Actividad-03. Parpadeo intermitente. Multitarea

> Página extraída de Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Enunciado

Ya hemos visto como hacer funcionar a los LEDs de diferentes formas, pero ahora vamos a introducir un concepto que nos puede resultar bastante útil en el futuro, se trata de la multitarea en ArduinoBlocks.

Teoría

Este apartado se extrae de ArduinoBlocks - FreeBook disponible en Free Book (online & updated).

ArduinoBlocks nos permite utilizar una capa para implementar un sistema multitarea avanzado basado en FreeRTOS (del inglés Real Time Operating System), que es un sistema operativo de tiempo real kernel para dispositivos embebidos para plataformas de microcontrolador que se distribuye bajo licencia MIT. Este sistema permite crear tareas que se ejecutarán de forma paralela (virtualmente). En microcontroladores modestos como el Arduino UNO, Nano o incluso MEGA la multitarea con FreeRTOS es bastante limitada y consume gran parte de los recursos de nuestro Arduino, en caso de necesitar de un sistema multitarea más potente podemos optar por usarlo en placas basadas en ESP8266 o ESP32 con mucha más potencia y recursos (especialmente el ESP32 con doble núcleo y gran potencia de procesamiento y memoria interna)

Los sistemas software de multitarea utilizan un planificador o scheduler que se encarga de repartir el tiempo de procesamiento entre las distintas tareas, de forma que a cada una le toca un tiempo de microcontrolador para ejecutar un poquito de su parte de programa.

En las web de freeRTOS, en su entrada de menú Kernel podemos encontrar los conceptos básicos de multitarea y de programación que vamos a extractar seguidamente.

Conceptos básicos de multitarea

Un procesador convencional como el de Arduino UNO solo puede ejecutar una tarea a la vez, pero al cambiar rápidamente entre tareas, un sistema operativo multitarea puede hacer que parezca que cada tarea se ejecuta simultáneamente. Esto es lo que se representa en el diagrama de la Figura siguiente que muestra el patrón de ejecución de tres tareas con respecto al tiempo. Los

nombres de las tareas están codificados por colores y escritos a la izquierda. El tiempo se mueve de izquierda a derecha y las líneas de colores muestran qué tarea se está ejecutando en un momento determinado. El diagrama superior demuestra el patrón de ejecución concurrente percibido, y el inferior el patrón de ejecución multitarea real.

