

# Sistemas embebidos Conceptos

# Agenda.

- Sistemas embebidos.
- Tipos de sistemas.
  - Superloop.
  - Foreground / Background.
  - Con planificador. Kernel. Sistema Operativo.
- Tareas. Estado y contexto
- Planificación. Tipos de planificadores, desarrollo.
- Sistemas en tiempo real.
- Sistemas gobernados por eventos (EDS).
- Sistemas gobernados por tiempo (TDS).
- ¿Por qué desarrollar sistemas TDS?

# Sistemas embebidos.

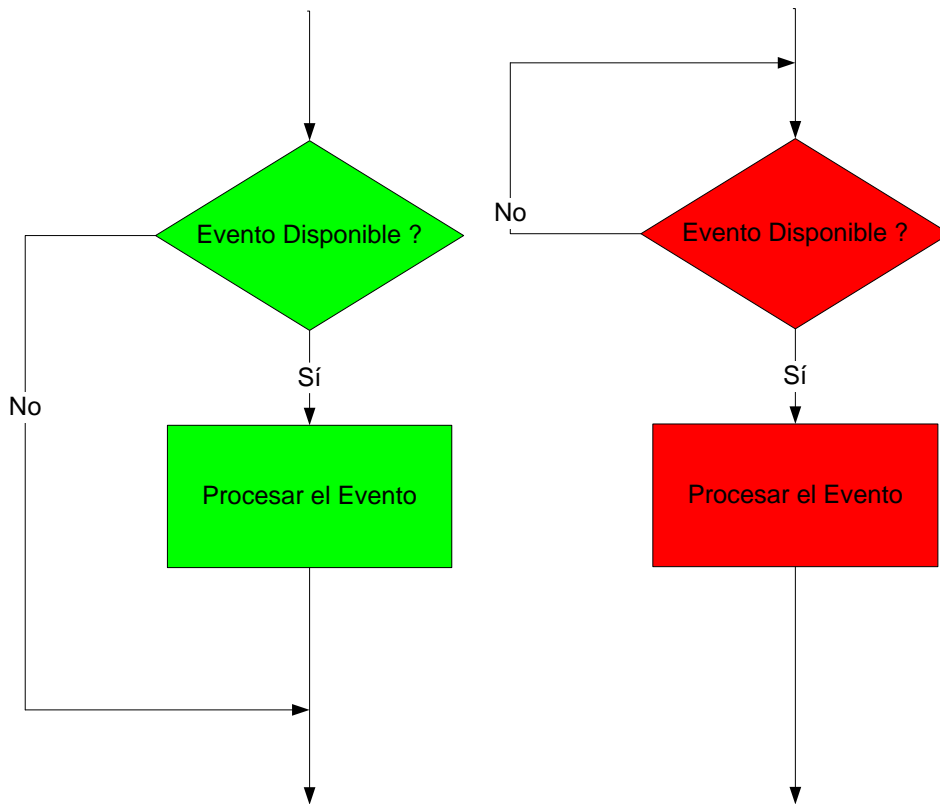
- El nombre sistema embebido se refiere a los equipos electrónicos que incluyen un procesamiento de datos, pero a diferencia de una PC, están **diseñados para satisfacer una función específica** (reloj digital, reproductor de MP3, teléfono celular, router, sistema de control de automóvil –ECU– o de satélite o de planta nuclear).
- Es un sistema electrónico que **está contenido (“embebido – empotrado”) dentro de un equipo completo** que incluye, por ejemplo, partes mecánicas y electromecánicas.
- Dentro de los sistemas embebidos nos dedicaremos a los sistemas embebidos de **“un único programa”**. Esto es, que la actualización o cambio de software no es una tarea usual (como si lo es a través de las apps en los teléfonos por ejemplo)
- 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 superlazo.

- Es la primera solución para implementar sistemas de tiempo real.
- Consiste en un lazo iterativo infinito que ejecuta las tareas de manera consecutiva.
- Los controladores lógicos programables se suelen implementar con esta configuración (bucle de scan).
  - ☐ Ciclos de lectura de entrada.
  - ☐ Ciclo de cálculo
  - ☐ Ciclo de actualización de salidas.

```
void main(void)
{
    inicializar();
    while(TRUE)
    {
        tarea1();
        tarea2();
        tarea3();
        ...
        ...
        ...
    }
}
```

# Superlazos. Procesamiento de eventos.

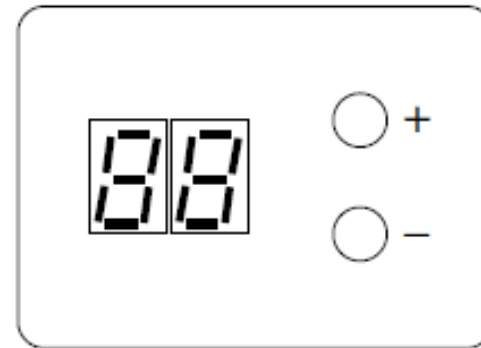
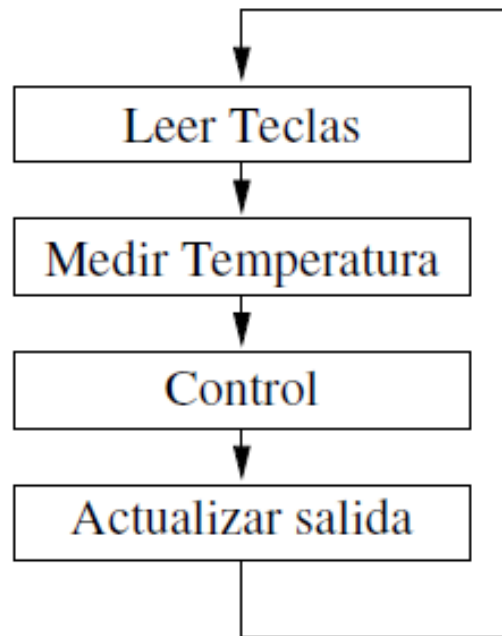


