

Sistemas Embebidos



Microkernels

Ángel Perles

r09

Contenido

- Objetivos
- Introducción
- FreeRTOS
 - Qué es
 - Tasks
 - Fundamentals
 - Example STM32
 - Using Cube
 - CMSIS-RTOS
 - Example
 - Mutual exclusion
 - Semaphores



Objetivos

- Introducir el concepto de microkernel
- Aplicar el microkernel FreeRTOS



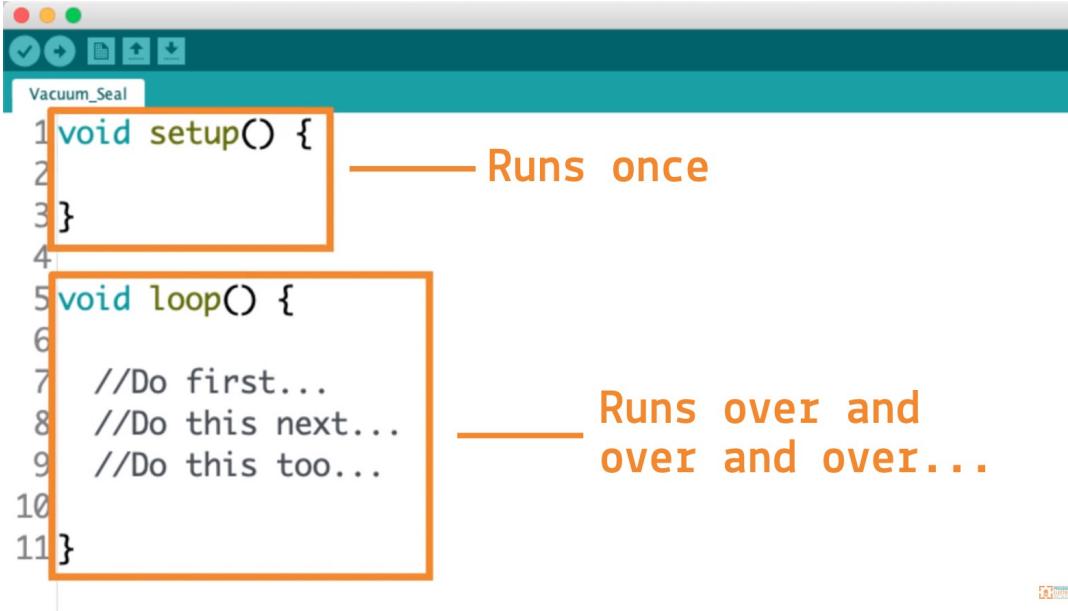
Introducción

- Tendencia hacia aplicaciones más complejas
 - Agravada por la necesidad de conectividad (Bluetooth, WiFi, USB, IoT ...)
 - Cumplimiento de plazos de ejecución (real-time)
 - Complejidad interfaces de usuario (gráficos, gestos, voz , ...)
- El mercado está lleno de aparatitos cada vez más inteligentes
- Quizá valga la pena hacer un flash-back



Introducción

- La manera básica de hacer aplicaciones es insuficiente



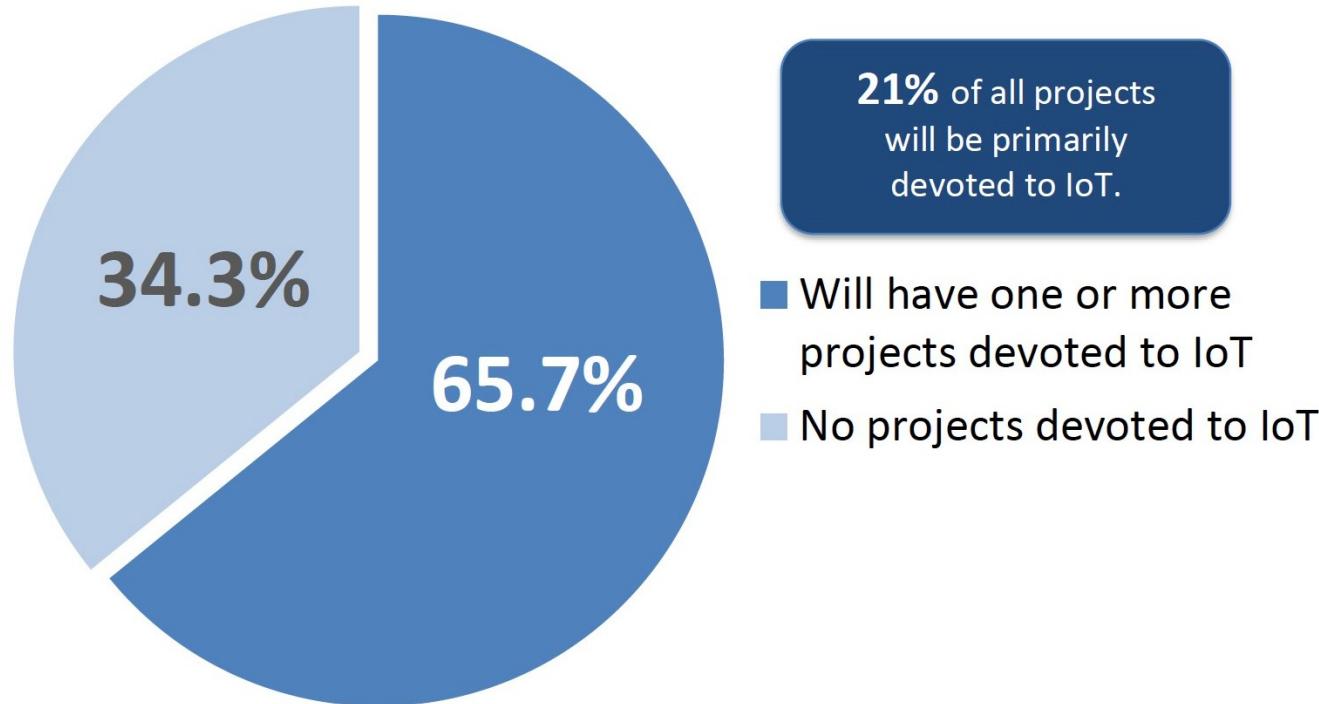
```
Vacuum_Seal
1 void setup() {
2
3 }
4
5 void loop() {
6
7 //Do first...
8 //Do this next...
9 //Do this too...
10
11 }
```

- Las interrupciones y lo que vimos de “a dos niveles” ayuda ...
- ... pero no es suficiente



