Carrera de Especialización en Sistemas Embebidos

Sistemas Operativos en Tiempo Real

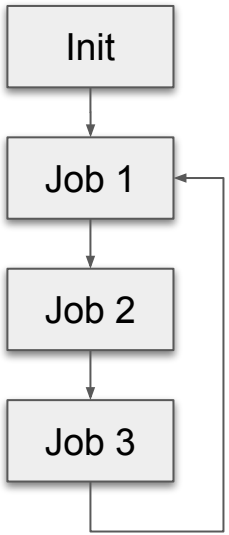
Clase 1: Introducción a los RTOS

ACSE
Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos
FACULTAD DE INGENIERIA
Universidad de Buenos Aires
CIM
CADIEEL

Técnicas baremetal

```
void main()
{
    Inicializar_Harware();

    Inicializar_Job1();
    Inicializar_Job2();

    while(1)
    {
        Job1();
        Job2();
    }
}
```

- Superloop
  - Repetición secuencial de jobs
  - Comportamiento cooperativo



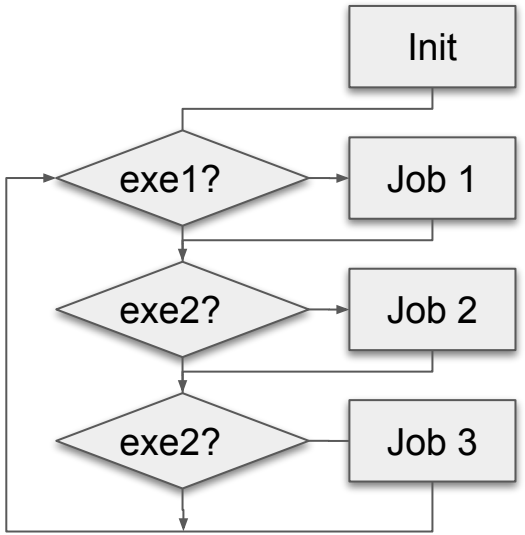
Arquitectura conocida como super-loop.

Técnicas baremetal

```c
int f1;

void main()
{
    Inicializar_Harware();
    Inicializar_Isr();
    Inicializar_Job1();

    f1=0;

    while(1)
    {
        if(f1)
        {
            Job1();
            f1=0;
        }
    }
}
```

```c
void isr()
{
    f1=1;
}
```

- Foreground:
  - Ejecución más prioritarias (ISR)
- Background:
  - Ejecución menos prioritarias (main)

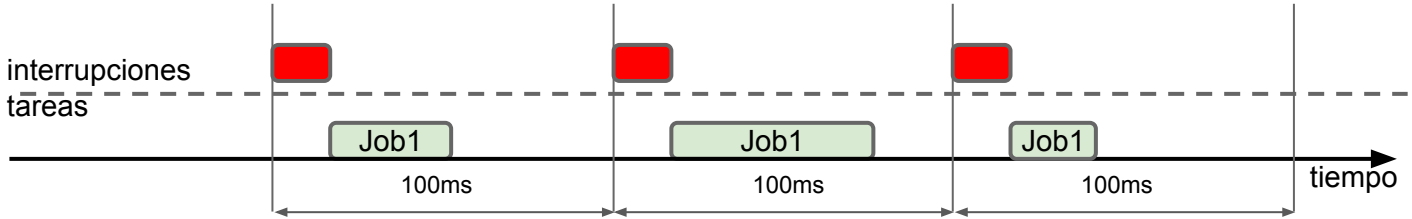


Tarea periódica

- Tareas periódicas
  - Ejecutan acciones constantemente y a una frecuencia determinada.
  - Los JOBs son disparados por tiempo.
    - Ej: Destellar un led, muestrear una señal, corregir un lazo de control, etc.

```
int f1;

void main()
{
    /* inicialización */
    Inicializar_Timer(100);
    f1=0;
    while(1)
    {
        if(f1)
        {
            Job1();
            f1=0;
        }
    }
}
```

```
void timer_isr()
{
    f1=1;
}
```

