Embebidos. Tareas y planificación

Juan Alarcón. jalarcon@frba.utn.edu.ar

Sistemas embebidos

- Los sistemas embebidos (embedded systems) se los puede definir como “un hardware” programable corriendo software diseñado para ejecutar una tarea específica.
- Ejemplos de este tipo de sistemas son: lavarropas, equipos de aire acondicionado, control de caldera de calefacción, semáforos, receptores de GPS, controladores de discos rígidos, sistemas de inyección electrónica, sistemas de antibloqueo de ruedas (ABS), de control de tracción (TCS), juguetes (Simon, por ejemplo), control de estabilización de un avión, etc.
- Los sistemas embebidos por lo general se enfrentan con las siguientes restricciones:
  - ***Memoria (Memory footprint):*** en muchas aplicaciones tienen que ser sistemas de bajo costo, tamaño y consumo. Esto suele limitar de manera notable la cantidad de memoria disponible.
  - ***Consumo:*** muchos embebidos son alimentados a batería, eso hace que se requiera esfuerzo de hardware y software para limitar la energía utilizada por el sistema.
  - ***Comportamiento en tiempo real:*** Se requiera un cómputo correcto en un tiempo acotado y predecible.
  - ***Seguridad (safety):*** Hay embebidos que si fallan pueden generar muertes, grandes pérdidas económicas y/o ambientales. Se debe poder estimar la probabilidad de fallas de estos sistemas.

Tareas

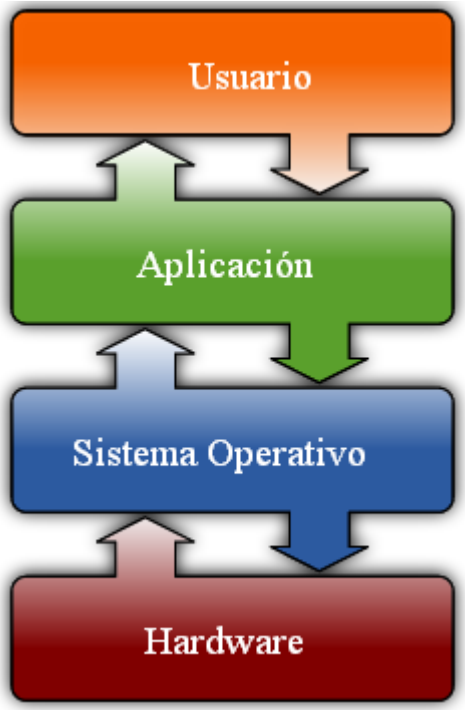
- Si bien un sistema embebido (embedded system) se los puede definir como “un hardware” programable corriendo software diseñado para realizar un misión específica, este se lo puede dividir en unidades funcionales más pequeñas denominadas ***tareas.***

- Desde un punto de vista de la aplicación, la suma de las ***tareas del sistema*** es la que va a cumplir con la misión del mismo y se van a ejecutar de manera ***cuasi independiente entre sí y con la “ilusión” de correr en paralelo.***

- Desde el punto de vista de la materia, el concepto de tarea, proceso o hilo van a ser sinónimos, si bien en otros contextos no va a ser válido (TD3, por ejemplo)

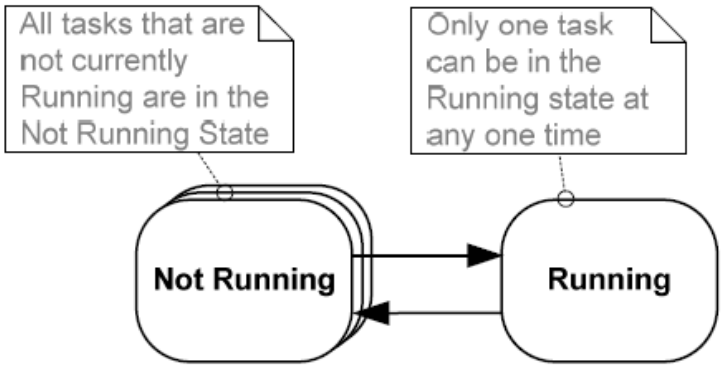
Sistema Operativo.

