

Sistemas Basados en Microprocesador

B2

Introducción a CMSIS-RTOS

Eduardo Barrera

Julian Nieto

Mariano Ruiz



POLITÉCNICA

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE SISTEMAS Y TELECOMUNICACIÓN



Sistemas Basados en Microprocesador

Contenidos

- Introducción a RTOS
- CMSIS-RTOS
- *Threads* (Hilos, Tareas)
 - Scheduler
 - API
 - Ejemplos
- Sincronización entre hilos
 - Señales, colas
 - Acceso a recursos compartidos
- Temporización
- Configuración CMSIS-RTOS – RTX en Keil

Desarrollo de aplicaciones con Microprocesadores

- Aplicaciones sencillas (fácil implementarlas mediante un bucle *while*, algunos *timers* y algunas interrupciones)
 - Si la complejidad de la aplicación crece, el número de temporizaciones aumenta frente al número de timers disponibles en el microprocesador y la lógica de la misma es compleja, la gestión, desarrollo y depuración de la aplicación se complican enormemente.
- Para simplificar esta situación se han desarrollado Sistemas Operativos “ligeros” para ser implementados en arquitecturas basadas en microprocesadores. Su utilización permite que el desarrollo de software sea más sencillo, seguro, y eficiente, redundando en un mantenimiento más sencillo del software.

CMSIS-RTOS

CMSIS-RTOS

- API C/C++ para sistemas operativos en tiempo real
- Diseñado para procesadores Cortex M
- Versión a utilizar **CMSIS-RTOS2**
- Está se puede configurar para usar los *kernels RTX o freeRTOS*

CMSIS-RTOS

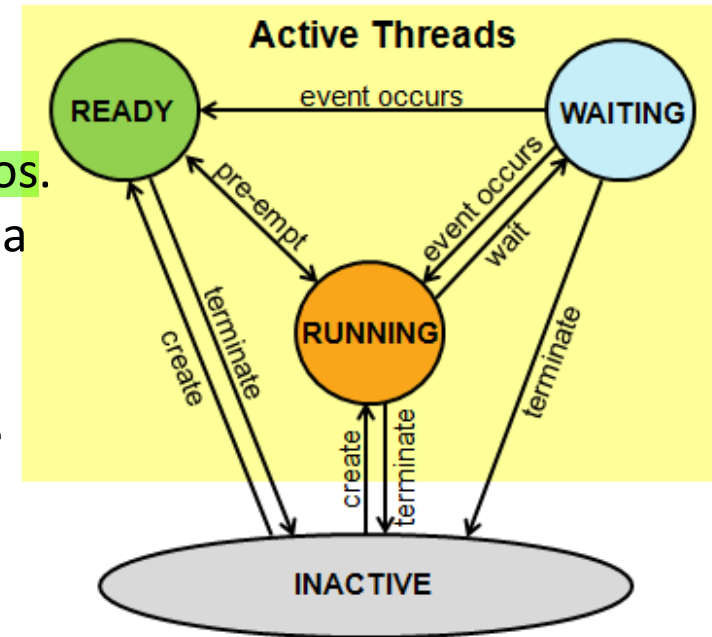
- Aplicaciones multihilo: basada en la utilización de hilos concurrentes
- Aporta mecanismos de comunicación y sincronización entre hilos
- **Thread**
 - Porción de código que realiza una tarea concreta
 - Típicamente es una función con un bucle infinito y sin retorno
 - El RTOS permite compartir la ejecución con otros threads
 - 4 estados: **Running, Ready, Waiting, Inactive**
- **Scheduler**
 - Planificación y compartición de los recursos de CPU
 - Gestiona la ejecución de threads asignando tiempo (SysTick) de procesador a cada hilo
- **Timeslice**
 - Período de tiempo asignado a cada hilo
 - **Múltiplo del Tick (5) generado por el SysTick (1ms)**

RTOS (Planificador)

- **Pre-emptive**
 - Desalojo de hilos “-prioritarios” por hilos “+prioritarios”
- **Round-Robin**
 - Todos los hilos con la misma prioridad
 - Se ejecutan unos detrás de otros en secuencial durante un “*timeslice*”
- **Round-Robin Pre-emptive**
 - Los hilos pueden tener distinta prioridad
 - Los hilos con igual prioridad se ejecutan de forma Round-Robin mientras no haya otro hilo de +prioridad en estado READY
 - Cuidado con las prioridades para no “colgar” la aplicación
 - Por defecto en CMSIS-RTOS - RTX
- **Cooperative Multitasking**
 - Todos los hilos con igual prioridad
 - No Round-Robin
 - Cada hilo se ejecuta hasta bloquearse (pasa a WAIT) o hasta pasar la ejecución a otro (“yield”)

