C2 - Aplicação Multitarefa

Residente: José Adriano Filho CPF: 266.212.823 - 20
Matrícula: 202420110943463
Monitor: Wellingson Rafael Teixeira

Sistema de Monitoramento Simples com 3 Tarefas

Criar uma aplicação embarcada no **FreeRTOS** com 3 tarefas que simulam o monitoramento de um sistema com sensores (como um botão e um LED). As tarefas irão cooperar para realizar diferentes funções, como ler o estado do botão e controlar o LED.

Resolução:

```
/*
// Nome do arquivo: U1C2Tarefa2.c
// Autor: José Adriano Filho
// Data: 20/10/2023
// Descrição: Criar uma aplicação embarcada no FreeRTOS com 3 tarefas que simulam
// o monitoramento de um sistema com sensores (como um botão e um
// LED). As tarefas irão cooperar para realizar diferentes funções, como ler
// o estado do botão e controlar o LED.
*/

#include <stdio.h>
#include "pico/stdlib.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"

// Defines  e Constantes
#define DELAY_TASK 100         // Tempo de debounce em milissegundos
#define LED_PIN 11             // Pino do LED
#define BUTTON_PIN 5           // Pino do botão
#define QUEUE_TAMANHO 2        // Tamanho da fila para comunicação entre tarefas
#define LED_ON 1               // Estado do LED ligado
#define LED_OFF 0              // Estado do LED desligado

// Definições de Variáveis
int8_t ledState = 0;           // Variável para armazenar o estado do LED
int8_t buttonState = 0;        // Variável para armazenar o estado do botão
int8_t ultimoButtonState = 0;  // Variável para armazenar o último estado do botão

// Definições de Semáforos e Filas
SemaphoreHandle_t xNotificacaoButton; // Semáforo para sincronização entre tarefas
```

```
QueueHandle_t xQueueLed;       // Fila para comunicação entre tarefas

//Protótipo das funções
void vTaskButtonRead(void *pvParameters);
void vTaskButtonProcess(void *pvParameters);
void vTaskControlLed(void *pvParameters);
void InitPins(void);

int main()
{
   stdio_init_all();          // Inicializa a comunicação serial
   InitPins();               // Inicializa os pinos do LED e do botão

   xNotificacaoButton = xSemaphoreCreateBinary(); // Cria o semáforo para sincronização entre
tarefas
   if (xNotificacaoButton == NULL) { // Verifica se o semáforo foi criado com sucesso
      printf("Erro ao criar o semáforo\n");
      return 1; // Retorna erro se não conseguiu criar o semáforo
   }

      xQueueLed = xQueueCreate(QUEUE_TAMANHO, sizeof(int8_t)); // Cria a fila para
comunicação entre tarefas
   if (xQueueLed == NULL) {    // Verifica se a fila foi criada com sucesso
      printf("Erro ao criar a fila\n");
      return 1;              // Retorna erro se não conseguiu criar a fila
   }

   xTaskCreate(vTaskButtonRead, "Leitura Botao", 1000, NULL, 1, NULL);
   xTaskCreate(vTaskButtonProcess, "Processamento Botao", 1000, NULL, 1, NULL);
   xTaskCreate(vTaskControlLed, "Controle Led", 1000, NULL, 1, NULL);
   vTaskStartScheduler();      // Inicia o agendador do FreeRTOS

   return 0; // O código não deve chegar aqui, pois o agendador do FreeRTOS assume o controle
}

/*********************************************************************** */

// Inicializa os pinos do LED e do botão
void InitPins(void) {
   gpio_init(LED_PIN);        // Inicializa o pino do LED
   gpio_set_dir(LED_PIN, GPIO_OUT); // Define o pino do LED como saída

   gpio_init(BUTTON_PIN);     // Inicializa o pino do botão
   gpio_set_dir(BUTTON_PIN, GPIO_IN); // Define o pino do botão como entrada
```

```
    gpio_pull_up(BUTTON_PIN);   // Ativa o resistor pull-up interno para evitar flutuações no estado do botão
}

/*************************************************************************/
// Área das Tasks

// Função de leitura do botão
void vTaskButtonRead(void *pvParameters) {

   while (true) {
      // Lógica para ler o estado do botão
      int8_t buttonStateAtual = gpio_get(BUTTON_PIN); // Lê o estado atual do botão
      // Se o botão for pressionado, notifique a tarefa de processamento
      if (buttonStateAtual != ultimoButtonState) {    // Se o estado do botão mudou
         ultimoButtonState = buttonStateAtual;      // Atualiza o último estado do botão
         if (buttonStateAtual == 0) {              // Botão pressionado (nível baixo)
            buttonState = 1;                    // Atualiza o estado do botão
         } else {
            buttonState = 0;                    // Atualiza o estado do botão
         }
         xSemaphoreGive(xNotificacaoButton);        // Notifica a tarefa de processamento
      } else {
         buttonState = 0;                       // Atualiza o estado do botão
      }
   }
      vTaskDelay(pdMS_TO_TICKS(DELAY_TASK));         // Delay para debounce
}

// Função de processamento do botão
void vTaskButtonProcess(void *pvParameters) {
   while (true) {
      // Aguarda a notificação da tarefa de leitura do botão
      xSemaphoreTake(xNotificacaoButton, portMAX_DELAY); // Aguarda a notificação
      // Lógica para processar o estado do botão
      printf("Estado do botão: %d\n", buttonState);   // Imprime o estado do botão no console
        long resultado = xQueueSend(xQueueLed, &buttonState, portMAX_DELAY); // Envia o estado do botão para a fila
      if(resultado == pdTRUE) {                    // Se o envio foi bem-sucedido
         printf("Estado do botão enviado para a fila: %d\n", buttonState); // Imprime no console que o estado do botão foi enviado para a fila
      } else {
         printf("Erro ao enviar o estado do botão para a fila\n"); // Imprime no console que houve um erro ao enviar o estado do botão para a fila
      }
```

```
      // Delay para evitar processamento excessivo
        vTaskDelay(pdMS_TO_TICKS(DELAY_TASK));            // Delay para evitar processamento excessivo
   }
}

// Função de controle do LED
void vTaskControlLed(void *pvParameters) {
   while (true) {
      printf("Esperando o estado do botão...\n");     // Imprime no console que está esperando o estado do botão
      // Lógica para controlar o LED com base no estado do botão
        long resultado = xQueueReceive(xQueueLed, &ledState, portMAX_DELAY); // Recebe o estado do botão da fila

      if (resultado == pdTRUE) {                  // Se o estado do botão foi recebido com sucesso
         printf("Estado do botão recebido: %d\n", ledState); // Imprime no console que o estado do botão foi recebido
         if (ledState == 1) {                    // Se o botão estiver pressionado
            gpio_put(LED_PIN, LED_ON);           // Liga o LED
            printf("LED ligado\n");              // Imprime no console que o LED está ligado
         } else {
            gpio_put(LED_PIN, LED_OFF);          // Desliga o LED
            printf("LED desligado\n");           // Imprime no console que o LED está desligado
         }
      } else {
         printf("Erro ao receber o estado do botão da fila\n"); // Imprime no console que houve um erro ao receber o estado do botão da fila
      }
      printf("Estado do LED: %d\n", ledState);       // Imprime o estado do LED no console
      // Delay para evitar controle excessivo do LED
      vTaskDelay(pdMS_TO_TICKS(DELAY_TASK));
   }
```