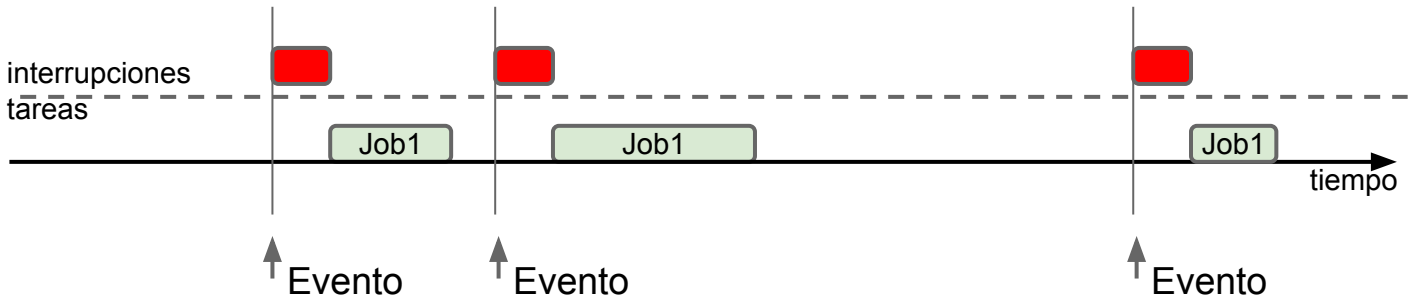


Tarea "disparo único"

- Tareas "disparo único" (asincronico, esporadicas, reactivas)
  - Atienden eventos que no se sabe cuándo van a ocurrir. Estas tareas están inactivas hasta que ocurre el evento de interés.
    - Ej: una parada de emergencia.

```c
int f1;

void main()
{
    /* inicialización */
    Inicializar_GPIO();
    f1=0;
    while(1)
    {
        if(f1)
        {
            Job1();
            f1=0;
        }
    }
}
```

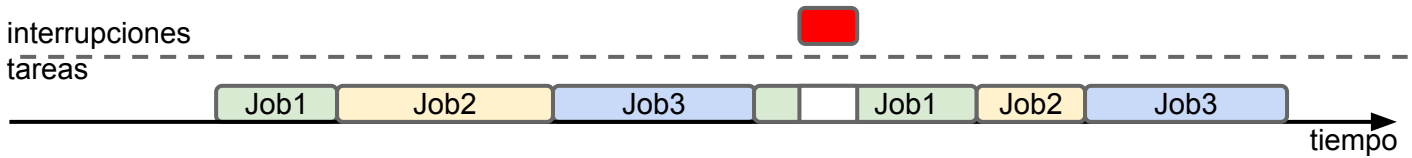
```c
void gpio_isr()
{
    f1=1;
}
```



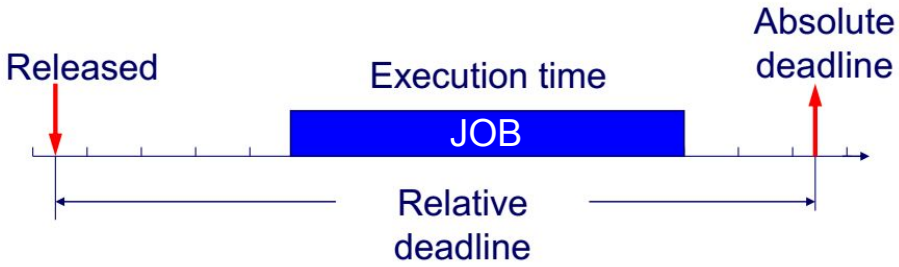
Tareas de procesamiento continuo.

- Tareas de procesamiento continuo
  - Son tareas que trabajan en régimen permanente (sin una periodicidad definida)
    - Ej: tareas de mantenimiento, en general de baja prioridad.

```
int f1;

void main()
{
    /* inicialización */

    while(1)
    {
        Job1();
        Job2();
        Job3();
    }
}
```

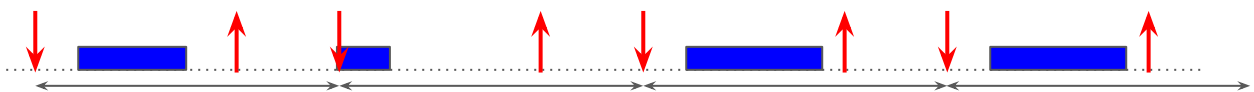


Ciclo de vida de las tareas



Atributo	Descripción
Release Time	Momento en el tiempo en el cual el job se vuelve disponible para ejecución
Relative Deadline	Tiempo máximo aceptable para que la ejecución del job sea completada.
Execution time	Tiempo que tarda el job en ejecutarse (peor caso)