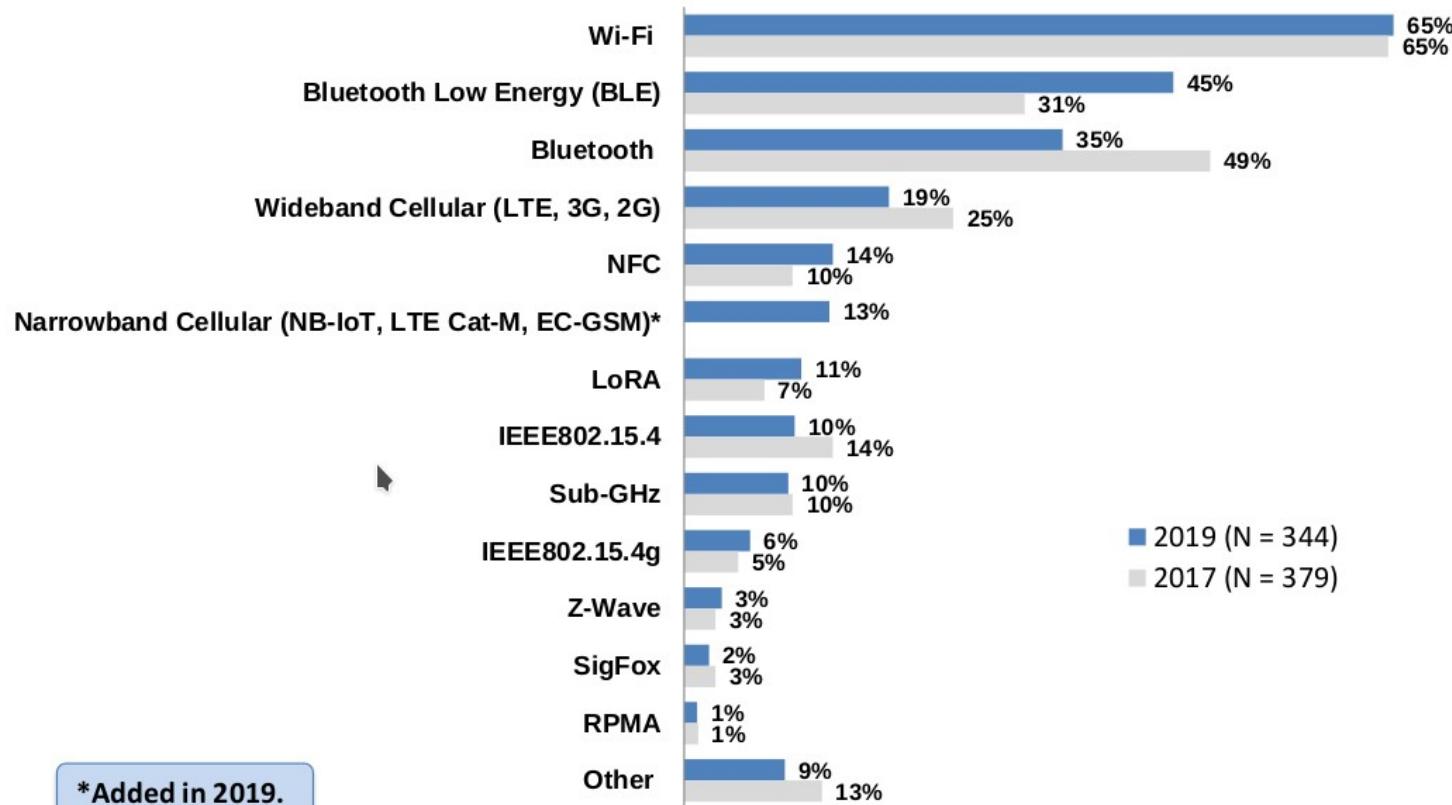
Introducción

- ¿Qué nos dice el mundo real?
- <https://www.embedded.com/2019-embedded-markets-study-reflects-emerging-technologies-continued-c-c-dominance/>



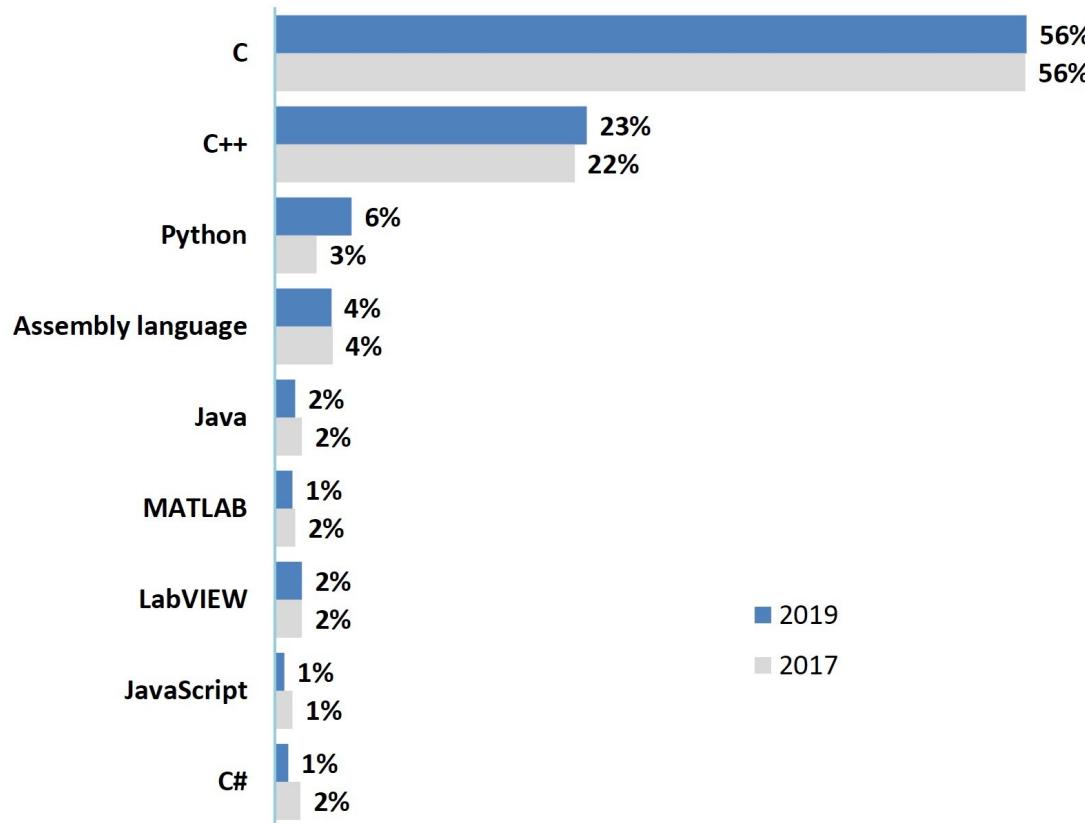
Introducción

- ¿Qué nos dice el mundo real?
- <https://www.embedded.com/2019-embedded-markets-study-reflects-emerging-technologies-continued-c-c-dominance/>



Introducción

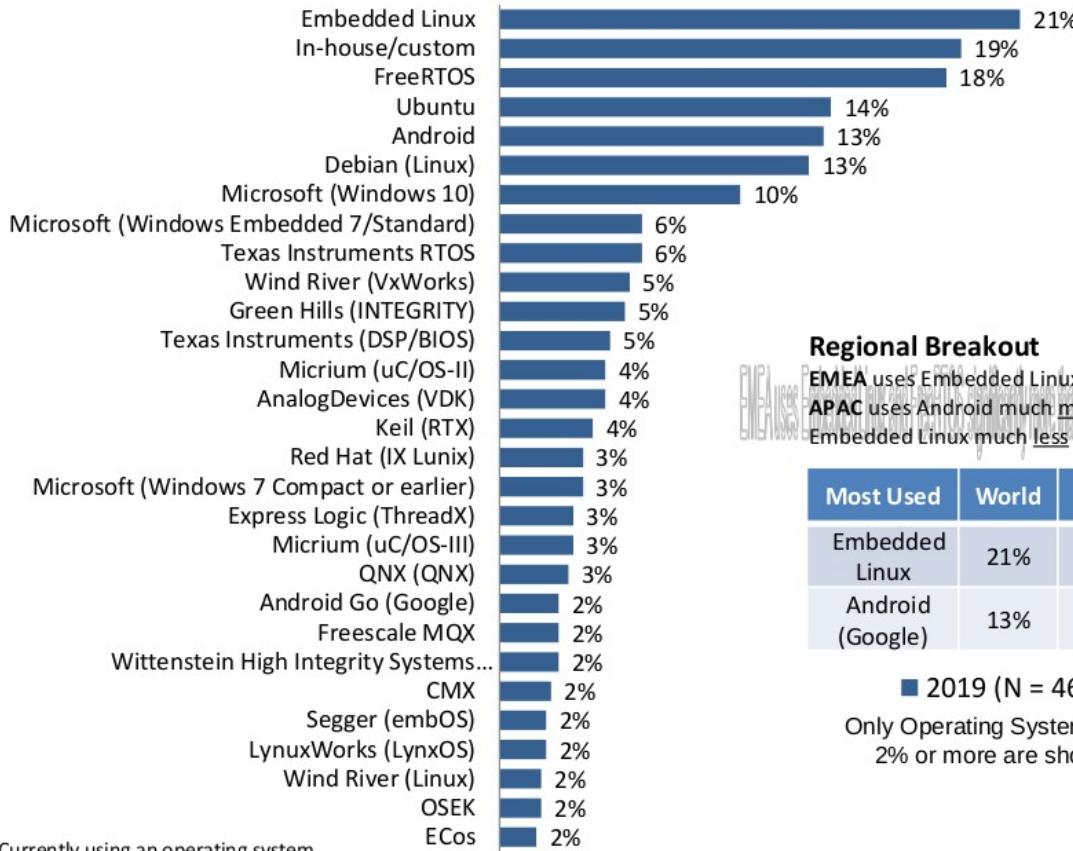
- ¿Qué nos dice el mundo real?
- <https://www.embedded.com/2019-embedded-markets-study-reflects-emerging-technologies-continued-c-c-dominance/>



Introducción

- ¿Qué nos dice el mundo real?

Please select ALL of the operating systems you are currently using.



Regional Breakout

EMEA uses Embedded Linux much more than other regions.
 APAC uses Android much more than other regions and uses Embedded Linux much less than others.

Most Used	World	Americas	EMEA	APAC
Embedded Linux	21%	21%	30%	15%
Android (Google)	13%	9%	14%	27%

■ 2019 (N = 468)

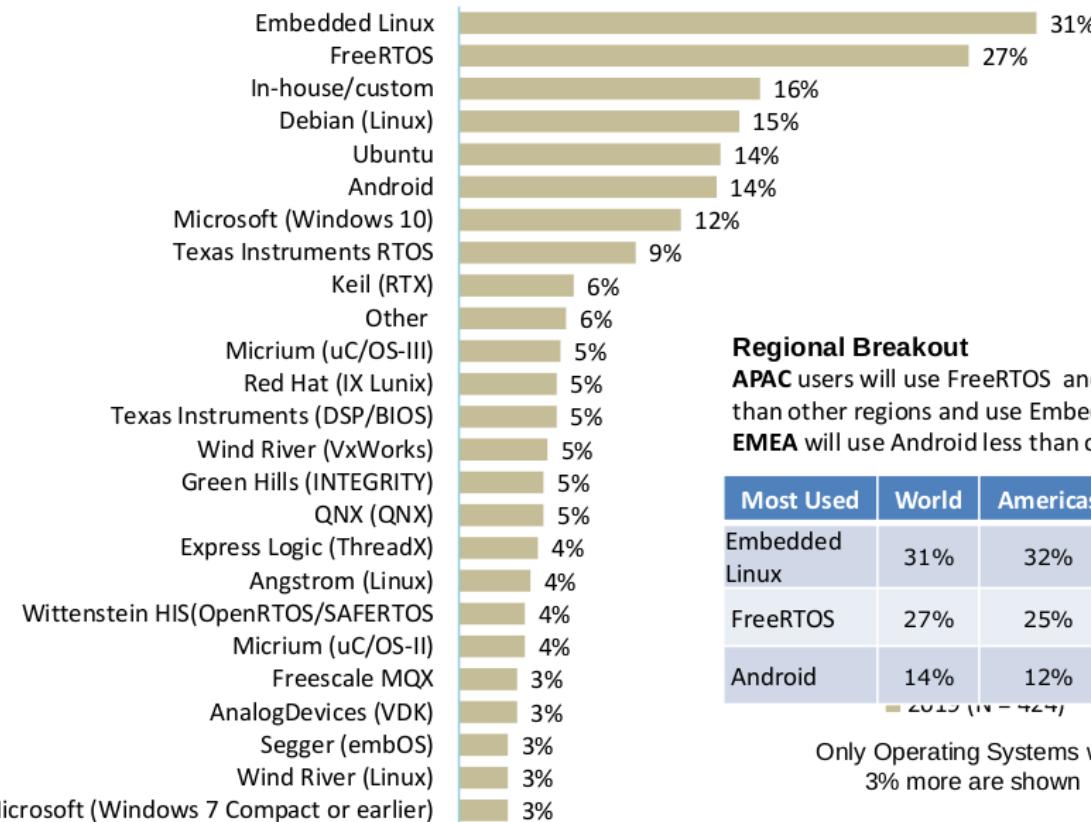
Only Operating Systems with 2% or more are shown.



