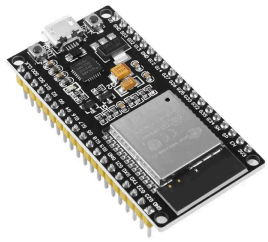
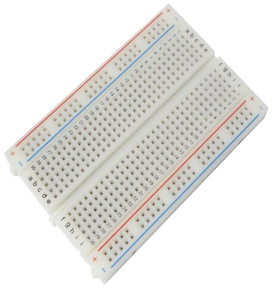
PRACTICA 4 : SISTEMAS OPERATIVOS EN TIEMPO REAL

MATERIAL

Para esta practica necesitaremos el microcontrolador ESP32, una luz led, protoboard y cables (opcional)

OBJETIVO Y FUNCIONALIDAD DE LA PRACTICA

El objetivo de la practica es comprender el funcionamiento de un sistema operativo en tiempo real.

Para lo cual realizaremos una practica donde generaremos varias tareas y veremos como se ejecutan dividiendo el tiempo de uso de la cpu.

Ejercicio Practico 1

Programar el siguiente codigo

```
void setup()
{
Serial.begin(112500);
/* we create a new task here */
xTaskCreate(
anotherTask, /* Task function. */
"another Task", /* name of task. */
10000, /* Stack size of task */
NULL, /* parameter of the task */
1, /* priority of the task */
NULL); /* Task handle to keep track of created task */
}

/* the forever loop() function is invoked by Arduino ESP32 loopTask */
void loop()
{
Serial.println("this is ESP32 Task");
delay(1000);
}

/* this function will be invoked when additionalTask was created */
void anotherTask( void * parameter )
{
/* loop forever */
for(;;)
{
Serial.println("this is another Task");
delay(1000);
}
/* delete a task when finish,
this will never happen because this is infinity loop */
vTaskDelete( NULL );
}
```

1. **Descibir la salida por el puerto serie**
   Cuando envias el código a la ESP32 aparece en el monitor serie el siguiente código:

> this is ESP32 Task

```
this is another Task
this is ESP32 Task
this is another Task
this is ESP32 Task
this is another Task
...
```

Estos mensajes aparecen intercalados porque las dos tareas se ejecutan en paralelo. Las funciones que se encargan de enviar estos mensajes son:

- *loop()*: Imprime "this is ESP32 Task" cada 1 segundo.
- *anotherTask()*: Imprime "this is another Task" cada 1 segundo en un hilo independiente.

2. **Explicar el funcionamiento**

- El ESP32 inicia la tarea principal (loop()), que imprime "this is ESP32 Task" cada 1 segundo.
- La función anotherTask() se ejecuta en otro núcleo o en la misma CPU con FreeRTOS, imprimiendo "this is another Task" cada 1 segundo.
- Ambas tareas se ejecutan en paralelo, compartiendo tiempo de CPU gracias a FreeRTOS.

Ejercicio Practico 2

1. **Realizar un programa que utilice dos tareas una enciende un led y otra lo apaga dichas tareas deben estar sincronizadas sugerencias utilizar un semaforo**

Este programa en ESP32 implementa multitarea con FreeRTOS para alternar el encendido y apagado de un LED de manera sincronizada.
Se utilizan dos tareas concurrentes (ledON y ledOFF), y su ejecución se sincroniza mediante semáforos binarios.

El ESP32 ejecuta dos tareas en paralelo:

- Primero, ledON se ejecuta, enciende el LED y habilita ledOFF.
- Luego, ledOFF toma el control, apaga el LED y habilita ledON.
- Este ciclo se repite indefinidamente, alternando el encendido y apagado del LED cada 1

segundo.