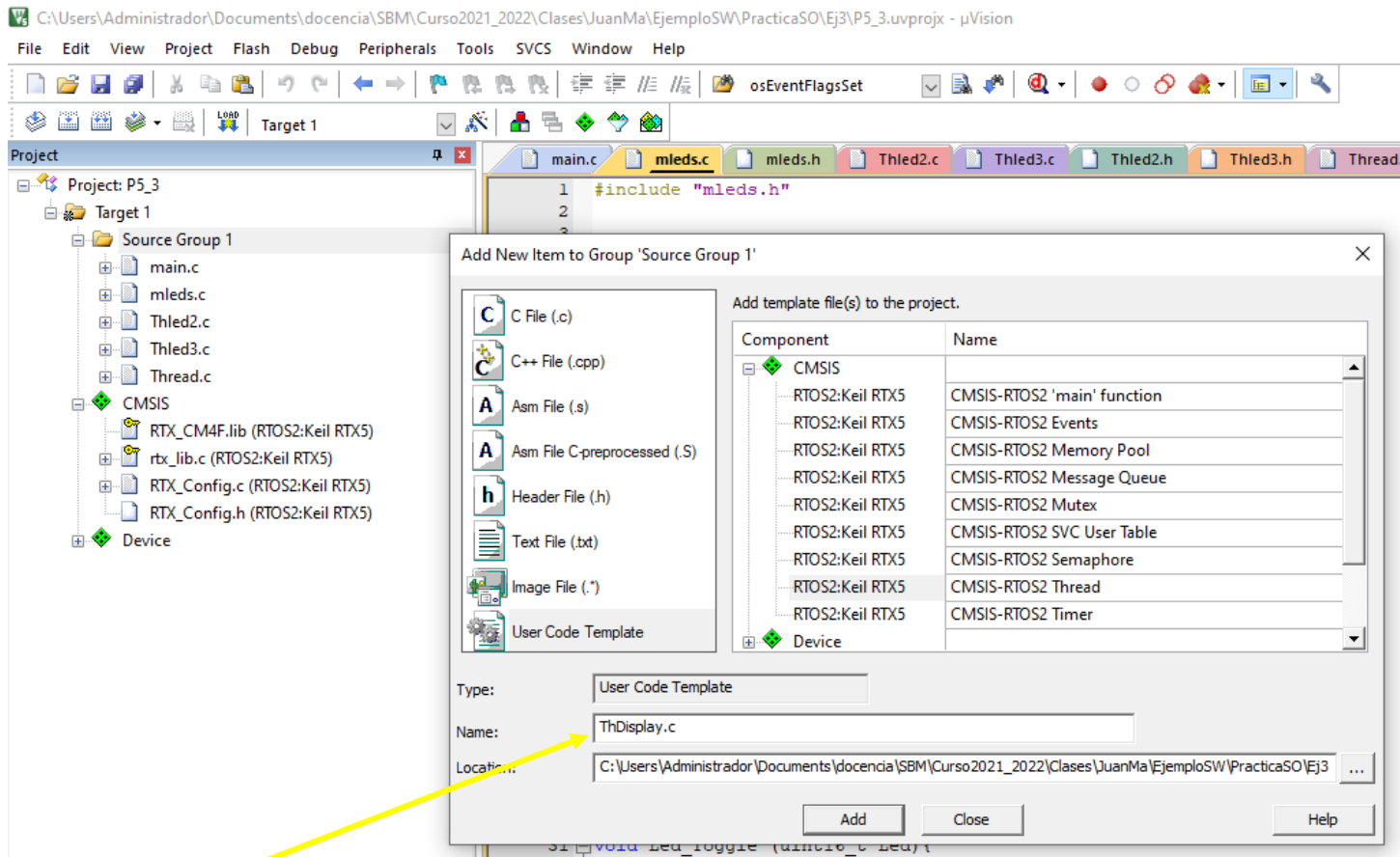
CMSIS-RTOS (Threads)

- El planificador del SO es el encargado de gestionar la ejecución de un thread
- Estados de un *thread*:
 - RUNNING**: *thread* actualmente en ejecución
 - READY**: *threads* preparados para ser ejecutados. Una vez que la tarea que se está ejecutando ha consumido su *timeslice*, la siguiente tarea con máxima prioridad pasa a RUNNING
 - WAITING**: *threads* esperando la ocurrencia de algún evento
 - INACTIVE**: *threads* NO creados o terminados. No consumen ningún recurso



Creación de Threads en Keil uVision

- Se facilita su utilización mediante la **utilización de *templates***.



- Asignar un nombre adecuado a la funcionalidad de la tarea a realizar.**

Threads en Keil uVision

CMSIS-RTOS2 Thread

```
#include "cmsis_os2.h"           // CMSIS RTOS header file

/*-----
 *   Thread 1 'Thread_Name': Sample thread
 *-----*/

osThreadId_t tid_Thread;        // thread id

void Thread (void *argument);    // thread function

int Init_Thread (void) {

    tid_Thread = osThreadNew(Thread, NULL, NULL);
    if (tid_Thread == NULL) {
        return(-1);
    }

    return(0);
}

void Thread (void *argument) {

    while (1) {
        ; // Insert thread code here...
        .....
    }

    // suspend thread
}
```

Identificador del thread. Ej -> tid_ThDisplay
Lo usaremos para referenciar al thread desde otras partes del código

Creación del thread. El primer parámetro es la función con el código del thread. Ej-> ThDisplay.

