1

#include <Arduino.h>

// Declaración de la función antes de su uso

void anotherTask(void \* parameter);

void setup() {

Serial.begin(115200); // Ajuste del baud rate correcto

/\* Creamos una nueva tarea \*/

xTaskCreate(

anotherTask, /\* Función de la tarea \*/

"Another Task", /\* Nombre de la tarea \*/

10000, /\* Tamaño de la pila \*/

NULL, /\* Parámetro de la tarea \*/

1, /\* Prioridad de la tarea \*/

NULL /\* Manejador de la tarea \*/

);

}

void loop() {

Serial.println("This is ESP32 Task");

delay(1000);

}

/\* Definición de la función de la tarea \*/

void anotherTask(void \* parameter) {

for(;;) { // Bucle infinito

Serial.println("This is another Task");

vTaskDelay(1000 / portTICK\_PERIOD\_MS); // Uso correcto de FreeRTOS delay

}

vTaskDelete(NULL); // Nunca se ejecutará debido al bucle infinito

}

[env:esp32-s3-devkitm-1]

platform = espressif32

board = esp32-s3-devkitm-1

framework = arduino

monitor\_speed = 115200

2

#include <Arduino.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/semphr.h"

// Definir los pines de los LEDs

#define LED1\_PIN 2 // Primer LED en GPIO 2

#define LED2\_PIN 4 // Segundo LED en GPIO 4

// Crear el semáforo

SemaphoreHandle\_t xSemaphore;

void turnOnLED1(void \* parameter) {

for(;;) {

if (xSemaphoreTake(xSemaphore, portMAX\_DELAY) == pdTRUE) {

digitalWrite(LED1\_PIN, HIGH); // Enciende LED 1

digitalWrite(LED2\_PIN, LOW); // Apaga LED 2

Serial.println("LED 1 ENCENDIDO");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

xSemaphoreGive(xSemaphore); // Libera el semáforo

}

vTaskDelay(1 / portTICK\_PERIOD\_MS);

}

}

void turnOnLED2(void \* parameter) {

for(;;) {

if (xSemaphoreTake(xSemaphore, portMAX\_DELAY) == pdTRUE) {

digitalWrite(LED1\_PIN, LOW); // Apaga LED 1

digitalWrite(LED2\_PIN, HIGH); // Enciende LED 2

Serial.println("LED 2 ENCENDIDO");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

xSemaphoreGive(xSemaphore); // Libera el semáforo

}

vTaskDelay(1/ portTICK\_PERIOD\_MS);

}

}

void setup() {

Serial.begin(115200);

pinMode(LED1\_PIN, OUTPUT);

pinMode(LED2\_PIN, OUTPUT);

// Crear el semáforo binario

xSemaphore = xSemaphoreCreateBinary();

if (xSemaphore != NULL) {

xSemaphoreGive(xSemaphore); // Inicia el semáforo disponible

// Crear las tareas

xTaskCreate(turnOnLED1, "Turn On LED 1", 1000, NULL, 1, NULL);

xTaskCreate(turnOnLED2, "Turn On LED 2", 1000, NULL, 1, NULL);

}

}

void loop() {

// No se usa en FreeRTOS

}

EJERCICIO EXTRA RELOJES

#include <Arduino.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "freertos/semphr.h"

// Definición de pines

#define LED\_SEGUNDOS 2 // LED que parpadea cada segundo

#define LED\_MODO 4 // LED que indica modo de ajuste

#define BTN\_MODO 16 // Botón para cambiar modo

#define BTN\_INCREMENTO 17 // Botón para incrementar valor

// Variables del reloj

volatile int horas = 0;

volatile int minutos = 0;

volatile int segundos = 0;

volatile int modo = 0; // 0: normal, 1: ajuste horas, 2: ajuste minutos

// Recursos FreeRTOS

QueueHandle\_t botonQueue;

SemaphoreHandle\_t relojMutex;

// Estructura de eventos de botones

typedef struct {

uint8\_t boton;

uint32\_t tiempo;

} EventoBoton;

// Prototipos de funciones (declaraciones antes de setup)

void TareaReloj(void \*pvParameters);

void TareaLecturaBotones(void \*pvParameters);

void TareaActualizacionDisplay(void \*pvParameters);

void TareaControlLEDs(void \*pvParameters);

void IRAM\_ATTR ISR\_Boton(void \*arg);

void setup() {

Serial.begin(115200);

pinMode(LED\_SEGUNDOS, OUTPUT);

pinMode(LED\_MODO, OUTPUT);

pinMode(BTN\_MODO, INPUT\_PULLUP);

pinMode(BTN\_INCREMENTO, INPUT\_PULLUP);

botonQueue = xQueueCreate(10, sizeof(EventoBoton));

relojMutex = xSemaphoreCreateMutex();

attachInterruptArg(BTN\_MODO, ISR\_Boton, (void\*)BTN\_MODO, FALLING);

attachInterruptArg(BTN\_INCREMENTO, ISR\_Boton, (void\*)BTN\_INCREMENTO, FALLING);

xTaskCreate(TareaReloj, "RelojTask", 2048, NULL, 1, NULL);

xTaskCreate(TareaLecturaBotones, "BotonesTask", 2048, NULL, 2, NULL);

xTaskCreate(TareaActualizacionDisplay, "DisplayTask", 2048, NULL, 1, NULL);

xTaskCreate(TareaControlLEDs, "LEDsTask", 1024, NULL, 1, NULL);

}

void loop() { vTaskDelay(portMAX\_DELAY); }

// 🛠️ Tarea que actualiza el reloj cada segundo

void TareaReloj(void \*pvParameters) {

TickType\_t xLastWakeTime = xTaskGetTickCount();

for (;;) {

vTaskDelayUntil(&xLastWakeTime, pdMS\_TO\_TICKS(1000));

if (xSemaphoreTake(relojMutex, portMAX\_DELAY) == pdTRUE) {

if (modo == 0) { // Solo actualiza si está en modo normal

segundos++;

if (segundos >= 60) { segundos = 0; minutos++; }

if (minutos >= 60) { minutos = 0; horas++; }

if (horas >= 24) { horas = 0; }

}

xSemaphoreGive(relojMutex);

}

}

}

// 🔘 ISR para manejar interrupciones de botones

void IRAM\_ATTR ISR\_Boton(void \*arg) {

uint8\_t numeroPulsador = (uint32\_t)arg;

EventoBoton evento;

evento.boton = numeroPulsador;

evento.tiempo = xTaskGetTickCountFromISR();

xQueueSendFromISR(botonQueue, &evento, NULL);

}

// 🎮 Tarea que maneja los pulsadores

void TareaLecturaBotones(void \*pvParameters) {

EventoBoton evento;

uint32\_t ultimoTiempoBoton = 0;

for (;;) {

if (xQueueReceive(botonQueue, &evento, portMAX\_DELAY) == pdPASS) {

if ((evento.tiempo - ultimoTiempoBoton) >= pdMS\_TO\_TICKS(300)) { // Anti-rebote

if (xSemaphoreTake(relojMutex, portMAX\_DELAY) == pdTRUE) {

if (evento.boton == BTN\_MODO) {

modo = (modo + 1) % 3; // Cambia entre 0, 1 y 2

Serial.printf("Modo cambiado: %d\n", modo);

} else if (evento.boton == BTN\_INCREMENTO) {

if (modo == 1) { horas = (horas + 1) % 24; }

else if (modo == 2) { minutos = (minutos + 1) % 60; segundos = 0; }

Serial.printf("Hora ajustada: %02d:%02d\n", horas, minutos);

}

xSemaphoreGive(relojMutex);

}

ultimoTiempoBoton = evento.tiempo;

}

}

}

}

// 📟 Tarea que muestra la hora en el puerto serie

void TareaActualizacionDisplay(void \*pvParameters) {

for (;;) {

if (xSemaphoreTake(relojMutex, portMAX\_DELAY) == pdTRUE) {

Serial.printf("%02d:%02d:%02d [Modo %d]\n", horas, minutos, segundos, modo);

xSemaphoreGive(relojMutex);

}

vTaskDelay(pdMS\_TO\_TICKS(500)); // Refresco cada 500ms

}

}

// 💡 Tarea que controla los LEDs

void TareaControlLEDs(void \*pvParameters) {

for (;;) {

digitalWrite(LED\_SEGUNDOS, segundos % 2); // Parpadeo cada segundo

digitalWrite(LED\_MODO, modo > 0); // Se enciende en modo ajuste

vTaskDelay(pdMS\_TO\_TICKS(500));

}

}