Código comentado:

```
#include <Arduino.h>

// Declaración de funciones para tareas FreeRTOS
void ledON(void *pvParameters);
void ledOFF(void *pvParameters);

int LED = 2;  // Pin del LED

// Crear dos semáforos binarios
SemaphoreHandle_t semaforo_ON;
SemaphoreHandle_t semaforo_OFF;

void setup() {
    Serial.begin(115200);
    pinMode(LED, OUTPUT);

    // Crear los semáforos
    semaforo_ON = xSemaphoreCreateBinary();
    semaforo_OFF = xSemaphoreCreateBinary();

    if (semaforo_ON != NULL && semaforo_OFF != NULL) {
        xSemaphoreGive(semaforo_ON);  // Comenzamos con `ledON` habilitado
    }

    // Crear tareas con la misma prioridad para alternar perfectamente
    xTaskCreate(ledON, "LED ON", 1000, NULL, 1, NULL);
    xTaskCreate(ledOFF, "LED OFF", 1000, NULL, 1, NULL);
}

void loop() {
    vTaskDelete(NULL);  // No se usa loop() en FreeRTOS
}

// Tarea para encender el LED
void ledON(void *pvParameters) {
    for (;;) {
        if (xSemaphoreTake(semaforo_ON, portMAX_DELAY)) {  // Espera a su turno
            Serial.println("Ejecutando tarea: LED ON");
            digitalWrite(LED, HIGH);
```

```
            vTaskDelay(pdMS_TO_TICKS(1000));  // LED encendido por 1 seg
            xSemaphoreGive(semaforo_OFF);  // Habilita la tarea `ledOFF`
        }
    }
}

// Tarea para apagar el LED
void ledOFF(void *pvParameters) {
    for (;;) {
        if (xSemaphoreTake(semaforo_OFF, portMAX_DELAY)) {  // Espera su turno
            Serial.println("Ejecutando tarea: LED OFF");
            digitalWrite(LED, LOW);
            vTaskDelay(pdMS_TO_TICKS(1000));  // LED apagado por 1 seg
            xSemaphoreGive(semaforo_ON);  // Habilita la tarea `ledON`
        }
    }
}
```

Las variables *semaforo_ON* y *semaforo_OFF* garantizan que solo una tarea acceda al LED a la vez.
La sincronización perfecta entre las tareas se debe al uso correcto de los semáforos.

Cada tarea cede el turno a la otra al finalizar, evitando ejecución desordenada.

Salida del monitor serie:

> Ejecutando tarea: LED ON
> Ejecutando tarea: LED OFF
> Ejecutando tarea: LED ON
> Ejecutando tarea: LED OFF
> ...

Muestra en que momento ejecuta cada tarea y el estado del LED correspondiente.

Ejercicios opcionales

Reloj

Hardware necesario

- ESP32 (cualquier modelo compatible con Arduino)
- 2 LEDs (con sus resistencias correspondientes)
- 2 Pulsadores Cables de conexión Protoboard