- Sistema Operativo: Programa o conjuntos de programas que proveen servicios a los programas de aplicación.
- En una definición amplia un sistema operativo es un software que provee un modelo de tareas y un mecanismo para la gestión de las mismas (planificación).
- Un sistema operativo en tiempo real (RTOS) intenta tener latencia de interrupciones y de cambio de tarea mínimas.

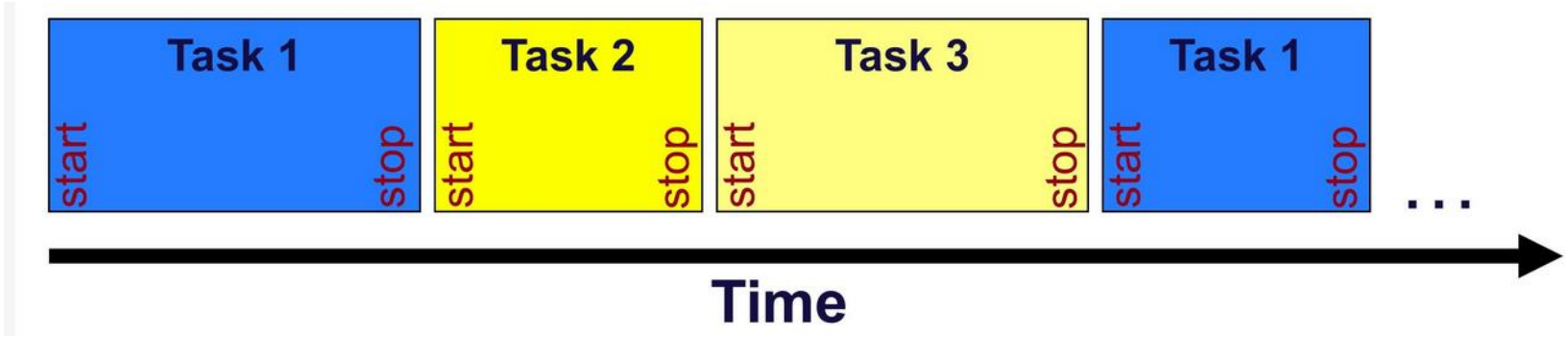


Planificador (scheduler).

- Una parte fundamental del sistema son las tareas, la otra 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 (en principio) van a tener por lo pronto dos estados:
  - Ejecutando
  - En espera



Planificador Cooperativo.

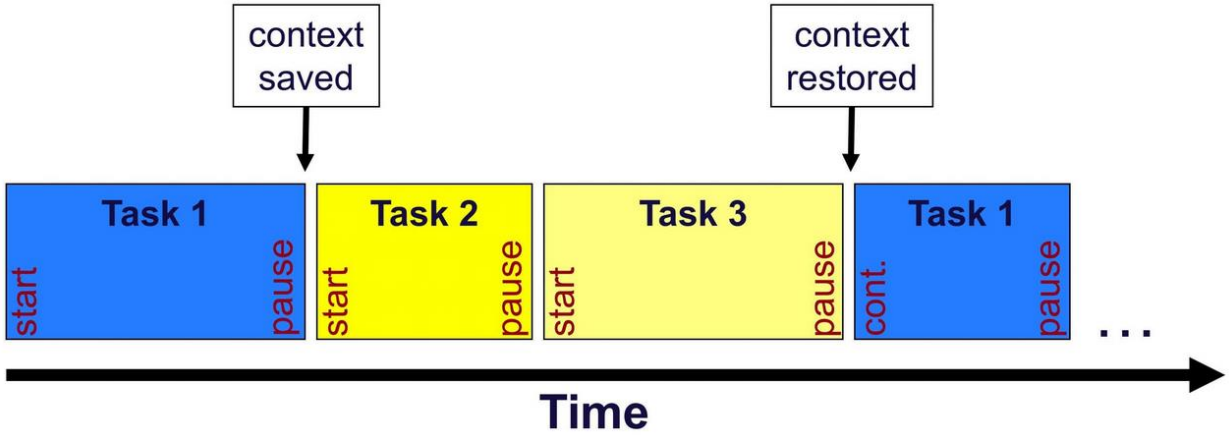


- El planificador cooperativo o RTC (run to completion scheduler) es el más sencillo de todos. Básicamente es una lista de tareas/funciones que se va a ejecutar en alguna secuencia (lista de secuencias) definida por el diseñador del sistema.

