



# Modelos de Multitarefa em sistemas embarcados

Aula síncrona (29/04/2025)

Prof. Wilton Lacerda Silva

Executores:



Coordenação:



Iniciativa:



# Sumário

- Objetivos
- Conceitos de aplicação Bare metal
- Conceitos de Sistema operacional de tempo real
- Sistemas multitarefa
- Instalação e configuração do FreeRTOS
- Exemplos de aplicação utilizando o FreeRTOS.

# Objetivos

- Compreender os conceitos de multitarefa, suas vantagens e aplicações.
- Configurar o ambiente de Visual Studio Code para operar com o FreeRTOS.
- Desenvolver alguns exemplos para fixação dos conceitos de sistemas operacionais em tempo real.



## Tarefas em aplicações embarcadas

No contexto de computação embarcada, o conceito de tarefa assume características bem específicas, pois esses sistemas geralmente operam com recursos limitados (como CPU, memória e energia) e precisam atender a restrições temporais rígidas para garantir a funcionalidade correta e em tempo hábil.

## Tarefa em Computação Embarcada

Uma tarefa em sistemas embarcados é uma unidade de trabalho responsável por realizar uma operação específica ou atender a um objetivo relacionado ao funcionamento do dispositivo. Exemplos incluem:

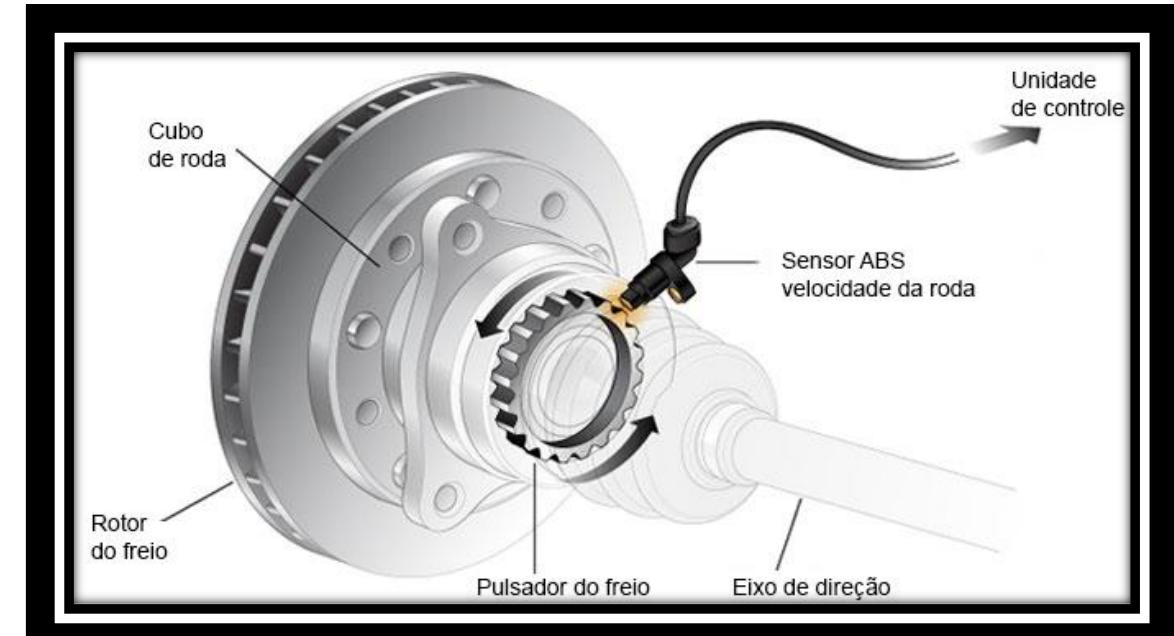
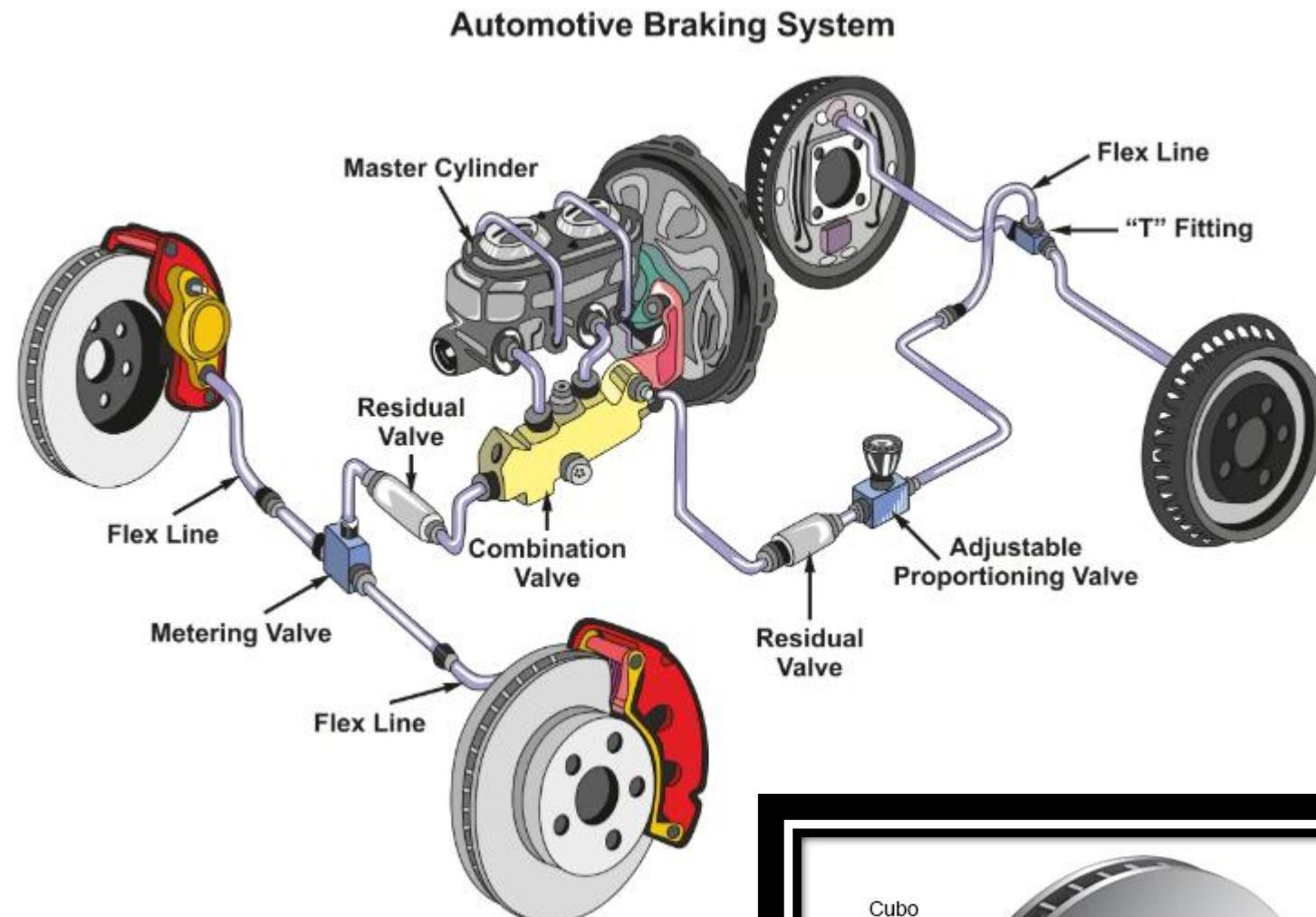
- Leitura de sensores.
- Controle de atuadores (ex.: motores, válvulas).
- Comunicação com outros dispositivos.
- Processamento de sinais ou dados capturados.