Tareas periódicas

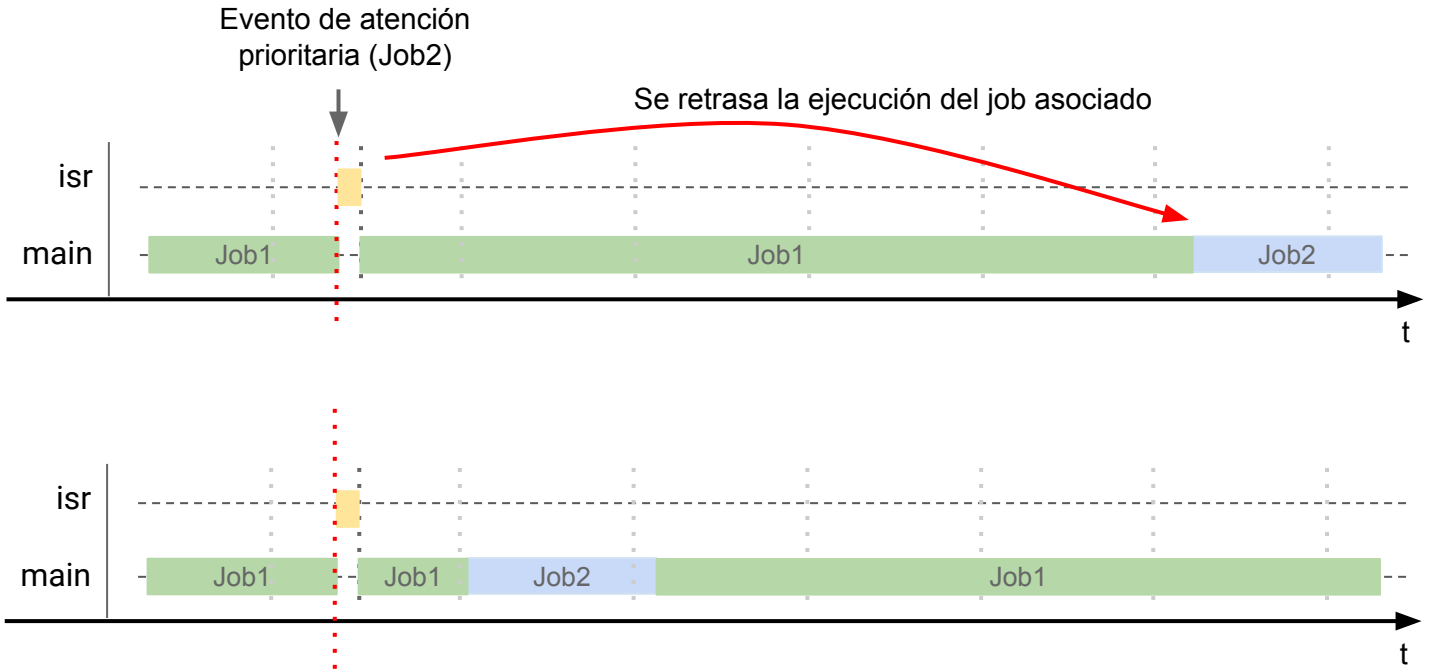


Tareas "disparo único"

Tareas continuas

Modo de operación cooperativo

- Las tareas deben cooperar para que todas posean el tiempo de CPU para que permitan su ejecución.



¿ Cómo querríamos que se comporte ?

Modo de operación cooperativo

- ¿Como hacer que las tareas "cooperen"?

Los JOBS que duran mucho se deben diseñar:

Multietapa

```
Job1();  ──►  Job1_pt1();

              Job1_pt2();
```

```
void main()
{
    f1=0;

    while(1)
    {
        Job1_pt1();
        if(f1)
        {
            Job2();
            f1=0;
        }
        Job1_pt2();
    }
}
```