Planificador Cooperativo (2)

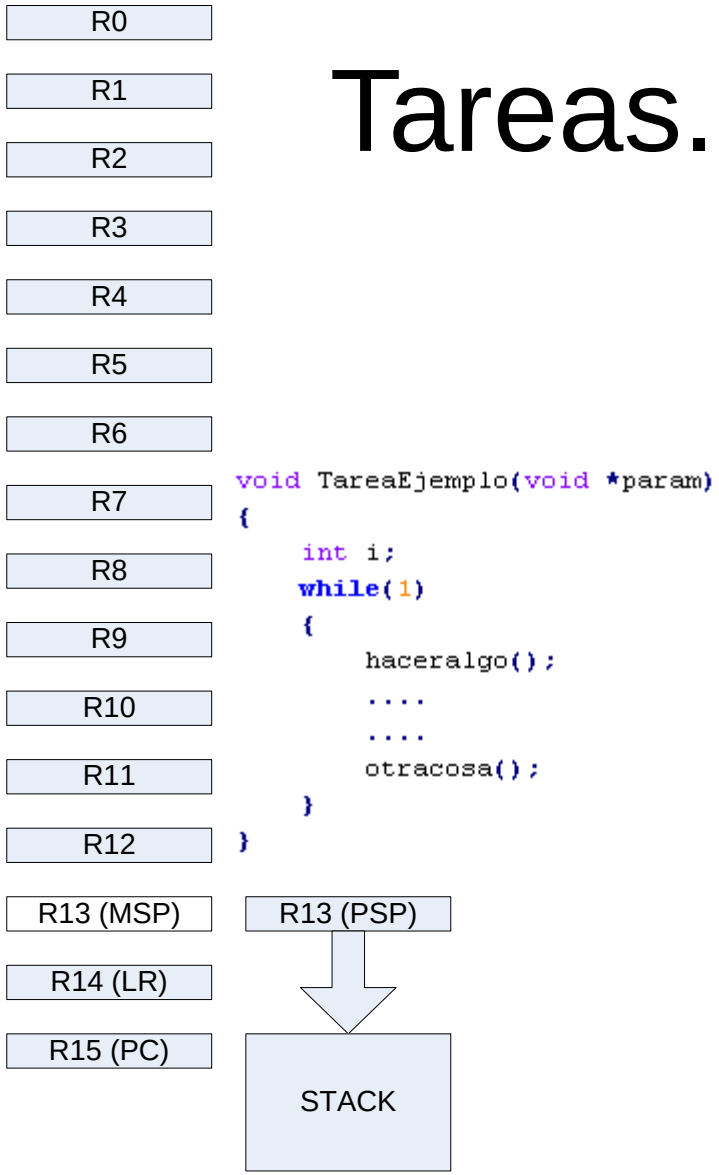
- Es muy sencillo y usa pocos recursos, por lo que suele ser tomado como primera opción, si la aplicación lo permite. ☺
- Suele ser muy portable porque en general no requiere de partes en assembler. ☺
- La comunicación entre tareas es muy sencilla y el fácil predecir su comportamiento. ☺
- Si una tarea se cuelga / pierde el sistema se pierde completo. ☹
- Si bien se puede estimar la latencia a un evento, la tarea que está corriendo tiene prioridad sobre todas las demás hasta su fin y después según el orden fijado. ☹

Planificador Round Robin (RR)



- El planificador RR no necesita que las tareas terminen, sino que el planificador por algún mecanismo (excepción) salva el contexto y ejecuta la tarea siguiente a pedido de la tarea.
- Es más flexible y complejo que el planificador cooperativo, la gran ventaja es que la planificación es independiente del código de las tareas.
- Su desventaja, la comunicación entre tareas es más compleja y la carga del planificador es mayor.
- Se necesita mantener una pila por cada tarea.

