Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 9.- Sistemas operativos en tiempo real (RTOS) en un sistema en chip

Sistemas en chip
Profesora: Ana Luz Barrales Lopez

Luciano Hernández Jonathan
Rodriguez Morales Iñaki

7 de diciembre del 2024

1 Introducción

En el presente reporte se describe la implementación de un sistema embebido multitarea utilizando la plataforma Arduino en conjunto con el sistema operativo en tiempo real FreeRTOS. Este proyecto tiene como objetivo principal la integración y el control concurrente de múltiples componentes electrónicos, incluyendo un sensor de ambiente BME280, una pantalla LCD y un conjunto de LEDs, mediante el uso de tareas específicas que optimizan la ejecución de procesos en tiempo real.

El código presentado implementa tres tareas principales: el control de LEDs en serie basado en la interacción con un botón, la lectura de datos ambientales (temperatura) desde el sensor BME280 y su visualización en un display LCD, así como la transmisión de datos por el puerto serial. Adicionalmente, se incluye una tarea para el parpadeo de un LED, demostrando el manejo eficiente de tareas periódicas con FreeRTOS.

1.1 Objetivo

- Aprenda sobre los conceptos del sistema operativo en tiempo real (RTOS) mediante la implementación de tareas simultáneas con FreeRTOS en una plataforma basada en Arduino.

1.2 Materiales

Los materiales utilizados en la práctica son los siguientes:

- Arduino Uno
- Arduino IDE
- Cable para conexiones
- Pantalla LCD 16x2
- Módulo BME280
- Potenciometro 10k
- 2 Resistencias 10k
- 4 Leds
- Boton Pulsador

2 Desarrollo de la Práctica

2.1 Conexiones

Para realizar las conexiones necesarias, el sensor BME280 debe conectarse a los pines I2C del Arduino (SDA a A4 y SCL a A5 en un Arduino Uno), mientras que la pantalla LCD 16x2 se conecta en modo de 4 bits utilizando los pines 2, 3, 4, 5, 6 y 7 para los datos y el control (RS y EN). El botón pulsador se conecta al pin digital 9, configurado como entrada, y el LED para control en serie al pin digital 8, con resistencias para limitar la

corriente. Adicionalmente, el LED destinado al parpadeo debe conectarse al pin digital 10 con su respectiva resistencia. Se recomienda alimentar el sistema mediante el puerto USB del Arduino o una fuente de energía adecuada, asegurando una correcta conexión a tierra común para todos los componentes.

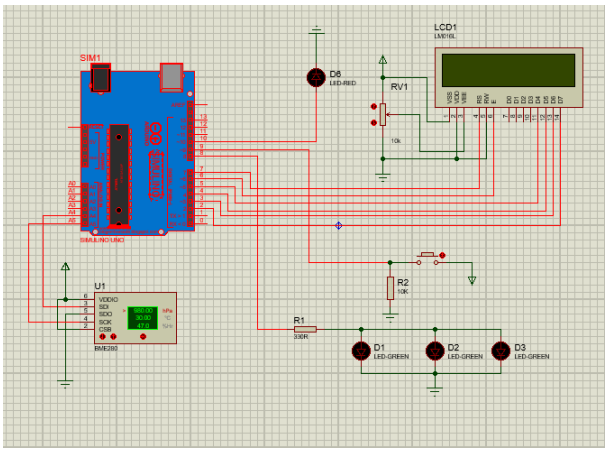

Figure 1: Diagrama de conexión

2.2 Desarrollo del código

2.2.1 Configuración de bibliotecas y definiciones de hardware

En esta sección, se incluyen las bibliotecas necesarias para el uso de FreeRTOS, el sensor BME280 y la pantalla LCD, además de definirse los pines de entrada y salida para los componentes electrónicos. Estas definiciones facilitan la conexión física y el manejo de cada componente dentro del código.

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <LiquidCrystal.h>

// Definiciones de hardware
#define BUTTON_PIN 9
#define LED_SERIES_PIN 8
#define LED_BLINK_PIN 10
#define LCD_RS 7
#define LCD_EN 6
#define LCD_D4 5
#define LCD_D5 4
#define LCD_D6 3
#define LCD_D7 2
```

2.2.2 Declaración de objetos y recursos de FreeRTOS

Aquí se inicializan los objetos necesarios para interactuar con el sensor BME280 y la pantalla LCD, además de declararse los recursos de FreeRTOS como un semáforo binario (xSerialSemaphore) para controlar el acceso al puerto serial y una cola (xSensorDataQueue) para la transferencia de datos entre tareas.

