URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

GSyC

Sistemas Empotrados y de Tiempo Real

FreeRTOS en Arduino

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

FreeRTOS

- **FreeRTOS** es un kernel de tiempo real para dispositivos embebidos que está portado a 35 plataformas de microcontroladores.
  - Intel, ARM, Atmel, ESP, Xilinx, ….
- Liberado como software libre (MIT)
  - https://www.freertos.org/
- Permite organizar aplicaciones para microcontroladores como una colección de hilos independientes de ejecución.
- Ejecución de hilos basada en prioridades.
- En FreeRTOS, cada hilo de ejecución se denomina **tarea**

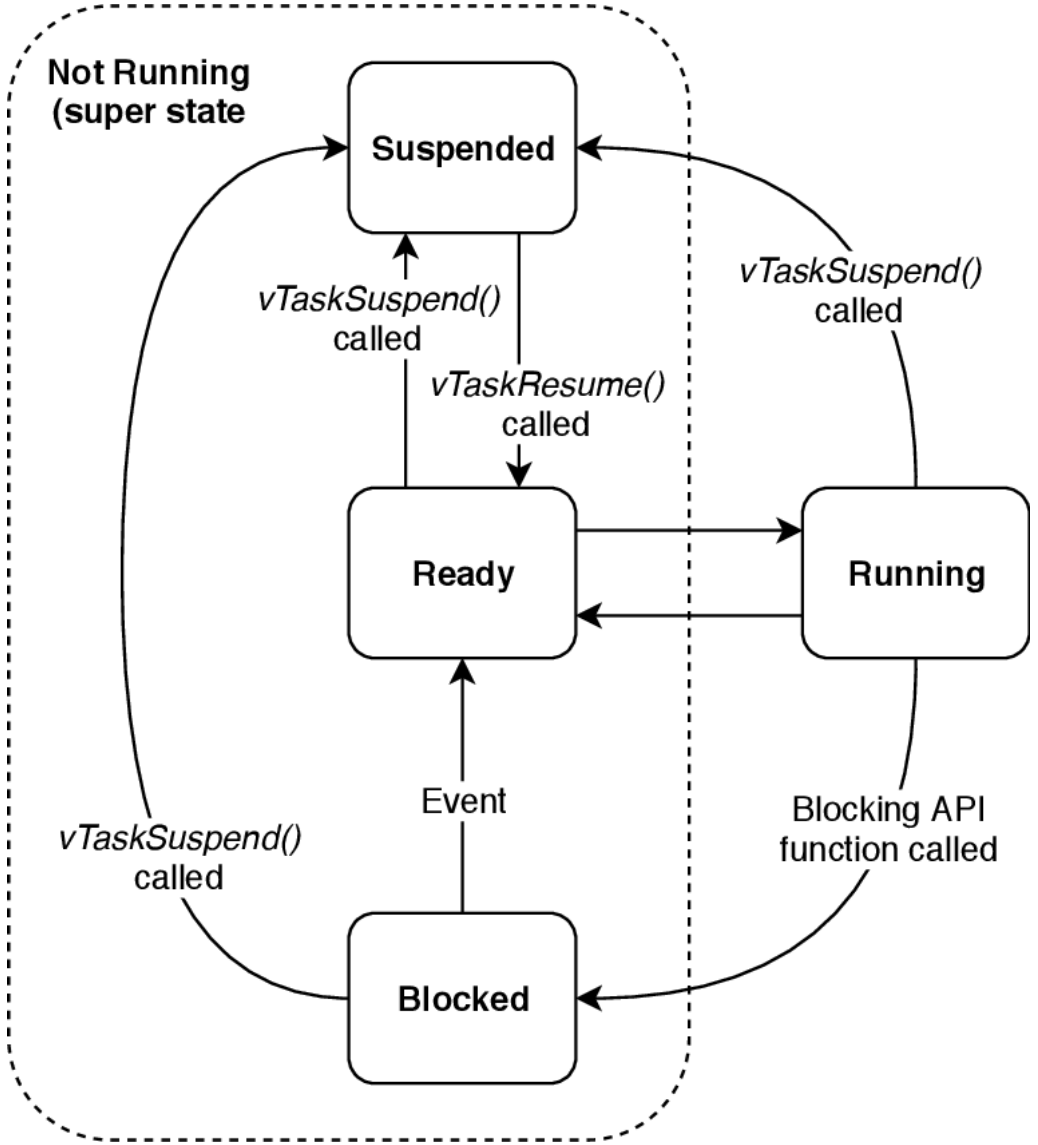
FreeRTOS - Beneficios

- Abstracción de la información del tiempo
- Escalabilidad
- Modularidad
- Facilita el desarrollo en equipo
- Utilización del tiempo “idle”

