

Arduino y FreeRTOS

Introducción

En esta ocasión vamos a realizar un breve repaso por el sistema operativo en tiempo real FreeRTOS para arduino. Como hemos comentado anteriormente, un sistema operativo en tiempo real (*Real Time Operating System, RTOS*) es aquel que pretende asegurar un patrón determinado (predecible) de ejecución de las tareas. Es particularmente interesante para sistemas embebidos con restricciones temporales de tiempo real. Es decir, si tenemos tres tareas de las cuales una tiene que ejecutarse necesariamente cada 3 milisegundos, el sistema operativo provee servicios para poder llevarlo a cabo.

FreeRTOS es un tipo RTOS diseñado para ser lo suficientemente pequeño para correr en un microcontrolador, aunque no está limitado a este tipo de hardware. Las características que posee son:

- Multitarea
- Semáforos y buzones de correo-
- Planificador cooperativo, Round Robin y basado en prioridades.
- Bajo consumo de memoria.
- Temporizador por software.

En este enlace están todas las características del RTOS:

http://www.freertos.org/FreeRTOS_Features.html (http://www.freertos.org/FreeRTOS_Features.html)

Las tareas son la mínima unidad de ejecución del sistema. En este tipo de sistema operativo, las tareas tienen su propio contexto de ejecución, independientes del resto y del Kernel. El contexto de ejecución es un espacio en memoria (normalmente memoria RAM) en la cuál se almacena el estado de la tarea. Dicho estado se representa con:

- Siguiente instrucción a ejecutar.
- Variables internas del proceso.
- Valor de los registros del procesador justo antes de abandonar la CPU.

El proceso de ejecutar una tarea a otra se llama *cambio de contexto* y consiste en almacenar *contexto* del proceso en el *stack* (zona de memoria reservada) de la tarea. Se ejecuta el planificador, que decide qué tarea ha de ejecutarse posteriormente, y por último se restaura el contexto en el procesador.



(<https://ingeware.files.wordpress.com/2016/08/mezcla.jpg>)


Respuesta temporal de los distintos tipos de algoritmos de planificación

- Primer ejemplo: *Blink Led*


Como primer ejemplo vamos a usar el código fuente que esta en este enlace:

<https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc>
(<https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc>)

En primer lugar, hay que cargar la librería de FreeRTOS y definimos las funciones (tareas) que vamos a ejecutar. *pvParameters* indica parámetros de entrada a la tarea.

 (https://ingeware.files.wordpress.com/2016/08/definicion2bllibrerias2by2btareas.png?w=300)
Declaración de librerías y tareas.

Dentro del *main* del programa, que en Arduino correspondería al *setup*, creamos las tareas en el sistema operativo.

 (https://ingeware.files.wordpress.com/2016/08/creacion2btareas2bfreertos.png)
Creación de tareas en el setup


Donde los parámetros para crear la función son:

1. Dirección de la función de la tarea.
2. Nombre de la tarea.
3. Tamaño de la tarea.
4. Parámetros de entrada de la tarea.
5. Prioridad.
6. Handler para la tarea.

Cuando termine el setup, el Kernel toma el control y comienza a ejecutar las tareas en función de su prioridad. Por último queda completar el código de cada tarea. En este caso tenemos una tarea que lee valores por el puerto serie y otra, que hace parpadear el led conectado al pin 13.

Observamos que el código central de la tarea está dentro de un bucle infinito, así estará constantemente ejecutando el mismo código. Destacar, aunque sea trivial, que, si no existiera el SO la primera tarea en ejecutarse, se adueñaría indefinidamente de la CPU. Es precisamente el sistema operativo el que provee mecanismos de intercambio entre tareas, lo que llamamos multitarea.

La función *vTaskDelay* duerme la tarea un determinado periodo de tiempo, tiempo durante el cual es aprovechado por otras tareas para realizar sus operaciones. Así no se malgasta ciclos de CPU haciendo nada.

