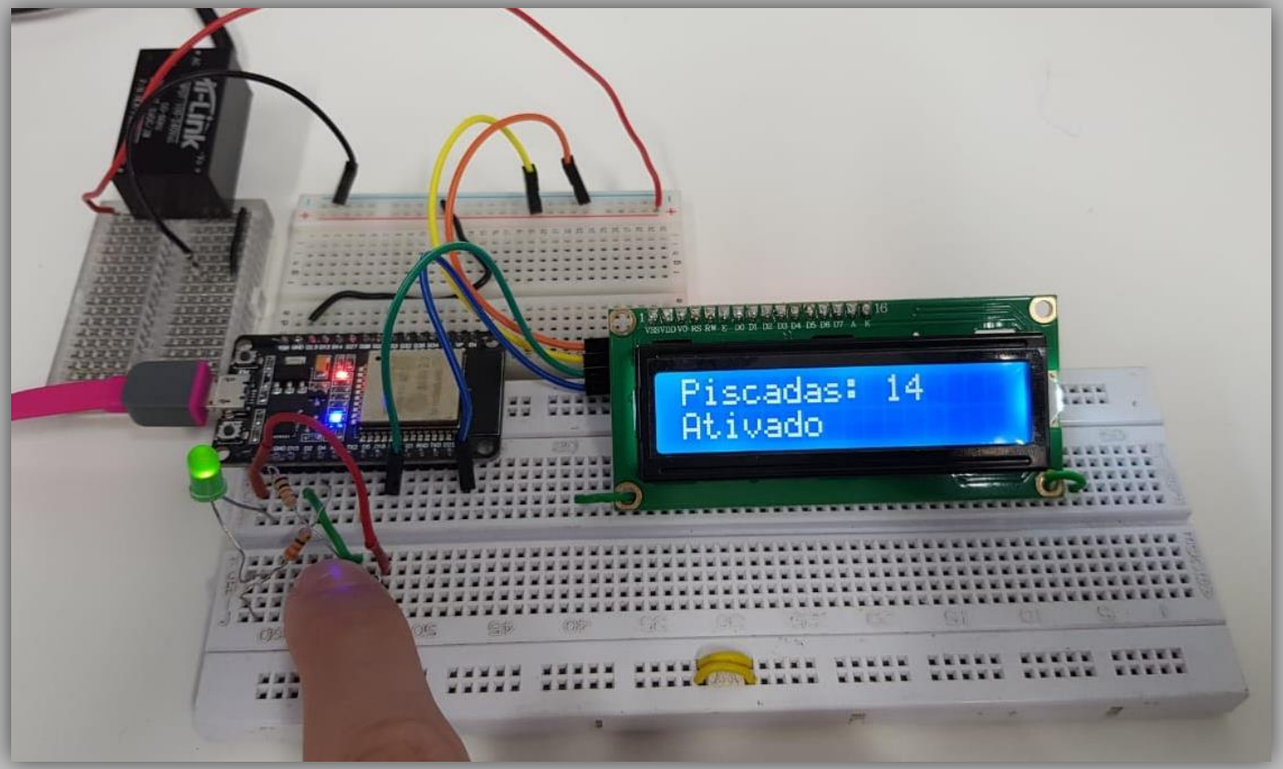
ESP32 com Arduino IDE - Programação Multi-Core



Por Fernando Koyanagi

Intenção da Aula

1. Introdução sobre a programação multi-core no ESP32

2. Conhecer as principais funções da programação multi-core

3. Criar um programa no qual diferentes tarefas são executadas simultaneamente em diferentes núcleos.

Introdução

Uma das muitas características interessantes do ESP32 é que ele tem dois **núcleos Tensilica LX6**, que podemos aproveitar para executar nosso código com maior desempenho e mais versatilidade.

Tanto o SETUP quanto as funções principais do LOOP são executadas com prioridade de 1 e no núcleo 1.

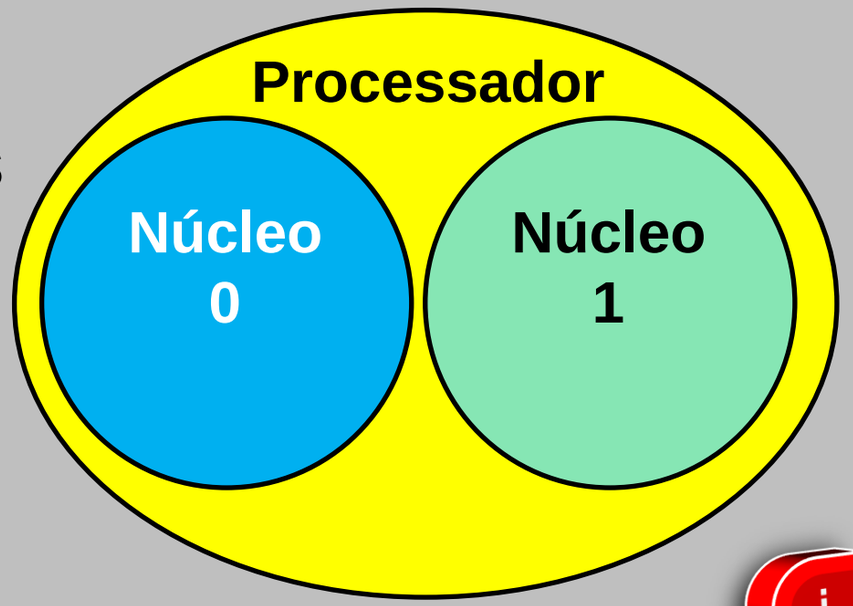
Prioridades podem ir de 0 a N, onde 0 é a menor prioridade.

Núcleo pode ser 0 ou 1.

Introdução

As tarefas têm uma prioridade atribuída que o agendador usa para decidir qual tarefa será executada. Tarefas de alta prioridade que estejam prontas para serem executadas terão preferência sobre as tarefas de menor prioridade. Em um caso extremo que a tarefa de maior prioridade precise da CPU o tempo todo, a de menor prioridade nunca seria executada.

Mas com dois núcleos disponíveis, as duas tarefas podem ser executadas, desde que elas sejas atribuídas para núcleos diferentes.



Funções

Vejamos agora alguma das funções importantes que podemos utilizar.

xTaskCreate

Cria uma nova tarefa e adiciona à lista de tarefas que estão prontas para serem executadas.

```
BaseType_t xTaskCreate(   TaskFunction_t pvTaskCode, //função que implementa a tarefa
                          const char * const pcName, //nome da tarefa
                          unsigned short usStackDepth, //número de palavras a serem alocadas para uso com a pilha da tarefa
                          void *pvParameters, //parâmetro de entrada para a tarefa (pode ser NULL)
                          UBaseType_t uxPriority, //prioridade da tarefa (0 a N)
                          TaskHandle_t *pxCreatedTask //referência para a tarefa (pode ser NULL)
                       );
```

xTaskCreatePinnedToCore

Essa função faz exatamente a mesma coisa que a xTaskCreate, porém temos um parâmetro adicional, que é onde definiremos em qual núcleo a tarefa será executada.

```
BaseType_t xTaskCreatePinnedToCore(
                          TaskFunction_t pvTaskCode, //função que implementa a tarefa
                          const char * const pcName, //nome da tarefa
                          unsigned short usStackDepth, //número de palavras a serem alocadas para uso com a pilha da tarefa
                          void *pvParameters, //parâmetro de entrada para a tarefa (pode ser NULL)
                          UBaseType_t uxPriority, //prioridade da tarefa (0 a N)
                          TaskHandle_t *pxCreatedTask, //referência para a tarefa (pode ser NULL)
                          BaseType_t xCoreID //núcleo para execução da tarefa (0 ou 1)
                        );
```

Funções

xPortGetCoreID

Essa função retorna o número do núcleo que está executando a tarefa atual.

TaskHandle_t

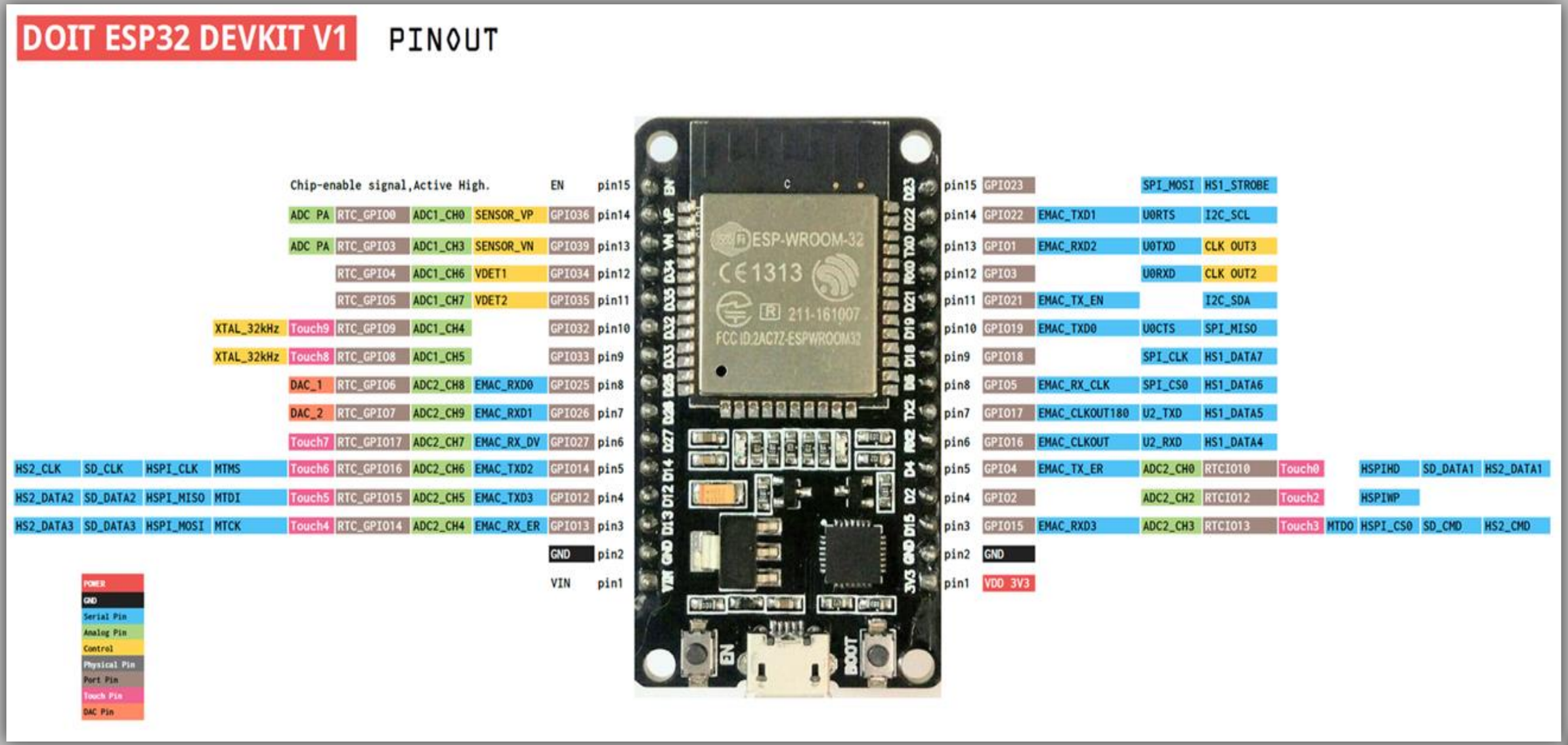
É o tipo de referência para a tarefa criada.
A chamada xTaskCreate retorna (como ponteiro para um parâmetro) um TaskHandle_t que pode ser usado como parâmetro por vTaskDelete para deletar uma tarefa.

vTaskDelete

Deleta uma tarefa criada.

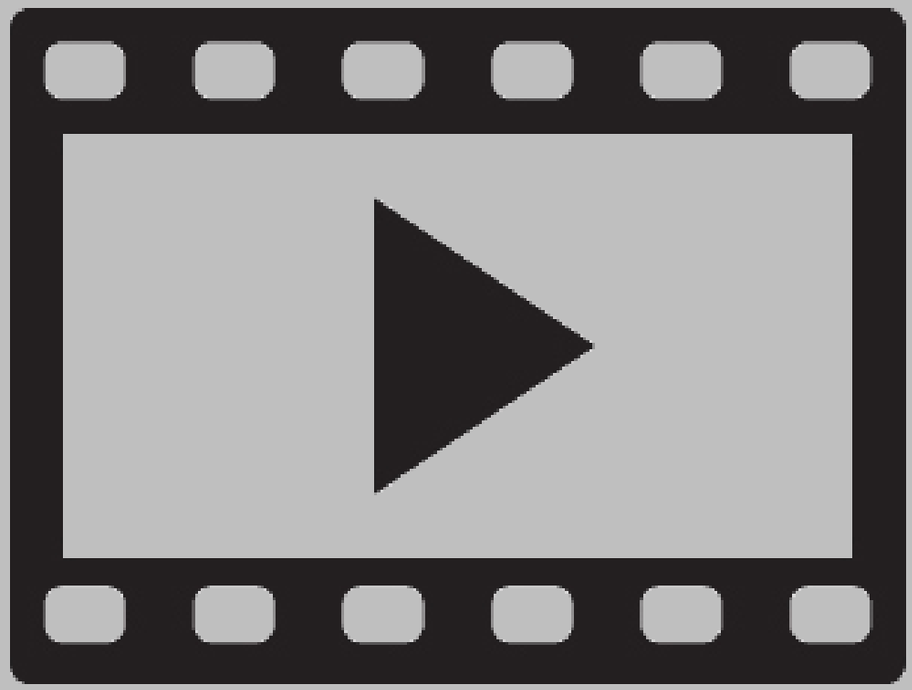
```
vTaskDelete(TaskHandle_t xTaskToDelete)
```

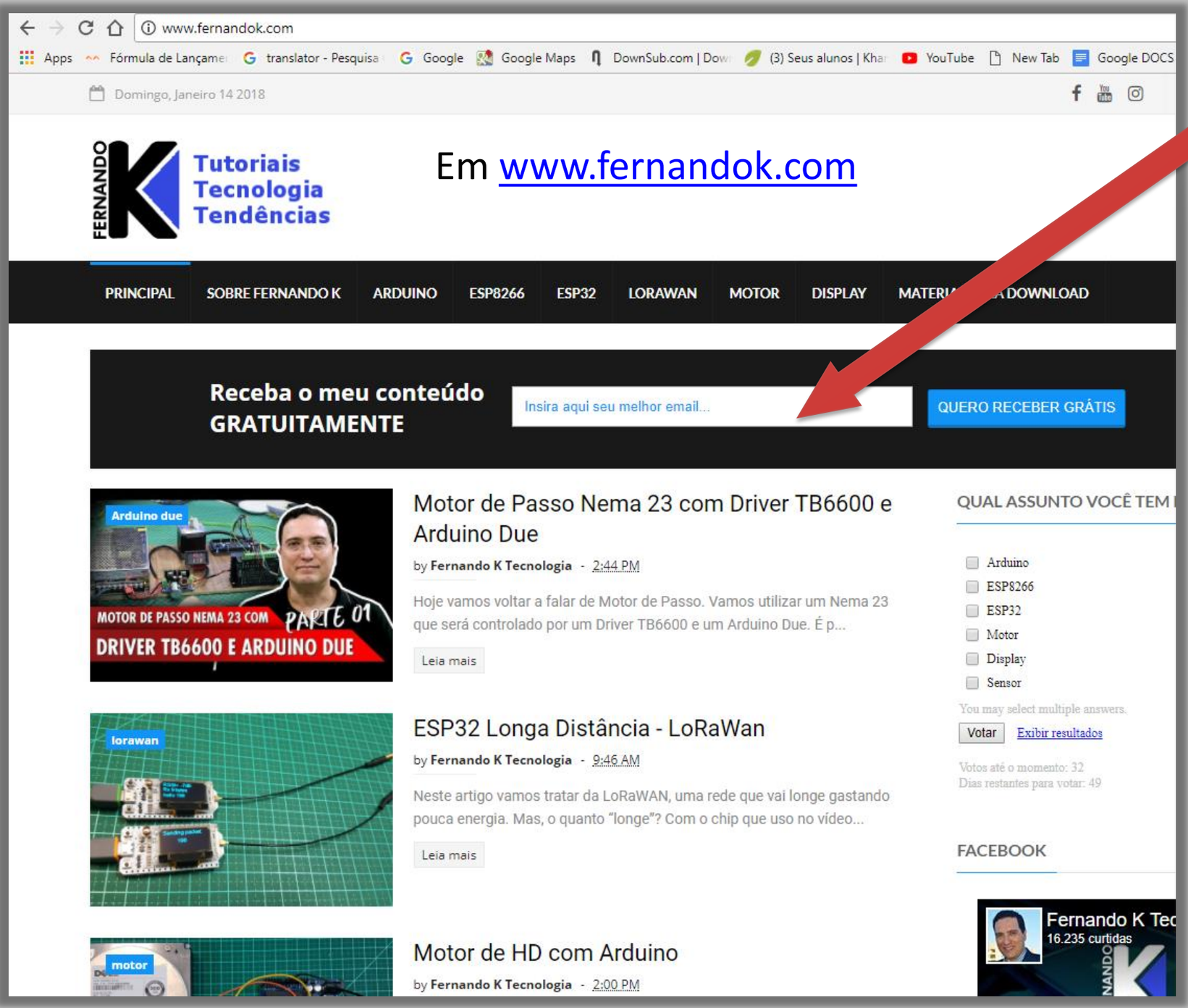
WiFi NodeMCU-32S ESP-WROOM-32



Display LCD 16x2 Serial com módulo i2c

Demonstração

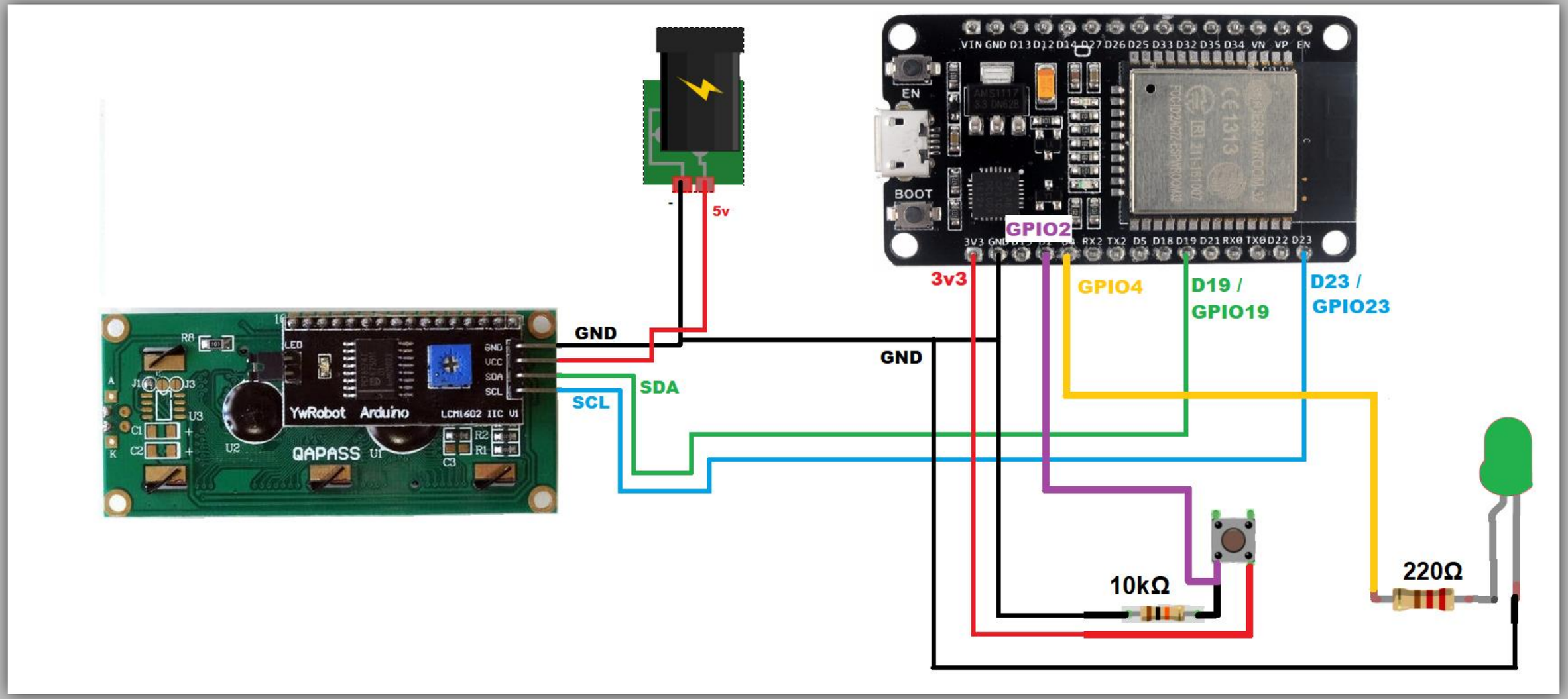




Seu e-mail

Inscreva-se

Montagem


5v
-
GND
SDA
SCL
GND
3v3
GPIO2
GPIO4
D19 /
GPIO19
D23 /
GPIO23
10kΩ
220Ω
EN
BOOT
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
GND
VCC
SDA
SCL
YwRobot Arduino
LCM1602 IIC V1
QAPASS

Biblioteca

Adicione a biblioteca “LiquidCrystal_I2C” para comunicação com o display LCD.

Acesse o link e faça download da biblioteca.

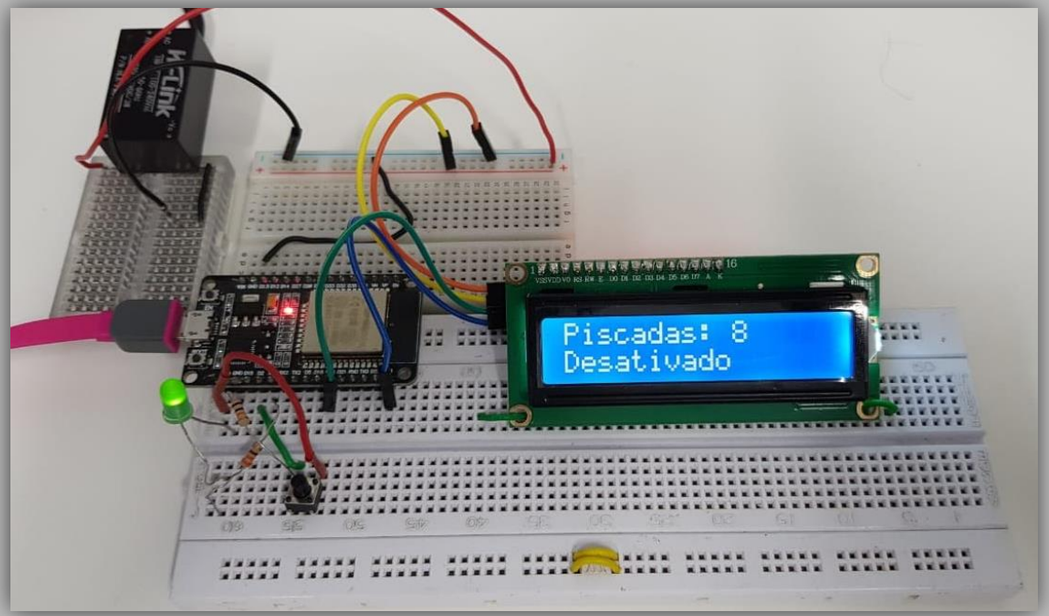
Descompacte o arquivo e cole na pasta de bibliotecas da IDE do arduino.

C:/Program Files (x86)/Arduino/libraries

Programa

Faremos um programa simples que consiste em fazer um **led piscar** e **contar quantas vezes** ele piscou, teremos também um **botão** que ao ser **pressionado muda uma variável de controle de estado** do mesmo, o **display ficará atualizando** todas essas **informações**.

Programaremos a tarefa de atualização do **display** para executar no **núcleo UM** do processador, e as **outras operações no núcleo ZERO**.

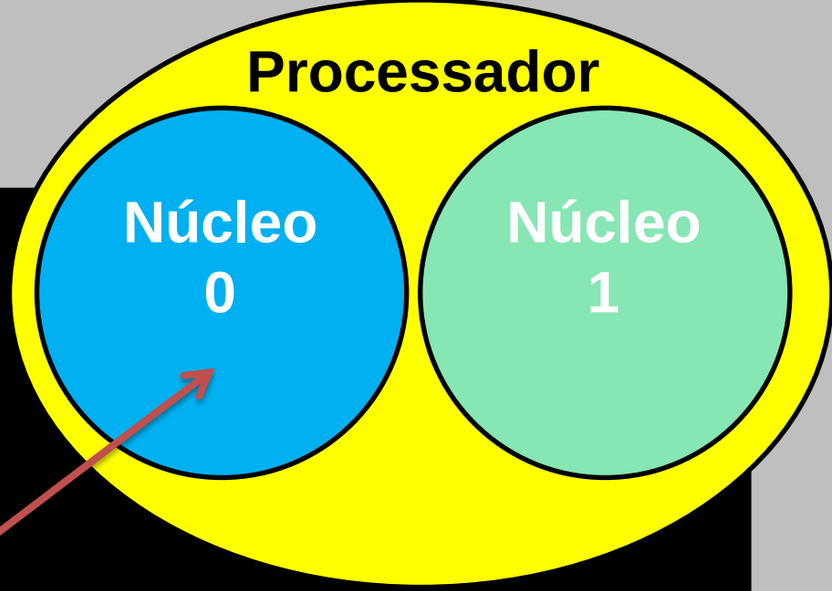


Bibliotecas e variáveis

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h> //biblioteca responsável pelo controle do display

LiquidCrystal_I2C lcd(0x27, 16, 2); //set the LCD address to 0x27 for a 16 chars and 2 line display

//variáveis para controle do LED
int count = 0;
int blinked = 0;

String statusButton = "DESATIVADO";

//pinos usados
const uint8_t pin_led = 4;
const uint8_t pin_btn = 2;

//variaveis que indicam o núcleo
static uint8_t taskCoreZero = 0;
static uint8_t taskCoreOne = 1;
```

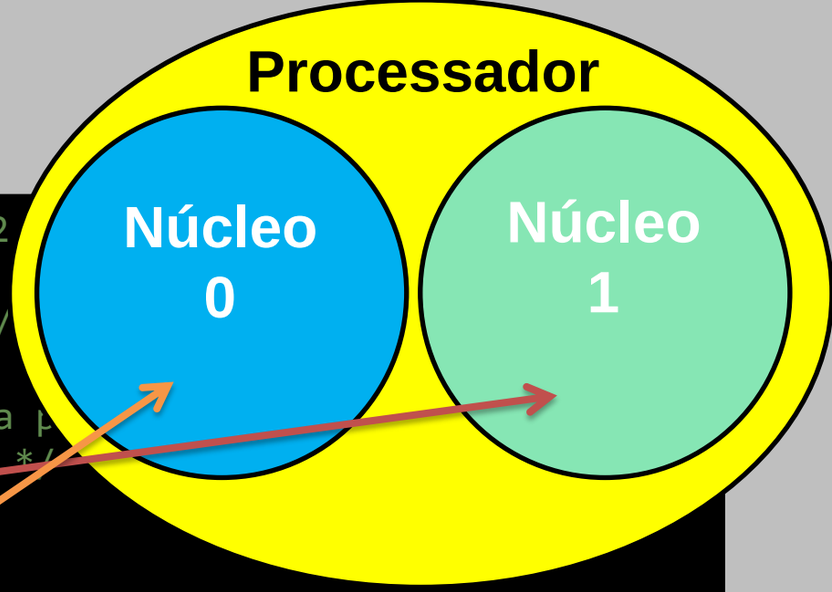
Setup

```
void setup() {
     pinMode(pin_led, OUTPUT);
     pinMode(pin_btn, INPUT);

     //inicializa o LCD com os pinos SDA e SCL
     lcd.begin(19, 23);

     // Liga a luz do display
     lcd.backlight();
     lcd.setCursor(0, 0);
     lcd.print("Piscadas:");

     //cria uma tarefa que será executada na função coreTaskZero, com prioridade 1 e execução no núcleo 0
     xTaskCreatePinnedToCore(
                    coreTaskZero, /* função que implementa a tarefa do LED */
                    "coreTaskZero", /* nome da tarefa */
                    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
                    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
                    1, /* prioridade da tarefa (0 a N) */
                    NULL, /* task handle referência para a tarefa (pode ser NULL) */
                    taskCoreZero); /* Núcleo que executará a tarefa */

      delay(500); //tempo para a tarefa iniciar
```



Setup (continuação)

```
    //cria uma tarefa que será executada na função coreTaskOne, com prioridade 2
    xTaskCreatePinnedToCore(
                    coreTaskOne, /* função que implementa a tarefa do Display */
                    "coreTaskOne", /* nome da tarefa */
                    10000, /* número de palavras a serem alocadas para uso com a p
                    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
                    2, /* prioridade da tarefa (0 a N) */
                    NULL, /* referência para a tarefa (pode ser NULL) */
                    taskCoreOne); /* Núcleo que executará a tarefa */

delay(500); //tempo para a tarefa iniciar

    //cria uma tarefa que será executada na função coreTaskTwo, com prioridade 2 e execução no núcleo 0
    xTaskCreatePinnedToCore(
                    coreTaskTwo, /* função que implementa a tarefa do Botão */
                    "coreTaskTwo", /* nome da tarefa */
                    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
                    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
                    2, /* prioridade da tarefa (0 a N) */
                    NULL, /* referência para a tarefa (pode ser NULL) */
                    taskCoreZero); /* Núcleo que executará a tarefa */
delay(500); //tempo para a tarefa iniciar
}

void loop() {}
```



TaskZero

Prioridade:

Núcleo de Execução:

```
//cria uma tarefa que será executada na função c
xTaskCreatePinnedToCore( coreTaskZero,   /* func
                         "coreTaskZero", /* nome
                         10000,      /* número d
                         NULL,       /* parâmetr
                         1,          /* priorida
                         NULL,       /* referênc
                         taskCoreZero);
```

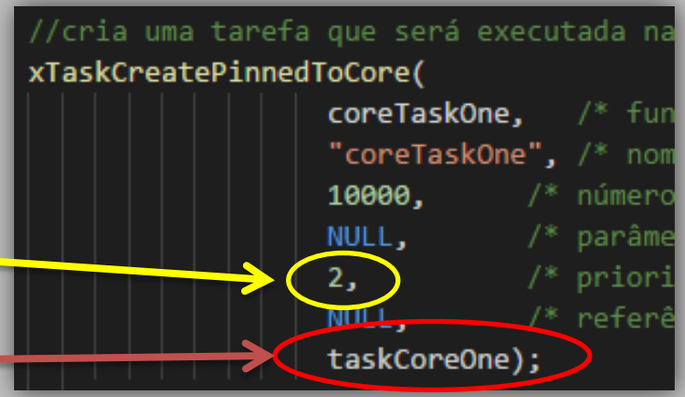
```
//essa função ficará mudando o estado do led a cada 1 segundo
//e a cada piscada (ciclo acender e apagar) incrementará nossa variável blinked
void coreTaskZero( void * pvParameters ){
    String taskMessage = "Tarefa executando no núcleo: ";
    taskMessage = taskMessage + xPortGetCoreID();
    Serial.println(taskMessage);
    while(true){
        digitalWrite(pin_led, !digitalRead(pin_led));

        if (++count % 2 == 0 )
                blinked++;

        delay(1000);
    }
}
```

Núcleo 0

TaskOne

Prioridade:

Núcleo de Execução:



```
//cria uma tarefa que será executada na
xTaskCreatePinnedToCore(
                    coreTaskOne,   /* fun
                    "coreTaskOne", /* nom
                    10000,       /* número
                    NULL,        /* parâme
                    2,           /* priori
                    NULL,        /* referê
                    taskCoreOne);
```

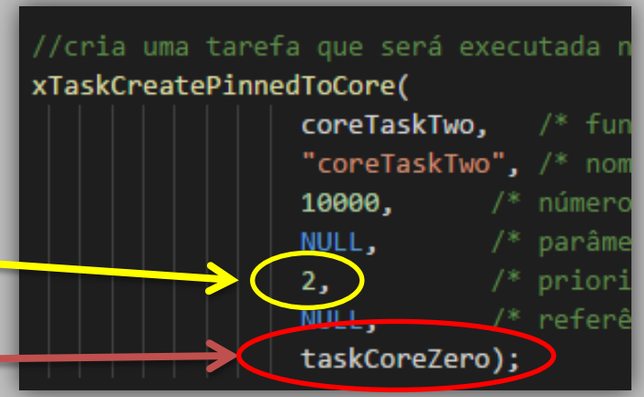
```
//essa função será responsável apenas por atualizar as informações no
//display a cada 100ms
void coreTaskOne( void * pvParameters ){
    while(true){
    lcd.setCursor(10, 0);
    lcd.print(blinked);

    lcd.setCursor(0,1);
    lcd.print(statusButton);
    delay(100);
    }
}
```

Núcleo 1

TaskTwo

Prioridade:

Núcleo de Execução:



```
//cria uma tarefa que será executada n
xTaskCreatePinnedToCore(
                coreTaskTwo,   /* fun
                "coreTaskTwo", /* nom
                10000,      /* número
                NULL,       /* parâme
                2,          /* priori
                NULL,       /* referê
                taskCoreZero);
```

```
//essa função será responsável por ler o estado do botão
//e atualizar a variavel de controle.
void coreTaskTwo( void * pvParameters ){
    while(true){
        if(digitalRead(pin_btn)){
                statusButton = "Ativado ";
        }
        else statusButton = "Desativado";

        delay(10);
    }
}
```

Núcleo 0

Em www.fernandok.com

Download arquivos PDF e INO do código fonte