Multiestado

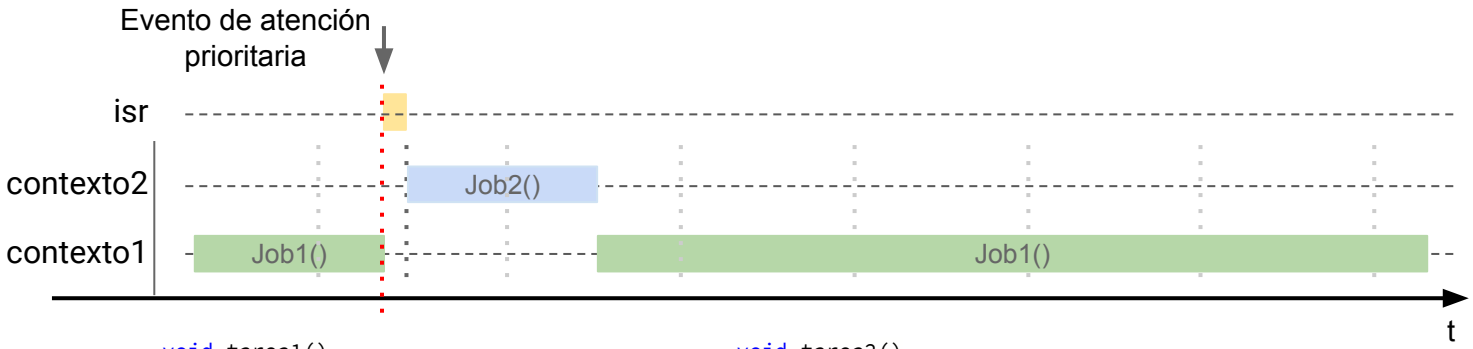
```
uint8_t j1_state=0;
void Job1()
{
    if(j1_state==0)
    {
        Job1_pt1();
        j1_state=1;
    }
    else if(j1_state==1)
    {
        Job1_pt1();
        j1_state=0;
    }
}
```

```
void main()
{
    f1=0;

    while(1)
    {
        Job1();
        if(f1)
        {
            Job2();
            f1=0;
        }
        Job1();
    }
}
```

Protothreads

Modo de operación apropiativo

- Se genera una pausa en la ejecución de la tarea actual, y se le brinda el uso de CPU a la nueva (la nueva tarea, se “apropia” del CPU). En ese momento ocurre el cambio de contexto.

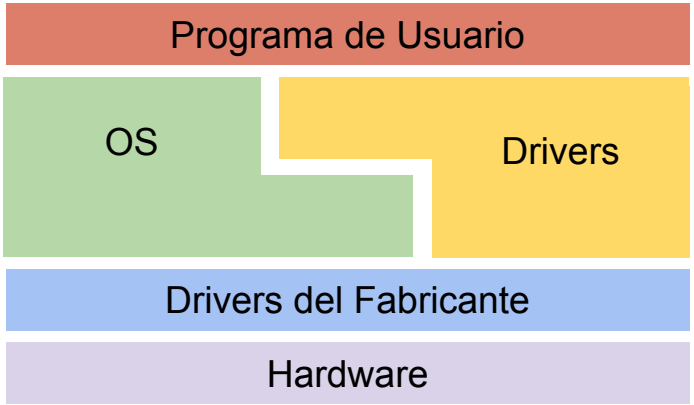



```c
void tarea1()
{
    while(1)
    {
        Job1();
    }
}
```

```c
void tarea2()
{
    evento_importante=0;

    while(1)
    {
        if(evento_importante)
        {
            Job2();
            evento_importante = 0;
        }
    }
}
```

¿ Qué es un OS?

- Es un conjunto de programas que ayuda al programador de aplicaciones a gestionar el tiempo del procesador.



- Asigna tiempo de ejecución a todos los programas que tiene cargados, en base a un juego de reglas conocido de antemano.

- A estos subprogramas se los llama tareas.

¿Qué es un RTOS?

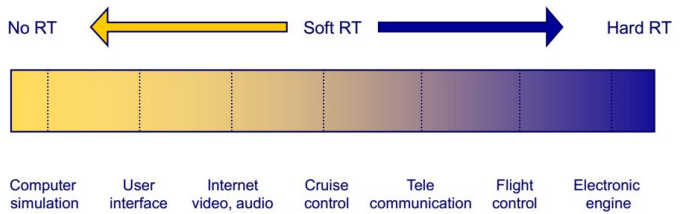
- Es un OS que está diseñado para que los programas de aplicación puedan cumplir compromisos temporales definidos por el programador.

- Se utilizan cuando tiempo de respuesta ante ciertos eventos es un parámetro crítico.
  - Respuestas demasiado tempranas o muy tardías podrían ser indeseables.

