[Using FreeRTOS multi-tasking in Arduino](https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc)[© GPL3+](http://opensource.org/licenses/GPL-3.0)

FreeRTOS, simple, easy to use, robust, optimised for Arduino IDE. Full access to FreeRTOS capabilities, within classic Arduino environment.

ABOUT THIS PROJECT

The Arduino IDE and environment has many drivers and libraries available within an arms reach, but the Arduino environment is limited to just setup() and loop()and doesn't support *multi-tasking* effectively.

This is a simple, easy to use and robust FreeRTOS implementation that can just shim into the Arduino IDE as a Library and allow the use of the best parts of both environments, seamlessly.

**Background**

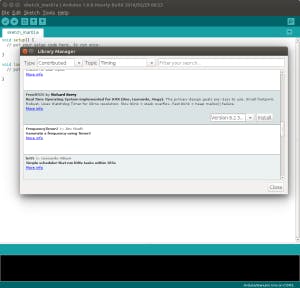
Most operating systems appear to allow multiple programs or threads to execute at the same time. This is called *multi-tasking*. In reality, each processor core can only be running a single program at any given point in time. A part of the operating system called the *scheduler*is responsible for deciding which program to run when, and provides the illusion of simultaneous execution by rapidly switching between each program.

The scheduler in a Real Time Operating System (RTOS) is designed to provide a predictable (normally described as *deterministic*) execution pattern. This is particularly interesting for embedded systems, like the Arduino devices, as embedded systems often have real time requirements.

Traditional real time schedulers, such as the scheduler used in [FreeRTOS](http://www.freertos.org/RTOS.html), achieve determinism by allowing the user to assign a *priority* to each thread of execution. The scheduler then uses the priority to know which thread of execution to run next. In FreeRTOS, a thread of execution is called a *Task*.

**Let's get started**

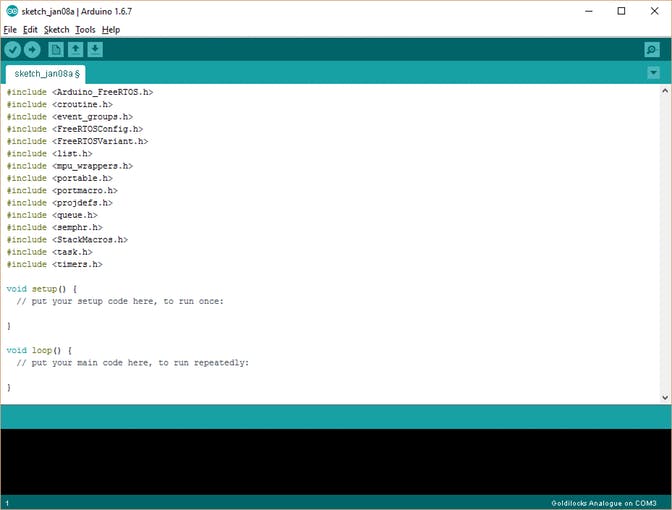
Firstly in the Arduino IDE Library manager, from Version 1.6.8, look for the [FreeRTOS Library](https://github.com/feilipu/Arduino_FreeRTOS_Library) under the Type: “Contributed” and the Topic: “Timing”.

[](javascript:openLightBox('ac28900afb',%200);)Searching in Arduino Library Manager

Ensure that the most recent FreeRTOS Library Release is installed. As of writing that is v10.2.0-1.

[](javascript:openLightBox('8631061492',%200);)FreeRTOS v8.2.3 Release 6 installed.

Then under the Sketch->Include Library menu, ensure that the FreeRTOS library is included in your sketch. A new empty sketch will look like this.

[](javascript:openLightBox('ba0a22c7b8',%200);)Empty Sketch with FreeRTOS Included.

Compile and upload this empty sketch to your Arduino Uno/Yun/Leonardo/Mega or Goldilocks 1284p device. This will show you how much of your flash is consumed by the FreeRTOS scheduler. As a guide the following information was compiled using Arduino v1.6.9 on Windows 10.

