FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Sistemas de Tiempo Real
Sistemas Operativos de Tiempo Real
(RTS & RTOS)

Laboratorio de Sistemas Embebidos

http://laboratorios.fi.uba.ar/lse/

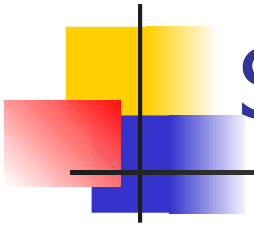
seminario-embebidos@googlegroups.com

66.48 & 66.66 Seminario de Electrónica: Sistemas Embebidos

http://laboratorios.fi.uba.ar/lse/seminario/

Ingeniería en Electrónica – FI – UBA

Buenos Aires, 23 de Octubre de 2012

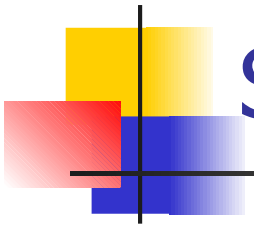


Sistemas embebidos

El nombre **sistema embebido** se refiere a los **equipos electrónicos** que incluyen un **procesamiento de datos**, pero a diferencia de una PC, están diseñados **para satisfacer una función específica** (reloj digital, reproductor de MP3, teléfono celular, router, sistema de control de automóvil –ECU- o de satélite o de planta nuclear)

Es un **sistema electrónico** que está **contenido** ("embebido") **dentro de un equipo completo** que incluye, por ejemplo, partes mecánicas y electromecánicas

El **cerebro** de un SE es típicamente un **microcontrolador**, aunque los datos también pueden ser procesados, por un **DSP**, una **FPGA**, un **microprocesador** o un **ASIC**, y su **diseño** está **optimizado** para reducir su **tamaño** y su **costo**, aumentar su **confiabilidad** y mejorar su **desempeño**. Algunas aplicaciones también tienen requisitos de **bajo consumo**, como por ejemplo un celular o un MP3, que se satisfacen gracias a los avances en la tecnología



Sistemas de Tiempo Real

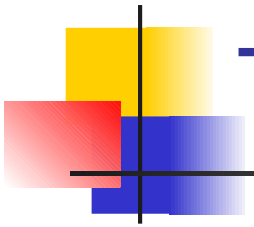
Un sistema de tiempo real es un sistema embebido que:

Interacciona repetidamente con su entorno físico

Responde a los estímulos que recibe del mismo dentro de un plazo de tiempo determinado

Para que el funcionamiento del sistema sea correcto no basta con que las acciones sean correctas (validez lógica), sino que tienen que ejecutarse dentro del intervalo de tiempo especificado (instante en que se produce la corrección)

El tiempo en que se ejecutan las acciones del sistema es significativo (asegurar: **determinismo** & **tiempo de respuesta**; p/tiempo de ejecución: contemplar el caso más desfavorable; atención: **Tiempo Real no es igual a Rápido** (ver caso por caso)