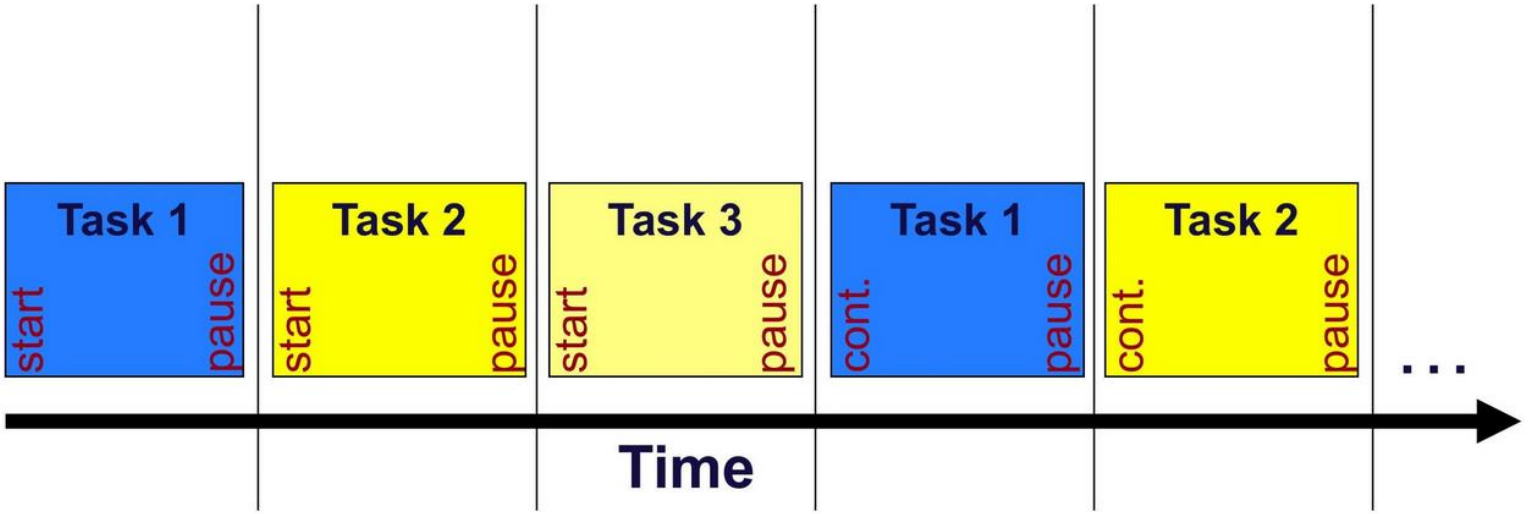
Tareas. Contexto.




```
void TareaEjemplo(void *param)
{
    int i;
    while(1)
    {
        haceralgo();
        ....
        ....
        otracosa();
    }
}
```