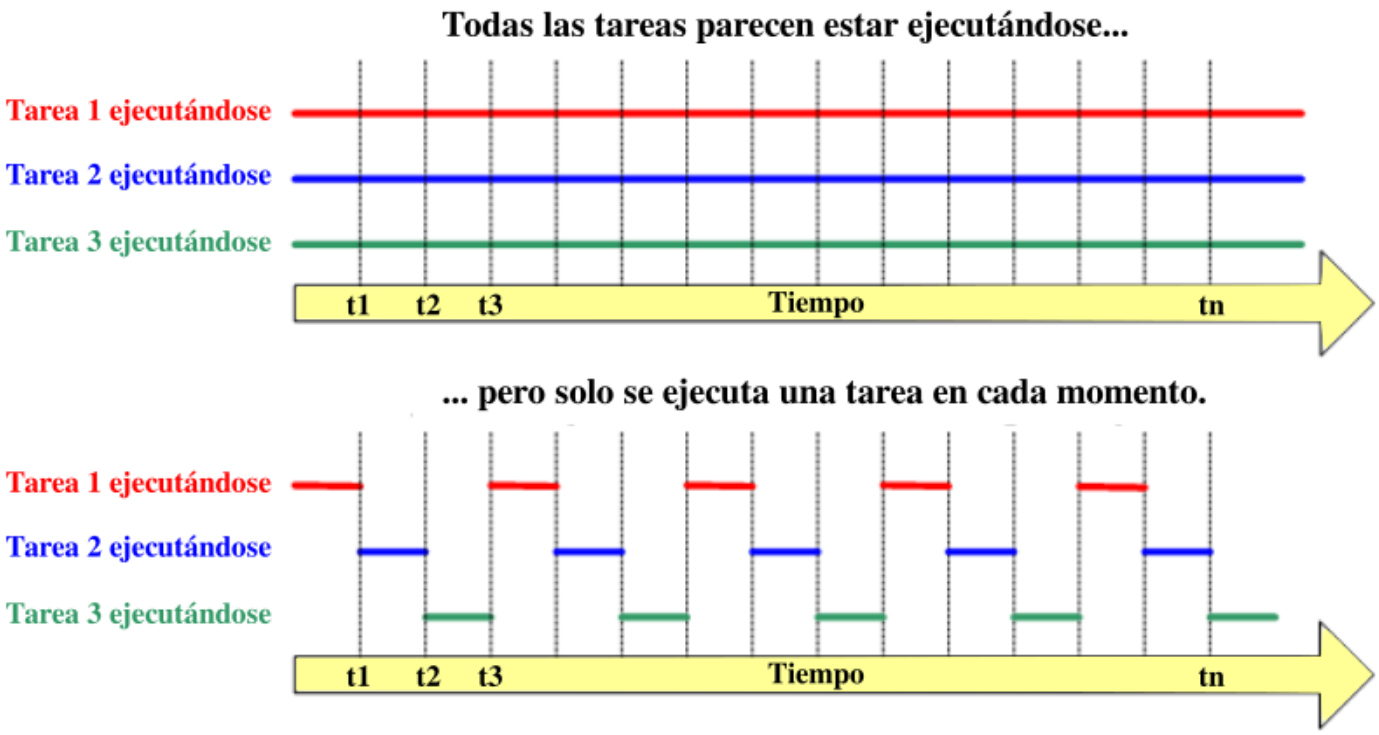

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Patrón de ejecución de tres tareas con respecto al tiempo

Programación

El programador es quien debe decidir qué tarea debe ejecutarse en un momento determinado. El kernel o núcleo puede suspender y luego reanudar una tarea muchas veces durante el tiempo de vida de la tarea.

Además de ser suspendida involuntariamente por el núcleo o kernel, una tarea puede optar por suspenderse a sí misma. Hará esto si desea retrasar (**dormir**) por un período fijo o esperar (**bloquear**) a que un recurso (por ejemplo, un puerto serie) esté disponible, o que ocurra un evento (por ejemplo, presionar una pulsador). Una tarea bloqueada o inactiva no se puede ejecutar y no se le asignará ningún tiempo de procesamiento.

En la Figura siguiente vemos un posible diagrama de ejecución de tres tareas analizado punto por punto en distintos instantes de tiempo. En los círculos se representan los instantes de tiempo t1 a t10.

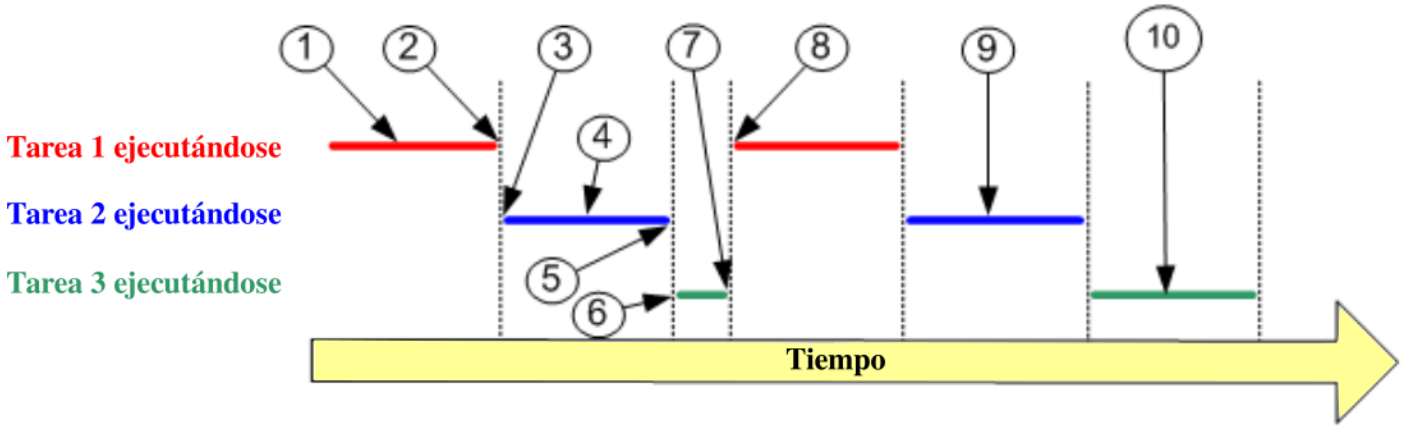

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Diagrama de ejecución de tres tareas en el tiempo