Función del thread. Ej -> ThDisplay
Típicamente un bucle sin fin con llamadas a funciones del SO o propias

- Es altamente aconsejable crear un fichero de cabecera donde se deberán declarar la función de inicio para poder ser utilizados por otros módulos del software

Threads en Keil uVision

Fichero ThDisplay.c

```
#include "cmsis_os2.h"           // CMSIS RTOS header file
#include "ThDisplay.h"

osThreadId_t tid_ThDisplay;      // thread id

void Thled1 (void *argument);     // thread function

int Init_ThDisplay (void) {

    tid_ThDisplay = osThreadNew(ThDisplay, NULL, NULL);
    if (tid_ThDisplay == NULL) {
        return(-1);
    }

    return(0);
}

void ThDisplay (void *argument) {

    int ciclo=0;

    Init_display();

    while (1) {

        .....

    }
}
```

Fichero de cabeceras ThDisplay.h

```
#include "stm32f4xx_hal.h"
#include "definiciones.h"

#ifndef _THDISPALY_H
#define _THDISPLAY_H

#define S_PINTA 0x00000001U

int Init_ThDisplay (void);

#endif
```

Fichero main Lanza la ejecución de los *threads*

```
int main(void)
{
    int status=0;

    HAL_Init();

    /* Configure the system clock to 168 MHz */
    SystemClock_Config();
    SystemCoreClockUpdate();

    /* Add your application code here
     */

#ifdef RTE_CMSIS_RTOS2
    /* Initialize CMSIS-RTOS2 */
    osKernelInitialize ();

    /* Create thread functions that start executing,
     Example: osThreadNew(app_main, NULL, NULL); */

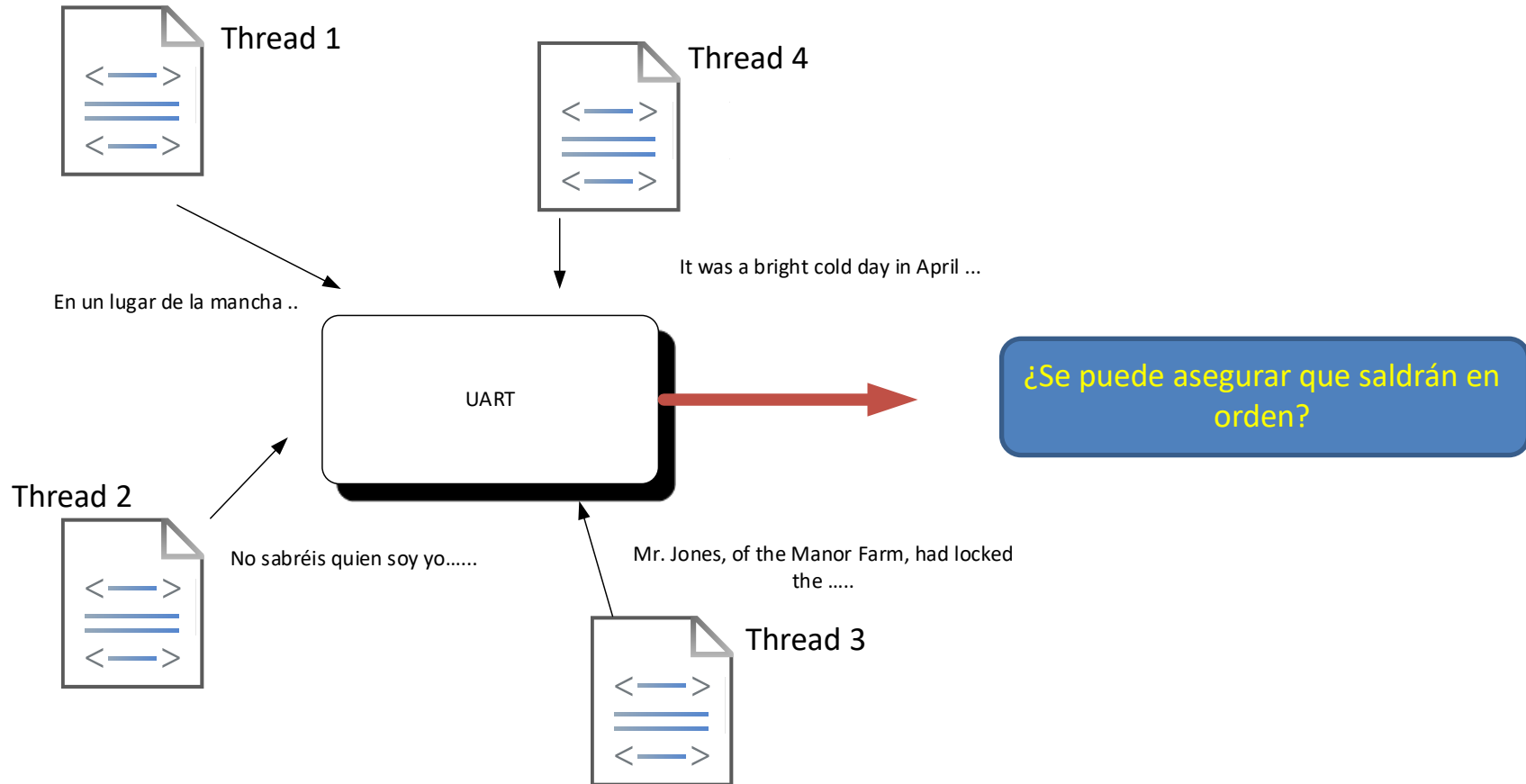
    status |= Init_ThDisplay ();

    /* Start thread execution */
    osKernelStart();
#endif

    /* Infinite loop */
    while (1)
    {
    }
}
```