Tareas de Tiempo Real

Las actividades de un sistema de tiempo real se llaman **tareas**

Tienen varios tipos de requisitos

Funcionales: qué hacen

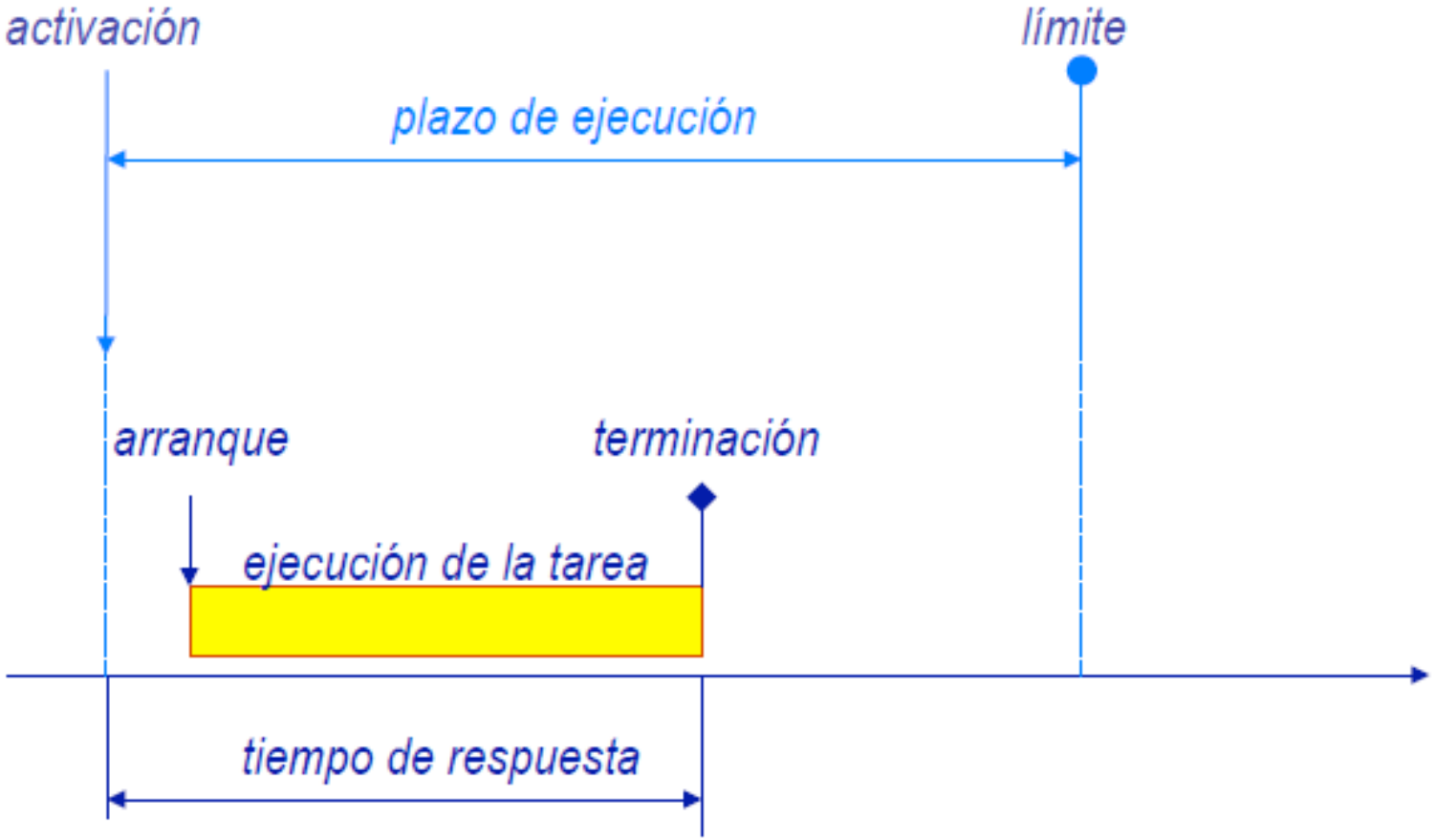
No funcionales: todos las demás

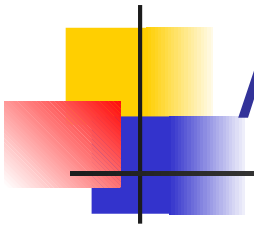
Temporales: cuándo se ejecutan, cuánto tardan, …

Fiabilidad, **seguridad**

Costo, **consumo de energía**, etc.

Ejecución de Tarea de Tiempo Real





Atributos Temporales

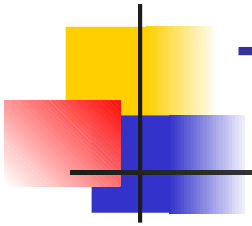
Activación

Periódica: a intervalos regulares, con período T
Aperiódica: cada vez que ocurre un evento determinado
Esporádica: separación mínima entre activaciones T
Estocástica, a rachas, irregular

Plazo de respuesta

Absoluto: tiempo límite para terminar
Relativo: intervalo desde la activación

Importante: Se trata de garantizar que la ejecución de cada tarea termina dentro de plazo



Tipos de Requisitos Temporales

Tiempo real **estricto** (**hard real-time**)

Todas las acciones deben terminar dentro d/plazo especificado.
Ejemplo: control de frenado

Tiempo real **flexible** (**soft real-time**)

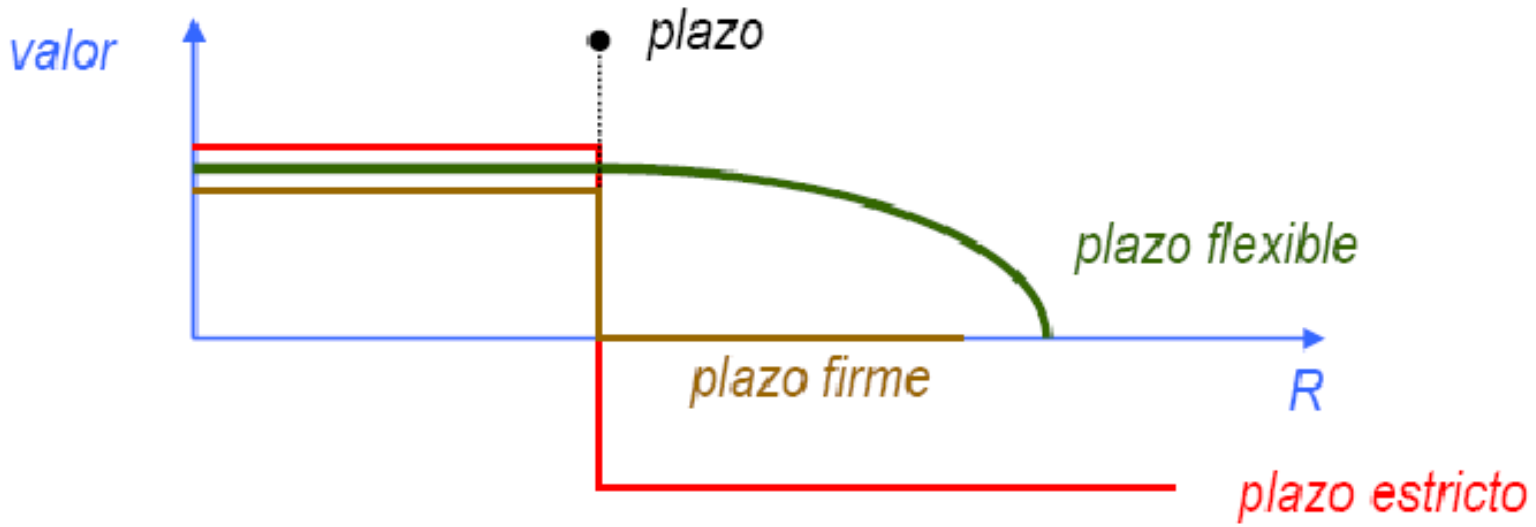
Se pueden perder plazos de vez en cuando. El valor de la respuesta decrece con el tiempo. Ej: adquisición de datos

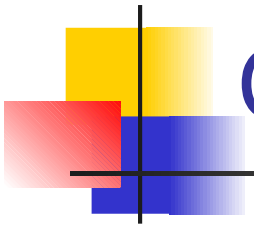
Tiempo real **firme** (**firm real-time**)

Se pueden perder plazos ocasionalmente. Una respuesta tardía no tiene valor. Ejemplo: sistemas multimedia