- Estos sistemas ***no pueden*** bloquearse esperando un evento externo, ya que atentan contra el procesamiento de las otras tareas.

# Superlazo. Ventajas y Desventajas.

- Son sistemas que prácticamente no exigen nada en particular al hardware que los implementa.
- Prácticamente todo el tiempo del sistema es para las tareas.
- El “tiempo del lazo” (tiempo que tarda una tarea en volver a ser ejecutada) depende del total de las tareas que ejecuta el procesador, por lo que es variable en función de la aplicación. No es fácil estimar este tiempo sin agregar hardware.
- No se puede asegurar “timing preciso” sin perder mucho tiempo del procesador y aumentar la latencia.
- Fuerte “determinismo” y “previsibilidad”.
- Todas las tareas afectan al resto. Si una tarea se “sienta” a esperar un evento el resto no se ejecuta nunca.
- La latencia no está asegurada. Ya que si un evento externo se da antes de su tarea la latencia es mínima, si se da después de que esta se ejecutó, debe esperar que se ejecute todo el lazo.
- Las tareas deben implementarse teniendo en cuenta la existencia de otras tareas.

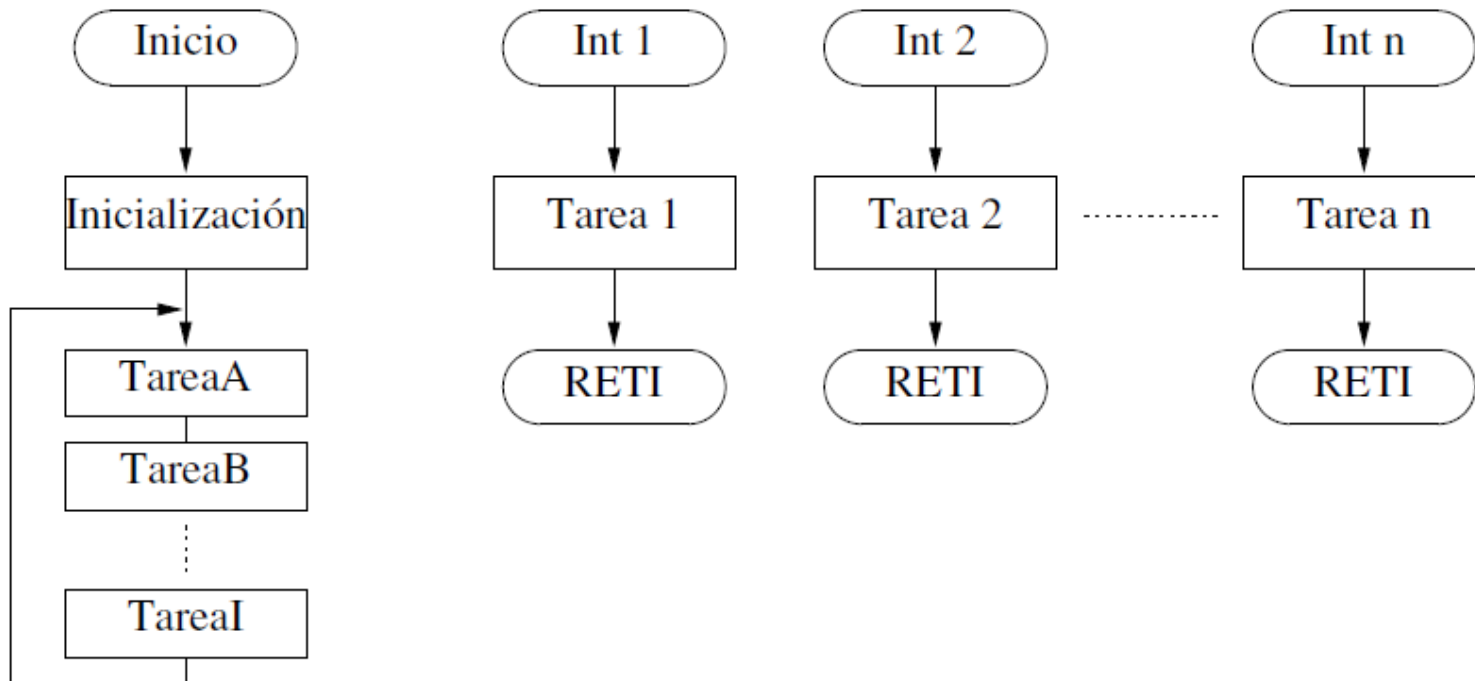
# Ejemplo. Control de temperatura



# Sistemas Foreground/Background (FB).

- Para “mejorar” la latencia de los sistemas de “superloop” se proponen los sistemas FB.
- Los sistemas FB exigen al hardware la existencia de interrupciones.
- Son sistemas manejan un lazo similar al superloop de latencia larga y tratamiento de eventos con baja latencia por medio de interrupciones.

