

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

Sistemas Embarcados

FREERTOS

- Softwares Timers -

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/)

- São utilizados quando é necessário agendar a execução de uma tarefa em um tempo definido ou de forma periódica.
- A função executada pelo terminal é chamada de função call-back do software timer.
- Os softwares timers são implementados e controlados pelo FreeRTOS.
 - *Não fazem parte dos timers/contadores que os microcontroladores possuem implementados em hardware.*
 - *Podem ser criados diversos timers, o limite é a memória disponível.*

- O software timer não usa tick do hardware e não executa as funções de callback do timer em contexto de interrupção.
- Todos os comandos do timer são enviados à tarefa “Timer Service ou Daemon task” por uma queue.
- Desvantagens:
 - *Latência varia de acordo com a prioridade da tarefa “Timer Service” e frequência do FreeRTOS, ambos configuráveis.*
 - *Limitação em 1kHz*
 - *Pode perder comandos se usado excessivamente durante um período curto de tempo, pois a comunicação com a tarefa “Timer Service” é feita por Queue que pode ficar lotada.*

- Em algumas versões do FreeRTOS o software Timers pode vir desabilitado por padrão, para habilitá-lo, procure pelo arquivo FreeRTOSConfig.h e habilite em configUSE_TIMERS.

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

- *Configure a prioridade do timer em: configTimer_TASK_PRIORITY*
 - Se comporta como se fosse uma tarefa.
- *Configure o tamanho da fila (Queue) em: configTIMER_QUEUE_LENGTH*
- *Configure o tamanho da stack da timer em: configTIMER_TASK_STACK_DEPTH*
- *É habilitado por padrão para o Arduino.*
- *Configuração padrão para o FreeRTOS para Arduino*

```
/* Timer definitions. */  
#define configUSE_TIMERS 1  
#define configTIMER_TASK_PRIORITY configMAX_PRIORITIES-1  
#define configTIMER_QUEUE_LENGTH ( 10 )  
#define configTIMER_TASK_STACK_DEPTH ( 85 )
```

- Tipos de Softwares Timers
 - *One-Shot*
 - *Auto Reload*

■ Criando um timer

<https://www.freertos.org/FreeRTOS-timers-xTimerCreate.html>

timers.h

```
TimerHandle_t xTimerCreate
(
    const char * const pcTimerName,
    const TickType_t xTimerPeriod,
    const UBaseType_t uxAutoReload,
    void * const pvTimerID,
    TimerCallbackFunction_t pxCallbackFunction );

xTimers = xTimerCreate
(
    /* Just a text name, not used by the RTOS
    kernel. */
    "Timer",
    /* The timer period in ticks, must be
    greater than 0. */
    pdMS_TO_TICKS(500),
    /* The timers will auto-reload themselves
    when they expire. */
    pdTRUE,
    /* The ID is used to store a count of the
    number of times the timer has expired, which
    is initialised to 0. */
    ( void * ) 0,
    /* Each timer calls the same callback when
    it expires. */
    vTimerCallback
);
```

pdTRUE = Auto reload
pdFALSE = One-Shot

Função de tratamento

- Iniciando o timer

```
BaseType_t xTimerStart( TimerHandle_t xTimer,  
                        TickType_t xBlockTime );
```

Handle do timer

Tempo de espera para iniciar (em ticks)

`xTimerStart (xTimer, 0);`



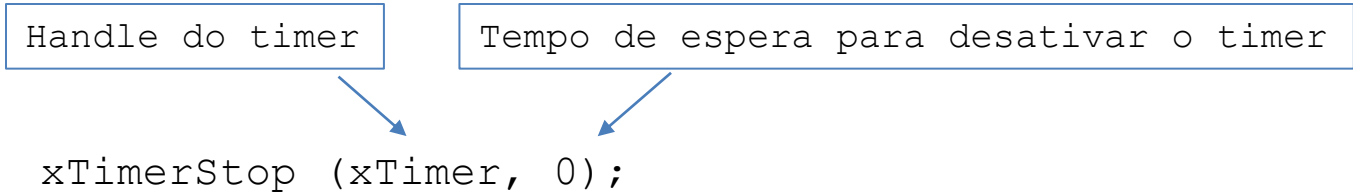
- Parando o timer

```
BaseType_t xTimerStop( TimerHandle_t xTimer,  
                       TickType_t xBlockTime );
```

