



## Multi-Tarea con Arduino (UNO, MEGA, Leonardo)



Juanjo López

<http://www.arduinoblocks.com>

## Introducción

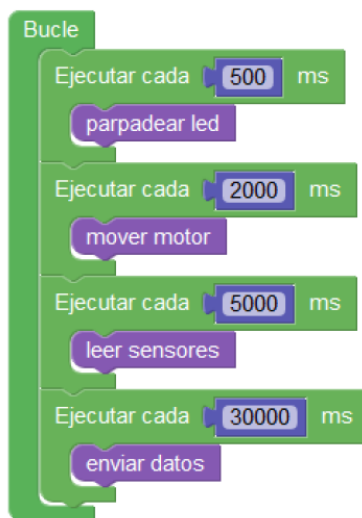
Arduino utiliza un microcontrolador bastante sencillo y modesto comparado con lo que podemos encontrar hoy actualmente en el mercado, pero sus fines educativos lo hacen aún un dispositivo muy útil en el aula (la mayoría de los proyectos apenas usan un 30% de la potencia del microcontrolador)

Dentro del entorno Arduino hay que destacar al Arduino MEGA, que aún siendo también modesto, nos ofrece un poco más de potencia y capacidad de memoria que sus hermanos pequeños UNO y Leonardo.

El problema la mayoría de las veces es que desaprovechamos mucha potencia por la forma en que programamos nuestro Arduino.

El microcontrolador sólo puede hacer una cosa a la vez, sólo puede ejecutar una instrucción de programa al mismo tiempo pues sólo tenemos un núcleo de procesamiento (no hay multi-core como en otros microcontroladores más potentes). Por lo que la única forma de simular una multitarea sería partir nuestro programa en pequeñas tareas y darle un tiempo a cada uno, y los más importante... ¡¡¡ que ninguna bloquee la ejecución del procesador durante mucho tiempo !!! Pues en ese caso las demás tareas se quedarán en el olvido... y ésto, a veces es fácil de implementar, pero otras no tanto...

*Aproximación a multitarea dividiendo en pequeñas acciones  
(cada tarea no debe bloquear al resto!):*

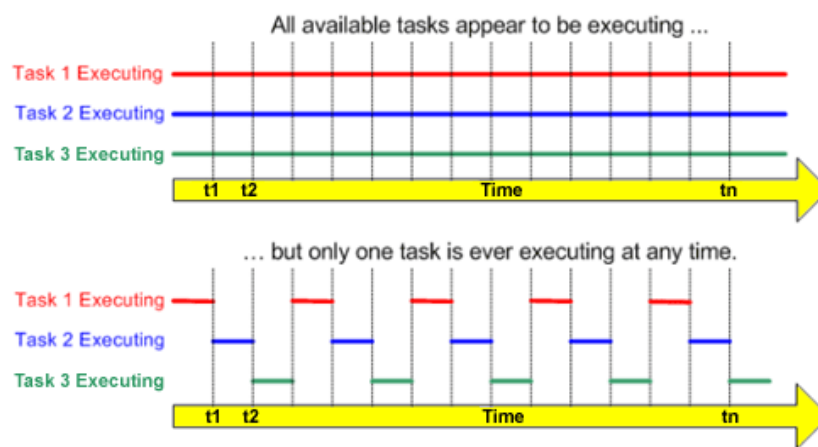


La opción que usan los sistema multitarea con un sólo procesador (por ejemplo los ordenadores que hasta no hace muchos años sólo tenían un núcleo) es implementar una capa por software que se encargue de gestionar el tiempo del microprocesador y dar a cada tarea el tiempo que crea que es necesario para que se ejecute correctamente... Esta tarea la realiza... ¡el sistema operativo!

Arduino no tiene un sistema operativo, pero sí podemos añadir una librería que nos haga las funciones de un sistema operativo, por lo menos en cuanto a la gestión del tiempo se refiere. Existen diferentes implementaciones para gestionar una multitarea más o menos real en Arduino, pero siempre tenemos que tener en cuenta la capacidad limitada de Arduino y que si le pedimos más de lo que puede procesar, el microcontrolador no dará a basto y no podrá ejecutar todas las tareas en tiempo real o incluso se reiniciará.