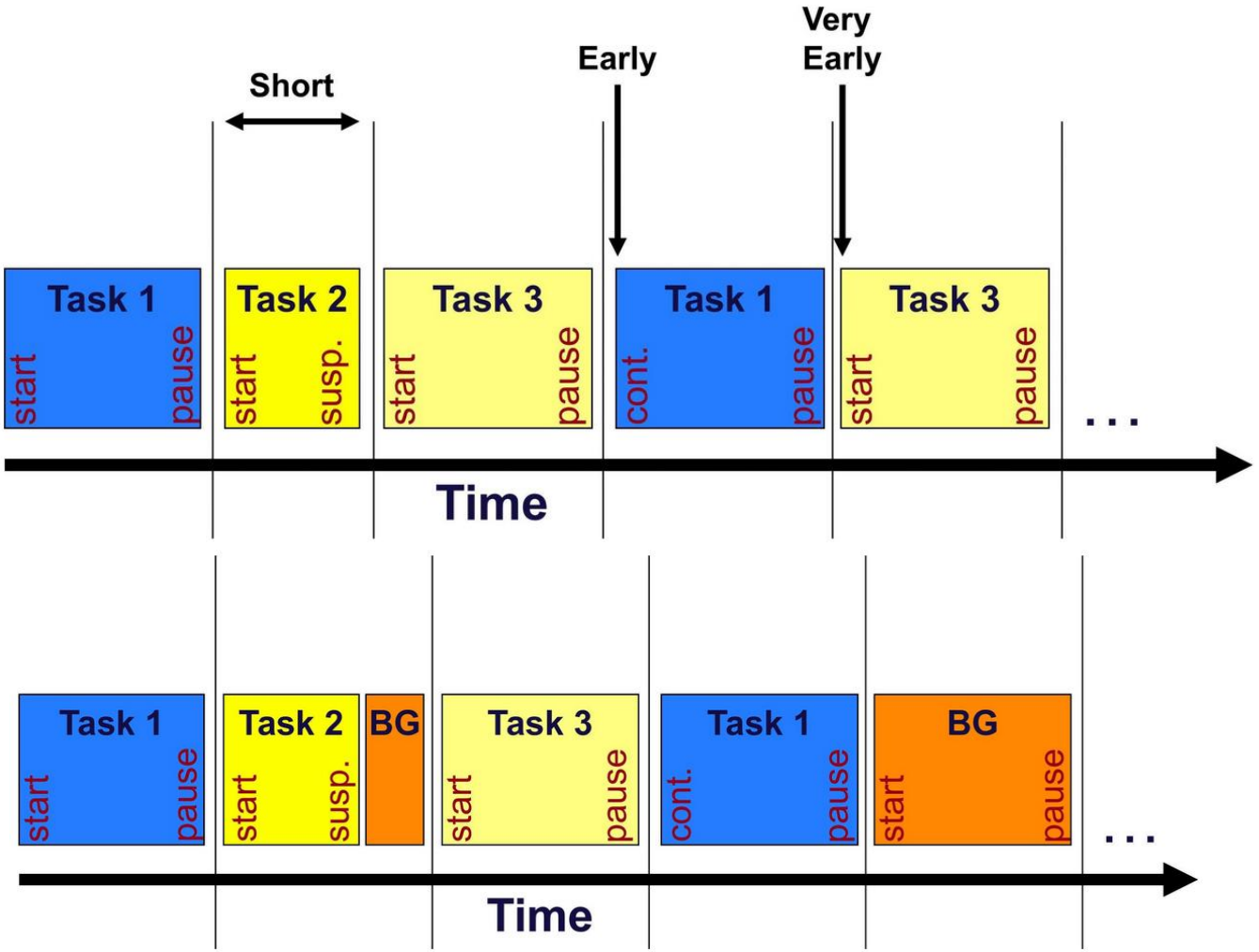
- El contexto va a estar definido por el contenido de ***todos los registros del procesador***, la pila del proceso y el código de la tarea, su estado y prioridad.
- La tarea va a estar compuesta por el programa y la “foto” del procesador en ese momento.

Planificador por división de tiempo



- El planificador por división de tiempo (time slice) es el paso siguiente al round robin. El planificador le asigna a cada tarea una cantidad de ticks en la que se ejecuta y luego cambia a la siguiente.
- Es simple de entender y predecible.
- La desventaja que puede presentar este sistema es que si una tarea termina o entrega el control antes de tiempo el sistema pierde la temporización.