Código comentado

```
#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 (tipo reloj)
#define LED_MODO 4           // LED que se enciende cuando estamos en modo ajuste
#define BTN_MODO 16          // Botón para cambiar el modo (normal, ajustar horas, ajustar
#define BTN_INCREMENTO 17    // Botón para incrementar horas o minutos dependiendo del mod

// ---------------------- VARIABLES DEL RELOJ -------------------------
volatile int horas = 0;
volatile int minutos = 0;
volatile int segundos = 0;
volatile int modo = 0; // 0 = normal, 1 = ajustar horas, 2 = ajustar minutos

// ---------------------- RTOS: COLA Y SEMÁFORO -------------------------
QueueHandle_t botonQueue;         // Cola para eventos de botones (usada por interrupcion
SemaphoreHandle_t relojMutex;     // Mutex para proteger acceso concurrente al reloj

// ---------------------- ESTRUCTURA PARA EVENTOS DE BOTONES -------------------------
typedef struct {
  uint8_t boton;       // Qué botón fue presionado
  uint32_t tiempo;     // Tiempo de la pulsación (para control de rebote)
} EventoBoton;

// ---------------------- PROTOTIPOS DE TAREAS -------------------------
void TareaReloj(void *pvParameters);
void TareaLecturaBotones(void *pvParameters);
void TareaActualizacionDisplay(void *pvParameters);
void TareaControlLEDs(void *pvParameters);

// ---------------------- ISR: GESTIÓN DE INTERRUPCIÓN DE BOTÓN -------------------------
void IRAM_ATTR ISR_Boton(void *arg) {
  uint8_t numeroPulsador = (uint32_t)arg;

  EventoBoton evento;
```

```
  evento.boton = numeroPulsador;
  evento.tiempo = xTaskGetTickCountFromISR(); // Tiempo del evento (ticks de RTOS)

  // Enviamos el evento a la cola desde interrupción
  xQueueSendFromISR(botonQueue, &evento, NULL);
}

void setup() {
  Serial.begin(115200);
  Serial.println("Inicializando Reloj Digital con RTOS");

  // Configuración de pines
  pinMode(LED_SEGUNDOS, OUTPUT);
  pinMode(LED_MODO, OUTPUT);
  pinMode(BTN_MODO, INPUT_PULLUP);
  pinMode(BTN_INCREMENTO, INPUT_PULLUP);

  // Crear recursos RTOS
  botonQueue = xQueueCreate(10, sizeof(EventoBoton));         // Cola para máximo 10 event
  relojMutex = xSemaphoreCreateMutex();                       // Mutex para sincronizar ac

  // Configurar interrupciones de botones
  attachInterruptArg(BTN_MODO, ISR_Boton, (void*)BTN_MODO, FALLING);
  attachInterruptArg(BTN_INCREMENTO, ISR_Boton, (void*)BTN_INCREMENTO, FALLING);

  // Crear tareas FreeRTOS
  xTaskCreate(TareaReloj, "RelojTask", 2048, NULL, 1, NULL);
  xTaskCreate(TareaLecturaBotones, "BotonesTask", 2048, NULL, 2, NULL);  // Prioridad mayo
  xTaskCreate(TareaActualizacionDisplay, "DisplayTask", 2048, NULL, 1, NULL);
  xTaskCreate(TareaControlLEDs, "LEDsTask", 1024, NULL, 1, NULL);
}

void loop() {
  // El loop no se usa porque todo lo maneja el sistema de tareas
  vTaskDelay(portMAX_DELAY);
}

// -------------------- TAREA: ACTUALIZA EL TIEMPO CADA SEGUNDO ----------------------
void TareaReloj(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
```

```
  const TickType_t xPeriod = pdMS_TO_TICKS(1000); // Cada 1000 ms (1 segundo)

  for (;;) {
    vTaskDelayUntil(&xLastWakeTime, xPeriod); // Espera exacta cada segundo

    if (xSemaphoreTake(relojMutex, portMAX_DELAY) == pdTRUE) {
      if (modo == 0) { // Solo avanza el reloj en modo normal
        segundos++;
        if (segundos >= 60) {
          segundos = 0;
          minutos++;
          if (minutos >= 60) {
            minutos = 0;
            horas++;
            if (horas >= 24) horas = 0;
          }
        }
      }
      xSemaphoreGive(relojMutex);
    }
  }
}

// ---------------------- TAREA: GESTIÓN DE BOTONES -------------------------
void TareaLecturaBotones(void *pvParameters) {