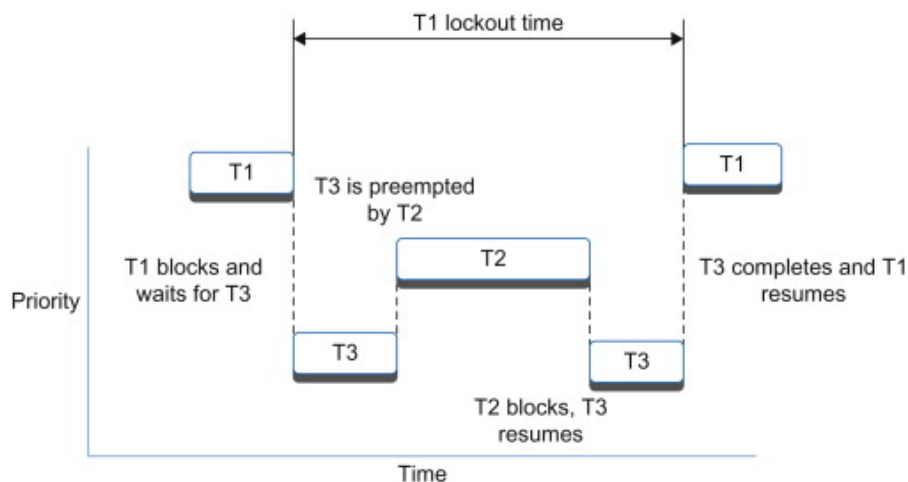
Los sistemas software de multitarea utilizan un planificador (scheduler) que se encarga de repartir el tiempo de procesamiento entre las distintas tareas, de forma que a cada una le toca un tiempo de microcontrolador para ejecutar un poquito de su parte de programa.

*Esquema con 3 tareas que se van alternando, y en apariencia parece que las 3 se ejecutan a la vez, en este caso el tiempo de procesador se divide igual para cada tarea:*



Los planificadores de multitarea permiten además 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...

*Esquema de varias tareas con distintas prioridades, variando así su tiempo de microprocesador asignado:*

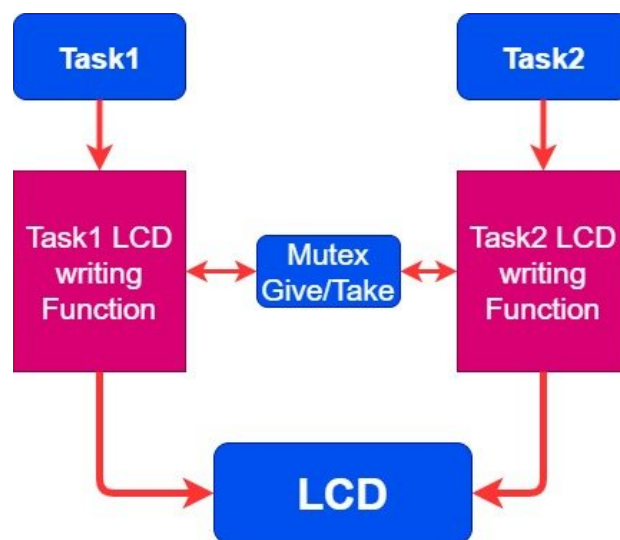


Cada tarea tiene su propio espacio de memoria, por lo que crear demasiadas tareas también puede dejarnos el procesador sin memoria. Si la memoria asignada a las tareas tampoco es suficiente para almacenar los datos se podría reiniciar de forma inesperada el Arduino, o funcionar incorrectamente... como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

Con esta introducción teórica a la multitarea, 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 solapan?

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.

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

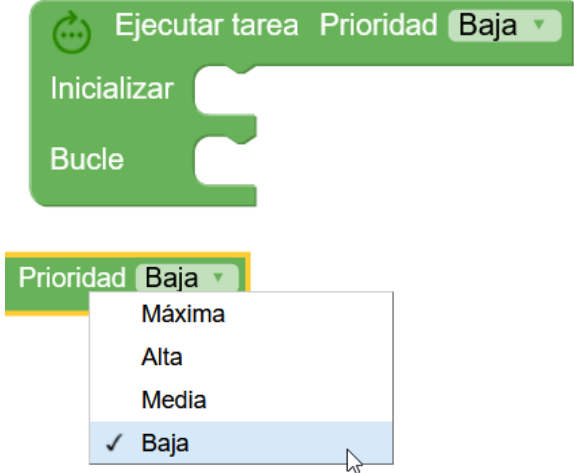

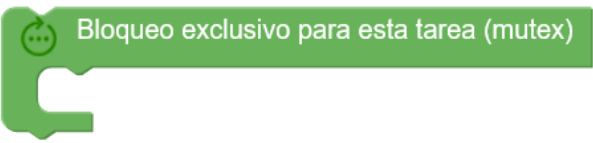
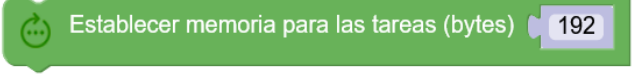
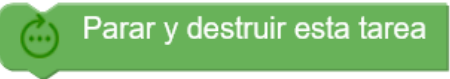