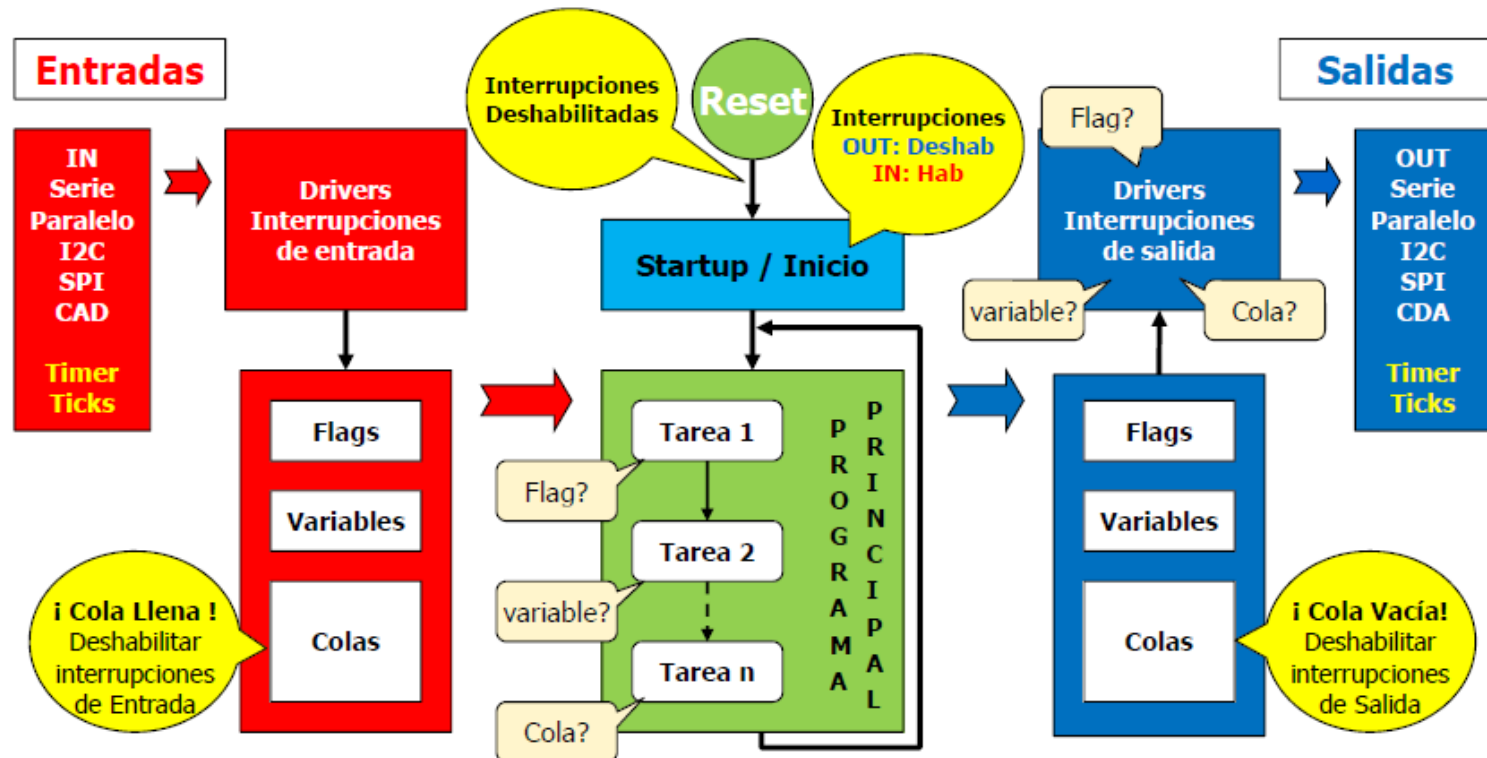
# FB. Esquema de tareas.



# FB. Latencia

- Para que los sistemas FB mantengan su baja latencia el código de atención de las interrupciones debe ser muy breve en el tiempo, ya que usualmente una interrupción bloquea a las otras.
- Esto implica el uso de colas para minimizar el tiempo en background (interrupciones) y maximizar el tiempo del lazo principal.