Handle do timer

Tempo de espera para desativar o timer

`xTimerStop (xTimer, 0);`



- Deletar o timer – para liberar memória

```
BaseType_t xTimerDelete( TimerHandle_t xTimer,  
                          TickType_t xBlockTime );
```

Handle do timer

Tempo de espera para deletar

`xTimerDelete (xTimer, 0);`



- Exemplo 1: Faça um código que desempenhe as seguintes funções:
 - *Timer 1: Tenha um LED piscando em 2Hz com um timer auto-reload.*
 - *Timer 2: Tenha um LED que pisque 10 vezes em 1Hz.*
 - O código não terá nenhuma tarefa.

```
#include "Arduino_FreeRTOS.h"
#include "timers.h"

#define LED1 7
#define LED2 8

/* Variáveis para armazenamento do handle das tasks */

/* Variáveis para armazenamento do handle dos Timers */
TimerHandle_t xTimer1, xTimer2;

/*protótipos das Tasks e Timers*/
void funcaoTimer1(TimerHandle_t xTimer1);
void funcaoTimer2(TimerHandle_t xTimer2);

void setup() {
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    xTimer1 = xTimerCreate("TIMER1", pdMS_TO_TICKS(250), pdTRUE, 0, funcaoTimer1);
    xTimer2 = xTimerCreate("TIMER2", pdMS_TO_TICKS(500), pdFALSE, 0, funcaoTimer2);
    xTimerStart(xTimer1, 0);
    xTimerStart(xTimer2, 0);
}
```

```
void funcaoTimer1(TimerHandle_t xTimer1)
{
    digitalWrite(LED1, !digitalRead(LED1));
}

int contagem=0;
void funcaoTimer2(TimerHandle_t xTimer2)
{
    digitalWrite(LED2, !digitalRead(LED2));
    contagem++;
    if (contagem < 20){
        xTimerStart(xTimer2,0);
    }
}
```

- Exercício 1: Faça um código que desempenhe as seguintes funções:
 - *Timer 1: Tenha um LED piscando inicialmente em 1Hz.*
 - *Quando um botão for pressionado (interrupção) ele aumente a frequência do LED para 5Hz. Se for pressionado novamente, volta para 1Hz.*
 - *Não tem tarefas.*
 - *Consulte o manual:*

<https://www.freertos.org/FreeRTOS-Software-Timer-API-Functions.html>

Software Timers

```
#include "Arduino_FreeRTOS.h"
#include "timers.h"

#define LED1 7
#define BOTAO 2

/* Variáveis para armazenamento do handle dos Timers */
TimerHandle_t xTimer1;

/*protótipos das Tasks e Timers*/
void funcaoTimer1(TimerHandle_t xTimer1);


bool x;
void funcaoISR() {
    if(x){xTimerChangePeriodFromISR(xTimer1, pdMS_TO_TICKS(100), NULL);}
    else{xTimerChangePeriodFromISR(xTimer1, pdMS_TO_TICKS(500), NULL);}
    x=!x;
}

void setup() {
    pinMode(LED1, OUTPUT);
    pinMode(BOTAO, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(BOTAO), funcaoISR, FALLING);
    xTimer1 = xTimerCreate("TIMER1", pdMS_TO_TICKS(500), pdTRUE, 0, funcaoTimer1);
    xTimerStart(xTimer1, 0);
}

void funcaoTimer1(TimerHandle_t xTimer1)
{
    digitalWrite(LED1, !digitalRead(LED1));
}

void loop() {
    //As funções são executadas nas tarefas
}
```