  EventoBoton evento;
  uint32_t ultimoTiempoBoton = 0;
  const uint32_t debounceTime = pdMS_TO_TICKS(300); // 300ms anti-rebote

  for (;;) {
    if (xQueueReceive(botonQueue, &evento, portMAX_DELAY) == pdPASS) {
      if ((evento.tiempo - ultimoTiempoBoton) >= debounceTime) {
        if (xSemaphoreTake(relojMutex, portMAX_DELAY) == pdTRUE) {
          if (evento.boton == BTN_MODO) {
            modo = (modo + 1) % 3; // Rota entre 0, 1, 2
            Serial.printf("Cambio de modo: %d\n", modo);
          } else if (evento.boton == BTN_INCREMENTO) {
            if (modo == 1) {
              horas = (horas + 1) % 24;
              Serial.printf("Horas ajustadas a: %d\n", horas);
```

```
            } else if (modo == 2) {
              minutos = (minutos + 1) % 60;
              segundos = 0; // Reiniciar segundos al cambiar minutos
              Serial.printf("Minutos ajustados a: %d\n", minutos);
            }
          }
          xSemaphoreGive(relojMutex);
        }
        ultimoTiempoBoton = evento.tiempo;
      }
    }
  }
}

// --------------------- TAREA: ACTUALIZA LA HORA POR PUERTO SERIE ----------------------
void TareaActualizacionDisplay(void *pvParameters) {
  int horasAnterior = -1, minutosAnterior = -1, segundosAnterior = -1, modoAnterior = -1;

  for (;;) {
    if (xSemaphoreTake(relojMutex, portMAX_DELAY) == pdTRUE) {
      bool cambios = (horas != horasAnterior) || (minutos != minutosAnterior) || (segundos

      if (cambios) {
        // Muestra formato de reloj
        Serial.printf("%02d:%02d:%02d", horas, minutos, segundos);

        // Muestra modo actual
        if (modo == 0) Serial.println(" [Modo Normal]");
        else if (modo == 1) Serial.println(" [Ajuste Horas]");
        else if (modo == 2) Serial.println(" [Ajuste Minutos]");

        // Actualiza los valores anteriores
        horasAnterior = horas;
        minutosAnterior = minutos;
        segundosAnterior = segundos;
        modoAnterior = modo;
      }
      xSemaphoreGive(relojMutex);
    }
    vTaskDelay(pdMS_TO_TICKS(100)); // Refresco cada 100ms
```

```
  }
}

// ---------------------- TAREA: CONTROL DE LOS LEDS -------------------------
void TareaControlLEDs(void *pvParameters) {
  bool estadoLedSegundos = false;

  for (;;) {
    if (xSemaphoreTake(relojMutex, portMAX_DELAY) == pdTRUE) {
      estadoLedSegundos = !estadoLedSegundos; // Cambia el estado del LED de segundos
      digitalWrite(LED_SEGUNDOS, estadoLedSegundos); // Parpadea cada 500ms

      digitalWrite(LED_MODO, modo > 0); // Enciende LED de modo si no estamos en modo norm

      xSemaphoreGive(relojMutex);
    }
    vTaskDelay(pdMS_TO_TICKS(500)); // Refresco cada 500ms
  }
}
```

Funcionamiento

Este código implementa un reloj en tiempo real (horas, minutos y segundos) utilizando un ESP32 con FreeRTOS. El tiempo se muestra por el monitor serie, y puedes ajustar la hora usando dos botones.

- El reloj avanza automáticamente cada segundo.
- Puedes ajustar horas o minutos usando dos pulsadores:
  - Uno para cambiar de modo (modo normal / ajustar horas / ajustar minutos)
  - Otro para incrementar el valor correspondiente (hora o minuto).
- Hay dos LEDs:
  - Uno parpadea cada segundo (simula el segundero).
  - Otro se enciende cuando estás en modo ajuste (ajustar horas o ajustar minutos).

Cada vez que cambia la hora, minuto, segundo o modo, aparece una línea como esta por el monitor serial:

```
14:25:03 [Modo Normal]
14:25:03 [Ajuste Horas]
14:25:03 [Ajuste Minutos]
```

También, al ajustar con los botones:

```
Horas ajustadas a: 15
Minutos ajustados a: 30
Cambio de modo: 1
```

Video demostración

https://drive.google.com/file/d/1384KYViyQ7NKEjaRN_xytpBGW6w5jnWe/view?usp=share_link