## Características das Tarefas em Sistemas Embarcados

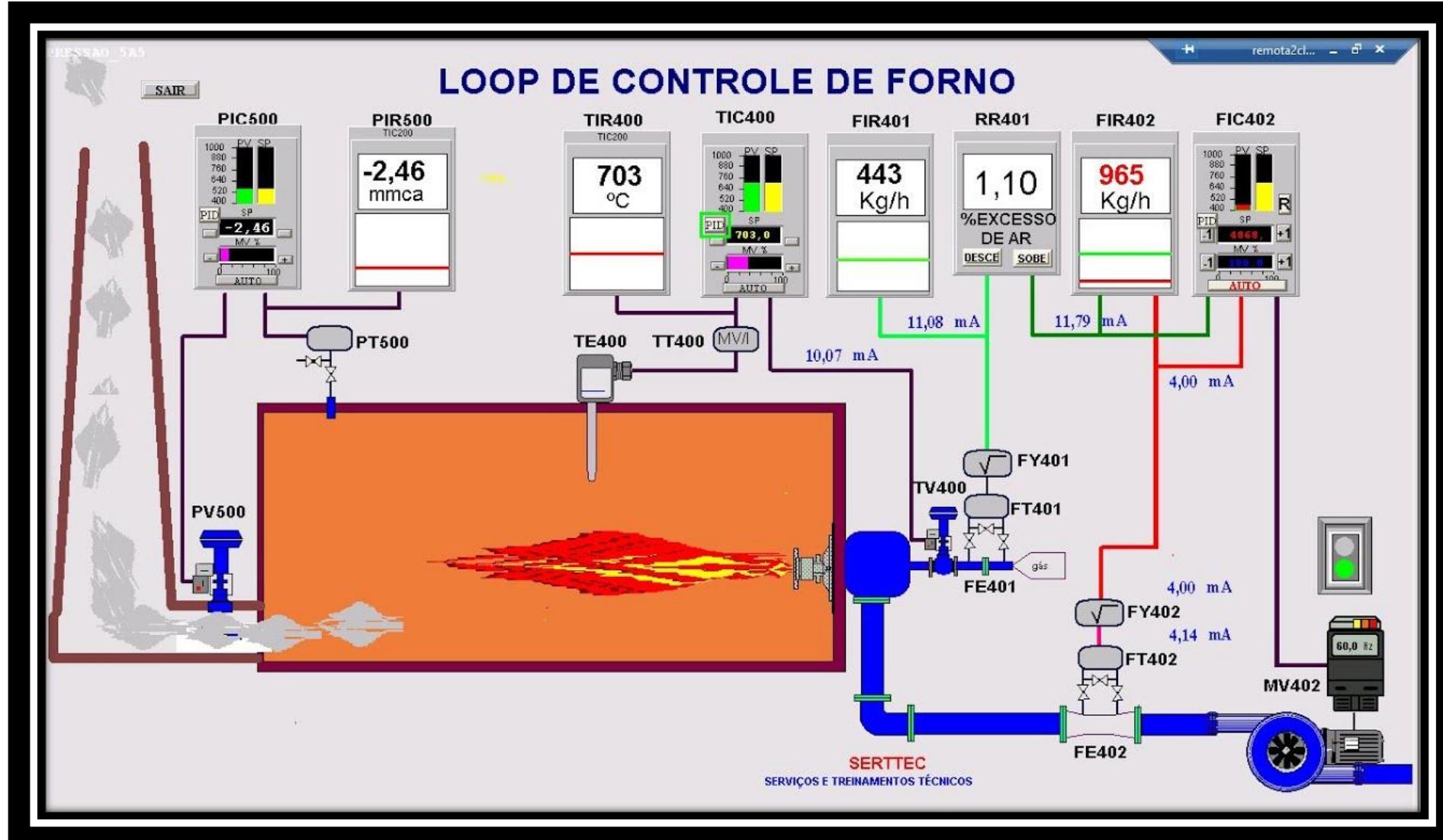
### 1. Restrições de Tempo (Tempo Real):

Muitas aplicações em sistemas embarcados implementam tarefas de tempo real, ou seja, **devem ser concluídas dentro de prazos definidos**. Elas podem ser classificadas como:

- Críticas (Hard Real-Time): Uma falha em atender ao prazo pode levar a danos ou falhas catastróficas (ex.: sistemas de frenagem em carros).
- Não-críticas (Soft Real-Time): Uma falha em atender ao prazo pode degradar o desempenho, mas sem consequências graves (ex.: atualização de uma interface de usuário).



## Características das Tarefas em Sistemas Embarcados



### 2. Prioridade:

A prioridade das tarefas permite garantir que tarefas mais importantes sejam concluídas a tempo. Por exemplo:

- Alta prioridade: Controle de temperatura em um forno industrial.
- Baixa prioridade: Registro de logs em memória.

### 3. Uso Eficiente de Recursos:

Como os recursos (CPU, memória, energia) são limitados, o sistema precisa gerenciar as tarefas de maneira eficiente, evitando desperdício e conflitos.

# Modelos de execução multitarefa

## Laço Único (Loop Único ou Laço Principal)

- Nesse modelo, todas as tarefas são executadas **sequencialmente dentro de um único laço**.
- Não há preempção automática: uma tarefa só cede o controle para a próxima quando termina sua execução ou ativamente retorna o fluxo ao laço principal.
- Isso caracteriza **multitarefa cooperativa**, pois as **tarefas precisam ser bem projetadas para não bloquear o sistema** e permitir a execução das demais.

## Laço com Interrupções

- Nesse modelo, o sistema conta com **interrupções de hardware**, que podem interromper a execução do laço principal para atender eventos externos (como pressionamento de botões, recebimento de dados, etc.).
- No entanto, o laço principal **ainda executa tarefas de forma cooperativa**, pois não há um **escalonador preemptivo**.
- A execução de uma tarefa só ocorre se o fluxo do programa permitir.

## Multitarefa Cooperativa vs. Preemptiva

- **Multitarefa Cooperativa:** As tarefas precisam liberar voluntariamente a CPU.
- **Multitarefa Preemptiva:** Um escalonador ou sistema operacional **interrompe e alterna entre tarefas automaticamente**.

# Definições

No desenvolvimento de sistemas embarcados, compreender a diferença entre programação **bare metal** e o uso de um sistema operacional em tempo real (RTOS) é fundamental para o projeto de sistemas com alto desempenho. A abordagem **bare metal** envolve a programação direta sobre o hardware, sem um sistema operacional intermediário. Isso permite controle total dos recursos da máquina e pode resultar em aplicações extremamente otimizadas para tarefas específicas, com baixo consumo de memória e mínima latência. No entanto, essa abordagem exige maior complexidade no gerenciamento manual de tarefas, temporização e eventos.

Por outro lado, um RTOS oferece uma estrutura mais organizada para lidar com múltiplas tarefas concorrentes, fornecendo mecanismos como **escalonamento, semáforos, filas e temporizadores**. Isso facilita o desenvolvimento de aplicações que exigem resposta determinística a eventos em tempo real, melhorando a modularidade e a manutenção do código, especialmente em projetos maiores ou mais complexos.

O **FreeRTOS** é um sistema operacional de tempo real (RTOS - *Real-Time Operating System*) leve e de código aberto, amplamente utilizado em sistemas embarcados. Ele fornece um **agendador de tarefas preemptivo** que permite executar múltiplas tarefas de forma concorrente, além de recursos como **semáforos, filas, timers e mutexes** para gerenciamento de sincronização e comunicação entre tarefas.