Importante: En un mismo sistema puede haber tareas con distintos tipos de requisitos temporales

Tipos de Requisitos Temporales





Características de los RTS

Gran tamaño y complejidad

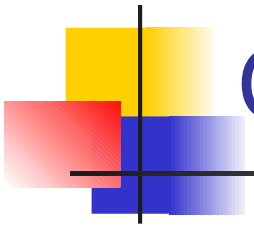
- Algunos STR tienen millones de líneas de código
- La variedad de funciones aumenta la complejidad incluso en sistemas relativamente pequeños

Simultaneidad de acciones (concurrencia)

- Los dispositivos físicos controlados funcionan al mismo tiempo
- Las tareas que los controlan actúan concurrentemente

Dispositivos de entrada y salida especiales

- Los manejadores de dispositivos forman parte del software de aplicación



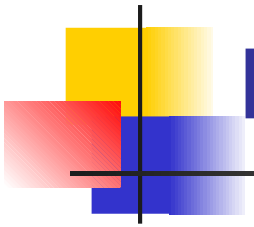
Características de los RTS

Seguridad y fiabilidad

- Sistemas críticos: fallos con consecuencias graves
- Pérdida de vidas humanas
- Pérdidas económicas
- Daños medioambientales

Determinismo temporal

- Acciones en intervalos de tiempo determinados
- Es fundamental que el comportamiento temporal de los STR sea determinista o, al menos, previsible
- No hay que confundirlo con la necesidad de que sea eficiente
- El sistema debe responder correctamente en todas las situaciones
- En los sistemas de tiempo real estricto hay que prever el comportamiento en el peor caso posible



Determinismo Temporal

Las tareas de tiempo real se ejecutan concurrentemente

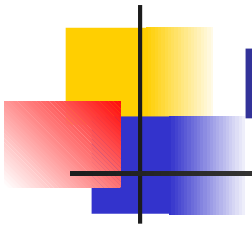
- Hilos (**threads**) o mecanismos similares

La planificación del uso del procesador debe permitir acotar el tiempo de respuesta

- **Prioridades** y otros métodos de planificación

Es conveniente analizar el comportamiento temporal del sistema antes de probarlo

- Las **pruebas** (tests) no siempre permiten asegurar el comportamiento en el peor caso



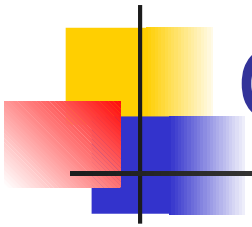
Modos de Funcionamiento

Muchos sistemas de tiempo real tienen varios modos de funcionamiento

- Por ejemplo: arranque, funcionamiento normal, funcionamiento seguro

El conjunto de tareas que se ejecutan y sus atributos temporales pueden variar de uno a otro modo

- Los cambios de modo deben ejecutarse garantizando la integridad temporal del sistema



Clases de Sistemas de Tiempo Real

Según las propiedades del sistema controlado

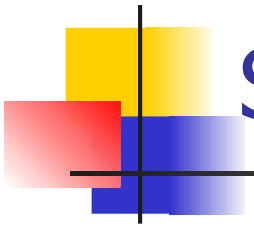
Sistemas críticos (hard RTS) y sistemas acríticos (soft RTS)
Sistemas con parada segura (fail safe) y sistemas con degradación aceptable (fail soft)

Según las propiedades del sistema de tiempo real

Sistemas con tiempo de respuesta garantizado (guaranted response) y sistemas que hacen lo que pueden (best effort)

Sistemas con recursos adecuados (resource-adequate) y sistemas con recursos inadecuados (resource-inadequate)

Sistemas dirigidos por tiempo y sistemas dirigidos por eventos



Sistemas críticos y acríticos

Se distinguen por sus **requisitos temporales y de fiabilidad**

Sistemas críticos (hard real-time systems)	**Sistemas acríticos** (soft real-time systems)
Plazo de respuesta **estricto**	Plazo de respuesta **flexible**
Comportamiento temporal determinado por el **entorno**	Comportamiento temporal determinado por el **computador**
Comportamiento en sobrecargas **predecible**	Comportamiento en sobrecargas **degradado**
Requisitos de seguridad **críticos**	Requisitos de seguridad **acríticos**
Redundancia activa	**Recuperación de fallos**
Volumen de datos reducido	Gran volumen de datos

Sistemas c/parada segura y c/degradación aceptable

Se distinguen por su **comportamiento en caso de avería**

Sistemas con parada segura (fail-safe)

Detención en estado seguro

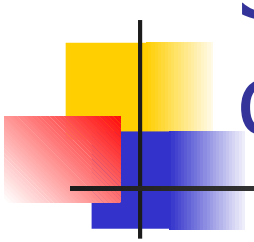
Probabilidad de detección de fallos elevada

Sistemas con degradación aceptable (fail-soft)

Funcionamiento con pérdida parcial de funcionalidad o prestaciones

También hay sistemas con tolerancia de fallos completa (fail operational)

Sistemas con respuesta garantizada y que hacen lo que pueden



Se distinguen por su grado de **determinismo temporal**

Sistemas con respuesta garantizada
(guaranteed response systems)

Comportamiento temporal garantizado analíticamente

Hace falta caracterizar con precisión la carga máxima y los posibles fallos

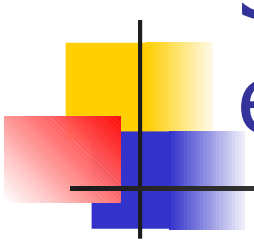
Sistemas que hacen lo que pueden
(best-effort systems)

Comportamiento temporal de tipo "lo mejor que se pueda"