Planificador por división de tiempo (2)

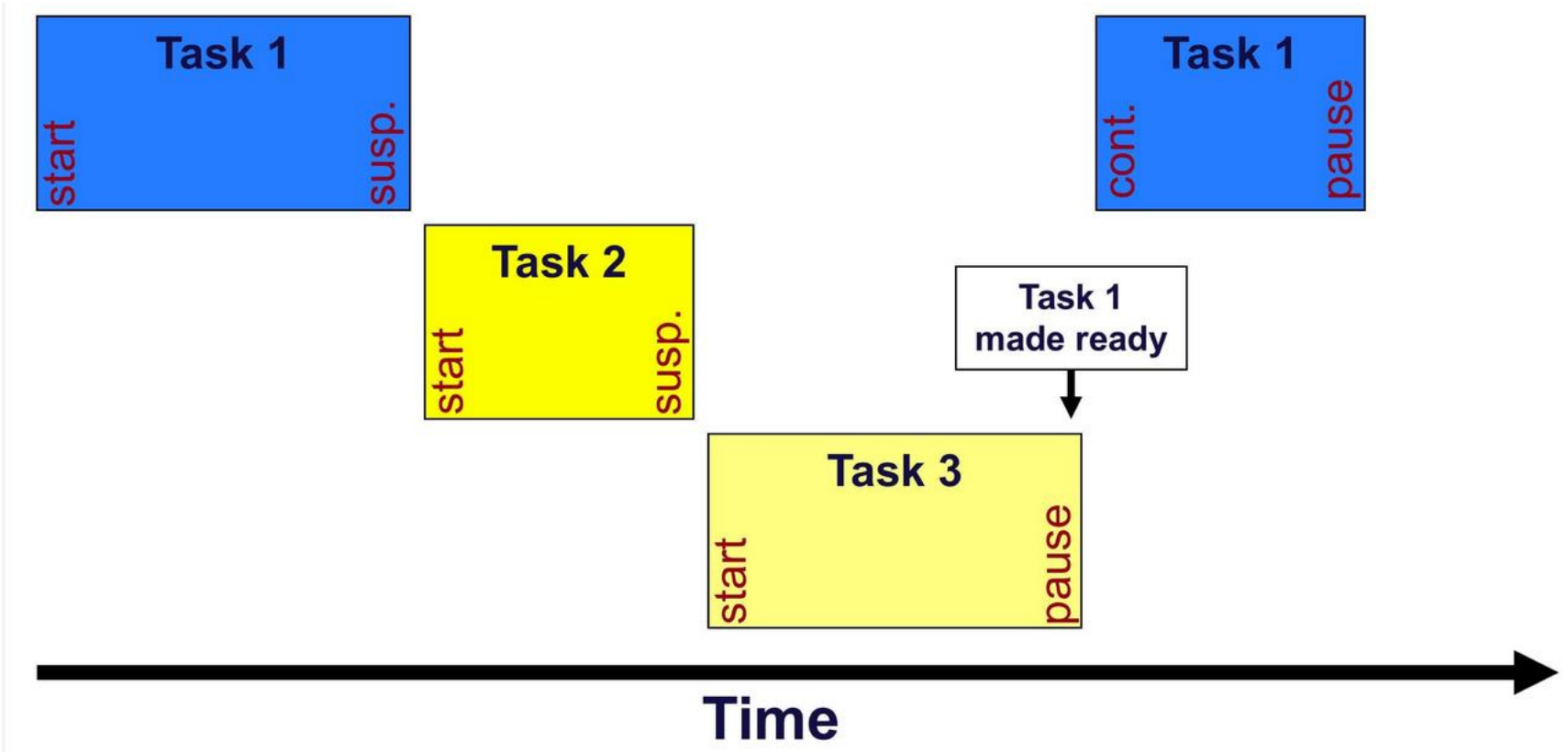


Planificador por división de tiempo (3)

- El planificador TS puede agregar una tarea que no hace nada (background - idle), sólo consumir tiempo, para evitar que el sistema pierda la temporización.

- Este tipo de sistemas no es muy apto para atender eventos que requieran poca latencia ya que en el peor caso hay que esperar los ticks de todas las tareas.

Planificador apropiativo



Planificador de tipo apropiativo (preemptive – priority scheduler).