# FB. Eventos y colas.



# FB. Ventajas y Desventajas.

- Requieren de recursos de hardware específicos (interrupciones).
- Mejoran la latencia con respecto a los sistemas de superloop (tiempos de interrupción).
- Pueden manejar con baja latencia tantos eventos como interrupciones disponga el procesador que lo implementa.
- Pueden manejar timing de manera precisa (timerticks).
- Las tareas deben implementarse de manera similar a los sistemas superloop. Deben conocer la existencia de sus tareas vecinas.
- Son sistemas que suelen trabajar con colas, para recibir los eventos externos con latencia baja y procesarlos en el lazo principal (foreground) más tarde.
- El uso de prioridades para el manejo de eventos está muy ligado al hardware utilizado (posibilidad o no de anidar interrupciones).
- Con múltiples interrupciones no se puede asegurar el cumplimiento de requerimientos temporales

# Sistema Operativo en tiempo real (RTOS).

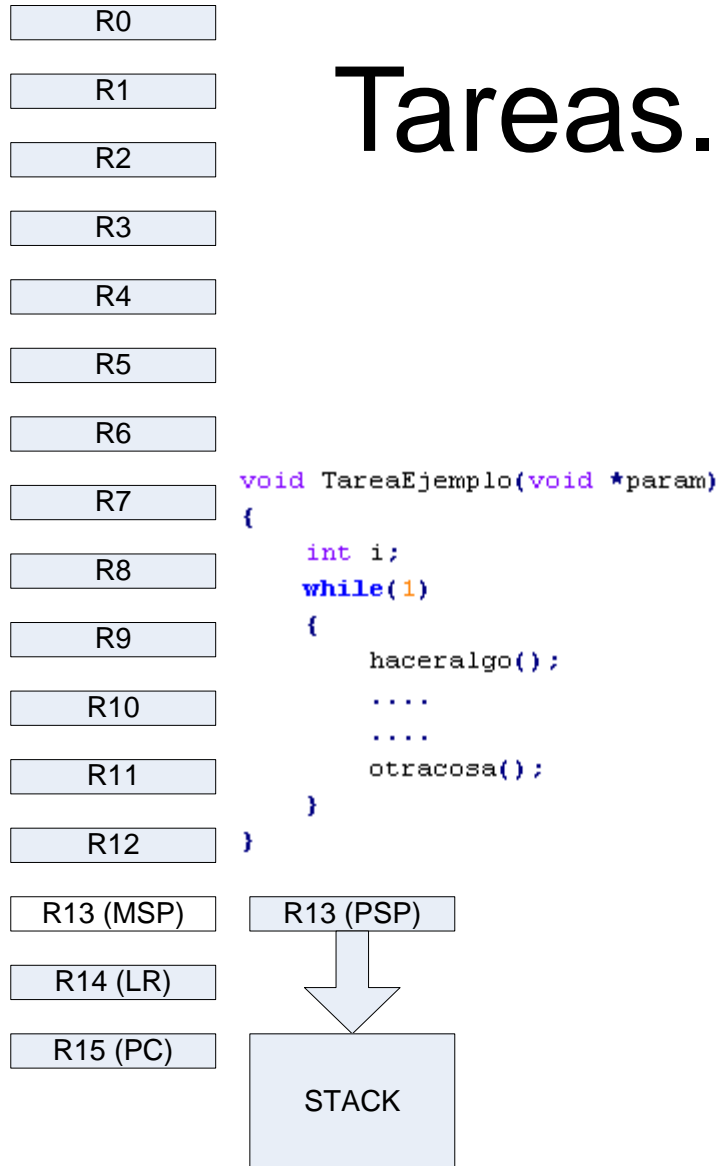
- Sistema Operativo: Programa o conjuntos de programas que proveen servicios a los programas de aplicación, que se ejecutan con más privilegios que los programas de aplicación.
- Un sistema operativo en tiempo real intenta tener latencia de interrupciones y de cambio de tarea mínimas.



# Procesos, Tareas.

- Un concepto fundamental de los sistemas operativos es el concepto de **proceso**.
- Un proceso es básicamente un tramo o un programa en ejecución.
- Es decir, un proceso no sólo está compuesto por el código que ejecuta, también por su **contexto**
- A lo largo del curso los términos proceso y tarea se van a utilizar como sinónimos (sólo válido para TD2).

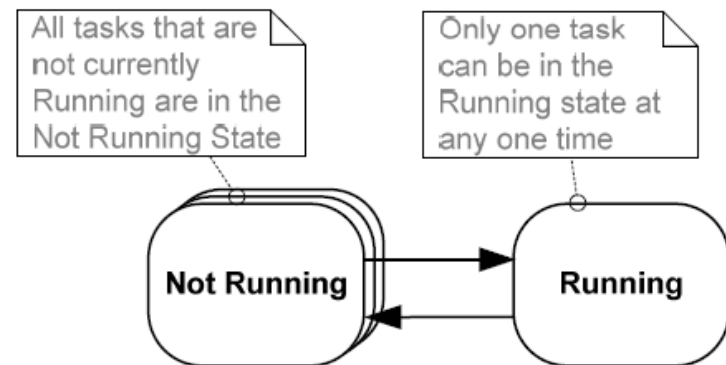
# Tareas. Contexto.



- El contexto va a estar definido por el contenido de los registros, la pila del proceso y el código de la tarea, su estado y prioridad.
- La tarea va a estar compuesta por el programa y la “foto” del procesador en ese momento.