Introducción

- ¿Qué nos dice el mundo real?

Please select ALL of the operating systems you are considering using in the next 12 months.



Regional Breakout

APAC users will use FreeRTOS and Android much more than other regions and use Embedded Linux much less. EMEA will use Android less than other regions.

Most Used	World	Americas	EMEA	APAC
Embedded Linux	31%	32%	31%	26%
FreeRTOS	27%	25%	24%	37%
Android	14%	12%	10%	26%

2015 (n = 424)

Only Operating Systems with
3% more are shown



Introducción

- Un “microkernel” es una buena opción
 - Se lo puede montar uno
 - O emplear un producto comercial/libre disponible <- recomendado
- “No es más” que más que código para proporcionar los servicios más básicos de un Sistema Operativo moderno
 - Multitarea, real-time (RTOS real-time operating system)
 - Buffering, colas, pilas, mailboxes, ...
 - Semáforos, zonas críticas, ...
- Por ejemplo un paradigma “productor-consumidor”

Que quiere la gente

- Availability of full source code (39%)
- No royalties (30%)
- Compatibility with other software and systems (27%)
- Availability of tech support (27%)



Introducción

- En “Wikipedia” hay una buena lectura con la idea de microkernel
 - <https://en.wikipedia.org/wiki/Microkernel>
 - Más hacia alto nivel tipo Unix
- Microkernels para microcontroladores hay muchos
- Ejemplos
 - Libre: FreeRTOS, ChibiOS, CoOS, mbedOS
 - Comerciales: Micrium uC/OS, Keil RTX
 - Especificaciones: ARM CMSIS RTOS
 - Los hay “seguros”, los hay “certificados” ... incluso demostrados matemáticamente (L4 microkernel) ... un mundo.
 - Piensa en aviática, satélites, tarjetas de crédito, SIM del móvil, ...



FreeRTOS: qué es

- FreeRTOS parece acertado para empezar
 - Múltiples micros
 - Echad un vistazo a la web <https://www.freertos.org/index.html>



- ¿Pega
 - Para promocionar que los dispositivos IoT usen su AWS
 - De hecho, se puede hacer un Alexa ahora con esto
- 2020: Lo cierto es que le ha dado un empujón brutal

FreeRTOS: tasks - fundamentals

- En la web explica los fundamentos de los RTOS genérico
- Leed cada uno de los apartados de “Fundamentals”
- Y vais preguntando dudas

 Quality RTOS & Embedded Software
[Download FreeRTOS](#)

[Kernel](#) [Libraries](#) [Resources](#) [Community](#) [Support](#) [Email List](#) 

[Home](#)
[Getting Started](#)
[Kernel Overview](#)
+ [Supported Devices](#)
+ [Demos](#)
+ [How FreeRTOS Works](#)
 └─ [Introduction](#)
 └─ [What is An RTOS?](#)
+ [Fundamentals](#)
 └─ [Multitasking Basics](#)
 └─ [Scheduling Basics](#)
 └─ [Context Switching](#)
 └─ [Real Time Applications](#)
 └─ [Real Time Scheduling](#)
+ [Implementation](#)
 └─ [Source Code Organization](#)
 └─ [Creating a New Project](#)
+ [Features](#)
+ [API Reference](#)
+ [Quality Management](#)
 └─ [Licensing](#)

Multitasking

[\[RTOS Fundamentals\]](#)

The **kernel** is the core component within an operating system. Operating systems such as Linux employ kernels that allow users access to the computer seemingly simultaneously. Multiple users can execute multiple programs apparently concurrently.

Each executing program is a **task** (or thread) under control of the operating system. If an operating system can execute multiple tasks in this manner it is said to be **multitasking**.

The use of a multitasking operating system can simplify the design of what would otherwise be a complex software application:

- The multitasking and inter-task communications features of the operating system allow the complex application to be partitioned into a set of smaller and more manageable tasks.
- The partitioning can result in easier software testing, work breakdown within teams, and code reuse.



FreeRTOS: tasks - fundamentals

- An application based on RTOS is structured in independent “tasks”
 - Each task has its own context
 - The RTOS is in charge of assigning CPU following a given scheduling policy

Task Summary



Simple.



No restrictions on use.



Supports full preemption.



Fully prioritised.



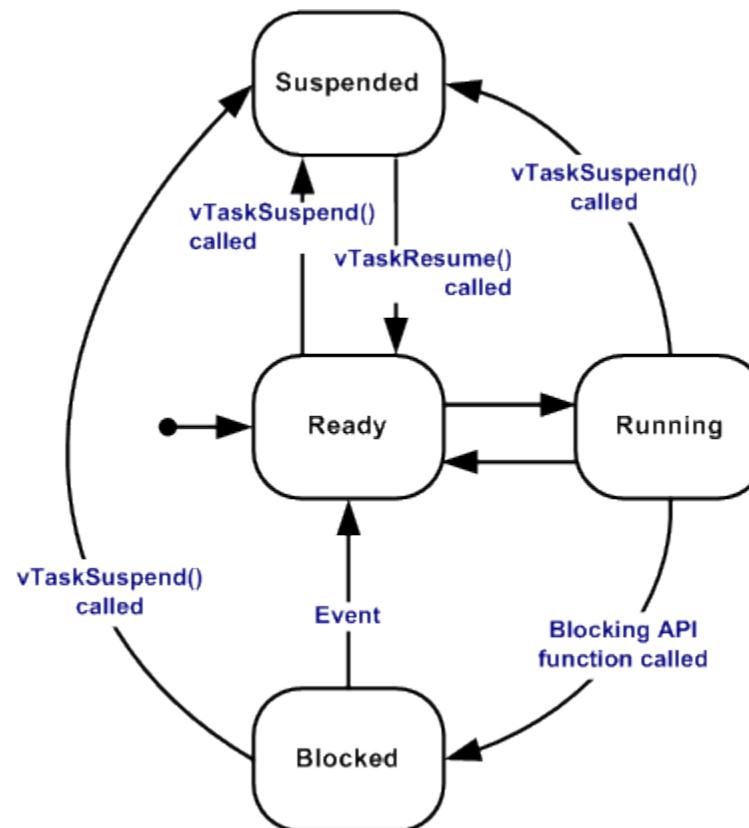
Each task maintains its own stack resulting in higher RAM usage.