FreeRTOS - Características

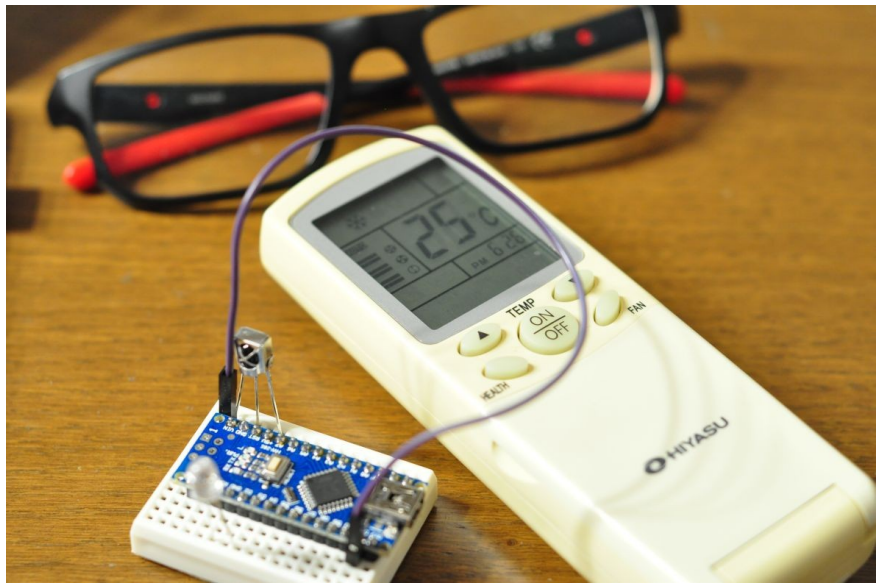
- Operación “pre-emptive”
- Asignación de prioridad a tareas muy flexible.
- Colas, utilizadas para el paso de mensajes entre tareas.
- Uso de semáforos binarios, recursivos, de conteo y de exclusión mutua.
- Funciones para consultar Tick/Idle
- Interrupciones anidadas

FreeRTOS

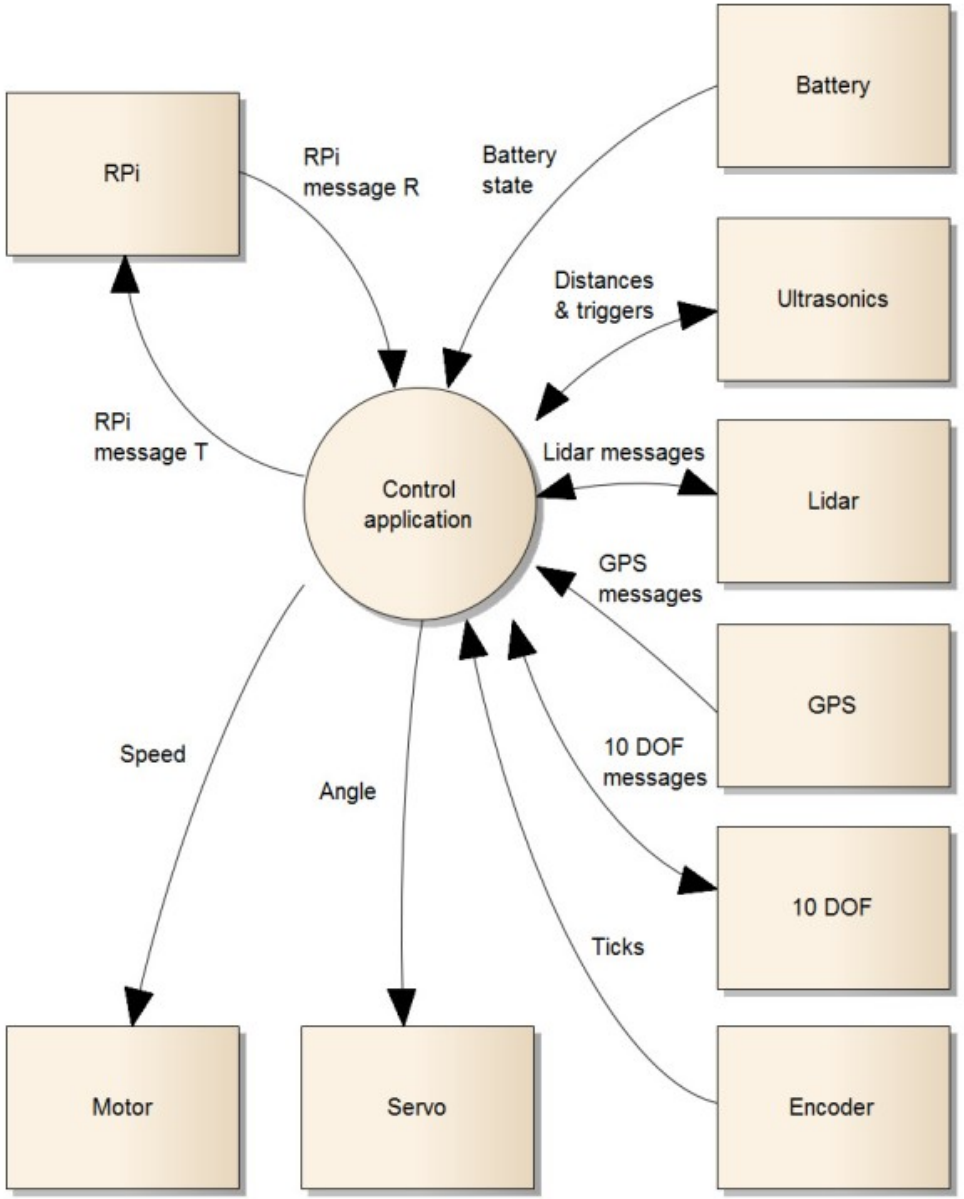

Not Running
(super state
Suspended
vTaskSuspend()
called
vTaskResume()
called
vTaskSuspend()
called
Ready
Running
Event
vTaskSuspend()
called
Blocking API
function called
Blocked