// Device:   loop() -> FreeRTOS | Additional Program Storage

// Uno:         444 ->   7018   |     20%

// Goldilocks:  502 ->   7086   |      5%

// Leonardo:   3618 ->  10166   |     23%

// Yun:        3612 ->  10160   |     23%

// Mega:        656 ->  7086  |      2%

At this stage FreeRTOS is already running on your device.

**Next steps**

Now upload and test the Blink sketch, with an underlying Real-Time Operating System simply by inserting #include <Arduino\_FreeRTOS.h> at the start of the sketch. That’s all there is to having FreeRTOS running in your sketches.

The next step is to explore the features provided by a professional RTOS within the Arduino IDE.

Blink\_AnalogRead.ino is a good way to start off as it combines two basic Arduino examples, Blink and AnalogRead into one sketch with two separate Tasks. Both Tasks perform their duties, managed by the FreeRTOS scheduler. This sketch can be found in the Examples Folder of the Arduino IDE.

If you're interested in [low power or battery applications](https://www.hackster.io/feilipu/battery-powered-arduino-applications-through-freertos-3b7401), it is easy to use FreeRTOS to support the AVR ATmega power reduction modes.

Later articles will cover the [use of Semaphores](https://www.hackster.io/feilipu/using-freertos-semaphores-in-arduino-ide-b3cd6c) to protect hardware resources (like the Serial port), Queues to transfer data between Tasks, or Timers to manage exact delays and timeouts. The [FreeRTOS Getting Started](http://www.freertos.org/FreeRTOS-quick-start-guide.html) page has lots of further information on how to use the advanced features, and provides [Demonstration Examples](http://www.freertos.org/a00102.html). Also many more [AVR FreeRTOS example applications](https://github.com/feilipu/avrfreertos) are available, that can be simply converted to use the Arduino environment.

CODE

Blink\_AnalogRead.inoArduino

These are two basic sketches from the Arduino IDE Built-in Examples, combined into one multi-tasking sketch using two FreeRTOS Tasks.

#include <Arduino\_FreeRTOS.h>

// define two tasks for Blink & AnalogRead

void TaskBlink( void \*pvParameters );

void TaskAnalogRead( void \*pvParameters );

// the setup function runs once when you press reset or power the board

void setup() {

// Now set up two tasks to run independently.

xTaskCreate(

TaskBlink

, (const portCHAR \*)"Blink" // A name just for humans

, 128 // Stack size

, NULL

, 2 // priority

, NULL );

xTaskCreate(

TaskAnalogRead

, (const portCHAR \*) "AnalogRead"

, 128 // This stack size can be checked & adjusted by reading Highwater

, NULL

, 1 // priority

, NULL );

// Now the task scheduler, which takes over control of scheduling individual tasks, is automatically started.

}

void loop()

{

// Empty. Things are done in Tasks.

}

/\*--------------------------------------------------\*/

/\*---------------------- Tasks ---------------------\*/

/\*--------------------------------------------------\*/

void TaskBlink(void \*pvParameters) // This is a task.

{

(void) pvParameters;

// initialize digital pin 13 as an output.

pinMode(13, OUTPUT);

for (;;) // A Task shall never return or exit.

{

digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)

vTaskDelay( 1000 / portTICK\_PERIOD\_MS ); // wait for one second

digitalWrite(13, LOW); // turn the LED off by making the voltage LOW

vTaskDelay( 1000 / portTICK\_PERIOD\_MS ); // wait for one second

}

}

void TaskAnalogRead(void \*pvParameters) // This is a task.

{

(void) pvParameters;

// initialize serial communication at 9600 bits per second:

Serial.begin(9600);

for (;;)

{

// read the input on analog pin 0:

int sensorValue = analogRead(A0);

// print out the value you read:

Serial.println(sensorValue);

vTaskDelay(1); // one tick delay (15ms) in between reads for stability

}

}