

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

Sistemas Embarcados

FreeRTOS

- Task Notifications -

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/)

Task Notifications

- Cada tarefa no FreeRTOS vem com uma variável de notificação de 32 bits.
- Uma Task Notification é um evento enviado diretamente a uma tarefa, podendo desbloqueá-la.
 - *O desbloqueio de uma tarefa utilizando Task Notification é 45% mais rápido e usa menos RAM do que desbloquear utilizando semáforo binário.*
- Podem ser utilizadas como semáforos binários, contadores ou grupos de eventos.

Task Notifications

- Quando utiliza algum objeto de comunicação, é utilizado um nó intermediário que acaba atrasando a mensagem, além de ocupar mais RAM.

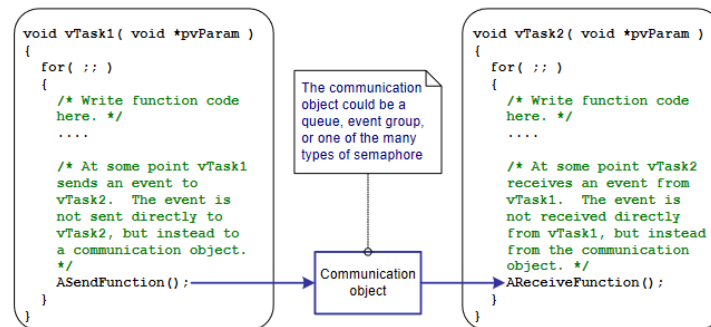


Figure 76 A communication object being used to send an event from one task to another

- A Task Notifications permite uma comunicação direta entre as tarefas, também permitindo a sincronização com interrupções.

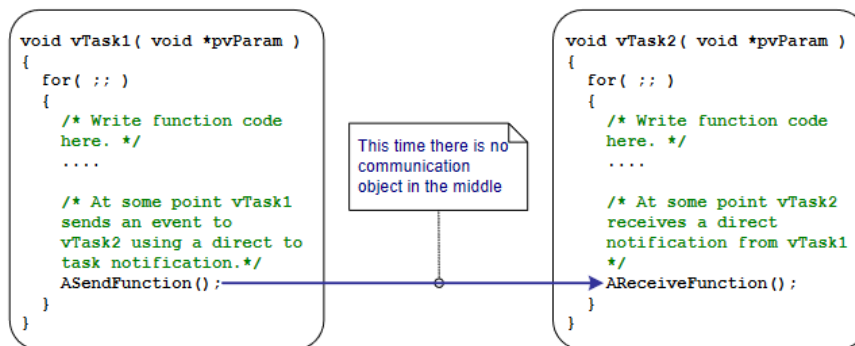


Figure 77 A task notification used to send an event directly from one task to another

Fonte: https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

Task Notifications

- Limitações
 - *Não é possível enviar uma notificação para uma interrupção. (é possível com as outras estruturas). Pode de uma interrupção para a tarefa.*
 - *Não é possível enviar a notificação para mais de uma tarefa ao mesmo tempo.*
 - *Não possui nenhum tipo de memória (Buffer de dados), igual acontece nas filas. Não permite aguardar a execução de uma notificação para enviar outra (porém, é possível fazer a contagem de notificações, e realizar o tratamento em uma tarefa).*
 - *O recurso fica bloqueado enquanto a notificação não é lida no destinatário.*
- Para utilizar, confira se `configUSE_TASK_NOTIFICATIONS` está setada em 1 no `FreeRTOSConfig.h`

Não envia valores, apenas notifica a tarefa

Normalmente é mais interessante utilizar uma fila, pois possui mais recursos

Task Notifications

- Função para notificar a tarefa

task.h

```
BaseType_t xTaskNotifyGive( TaskHandle_t xTaskToNotify );
```

- Notificar uma tarefa a partir de uma interrupção

Altera a prioridade da tarefa notificada

```
void vTaskNotifyGiveFromISR( TaskHandle_t xTaskToNotify,  
                             BaseType_t *pxHigherPriorityTaskWoken );
```