- Es un planificador que cambia de tarea ***en función de eventos.***
- 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.

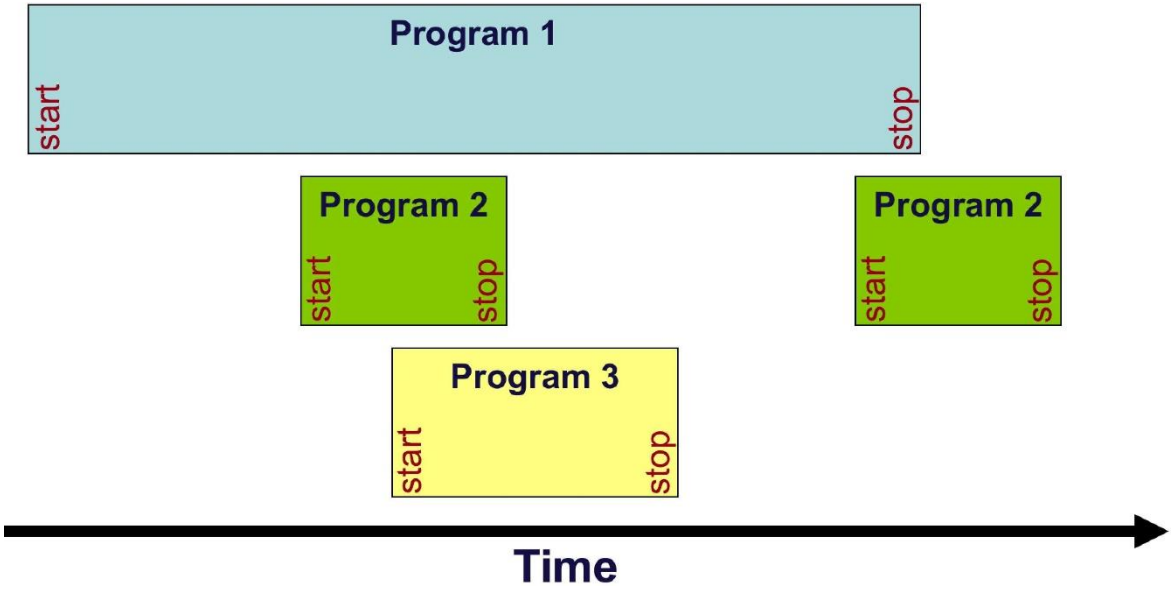
Planificadores compuestos

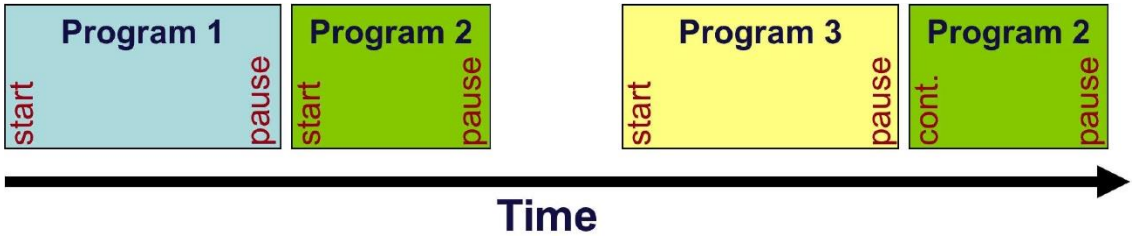
- Los modelos de planificación no son excluyentes entre sí.
- La mayoría de los planificadores toman enfoques compuestos de los modelos planteados, por ejemplo, FreeRTOS es apropiativo, pero dentro de la misma prioridad de tareas usa una planificación TS.

¿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

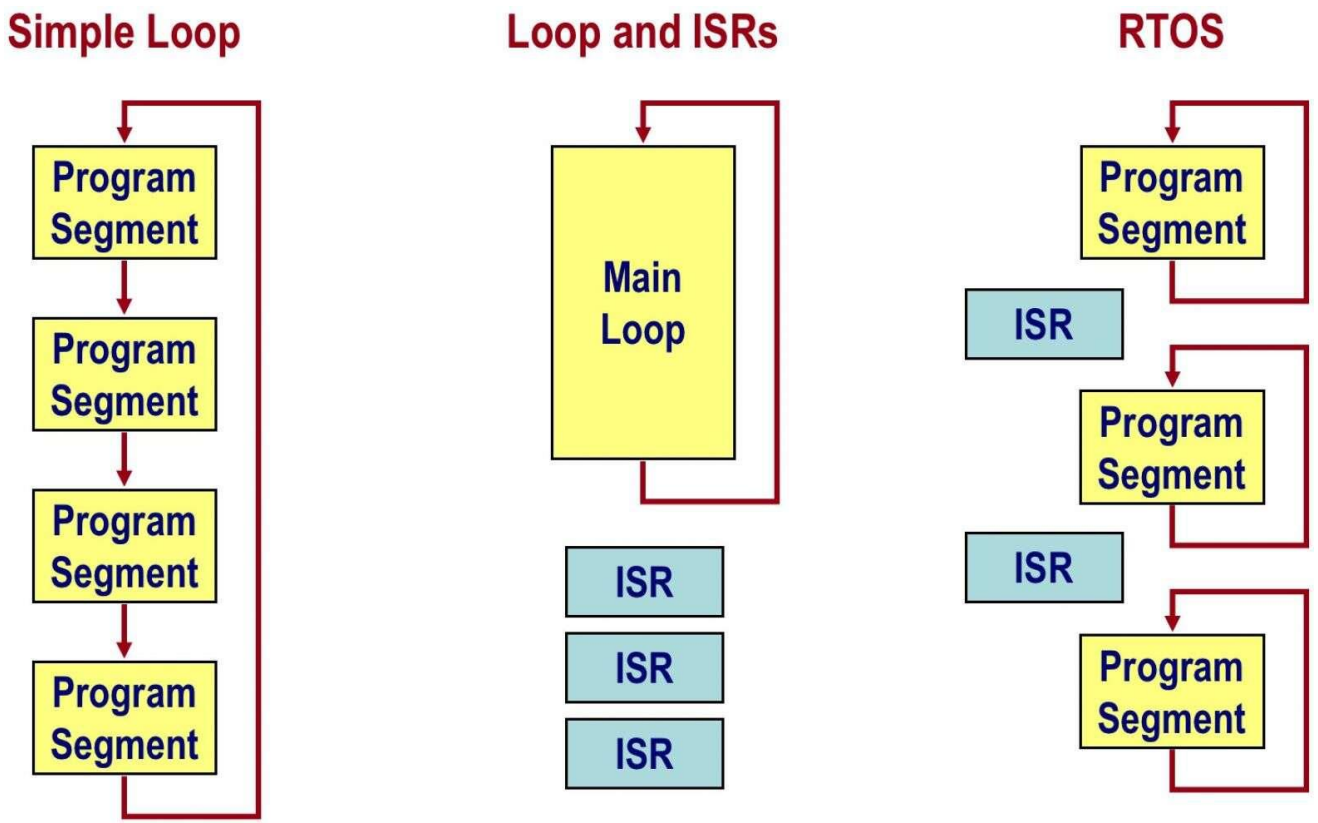
Estructura de software en un embebido





- Antes de elegir la necesidad o no de un SO es necesario estudiar la solución a implementar respecto de las restricciones de tiempo real, memoria, confiabilidad, interfaz .
- ¿Qué tipo de sistema utilizaría para las estructuras de programa de la izquierda y por que?

Estructura de un embebido



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
- Sistemas Empotrados en Tiempo Real. José Daniel Muñoz Frías. http://www.lulu.com/items/volume_67/1349000/1349482/8/print/ApuntesTR.pdf
- FreeRTOS http://www.freertos.org/
- Sistemas de Tiempo Real y Lenguajes de Programación. Alan Burns, Andy Wellings. Tercera Edición. Addison Wesley.