Re-entrancy must be carefully considered if using preemption.

FreeRTOS: tasks - fundamentals

- A task can be in one of the following states:
 - Ready, suspended, running,blocked



FreeRTOS: tasks - fundamentals

- A task has this appearance
 - It is a simple infinite loop!
 - (But it wastes a lot of CPU doing nothing?)

```
void vATaskFunction( void *pvParameters )
{
    for( ;; )
    {
        -- Task application code here. --
    }

    /* Tasks must not attempt to return from their implementing
    function or otherwise exit. In newer FreeRTOS port
    attempting to do so will result in an configASSERT() being
    called if it is defined. If it is necessary for a task to
    exit then have the task call vTaskDelete( NULL ) to ensure
    its exit is clean. */
    vTaskDelete( NULL );
}
```



FreeRTOS: tasks - fundamentals

- A task has this appearance
 - Well, better we wait for something to not waste CPU

```

void vTaskFunction( void *pvParameters )
{
    for( ;; )
    {
        /* Psudeo code showing a task waiting for an event
         * with a block time. If the event occurs, process it.
         * If the timeout expires before the event occurs, then
         * the system may be in an error state, so handle the
         * error. Here the pseudo code "WaitForEvent()" could
         * replaced with xQueueReceive(), ulTaskNotifyTake(),
         * xEventGroupWaitBits(), or any of the other FreeRTOS
         * communication and synchronisation primitives. */
        if( WaitForEvent( EventObject, TimeOut ) == pdPASS )
        {
            -- Handle event here. --
        }
        else
        {
            -- Clear errors, or take actions here. --
        }
    }

    /* As per the first code listing above. */
    vTaskDelete( NULL );
}
  
```



FreeRTOS: tasks - fundamentals

- Tasks have “priorities” ...
 - To allow the scheduler to decide which to pick up and run
 - Each task is assigned a priority from 0 to (configMAX_PRIORITIES - 1), where configMAX_PRIORITIES is defined within FreeRTOSConfig.h.
 - Time sliced round-robin scheduling (Buscad qué significa eso)



FreeRTOS: tasks - example

- Ejemplo básico (Para STM32 “antiguamente”)

```

// LED 1
xTaskHandle hLED1Task; // referencia de la tarea

portTASK_FUNCTION( vLED1Task, pvParameters ) { // la tarea
    while(1) {
        STM_EVAL_LEDToggle(LED1);
        vTaskDelay(200 / portTICK_RATE_MS); // dormir 200 ms
    }
}

int main( void ) {
    // ...

    xTaskCreate( vLED1Task, (signed char *) "LED1",
                configMINIMAL_STACK_SIZE,
                NULL, mainLED_TASK_PRIORITY, &hLED1Task );

    vTaskStartScheduler(); // Iniciar el planificador
}
  
```



FreeRTOS: cube

- Ejemplos en el paquete de las Cube
 - Eso quiere decir que ST también lo considera estratégico

exploits many features to configure a microSD drive.		
FreeRTOS	FreeRTOS_LowPower	How to enter and exit low-power mode with CMSIS RTOS API.
	FreeRTOS_LowPower_LPTIM	How to enter and exit low-power mode with CMSIS RTOS API with LPTIM used as clock source for both RTOS and HAL ticks.
	FreeRTOS MPU	How to use the MPU feature of FreeRTOS.
	FreeRTOS_Mail	How to use mail queues with CMSIS RTOS API.
	FreeRTOS_Mutexes	How to use mutexes with CMSIS RTOS API.
	FreeRTOS_Queues	How to use message queues with CMSIS RTOS API.
	FreeRTOS_Semaphore	How to use semaphores with CMSIS RTOS API.
	FreeRTOS_SemaphoreFromISR	How to use semaphore from ISR with CMSIS RTOS API.
	FreeRTOS_Signal	How to perform thread signaling using CMSIS RTOS API.
	FreeRTOS_SignalFromISR	How to perform thread signaling from an interrupt using CMSIS RTOS API.
	FreeRTOS_ThreadCreation	How to implement thread creation using CMSIS RTOS API.
	FreeRTOS_Timers	How to use timers of CMSIS RTOS API.



FreeRTOS: cube

- Ejemplos en el paquete de las Cube
 - No todos los ejemplos están para la placa Nucleo-L476
 - Este sí está

FreeRTOS_MPU	How to use the MPU feature of FreeRTOS.
--------------	---

- Interesante comentario en el *Readme* del ejemplo

For more details about FreeRTOS implementation on STM32Cube, please refer to UM1722 "Developing Applications on STM32Cube with RTOS".



FreeRTOS: cube

- ¡Anda, si hasta CubeMX lo incorpora!

 Quality RTOS & Embedded Software
Download FreeRTOS

Kernel Libraries Resources Community Support [Email List](#) 

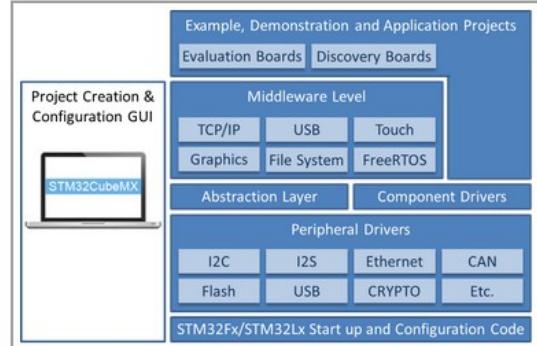
Home
Forums
Ecosystem
 Ecosystem Overview
 SafeRTOS
 OpenRTOS
 Fail Safe File System
 Tracealyzer
 WolfSSL SSL / TLS
 3rd Party BSPs
 Atmel
 Future Designs Inc.
 NXP
 ST
 Introduction
 STM32Cube
 Comprehensive Demo
 STM32CubeMX
 Work-flow Example
 FreeRTOS Interactive

STM32Cube
STM32 ARM Cortex-M Tools and Embedded Software

Introduction

STM32Cube is free embedded software from ST that provides all the drivers and middleware components necessary to get up and running quickly on STM32 ARM Cortex-M microcontrollers. STM32Cube includes FreeRTOS, but the use of FreeRTOS is optional.