```
// Objeto para el sensor BME280 y LCD
Adafruit_BME280 bme;
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

// Definiciones de FreeRTOS
SemaphoreHandle_t xSerialSemaphore;
QueueHandle_t xSensorDataQueue;
```

2.2.3 Configuración inicial del sistema

En esta sección, se configura el sistema, incluyendo la inicialización de los pines, el sensor BME280, y la pantalla LCD. También se crean los recursos de FreeRTOS (semáforo y cola) y se definen las tareas con sus prioridades. Finalmente, se inicia el planificador de FreeRTOS, permitiendo la ejecución multitarea.

```
void setup() {
  // Inicializaci n de componentes
  pinMode(BUTTON_PIN, INPUT);
  pinMode(LED_SERIES_PIN, OUTPUT);
  pinMode(LED_BLINK_PIN, OUTPUT);

  lcd.begin(16, 2);
  if (!bme.begin(0x76)) {
    lcd.print("BME280 error!");
    while (1);
  }

  Serial.begin(9600);

  // Crear sem foro y cola
  xSerialSemaphore = xSemaphoreCreateBinary();
  xSemaphoreGive(xSerialSemaphore);
  xSensorDataQueue = xQueueCreate(5, sizeof(float));

  // Crear tareas
  xTaskCreate(taskButtonLEDs, "ButtonLEDs", 128, NULL, 2, NULL);
  xTaskCreate(taskReadBME280, "ReadBME280", 128, NULL, 3, NULL);
  xTaskCreate(taskBlinkLED, "BlinkLED", 128, NULL, 1, NULL);

  vTaskStartScheduler(); // Inicia el sistema FreeRTOS
}
```

2.2.4 Tarea 1: Control de LEDs mediante botón

Esta tarea verifica continuamente el estado del botón pulsador conectado al pin digital 9. Si el botón está presionado, enciende los LEDs en serie; de lo contrario, los apaga. La tarea se ejecuta periódicamente cada 100 ms para evitar lecturas innecesarias.

```
    void taskButtonLEDs(void *pvParameters) {
  for (;;) {
    if (digitalRead(BUTTON_PIN) == HIGH) {
      digitalWrite(LED_SERIES_PIN, HIGH);
    } else {
      digitalWrite(LED_SERIES_PIN, LOW);
    }
    vTaskDelay(pdMS_TO_TICKS(100)); // Espera 100 ms
```

```
  }
}
```

2.2.5 Tarea 2: Lectura de datos del sensor BME280

Esta tarea lee continuamente la temperatura del sensor BME280, mostrando los datos en la pantalla LCD y enviándolos al puerto serial. Utiliza un semáforo binario para asegurar el acceso controlado al puerto serial y envía los datos a la cola para que puedan ser utilizados por otras tareas.

```
    void taskReadBME280(void *pvParameters) {
  for (;;) {
    float temperature = bme.readTemperature();

    if (xSemaphoreTake(xSerialSemaphore, portMAX_DELAY)) {
      Serial.print("Temp: ");
      Serial.println(temperature);
      xSemaphoreGive(xSerialSemaphore);
    }

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temp:");
    lcd.setCursor(6, 0);
    lcd.print(temperature, 1);
    lcd.print("C");

    xQueueSend(xSensorDataQueue, &temperature, portMAX_DELAY);
    vTaskDelay(pdMS_TO_TICKS(500)); // Espera 500 ms
  }
}
```

2.2.6 Tarea 3: Parpadeo de LED

Esta tarea controla el parpadeo de un LED conectado al pin digital 10. El LED se enciende y apaga con un intervalo de 500 ms, demostrando el uso de tareas periódicas en FreeRTOS.