Colocando pdTRUE ele executa rapidamente o timer
service/daemon task



- Exercício 2: Faça um código que desempenhe as seguintes funções:
 - Timer 1: Tenha um LED piscando inicialmente em 1Hz.*
 - Tenha uma tarefa que faça a leitura da serial e altere a frequência do LED do timer 1 de acordo com o valor digitado (entre 1 e 10 Hz).*

```
void vTarefa1(void *pvParameters){
    int valor;
    Serial.println("Digite um valor para a frequência entre 1 e 10Hz");
    while(1)
    {
        if (Serial.available()>0){
            valor = Serial.parseInt();
            if(valor>10)valor = 10;
            if(valor<1)valor = 1;
            Serial.println("Frequência digitada: " +String(valor));
            valor = 1000/(valor*2);
            xTimerChangePeriod(xTimer1, pdMS_TO_TICKS(valor), pdMS_TO_TICKS(50));
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

Aguarda um tempo até o período ser alterado



- Exercício 3: Com base no exercício 2, faça um código que desempenhe as seguintes funções:
 - *Timer 1: Tenha um LED piscando inicialmente em 1Hz.*
 - *Tarefa 1: Faça a leitura da serial para controlar a frequência do LED do timer 1 de acordo com o valor digitado (entre 1 e 10 Hz). O valor deve ser enviado para a tarefa 2 através de uma fila (Queue).*
 - *Tarefa 2: Faça a leitura da fila e altere a frequência do LED.*

Software Timers

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

#define LED1 7

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t xTarefa1Handle = NULL;
TaskHandle_t xTarefa2Handle = NULL;

/* Variáveis para armazenamento do handle dos Timers */
TimerHandle_t xTimer1;

/* Variáveis para armazenamento do handle das filas */
QueueHandle_t xFila;

/*protótipos das Tasks e Timers*/
void funcaoTimer1(TimerHandle_t xTimer1);
void vTarefa1(void *pvParameters);
void vTarefa2(void *pvParameters);

void setup() {
    Serial.begin(9600);
    Serial.setTimeout(50);
    pinMode(LED1, OUTPUT);
    xFila = xQueueCreate(5, sizeof(int));
    xTimer1 = xTimerCreate("TIMER1", pdMS_TO_TICKS(250), pdTRUE, 0, funcaoTimer1);
    xTaskCreate(vTarefa1, "Tarefa1", 128, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "Tarefa2", 128, NULL, 1, &xTarefa2Handle);
    xTimerStart(xTimer1, 0);
}
```

Você vai perceber que vai demorar um tempo para atualizar a serial e mudar o frequência, isso se deve pelo fato da função `Serial.parseInt` ter um timeout de 1 segundo. Para reduzir, acrescente essa função e coloque um timeout menor. Cuidado para ter tempo suficiente para fazer a leitura, na velocidade de 9600, demora em torno de 1ms por caractere.

Software Timers

```
void funcaoTimer1(TimerHandle_t xTimer1)
{
    digitalWrite(LED1, !digitalRead(LED1));
}

void loop() {
    //As funções são executadas nas tarefas
}

void vTarefa1(void *pvParameters){
    int valor;
    Serial.println("Digite um valor para a frequência entre 1 e 10Hz");
    while(1)
    {
        if (Serial.available()>0){
            valor = Serial.parseInt();
            if(valor>10)valor = 10;
            if(valor<1)valor = 1;
            Serial.println("Frequência digitada: " +String(valor));
            valor = 1000/(valor*2);
            xQueueSend(xFila, &valor, portMAX_DELAY);
        }
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

void vTarefa2(void *pvParameters){
    int valor_freq;
    while(1)
    {
        if(xQueueReceive(xFila,&valor_freq, pdMS_TO_TICKS(500))==pdTRUE){
            Serial.println("Informação recebida na tarefa2, atualizando frequência...");
            xTimerChangePeriod(xTimer1, pdMS_TO_TICKS(valor_freq), pdMS_TO_TICKS(50));
        }
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
```