Y por último, pero no menos importante ¿Qué pasa con los bloques tipo “esperar” que estaban tan prohibidos en la programación de Arduino cuando queríamos simular una multitarea antes de tener estos bloques? Pues seguimos teniéndoles bastante tirria. Aunque en teoría podríamos usarlos, un bloque esperar 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!

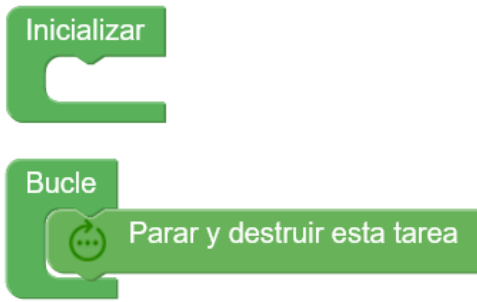
Tenemos una solución, tenemos un nuevo bloque de esperar “*task friendly*” que en lugar de esperar sin hacer nada le dice al sistema: *¡voy a estar un rato sin hacer nada, permite ejecutar otras tareas mientras y luego vuelves!* ....Mucho más “*friendly*”, claro que sí.

## Bloques

Con toda esta información pasamos a ver los bloques disponibles para poner todo ésto en marcha y después algunos ejemplos:

	<p>Permite crear una nueva tarea con su bloque de “inicializar” y su “bucle” al igual que la tarea original de Arduino.</p> <p>Debemos asignar una prioridad a cada tarea, por defecto dejaría todas a “baja” y luego iría ajustando si hace falta.</p> <p>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.</p>
	<p>El bloque esperar óptimo para tareas, pues deja funcionar al resto de tareas de forma más óptima mientras se espera en ésta.</p> <p><u>Este bloque tiene menos precisión que el bloque “esperar” original</u>, 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.</p>
	<p>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)</p>
	<p>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.</p> <p>Un mal ajuste puede provocar reinicios del microcontrolador o mal funcionamiento.</p>
	<p>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.</p>

*Ejemplo: si no quiero usar el bucle principal:*



*Si implementamos todo nuestro programa en tareas y no usamos la tarea "principal" de Arduino, es decir el "bucle" principal, podemos eliminar esa tarea autocreada y así quedarnos sólo con el resto.*