```
    void taskBlinkLED(void *pvParameters) {
  for (;;) {
    digitalWrite(LED_BLINK_PIN, HIGH);
    vTaskDelay(pdMS_TO_TICKS(500));
    digitalWrite(LED_BLINK_PIN, LOW);
    vTaskDelay(pdMS_TO_TICKS(500));
  }
}
```

2.2.7 Función principal (loop)

En sistemas basados en FreeRTOS, el ciclo principal (loop) no se utiliza, ya que todas las funcionalidades están delegadas a las tareas manejadas por el planificador del sistema operativo. Sin embargo, es necesario declararlo en el codigo para su funcionamiento.

2.3 Simulación

A continuación se muestra el funcionamiento del sistema realizando las conexiones necesarias para la practica, las cuales fueron mostradas en el primer esquema.

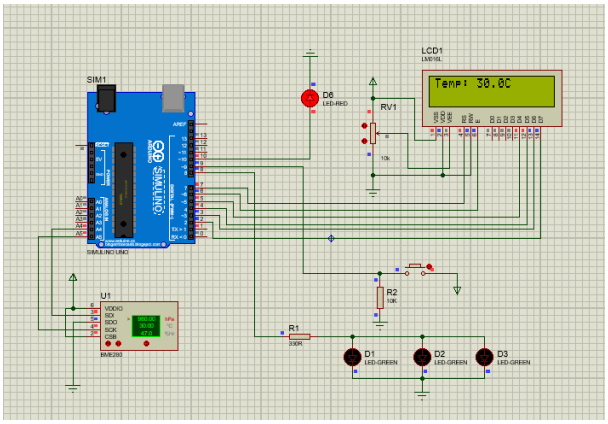

Figure 2: Funcionamiento del circuito

Primera imagen donde se puede observar como se inicializa el programa, mostrando los datos del BME280 en la pantalla LCD de manera correcta.

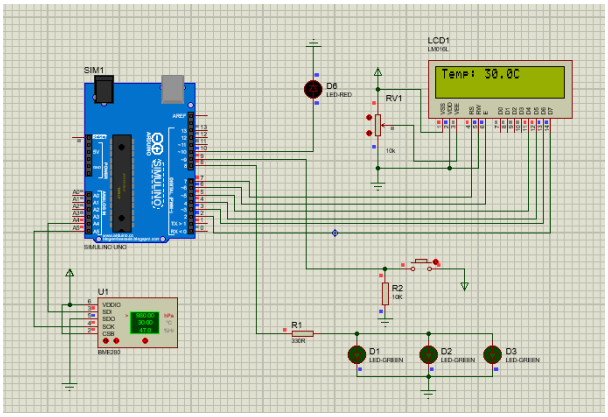

Figure 3: Simulación lectura de datos

En esta segunda imagen se puede observar que se realizan las tres tareas de manera simultánea (mediante los hilos). En este caso, al presionar el botón se prenden los leds en serie.

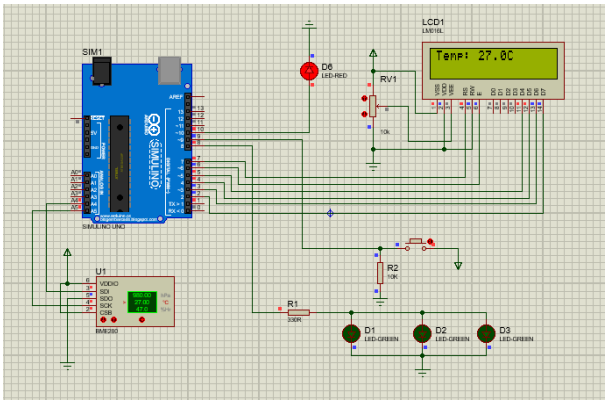

Figure 4: Simulación actualización de datos

Finalmente en esta simulación se pueden ver todas las tareas siendo ejecutadas de manera simultánea en el sistema, en el que incluso como se observa, se actualiza la lectura de los datos del BME280.

2.4 Implementación

Una vez mostrada la simulación del sistema utilizando proteus, se procede a mostrar el funcionamiento del sistema utilizando componentes reales y el mismo equema que se motro al principio de la práctica.

A continuación se muestran las fotografías de su correcto funcionamiento.

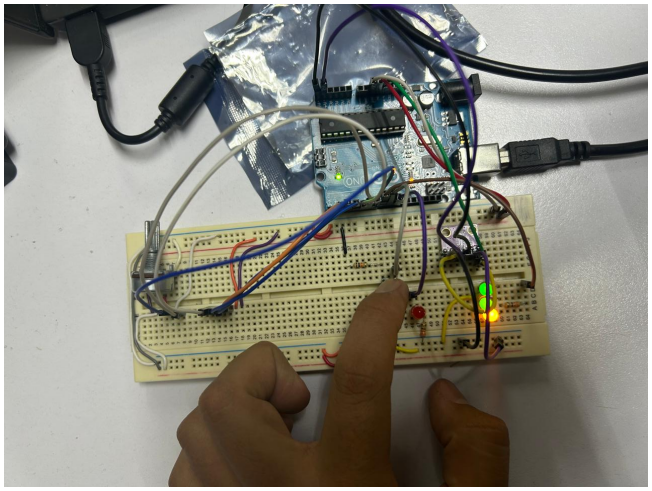