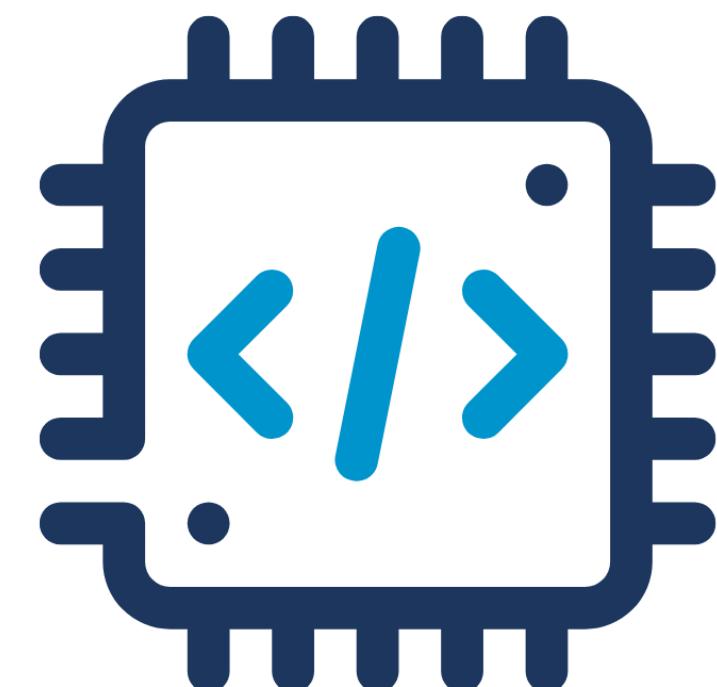
# FreeRTOS: Definições

O termo "**bare metal**" (metal nu) refere-se à programação direta do hardware **sem o uso de um sistema operacional**. Isso significa que o código é escrito para interagir diretamente com os registradores e periféricos do microcontrolador, sem a intermediação de um sistema operacional como Linux, FreeRTOS ou qualquer outro.

## Bare-Metal PROGRAMMING LANGUAGES



IN Tech House

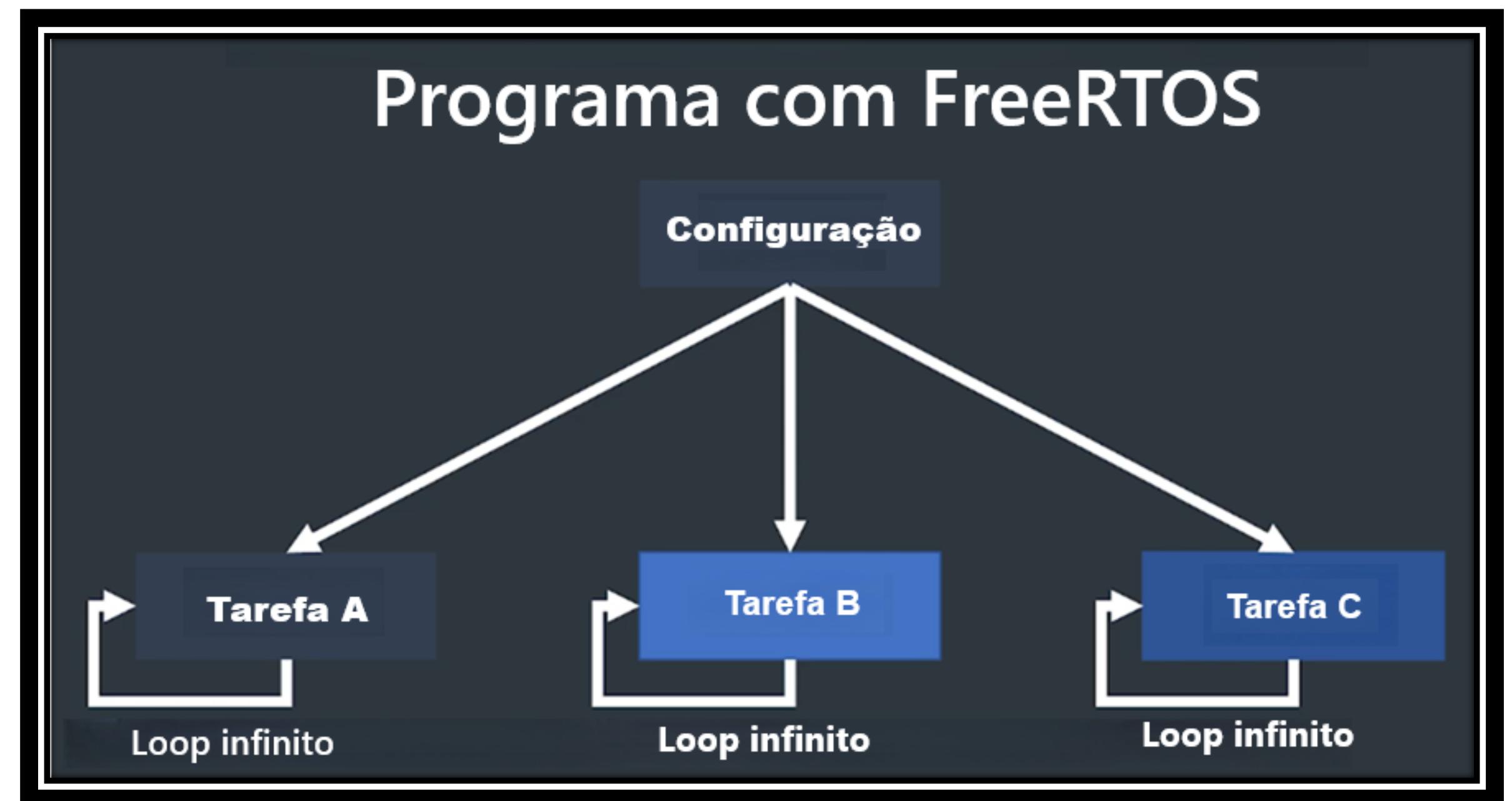
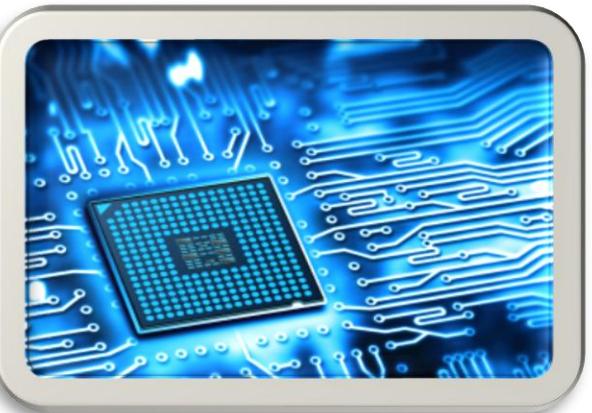
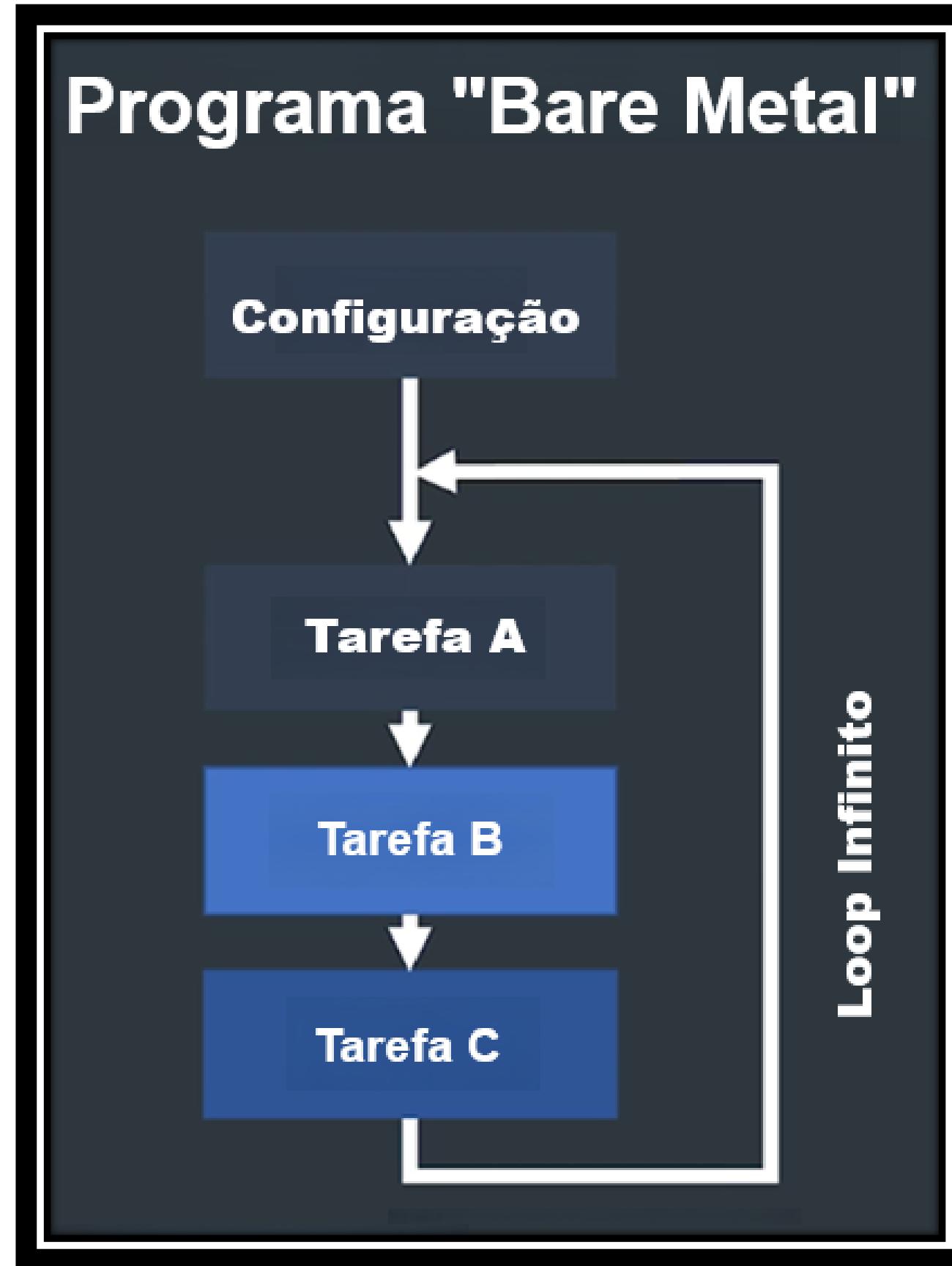