No se hace una caracterización precisa de carga y fallos

Sólo sirve para sistemas acríticos

Sistemas con recursos adecuados e inadecuados



Se distinguen por la cantidad de **recursos disponibles**

Sistemas con recursos adecuados
(resource-adequate systems)

Diseño con suficientes recursos para garantizar el comportamiento temporal con máxima carga y en caso de fallos

Sistemas con recursos inadecuados
(resource-inadequate systems)

Diseño con recursos "razonables" desde un punto de vista económico

Sólo sirve para sistemas acríticos

Sistemas dirigidos por tiempo y por eventos

Se distinguen por la **forma de arrancar la ejecución de sus actividades**

Sistemas dirigidos por eventos
(event-triggered systems)

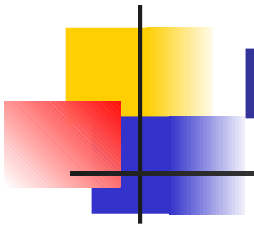
Arranque cuando se produce un **evento de cambio de estado**

Mecanismo básico:
interrupciones

Sistemas dirigidos por tiempo
(time-triggered systems)

Arranque en **instantes de tiempo predeterminados**

Mecanismo básico:
reloj



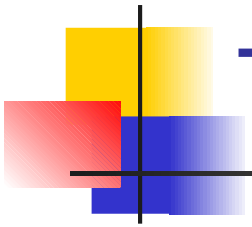
Resumiendo

Los sistemas de tiempo real interaccionan con su entorno y ejecutan sus acciones dentro de intervalos de tiempo determinados

Tienen requisitos muy exigentes

- Tamaño y complejidad
- Concurrencia
- Interfaces de hardware específicas
- Fiabilidad y seguridad
- Determinismo temporal

Hay varias clases de sistemas de tiempo real, con distintos requisitos temporales y de seguridad



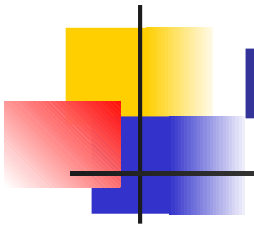
Tecnologías de implementación

Los métodos, herramientas y tecnología que se usan para construir otros tipos de sistemas no sirven para los sistemas embebidos de tiempo real, pues:

- No son suficientemente fiables
- Sólo contemplan el tiempo de respuesta medio, no el peor
- No garantizan los requisitos temporales

Las plataformas de desarrollo y ejecución suelen ser diferentes

- Es difícil hacer pruebas en la plataforma de ejecución
- Es difícil medir los tiempos con precisión



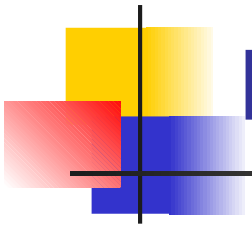
Diseño de Sistemas de Tiempo Real

El diseño de un sistema tiene varios aspectos

- **Funcional**: relación entre valores de entrada y de salida
- **Concurrente**: actividades concurrentes, sincronización, comunicación
- **Temporal**: requisitos temporales
- **Arquitectónico**: componentes, relaciones entre ellos

Cada aspecto se expresa mejor con un tipo de notación, por ejemplo:

- Simulink para el aspecto **funcional**
- UML (con perfiles específicos) para el diseño detallado de **componentes**
- AADL (Analysable Architecture Description Language) para los aspectos de **concurrencia** y **arquitectura**



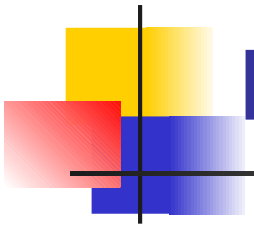
Diseño de Sistemas de Tiempo Real

En los sistemas de control existen cuatro niveles jerárquicos de software:

- Adquisición de datos / Actuadores
- Algoritmos de control (PID)
- Algoritmos de supervisión (trayectorias)
- Interfaz de usuario, registro de datos, etc.

Un programa secuencial se divide en funciones que se ejecutan según un orden preestablecido, mientras que un sistema en tiempo real se divide en tareas que se ejecutan en paralelo

- La ejecución en paralelo se consigue como la sucesión rápida de actividades secuenciales