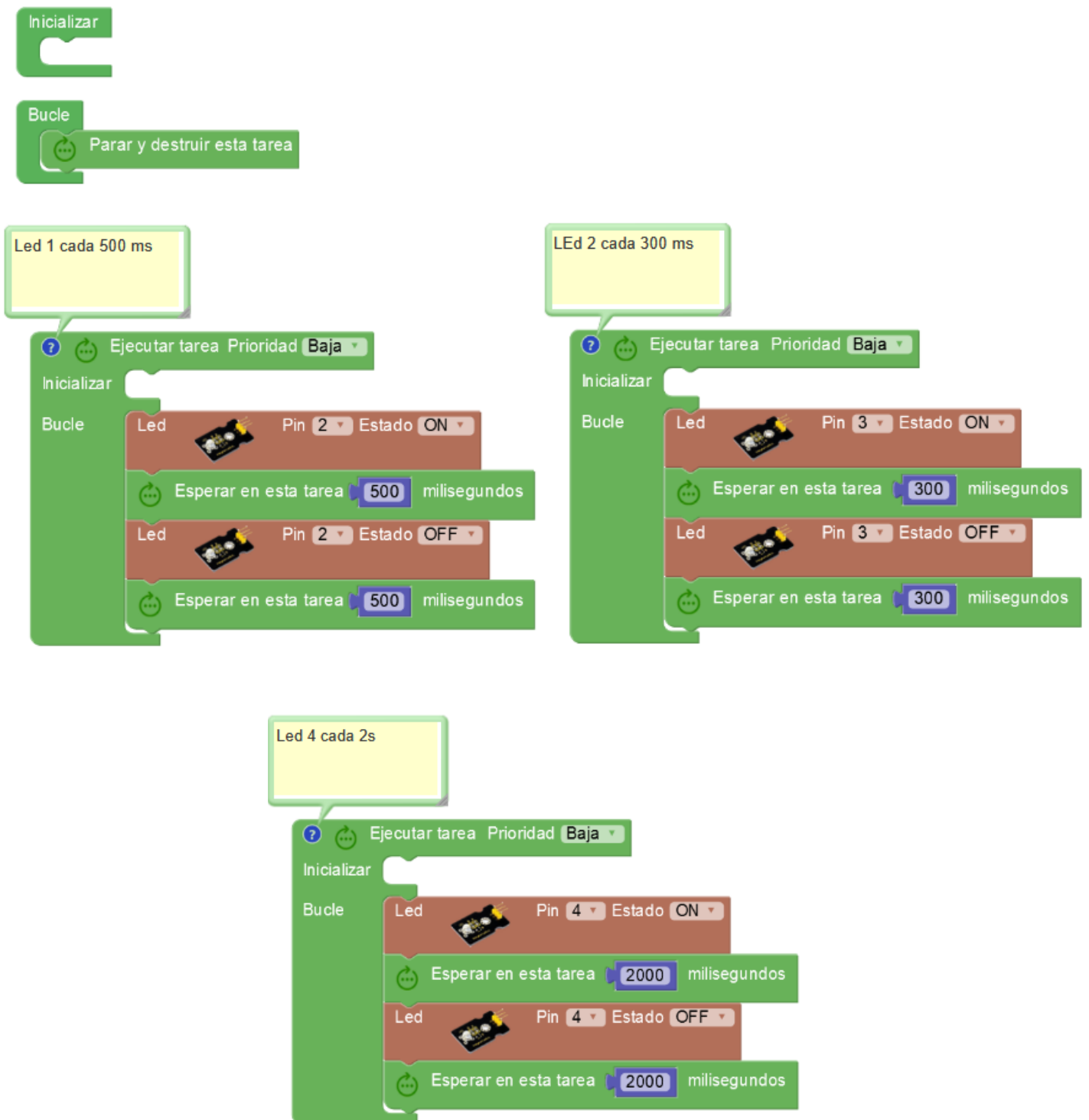
### **IMPORTANTE:**

Puede haber bloques que sean incompatibles con el sistema de multitarea (conflictos en las librerías internas, que utilizan los mismos recursos). En ese caso al utilizar los bloques correspondientes el sistema podría dejar de funcionar o funcionar de forma inestable.

#### Bloques detectados con conflictos:

- Sensor DHT11 Temp/Humedad

## Ejemplo 1: Parpadear varios leds a distintas velocidades



## Ejemplo 2: Parpadear led + Mover servo





### Ejemplo 3: Melodía RTTTL + Pantalla LCD + Sensor de luz LDR

Tarea 1: Muestra el valor de la LDR en la LCD y el nombre de la canción que suena.

Tarea 2: Va reproduciendo distintas canciones y actualiza el nombre en la variable.

```
Inicializar
  LCD Iniciar (I2C)
    ArduinoBlocks
    LCD I2C
    2x16
    ADDR 0x27
```

```
Bucle
  Parar y destruir esta tarea
```

```
Ejecutar tarea Prioridad Media
Inicializar
Bucle
  Establecer nivel de luz = Nivel de luz (LDR) Pin A0 %
  Bloqueo exclusivo para esta tarea (mutex)
  LCD Limpiar
  LCD Imprimir Columna 0 Fila 0 "Luz"
  LCD Imprimir Columna 6 Fila 0 nivel de luz
  LCD Imprimir Columna 0 Fila 1 nombre cancion
  Esperar en esta tarea 1000 milisegundos
```

```
Ejecutar tarea Prioridad Media
Inicializar
Bucle
  Establecer nombre cancion = " Simpsons "
  Zumbador Pin 8 Reproducir RTTTL RTTTL The Simpsons
  Esperar en esta tarea 1000 milisegundos
  Establecer nombre cancion = " Indy "
  Zumbador Pin 8 Reproducir RTTTL RTTTL Indiana Jones
  Esperar en esta tarea 1000 milisegundos
```