**ASSEMBLY**  
**C**    **C++**

Bare metal programming, which involves programming directly on the hardware without an operating system, typically utilizes the following languages:

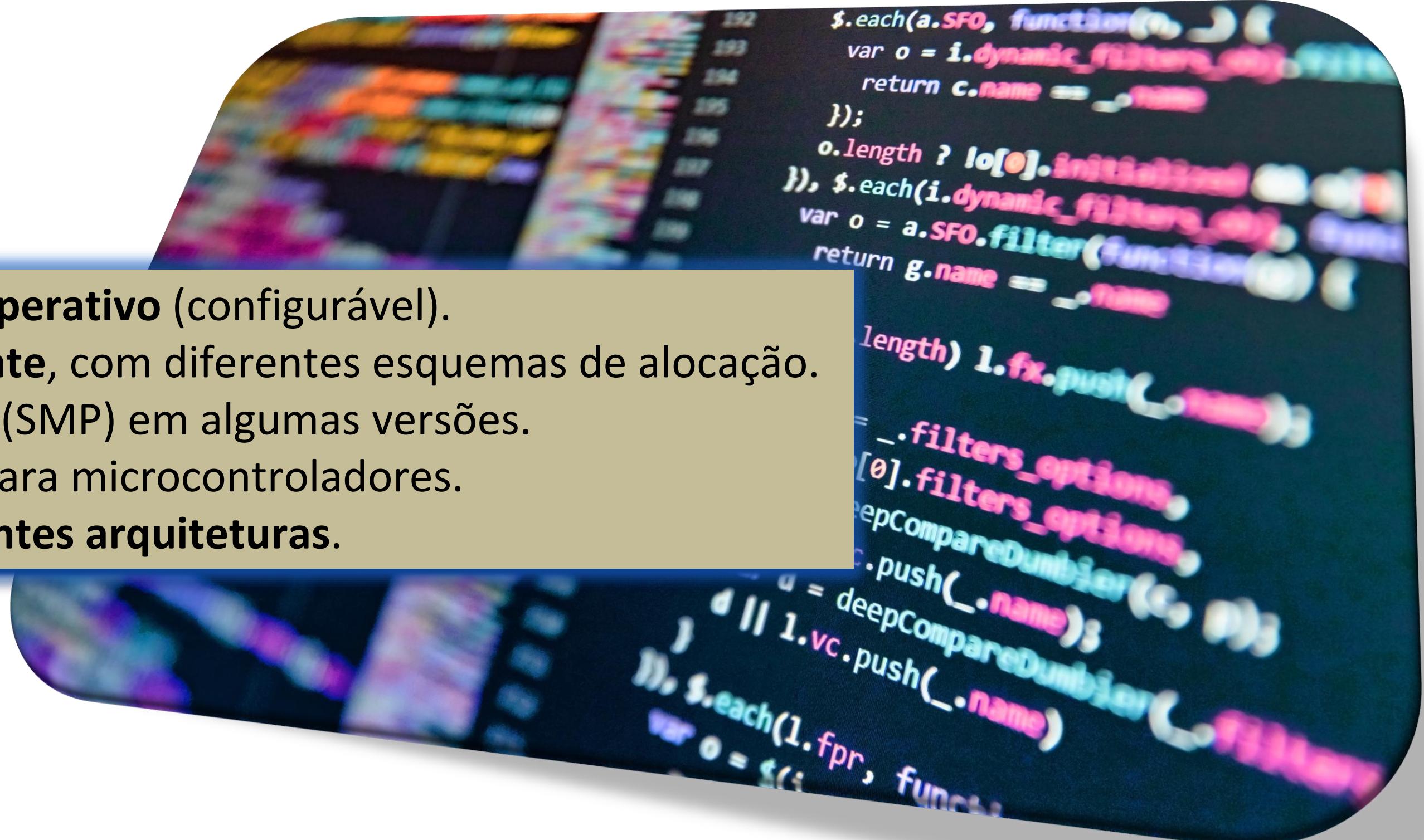
<https://intechhouse.com/blog/what-is-bare-metal-programming-in-embedded-system/>

# Comparação entre a execução no bare metal e no FreeRTOS



# Principais Características do FreeRTOS

- Escalonamento preemptivo ou cooperativo (configurável).
- Gerenciamento de memória eficiente, com diferentes esquemas de alocação.
- Suporte a múltiplos processadores (SMP) em algumas versões.
- Baixo consumo de recursos, ideal para microcontroladores.
- Ampla compatibilidade com diferentes arquiteturas.