- Receber a notificação

Limpar a notificação

```
uint32_t ulTaskNotifyTake( BaseType_t xClearCountOnExit,  
                           TickType_t xTicksToWait );
```

Pode utilizar uma variável para contar o número de notificações recebidas, e utilizar false para não limpar as notificações.
Contagem = ulTaskNotifyTake(pdFALSE, portMAX_DELAY);

Task Notifications

- Exercício 1: Faça um código que tenha 2 tarefas e uma interrupção por um botão.
 - *Quando pressionar o botão (borda de descida) ele irá enviar uma notificação para a tarefa 1*
 - *Tarefa 1: Imprima "Tarefa 1: Notificação recebida pela interrupção!" quando receber a notificação, ative o LED1 e envie uma notificação para a Tarefa 2.*
 - *Tarefa 2: Imprima "Tarefa 2: Notificação recebida da tarefa 1!" quando receber a notificação e ative o LED2.*

Task Notifications

- Exercício 1:

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

```
#define LED1 7
#define LED2 8
#define BOTAO 2
```

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

```
TaskHandle_t xTarefa1Handle;
```

```
TaskHandle_t xTarefa2Handle;
```

```
/*protótipos das Tasks*/
```

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

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

```
void ISR_BOTAO(void) {
```

```
    vTaskNotifyGiveFromISR(xTarefa1Handle, pdFALSE);
```

```
}
```

```
void setup() {
```

```
    Serial.begin(9600);
```

```
    pinMode(BOTAO, INPUT_PULLUP);
```

```
    attachInterrupt(digitalPinToInterrupt(BOTAO), ISR_BOTAO, FALLING);
```

```
    xTaskCreate(vTarefa1, "vTarefa1", 256, NULL, 1, &xTarefa1Handle);
```

```
    xTaskCreate(vTarefa2, "vTarefa2", 256, NULL, 1, &xTarefa2Handle);
```

```
}
```

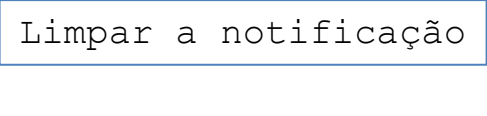
Não altera a prioridade da tarefa que recebe a notificação



Task Notifications

■ Exercício 1:

```
void loop() {  
  //As funções são executadas nas tarefas  
}  
  
void vTarefa1(void *pvParameters){  
  pinMode(LED1, OUTPUT);  
  while(1)  
  {  
    ulTaskNotifyTake(pdTRUE, portMAX_DELAY);  
    Serial.println("Tarefa 1: Notificação recebida do botão!");  
    digitalWrite(LED1,HIGH);  
    vTaskDelay(pdMS_TO_TICKS(500));  
    Serial.println("Tarefa 1: Enviando notificação para a tarefa 2!");  
    vTaskDelay(pdMS_TO_TICKS(500));  
    xTaskNotifyGive(xTarefa2Handle);  
  }  
}  
  
void vTarefa2(void *pvParameters){  
  pinMode(LED2, OUTPUT);  
  while(1)  
  {  
    ulTaskNotifyTake(pdTRUE, portMAX_DELAY);  
    Serial.println("Tarefa 2: Notificação recebida da tarefa 1!");  
    digitalWrite(LED2,HIGH);  
  }  
}
```



- Exercício 2: Faça um código que tenha 2 tarefas:
 - *Tarefa 1: Faz a leitura de um botão (sem usar interrupção) com debounce. No momento em que ele é pressionado (nível LOW), imprima "Tarefa 1: Botão Pressionado!" e envie uma notificação para a tarefa 2. A notificação deve ser enviada apenas 1 vez para cada vez que o botão é pressionado.*
 - *Tarefa 2: Imprima "Tarefa 2: Processando notificação: (notificação atual)" a cada 1 segundo quando receber a notificação e inverta o valor do LED1. Sendo que: (notificação atual) é a notificação relativa ao pressionamento do botão, por exemplo: se pressionar o botão 5 vezes em 1 segundo, terão 5 notificações para serem processadas.*
 - *Dica 1: Não limpe a notificação ao ler a quantidade atual.*