t1: la tarea 1 se está ejecutando.
t2: en el kernel se suspende, o mejor dicho se intercambia, la tarea 1 .
t3: se reanuda la tarea 2.
t4: mientras se ejecuta la tarea 2 el procesador bloquea el puerto serie para su acceso exclusivo.
t5: el kernel suspende la tarea 2.
t6: el kernel reanuda la tarea 3.
t1: la tarea 3 intenta acceder al puerto serie y lo encuentra bloqueado por lo que no puede continuar y se suspende.
t8: el kernel reanuda la tarea 1 .
t9: al ejecutarse de nuevo la tarea 2 se desbloquea el puerto serie.
t10: la tarea 3 ahora si puede acceder al puerto serie y se ejecuta al completo

Planificadores

Los planificadores de multitarea permiten asignar a cada tarea una prioridad, para así darle preferencia a las tareas más críticas o que necesitan más tiempo de procesamiento. Si creamos muchas tareas con “alta” prioridad puede que afectemos a las demás dejando poco tiempo de procesamiento para ellas. En la Figura siguiente vemos un esquema de varias tareas con distintas prioridades, variando así su tiempo de microprocesador asignado.

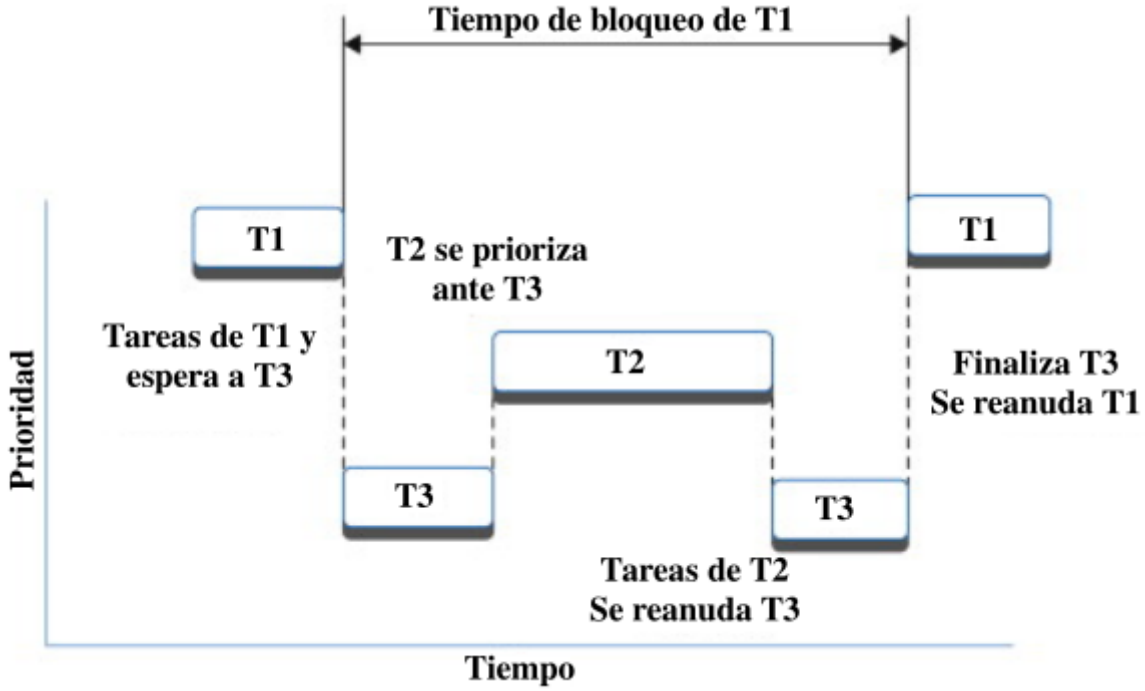

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Distintas prioridades en tres tareas

Cada tarea tiene su propio espacio de memoria, por lo que crear demasiadas tareas también puede dejarnos el procesador sin memoria. Si la memoria asignada a las tareas tampoco es suficiente para almacenar los datos se podría reiniciar de forma inesperada el Arduino, o funcionar incorrectamente, es decir que como siempre, hay que ser consciente de los limitados recursos de los que disponemos.