- [Read more about STM32Cube embedded software](#)
- [Build a comprehensive demonstration project](#)


Example, Demonstration and Application Projects
Evaluation Boards | Discovery Boards
Project Creation & Configuration GUI
STM32CubeMX
Middleware Level
TCP/IP | USB | Touch
Graphics | File System | FreeRTOS
Abstraction Layer | Component Drivers
Peripheral Drivers
I2C | I2S | Ethernet | CAN
Flash | USB | CRYPTO | Etc.
STM32Fx/STM32Lx Start up and Configuration Code
[Click to Enlarge](#)

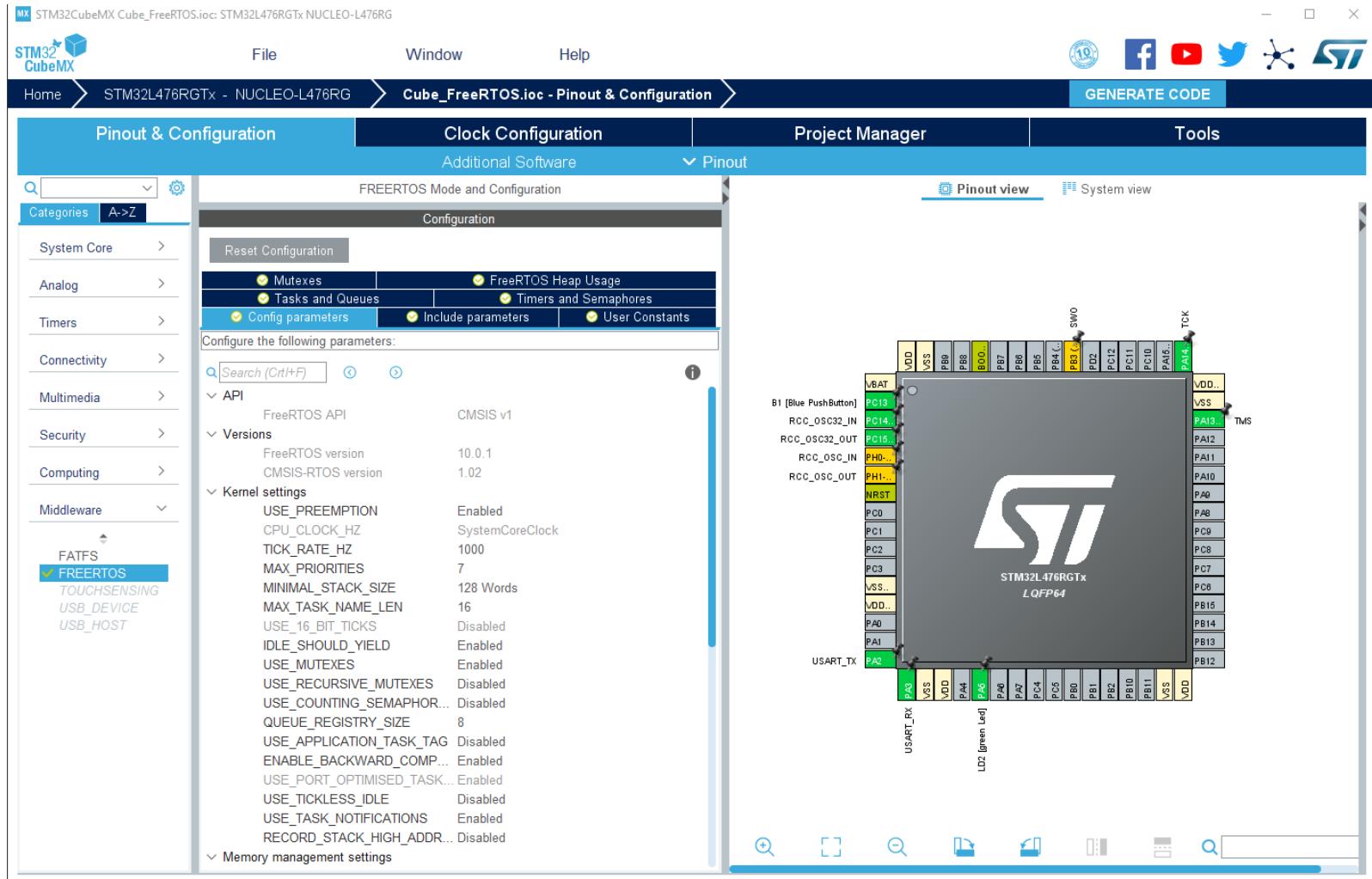
 STM32Cube_EMBEDDED SOFTWARE

https://www.freertos.org/FreeRTOS-Plus/BSP_Solutions/ST/index.html



FreeRTOS: cube

- ¡Anda, si hasta CubeMX lo incorpora!



FreeRTOS: cube

- Se genera algo como esto
- La notación de las llamadas se basa en CMSIS RTOS

File structure:

```

Application/User
  - main.c
  - freertos.c
  - stm32l4x_it.c
  - stm32l4xx_hal_msp.c
  - stm32l4xx_hal_timebase.c
Drivers/STM32L4xx_HAL_Driv
Drivers/CMSIS
Middlewares/FreeRTOS

```

Code content (main.c):

```

119
120     /* Create the thread(s) */
121     /* definition and creation of defaultTask */
122     osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
123     defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
124
125     /* definition and creation of myTaskPepe */
126     osThreadDef(myTaskPepe, StartTask02, osPriorityRealtime, 0, 128);
127     myTaskPepeHandle = osThreadCreate(osThread(myTaskPepe), NULL);
128
129     /* USER CODE BEGIN RTOS_THREADS */
130     /* add threads, ... */
131     /* USER CODE END RTOS_THREADS */
132
133     /* Start scheduler */
134     osKernelStart();
135
136     /* We should never get here as control is now taken by the scheduler */
137
138     /* Infinite loop */
139     /* USER CODE BEGIN WHILE */
140     while (1)
141     {
142         /* USER CODE END WHILE */
143
144         /* USER CODE BEGIN 3 */
145     }
146     /* USER CODE END 3 */

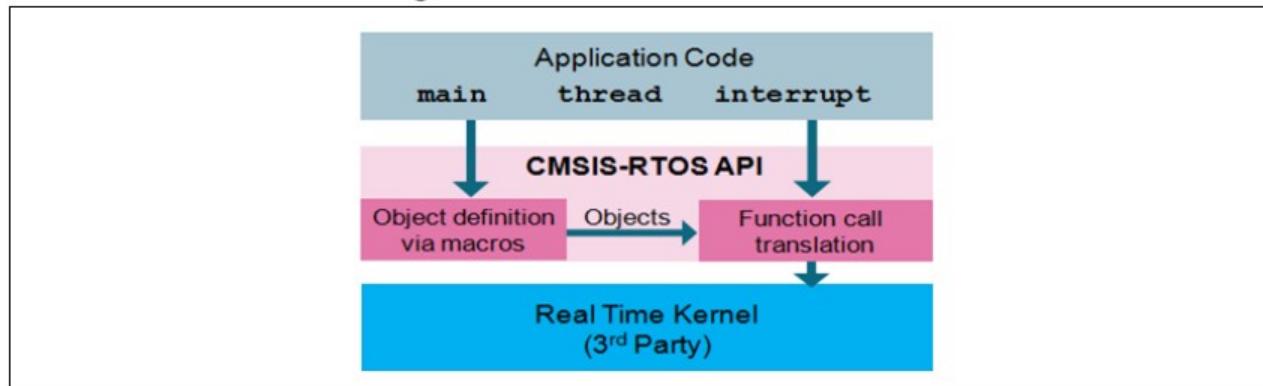
```