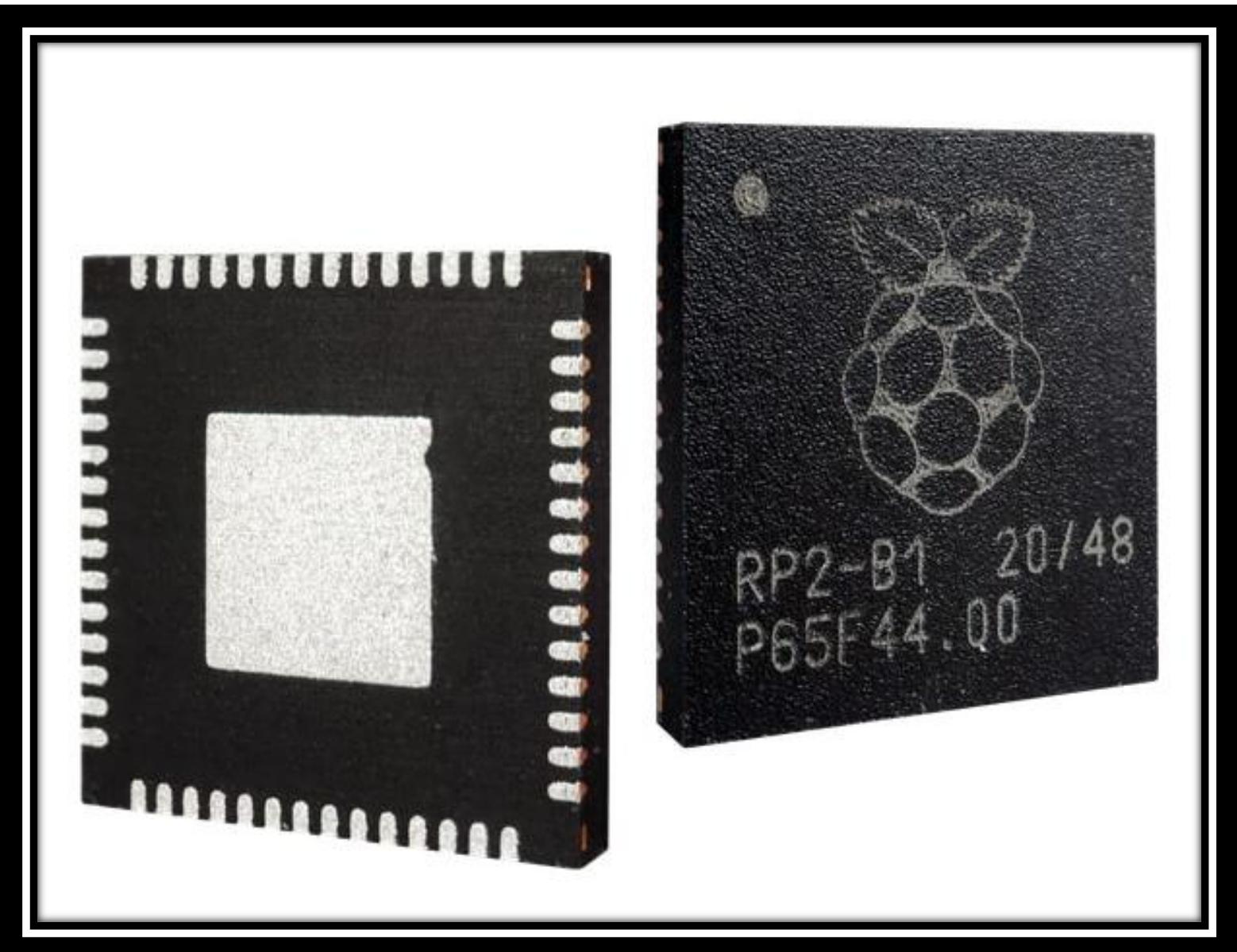
# FreeRTOS no RP2040

## Pré-requisitos

Antes de começar, certifique-se de que você tem instalado:

- **VS Code** (editor)
- **CMake** (3.13 ou superior)
- **Ninja** ou **Make** (para build)
- **Compilador** que suporte ARM
- **Pico SDK** (já baixado e funcionando)
- **Python 3** (para o CMake detectar o SDK)

Se você já consegue compilar um "blink" tradicional do RP2040, já está pronto.