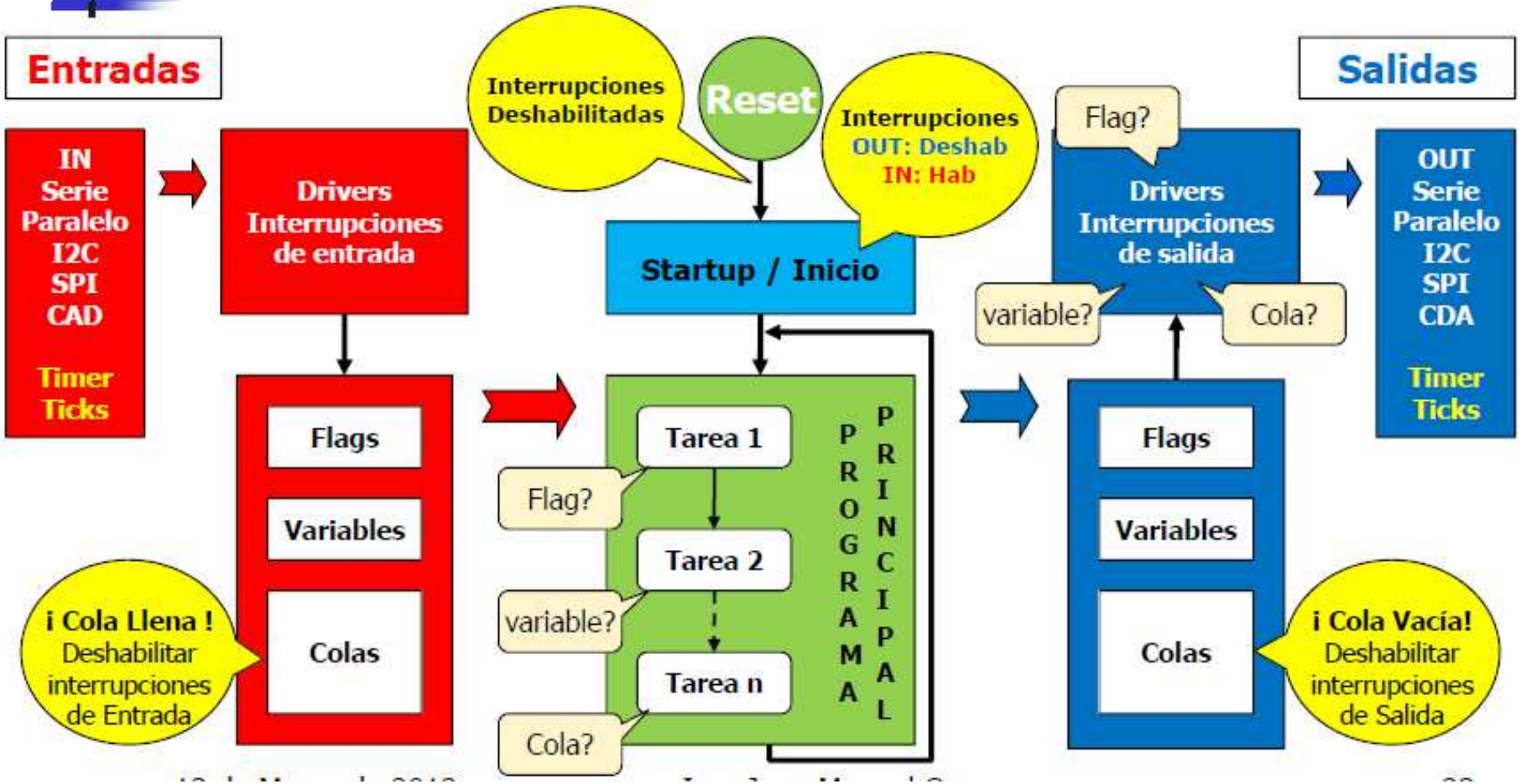
Diseño de Sistemas de Tiempo Real

Contamos con varias técnicas para ejecutar tareas en paralelo:

Procesamiento secuencial (Lazo de barrido o scan loop)
Interrupciones (Background/Foreground)
Multitarea cooperativo o expropiativo (preemptive)

Los más usados por los principiantes son el **Procesamiento Secuencial** (lazo perpetuo de barrido de tareas) e **Interrupciones** (una o más tareas asociadas a las interrupciones en 1° plano y otra tarea en 2° plano encargada de ejecutar un lazo perpetuo dentro del cual puede implementarse un procesamiento secuencial, la interfaz de usuario o a no hacer nada). En conjunto con las **máquinas de estado finito** constituyen una de las soluciones posibles de lo que se denomina **programación gobernada por eventos** (event driven programming)

Diseño con Interrupciones

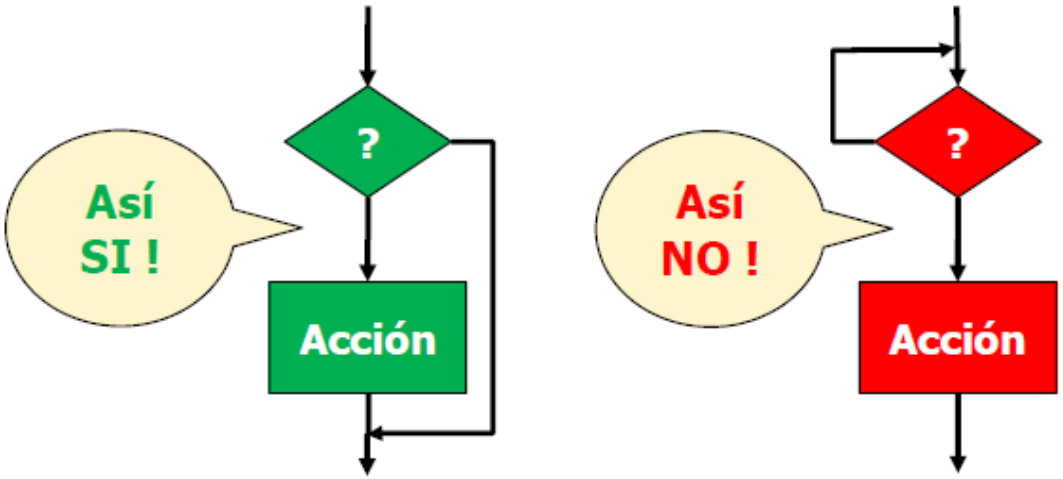


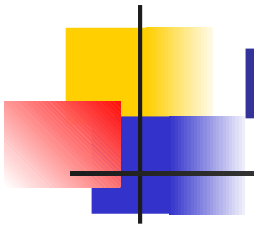
Diseño con Interrupciones

Comunicar los módulos simples mediante:

- Flags / Semáforos
- Variables
- Colas

Garantizar que ningún módulo se apropie de la CPU (comportamiento comunitario o no bloqueante)

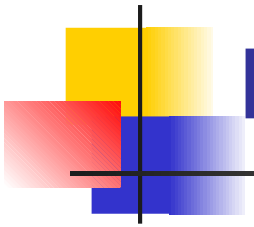




Diseño con Interrupciones

En las rutinas de atención de interrupción (ISR) sólo se hará lo estrictamente necesario para atender al hardware. Por lo que es necesaria una comunicación entre las ISR´s (1° plano) y las tareas en 2° plano