FreeRTOS: CMSIS-RTOS

- Kernel call are different in the first example and in the code generated by Cube, Why?
 - CMSIS-RTOS is the reason
 - Is a layer on top of any RTOS to exhibit a common layer
 - Check “UM1722: Developing applications on STM32Cube with RTOS” at
https://www.st.com/content/ccc/resource/technical/document/user_manual/2d/60/ff/15/8c/c9/43/77/DM00105262.pdf/files/DM00105262.pdf/jcr:content/translations/en.DM00105262.pdf

Figure 5. CMSIS-RTOS architecture



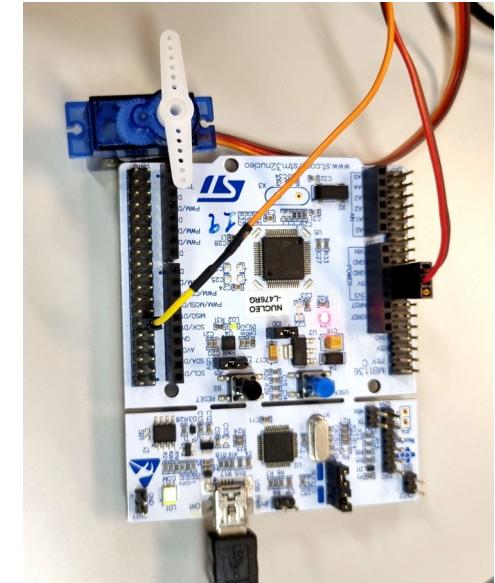
FreeRTOS: example “servo”

- Let it be the following mean program
 - And let have a trip to transform it to a “task”

```

42
43     error_code = servo_Init();
44     if (error_code != Error_NoError)
45     {
46         error_Handler(__FILE__, __LINE__, error_code);
47     }
48
49     servo_SetPosition(0); HAL_Delay(1000);
50
51     for(;;)
52     {
53         servo_SetPosition(-100); HAL_Delay(1000);
54         servo_SetPosition(-200); HAL_Delay(1000);
55         servo_SetPosition(-300); HAL_Delay(1000);
56         servo_SetPosition(-700); HAL_Delay(1000);
57         servo_SetPosition(-900); HAL_Delay(1000);
58         servo_SetPosition(0); HAL_Delay(1000);
59         servo_SetPosition(350); HAL_Delay(1000);
60         servo_SetPosition(900); HAL_Delay(1000);
61         servo_SetPosition(350); HAL_Delay(1000);
62         servo_SetPosition(900); HAL_Delay(1000);
63         servo_SetPosition(100); HAL_Delay(1000);
64         servo_SetPosition(200); HAL_Delay(1000);
65         servo_SetPosition(0); HAL_Delay(1000);
66     }
67

```



FreeRTOS: example “servo”

- First, let me transform the code in something better

```

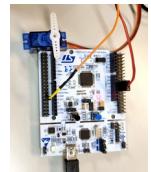
/* Includes -----
#include <stdio.h>
#include "stm32l4xx_hal.h"
#include "error.h"
#include "servo.h"

/* Private typedef -----
typedef struct
{
    int16_t tenth_degree;
    uint16_t timems;
} servostep_t;

/* Private define -----
#define NUM_STEPS 13

/* Private variables -----
const servostep_t my_sequence [NUM_STEPS] = {
{-100,1000},
{-200,1000},
{-300,1000},
{-700,1000},
{-900,1000},
{0,1000},
{350,1000},
{900,1000},
{350,1000},
{900,1000},
{100,1000},
{200,1000},
{0,1000}
};

uint8_t index = 0;
for(;;)
{
    servo_SetPosition(my_sequence[index].tenth_degree);
    HAL_Delay(my_sequence[index].timems);
    index++;
    if (index == NUM_STEPS)
    {
        index = 0;
    }
}
  
```

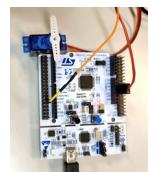


FreeRTOS: example “servo”

- Now, this is similar to a simple task, isn't it?
- Let's transform it in a task/thread!

```
uint8_t index = 0;
for(;;)
{
    servo_SetPosition(my_sequence[index].tenths_degree);
    HAL_Delay(my_sequence[index].timems);
    index++;
    if (index == NUM_STEPS)
    {
        index = 0;
    }
}
```

```
/*
* @brief Function implementing the myServo thread.
* @param argument: Not used
* @retval None
*/
void StartServo(void *argument)
{
    static uint8_t index = 0;
    for(;;)
    {
        servo_SetPosition(my_sequence[index].tenths_degree);
        osDelay(my_sequence[index].timems);
        index++;
        if (index == NUM_STEPS)
        {
            index = 0;
        }
    }
}
```



FreeRTOS: lab session

- CHALLENGE:
 - Create an app with two tasks:
 - 1 -> Increases a shared integer variable every 2 seconds. Reset the variable when it reaches 47 (counts from 0 to 46)
 - 2 -> Blinks the LED for 50 ms as many times as the value contained in the previous variable, then turns off the LED and halts for 2.5 seconds



FreeRTOS: lab session

- CHALLENGE:
 - Create an app with two tasks:
 - 1 -> Increases a shared integer variable every 2 seconds. Reset the variable when it reaches 47 (counts from 0 to 46)
 - 2 -> Blinks the LED for 50 ms as many times as the value contained in the previous variable, then turns off the LED and halts for 2.5 seconds