Figure 5: Funcionamiento del cricuito

En esta primera imagen se puede observar como al presionar el botón se encienden los leds en serie mientras el led rojo se mantiene parpadeando. Al mismo tiempo que se mantiene leyendo las lecturas del sensor BME280 (se tenía la conexión para mostrar en el lcd como se muestra en la imagen, sin emabrgo, también se muestra en el monitor serial.)

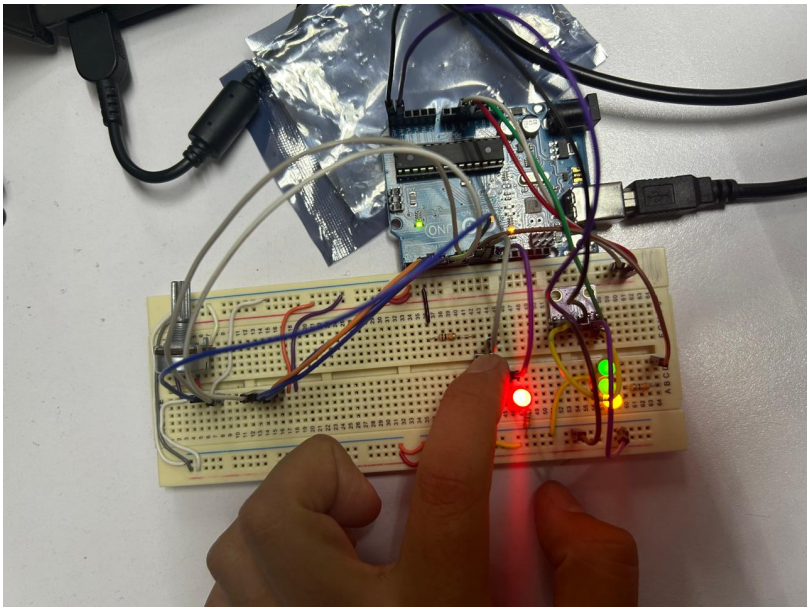

Figure 6: Funcionamiento aumento de temperatura

En esta segunda fotografía, aun presionando el botón, se puede observar como se enciende el led que se encuentra parpadeando mientras los leds seriales se prenden al presionar el push button.

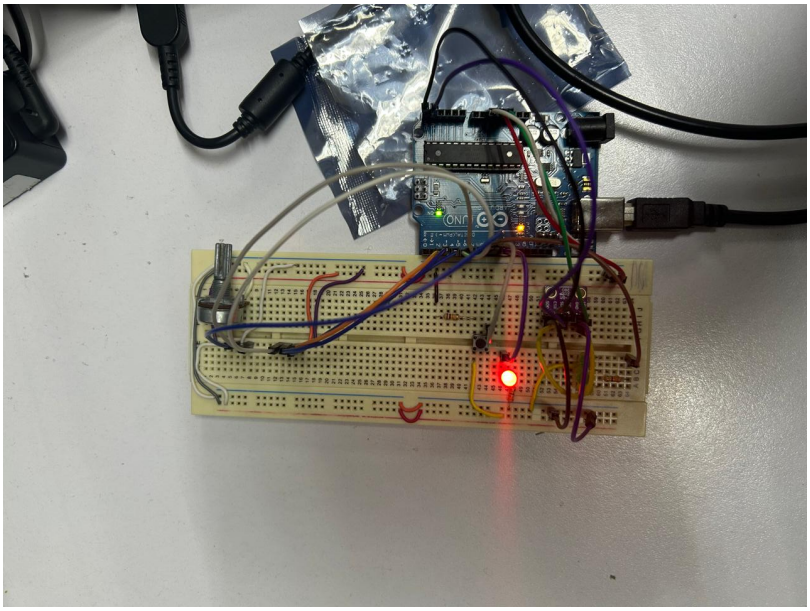

Figure 7: Funcionamiento actualización de voltaje

En esta tercera imagen se observa como al no presionar el boton, los leds seriales se apagan.

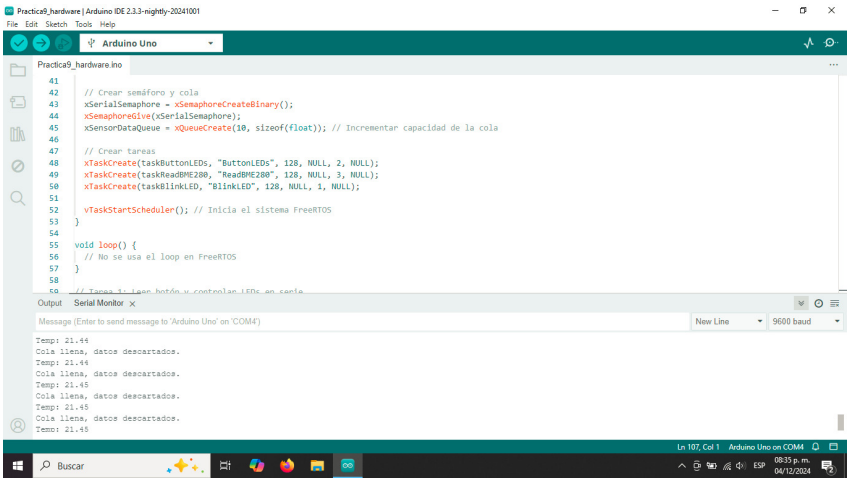

Figure 8: Caption

En esta captura de pantalla se puede observar como además, se leen de manera correcta los datos del sensor BME280 en el sistema, mostrandose de manera correcta en el monitor serial. Adicional a ello, se puede observa como al llenarse el chip de las tareas, lo manda a la cola en orden de prioridad.

3 Conclusiones

- **Morales Rodriguez Iñaki** La práctica demuestra cómo la integración de diversos componentes de hardware, como el sensor BME280, una pantalla LCD y LEDs controlados mediante un botón, puede manejarse de manera eficiente mediante el uso de FreeRTOS. La implementación de tareas multitarea permite distribuir las funcionalidades del sistema en bloques independientes: la lectura de datos del sensor, el control del LED en serie mediante el botón y el parpadeo periódico de un LED. Este enfoque garantiza la ejecución concurrente y ordenada de las actividades, optimizando