- Exercício 4: Faça um código que desempenhe as seguintes funções:
 - *Que tenha uma tarefa que faça a leitura de um botão, e assim que o botão for pressionado, realize a aquisição de 5 amostras por segundo (5 Hz) de uma entrada analógica e imprima na serial. O controle da frequência de aquisição deve ser feita por um software Timer one-shot. Enquanto a aquisição estiver sendo feita, o LED2 deve ficar ligado.*
 - *Que tenha um LED (LED1) que fique piscando em 0,5 Hz, utilizando um software Timer auto reload.*

Software Timers

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

#define LED1 7
#define LED2 8
#define ENTRADA A0
#define BOTAO 2

TaskHandle_t xTarefaADHandle;
TimerHandle_t xTimer1, xTimer2;

void vTarefaAD(void *pvParametes);

void funcaoTimer1(TimerHandle_t xTimer1);
void funcaoTimer2(TimerHandle_t xTimer2);

void setup()
{
    Serial.begin(9600);
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    pinMode(BOTAO, INPUT_PULLUP);
    xTimer1 = xTimerCreate("TIMER1", pdMS_TO_TICKS(1000), pdTRUE, 0, funcaoTimer1);
    xTimer2 = xTimerCreate("TIMER2", pdMS_TO_TICKS(200), pdFALSE, 0, funcaoTimer2);

    xTaskCreate(vTarefaAD, "TASK1", 256, NULL, 1, &xTarefaADHandle);
    xTimerStart(xTimer1, 0);
}
```

0,5 Hz
5 Hz

pdTRUE = Auto reload
pdFALSE = One-Shot

Inicia o Timer1 (LED em 0,5 Hz)

Software Timers

```
void vTarefaAD(void *pvParameters)
{
    while (1)
    {
        if((digitalRead(BOTAO) == LOW) && (xTimerIsTimerActive(xTimer2) == pdFALSE))
        {
            digitalWrite(LED2,HIGH);
            xTimerStart(xTimer2,0);
            Serial.println("Iniciando Aquisições...");
        }
    }
}
```

Entra na rotina apenas quando não estiver aquisições sendo feitas

Ativa o Timer2, que faz as aquisições

Funções chamadas pelos Timers

```
void funcaoTimer1(TimerHandle_t xTimer1)
{
    digitalWrite(LED1, !digitalRead(LED1));
}

uint8_t contagem=0;
void funcaoTimer2(TimerHandle_t xTimer2)
{
    contagem++;
    Serial.println("ValorAD: " + String(analogRead(ENTRADA)));
    if (contagem<5){
        xTimerStart(xTimer2,0);
    }
    else{
        digitalWrite(LED2, LOW);
        contagem=0;
    }
}
```

- DENARDIN, G. W.; BARRIQUELLO, C. H. Sistemas Operacionais de Tempo Real e sua Aplicação em Sistemas Embarcados. 1ª edição, São Paulo, Blucher, 2019.
- DENARDIN, G. W. Notas de aula de sistemas embarcados. UTFPR.
- STALLINGS, William. Operating systems: internals and design principles. 5.ed. Upper Saddle River: Pearson Prentice Hall. 2004.
- TANENBAUM, Andrew. Sistemas operacionais modernos. Rio de Janeiro: LTC. 1999.
- BACURAU, R.M. Notas de aulas de projeto de sistemas embarcados. UNICAMP, 2020. Disponível em: <https://sites.google.com/site/rodrigobacurau/cursos-2020-1/es670---projeto-de-sistemas-embarcados>
- <https://www.embarcados.com.br/>

- FREERTOS - Event Groups