Systemy operacyjne czasu rzeczywistego

freeRTOS – System czasu rzeczywistego dla systemów wbudowanych – zadania, mutex'y, szeregowanie zadań, kolejki, timery.

Paweł Strączyński
pstraczynski@tu.kielce.pl
Katedra Urządzeń Elektrycznych i Automatyki

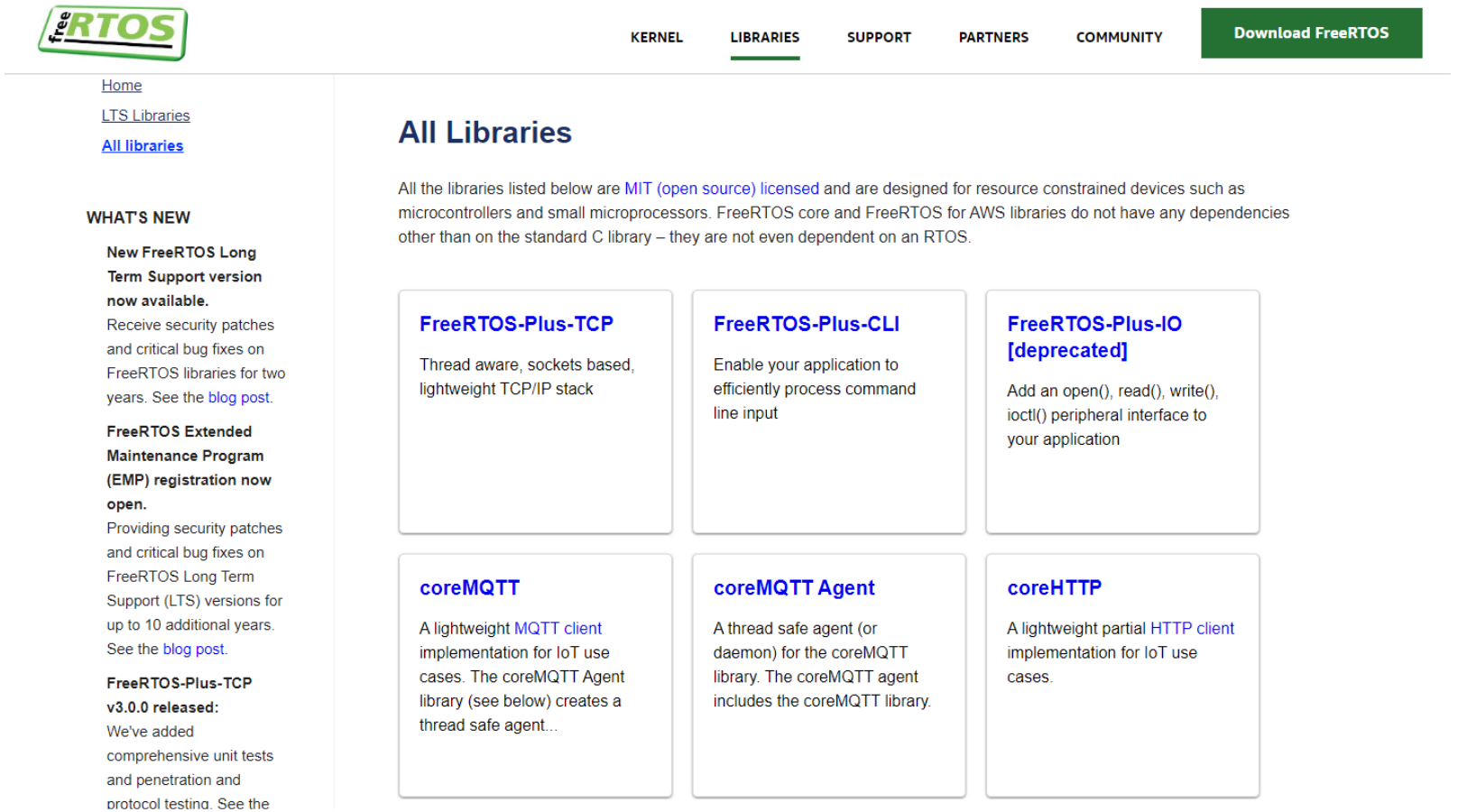
freeRTOS

freeRTOS jest systemem operacyjnym czasu rzeczywistego (ang. real time operating system) dla systemów wbudowanych opartych o mikrokontrolery. Został zaprojektowany z myślą o jak najprostszym i jak najmniejszym kodzie źródłowym.

System freeRTOS wspiera wiele architektur sprzętowych m.in.:

- Altera - Cyclone V SoC (ARM Cortex-A9), Nios II
- Atmel - SAMV7 (ARM Cortex-M7), SAM3 (ARM Cortex-M3), SAM4 (ARM Cortex-M4), SAMD20 (ARM Cortex-M0+), SAMA5 (ARM Cortex-A5), SAM7 (ARM7), SAM9 (ARM9), AT91, AVR and AVR32 UC3
- Microchip - PIC32MX, PIC32MZ, PIC32MZ EF, PIC24, PIC24EP, dsPIC, MEC14xx, CEC13xx, CEC17xx, MEC17xx, MEC51xx
- Cypress - PSoC 5 ARM Cortex-M3
- TI - RM48, TMS570, ARM Cortex-M4F MSP432, MSP430, MSP430X, SimpleLink, Stellaris (ARM Cortex-M3, ARM Cortex-M4F)
- I wiele innych…

freeRTOS


KERNEL
LIBRARIES
SUPPORT
PARTNERS
COMMUNITY
Download FreeRTOS
Home
LTS Libraries
All libraries
WHAT'S NEW
New FreeRTOS Long Term Support version now available.
Receive security patches and critical bug fixes on FreeRTOS libraries for two years. See the blog post.
FreeRTOS Extended Maintenance Program (EMP) registration now open.
Providing security patches and critical bug fixes on FreeRTOS Long Term Support (LTS) versions for up to 10 additional years. See the blog post.
FreeRTOS-Plus-TCP v3.0.0 released:
We've added comprehensive unit tests and penetration and protocol testing. See the
All Libraries
All the libraries listed below are MIT (open source) licensed and are designed for resource constrained devices such as microcontrollers and small microprocessors. FreeRTOS core and FreeRTOS for AWS libraries do not have any dependencies other than on the standard C library – they are not even dependent on an RTOS.
FreeRTOS-Plus-TCP
Thread aware, sockets based, lightweight TCP/IP stack
FreeRTOS-Plus-CLI
Enable your application to efficiently process command line input
FreeRTOS-Plus-IO [deprecated]
Add an open(), read(), write(), ioctl() peripheral interface to your application
coreMQTT
A lightweight MQTT client implementation for IoT use cases. The coreMQTT Agent library (see below) creates a thread safe agent...
coreMQTT Agent
A thread safe agent (or daemon) for the coreMQTT library. The coreMQTT agent includes the coreMQTT library.
coreHTTP
A lightweight partial HTTP client implementation for IoT use cases.

freeRTOS

AWS IoT Fleet Provisioning

Provision new IoT devices without device certificates.

AWS Signature Version 4

Generate a signature and authorization header that complies with the AWS Signature Version 4...

LoRaWAN

A buildable project and documentation article demonstrating the use of LoRaWAN with FreeRTOS.

FreeRTOS-Plus-POSIX

POSIX threading wrapper for the FreeRTOS kernel's native API. Implements a subset of the POSIX threading API.

FreeRTOS-Plus-FAT

Thread aware FAT file system — with optional long file names, caching, and directory name hashing.

FreeRTOS-Plus-TCP IPv6

A git branch that is adding IPv6 support into FreeRTOS-Plus-TCP.

FreeRTOS-Plus-TCP Multiple Interfaces

A git branch that is adding support for multiple interfaces and multiple endpoints into FreeRTOS-Plus-TCP.

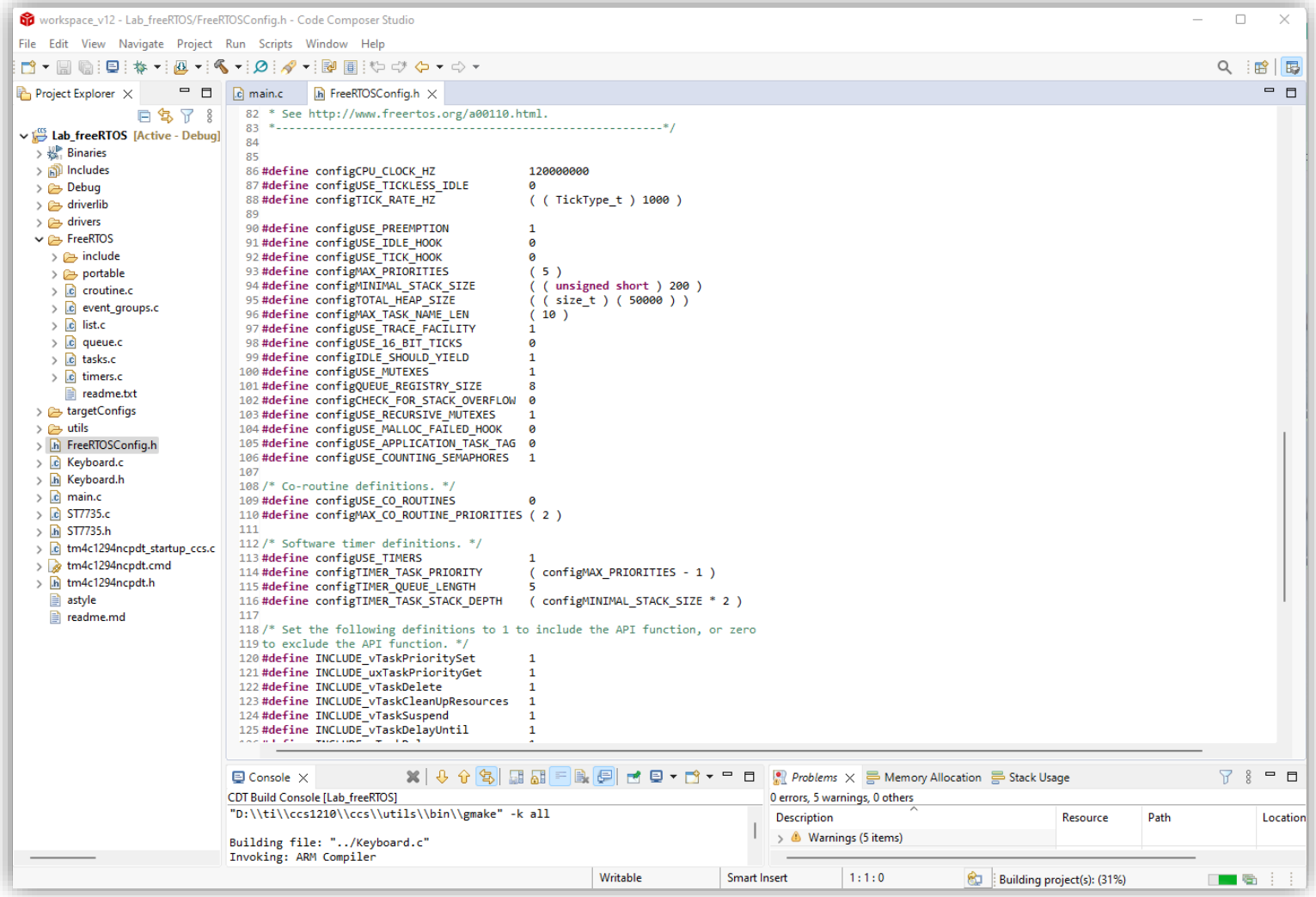
FreeRTOS MCUBoot

MCUBoot is a configurable secure bootloader maintained by several industry leaders, with support for cryptographic verification of software images.

Delta Over-the-Air Updates

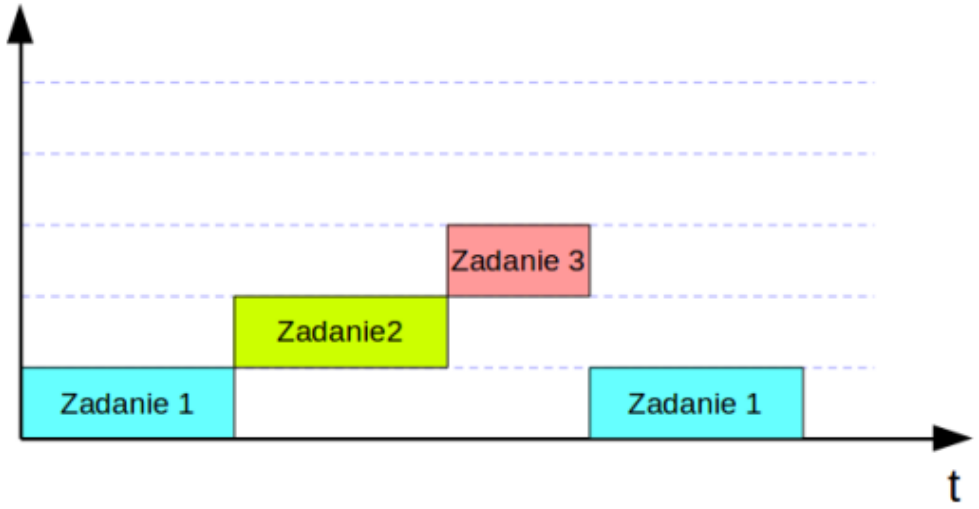
Delta Over-the-Air Updates reduce the size of the OTA update by sending only the binary difference...

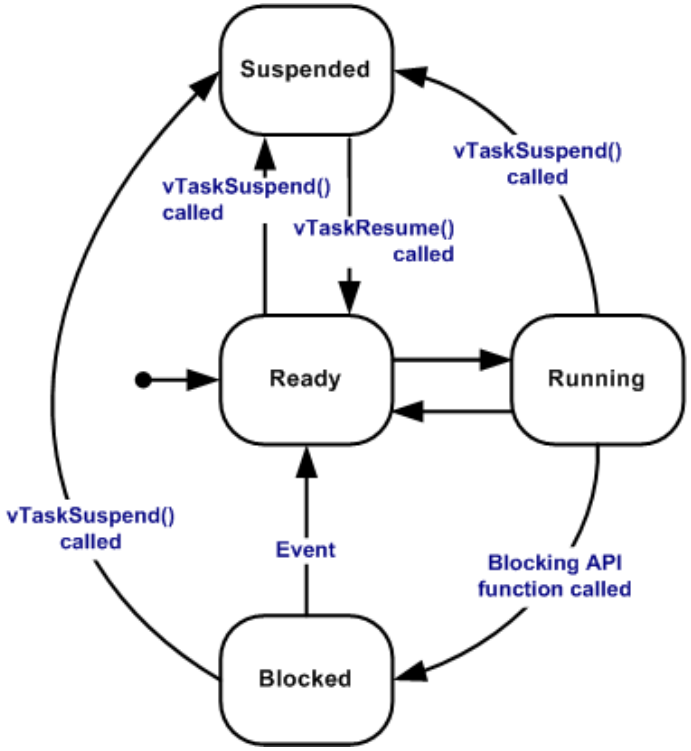
freeRTOS – Texas Instruments Tiva C Series



freeRTOS – priorytety i stany zadań

Domyślnie FreeRTOS stosuje politykę planowania z wywłaszczaniem o stałym priorytecie, z cyklicznym podziałem czasu zadań o równym priorytecie





Każde zadanie ma przypisany priorytet od 0 do (configMAX_PRIORITIES - 1), gdzie configMAX_PRIORITIES jest zdefiniowany w FreeRTOSConfig.h
Niska liczby oznaczają zadania o niskim priorytecie. Bezczynne zadanie ma priorytet zero (tskIDLE_PRIORITY).

freeRTOS – tworzenie i obsługa zadań

```
xTaskCreate(vTaskFunction, "Task1", 128, 1000, 1, NULL);
```

```
BaseType_t xTaskCreate(
    TaskFunction_t pvTaskCode,
    const char * const pcName,
    const configSTACK_DEPTH_TYPE usStackDepth,
    void * const pvParameters,
    UBaseType_t uxPriority,
    TaskHandle_t * const pxCreatedTask
);
```

```
void vTaskFunction( void *pvParameters )
{
        for( ;; )
        {
            -- Task application code here. --
        }
        /* Tasks must not attempt to return from their implementing
        function or otherwise exit.  In newer FreeRTOS port
        attempting to do so will result in an configASSERT() being
        called if it is defined.  If it is necessary for a task to
        exit then have the task call vTaskDelete( NULL ) to ensure
        its exit is clean. */

        vTaskDelete( NULL );
    }
```

pvTaskCode
Wskaźnik na funkcję implementującą kod zadania. Funkcja ta musi być zgodna z prototypem. Argument pvParameters pozwala przekazać dane wejściowe do zadania.

pcName
Nazwa zadania w postaci ciągu znaków (string). Jest to opcjonalne, głównie wykorzystywane do debugowania.

usStackDepth
Liczba słów (ang. words) na stos przydzielony dla zadania. Wartość ta zależy od architektury procesora i powinna być wystarczająca do obsługi zmiennych lokalnych i wywołań funkcji w zadaniu.

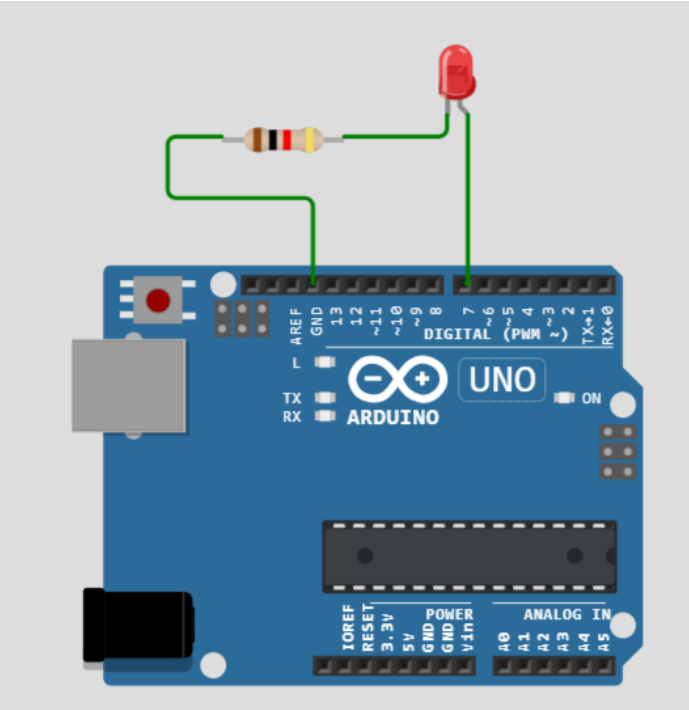
pvParameters
Wskaźnik na dane, które zostaną przekazane do funkcji zadania w momencie jego uruchomienia.

uxPriority
Priorytet zadania. Wyższa wartość oznacza wyższy priorytet.

pxCreatedTask
Wskaźnik, w którym zostanie zapisany uchwyt do nowo utworzonego zadania. Jeśli uchwyt nie jest potrzebny, można podać wartość NULL.

- API Reference
  - Task Creation
    - TaskHandle_t (type)
    - xTaskCreate()
    - xTaskCreateStatic()
    - xTaskCreateRestrictedStatic()
    - vTaskDelete()
  - Task Control
    - vTaskDelay()
    - vTaskDelayUntil()
    - xTaskDelayUntil()
    - uxTaskPriorityGet()
    - vTaskPrioritySet()
    - vTaskSuspend()
    - vTaskResume()
    - xTaskResumeFromISR()
    - xTaskAbortDelay()

freeRTOS dla Arduino - zadania

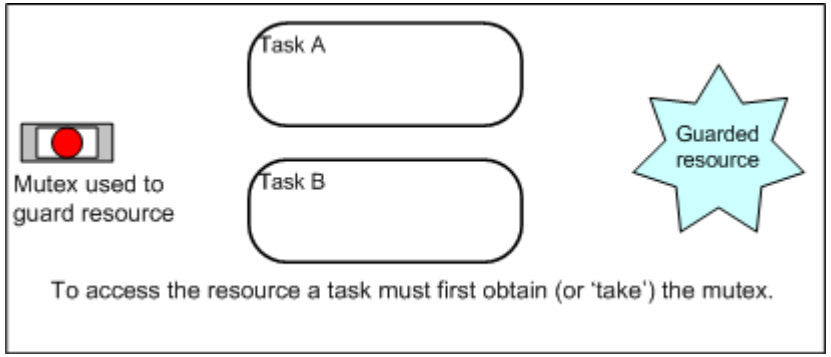


[Link do przykładu]

```
 #include <Arduino_FreeRTOS.h>
 #define LED 7

int i;

 void setup() {
    Serial.begin(9600);
    xTaskCreate(ledTask, "Led Task", 128, NULL, 1, NULL);
    xTaskCreate(consoleTask, "Console Task", 128, NULL, 1, NULL);
 }

 void loop() {}


void ledTask(void *pvParameters){
    pinMode(LED, OUTPUT);
    while(1){
      digitalWrite(LED, HIGH);
      delay(500);
      digitalWrite(LED, LOW);
      delay(500);
    }
}

void consoleTask(void *pvParameters){
    while(1){
      Serial.println(i);
      i++;
    }
}
```

freeRTOS dla Arduino – semafory i mutexy

```
SemaphoreHandle_t xSemaphoreCreateMutex( void );

SemaphoreHandle_t xSemaphoreCreateCounting( UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);

xSemaphoreTake( SemaphoreHandle_t xSemaphore,  TickType_t xTicksToWait );

xSemaphoreGive( SemaphoreHandle_t xSemaphore );
```



- Semaphore / Mutexes
  - xSemaphoreCreateBinary()
  - xSemaphoreCreateBinaryStatic()
  - vSemaphoreCreateBinary()
  - xSemaphoreCreateCounting()
  - xSemaphoreCreateCountingStatic()
  - xSemaphoreCreateMutex()
  - xSemaphoreCreateMutexStatic()
  - xSem'CreateRecursiveMutex()
  - xSem'CreateRecursiveMutexStatic()
  - vSemaphoreDelete()
  - xSemaphoreGetMutexHolder()
  - uxSemaphoreGetCount()
  - xSemaphoreTake()
  - xSemaphoreTakeFromISR()
  - xSemaphoreTakeRecursive()
  - xSemaphoreGive()
  - xSemaphoreGiveRecursive()
  - xSemaphoreGiveFromISR()

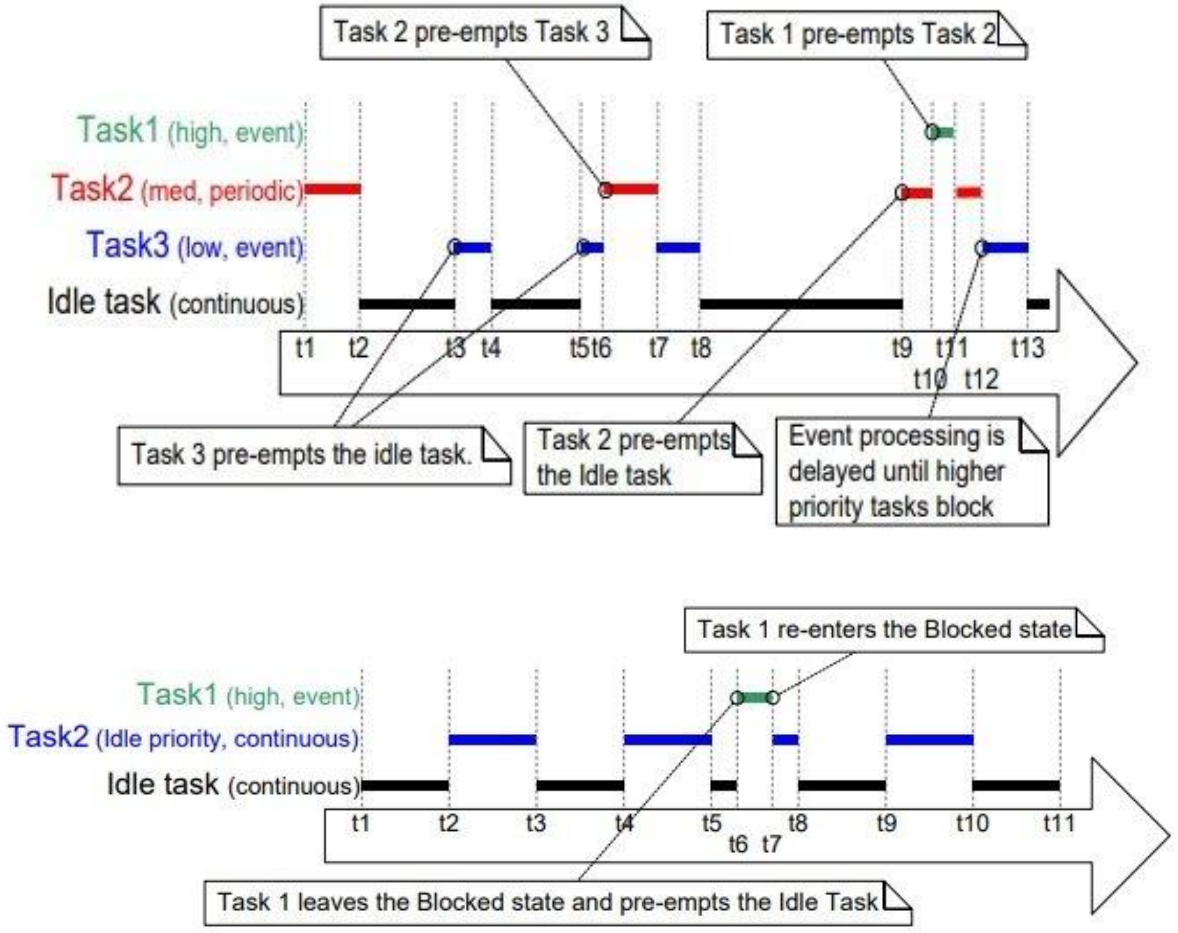
freeRTOS dla Arduino - mutexy

```ino
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

SemaphoreHandle_t mutex;

int globalCount;

void setup() {
    Serial.begin(9600);
    mutex = xSemaphoreCreateMutex();
    if (mutex != NULL) {
        Serial.println("Mutex created");
    }
    xTaskCreate(TaskMutex, "Task1", 128, 1000, 1, NULL);
    xTaskCreate(TaskMutex, "Task2", 128, 1000, 1, NULL);
}

void loop() {}
```

```ino
void TaskMutex(void *pvParameters)
{
    TickType_t delayTime = *((TickType_t*)pvParameters);
    while(1)
    {
        if (xSemaphoreTake(mutex, 10) == pdTRUE)
        {
            Serial.print(pcTaskGetName(NULL));
            Serial.print(", Count read value: ");
            Serial.print(globalCount);

            globalCount++;

            Serial.print(", Updated value: ");
            Serial.print(globalCount);

            Serial.println();
            xSemaphoreGive(mutex);
        }

    vTaskDelay(delayTime / portTICK_PERIOD_MS);
  }
}
```

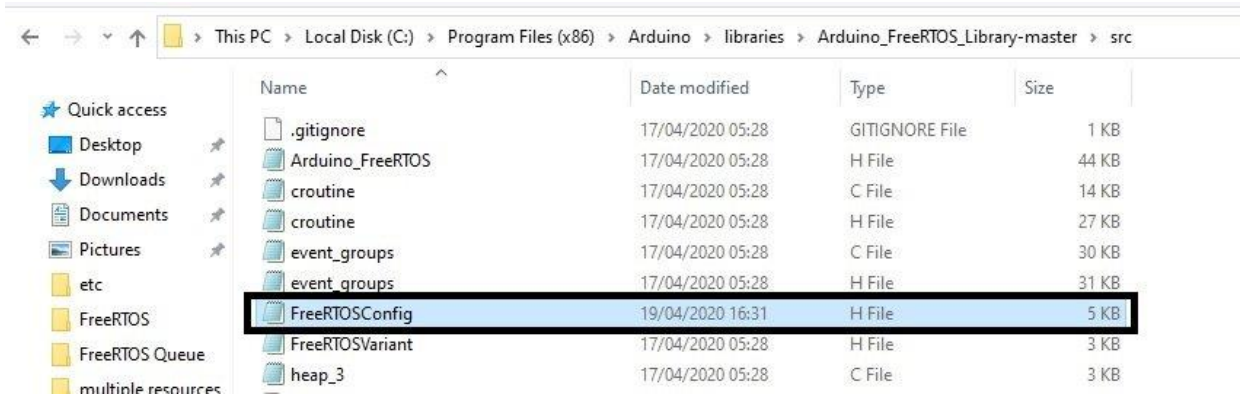
Link do przykładu

freeRTOS dla Arduino – szeregowanie zadań



Źródło: https://microcontrollerslab.com/

freeRTOS dla Arduino – szeregowanie zadań




```
// And on to the things the same no matter the AVR type...
#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     1
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( ( uint32_t ) F_CPU )
#define configMAX_PRIORITIES                    4
#define configMINIMAL_STACK_SIZE                ( 192 )
#define configMAX_TASK_NAME_LEN                 ( 8 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  1
#define configIDLE_SHOULD_YIELD                 1

#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_QUEUE_SETS                    1
#define configQUEUE_REGISTRY_SIZE               1
#define configUSE_TIME_SLICING                  1
#define configCHECK_FOR_STACK_OVERFLOW          1
#define configUSE_MALLOC_FAILED_HOOK            1
```

Źródło: https://microcontrollerslab.com/

freeRTOS dla Arduino – konfiguracja systemu

Ogólne parametry konfiguracji FreeRTOS:

configUSE_PREEMPTION
Włącza (1) lub wyłącza (0) wywłaszczanie. Wywłaszczanie pozwala zadaniom o wyższym priorytecie przerywać zadania o niższym priorytecie.

configUSE_IDLE_HOOK
Włącza (1) możliwość dodania funkcji hook, która jest wywoływana w pętli "bezczynności" systemu.

configUSE_TICK_HOOK
Włącza (1) funkcję hook, która jest wykonywana przy każdym przerwaniu zegarowym (tick interrupt).

configCPU_CLOCK_HZ
Częstotliwość zegara procesora w Hz.
Parametr używany do obliczania interwałów przerwań zegarowych.

configMAX_PRIORITIES
Maksymalna liczba priorytetów dostępnych w systemie (np. 4 oznacza priorytety od 0 do 3).

configMINIMAL_STACK_SIZE
Minimalny rozmiar stosu w słowach dla zadania (np. 192 oznacza 192 słowa pamięci).

configMAX_TASK_NAME_LEN
Maksymalna długość nazwy zadania w znakach (np. 8).

configUSE_TRACE_FACILITY
Włącza (1) funkcje umożliwiające śledzenie (trace) zadań, np. do debugowania.

configUSE_16_BIT_TICKS
Określa rozmiar zmiennej przechowującej ticki systemowe:1: 16-bitowe (mniejszy zakres, mniejsze zużycie pamięci),0: 32-bitowe (większy zakres, ale większe zużycie pamięci).

configIDLE_SHOULD_YIELD
Określa, czy zadanie bezczynności (Idle Task) powinno oddawać czas procesora (1) innym zadaniom o tym samym priorytecie.Parametry dotyczące synchronizacji i kolejek:

freeRTOS dla Arduino – konfiguracja systemu

Parametry dotyczące synchronizacji i kolejek oraz te związane z czasem i bezpieczeństwem:

configUSE_MUTEXES
Włącza (1) wsparcie dla mutexów.

configUSE_RECURSIVE_MUTEXES
Włącza (1) wsparcie dla rekurencyjnych mutexów.

configUSE_COUNTING_SEMAPHORES
Włącza (1) liczniki semaforów.

configUSE_QUEUE_SETS
Włącza (1) funkcje związane z zestawami kolejek (queue sets).

configQUEUE_REGISTRY_SIZE
Liczba kolejek, które mogą być zarejestrowane do debugowania (np. 1).Parametry związane z czasem i bezpieczeństwem:

configUSE_TIME_SLICING
Włącza (1) mechanizm podziału czasu między zadania o równym priorytecie.

configCHECK_FOR_STACK_OVERFLOW
Włącza (1) funkcje sprawdzania przepełnienia stosu.

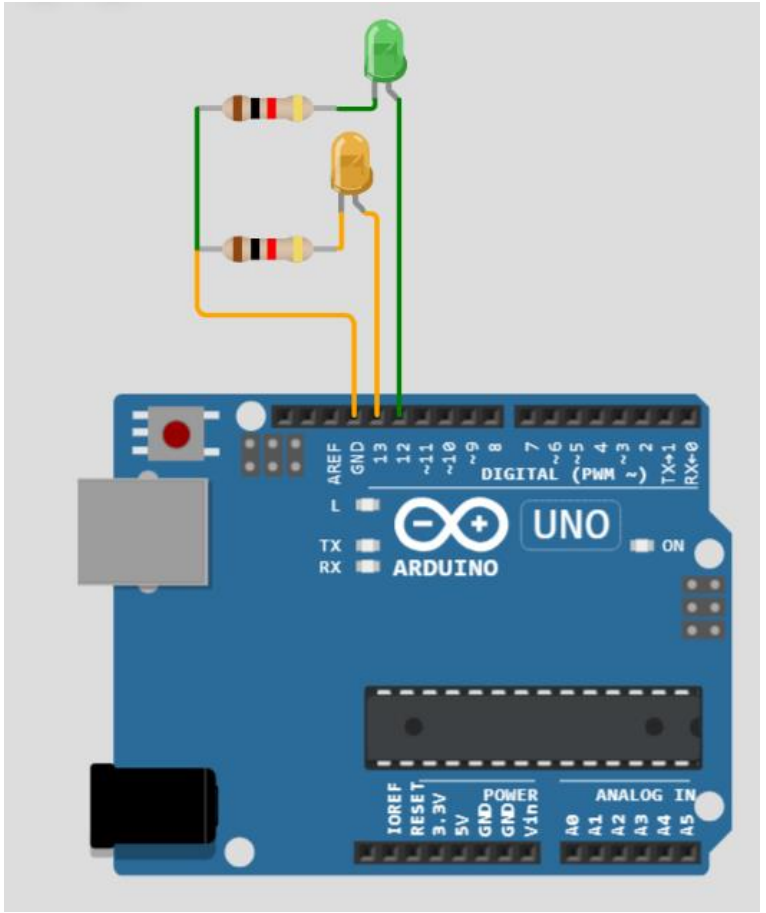
configUSE_MALLOC_FAILED_HOOK
Włącza (1) funkcję hook, która jest wywoływana, gdy malloc zwraca błąd (brak pamięci).

freeRTOS dla Arduino – ustawianie priorytetu

```
1  /*
2   * - Ustawia nowy priorytet dla wskazanego zadania.
3   * - Parametry:
4   *    - pxTask: Uchwyt do zadania, którego priorytet chcemy zmienić.
5   *              Jeśli NULL, priorytet zostanie zmieniony dla zadania wywołującego.
6   *    - uxNewPriority: Nowy priorytet dla zadania (0 - configMAX_PRIORITIES - 1).
7   */
8  void vTaskPrioritySet( TaskHandle_t pxTask, UBaseType_t uxNewPriority );
9
10 /*
11  *
12  * - Zwraca bieżący priorytet wskazanego zadania.
13  * - Parametry:
14  *    - pxTask: Uchwyt do zadania, którego priorytet chcemy odczytać.
15  *              Jeśli NULL, zwracany jest priorytet zadania wywołującego.
16  * - Zwracana wartość: Aktualny priorytet zadania (typu UBaseType_t).
17  */
18 UBaseType_t uxTaskPriorityGet( TaskHandle_t pxTask );
19
```

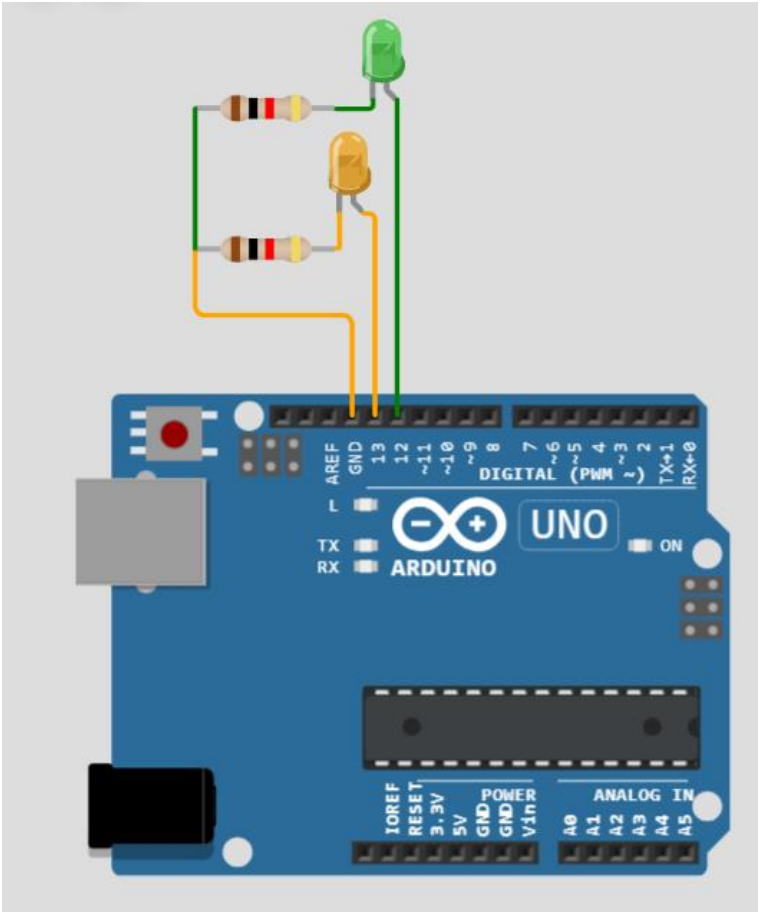
freeRTOS dla Arduino – szeregowanie zadań

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#define LED1 13
#define LED2 12



void Task1( void *pvParameters );
void Task2( void *pvParameters );

TaskHandle_t TaskHandle_1;
TaskHandle_t TaskHandle_2;

void setup()
{
  Serial.begin(9600);

   xTaskCreate(Task1, "LED1", 400, NULL, 3, &TaskHandle_1);
   xTaskCreate(Task2, "LED2", 400, NULL, 2, &TaskHandle_2);
   vTaskStartScheduler();
}

void loop() {}
```



freeRTOS dla Arduino – szeregowanie zadań

```
void Task1(void* pvParameters)
{
    pinMode(LED1, OUTPUT);
    UBaseType_t uxPriority = uxTaskPriorityGet( NULL );
    while(1)
    {
      if (digitalRead(LED1)) digitalWrite(LED1, LOW);
      else digitalWrite(LED1, HIGH);
      Serial.println("Task1 is running and about to raise Task2 Priority");
      vTaskPrioritySet( TaskHandle_2, ( uxPriority + 1 ) );

    }
}
void Task2(void* pvParameters)
{
    pinMode(LED2, OUTPUT);
    UBaseType_t   uxPriority = uxTaskPriorityGet( NULL );
    while(1)
    {
      if (digitalRead(LED2)) digitalWrite(LED2, LOW);
      else digitalWrite(LED2, HIGH);
      Serial.println("Task2 is running and about to lower Task2 Priority");
      vTaskPrioritySet( TaskHandle_2, ( uxPriority - 2 ) );

    }
}
```

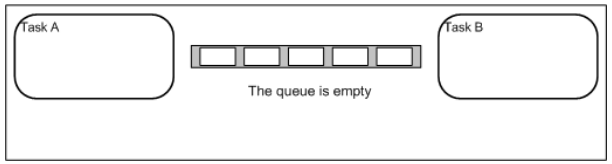


[Link do przykładu]

freeRTOS dla Arduino - kolejki

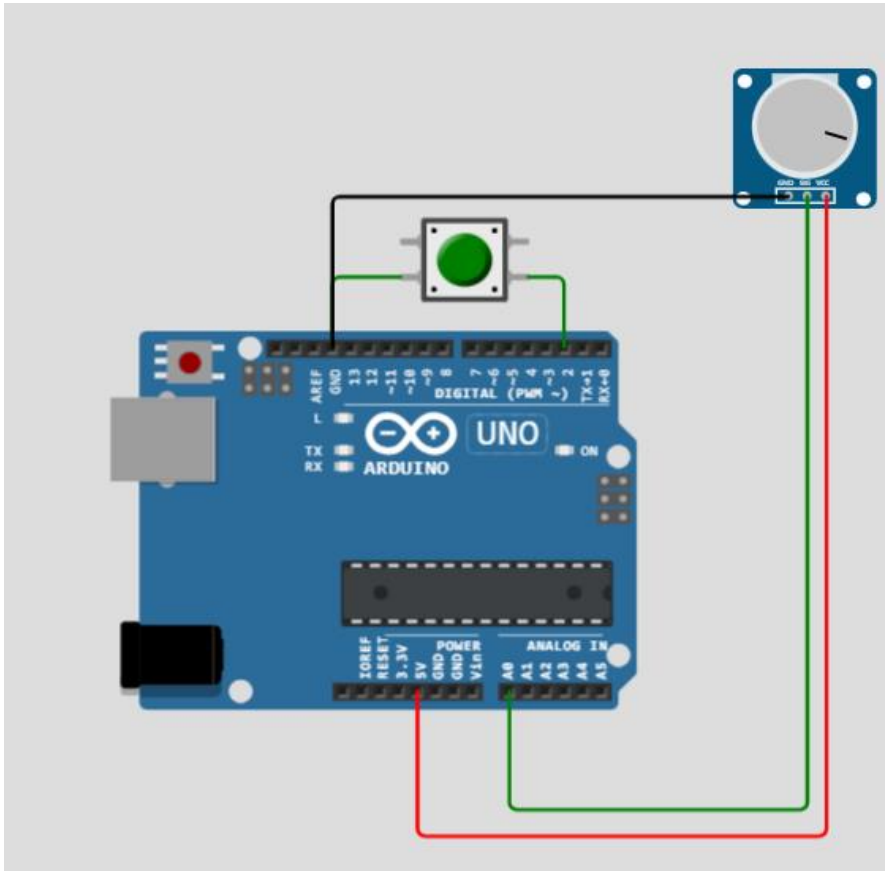
```c
/*
 * - Tworzy nową kolejkę.
 * - Parametry:
 *    - uxQueueLength: Maksymalna liczba elementów, które kolejka może przechowywać jednocześnie.
 *    - uxItemSize: Rozmiar w bajtach wymagana do przechowywania każdego elementu w kolejce.
 * - Zwracana wartość: Uchwyt do nowo utworzonej kolejki (typu QueueHandle_t).
 */
QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength, UBaseType_t uxItemSize);

/*
 * - Wysyła dane do kolejki.
 * - Parametry:
 *    - xQueue: Uchwyt do kolejki, do której element ma zostać wysłany.
 *    - pvItemToQueue: Wskaźnik do elementu, który ma zostać umieszczony w kolejce.
 *    - xTicksToWait: Maksymalny czas, przez jaki zadanie będzie czekać na zwolnienie miejsca w kolejce, jeśli jest pełna.
 * - Zwracana wartość:
 *    - pdTRUE, jeśli dane zostały pomyślnie umieszczone w kolejce.
 *    - pdFALSE, jeśli operacja się nie powiodła.
 */
BaseType_t xQueueSend(QueueHandle_t xQueue, const void *pvItemToQueue, TickType_t xTicksToWait);

/*
 * - Odbiera dane z kolejki.
 * - Parametry:
 *    - xQueue: Uchwyt do kolejki, z której element ma zostać odebrany.
 *    - pvBuffer: Wskaźnik do bufora, do którego zostanie skopiowany odebrany element.
 *    - xTicksToWait: Maksymalny czas, przez jaki zadanie powinno blokować oczekiwanie na dostępność elementu w kolejce.
 * - Zwracana wartość:
 *    - pdTRUE, jeśli dane zostały pomyślnie odebrane.
 *    - pdFALSE, jeśli operacja się nie powiodła.
 */
BaseType_t xQueueReceive(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait);

```

- xQueueCreate
- xQueueCreateStatic
- vQueueDelete
- xQueueSend
- xQueueSendFromISR
- xQueueSendToBack
- xQueueSendToBackFromISR
- xQueueSendToFront
- xQueueSendToFrontFromISR
- xQueueReceive
- xQueueReceiveFromISR
- uxQueueMessagesWaiting
- uxQueueMessagesWaitingFromISR
- uxQueueSpacesAvailable
- xQueueReset
- xQueuePeek
- xQueuePeekFromISR
- vQueueAddToRegistry
- pcQueueGetName
- vQueueUnregisterQueue
- xQueueIsQueueEmptyFromISR
- xQueueIsQueueFullFromISR
- xQueueOverwrite
- xQueueOverwriteFromISR



freeRTOS dla Arduino - kolejki

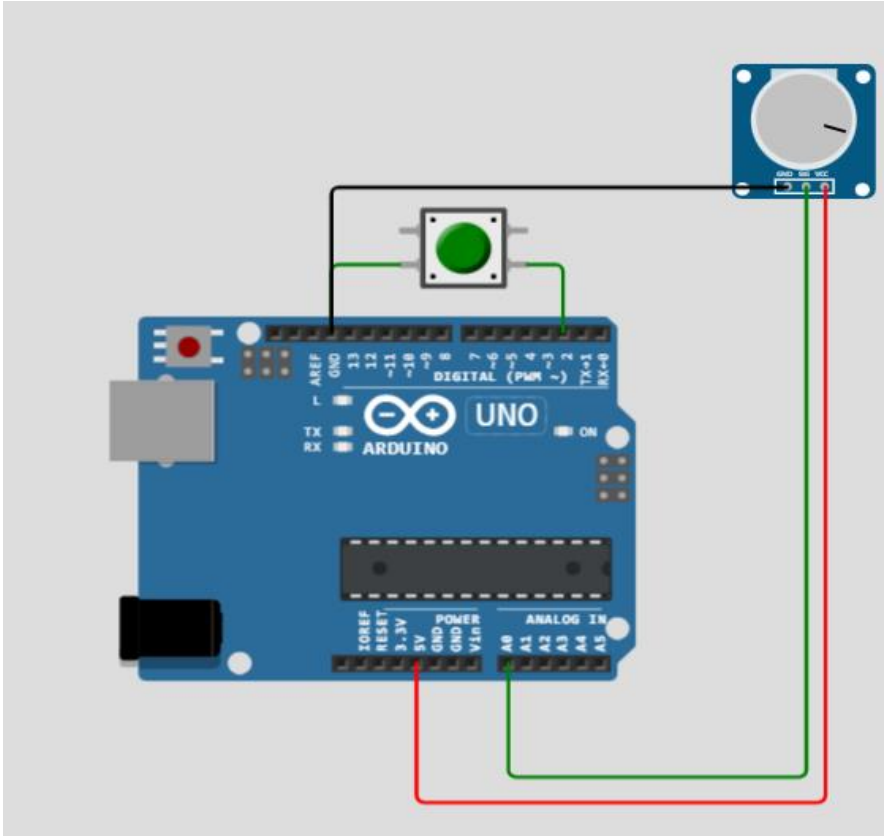
```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

QueueHandle_t serialQueue;

void TaskAnalogRead( void *pvParameters );
void TaskSerialWrite( void *pvParameters );

void setup() {
  Serial.begin(9600);
  while (!Serial) {}
  serialQueue = xQueueCreate(16, sizeof(int));
  if(serialQueue == NULL){
    Serial.println("Blad tworzenia kolejki");
    while(true);
  }
  xTaskCreate( TaskAnalogRead,  "AnalogRead", 128, NULL, 1, NULL);
  xTaskCreate( TaskSerialWrite, "SerialWrite",  128,  NULL, 1,  NULL);
}

void loop(){}
```



[Link do przykładu]

freeRTOS dla Arduino - kolejki

```
void TaskAnalogRead( void *pvParameters )
{
  while(1)
  {
    uint8_t analogIn = A0;
    int sensorValue = analogRead(analogIn);
    xQueueSend(serialQueue, &sensorValue, portMAX_DELAY);
    vTaskDelay(1);
  }
}

void TaskSerialWrite( void *pvParameters
__attribute__((unused)) )
{
  while(1)
  {
    int currentItem;
    while(xQueueReceive(serialQueue, &currentItem,
portMAX_DELAY) == pdTRUE) {
      Serial.println(currentItem);
    }
    vTaskDelay(1);
  }
}
```

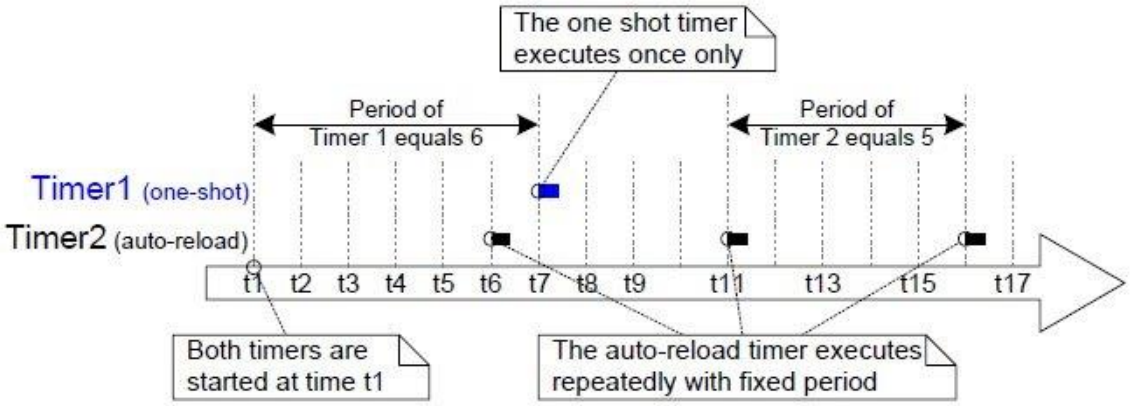


freeRTOS dla Arduino - timer

Timery freeRTOS można skonfigurować w dwóch trybach:
- one-shot,
- auto-reload.

One-shot timer to jednorazowy licznik który wykona funkcje tylko raz. Ponowne uruchomienie należy wyzwolić ręcznie

Auto-reload timer licznik który uruchamia się ponownie automatycznie, cyklicznie wykonując zadaną funkcję.



Źródło: https://microcontrollerslab.com/

freeRTOS dla Arduino - timer

```c
/*
 * - Tworzy nowy timer.
 * - Parametry:
 *    - pcTimerName: Nazwa timera (łańcuch znaków, tylko do celów debugowania).
 *    - xTimerPeriodInTicks: Czas po którym zostanie wywołana funkcja obsługi timera, wyrażony w tickach.
 *    - uxAutoReload: Jeśli ustawione na pdTRUE, timer automatycznie przeładowuje się po wygaśnięciu.
 *    - pvTimerID: Wskaźnik do unikalnego identyfikatora timera (dowolny wskaźnik).
 *    - pxCallbackFunction: Wskaźnik do funkcji wywoływanej po wygaśnięciu timera.
 * - Zwracana wartość: Uchwyt do nowo utworzonego timera (typu TimerHandle_t).
 */
TimerHandle_t xTimerCreate(const char *const pcTimerName,
                           TickType_t xTimerPeriodInTicks,
                           UBaseType_t uxAutoReload,
                           void *pvTimerID,
                           TimerCallbackFunction_t pxCallbackFunction);

/*
 * - Uruchamia timer.
 * - Parametry:
 *    - xTimer: Uchwyt do timera, który ma zostać uruchomiony.
 *    - xTicksToWait: Maksymalny czas oczekiwania w tickach na udostępnienie timera do uruchomienia.
 * - Zwracana wartość:
 *    - pdTRUE, jeśli timer został pomyślnie uruchomiony.
 *    - pdFALSE, jeśli uruchomienie timera się nie powiodło.
 */
BaseType_t xTimerStart(TimerHandle_t xTimer, TickType_t xTicksToWait);

```

freeRTOS dla Arduino - timer

```
#include <Arduino_FreeRTOS.h>
#include <timers.h>
#include <task.h>

#define mainONE_SHOT_TIMER_PERIOD pdMS_TO_TICKS( 3333 )

TimerHandle_t  xOneShotTimer;
BaseType_t xTimer1Started;

void setup()
{
  Serial.begin(9600); // Enable serial communication library.

  xOneShotTimer = xTimerCreate("OneShot", 3333/portTICK_PERIOD_MS,pdFALSE,
NULL, prvOneShotTimerCallback);

  if( xOneShotTimer != NULL )
  {

    xTimer1Started = xTimerStart( xOneShotTimer, 0 );

    if( xTimer1Started == pdPASS )
    {
      vTaskStartScheduler();
    }
  }
}
```

```
void loop() {}

static void prvOneShotTimerCallback( TimerHandle_t xTimer )
{
      TickType_t xTimeNow;
      xTimeNow = xTaskGetTickCount();
      Serial.print("One-shot timer callback executing ");
      Serial.println(xTimeNow);
}
```

Link do przykładu