Pude haber **problemas de coherencia** de datos si la tarea de 2° plano usa **datos compartidos** de una manera no atómica
Una parte de programa es **atómica** si **no puede ser interrumpida**
Usar facilidades del lenguaje & compilador (p/ej en C **volatile**)
Existen varias técnicas para no inhabilitar las interrupciones, entre las cuales la más segura es el uso de **colas circulares**
En caso de tratarse de **flags** lo ideal es que las tareas **productoras** los fuercen a "**1**" y las **consumidoras** los lean y los fuercen a "**0**"

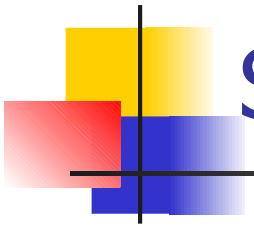


Diseño con Interrupciones

El uso de máquinas de estado facilitará la resolución tanto en las tareas de 1° como en las de 2° plano. Permiten la fragmentación de problemas de naturaleza secuencial en una secuencia de acciones/actividades gobernadas tanto por eventos como por sincronismos. Son una de las soluciones posibles de la programación gobernada por eventos (event driven programming)

Puede recurrirse a un planificador o scheduler para administrar las tareas en 2° plano del tipo:

- Lazo de barrido
- Cola de funciones sin prioridades
- Cola de funciones con prioridades



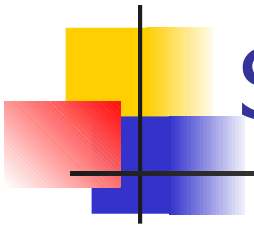
Sistemas Operativos

Los sistemas operativos convencionales no son adecuados para implementar sistemas de tiempo real

- No tienen un comportamiento determinista
- No permiten garantizar los tiempos de respuesta
- Algunos de ellos son poco fiables

Un sistema operativo de tiempo real (**RTOS**) debe soportar

- **Concurrencia**: procesos ligeros (hebras o threads)
- **Temporización**: medida de tiempos y ejecución periódica
- **Planificación determinista**: gestión del procesador y otros recursos
- **Dispositivos de E/S**: acceso a recursos de hardware e interrupciones

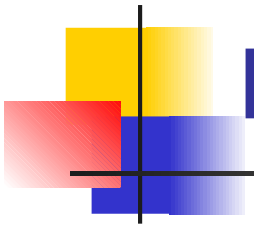


Sistemas Operativos

Un OS es un conjunto de programas que ayudan al programador de aplicaciones a gestionar los recursos de hardware disponibles, (entre ellos el tiempo del procesador y la memoria) aportando además mecanismos de comunicación y sincronismo entre tareas

La gestión del tiempo del procesador permite al programador de aplicaciones escribir múltiples tareas (subprogramas) como si cada uno fuera la única que utiliza la CPU (una parte del OS se encarga de asignar tiempo de ejecución a todos las tareas que tiene cargadas en base a un juego de reglas conocido de antemano)

Con esto se logra la ilusión de que múltiples tareas se ejecutan simultáneamente, aunque en realidad sólo pueden hacerlo de a uno a la vez (en sistemas con un sólo núcleo, como es el caso general de los sistemas embebidos)

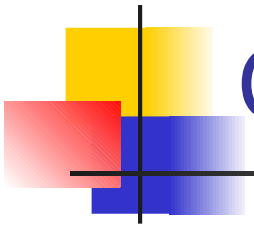


Kernel monolítico vs microkernel

Básicamente un OS es un conjunto de programas organizados como:

Kernel (Núcleo monolítico): todas las funciones residentes del SO están en el núcleo

Microkernel: Algunas funciones del SO se implementan como tareas similares a las de la aplicación



Componentes de un OS

Reloj de Tiempo Real (**Real Time Clock**, RTC)

Proporciona la temporización del sistema

Programador de Procesos (**Scheduler**)

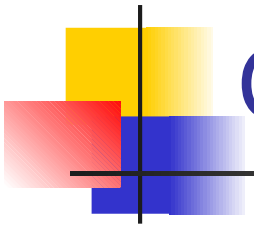
Establece el orden de ejecución de las tareas de acuerdo a una estrategia establecida

Ejecutor (**Dispatcher**)

Gestiona el inicio y la finalización de la ejecución de una tarea (cambios de contexto, stack, etc.)

Administrador de Interrupciones (**Interrupt Handler**)

Gestiona las peticiones de interrupciones de hardware o software



Componentes de un OS

Gestor de Recursos (**Resourse Manager**)

Administra el acceso a los recursos de hardware del sistema (procesador, memoria, dispositivos de I/O, comunicaciones, etc.)

Gestor de Configuraciones (**Configuration Manager**)

Administra la configuración del sistema (cambio de componentes de hardware o actualización de software) sin interrumpir el funcionamiento del mismo