MAL
¡¡ Chapucero !!
Hay condiciones
de carrera



FreeRTOS: lab session

- Pensadlo ...

```

volatile uint32_t shared_counter;

void StartIncVar(void *argument)
{
    for(;;)
    {
        osDelay(2000);
        shared_counter++;
        if (shared_counter > 46)
        {
            shared_counter = 0;
        }
    }
}

void StartLEDBlink(void *argument)
{
    for(;;)
    {
        for(uint8_t i=0;i<shared_counter;i++)
        {
            LED_On();
            osDelay(50);
            LED_Off();
            osDelay(50);
        }
        osDelay(2500);
    }
}

```



FreeRTOS: Mutex

- Race conditions and interdependency problems can be solved using “mutual exclusion” mechanisms
- See https://en.wikipedia.org/wiki/Mutual_exclusion
- In the CS jargon, this is called a **MUTEX**
- In FreeRTOS you can use *xCreateMutex* to benefit of this mechanism
- But you are using CMSIS-RTOS overlay, so you can use
 - `osMutexCreate`, `osMutexAcquire`, `osMutexRelease`



FreeRTOS: Mutex

- Using Mutex in CMSIS-RTOS

```
/* Define the mutex */  
osMutexDef(osMutex);  
  
/* Create the mutex */  
osMutexId osMutex = osMutexCreate(osMutex(osMutex));  
  
if (osMutexWait(osMutex, mutexTWO_TICK_DELAY) != osOK)  
{  
    /* Toggle LED3 to indicate error */  
    BSP_LED_Toggle(LED3);  
}  
  
if (osMutexRelease(osMutex) != osOK)  
{  
    /* Toggle LED3 to indicate error */  
    BSP_LED_Toggle(LED3);  
}
```

(Fragmentos sacados de los ejemplos de la Nucleo-F413ZH)

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



FreeRTOS: Mutex

- Lets correct our lab work ...

```
void StartIncVar(void *argument)
{
    for(;;)
    {
        osDelay(2000);
        osMutexAcquire(myMutexBlinkHandle,0U);
        shared_counter++;
        if (shared_counter > 46)
        {
            shared_counter = 0;
        }
        osMutexRelease(myMutexBlinkHandle);
    }
}
```

```
void StartLEDBlink(void *argument)
{
    for(;;)
    {
        uint32_t cnt;
        osMutexAcquire(myMutexBlinkHandle,0U);
        cnt = shared_counter;
        osMutexRelease(myMutexBlinkHandle);
        for(uint8_t i=0;i<cnt;i++)
        {
            LED_On();
            osDelay(50);
            LED_Off();
            osDelay(50);
        }
        osDelay(2500);
    }
}
```



FreeRTOS: Semaphores

- Lets imagine that we want to reset the shared counter when the user key is pressed
 - It seems that an EXTx interrupt is adequate for this task, isn't it?

```
416 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
417 {
418     osMutexAcquire(myMutexBlinkHandle, 0U);
● 419     shared_counter = 0;
420     osMutexRelease(myMutexBlinkHandle);
421 }
```



FreeRTOS: Semaphores

- Interrupts is a typical hell for a RTOS-based app ... but necessary
- Handling they with the two-levels approach proposed in the subject continues being a great solution
- Let do it in the right way using semaphores
- Semaphores are used to synchronize and/or lock things,
 - e.g., fopen(), fprintf(), socket() ...

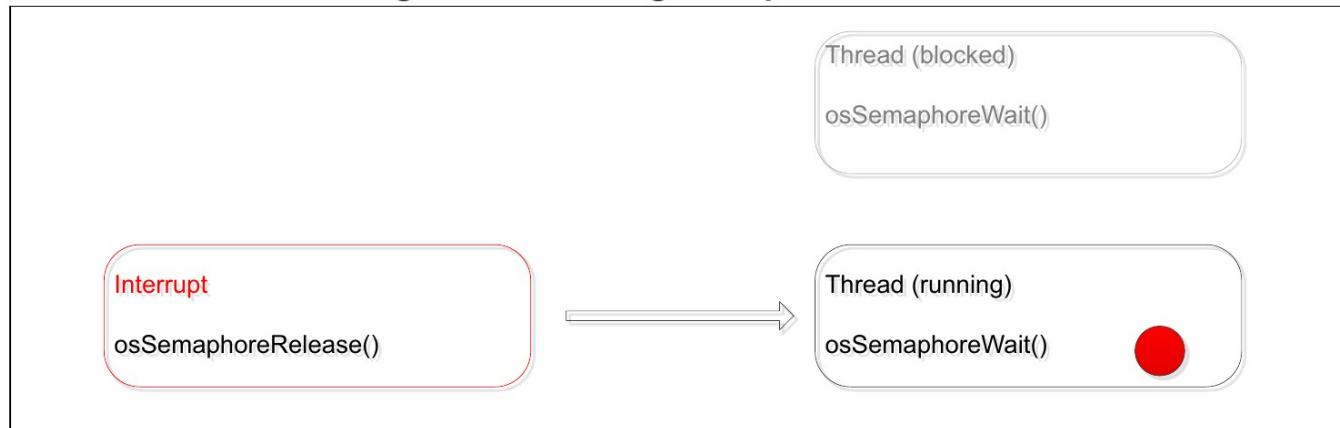


[https://en.wikipedia.org/wiki/Semaphore_\(programming\)](https://en.wikipedia.org/wiki/Semaphore_(programming))

FreeRTOS: Semaphores

- Lets go for a binary semaphore in CMSIS-RTOS applied to an ISR
 - From the recommend manual from St ...
 - NOTE: osSemaphoreWait() → osSemaphoreAcquire()

Figure 8. Obtaining semaphore from ISR



FreeRTOS: Semaphores

- Applying it!

```
/* Definitions for myBinarySemButton */
osSemaphoreId_t myBinarySemButtonHandle;
const osSemaphoreAttr_t myBinarySemButton_attributes = {
    .name = "myBinarySemButton"
};

/* creation of myBinarySemButton */
myBinarySemButtonHandle = osSemaphoreNew( 1, 0, &myBinarySemButton_attributes);
```

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    osSemaphoreRelease(myBinarySemButtonHandle);
}
```

```
void StartButtonTask(void *argument)
{
    for(;;)
    {
        if (osSemaphoreAcquire(myBinarySemButtonHandle, 0U) == osOK)
        {
            osMutexAcquire(myMutexBlinkHandle, 0U);
            shared_counter = 0;
            osMutexRelease(myMutexBlinkHandle);
        };
        osDelay(50);
    }
}
```





(Cajón de sastre)

- CMSIS RTOS
- Ver el UM xxxx de St sobre RTOS

