sobre qué tarea(s) ejecutar en un cierto tiempo. Para este propósito, el OS/RTS usa un algoritmo de scheduling. Métodos más comunes:

- Round-robin: Cada tarea se ejecuta durante un cierto fragmento de tiempo (time-slice) y una vez éste se consume, da paso a la siguiente tarea en una cola circular.
  - Es común en sistemas operativos de propósito general (es justo)
  - Bajo rendimiento en RTS
- Priority: Cada tarea tiene una prioridad (una medida de su importancia o urgencia). La tarea de mayor prioridad que esté lista se envía a ejecutar cuando un procesador está disponible. Permite un comportamiento predecible en RTS.
  - Prioridad estática: mejor predictibilidad (nos interesa para RTS).
  - Prioridad dinámica: mejor utilización del procesador.
  - Pre-emptive (con desalojo): Si una tarea de mayor prioridad que la que actualmente se está ejecutando pasa a estar lista, la desaloja del procesador para ejecutar la de mayor prioridad. Esto nos interesa más en RTS.
  - Non-pre-emptive (sin desalojo): La tarea que está ejecutándose puede continuar hasta que se suspenda, incluso si una tarea de mayor prioridad ha pasado a estar lista.

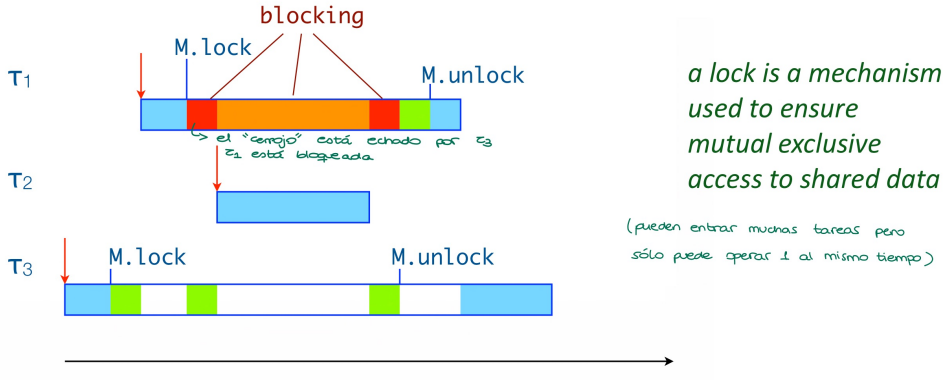
FPPS (Fixed-priority pre-emptive scheduling): método de scheduling común en RTS.

Una forma de asignar prioridades a tareas es mediante **Deadline-monotonic scheduling (DMS)**:

- Las tareas con menores deadlines tienen mayores prioridades.
- Es óptimo.
- Si *D*=*T* para todas las tareas, coincide con Rate-monotonic scheduling (RMS).

5.5. INVERSIÓN DE PRIORIDAD

Cuando dos tareas acceden a datos compartidos en exclusión mutua, puede aparecer la inversión de prioridad: una tarea de alta prioridad se retrasa por una tarea de baja prioridad. Ejemplo:



La inversión de prioridad no se puede eliminar completamente, pero puede suavizarse usando un protocolo de acceso apropiado para los objetos compartidos.

Habitualmente se usa el **ICPP (Immediate Ceiling Priority Protocol)** / ceiling locking / highest locker protocol:

- Cada recurso compartido tiene una prioridad *ceiling* (prioridad más alta de las tareas que la usan)
- Cuando una tarea intenta tener acceso exclusivo a un objeto compartido, inmediatamente hereda su prioridad *ceiling*.
- Sigue siendo posible el bloqueo de una tarea de alta prioridad, pero está limitado a la operación más larga del objeto compartido.