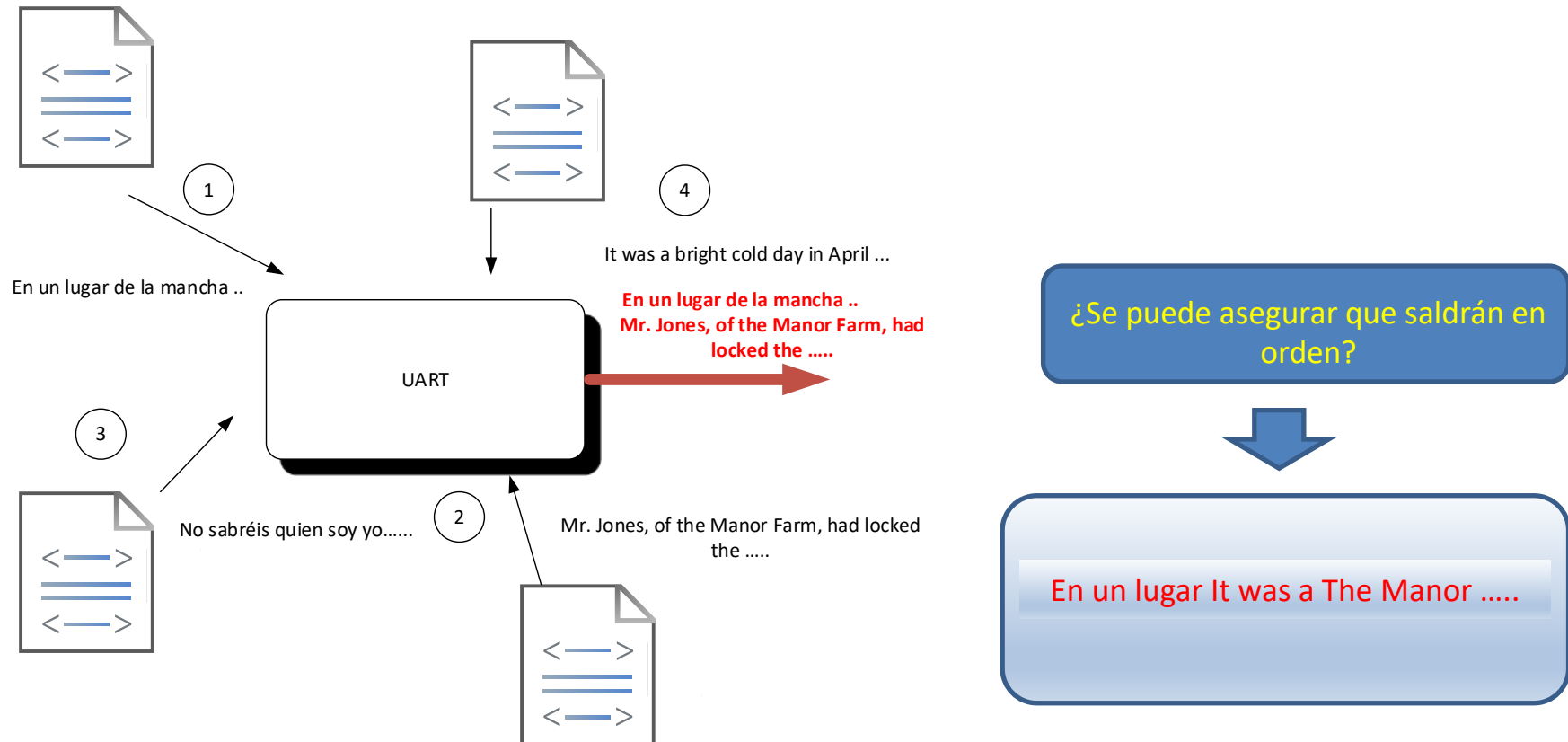
CMSIS-RTOS (Sincronización)

- Una de las claves a la hora de utilizar un sistema operativo es la sincronización entre los diferentes elementos que componen una aplicación.