# Planificador (scheduler).

- Una de las patas de los RTOS son las tareas. La otra pata es el planificador.
- El planificador es un programa que se va a encargar (con algún criterio determinado) de ejecutar, salvar, recuperar y dejar de ejecutar las diferentes tareas de la aplicación.
- Las tareas van a tener por lo pronto dos estados:
  - ☐ Ejecutando
  - ☐ En espera



# Planificador Cooperativo.

- El planificador cooperativo se basa en que cada tarea en ejecución “gentilmente” devuelve el control al planificador por medio de alguna función especial (usualmente función Yield() o similar) y éste luego pasa la ejecución a otra tarea.
- En este tipo de planificadores, la prioridad la tiene siempre la tarea en ejecución hasta que “voluntariamente” la cede.

# Planificador “Calesita”. Round Robin.

- Cuando un planificador funciona como round robin, cada tarea tiene asignada una determinada cantidad de “ticks” de procesador. Cada tarea retiene al procesador durante sus ticks asignados. Luego se pasa a las siguientes de la misma manera.
- En este tipo de planificación, todas las tareas tienen la misma prioridad.

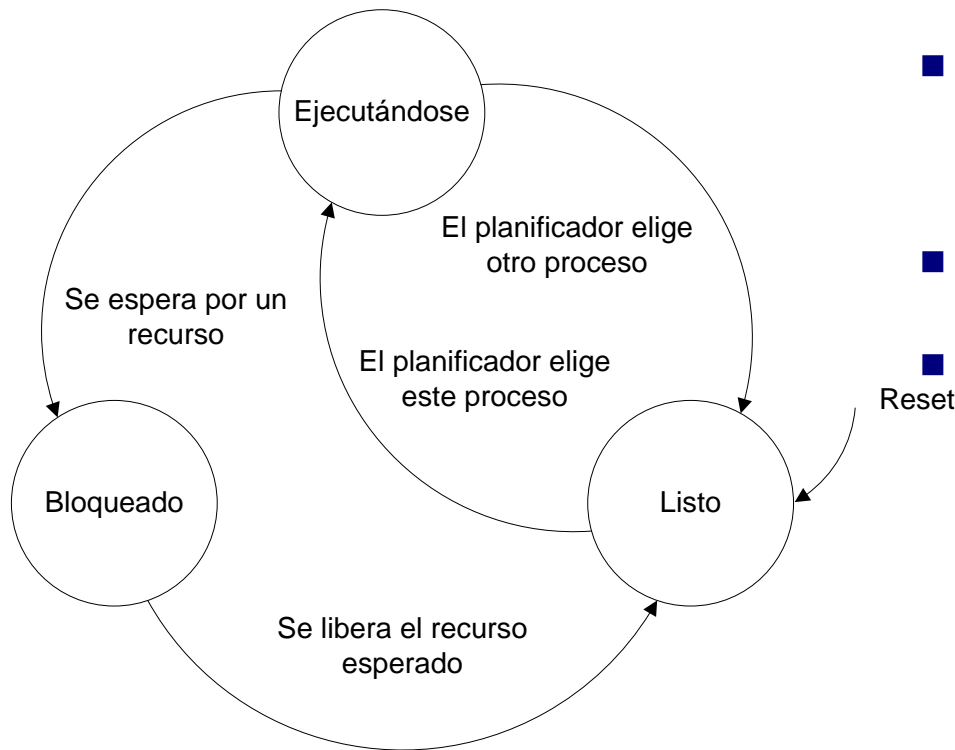
# Planificador de tipo apropiativo (preemptive).

- En este tipo de planificación cada tarea tiene una prioridad determinada. El planificador va a ejecutar la tarea de mayor prioridad que esté en condiciones de ejecutarse.
- Este tipo de planificador es el que va a minimizar la latencia, ya que los eventos suelen modelarse con tareas de prioridad elevada.

# ¿Quién escribe el Planificador?

- El planificador va a ser provisto por el proveedor del RTOS. Usualmente se obtiene el planificador y código para el manejo de hardware de uso común.
- Algunos RTOS:
  - FreeRTOS.
  - uC/OS-II
  - uCLinux
  - QNX
  - VxWorks
  - LynxOS

# Tareas. Tarea Bloqueada.



- Anteriormente se mencionó que una tarea puede estar en dos posibles estados. Ejecutándose o no.
- Una tarea en ejecución es la que está usando el procesador.
- Las tareas que no están ejecutando se van a dividir en dos:
  - Listas. Son las que están en condiciones de ejecutarse.
  - Bloqueadas. Están esperando algún recurso y no serán planificadas hasta que el recurso se libere.