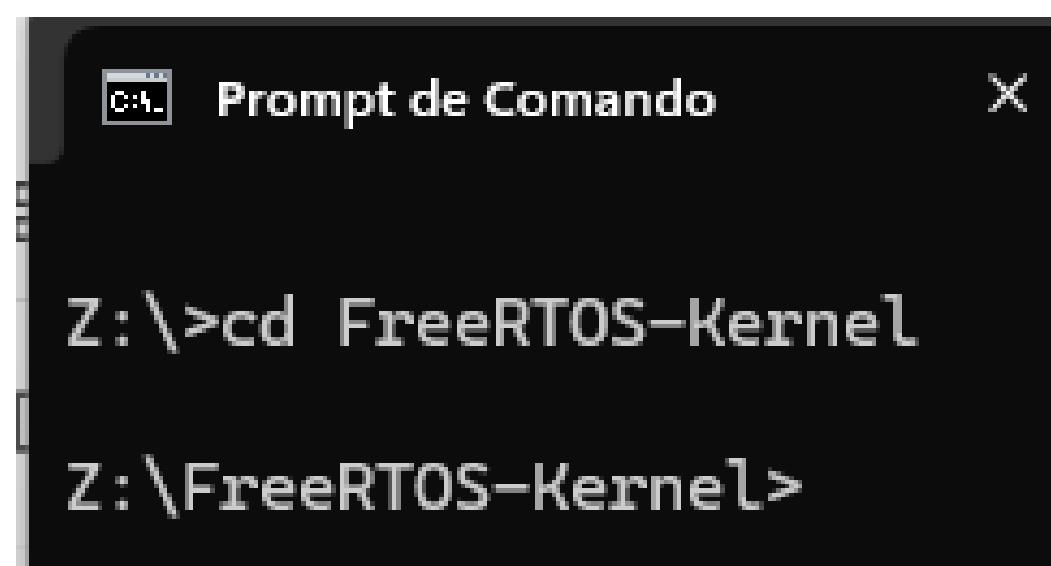
# FreeRTOS no RP2040

## Baixar o FreeRTOS

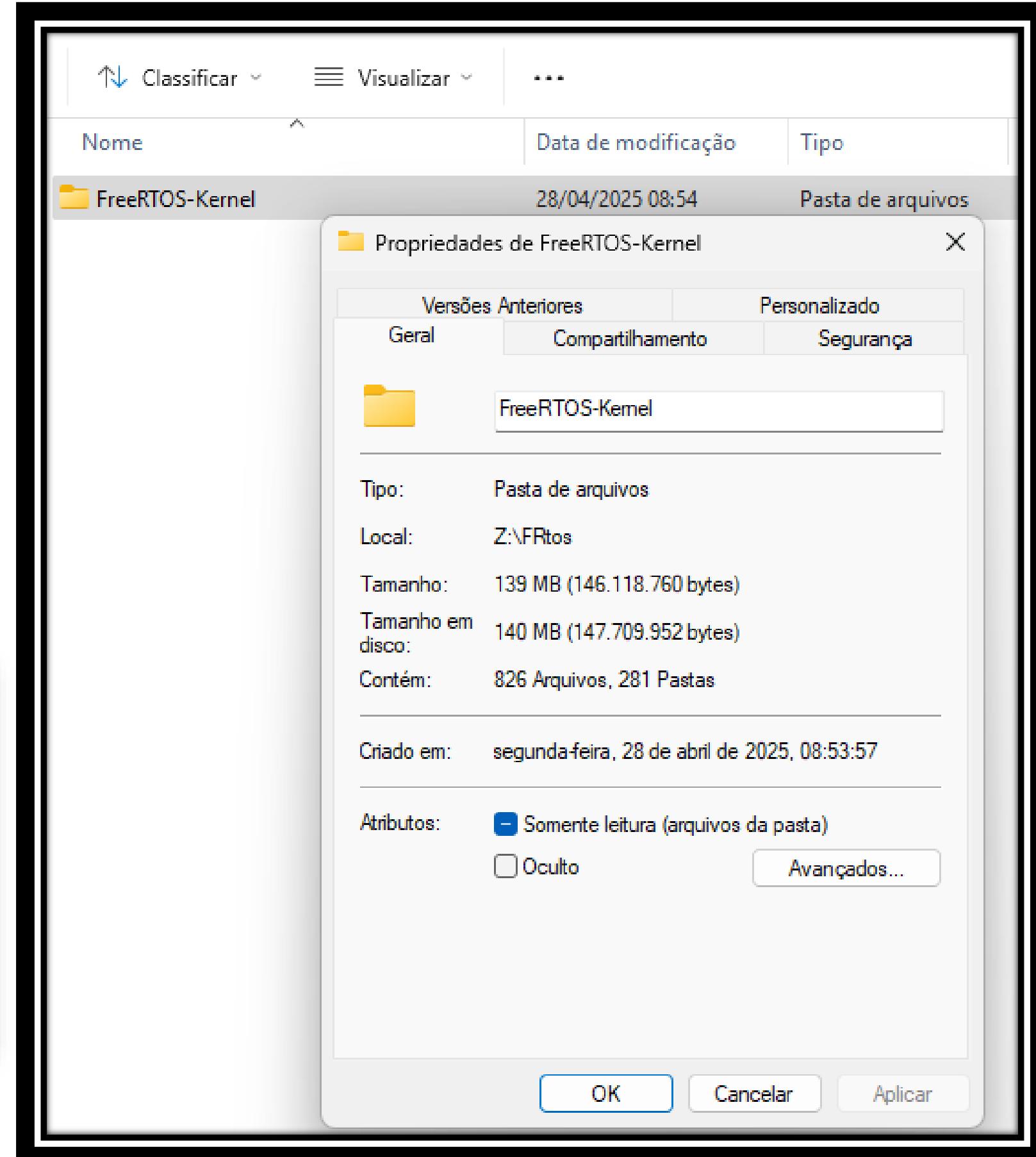
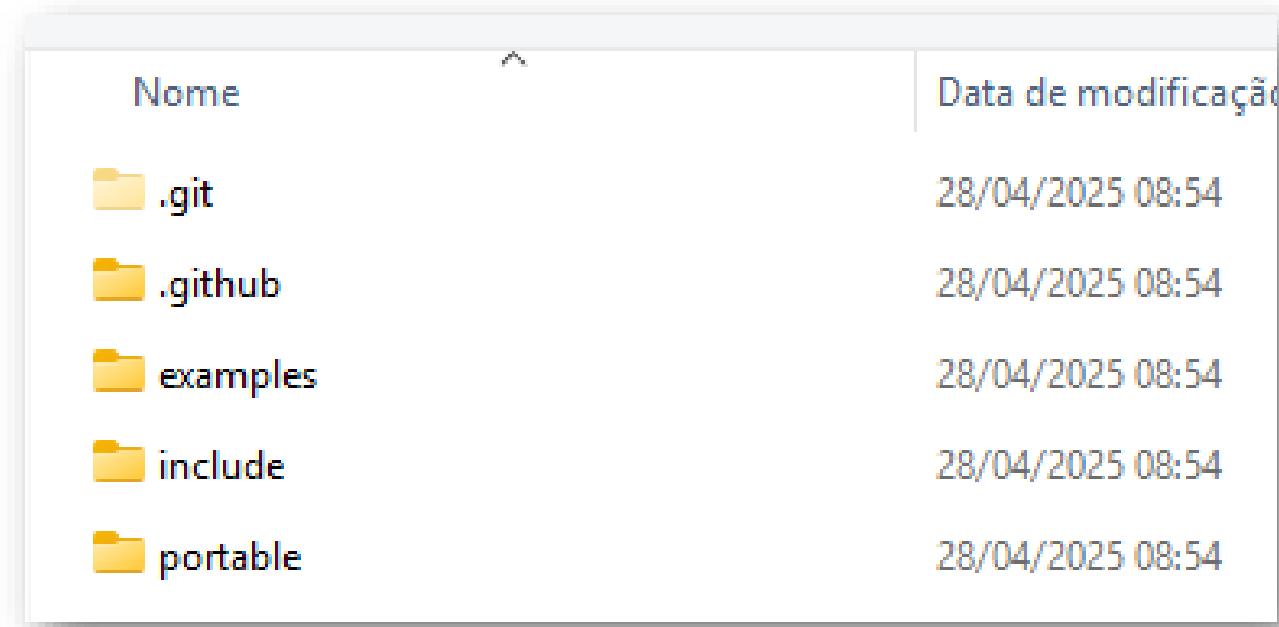
```
git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```



```
Z:\>git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```



```
Z:\>cd FreeRTOS-Kernel
Z:\FreeRTOS-Kernel>
```



```
M CMakeLists.txt
18 cmake_minimum_required(VERSION 3.13)
19 set(CMAKE_C_STANDARD 11)
20 set(CMAKE_CXX_STANDARD 17)
21 set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
22 set(PICO_BOARD pico_w CACHE STRING "Board type")
23 include(pico_sdk_import.cmake)
24 set(FREERTOS_KERNEL_PATH "Z:/FreeRTOS-Kernel")
25 include(${FREERTOS_KERNEL_PATH}/portable/ThirdParty/GCC/RP2040/FreeRTOS_Kernel_import.cmake)
26
27 project(PiscaLed C CXX ASM)
28 pico_sdk_init()
29
30 include_directories(${CMAKE_SOURCE_DIR}/include) Local do arquivo FreeRTOSConfig.h
31
32 add_executable(${PROJECT_NAME}
33     blink.c
34 )
35
36 target_include_directories(${PROJECT_NAME} PRIVATE ${CMAKE_SOURCE_DIR})
37
38 target_link_libraries(${PROJECT_NAME}
39     pico_stlport
40     FreeRTOS-Kernel Kernel e
41     FreeRTOS-Kernel-Heap4 Gerenciador de memória
42 )
43
44 pico_enable_stdio_usb(${PROJECT_NAME} 1)
45 pico_enable_stdio_uart(${PROJECT_NAME} 0)
46
47 pico_add_extra_outputs(${PROJECT_NAME})
```

# FreeRTOS no RP2040

## Configuração do FreeRTOSConfig.h

```
C FreeRTOSConfig.h x
include > C FreeRTOSConfig.h > ...
28 #ifndef FREERTOS_CONFIG_H
31 /*-----
42
43 /* Scheduler Related */
44 #define configUSE_PREEMPTION 1
45 #define configUSE_TICKLESS_IDLE 0
46 #define configUSE_IDLE_HOOK 0
47 #define configUSE_TICK_HOOK 0
48 #define configTICK_RATE_HZ ( ( TickType_t ) 1000 )
49 #define configMAX_PRIORITIES 32
50 #define configMINIMAL_STACK_SIZE ( configSTACK_DEPTH_TYPE ) 256
51 #define configUSE_16_BIT TICKS 0
52
53 #define configIDLE_SHOULD_YIELD 1
54
55 /* Synchronization Related */
56 #define configUSE_MUTEXES 1
57 #define configUSE_RECURSIVE_MUTEXES 1
58 #define configUSE_APPLICATION_TASK_TAG 0
59 #define configUSE_COUNTING_SEMAPHORES 1
60 #define configQUEUE_REGISTRY_SIZE 8
61 #define configUSE_QUEUE_SETS 1
62 #define configUSE_TIME_SLICING 1
63 #define configUSE_NEWLIB_REENTRANT 0
64 #define configENABLE_BACKWARD_COMPATIBILITY 0
65 #define configNUM_THREAD_LOCAL_STORAGE_POINTERS 5
```

# FreeRTOS no RP2040

## Exemplo 1:Blink Led com FreeRTOS

```
C blink.c x
C blink.c > ...

1 #include "pico/stdlib.h"
2 #include "FreeRTOS.h"
3 #include "task.h"
4 #include <stdio.h>
5
6 #define led_pin_red 12

7
8 void vBlinkTask() Tarefa
9 {
10     gpio_init(led_pin_red);
11     gpio_set_dir(led_pin_red, GPIO_OUT);
12     while (true)
13     {
14         gpio_put(led_pin_red, true);
15         vTaskDelay(pdMS_TO_TICKS(200));
16         gpio_put(led_pin_red, false);
17         vTaskDelay(pdMS_TO_TICKS(200));
18         printf("Blink\n");
19     }
20 }
21
22 // Trecho para modo BOOTSEL com botão B
```

```
C blink.c x
C blink.c > ...

21
22 // Trecho para modo BOOTSEL com botão B
23 #include "pico/bootrom.h"
24 #define botaoB 6
25 void gpio_irq_handler(uint gpio, uint32_t events)
26 {
27     reset_usb_boot(0, 0);
28 }

29
30 int main()
31 {
32     // Para ser utilizado o modo BOOTSEL com botão B
33     gpio_init(botaoB);
34     gpio_set_dir(botaoB, GPIO_IN);
35     gpio_pull_up(botaoB);
36     gpio_set_irq_enabled_with_callback(botaoB, GPIO_IRQ_EDGE_FALL,
37                                         true, &gpio_irq_handler);
38
39     // Fim do trecho para modo BOOTSEL com botão B

40     stdio_init_all();

41
42     xTaskCreate(vBlinkTask, "Blink Task",
43                 configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL);
44     vTaskStartScheduler();
45     panic_unsupported();
46 }
47
```

Push button reset

Trecho Principal

```
44     xTaskCreate(vBlinkTask, "Blink Task",
45                 configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL);
```

- **vBlinkTask**: É o ponteiro para a função que será executada como tarefa.
- "Blink Task": Nome da tarefa (apenas para depuração).
- **configMINIMAL\_STACK\_SIZE**: Quantidade de **stack** alocada para essa tarefa (tamanho mínimo definido na configuração do FreeRTOS).
- **NULL**: Ponteiro para os **parâmetros passados à tarefa** (não há nenhum neste caso).
- **tskIDLE\_PRIORITY**: Prioridade da tarefa. Essa é a menor possível (igual à da tarefa *idle* do FreeRTOS).
- **NULL**: Ponteiro para receber o **handle da tarefa criada** (não está sendo usado aqui).