FreeRTOS

- Para comportamientos muy sencillos (1 actuador + 1 sensor) seguramente un sketch simple en Arduino es más que suficiente para asegurar el muestreo constante del sensor y comandar acciones al actuador.



FreeRTOS


RPi
RPi message R
Battery state
Battery
Distances & triggers
Ultrasonics
RPi message T
Control application
Lidar messages
Lidar
GPS messages
GPS
Speed
Angle
10 DOF messages
10 DOF
Ticks
Motor
Servo
Encoder

FreeRTOS

- La versión FreeRTOS para AVR se puede encontrar en:
  - https://github.com/feilipu/Arduino_FreeRTOS_Library
- **FreeRTOS** puede configurarse para diferentes time slices:
  - 15ms, 30ms, 60ms, 120ms, 250ms, 500ms, 1seg y 2seg
  - Si una tarea acaba antes del time-slice, ese tiempo restante es gestionado por el planificador.
- Tareas de la misma prioridad ejecutaran en base al TIME SLICE en round-robin
- Compatible con ATmega328, ATmega1284p, ATmega2560.
- Ejemplos:
  - https://github.com/feilipu/avrfreertos

FreeRTOS

```
#include <Arduino_FreeRTOS.h>

xTaskCreate(MyTask1, "TaskLed1", 100, NULL, 1, NULL);
```

- MyTask1: función que ejecutar la tarea
- “TaskLed1”: Nombre identificativo
- 100: Numero de palabras reservadas en la pila.
- NULL: Parámetros pasados a la función dela tarea
- 1: Prioridad asignada a la tarea
- NULL: puntero opcional para creación de tarea.

FreeRTOS

- Ejemplo de uso de FreeRTOS en Arduino.
- Prioridades en FreeRTOS → [0 – MAX_PRIORITY]
- 3 Tareas:
  - La primera tarea con prioridad 1 enciende el Led-1 durante 100 ms cada 1000ms.
  - La segunda tarea con prioridad 2 enciende el Led-2 durante 800 ms cada 1000ms.
  - La tercera tarea “IDLE” enciende el Led-3 cuando el sistema está ocioso

```
xTaskCreate(MyTask1, "TaskLed1", 100, NULL, 1, NULL);
xTaskCreate(MyTask2, "TaskLed2", 100, NULL, 2, NULL);
xTaskCreate(MyIdleTask, "IdleTask", 100, NULL, 0, NULL);
```

FreeRTOS

```
static void MyTask1(void* pvParameters)
{
  TickType_t xLastWakeTime, aux;

  while(1)
  {
    xLastWakeTime = xTaskGetTickCount();
    aux = xLastWakeTime;

    digitalWrite(PIN_T1,HIGH);
    digitalWrite(PIN_T2,LOW);
    digitalWrite(PIN_IDLE,LOW);


    while ( (aux - xLastWakeTime)*portTICK_PERIOD_MS < TIME_ON_T1) {
      aux = xTaskGetTickCount();
    }

    Serial.println("Task1");
    xTaskDelayUntil( &xLastWakeTime, ( DELAY_T1 / portTICK_PERIOD_MS ) );

  }
}
```

Si nuestra tarea es periódica debemos programarla como bucle infinito.

Definimos los encendidos y apagados de los LEDS de la aplicación.

Definimos el tiempo que dura el encendido de los LEDS

Utilizamos xTaskDelayUntil para definir el delay necesario antes de realizar la siguiente iteración.

FreeRTOS

```
// IDLE Task
static void MyIdleTask(void* pvParameters)
{
  while(1)
   {
    digitalWrite(PIN_T1,LOW);
    digitalWrite(PIN_T2,LOW);
    digitalWrite(PIN_IDLE,HIGH);
    Serial.println("Idle state");
    xTaskDelayUntil( &xLastWakeTime, ( PERIODIC_IDLE / portTICK_PERIOD_MS ) );
  }
}
```

Nuestra tarea IDLE solo ejecutará cuando el MCU esté “ocioso”, esto es, cuando no ejecuta ni la Tarea1 ni la Tarea2

Bibliografía

- FreeRTOS
  - https://es.wikipedia.org/wiki/FreeRTOS
- FreeRTOS documentation
  - https://www.freertos.org/Documentation/RTOS_book.html
- Libro
  - Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC Warren Gay, Apress, 2018.
- Paper:
  - Control System Based on FreeRTOS for Data Acquisition and Distribution on Swarm Robotics Platform

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación