

Universidade Tecnológica Federal do Paraná – Toledo
Engenharia da Computação – COENC

Sistemas Embarcados

Introdução ao FreeRTOS **- Tarefas -**

Tiago Piovesan Vendruscolo



Esta licença permite que outros remixem, adaptem e criem a partir do trabalho para fins não comerciais, desde que atribuam o devido crédito aos autores originais. [4.0 international](https://creativecommons.org/licenses/by-nc-nd/4.0/)

- É o RTOS de código aberto mais utilizado no mundo.
 - Desenvolvido por Richard Barry em 2003.
 - Pode ser utilizado em sistemas comerciais sem custos de licença
 - Aceita portabilidade em um grande número de microcontroladores e plataformas
- Em 2017 o FreeRTOS foi adquirido pela Amazon Web Services (AWS), lançando customizações para seus serviços de IoT. <https://aws.amazon.com/pt/freertos/>
 - *Projetado de forma enxuta o suficiente para ser utilizado em microcontroladores.*
 - *Utiliza linguagem C, facilmente portátil, praticamente sem restrição de chips e compiladores.*
 - *Pode trabalhar de forma preemptiva ou colaborativa.*
 - *API de fácil utilização*
 - *Temporizadores de software altamente eficientes*



- Quais facilidades ganhamos com um RTOS?
 - *Gerenciamento de múltiplas tarefas*
 - *Uso de recursos compartilhados de forma segura*
 - *Sistema de prioridade*
 - *Comunicação segura entre tarefas: filas, notificações*
 - *Portabilidade/reaproveitamento de código fácil (utilizando o mesmo RTOS)*
- Ports do FreeRTOS
 - *Embora a maior parte do Kernel do FreeRTOS é igual para todas as plataformas, existem algumas funções extras específicas para apenas algumas plataformas. Essas informações são encontradas com o fabricante da plataforma.*

Documentação padrão:

https://www.freertos.org/Documentation/RTOS_book.html

API Reference:

<https://www.freertos.org/a00106.html>



- O que aprenderemos sobre FreeRTOS?
 - *Tarefas: Como criar, deletar, suspender, reiniciar, utilizar múltiplas instâncias e selecionar núcleo de processamento*
 - *Filas (Queue) com e sem interrupção*
 - *Semáforos: Binários, contadores e MUTEX*
 - *Software Timer*
 - *Event Groups*
 - *Task Notifications*



▪ Convenções

- Variáveis unsigned começam com “u”. Variáveis do tipo char (8bits) começam com “c”
- Variáveis do tipo short(16bits) começam com “s”
- Variáveis do tipo long(32bits) começam com “l”
- Ponteiros começam com “p”
- Funções privadas em um arquivo começam com “prv”
- As funções da API são prefixadas com seu tipo de retorno, de acordo com a convenção das variáveis, com a adição do prefixo v para void.
- Os nomes das funções da API começam com o nome do arquivo no qual estão definidos. Por exemplo, vTaskDelete é definido em tasks.c e possui um tipo de retorno void. “x” significa que retorna um tipo não standard (padrão FreeRTOS).

▪ Indentação

- É utilizado o Tab, ou 4 espaços

▪ Comentários

- Nunca passam da coluna 80, a menos que sigam e descrevam um parâmetro
- Não é utilizado “//” para comentários

Tipo de variável padrão para o FreeRTOS é **UBaseType_t**

- BaseType_t

This is defined to be the most efficient, natural type for the architecture. For example, on a 32-bit architecture, BaseType_t will be defined to be a 32-bit type. On a 16-bit architecture, BaseType_t will be defined to be a 16-bit type. If BaseType_t is defined to char, then particular care must be taken to ensure signed chars are used for function return values that can be negative to indicate an error.

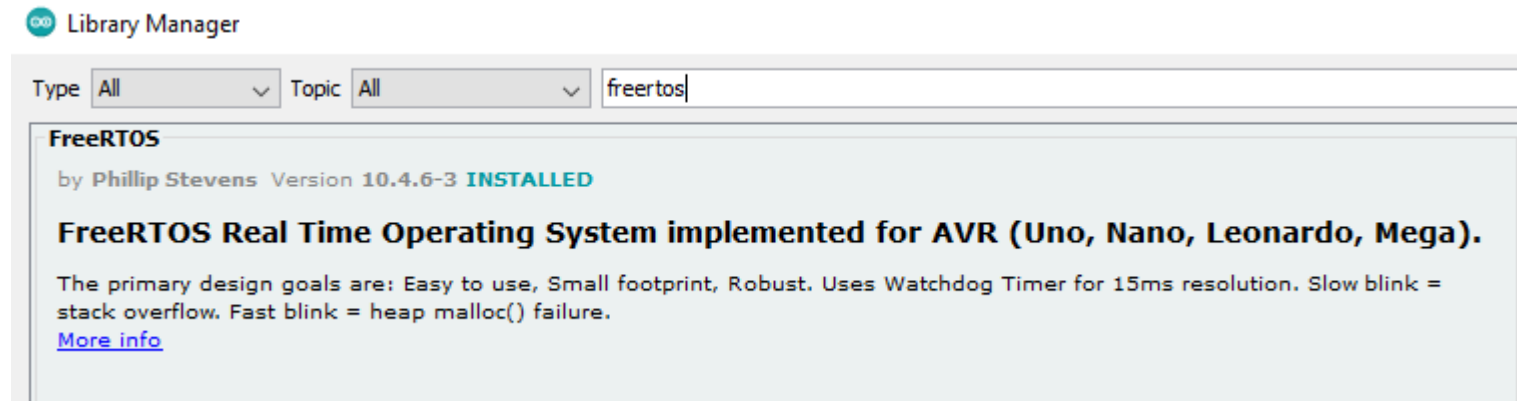
- UBaseType_t

This is an unsigned BaseType_t.

<https://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html>

Exemplos de projetos com FreeRTOS

- Como baixar a biblioteca FreeRTOS no Arduino IDE para o Arduino.
 - *Abra o gerenciador de biblioteca, pesquise por FreeRTOS e baixe a versão mais atual do Richard Barry (atualmente aparece Phillip Stevens)*



- `#include <Arduino_FreeRTOS.h>`
- Para o PlatformIO, pesquise pela mesma biblioteca acima e adicione no projeto, após, adicione os seguintes includes:
 - `#include <Arduino_FreeRTOS.h>`
 - `#include <Arduino.h>`

```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
lib_deps = feilipu/FreeRTOS@^10.4.4-2
```

Utilizando tarefas no FreeRTOS

- Pontos importantes:
 - *Multitarefas: Entender que cada tarefa funciona de forma independente.*
 - *A ordem de execução é desconhecida.*
 - *Isso requer um planejamento para situações de sincronia.*
 - *Compartilhamento de variáveis: O mau uso de variáveis globais pode facilmente acarretar em dados corrompidos. Use com cuidado!*
 - Utilize as ferramentas do SO para organizar o acesso as variáveis – Mutex e filas.
 - *Temporização/Delay: O uso incorreto de rotinas de delay em uma tarefa pode gerar atrasos desnecessários em todo o código.*
 - Delay \neq sincronia.

Criando uma tarefa com FreeRTOS

Necessário incluir a library `task.h`

- Passos para a criação de um código com RTOS:
 - Criação das tarefas (*`xTaskCreate`*);
 - Desenvolvimento da função da tarefa;

Protótipo de uma tarefa

```
void vTarefa(void *pvParameters);
```

Corpo de uma tarefa

```
void vTarefa(void *pvParameters)
{
    /*Declarações*/
    while(1)
    {
        /*Código da tarefa */
    }
}
```

<https://www.freertos.org/a00019.html>

Criando uma tarefa com FreeRTOS

- Criação das tarefas (xTaskCreate)

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

Função chamada pela tarefa

String com um nome para identificar a tarefa / fazer depuração

Parâmetros para a tarefa (caso a tarefa não tenha, use NULL)

```
xTaskCreate(vTarefa1, "Tarefa1", 128, NULL, 1, NULL);
```

Tamanho da pilha em **palavras**, sendo que cada palavra tem o tamanho padrão da pilha, pode-se utilizar a API `uxTaskGetStackHighWaterMark()` para analisar a quantidade de palavras que estão sobrando. `configMINIMAL_STACK_SIZE` para usar o menor tamanho possível.

Prioridade da tarefa (valor inteiro). Quanto menor o número, menor a prioridade.

Handle da tarefa, armazena as informações da tarefa. Opcional, desde que você não precise suspender e reiniciar a tarefa durante o tempo de execução da tarefa. (Caso não utilize, use NULL)

Exemplos de projetos com FreeRTOS

- Exemplo 1: Faça 1 LED piscar em 1Hz e o outro em 0,5 HZ. LED1 pino 7 e LED2 pino 8

```
#include "Arduino_FreeRTOS.h"
#include "task.h"
```

```
#define LED1 7
#define LED2 8
```

```
/* Variáveis para armazenamento do handle das tasks
```

```
TaskHandle_t Tarefa1Handle = NULL;
```

```
TaskHandle_t Tarefa2Handle = NULL; */
```

```
/*protótipos das Tasks*/
```

```
void vTarefa1(void *pvParameters);
```

```
void vTarefa2(void *pvParameters);
```

```
void setup() {
```

```
    xTaskCreate(vTarefa1, "Tarefa1",128, NULL, 1, NULL);
```

```
    xTaskCreate(vTarefa2, "Tarefa2",128, NULL, 1, NULL);
```

```
    /*vTaskStartScheduler(); */
```

```
}
```

Nessa situação em que as duas tarefas possuem a mesma prioridade, elas são executadas pelo escalonamento Round Robin (Time Slicing), ou seja, executa as tarefas dentro de uma janela de tempo.

Handle das tarefas (nesse caso não está sendo utilizado)

Inicia o gerenciador de tarefas, não é necessário no framework Arduino

Exemplos de projetos com FreeRTOS

```
void loop() {  
    /*As funções são executadas nas tarefas*/  
}  
  
void vTarefa1(void *pvParameters){  
    pinMode(LED1, OUTPUT);  
    while(1) {  
        digitalWrite(LED1,!digitalRead(LED1));  
        vTaskDelay(pdMS_TO_TICKS(500));  
    }  
}  
  
void vTarefa2(void *pvParameters){  
    pinMode(LED2, OUTPUT);  
    while(1) {  
        digitalWrite(LED2,!digitalRead(LED2));  
        vTaskDelay(pdMS_TO_TICKS(1000));  
    }  
}
```

No FreeRTOS a função `loop()` é chamada pela `idleTask` (prioridade zero) no momento em que não há nenhuma outra tarefa solicitando o processador. A partir do momento que uma tarefa solicita o uso do processador, o loop deixa de ser executado. Dessa forma, o ideal é não ter nada executando no `loop()`.

Application Hooks
`variantHooks.cpp`

A entrada do `vTaskDelay` é em ticks, utilizando a função `pdMS_TO_TICKS` ele converte de milissegundos para ticks.

Pode ser colocado no setup. No entanto, dentro da task é útil caso queira parametrizar qual pino usar.

Application Hooks

- C:\...\Arduino\libraries\FreeRTOS\src\variantHooks.cpp
- Idle task – Padrão arduino: loop ()
- Malloc fail
 - *Permite que o desenvolvedor programe alguma ação quando der erro de Malloc (alocação de memória).*
- Stack overflow
- Tick: Context switch

Exemplos de projetos com FreeRTOS

- Exercício 1: Adicione uma terceira tarefa, com um contador de segundos, que imprima na Serial a contagem a cada 1 segundo. Utilize o controle de prioridade, de forma que a serial tenha a prioridade máxima, e a tarefa 1 tenha a prioridade mínima.

```
void setup() {  
    Serial.begin(9600);  
    xTaskCreate(vTarefa1, "Tarefa1",128, NULL, 1, &Tarefa1Handle);  
    xTaskCreate(vTarefa2, "Tarefa2",128, NULL, 2, &Tarefa2Handle);  
    xTaskCreate(vTarefa3, "Tarefa3",128, NULL, 3, &Tarefa3Handle);  
}
```

```
void vTarefa3(void *pvParameters)  
{  
    int contador = 0;  
    while(1)  
    {  
        Serial.println("Tarefa 3: " + String(contador++));  
        vTaskDelay(pdMS_TO_TICKS(1000));  
    }  
}
```

Deletando uma tarefa com FreeRTOS

- Exercício 2: Na tarefa 3, delete a tarefa 1 e deixe o LED em nível baixo quando o contador chegar em 5. Imprima na serial quando for deletado.

```
vTaskDelete(Tarefa1Handle);
```

- Precisamos utilizar o Handle*

```
#define LED1 7
#define LED2 8

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t Tarefa1Handle = NULL;
TaskHandle_t Tarefa2Handle = NULL;
TaskHandle_t Tarefa3Handle = NULL;

/*protótipos das Tasks*/
void vTarefa1(void *pvParameters);
void vTarefa2(void *pvParameters);
void vTarefa3(void *pvParameters);

void setup() {
    Serial.begin(9600);
    xTaskCreate(vTarefa1, "Tarefa1",128, NULL, 1, &Tarefa1Handle);
    xTaskCreate(vTarefa2, "Tarefa2",128, NULL, 2, &Tarefa2Handle);
    xTaskCreate(vTarefa3, "Tarefa3",128, NULL, 3, &Tarefa3Handle);
}
```


Deletando uma tarefa com FreeRTOS

- Exercício 2: Na tarefa 3, delete a tarefa 1 e deixe o LED em nível baixo quando o contador chegar em 5. Imprima na serial quando for deletado.

```
void vTarefa3(void *pvParameters)
{
    int contador = 0;
    while(1)
    {
        Serial.println("Tarefa 3: " + String(contador++));
        if(contador==5){
            if(Tarefa1Handle!=NULL){
                Serial.println("Deletando tarefa 1");
                vTaskDelete(Tarefa1Handle);
                digitalWrite(LED1, LOW);
            }
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

Verifica se a tarefa está ativa

Deleta a tarefa (retira do Scheduler e **deleta** sua pilha). Isso é útil quando uma tarefa é utilizada por pouco tempo e após é necessário liberar memória. A tarefa não poderá ser recuperada.

```
19:20:07.797 -> Tarefa 3: 0
19:20:08.920 -> Tarefa 3: 1
19:20:10.046 -> Tarefa 3: 2
19:20:11.167 -> Tarefa 3: 3
19:20:12.290 -> Tarefa 3: 4
19:20:12.290 -> Deletando task 1
19:20:13.369 -> Tarefa 3: 5
```

Deletando uma tarefa com FreeRTOS

- Exercício 3: Na tarefa 3, delete a tarefa 1 quando o contador chegar em 5, a tarefa 2 quando o contador chegar em 10 e a própria tarefa 3 quando o contador chegar em 15. Deixe o LED em nível baixo quando deletar a tarefa.

```
Tarefa 3: 0
Tarefa 3: 1
Tarefa 3: 2
Tarefa 3: 3
Tarefa 3: 4
Deletando task 1
Tarefa 3: 5
Tarefa 3: 6
Tarefa 3: 7
Tarefa 3: 8
Tarefa 3: 9
Deletando task 2
Tarefa 3: 10
Tarefa 3: 11
Tarefa 3: 12
Tarefa 3: 13
Tarefa 3: 14
Deletando task 3
```

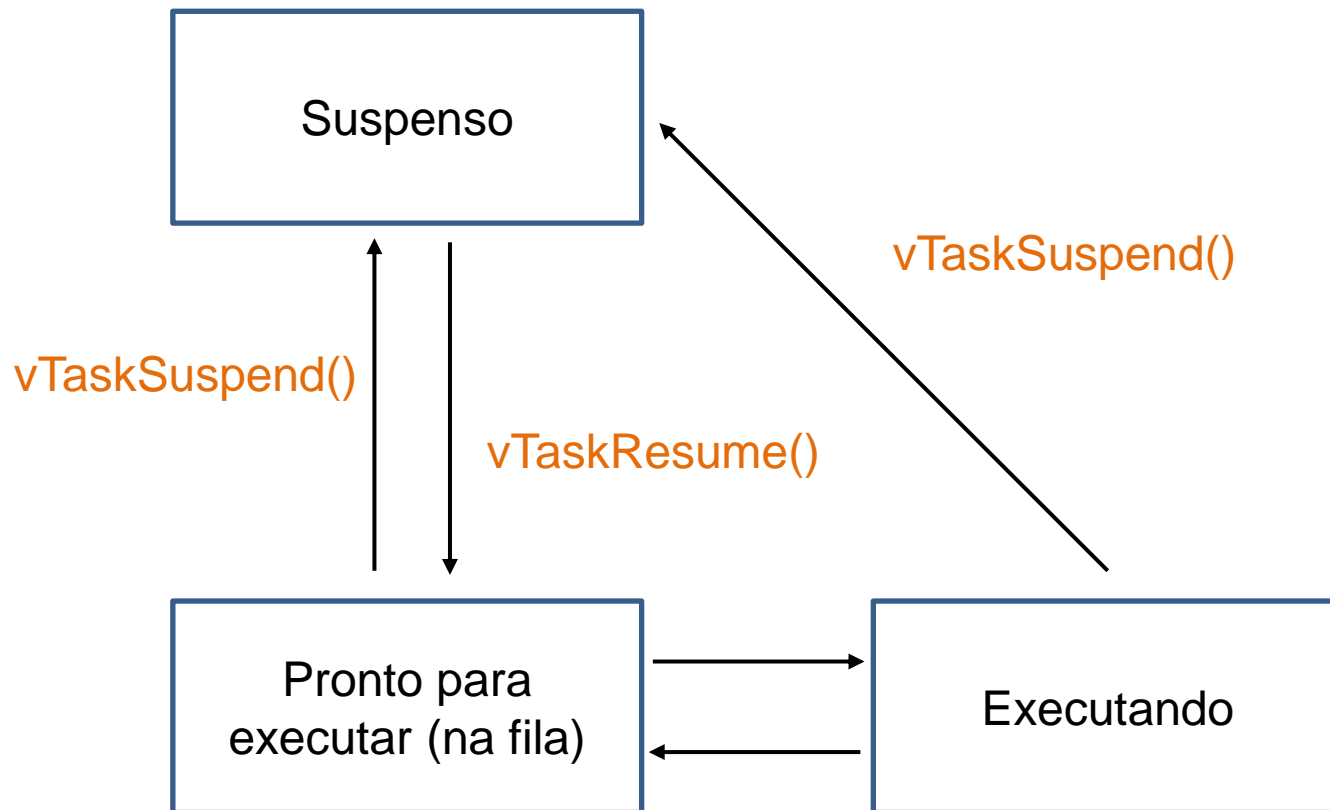
```
void vTarefa3(void *pvParameters)
{
    int contador = 0;
    while(1)
    {
        Serial.println("Tarefa 3: " + String(contador++));
        if(contador==5){
            if(Tarefa1Handle!=NULL){
                Serial.println("Deletando task 1");
                vTaskDelete(Tarefa1Handle);
                digitalWrite(LED1,LOW);
            }
        }
        if(contador==10){
            if(Tarefa2Handle!=NULL){
                Serial.println("Deletando task 2");
                vTaskDelete(Tarefa2Handle);
                digitalWrite(LED2,LOW);
            }
        }
        if(contador==15){
            if(Tarefa3Handle!=NULL){
                Serial.println("Deletando task 3");
                vTaskDelete(Tarefa3Handle);
            }
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

- Voltando para o slide do início da aula....
- Pontos importantes:
 - *Multitarefas: Entender que cada tarefa funciona de forma independente.*
 - *A ordem de execução é desconhecida.*
 - *Isso requer um planejamento para situações de sincronia.*

O que fazer quando tenho uma tarefa de execução crítica que não pode ser interrompida?

Suspendendo e reiniciando uma tarefa

- Ao suspender uma tarefa, você retira essa tarefa do Scheduler (mas a pilha continua existindo)



Suspendendo e reiniciando uma tarefa

- Quando o sistema possui uma tarefa em que a sua execução não pode ser interrompida, podemos utilizar as seguintes API's: (caso o sistema aceite pausas)

`vTaskSuspend()`

`vTaskResume()`

<https://www.freertos.org/a00130.html>

- `vTaskSuspend()` irá suspender (retirar do scheduler) todas as outras tarefas, independentes da prioridade, após, deve-se utilizar `vTaskResume()` para reativar as tarefas. As interrupções continuam funcionando. Para utilizar essas API's, é necessário alterar `INCLUDE_ vTaskSuspend` para 1, no arquivo `FreeRTOSConfig.h`, que fica na pasta:

`C:\Users\xxxx\Documents\Arduino\libraries\FreeRTOS\src`

```
/* Optional functions - most linkers will remove unused functions anyway. */  
#define INCLUDE_vTaskPrioritySet      1  
#define INCLUDE_uxTaskPriorityGet     1  
#define INCLUDE_vTaskDelete          1  
#define INCLUDE_vTaskSuspend         1
```

- Segunda opção:

```
taskENTER_CRITICAL()  
taskEXIT_CRITICAL()  
https://www.freertos.org/taskENTER\_CRITICAL\_  
taskEXIT\_CRITICAL.html
```

- `taskENTER_CRITICAL()` irá iniciar uma seção de processamento crítico, que não pode ser interrompido até que a API `taskEXIT_CRITICAL()` seja utilizada. Essa função é mais restritiva que a `vTaskSuspend()`, pois ela também desabilita todas as interrupções, dessa forma, sempre analise com cuidado se ela pode ser utilizada sem prejudicar o sistema.
 - Se a seção crítica travar, todo o sistema irá travar.