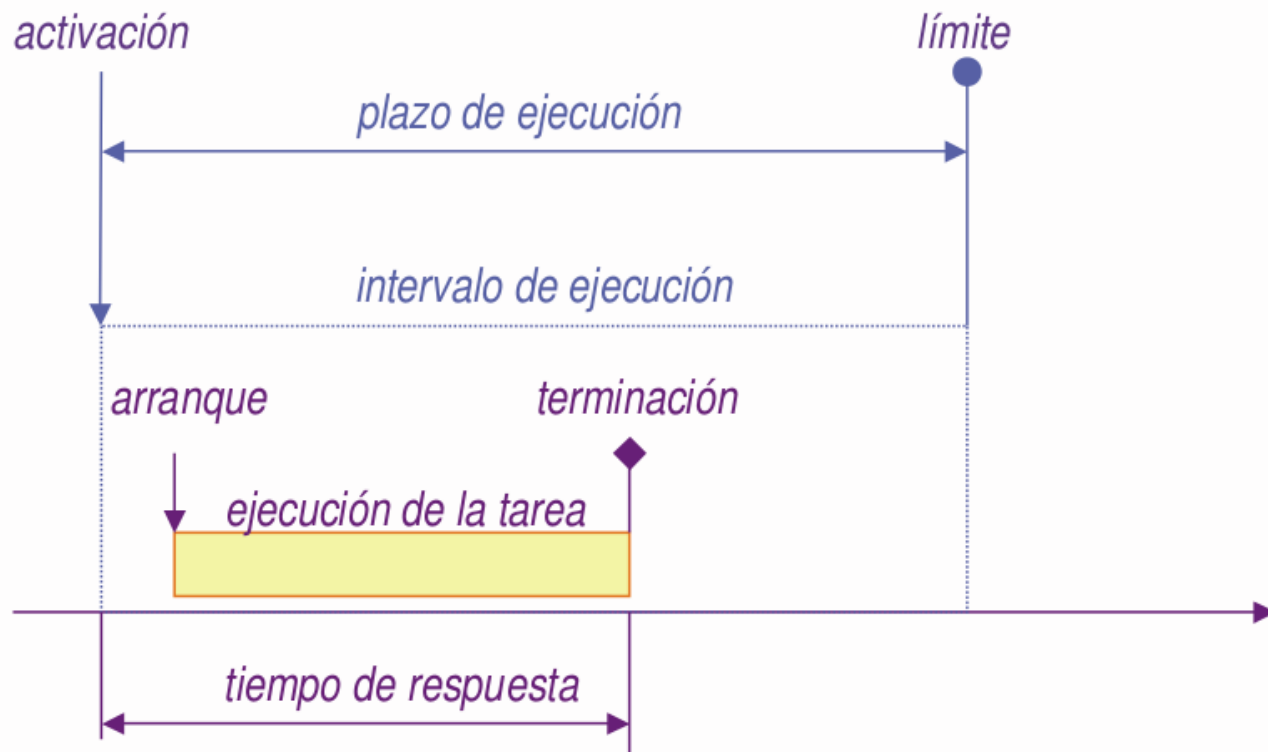
# ¿Si están todas las tareas bloqueadas, que pasa?

- Puede darse el caso de que todas las tareas se encuentren bloqueadas en un determinado punto del sistema.
- Como los RTOS deben estar ejecutando tareas todo el tiempo, el RTOS define una tarea de prioridad mínima (la tarea ociosa o “idle task”) que se va a ejecutar cuando todas las otras tareas estén bloqueadas.

# Sistemas en tiempo real.

- Un sistema en tiempo real se puede definir como aquel en el que el resultado de una operación tiene que ser lógicamente correcta y dada en un tiempo acotado y predecible .

# Sistemas en tiempo real.



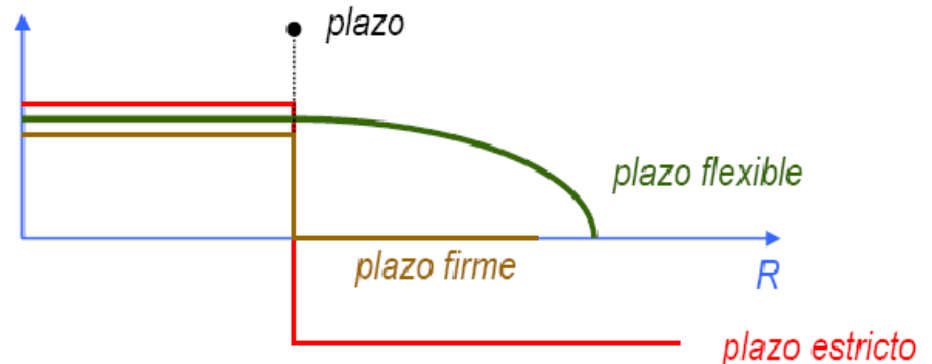
# Sistemas en tiempo real.

- Se definen tres tipos de sistemas de tiempo real:

- **Estrictos (hard real time).** Si no se alcanza la respuesta en el tiempo determinado se considera que el sistema **falla**.  
*valor*

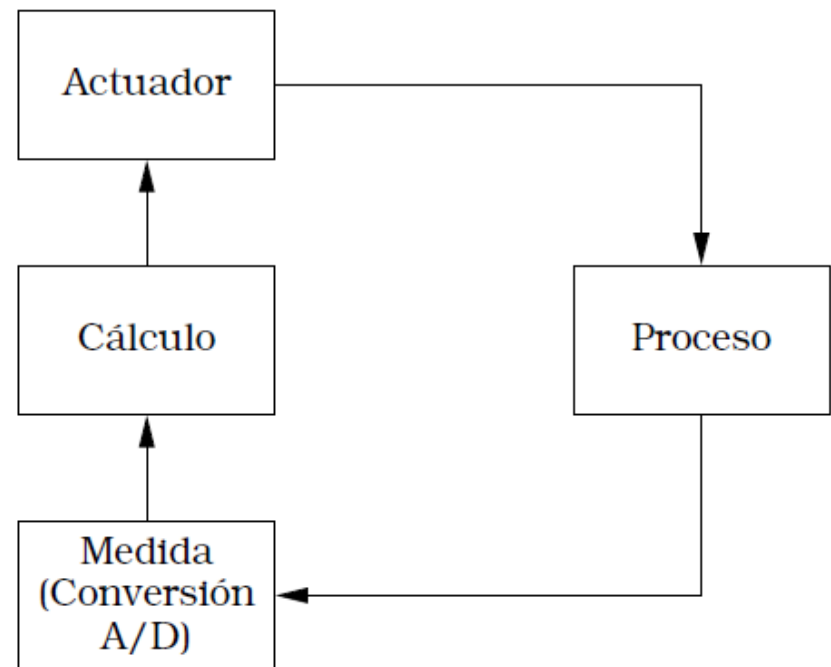
- **Flexible (soft real time).** Pueden no alcanzar el tiempo de generación de la salida ocasionalmente. El valor de la respuesta decrece con el tiempo (adquisición de datos).