Task Notifications


■ Exercício 2:

```
void setup() {
    Serial.begin(9600);
    pinMode(BOTAO, INPUT_PULLUP);
    xTaskCreate(vTarefa1, "vTarefa1", 128, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "vTarefa2", 128, NULL, 1, &xTarefa2Handle);
    //Dependendo como colocar as prioridades, nao vai funcionar
}
```

```
void vTarefa1(void *pvParameters)
{
    uint8_t debouncingContagem = 0;
    uint8_t x = 0;
    while (1)
    {
        if(digitalRead(BOTAO) == LOW && x==0)
        {
            debouncingContagem++;
            if(debouncingContagem >= 5){
                debouncingContagem = 0;
                Serial.println("Tarefa 1: Botão Pressionado!");
                xTaskNotifyGive(xTarefa2Handle);
                x=1;
            }
            vTaskDelay(pdMS_TO_TICKS(10));
        }
        else
        {
            debouncingContagem = 0;
        }
        if(digitalRead(BOTAO) == HIGH){
            x=0;
        }
    }
}
```

```
void vTarefa2(void *pvParameters){
    pinMode(LED1, OUTPUT);
    uint8_t cont_notificacao= 0;
    while(1)
    {
        cont_notificacao = ulTaskNotifyTake(pdFALSE, portMAX_DELAY);
        Serial.println("Tarefa 2: Processando notificação: " + String(cont_notificacao));
        digitalWrite(LED1, !digitalRead(LED1));
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

Não limpa a notificação



- Exercício 3: Faça um código que tenha 2 tarefas e uma interrupção:
 - *Quando pressionar o botão, ele irá notificar a tarefa 1.*
 - *Tarefa 1: Faz a leitura do número de vezes que foi notificada, imprime o número de notificações e, conforme tiver notificações disponíveis, notifica a tarefa 2 a cada 1 segundo.*
 - *Tarefa 2: Quando receber uma notificação, imprima "Tarefa 2: Processando notificação: (notificação atual)" a cada 1 segundo e mantenha o LED ligado enquanto tiver notificações sendo executadas.*
 - *Dica 1: Não limpe a notificação ao ler a quantidade atual.*
 - *Dica 2: O controle de tempo fica a cargo da tarefa 1.*

Task Notifications

- Exercício 3:

```
void ISR_BOTAO(void) {
    vTaskNotifyGiveFromISR(xTarefa1Handle, pdTRUE);
    portYIELD_FROM_ISR();
}

void setup() {
    Serial.begin(9600);
    pinMode(BOTAO, INPUT_PULLUP);
    pinMode(LED1, OUTPUT);
    attachInterrupt(digitalPinToInterrupt(BOTAO), ISR_BOTAO, FALLING);
    xTaskCreate(vTarefa1, "vTarefa1", 128, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "vTarefa2", 128, NULL, 1, &xTarefa2Handle);
    //Dependendo como colocar as prioridades, nao vai funcionar
}
```

Task Notifications

■ Exercício 3:

```
uint8_t cont_notificacao= 0;
void vTarefa1(void *pvParameters)
{
    while (1)
    {
        cont_notificacao = ulTaskNotifyTake(pdFALSE, portMAX_DELAY);
        Serial.println("Tarefa 1: Notificações recebidas: " + String(cont_notificacao));
        Serial.println("Enviando notificação para a tarefa 2...");
        xTaskNotifyGive(xTarefa2Handle);
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void vTarefa2(void *pvParameters){
    while(1)
    {
        if(ulTaskNotifyTake(pdTRUE, pdMS_TO_TICKS(1000))==pdTRUE){
            digitalWrite(LED1,HIGH);
            Serial.println("Tarefa 2: Processando notificação: " + String(cont_notificacao));
        }
        else
        {digitalWrite(LED1,LOW);}
    }
}
```

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/>