los recursos del microcontrolador y asegurando un comportamiento predecible en sistemas embebidos. Además, el uso de recursos de sincronización como semáforos y colas destaca la importancia de la coordinación entre tareas para evitar conflictos en el acceso a recursos compartidos, como el puerto serial.

- **Luciano Hernández Jonathan** Esta práctica permite aplicar conceptos avanzados en sistemas embebidos, como la gestión de dispositivos de entrada y salida, la lectura de sensores y la implementación de sistemas interactivos en tiempo real. La configuración inicial del hardware y la integración de bibliotecas específicas, como Adafruit para el sensor BME280 y LiquidCrystal para la pantalla LCD, destacan la necesidad de diseñar software modular y reutilizable. Por otro lado, la estructuración del código basada en FreeRTOS refuerza la capacidad de implementar sistemas robustos y escalables, adecuados para aplicaciones industriales o proyectos avanzados. El resultado final es un sistema que demuestra el valor del diseño bien estructurado, facilitando la comunicación entre componentes y asegurando que cada tarea cumpla su función en tiempo y forma.

4 Referencias

1 ProgrammingKnowledge, "FreeRTOS With Arduino Tutorials 1 - Setting Up FreeRTOS on Arduino," YouTube, 10-Jul-2018. [Online].
Available: https://www.youtube.com/watch?v=PcE-2VjtYF4. [Accessed: 06-Jan-2025].

2 Programación de microcontroladores y electrónica. Introducción a la Programación de Microcontroladores Básicos 1. YouTube, 20 mar. 2024. [En línea]. Disponible en: https://www.youtube.com/watch?v=QqIsp6bjAwA. [Accedido: 2 dic. 2024].

3 M. Moreudev, "Simulando hardware en Proteus - Tutorial," YouTube, Oct. 2024. [Online]. Available: https://www.youtube.com/watch?v=ExQGweONqkU. [Accessed: Dec. 2, 2024].