- **Firme (firm real time).** Se puede no alcanzar el tiempo de generación de salida ocasionalmente. Una respuesta tardía no tiene valor (ej: sistema multimedia)



# Ejemplos de sistemas en tiempo real.

- Sistemas de Control (control de temperatura de horno, velocidad de motores, de posición, etc.)
- Sistemas de comunicación (reservas aéreas, sistemas bancarios online, etc.).
- Sistemas multimedia (Sistemas que tienen que dibujar pantallas cada 20 ms para ver movimiento fluído).



# Características de los sistemas en tiempo real.

- Gran tamaño y complejidad
  - algunos sistemas de tiempo real tienen millones de líneas de código
  - la variedad de funciones aumenta la complejidad incluso en sistemas relativamente pequeños
- Simultaneidad de acciones (conurrencia)
  - Los dispositivos físicos controlados funcionan al mismo tiempo
  - Los componentes de software que los controlan actúan concurrentemente
- Dispositivos de entrada y salida especiales
  - Los manejadores de dispositivos forman parte del software de aplicación

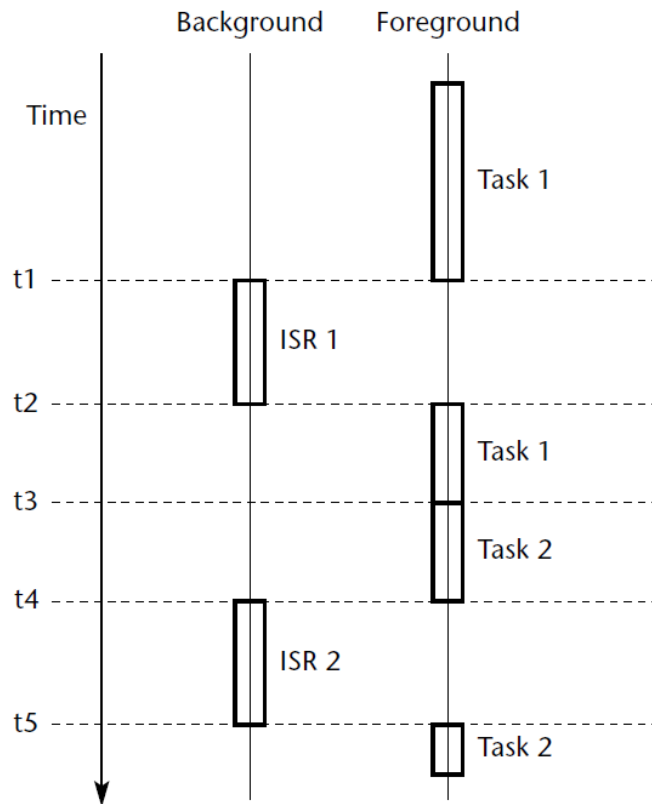
# Características de los sistemas en tiempo real.

- 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 sea determinista
    - No hay que confundirlo con la necesidad de que sea eficiente
    - El sistema debe responder correctamente en todas las situaciones
    - Hay que prever el comportamiento en el peor caso posible

# Sistemas de tiempo real. Resumen.

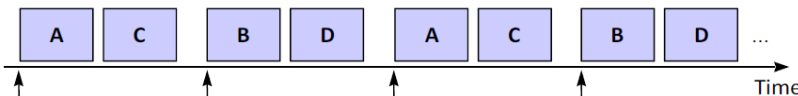
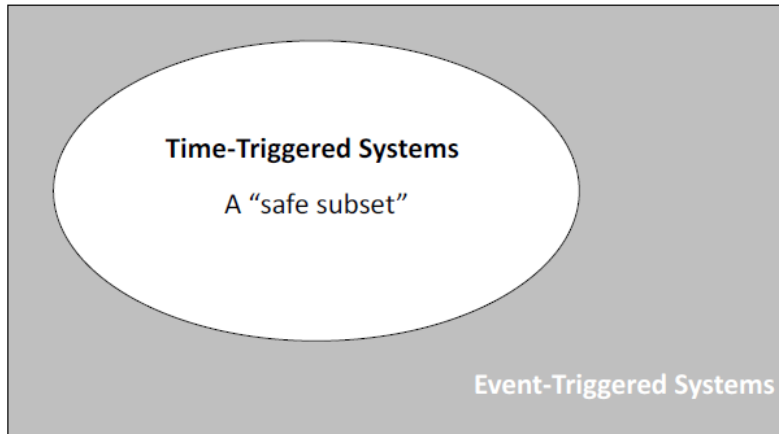
- Los sistemas de tiempo real interaccionan con su entorno y ejecutan sus acciones dentro de intervalos de tiempo determinados
- Tienen requisitos muy exigentes
  - Tamaño y complejidad
  - Concurrencia
  - Interfaces de hardware específicas
  - Fiabilidad y seguridad
  - Determinismo temporal
- Hay dos clases principales de requisitos de tiempo real
  - Tiempo real estricto (hard real-time)
  - Tiempo real flexible (soft real-time)