Semáforos

Con la introducción teórica a la multitarea vista, debemos hacernos otra pregunta: ¿Qué pasa si una tarea accede a un recurso o variable, y el sistema multitarea le da el control a otra tarea y por tanto ese proceso falla o quizás otra tarea acceda al mismo recurso y se solapen?

Para ese problema de convivencia entre tareas se inventaron los “semáforos”, en concreto el que más nos interesa es el semáforo “mutex” o de exclusión mutua, que permite que bloqueemos el sistema multitarea, hagamos lo que tengamos que hacer crítico, y luego liberemos el control. Por supuesto estas tareas críticas deben ser lo más cortas y atómicas posibles: una escritura crítica en una variable, un envío de un dato, una actualización de una pantalla LCD,... siempre cosas simples. Los semáforos debemos usarlos en casos que tengamos claro que se pueden crear conflictos, pues su abuso puede hacer que el sistema multitarea empiece a fallar.

En la Figura siguiente vemos el esquema de acceso a un recurso desde dos tareas diferentes.

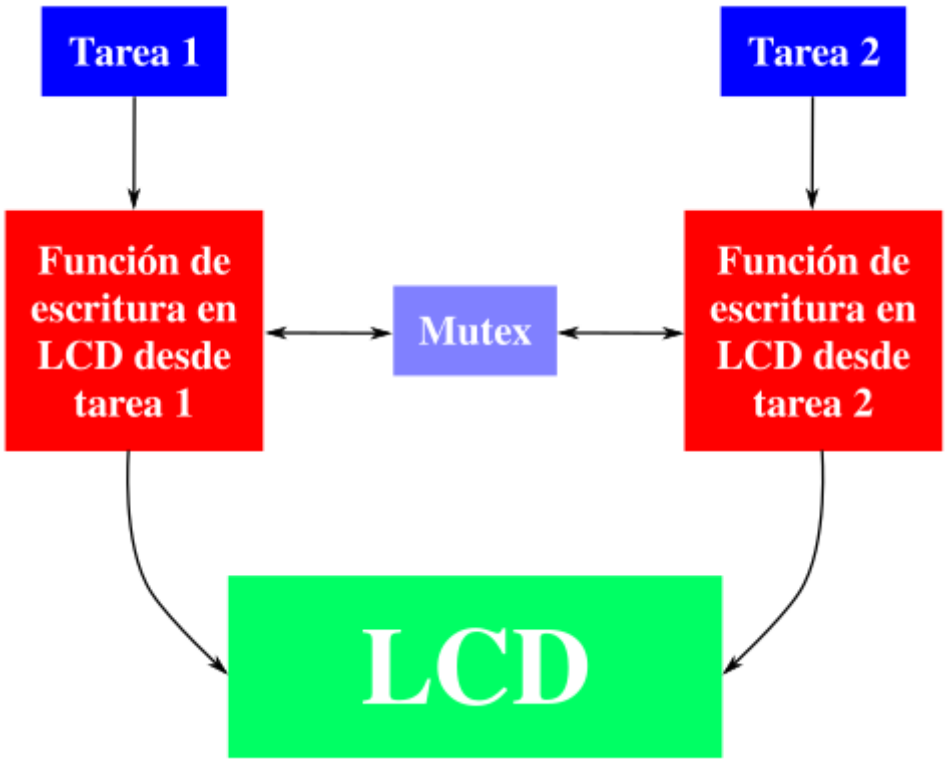

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Esquema de acceso a un mismo recurso por parte de 2 tareas diferentes

En la ESP32 STEAMakers

Una ventaja que posee la ESP32 Plus STEAMakers es que, al estar basado en un ESP32 con dos microcontroladores internos, podemos hacer que cada microcontrolador trabaje en una tarea y esto si sería multitarea real, o en varias tareas mezclando la multitarea real con la simulada. Esto también se puede hacer en las placas Arduino UNO, pero con el ESP32 tenemos más potencia para realizar estas tareas.

