RORRAS NEVES DA SILVA / 20251RSE.MTC0099

Desenvolva um sistema multitarefa no Raspberry Pi Pico W, utilizando o FreeRTOS e o SDK do Pico. O sistema deve realizar a leitura de um joystick analógico (eixos X e Y), detectar o pressionamento de um botão e gerar feedback sonoro por meio de um buzzer. O projeto deve utilizar filas, mutexes e semáforo contador para comunicação e sincronização entre tarefas. A aplicação deverá ser composta por quatro tarefas, conforme descrito a seguir:

- Tarefa 1: Leitura dos Eixos (VRx/VRy)

Lê os valores dos pinos analógicos a cada 100ms. Envia os dados para a fila.

- Tarefa 2: Leitura do Botão (SW)

Verifica o estado do botão a cada 50ms. Em caso de clique, envia evento para a fila.

- Tarefa 3: Processamento dos Dados

Recebe dados da fila e exibe no terminal (protegido por mutex).
Aciona o buzzer apenas se houver movimento significativo ou botão pressionado.

- Tarefa 4: Controle do Buzzer (Implícita na Tarefa 3)

Buzzer ligado por 100ms somente se houver evento válido.
Controlado por semáforo contador com máximo de 2 acessos simultâneos.

```c
#include <stdio.h>
#include <stdio.h>
#include "pico/stdlib.h"
#include "FreeRTOS.h"
#include "hardware/adc.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "pwm.h"

// Pinos
#define VRX_PIN     27  // ADC1
#define VRY_PIN     26  // ADC0
#define BUTTON_PIN  22  // Pull-up


// Tipos de eventos
typedef enum {
    EVENTO_EIXOS,
    EVENTO_BOTAO
} TipoEvento;

typedef struct {
    TipoEvento tipo;
    uint16_t vrx;
    uint16_t vry;
} Evento;

// handles
QueueHandle_t fila_eventos;
SemaphoreHandle_t mutex_printf;
SemaphoreHandle_t semaphore_buzzer;

// === Tarefa 1: Leitura dos Eixos ===
void tarefa_leitura_eixos(void *params) {
    while (true) {
        Evento ev;
        ev.tipo = EVENTO_EIXOS;

        adc_select_input(1); // VRx -> canal 1
        ev.vrx = adc_read();
        adc_select_input(0); // Volta para canal 0 (VRy)
```

```
        ev.vry = adc_read();

        xQueueSend(fila_eventos, &ev, 0);
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

// === Tarefa 2: Leitura do Botão ===
void tarefa_leitura_botao(void *params) {
    bool botao_anterior = true;
    while (true) {
        bool estado = gpio_get(BUTTON_PIN);

        if (!estado && botao_anterior) {
            Evento ev = { .tipo = EVENTO_BOTAO };
            xQueueSend(fila_eventos, &ev, 0);
        }
        botao_anterior = estado;
        vTaskDelay(pdMS_TO_TICKS(50));
    }
}

// === Tarefa 3: Processamento ===
void tarefa_processamento(void *params) {
    Evento ev;
    while (true) {
        if (xQueueReceive(fila_eventos, &ev, portMAX_DELAY)) {
            xSemaphoreTake(mutex_printf, portMAX_DELAY);
            if (ev.tipo == EVENTO_EIXOS) {
                printf("Eixos - VRx: %d, VRy: %d\n", ev.vrx, ev.vry);
                if (ev.vrx > 3000 || ev.vry > 3000) {
                    if (xSemaphoreTake(semaphore_buzzer, 0) == pdTRUE) {
                        buzzer_pwm_init();
                        vTaskDelay(pdMS_TO_TICKS(100));
                        buzzer_pwm_stop();
                        xSemaphoreGive(semaphore_buzzer);
                    }
                }
            } else if (ev.tipo == EVENTO_BOTAO) {
                printf("Botão pressionado!\n");
                if (xSemaphoreTake(semaphore_buzzer, 0) == pdTRUE) {
                    buzzer_pwm_init();
                    vTaskDelay(pdMS_TO_TICKS(100));
                    buzzer_pwm_stop();
                    xSemaphoreGive(semaphore_buzzer);
                }
            }
            xSemaphoreGive(mutex_printf);
        }
    }
}

// === Configuração dos periféricos ===
void setup_hardware() {
    stdio_init_all();

    adc_init();
    adc_gpio_init(VRY_PIN);  // ADC0
    adc_gpio_init(VRX_PIN);  // ADC1
    adc_select_input(0);     // Inicialmente VRy

    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_up(BUTTON_PIN);


}
```

```c
// === Função principal ===
int main() {
    setup_hardware();

    fila_eventos = xQueueCreate(10, sizeof(Evento));
    mutex_printf = xSemaphoreCreateMutex();
    semaphore_buzzer = xSemaphoreCreateCounting(2, 2);

    xTaskCreate(tarefa_leitura_eixos, "LeituraEixos", 1024, NULL, 1, NULL);
    xTaskCreate(tarefa_leitura_botao, "LeituraBotao", 1024, NULL, 1, NULL);
    xTaskCreate(tarefa_processamento, "Processamento", 1024, NULL, 2, NULL);

    vTaskStartScheduler();

    while (1); // Nunca deve chegar aqui
}
```