# Sistemas embebidos gobernados por eventos (EDS).



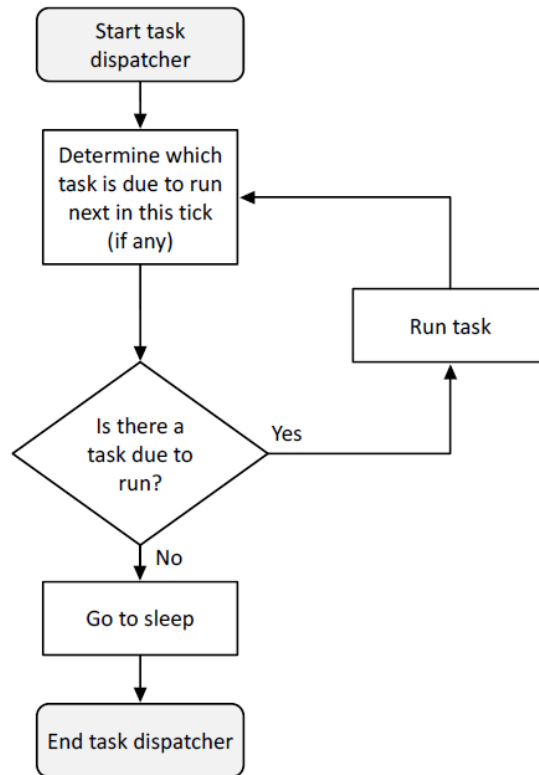
- Los EDS son sistemas embebidos a los que cada evento a manejar (teclas, periféricos, temporizaciones, etc). Son manejados por interrupciones.
- Los EDS son sistemas de mínima latencia pero de previsibilidad compleja ya que las interrupciones pueden aparecer en cualquier momento (por definición las interrupciones son y se consideran asincrónicas).

# Sistemas embebidos gobernados por tiempo (TDS)



- Los sistemas gobernados por tiempo sólo utilizan una única interrupción (de temporización) para ejecutar un planificador.
- El planificador suele ser estático y cooperativo (las tareas están definidas por el/la desarrollador/a y no se interrumpen entre ellas)
- Estas condiciones hacen que sean sistemas de **alta previsibilidad temporal y por lo tanto verificables y validables**.
- Su aplicación típica es en sistemas que requieren alta confiabilidad (safety systems)

# Planificador TDS.

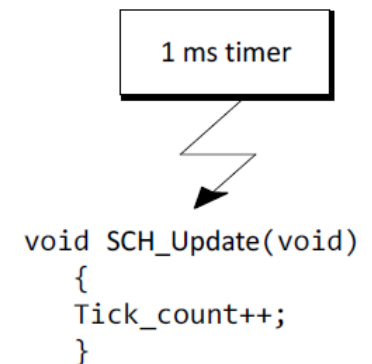


```
int main(void)
{
    SYSTEM_Init();

    SCH_Start();

    while(1)
    {
        SCH_Dispatch_Tasks();
    }

    return 1;
}
```



# Manos a la obra.

- Utilizar un stick de LPC1769(se puede utilizar cualquier otro Cortex-Mx) para las siguientes tareas:
  - Programar el SysTick para que interrumpa una vez por milisegundo, para poder disparar un planificador TDS.
  - Programar el reloj de sistema en 100MHz y verificar que queda en esta configuración (usar LPCOpen)
  - El lazo principal del programa debe dormir al procesador hasta la próxima interrupción.
  - Verificar con las herramientas de debug.

# Bibliografía.

- Sistemas Operativos. Diseño e Implementación. Andrew S. Tanenbaum.
- Using the FreeRTOS Real Time Kernel. NXP LPC17xx Edition. Richard Barry.
- VisualDSP++ 5.0 VDK (Kernel) User's Guide  
[http://www.analog.com/static/imported-files/software\\_manuals/50\\_vdk\\_mn\\_rev\\_3.5.pdf](http://www.analog.com/static/imported-files/software_manuals/50_vdk_mn_rev_3.5.pdf)
- Sistemas Empotrados en Tiempo Real. José Daniel Muñoz Frías.  
[http://www.lulu.com/items/volume\\_67/1349000/1349482/8/print/AnnotationsTR.pdf](http://www.lulu.com/items/volume_67/1349000/1349482/8/print/AnnotationsTR.pdf)
- FreeRTOS <http://www.freertos.org/>
- Sistemas de Tiempo Real y Lenguajes de Programación. Alan Burns, Andy Wellings. Tercera Edición. Addison Wesley.