- Un RTOS se emplea cuando hay sistema funciona correctamente no solo con los resultados lógicos, sino también el tiempo en que se producen.

Tipos de RTOS: Según tiempo de respuesta

- Duros (Hard Real Time):
  - Restricciones de tiempo estrictas
  - Consecuencias catastróficas si se pierden metas.
  - Ejemplos:
    - Controles de lazo cerrado (controles de motores, aviónica):
      - Los retrasos afectan la estabilidad
- Blandos (Soft Real Time):
  - Restricciones de tiempo menos rigurosas
  - No se considera crítico que no se cumplan los plazos.
  - Ejemplos:
    - Interfaz de teclado al usuario
    - Juegos
    - Servidor web



Scheduler, dispatcher y base de tiempo

- Scheduler (planificador):
  - Determina qué tarea debe estar en ejecución a cada momento.
  - Utiliza uno o más algoritmos para decidir cual tarea ejecutar y éstos se denomina POLÍTICA DE PLANIFICACIÓN

- Dispatcher: Provee el control del CPU a una cierta tarea.

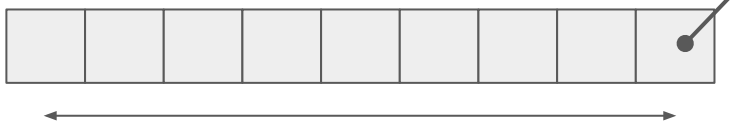
- Base de tiempo (tick): Provee la mínima unidad de medida de tiempo para las operaciones temporizadas del OS (delays y timeouts).

Caso estudio: SEOS PONT

- Todo el sistema está en C, modularizado en forma de biblioteca:
  - Portable muy facilmente.
- El sistema es simple y adecuado a plataformas con bajos recursos (CPU ,RAM)
- Si se requiere alta velocidad de respuesta, el sistema debe estar diseñado cuidadosamente.
- URL del autor:
  - Libro
  - Ejemplos de código
- Port sobre sAPI + ejemplos

SEOS PONT: Arquitectura

Array con bloques de control de as tareas planificadas.



Máxima cantidad de jobs planificables

```
typedef struct{
   //Puntero a la tarea
   void (* pTask)(void*);
   /* puntero a zona de memoria conparámetros para la tarea */
   void* taskParam;
   /* tiempo que falta (en ticks) para que se ejecute el Job */
   int32_t delay;
   /* tiempo de periodicidad (en ticks) del Job */
   int32_t period;
   /* flag que indica que el Job debe ejecutarse */
   int32_t runMe;
} sTask_t;
```

API

```
int32_t schedulerAddTask( void (* pTask)(void*), void* taskParam, const int32_t DELAY, const int32_t PERIOD )

int8_t schedulerDeleteTask( int32_t taskIndex )

void schedulerDispatchTasks( void )

void schedulerInit( void )

void schedulerUpdate( void *ptr )

void schedulerStart( tick_t tickRateMs )
```

OS apropiativo

- Permite al programador de aplicaciones escribir múltiples tareas como si cada una fuera la única que utiliza la CPU.

- Con esto se logra la ilusión de que múltiples programas se ejecutan simultáneamente, aunque en realidad sólo pueden hacerlo de a uno a la vez (en sistemas con un sólo núcleo).

Contexto de Ejecución en Baremetal

- Se llama contexto de ejecución el mínimo conjunto de recursos utilizados por una tarea con los cuales se permita ejecutarse:
  - IP (instruction pointer)
  - SP (stack pointer)
  - Registros del CPU
  - Contenido de la pila en uso

- Cuando ocurra una interrupción, hay un cambio de contexto automático (por hardware).
  - Se guardan registros importantes, PC, etc

- Todo ocurre con un mismo stack.

Contexto de Ejecución en Apropiativo

- Para simular que hay varias tareas en paralelo, se debe crear un contexto "artificial" para cada una.
- Cuando el planificador determina que debe cambiarse el contexto de ejecución, invoca al dispatcher para que guarde el contexto completo de la tarea actual y lo reemplace por el de la tarea entrante.
- Cada uno de estos contextos incluirá:
  - Registros importantes del CPU
  - Stack de memoria propio
  - Funcion "main" propia (código de la tarea en si)

¿ Como se realiza el cambio de contexto?