 (https://ingeware.files.wordpress.com/2016/08/task2banalog2bread.png)
TaskAnalogRead


◦ Segundo ejemplo: Semáforos

Un semáforo es un mecanismo de sincronización entre tareas, que permite controlar el acceso a un único recurso por parte de diferentes tareas. Por ejemplo, supongamos que varias tareas necesitan hacer uso del puerto serie. Si la tarea A está enviando caracteres y salta la interrupción para realizar el cambio de contexto,


y la tarea B comienza a usar el puerto serie, ¿Qué sucede? Lo que sucede es, que el equipo que esté al otro lado del puerto serie leyendo las comunicaciones, lee un fragmento de lo que envía la tarea A y lo que envía la tarea B, dando como resultado cualquier cosa. Para evitar esto, se utilizan los semáforos. Su funcionamiento es parecido a un semáforo de tráfico. La tarea que vaya a usar un recurso, lo notifica al semáforo. Si otra tarea va a usar el mismo recurso, el semáforo le indica que no está disponible y no le dejaría usarlo. Cuando la tarea originaria termina con el recurso, le vuelve a notificar al semáforo, que dicho recurso queda libre, así la segunda tarea puede conseguir el control del recurso compartido. A este tipo de semáforos, se les denomina *semáforos binarios*.

El código de ejemplo se encuentra en este enlace: <https://www.hackster.io/feilipu/using-freertos-semaphores-in-arduino-ide-b3cd6c> (<https://www.hackster.io/feilipu/using-freertos-semaphores-in-arduino-ide-b3cd6c>)

En primer lugar, hay que declarar el tipo de variable correspondiente para usar los semáforos. Dicha variable es global.


 https://ingeware.files.wordpress.com/2016/08/declaracion2bvariable2bglobla2btipo2bsemaforo.png?w=300	Creación variable tipo semáforo
--	---------------------------------

En el setup se crea el semáforo mutex, se comprueba que no sea nulo (es decir, que se ha creado) y se libera el semáforo (es decir, que la tarea puede acceder al recurso).

 https://ingeware.files.wordpress.com/2016/08/creacion2bde2bsemaforo2ben2bsetup.png	Creación semáforo en el setup
--	-------------------------------

Ahora es responsabilidad de cada tarea hacer uso de este mecanismo. Antes de acceder al recurso compartido, se pregunta si está libre. Si lo está accedemos a él. En este caso, se espera durante 5 ticks del sistema, hasta que el recurso quede liberado. Una vez realizado las operaciones, liberamos el recurso con: `xSemaphoreGive(xSerialSemaphore)`.

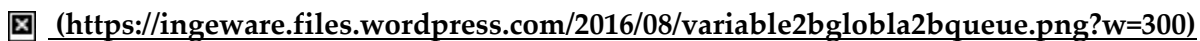
Lo mismo ocurre con todas las tareas que pretendan usar el mismo recurso.

 https://ingeware.files.wordpress.com/2016/08/task2banalog2bread.png	TaskAnalogRead
--	----------------

◦ Tercer ejemplo: Comunicación entre tareas

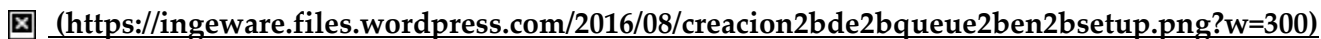
El sistema operativo también ofrece servicios para enviar información entre las tareas. Nosotros vamos a usar las *queue* o colas. Las colas son un tipo de dato, en la que la información se va apilando, de forma que es necesario el acceso secuencial a los datos. Conceptualmente ayuda imaginarse este tipo de datos como una pila de libros unos encima de otros. Comencemos con la explicación del código:

Primero hay que declarar el tipo *queue*, como una variable global.

 <https://ingeware.files.wordpress.com/2016/08/variable2bglobla2bqueue.png?w=300>

Declarar variable tipo queue

En el setup, notificamos al Kernel crear un queue. El primer argumento indica la cantidad de elementos que se pueden almacenar de forma secuencial. El segundo el tamaño que ocupa cada objeto.

 <https://ingeware.files.wordpress.com/2016/08/creacion2bde2bqueue2ben2bsetup.png?w=300>

Creación semáforo en el setup

En nuestro código de ejemplo, tenemos dos tareas. Una tarea que atiende a las comunicaciones entrantes por el puerto serie y otra que escribe por él. Lo que hacemos es un eco, es decir, el número que enviamos por el puerto serie lo recoge una tarea, y esta le envía el mismo número a la tarea que se encarga de escribir. El código de ambas tareas se muestran en las imágenes siguientes:

 <https://ingeware.files.wordpress.com/2016/08/serial2bin.png?w=274>

TaskSerial_in

 <https://ingeware.files.wordpress.com/2016/08/serial2bout.png?w=300>

TaskSerial_out

Enlaces:

<http://www.freertos.org/> (<http://www.freertos.org/>)

https://github.com/andres7293/FreeRTOS-InterTask_Comm-Example

(https://github.com/andres7293/FreeRTOS-InterTask_Comm-Example)