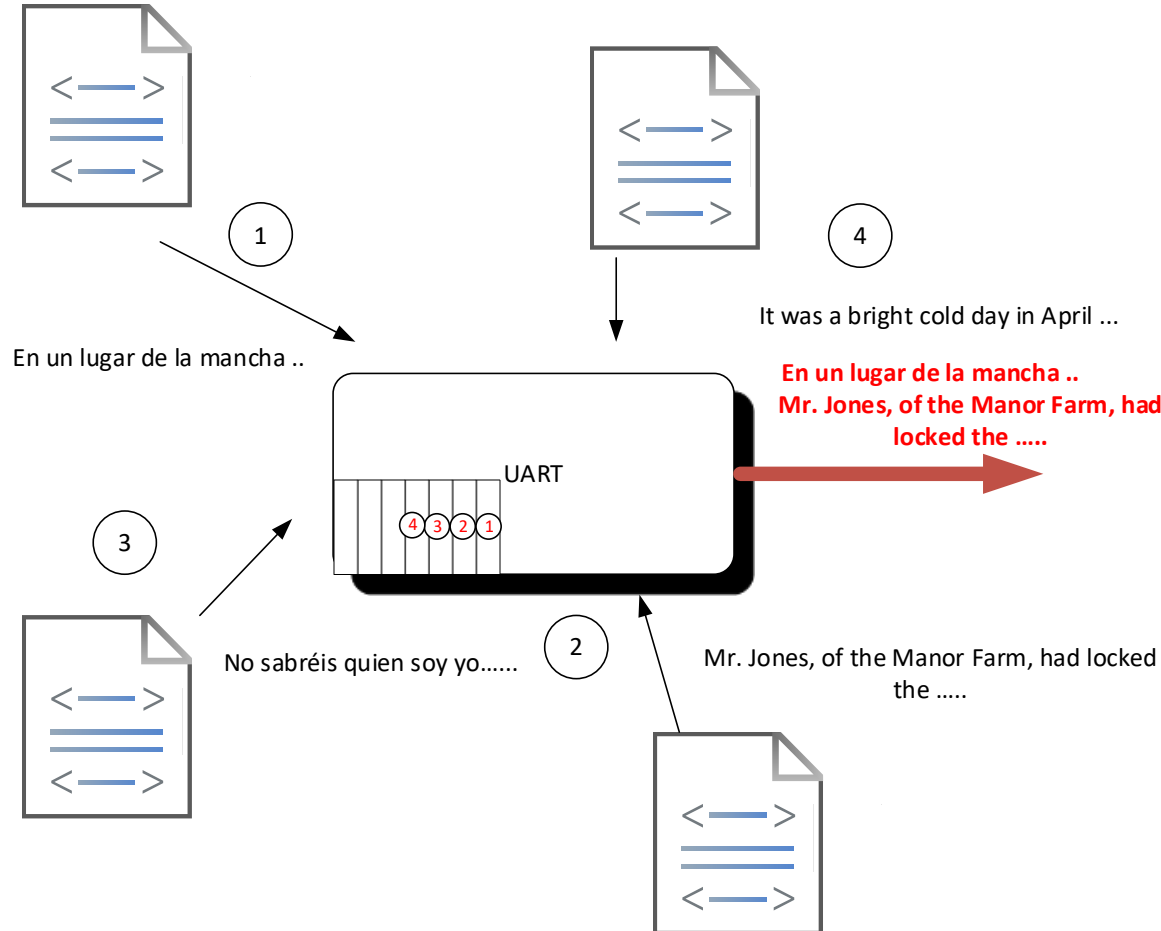
CMSIS-RTOS (Sincronización)

- Una de las claves a la hora de utilizar un sistema operativo es la sincronización entre los diferentes elementos que componen una aplicación.



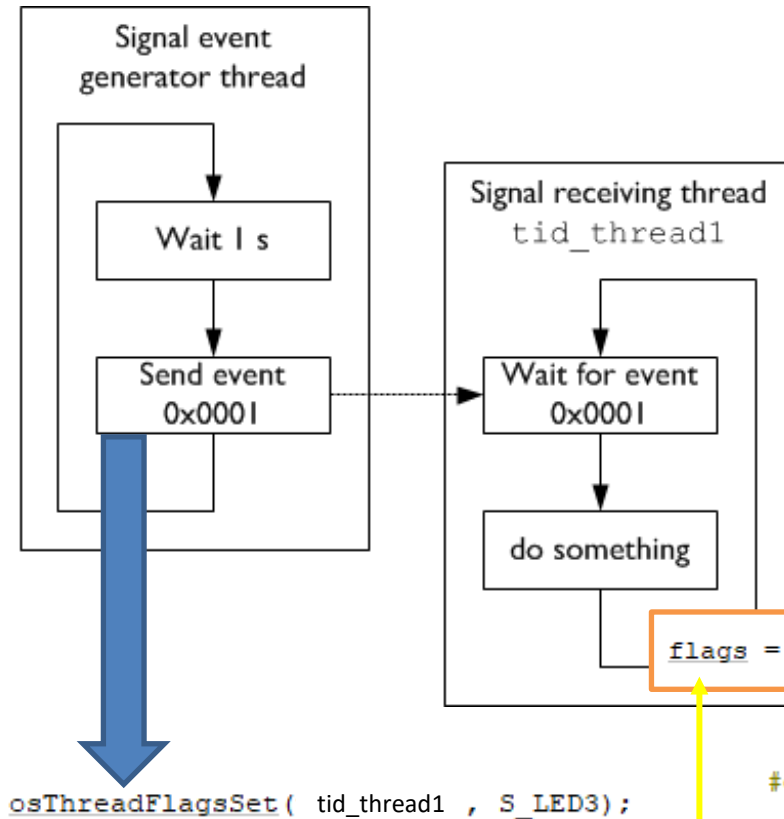
CMSIS-RTOS (Sincronización)

- RTX tiene múltiples mecanismos de sincronización: Flags, Colas, semáforos, Mutex, etc.



Ahora se asegura que no salen mezclados, pero no se puede decir que salgan en el orden deseado. Hay que utilizar/complementar con otros métodos de sincronización.

CMSIS-RTOS (Sincronización): Thread Flags



- ✓ Los hilos pueden detener su ejecución esperando a la activación de un flag desde otro hilo. El hilo pasa al estado de waiting.
- ✓ Cuando el flag/conjunto de flags son activados, el hilo pasa al estado de ready.

```
flags = osThreadFlagsWait(S_LED3, osFlagsWaitAny, osWaitForever);
```

```
#define S_LED3 0x00000002U
```

Temporización en ticks o infinito

```
// Flags options (\ref osThreadFlagsWait and \ref osEventFlagsWait).  
#define osFlagsWaitAny      0x00000000U ///< Wait for any flag (default).  
#define osFlagsWaitAll      0x00000001U ///< Wait for all flags.  
#define osFlagsNoClear      0x00000002U ///< Do not clear flags which have been specified to wait for.
```

Retorna los flags antes de ser borrados

NO se puede utilizar en las ISR

CMSIS-RTOS (Thread Flags)

- **uint32_t osThreadFlagSet(osThreadId_t thread_id, uint32_t flags)**
 - Activa el flag especificado de una tarea activa
 - Se puede llamar desde una interrupción
- **uint32_t osThreadFlagClear(osThreadId_t thread_id, uint32_t signals)**
 - Borra el flag especificado de una tarea activa
 - No se puede llamar desde una interrupción
- **uint32_t osThreadFlagsWait(uint32_t flags, uint32_t options, uint32_t timeout)**
 - Espera a uno o mas Flags para continuar la ejecución
 - No se puede llamar desde una interrupción
- **uint32_t osThreadFlagGet(void)**
 - Retorna los flags del thread en ejecución
 - No se puede llamar desde una interrupción

Documentación y valores de retorno

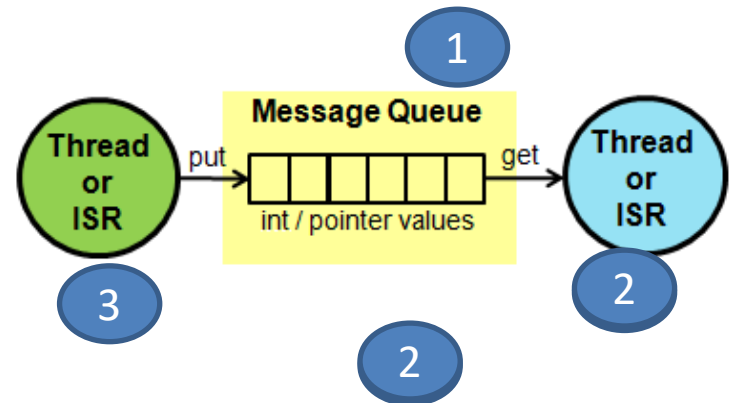
https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__ThreadFlagsMgmt.html

CMSIS-RTOS (Sincronización)

Colas de mensajes

- ✓ Mecanismo para intercambio de datos “mensajes” entre hilos.
- ✓ Paradigma productor/consumidor.
- ✓ El consumidor puede extraer datos de la cola de forma temporizada.

```
osMessageQueuePut(mid_MsgQueue, &msg, 0U, 0U);
```



```
status = osMessageQueueGet(mid_MsgQueue, &msg, NULL, osWaitForever); // wait for message
```

Temporización en ticks

Puntero donde se recoge el mensaje

```
typedef struct {  
    uint8_t Buf[32];  
    uint8_t Idx;  
} MSGQUEUE_OBJ_t;
```

```
MSGQUEUE_OBJ_t msg;
```

```
osMessageQueueId_t mid_MsgQueue; // message queue id  
  
int Init_MsgQueue (void) {  
    mid_MsgQueue = osMessageQueueNew(MSGQUEUE_OBJECTS, sizeof(MSGQUEUE_OBJ_t), NULL);  
    if (mid_MsgQueue == NULL) {  
        return(0);  
    }  
}
```

Se pueden utilizar en las ISR



CMSIS-RTOS (Message Queue)

osMessageQueueId_t **osMessageQueueNew** (uint32_t msg_count, uint32_t msg_size, const **osMessageQueueAttr_t** *attr)
Create and Initialize a Message Queue object. [More...](#)

const char * **osMessageQueueGetName** (**osMessageQueueId_t** mq_id)
Get name of a Message Queue object. [More...](#)

osStatus_t **osMessageQueuePut** (**osMessageQueueId_t** mq_id, const void *msg_ptr, uint8_t msg_prio, uint32_t timeout)
Put a Message into a Queue or timeout if Queue is full. [More...](#)

osStatus_t **osMessageQueueGet** (**osMessageQueueId_t** mq_id, void *msg_ptr, uint8_t *msg_prio, uint32_t timeout)
Get a Message from a Queue or timeout if Queue is empty. [More...](#)

uint32_t **osMessageQueueGetCapacity** (**osMessageQueueId_t** mq_id)
Get maximum number of messages in a Message Queue. [More...](#)

uint32_t **osMessageQueueGetMsgSize** (**osMessageQueueId_t** mq_id)
Get maximum message size in a Message Queue. [More...](#)

uint32_t **osMessageQueueGetCount** (**osMessageQueueId_t** mq_id)
Get number of queued messages in a Message Queue. [More...](#)

uint32_t **osMessageQueueGetSpace** (**osMessageQueueId_t** mq_id)
Get number of available slots for messages in a Message Queue. [More...](#)

osStatus_t **osMessageQueueReset** (**osMessageQueueId_t** mq_id)
Reset a Message Queue to initial empty state. [More...](#)

osStatus_t **osMessageQueueDelete** (**osMessageQueueId_t** mq_id)
Delete a Message Queue object. [More...](#)

Documentación y valores de retorno

https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__Message.html#details

CMSIS-RTOS (Retardos y acciones temporizadas)

- **Funciones genéricas de espera**

- Se utilizan para hacer esperas por una determinada cantidad de tiempo

```
osStatus_t status; // capture the return status
uint32_t delayTime; // delay time in milliseconds

delayTime = 1000U; // delay 1 second
status = osDelay(delayTime); // suspend thread execution
```

HAL_Delay
Esta prohibido!!!!

Expresado en ticks del Sistema. Por defecto el tick es de 1ms

```
uint32_t tick;

tick = osKernelGetTickCount(); // retrieve the number of system ticks
for (;;) {
    tick += 1000U; // delay 1000 ticks periodically
    osDelayUntil(tick);
    // ...
}
```

NO se pueden utilizar en las ISR

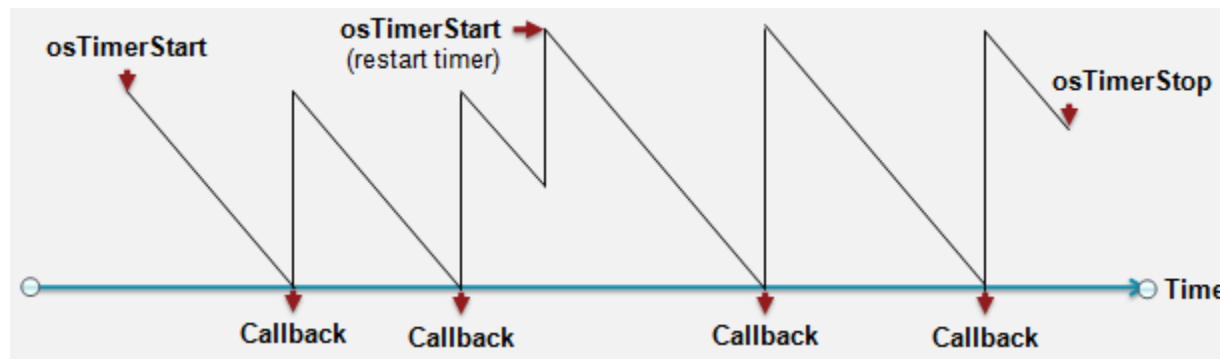
- **Timers virtuales**

- Ejecutan una función (**callback**) cuando el tiempo expira
- La función de callback es definida por el usuario

CMSIS-RTOS (*Timers Virtuales*)

- Los *timers* funcionan en dos modos
 - one-shot**: una única ejecución (se arrancan y cuando pasa el tiempo se ejecuta la callback)
 - periodic**: se repite hasta que se para o se elimina (se arrancan y la callback se ejecuta periódicamente).

Todos los *timers* se pueden arrancar, parar o reanudar.



Source: https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__TimerMgmt.html#details

CMSIS-RTOS (*Timers Virtuales*)

```
/*----- Periodic Timer Parpadeo LED -----*/
osTimerId_t tim_id; // timer id
static uint32_t exec;

static void TimerLed_Callback (void const *arg) {
    //Añadir código aquí
    Led_Toggle(LED1);
}

int Init_Timers (void) {
    osStatus_t status;

    Init_led(LED1);

    // Se crea timer
    exec = 2U;
    tim_id = osTimerNew((osTimerFunc_t)&TimerLed_Callback, osTimerPeriodic, &exec, NULL);
    if (tim_id != NULL) { // Se ha creado
        // Se inicializa con un intervalo de 1s
        status = osTimerStart(tim_id, 1000U);
        if (status != osOK) {
            return -1;
        }
    }
    return NULL;
}
```

Identificador del *timer*
Lo usaremos para referenciar al *timer*
exec posibilidad de pasar un parámetro a la *callback*

Función *Callback* a ejecutar

Tipo de *timer*:
osTimerPeriodic
osTimerOnce

Asignar la *callback* al *timer*

CMSIS-RTOS (Virtual Timers)

osTimerId_t osTimerNew (osTimerFunc_t func, osTimerType_t type, void *argument, const osTimerAttr_t *attr)
Create and Initialize a timer. More...

const char * **osTimerGetName** (osTimerId_t timer_id)
Get name of a timer. More...

osStatus_t osTimerStart (osTimerId_t timer_id, uint32_t ticks)
Start or restart a timer. More...

Si el timer esta arrancado y se llama de nuevo a la function se rearranca!!!!

osStatus_t osTimerStop (osTimerId_t timer_id)
Stop a timer. More...

uint32_t **osTimerIsRunning** (osTimerId_t timer_id)
Check if a timer is running. More...

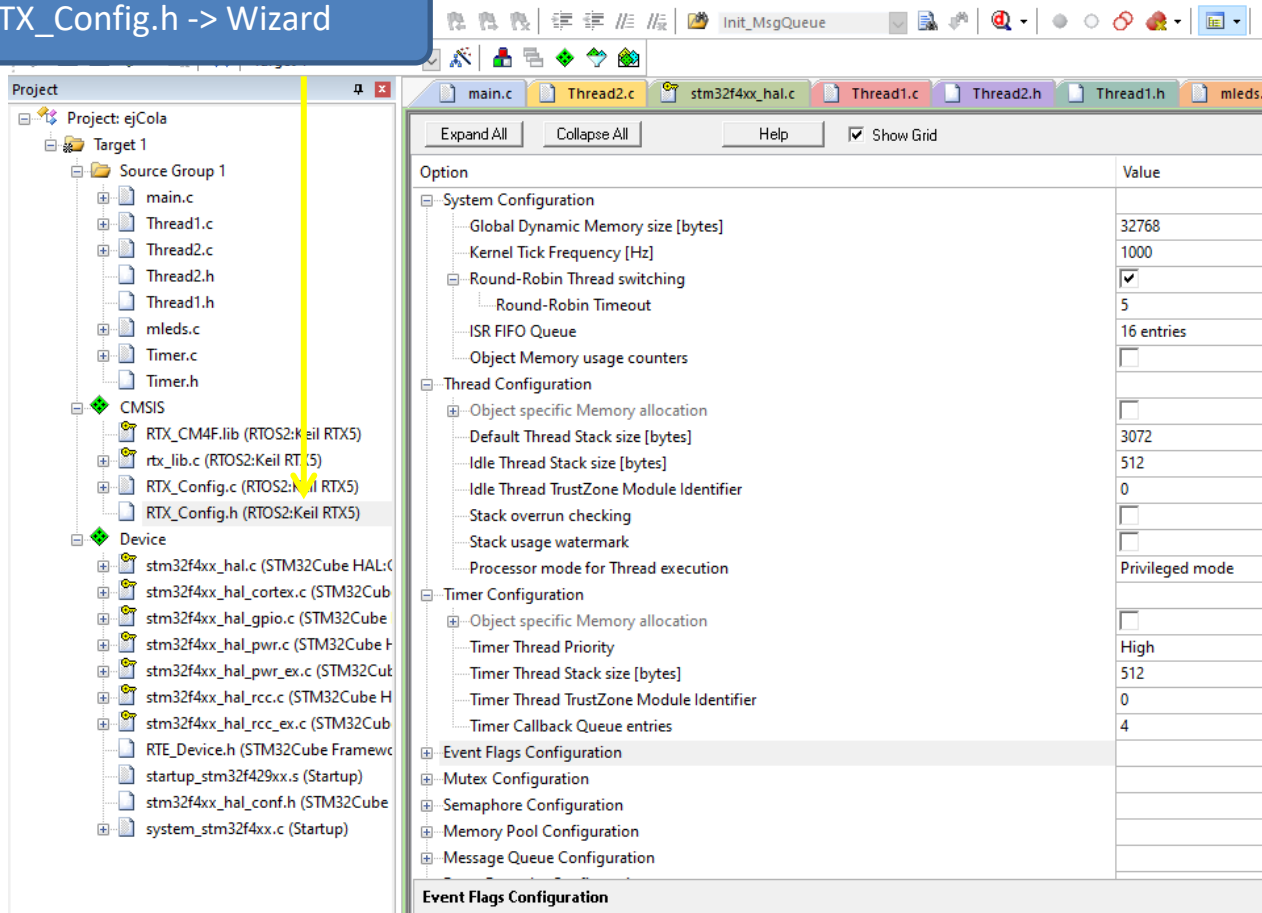
osStatus_t osTimerDelete (osTimerId_t timer_id)
Delete a timer. More...

Documentación y valores de retorno

https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__TimerMgmt.html

CMSIS-RTOS (Herramientas de configuración)

RTX_Config.h -> Wizard



Permite configurar diversos parámetros del S.O.

Documentación

<https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html>



POLITÉCNICA



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE SISTEMAS Y TELECOMUNICACIÓN



Sistemas Basados en Microprocesador

CMSIS-RTOS (Herramientas de depuración)

RTX RTOS	
Property	Value
System	
Threads	
id: 0x200081E8 "osRtxIdleThread"	osThreadRunning, osPriorityIdle, Stack Used: 0%
id: 0x2000822C "osRtxTimerThread"	osThreadBlocked, osPriorityHigh, Stack Used: 18%
id: 0x20000170 "Thread2"	osThreadBlocked, osPriorityNormal, Stack Used: 4%
State	osThreadBlocked
Priority	osPriorityNormal
Attributes	osThreadDetached
Waiting	Message Get, Timeout: osWaitForever
id: 0x20001A20	
Stack	Used: 4% [128]
Flags	0x00000000
id: 0x20000DC8 "Thread1"	osThreadBlocked, osPriorityNormal, Stack Used: 3%
State	osThreadBlocked
Priority	osPriorityNormal
Attributes	osThreadDetached
Waiting	Delay, Timeout: 5272
Stack	Used: 3% [104]
Flags	0x00000000
Timers	
id: 0x20001D68	Running, Tick: 272
State	Running
Type	osTimerPeriodic
Tick	272
Load	1000
Callback	Func: TimerLed_Callback, Arg: 0x20000114
Message Queues	
id: 0x20008164	Messages: 0, Max: 4
id: 0x20001A20	Messages: 0, Max: 16
Messages	0
Max Messages	16
Message size	33
Threads waiting (1)	
id: 0x20000170	Timeout: osWaitForever

View->Watch Windows -> RTX RTOS

Muestra la información del Sistema Operativo: *Threads, timers, Queues*



POLITÉCNICA



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE SISTEMAS Y TELECOMUNICACIÓN



Sistemas Basados en Microprocesador

CMSIS-RTOS (Consideraciones)

- Antes de abordar una aplicación, se diseñará una estructura de la misma, de forma que en la medida de lo posible **cada periférico será gestionado por un módulo software** que estará compuesto por el correspondiente *thread*, señales, colas, y estructuras de datos. Concepto de “*driver software*”.
- Las ISRs serán manejadas por la HAL y en las respectivas *Callbacks* se enviarán a los distintos *threads* las señales, mensajes necesarios. **Serán rutinas cortas y se evitarán bucles**. Vigilar que elementos del S.O. se pueden implementar en su interior.
- Prestar especial **atención a la sincronización** entre *threads*.
- **Se reducirá al máximo** la utilización de **variables globales**.