Suspendendo e reiniciando uma tarefa

- Exercício 4: Use o exercício 2 como base. Na tarefa 3, suspenda a tarefa 1 quando o contador chegar em 5. Quando o contador chegar em 10, suspenda a tarefa 2 e reative a tarefa 1.

```
vTaskSuspend(TarefaHandle);
```

```
vTaskResume(TarefaHandle);
```

- Nesse caso, o handle da tarefa suspensa continua existindo, então não é feita a checagem

```
if(Tarefa1Handle!=NULL) {
```

Suspendendo e reiniciando uma tarefa

■ Exercício 4:

```
void vTarefa3(void *pvParameters)
{
    int contador = 0;
    while(1)
    {
        Serial.println("Tarefa 3: " + String(contador++));
        if(contador==5){
            Serial.println("Suspendendo tarefa 1");
            vTaskSuspend(Tarefa1Handle);
            digitalWrite(LED1,LOW);
        }
        if(contador==10){
            Serial.println("Reativando tarefa 1");
            vTaskResume(Tarefa1Handle);
            Serial.println("Suspendendo tarefa 2");
            vTaskSuspend(Tarefa2Handle);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```


Verificando o consumo da pilha

- Podemos utilizar a função `uxTaskGetStackHighWaterMark()` para verificar a quantia de memória restante da pilha. O ideal é que sobre no mínimo 10% - 20% para evitar *overflow*.

Example


```
void vTask1( void * pvParameters )
{
    UBaseType_t uxHighWaterMark;

    /* Inspect the high water mark of the calling task when the task starts to
    execute. */
    uxHighWaterMark = uxTaskGetStackHighWaterMark( NULL );

    for( ;; )
    {
        /* Call any function. */
        vTaskDelay( 1000 );

        /* Calling a function will have used some stack space, so it will be
        expected that uxTaskGetStackHighWaterMark() will return a lower value
        at this point than when it was called on entry to the task function. */
        uxHighWaterMark = uxTaskGetStackHighWaterMark( NULL );
    }
}
```

A própria função consome um pouco de memória



https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf

Verificando o consumo da pilha

■ Exemplo 2: Verifique a memória restante das 3 tarefas (exercício 1)

```
void vTarefa1(void *pvParameters){
    UBaseType_t uxHighWaterMark;
    pinMode(LED1, OUTPUT);
    while(1)
    {
        digitalWrite(LED1,!digitalRead(LED1));
        vTaskDelay(pdMS_TO_TICKS(500));
        uxHighWaterMark = uxTaskGetStackHighWaterMark(NULL);
        Serial.print("Tarefa 1: ");
        Serial.println(uxHighWaterMark);
    }
}

void vTarefa2(void *pvParameters){
    UBaseType_t uxHighWaterMark;
    pinMode(LED2, OUTPUT);
    while(1)
    {
        digitalWrite(LED2,!digitalRead(LED2));
        vTaskDelay(pdMS_TO_TICKS(1000));
        uxHighWaterMark = uxTaskGetStackHighWaterMark(NULL);
        Serial.print("Tarefa 2: ");
        Serial.println(uxHighWaterMark);
    }
}

void vTarefa3(void *pvParameters){
    UBaseType_t uxHighWaterMark;
    int contador = 0;
    while(1)
    {
        Serial.println("Contador Tarefa 3: " + String(contador++));
        vTaskDelay(pdMS_TO_TICKS(1000));
        uxHighWaterMark = uxTaskGetStackHighWaterMark(NULL);
        Serial.print("Tarefa 3: ");
        Serial.println(uxHighWaterMark);
    }
}
```

```
xTaskCreate(vTarefa1, "Tarefa1",128, NULL, 1, &Tarefa1Handle);
xTaskCreate(vTarefa2, "Tarefa2",128, NULL, 2, &Tarefa2Handle);
xTaskCreate(vTarefa3, "Tarefa3",128, NULL, 3, &Tarefa3Handle);
```

Serial:

```
Tarefa 1: 65
Tarefa 1: 65
Tarefa 2: 65
Tarefa 3: 39
Contador Tarefa 3: 2
Tarefa 1: 65
Tarefa 1: 65
Tarefa 2: 65
Tarefa 3: 39
Contador Tarefa 3: 3
Tarefa 1: 65
Tarefa 1: 65
Tarefa 2: 65
Tarefa 3: 39
```

- FreeRTOS – Passagem de parâmetros e múltiplas instâncias

Referências

- <https://www.freertos.org/>
- https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf
- https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf
- <https://www.embarcados.com.br/>