44

### vTaskStartScheduler();

Essa função **inicia o agendador de tarefas do FreeRTOS**, ou seja, começa a executar as tarefas criadas com xTaskCreate.

A partir desse ponto, o controle é passado ao **RTOS**, que gerenciará o tempo de CPU entre as tarefas conforme as prioridades e o escalonador.

```
panic_unsupported();
```

**Usada somente para prevenção de falhas.**

Essa função só será chamada se o vTaskStartScheduler() **retornar algo**, o que **não deveria acontecer** em um sistema corretamente configurado. Se retornar, é sinal de falha crítica (por exemplo, se não houver heap suficiente para criar as tarefas básicas do sistema).

```

C blinkConta.c x
C blinkConta.c > vBlinkLed2Task()
1 #include "pico/stdlib.h"
2 #include "hardware/gpio.h"
3 #include "hardware/i2c.h"
4 #include "lib/ssd1306.h"
5 #include "lib/font.h"
6 #include "FreeRTOS.h"
7 #include "FreeRTOSConfig.h"
8 #include "task.h"
9 #include <stdio.h>
10
11 #define I2C_PORT i2c1
12 #define I2C_SDA 14
13 #define I2C_SCL 15
14 #define endereco 0x3C
15
16 #define led1 11
17 #define led2 12
18
19 void vBlinkLed1Task() Tarefa 1
20 {
21     gpio_init(led1);
22     gpio_set_dir(led1, GPIO_OUT);
23     while (true)
24     {
25         gpio_put(led1, true);
26         vTaskDelay(pdMS_TO_TICKS(250));
27         gpio_put(led1, false);
28         vTaskDelay(pdMS_TO_TICKS(1223));
29     }
30 }
31

```

```

31
32 void vBlinkLed2Task() Tarefa 2
33 {
34     gpio_init(led2);
35     gpio_set_dir(led2, GPIO_OUT);
36     while (true)
37     {
38         gpio_put(led2, true);
39         vTaskDelay(pdMS_TO_TICKS(500));
40         gpio_put(led2, false);
41         vTaskDelay(pdMS_TO_TICKS(2224));
42     }
43 }
44
45 void vDisplay3Task() Tarefa 3
46 {
47     // I2C Initialisation. Using it at 400Khz.
48     i2c_init(I2C_PORT, 400 * 1000);
49
50     gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
51     gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);

```

<https://github.com/wiltonlacerda/EmbarcaTechResU1Ex02>

```

45 void vDisplay3Task()
46 {
47     // I2C Initialisation. Using it at 400Khz.
48     i2c_init(I2C_PORT, 400 * 1000);
49
50     gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
51     gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
52     gpio_pull_up(I2C_SDA);
53     gpio_pull_up(I2C_SCL);
54     ssd1306_t ssd;
55     ssd1306_init(&ssd, WIDTH, HEIGHT, false, endereco, I2C_PORT);
56     ssd1306_config(&ssd);
57     ssd1306_send_data(&ssd);
58     // Limpa o display. O display inicia com todos os pixels apagados
59     ssd1306_fill(&ssd, false);
60     ssd1306_send_data(&ssd);
61
62     char str_y[5]; // Buffer para armazenar a string
63     int contador = 0;
64     bool cor = true;
65     while (true)
66     {
67         sprintf(str_y, "%d", contador); // Converte em string
68         contador++; // Incrementa o contador
69         ssd1306_fill(&ssd, !cor); // Limpa o display
70         ssd1306_rect(&ssd, 3, 3, 122, 60, cor, !cor); // Desenho de um retângulo
71         ssd1306_line(&ssd, 3, 25, 123, 25, cor); // Desenho de uma linha vertical
72         ssd1306_line(&ssd, 3, 37, 123, 37, cor); // Desenho de uma linha horizontal
73         ssd1306_draw_string(&ssd, "CEPEDI TIC37", 8, 6); // Desenha a string CEPEDI TIC37
74         ssd1306_draw_string(&ssd, "EMBARCATECH", 20, 16); // Desenha a string EMBARCATECH
75         ssd1306_draw_string(&ssd, " FreeRTOS", 10, 28); // Desenha a string FreeRTOS
76         ssd1306_draw_string(&ssd, "Contador LEDs", 10, 41); // Desenha a string Contador LEDs
77         ssd1306_draw_string(&ssd, str_y, 40, 52); // Atualiza o valor do contador na tela
78         ssd1306_send_data(&ssd); // Atualiza o display
79         sleep_ms(735);
80     }
}