Esquema de acceso a un mismo recurso por parte de 2 tareas diferentes

Bloques esperar

¿Qué pasa con los bloques tipo “esperar” que estaban tan prohibidos en la programación de Arduino cuando queríamos simular una multitarea antes de tener estos bloques? Pues seguimos teniéndoles bastante tirria. Aunque en teoría podríamos usarlos, un bloque esperar hace pensar al microcontrolador que está haciendo algo útil, cuando en realidad no es así, por lo que el sistema multitarea querrá asignarle tiempo de procesamiento a la tarea, aunque sea para eso, ¡para no hacer nada!

Tenemos una solución, tenemos un nuevo bloque de esperar “task friendly” que en lugar de esperar sin hacer nada le dice al sistema: ¡voy a estar un rato sin hacer nada, permite ejecutar otras tareas mientras y luego vuelves! ….Mucho más “friendly”, claro que sí.

Bloques

Con toda esta información pasamos a ver los bloques disponibles para poner todo ésto en marcha.

Imagenes Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Bloque	Descripción
Bloque ejecutar tarea	Permite crear una nueva tarea con su bloque de “inicializar” y su “bucle” al igual que la tarea original de Arduino. Debemos asignar una prioridad a cada tarea, por defecto dejaría todas a “baja” y luego iría ajustando si hace falta. Para gestionar mejor las prioridades, es recomendable en algunos casos no utilizar el “inicializar” y “bucle” propio de Arduino que suele tener preferencia sobre todas estas tareas y es más difícil de equilibrar las prioridades.
Bloque esperar en esta tarea	El bloque esperar óptimo para tareas, pues deja funcionar al resto de tareas de forma más óptima mientras se espera en ésta. Este bloque tiene menos precisión que el bloque “esperar” original, si necesitamos hacer esperas muy precisas (o de menos de 20 ms) debemos usar el “esperar” tradicional. Pero nos servirá en la mayoría de casos.
Bloque bloqueo exclusivo	Si tenemos que hacer alguna acción crítica que no queremos que sea interrumpida internamente por el planificador del sistema multitarea podemos poner este bloque y dentro los bloques críticos. (no utilizar si no es estrictamente necesario)
Bloque establecer memoria	Cada tarea tiene su propio espacio de memoria reservado, esta es la cantidad por defecto para las tareas (192 bytes), si necesitamos ajustarla podemos utilizar este bloque en el “inicializar” principal y se ajustará para todas las tareas. Un mal ajuste puede provocar reinicios del microcontrolador o mal funcionamiento.
Bloque y destruir tarea	Las tareas en principio, igual que el bucle de Arduino, están pensadas para ejecutarse de forma indefinida, si en un caso una tarea deja de ser necesaria la forma de terminarla es con este bloque que parará la ejecución y liberará la memoria de la tarea en la que se ejecuta.

En la TdR STEAM

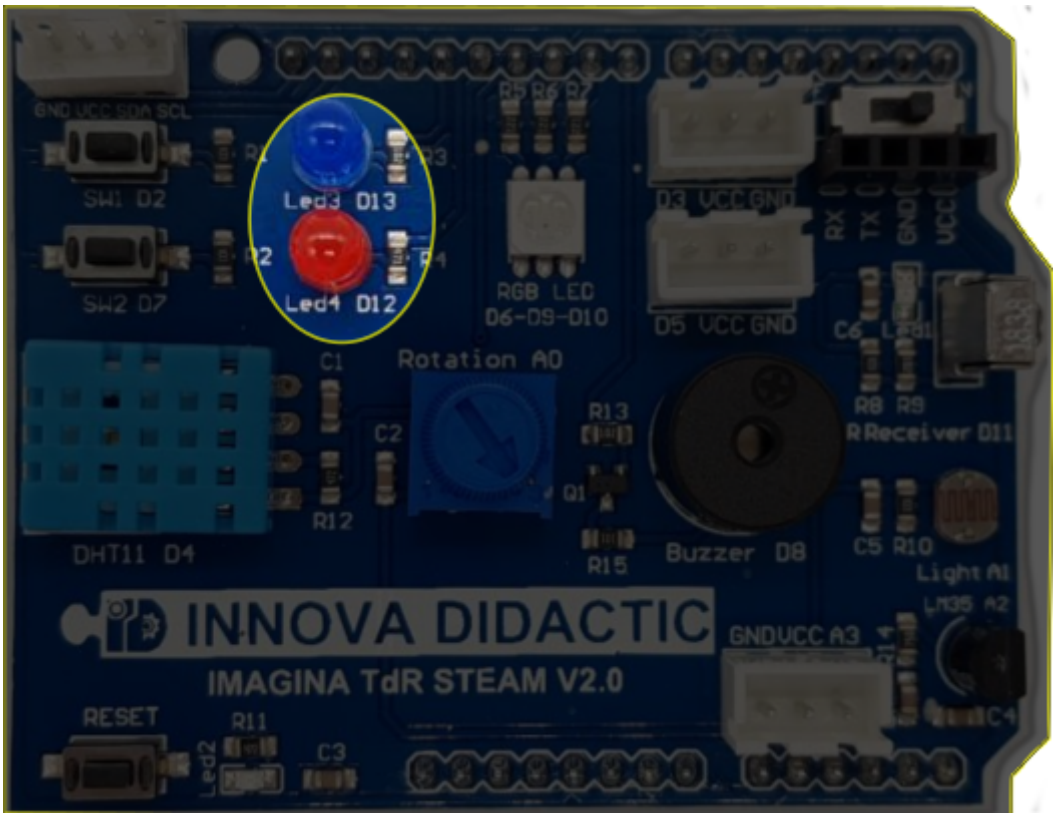

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Programando la Actividad

Vamos a hacer que los dos diodos parpadeen de forma independiente con un programa como el siguiente, que lo tenemos disponible en Actividad-03. Parpadeo intermitente. Multitarea

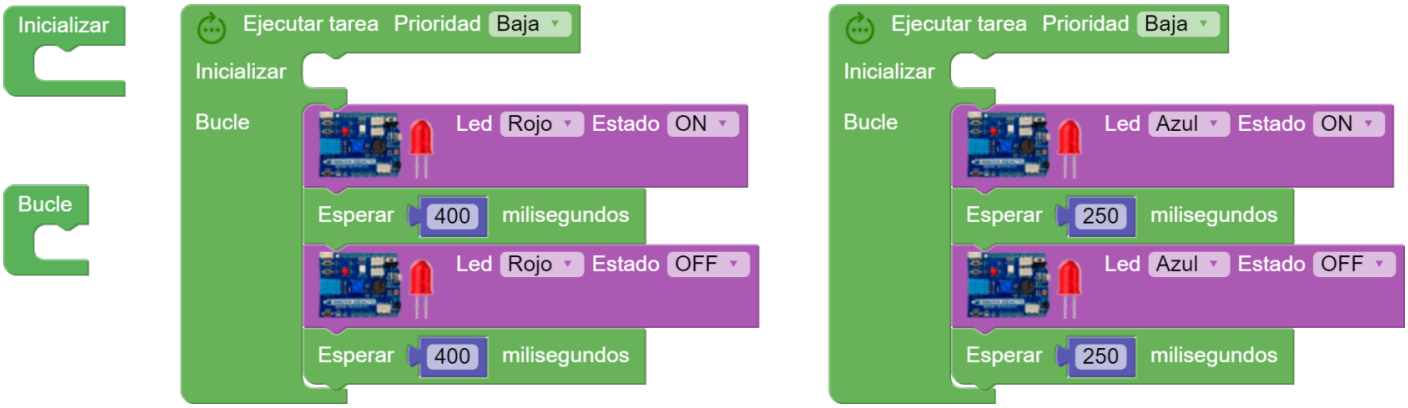

Imagen Federico Coca Notas sobre ESP32 STEAMakers CC-BY-SA

Retos de ampliación

A3.R1. Hacer que los LEDs rojo y azul trabajen en multitarea modificando los tiempos.

Revision #2
Created 27 December 2022 20:45:44 by Javier Quintana
Updated 10 January 2023 09:50:52 by Javier Quintana