Estado del CPU

Estado: Running

```
TAREA( nombre_de_tarea1 )
{
    char muestras_temp[100];
    char muestra;

    inicializar( muestras_temp );

    while(1)
    {
        muestra = obtener_temperatura();

        agregar_muestra( muestras_temp ,  muestra );
    }

    TERMINAR_TAREA();
}
```

Estado: Ready

```
TAREA( nombre_de_tarea2 )
{
    char muestras_pres[100];
    char muestra;

    inicializar( muestras_pres );

    while(1)
    {
        muestra = obtener_presion();

        agregar_muestra( muestras_pres ,  muestra );
    }

    TERMINAR_TAREA();
}
```

Estado del CPU de Tarea 2

¿ Como se realiza el cambio de contexto?

Estado del CPU

Estado: Reading

```
TAREA( nombre_de_tarea1 )
{
    char muestras_temp[100];
    char muestra;

    inicializar( muestras_temp );

    while(1)
    {
        muestra = obtener_temperatura();

        agregar_muestra( muestras_temp ,  muestra );
    }

    TERMINAR_TAREA();
}
```

Estado del CPU de Tarea 1

Estado: Reading

```
TAREA( nombre_de_tarea2 )
{
    char muestras_pres[100];
    char muestra;

    inicializar( muestras_pres );

    while(1)
    {
        muestra = obtener_presion();

        agregar_muestra( muestras_pres ,  muestra );
    }

    TERMINAR_TAREA();
}
```

Estado del CPU de Tarea 2

¿Cuándo se realiza el cambio de contexto?

- Los cambios de contexto se realizan de forma transparente para la tarea. Cuando la tarea retoma su ejecución no muestra ningún síntoma de haberse pausado alguna vez.

- Ocurren cuando el planificador lo determina.

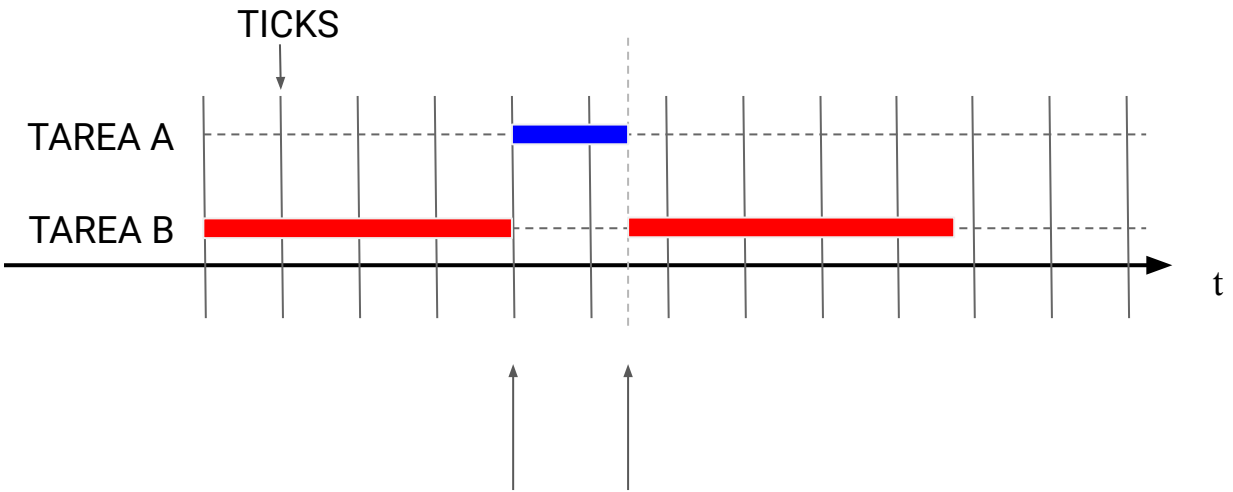
- Ejemplos:

  - Luego que se cumpla un cierto tiempo

  - Que el programador llame a una API que, indirectamente, invoque al scheduler.

    - Estos llamados se conocen como RESCHEDULING POINT

¿Cuándo se realiza el cambio de contexto?



En estos momentos el planificador decide (con algún criterio) que se debe producir el cambio.

Notar que los cambios pueden ocurrir alineados con el tick del sistema, o sin estar alineados.

¿Por qué usar un RTOS?

- Para cumplir con compromisos temporales estrictos
  - El RTOS ofrece funcionalidad para asegurar que una vez ocurrido un evento, la respuesta ocurra dentro de un tiempo acotado. Es importante aclarar que esto no lo hace por sí solo sino que brinda al programador herramientas para hacerlo de manera más sencilla que si no hubiera un RTOS.

- Para no tener que manejar el tiempo “a mano”
  - El RTOS absorbe el manejo de temporizadores y esperas, de modo que hace más fácil al programador el manejo del tiempo.

- Para contar con un framework de trabajo con muchas herramientas ya probadas y funcionales.

¿Por qué usar un RTOS?

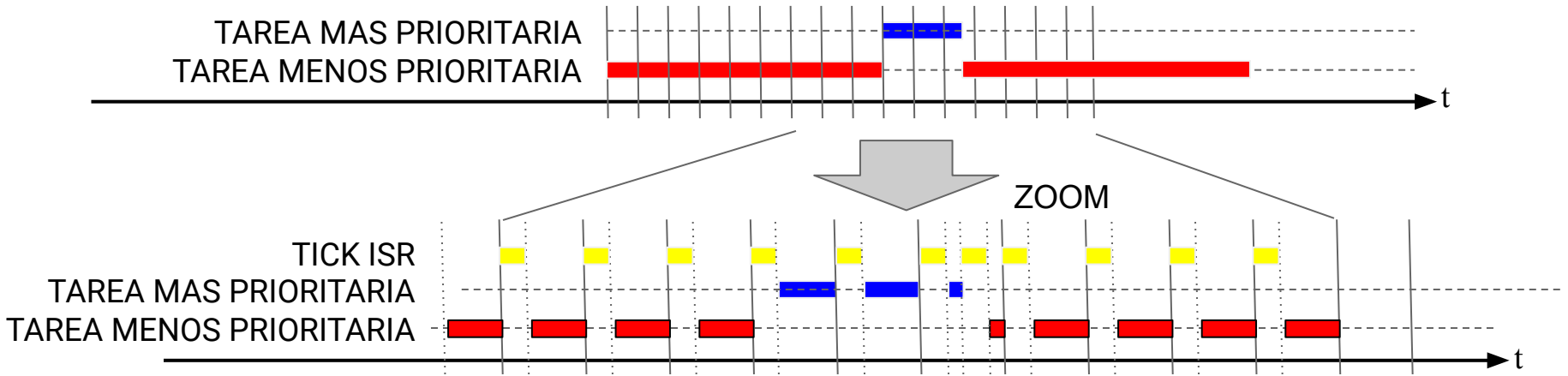
- Multitarea
  - Simplifica sobremanera la programación de sistemas con varias tareas.

- Escalabilidad
  - Al tener ejecución concurrente de tareas se pueden agregar las que haga falta, teniendo el único cuidado de insertarlas correctamente en el esquema de ejecución del sistema.

- Mayor reusabilidad del código
  - Si las tareas se diseñan bien (con pocas o ninguna dependencia) es más fácil incorporarlas a otras aplicaciones.

Letra chica 1

- Latencias temporales
  - Se gasta tiempo del CPU en determinar en todo momento qué tarea debe estar corriendo. Si el sistema debe manejar eventos que ocurren demasiado rápido tal vez no haya tiempo para esto.
  - Se gasta tiempo del CPU cada vez que debe cambiarse la tarea en ejecución.



Letra chica 2

- Memoria
  - Se gasta memoria de código para implementar la funcionalidad del RTOS.
  - Se gasta memoria de datos en mantener una pila y un TCB (bloque de control de tarea) por cada tarea.

- Para que las partes del sistema que requieren temporización estricta, debe hacerse un análisis de tiempos, eventos y respuestas muy cuidadoso.

  Una aplicación mal diseñada puede fallar en la atención de eventos aún cuando se use un RTOS.

Bibliografia

- https://www.freertos.org
- FreeRTOS Kernel Documentation
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- Introducción a los Sistemas Operativos de Tiempo Real, Pablo Ridolfi, UTN, 2015.
- Introducción a Planificación de Tareas, CAPSE, Franco Bucafusco, 2017
- Introducción a Sistemas cooperativos, CAPSE, Franco Bucafusco, 2017
- FreeRTOS - Temporización, Cursos INET, Franco Bucafusco, 2017
- Rate-monotonic scheduling, Wikipedia Consultado 19-5-2
- Earliest Deadline First, Wikipedia Consultado 19-5-2

Licencia

"Introducción a los RTOS"