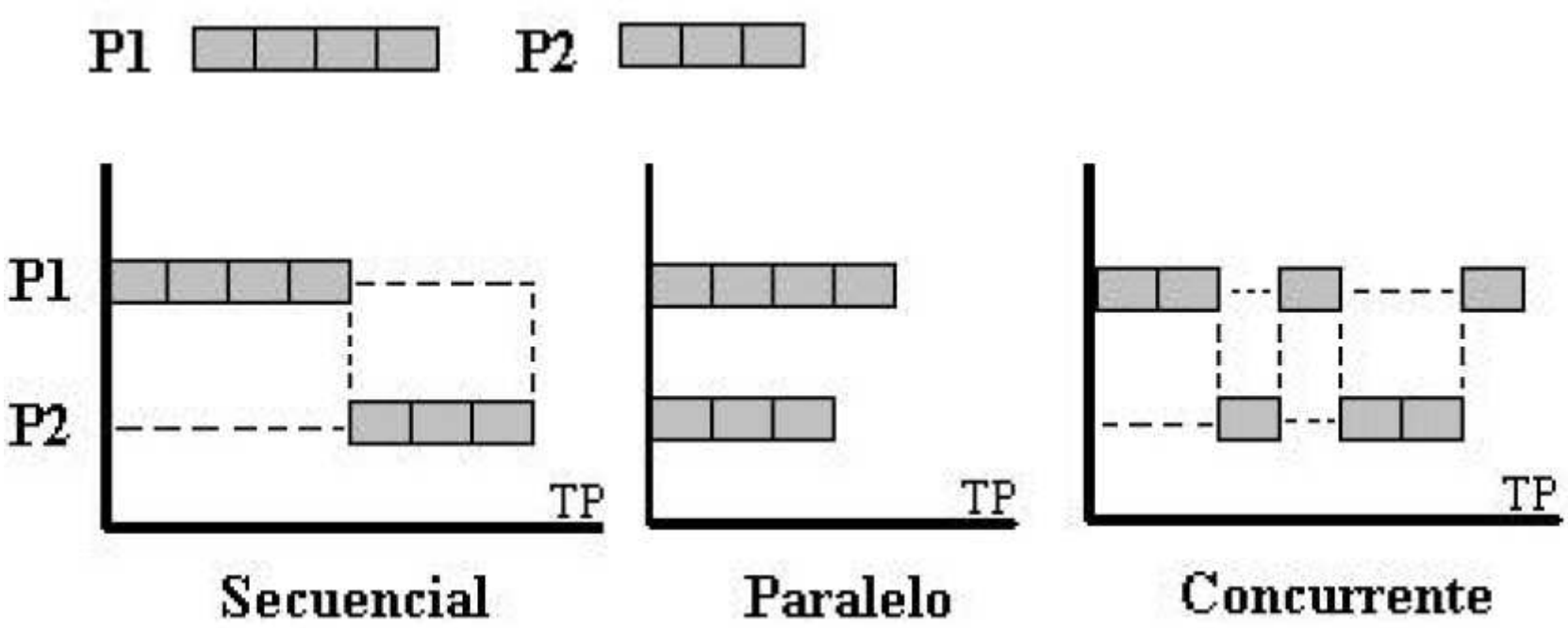
Gestor de Fallos (**Fault Manager**)

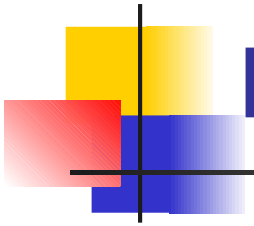
Detecta fallos de hardware y software y toma acciones para asegurar la disponibilidad del sistema

Estos dos últimos imprescindibles en OS de misión crítica

Multitasking

Se pretende que repartir el tiempo de procesamiento entre las distintas tareas creando la ilusión de procesamiento **concurrente**





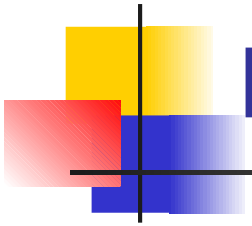
Prioridades de las Tareas

El concepto de prioridades es básico en sistemas embebidos, especialmente asociado a interrupciones

El procesamiento de determinados estímulos suele ser más importante que el de otros, surge entonces la noción de prioridad de las tareas

Las tareas de alta prioridad están asociadas a estímulos que requieren muy poca latencia hasta su procesamiento y respuesta

Las tareas de baja prioridad se asocian a estímulos en los que es posible mayor latencia, a veces por la naturaleza del proceso a controlar y otras veces porque el hardware provee alguna funcionalidad de buffering de datos



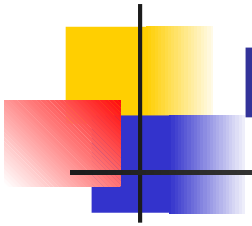
Ejecución de Tareas

El programador de procesos (**scheduler**) determina cuál será la siguiente tarea a ser ejecutada, elaborando la lista de tareas a ejecutar

Tal determinación obedece a una estrategia de programación (scheduling strategy) preestablecida

Usualmente Su accionar se repite a intervalos regulares establecidos por el reloj de tiempo real (**RTC**)

El ejecutor (**dispatcher**) toma el primer proceso de la lista generada por el scheduler, le asigna memoria y tiempo de procesador (se encarga del cambio de contexto) e inicia su ejecución (le transfiere el control)



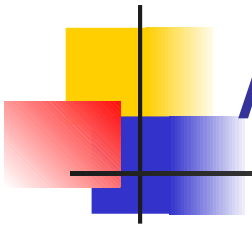
Estrategias de Scheduling

Cooperativa o No Expropiativa (**Non preemptive**)

Cada tarea se ejecuta hasta su finalización, se autobloquea (p/ej: esperando una respuesta de hardware) o suspende voluntariamente su ejecución para permitir la ejecución de otra tarea

Expropiativa (**Preemptive**)

La ejecución de la tarea puede ser suspendida por el scheduler al surgir una tarea con mayor prioridad (expropia la CPU a la tarea que se esté ejecutando y se la cede cambio de contexto mediante) o al finalizar el tiempo asignado a esa tarea



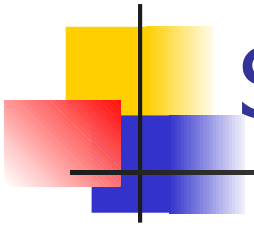
Algoritmos de Scheduling

Round-Robin: Las tareas se ejecutan en secuencia (en ronda), pueden ejecutarse:

- Durante un tiempo preestablecido (**time slice**) e igual para todos
- Con un tiempo asociado a cada tarea
- Una tarea comienza cuando la anterior finaliza

Crecimiento monolítico: Cada tarea tiene asociado un nivel de prioridad único y se ejecuta hasta su finalización o bloqueo

Menor tiempo de finalización primero: Se ejecuta primero la tarea que requiere menos tiempo estimado para su finalización



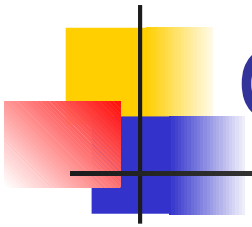
Status de una Tarea

Una tarea escencialmente puede estar en dos estados:

Not Running: No está en ejecución (es un estado compuesto por subestados)
- **Ready**: Lista para su ejecución
- **Bloqued**: A espera de un evento para volver a Ready
- **Suspended**: Totalmente inactiva

Running: En ejecución

Conforme al estado de la tarea, la prioridades asignadas a ellas y a la estrategia de scheduling establecida, el scheduler elabora la listas de tareas ready, para que oportunamente el dispatcher le asigna memoria y tiempo de procesador e inicia su ejecución



Comunicación entre Tareas

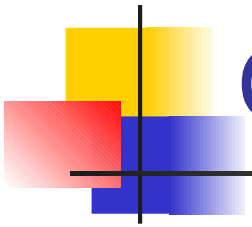
Queues (colas)

Es una colección de estructuras de datos ordenada

Las operaciones que se pueden realizar son agregar a la estructura al final, sacar una estructura del inicio (Fist In – First Out), sacar del final (Firt In – Last Out), información del estado de la cola y copias datos (sin modificar la cola)

Pueden usarse como buffers de datos si una tarea genera más información que la que otra es capaz de procesar

Las implementaciones más comunes son mediante buffers circulares y listas enlazadas y su uso correcto es responsabilidad del desarrollador



Comunicación entre Tareas

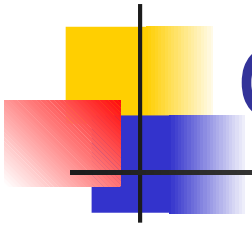
Mutex (exclusión mutual)

Se utilizan para bloquear el acceso a recursos de hardware o software que deben ser compartidos por distintas tareas

El mutex actúa como una ticket (**token**) que debe ser adquirido por la tarea que desea acceder al recurso compartido

Una vez que una tarea adquiere el mutex asociado a un recurso, ningún otra tarea puede adquirirlo hasta que sea liberado por la tarea que lo adquirió primero

Su uso correcto es responsabilidad del desarrollador



Comunicación entre Tareas

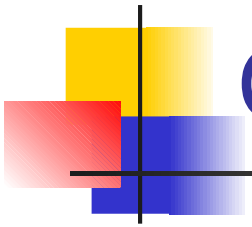
Semaphores (semáforos)

Se utilizan para sincronizar tareas

Pueden ser binarios (tienen dos estados) o contadores (llevan un conteo)

Una tarea es propietaria del semáforo y fija su estado o valor de conteo, otras tareas pueden leer el estado o el conteo y actuar en consecuencia

Su uso correcto es responsabilidad del desarrollador



Comunicación entre Tareas

Sockets

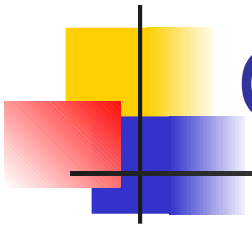
En general se utilizan para sincronizar tareas que se ejecutan en distintos dispositivos conectados en red

Suelen utilizarse una arquitectura cliente-servidor en la que una tarea (servidor) provee información o recursos a otras tareas (clientes)

Cada tarea dispone de un socket y éstos se conectan entre sí

Cada socket puede aceptar conexiones de un número distintos de otros sockets

Son bidireccionales, secada tarea puede leer y escribir en ellos



Comunicación entre Tareas

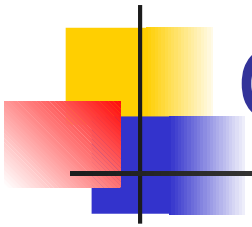
Shared Memory (memoria compartida)

Es un área de memoria física compartida entre dos o más tareas. Todas las tareas las ven como parte de su propia área de memoria

Normalmente las tareas acceden a ella mediante punteros. Es un método de comunicación de bajo nivel

No hay controles sobre el acceso a la memoria, los mismos deben implementarse mediante semáforos u otras técnicas

Su uso correcto es responsabilidad del desarrollador



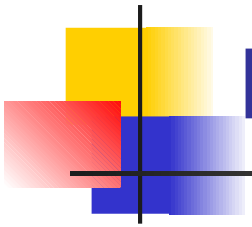
Comunicación entre Tareas

En todos los casos presentados es de destacar que

- Su uso correcto es responsabilidad del desarrollador

Problemas de sincronización a evitar

- Deadlocks
- Starvation (inanición)
- Inversión de prioridades



Referencias

Real-Time Systems and Programming Languages (Fourth Edition) Ada 95, Real-Time Java and Real-Time POSIX by Alan Burns and Andy Wellings
http://www.cs.york.ac.uk/rts/books/RTSBookFourthEdition.html

Sistemas de Tiempo Real, Juan Antonio de la Puente, Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid http://web.dit.upm.es/~jpuente/strl/

Introducción al tiempo real en sistemas empotrados (Master Oficial en Ingeniería de Sistemas Empotrados), Alberto Lafuente, Universidad del País Vasco,
http://www.sc.ehu.es/acwlaroa/ITRSE.htm

Sistemas en Tiempo Real, Escuela Universitaria de Ingeniería de Gijón http://isa.uniovi.es/docencia/TiempoReal/

Sistemas Empotrados en Tiempo Real - José Daniel Muñoz Frías
www.lulu.com/product/ebook/sistemas-empotrados-en-tiempo-real/