```

### Tarefa 3

```

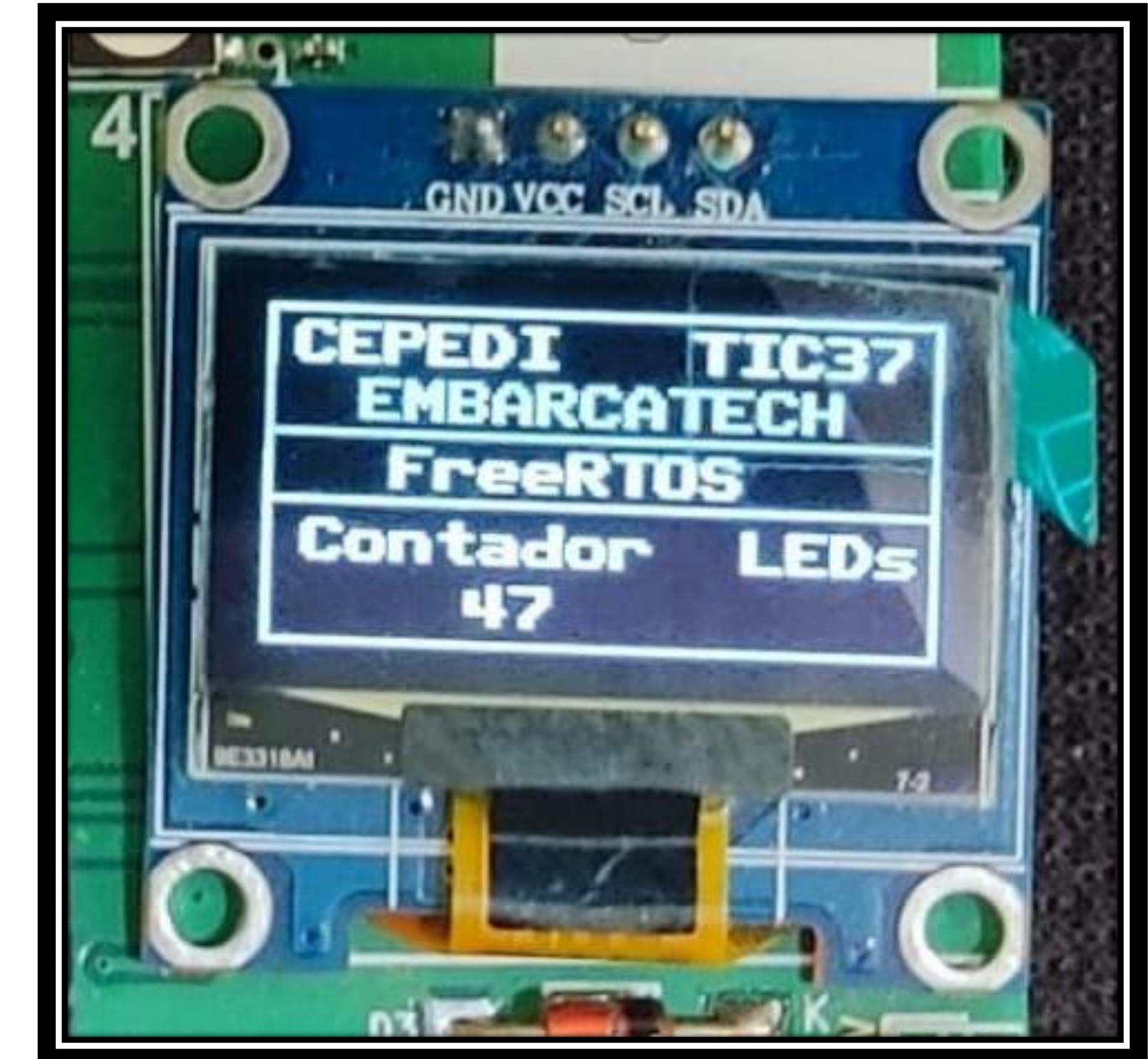
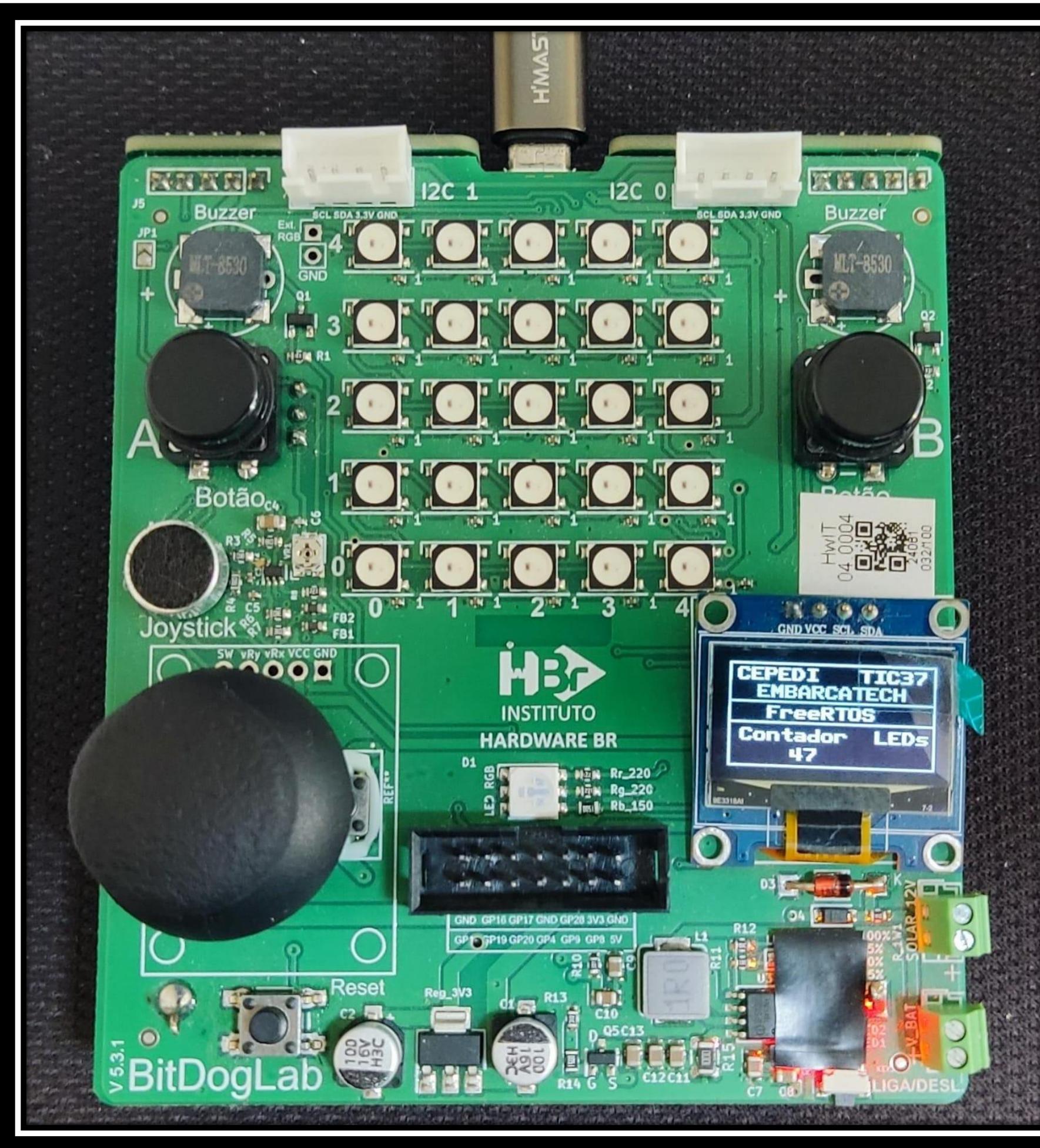
81 }
82
83 // Trecho para modo BOOTSEL com botão B
84 #include "pico/bootrom.h"
85 #define botaoB 6
86 void gpio_irq_handler(uint gpio, uint32_t events)
87 {
88     reset_usb_boot(0, 0);
89 }
90
91 int main()
92 {
93     // Para ser utilizado o modo BOOTSEL com botão B
94     gpio_init(botaoB);
95     gpio_set_dir(botaoB, GPIO_IN);
96     gpio_pull_up(botaoB);
97     gpio_set_irq_enabled_with_callback(botaoB, GPIO_IRQ_EDGE_FALL, true, &gpio_
98     // Fim do trecho para modo BOOTSEL com botão B
99
100 stdio_init_all();
101
102 xTaskCreate(vBlinkLed1Task, "Blink Task Led1", configMINIMAL_STACK_SIZE,
103             NULL, tskIDLE_PRIORITY, NULL);
104 xTaskCreate(vBlinkLed2Task, "Blink Task Led2", configMINIMAL_STACK_SIZE,
105             NULL, tskIDLE_PRIORITY, NULL);
106 xTaskCreate(vDisplay3Task, "Cont Task Disp3", configMINIMAL_STACK_SIZE,
107             NULL, tskIDLE_PRIORITY, NULL);
108 vTaskStartScheduler(); ←
109 panic_unsupported();
110 }

```

### Botão reset

# FreeRTOS no RP2040

Exemplo 2: Contagem Display



## Tarefa: Semáforo Inteligente com Modo Noturno e Acessibilidade

Você deverá implementar, **individualmente**, um sistema de semáforo inteligente com dois modos de operação distintos utilizando apenas tarefas do FreeRTOS (sem uso de filas, semáforos ou mutexes). O sistema será representado por: matriz de LEDs, LED RGB, display e buzzer, disponíveis na plaquinha BitDog Lab com o RP2040.

Modos de operação:

**Modo Normal:** Semáforo com ciclo de cores (verde => amarelo => vermelho) com sinalização sonora através do buzzer.

**Modo Noturno:** Apenas a luz amarela piscando lentamente, também com sinalização sonora através do buzzer. O modo será alternado quando o botão A for pressionado. (criar uma flag global que será modificada por uma tarefa).

**Emitir sinais sonoros com o buzzer para feedback a pessoas cegas.**

Isso permite que elas identifiquem o estado do semáforo por sons distintos.

Modo Normal (Verde: 1 beep curto por um segundo “pode atravessar”; Amarelo: beep rápido intermitente “atenção”; Vermelho: tom contínuo curto (500ms ligado e 1.5s desligado) “pare”;

Modo Noturno: beep lento a cada 2s.



# Obrigado!

Executores:



Coordenação:



Iniciativa:

