**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));

    }