Catatan OS

1 xTaskCreate

src: wokwi
Q: How do the Task no. X behaves by settings and modifying the Priority if we have the same delay and the same duration

```
// original code
/*
  The original code:
  https://microcontrollerslab.com/use-freertos-arduino/
  Modified by Barbu Vulc!
  January 4th, 2023
*/
//RTOS = Real-time operating system
//FreeRTOS site: https://www.freertos.org/

//FreeRTOS library:
#include <Arduino_FreeRTOS.h>

//Variables for buzzer & first 3 LEDs!
const int buzzer = 7;     //Buzzer
const int LED1 = 8;       //Red LED
const int LED2 = 9;       //Yellow LED
const int LED3 = 10;      //Green LED
/*
 * I named this 'extender' (see below) because I can optionally...
 * ...put a 4th task to this LED! Now is task-free! :))
 */
const int extender = 11; //Blue LED

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

  //LEDs' initialization!
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(extender, OUTPUT);

  //Buzzer initialization!
  pinMode(buzzer, OUTPUT);

  /*
    Create 3 tasks with labels 'Task_1', 'Task_2' and 'Task_3' and
    assign the priority as 1, 2 and 3 respectively.
  */
  //'Neutral_Task' - the task-free function!
  xTaskCreate(Task_1,       "Task␣no.␣␣1!",  100, NULL, 4, NULL);
  xTaskCreate(Task_2,       "Task␣no.␣␣2!",  100, NULL, 2, NULL);
  xTaskCreate(Task_3,       "Task␣no.␣␣3!",  100, NULL, 3, NULL);
  xTaskCreate(Neutral_Task, "Neutral_Task!", 100, NULL, 0, NULL);
}

//The following function is Task1. We display the task label on Serial monitor.
static void Task_1(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(extender, LOW);
    tone(buzzer, 200, 1000 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣1!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

//Task 2
static void Task_2(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, LOW);
    digitalWrite(extender, LOW);
    tone(buzzer, 300, 1000 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣2!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

//Task 3
static void Task_3(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, HIGH);
    digitalWrite(extender, LOW);
    tone(buzzer, 400, 1000 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣3!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

//Task 4 (the last one)!
//This is an extension which can be task-free!
static void Neutral_Task(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(extender, HIGH);
    tone(buzzer, 500, 1000);
    Serial.println(F("Neutral_Task"));
    delay(1000);
  }
}

//We don't need to use "loop" function here!
void loop() {}
```

- Those 2 task may look random when they run but they are controlled under `vTaskDelay` and `xTaskCreate` priority.
- When `vTaskDelay` args for all of the 3 functions changed to be the same (example below)

```
...

  xTaskCreate(Task_1, "Task␣no.␣1!", 100, NULL, 1, NULL);
  xTaskCreate(Task_2, "Task␣no.␣␣2!", 100, NULL, 2, NULL);
  xTaskCreate(Task_3, "Task␣no.␣␣3!", 100, NULL, 3, NULL);
  xTaskCreate(Neutral_Task, "Neutral_Task!", 100, NULL, 0, NULL);

...

static void Task_1(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(extender, LOW);
    tone(buzzer, 200, 1000 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣1!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

//Task 2
static void Task_2(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, LOW);
    digitalWrite(extender, LOW);
    tone(buzzer, 300, 1100 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣2!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

//Task 3
static void Task_3(void* pvParameters) {
  while (1) {
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, HIGH);
    digitalWrite(extender, LOW);
    tone(buzzer, 400, 1200 / portTICK_PERIOD_MS);
    Serial.println(F("Task␣no.␣3!"));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

...
```

- Then the output will be

```
Neutral_Task
Task no. 3!
Task no. 2!
Task no. 1!
```

- When the priority changed to `1` for all the functions (example below)

```
xTaskCreate(Task_1, "Task␣no.␣1!", 100, NULL, 1, NULL);
xTaskCreate(Task_2, "Task␣no.␣␣2!", 100, NULL, 1, NULL);
xTaskCreate(Task_3, "Task␣no.␣␣3!", 100, NULL, 1, NULL);
xTaskCreate(Neutral_Task, "Neutral_Task!", 100, NULL, 0, NULL);
```

- Then the output will be

```
Neutral_Task
Task no. 1!
Task no. 2!
Task no. 3!
```

- From that example we can say that if the `xTaskCreate` priority args set to be bigger then it will have the most priority and get to run first
- priority is `uint8_t` with the alias of `UBaseType_t`
- so any unsigned integer 8 bits are possible to be an args for the `xTaskCreate` function.
- when the priority is `0` it will be an idle task that will be only run when idle (no other work to do).

2 Mutex

src : wokwi

- a Mutex is a var that will lock itself when used by thread x and make thread y to wait for thread x to finish its job.
- type that used for the mutex : `SemaphoreHandle_t`
- there is 2 func that will be used to lock and unlock the var.

  - `xSemaphoreTake();`
  - `xSemaphoreGive();`

- `xSemaphoreTake()` will accept 2 args which is the handler which is the `SemaphoreHandl_t` object and the tick (time to check again).
- `xSemaphoreGive()` just need 1 arg which is the handler.

2.1 Adjust Task Parameter to Produce Conflic

when using the Semaphore or Mutex it will not make the thread conflict each other even if the priority changed.

NOTE : when the priority of one thread is on '0' which is idle it will just sleep forever.

2.2 Remove take and give semaphore to produe conflict

the original code :

```
...

void TaskMutex(void *pvParameters)
{
  TickType_t delayTime = *((TickType_t*)pvParameters);
  // Use task parameters to define delay

  for (;;)
  {
    /**
       Take mutex
       https://www.freertos.org/a00122.html
    */
    if (xSemaphoreTake(mutex, 10) == pdTRUE)
    {
      Serial.print(pcTaskGetName(NULL)); // Get task name
      Serial.print(", Count read value: ");
      Serial.print(globalCount);

      globalCount++;

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

      Serial.println();
      /**
         Give mutex
         https://www.freertos.org/a00123.html
      */
      xSemaphoreGive(mutex);
    }

    vTaskDelay(delayTime / portTICK_PERIOD_MS);
  }
}

...
```

with the output :

```
Mutex created

Task1, Count read value: 0, Updated value: 1

Task2, Count read value: 1, Updated value: 2

Task1, Count read value: 2, Updated value: 3

Task2, Count read value: 3, Updated value: 4
```

Now to make it conflict is by modifying the function to not do ‘xSemaphore-Take‘ and ‘xSemaphoreGive‘ checking or locking that will look like this :

```

...

void TaskMutex(void *pvParameters)
{
    TickType_t delayTime = *((TickType_t*)pvParameters); // Use task parameters to define dela

    for (;;)
    {
        /**
        Take mutex
        https://www.freertos.org/a00122.html
        */
        Serial.print(pcTaskGetName(NULL)); // Get task name
        Serial.print(",␣Count␣read␣value:␣");
        Serial.print(globalCount);

        globalCount++;

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

        Serial.println();

        vTaskDelay(delayTime / portTICK_PERIOD_MS);
    }
    }

    ...
```

output :

```
Mutex created

Task1, Count read value: 0, Updated value: 1

Task2, Count read Talue: 1, Updated value: 2, UpdTted value: 3

ad value: 3, UpdTted value: 4
ad value: 4, UpTated value: 5

d value: 5, UpdTted value: 6

ad value: 6, UpdTted value: 7
```

1. The Bounded-Buffer Problem
    - A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.
    - so if there is 4 threads, 2 is reader and the other 2 is writer and when one of the writer is writing to the buffer and the buffer is full this writer thread need to tell the other writer thread that the buffer is full so it didnt get overflowed. With the reader threads perspective when the buffer is empty its better to tell the other reader thread to not bother reading the buffer because the buffer is empty.
2. The Readers–Writers Problem
    - so the problem is the reader thread is not consisten with other reader thread because one thread access it to fast and the other to slow and so on.
    - Problem arises when balancing the need for simultaneous access by multiple readers against exclusive access for a single writer to ensure data consistency and integrity
3. The Dining-Philosophers Problem
    - a group of philosophers sitting at a table doing one of two things - eating or thinking. While eating, they are not thinking, and while thinking, they are not eating.
    - so let say there is 2 thread that need to communicate to each other and to communicate they need a permission or flag from the other thread first.
    - The problem arise when 1 thread have an error or blocking and did not send the permission or flag so deadlock happen.

3 Multithreading

Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading. With the use of multithreading, multitasking can be achieved.

So Multithreading is when a program will launch more than 1 task that need to be not blocking.

Thread model :

- Many to One
  - the program will only spawn 1 kernel thread and spawn userspace thread whenever needed.
- One to Many
  - the program will create new kernel thread every time the program need a new thread.
- Many to Many
  - program will create as many kernel thread or userspace thread whenever the program need to.

4 Struct Array

src : wokwi

Purpose

The purpose of this demo is so we don't need to define a lot of variables with the type `struct` that we need for certain functions.

Reason for OS

The reason for using an OS is so we don't need to define and create our own scheduler, thread model, mutex, etc.

Example Source Code

```
/*
 * Example of a basic FreeRTOS queue
 * https://www.freertos.org/Embedded-RTOS-Queues.html
 * src : https://wokwi.com/projects/new/arduino-uno
 */

// Include Arduino FreeRTOS library
#include <Arduino_FreeRTOS.h>

// Include queue support
#include <queue.h>

// Define a Structure Array
struct Arduino{
  int pin[2];
  int ReadValue[2];
};

//Function Declaration
void Blink(void *pvParameters);
void POT(void *pvParameters);
void TaskSerial(void *pvParameters);

/*
 * Declaring a global variable of type QueueHandle_t
 *
 */
QueueHandle_t structArrayQueue;

void setup() {

  /**
   * Create a queue.
   * https://www.freertos.org/a00116.html
   */
structArrayQueue=xQueueCreate(10, //Queue length
                              sizeof(struct Arduino)); //Queue item size

if(structArrayQueue!=NULL){


  xTaskCreate(TaskBlink,  // Task function
              "Blink",// Task name
              128,// Stack size
              NULL,
              0,// Priority
              NULL);

 // Create other task that publish data in the queue if it was created.
  xTaskCreate(POT,// Task function
              "AnalogRead",// Task name
              128,  // Stack size
              NULL,
              2,// Priority
              NULL);

   // Create task that consumes the queue if it was created.
   xTaskCreate(TaskSerial,// Task function
              "PrintSerial",// A name just for humans
              128,// This stack size can be checked & adjusted by reading the Stack Highwater
              NULL,
              1, // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being
              NULL);
}
}

void loop() {}

/*
 * Blink task.
 * See Blink_AnalogRead example.
 */
void TaskBlink(void *pvParameters){
  (void) pvParameters;

  pinMode(13,OUTPUT);

  digitalWrite(13,LOW);

  for(;;)
  {
    digitalWrite(13,HIGH);
    vTaskDelay(250/portTICK_PERIOD_MS);
    digitalWrite(13,LOW);
    vTaskDelay(250/portTICK_PERIOD_MS);
  }
}

/**
 * Analog read task for Pin A0 and A1
 * Reads an analog input on pin 0 and pin 1
 * Send the readed value through the queue.
 * See Blink_AnalogRead example.
 */
void POT(void *pvParameters){
  (void) pvParameters;
  pinMode(A0,INPUT);
  pinMode(A1,INPUT);
  for (;;){
  // Read the input on analog pin 0:
  struct Arduino currentVariable;
  currentVariable.pin[0]=0;
  currentVariable.pin[1]=1;
  currentVariable.ReadValue[0]=analogRead(A0);
  currentVariable.ReadValue[1]=analogRead(A1);

  /**
    * Post an item on a queue.
    * https://www.freertos.org/a00117.html
    */
  xQueueSend(structArrayQueue,&currentVariable,portMAX_DELAY);

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

/**
 * Serial task.
 * Prints the received items from the queue to the serial monitor.
 */
void TaskSerial(void *pvParameters){
  (void) pvParameters;

  // Init Arduino serial
  Serial.begin(9600);

  // Wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and othe
  while (!Serial) {
    vTaskDelay(1)
  }

  for (;;){

    struct Arduino currentVariable;

   /**
     * Read an item from a queue.
     * https://www.freertos.org/a00118.html
     */
    if(xQueueReceive(structArrayQueue,&currentVariable,portMAX_DELAY) == pdPASS ){
      for(int i=0;i<2;i++){
      Serial.print("PIN:");
      Serial.println(currentVariable.pin[i]);
      Serial.print("value:");
      Serial.println(currentVariable.ReadValue[i]);
      }
    }
    vTaskDelay(500/portTICK_PERIOD_MS);
  }
}
```

Explanation

- The new data structure is defined at this line with the member `pin` and `ReadValue` as an array of integers with a size of 2.

  ```
  struct Arduino{
    int pin[2];
    int ReadValue[2];
  };
  ```

- This line will create the handler for the queue.

  ```
  QueueHandle_t structArrayQueue;
  ```

- The queue will be created using the `xQueueCreate` function that accepts the length of the queue and the size of the data that will be put inside the queue.

  ```
  structArrayQueue=xQueueCreate(10, sizeof(struct Arduino));
  ```

- The first function that will be launched with the thread is `TaskBlink` that accepts a pointer of void as the argument.

  ```
  void TaskBlink(void *pvParameters){
    (void) pvParameters;

    pinMode(13,OUTPUT);

    digitalWrite(13,LOW);
  ```

```
  for(;;)
  {
    digitalWrite(13,HIGH);
    vTaskDelay(250/portTICK_PERIOD_MS);
    digitalWrite(13,LOW);
    vTaskDelay(250/portTICK_PERIOD_MS);
  }
}
```

- `pinMode(13, OUTPUT)` sets the 13th pin to be output.
- `digitalWrite(13, LOW)` will give low electricity to pin 13.
- `digitalWrite(13, HIGH)` will give high electricity to pin 13.
- The second function that will be launched with the second task is `POT`.

```
void POT(void *pvParameters){
  (void) pvParameters;
  pinMode(A0,INPUT);
  pinMode(A1,INPUT);
  for (;;){
  // Read the input on analog pin 0:
  struct Arduino currentVariable;
  currentVariable.pin[0]=0;
  currentVariable.pin[1]=1;
  currentVariable.ReadValue[0]=analogRead(A0);
  currentVariable.ReadValue[1]=analogRead(A1);

  /**
    * Post an item on a queue.
    * https://www.freertos.org/a00117.html
    */
  xQueueSend(structArrayQueue,&currentVariable,portMAX_DELAY);

  // One tick delay (15ms) in between reads for stability
  vTaskDelay(1);
  }
}
```

- `xQueueSend()` accepts the handler, the data, and lastly the delay. The delay has a maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if the queue is full and `xTicksToWait` is set to 0. The time is defined in tick periods so the constant `portTICK_PERIOD_MS` should be used to convert to real time if this is required.
- The last task that will be spawned is `TaskSerial`.

```
void TaskSerial(void *pvParameters){
  (void) pvParameters;

  // Init Arduino serial
  Serial.begin(9600);
  // Wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and
  while (!Serial) {
    vTaskDelay(1);
  }

  for (;;){

    struct Arduino currentVariable;

   /**
     * Read an item from a queue.
     * https://www.freertos.org/a00118.html
     */
    if(xQueueReceive(structArrayQueue,&currentVariable,portMAX_DELAY) == pdPASS ){
      for(int i=0;i<2;i++){
      Serial.print("PIN:");
      Serial.println(currentVariable.pin[i]);
      Serial.print("value:");
      Serial.println(currentVariable.ReadValue[i]);
      }
    }
    vTaskDelay(500/portTICK_PERIOD_MS);
  }
}
```

- `Serial.begin()` will initialize the lowest serial frequency.
- `xQueueReceive()` accepts the handler, the data, and lastly the delay. The function will return `pdPASS` or `pdFAIL`.
- The array size can be changed. In the example, it has a length of 2.

```
#define n 10;
struct Arduino{
  int pin[n];
  int ReadValue[n];
};
```

- With that code, the length of the array will be 10.
- About `xTaskCreate(function, readable function name, stack size, args to func, priority, handler for the task)`:
  - From the example, if the `xTaskCreate` priority argument is set to be bigger, it will have the most priority and get to run first.
  - Priority is `uint8_t` with the alias of `UBaseType_t`.
  - Any unsigned 8-bit integer is possible as an argument for the `xTaskCreate` function.
  - When the priority is 0, it will be an idle task that will run only when idle (no other work to do).
- About `vTaskDelay(how long the task will be delayed in ticks)`:

– `vTaskDelay()` specifies a time at which the task wishes to unblock relative to the time at which `vTaskDelay()` is called. For example, specifying a block period of 100 ticks will cause the task to unblock 100 ticks after `vTaskDelay()` is called.