Trabajo Práctico N°2 - Sistemas Embebidos

Licenciatura en Ciencias de la Computación, Facultad de Ingeniería, UNCuyo

Integrantes:

- Masuelli, Luciano
- Silva, Yeumen
- Yornet de Rosas, Agustín

Actividad 1

> **Enunciado 2.** Implementar una aplicación que realice las siguientes tareas usando FreeRTOS:

> **a.** Lea constantemente el valor de la intensidad luminosa

> **b.** Cada 3 segundos, envíe a través del puerto serial el último valor leído (Nota: la escritura en el monitor serial demora cierto tiempo. No debe ser interrumpida por otra tarea). Muestre el valor leído en una página web y en la aplicación Python.

Para el inciso `a` y `b`, se empleó el código que se encuentra en `src/main.cpp`. La siguiente porción de código es la encargada de declarar el valor donde se va a guardar la intensidad luminosa, un mutex para controlar el acceso a esta variable y una variable para saber cuando el sistema este activado gracias al inciso `c`.

```
int sensorValue = 0; // Variable para almacenar el valor
SemaphoreHandle_t xMutex;  // Mutex para proteger la variable compartida
```

A continuación se declaran las tareas que seran utilizadas para el problema.

```
void TaskAnalogRead(void *pvParameters);
void TaskSerialWrite(void *pvParameters);
```

Se crea el mutex y se definen las tareas dentro de setup.

```
void setup(){
    xMutex = xSemaphoreCreateMutex();
  if (xMutex == NULL) {
    Serial.println("Error creando el mutex");
    while (1) {}
  }

  xTaskCreate(TaskAnalogRead, "AnalogRead", 128, NULL, 1, NULL);
  xTaskCreate(TaskSerialWrite, "SerialWrite", 128, NULL, 2, NULL);
}
```

Las funciones ejecutan el siguiente código.

```
//Tarea para leer la intesidad luminosa
void TaskAnalogRead(void *pvParameters) {
  for (;;) {
    if (lecturaHabilitada) { //Si se presionó el botón para activar la
lectura
      int tempValue = analogRead(A3); //Se lee la intesidad
      if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
        sensorValue = tempValue; //Se toma el semáforo y se escribe el
valor
        xSemaphoreGive(xMutex);
      }
    }
    vTaskDelay(pdMS_TO_TICKS(15));
  }
}

//Tarea para escribir la intesidad luminosa en el puerto
void TaskSerialWrite(void *pvParameters) {
  for (;;) {
    if (lecturaHabilitada) { //Si se presionó el botón para activar la
lectura
      int valueToSend;
      if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
        valueToSend = sensorValue; //Se toma el semáforo para leer el valor
actual de la intensidad y se guarda en una variable
        xSemaphoreGive(xMutex);
      }
      Serial.println(valueToSend); //Se envia el valor por el puerto
    } else {
      Serial.println("--");
    }
    vTaskDelay(pdMS_TO_TICKS(3000));
  }
}
```

> **c.** La lectura puede iniciarse y detenerse a través de los pulsadores de la placa Arduino y desde botones en una página web.

Declaramos los pines tanto para iniciar como para detener la lectura.

```
#define BUTTON_START 2  // Pin para iniciar lectura
#define BUTTON_STOP 3   // Pin para detener lectura
```

Definimos funciones para iniciar y deneter la lectura

```
void iniciarLectura();
void detenerLectura();

void iniciarLectura() {
  lecturaHabilitada = true;
}

void detenerLectura() {
  lecturaHabilitada = false;
}
```

Dentro de setup declaramos los pines como INPUT y creamos dos interrupciones para cuando se presionen los botones tanto de activar lectura como de desactivar lectura.

```
void setup(){
    pinMode(BUTTON_START, INPUT);
    pinMode(BUTTON_STOP, INPUT);

    attachInterrupt(digitalPinToInterrupt(BUTTON_START), iniciarLectura,
RISING);
    attachInterrupt(digitalPinToInterrupt(BUTTON_STOP), detenerLectura,
RISING);
}
```

> **d.** Parpadee el led 11 cada un segundo si la lectura está activada. No parpadee el led 11 si la lectura está desactivada.

> **e.** Si detecta que el valor de intensidad luminosa supera 800, encienda una alarma que:

- > i. Haga parpadear el led 12 con periodo de 0.1 segundo
- > ii. Indique en la aplicación web la situación.
- > iii. La alarma se desactiva solo si la lectura se desactiva (la alarma no debe desactivarse si la lectura vuelve a estar por debajo de 800).

Para los incisos `d` y `e` comenzamos declarando los pines 11 y 12 y las respectivas tareas

```
const int led11 = 11;
const int led12 = 12;

void TaskBlinkLed11(void *pvParameters);
void TaskStartAlarm(void *pvParameters);
```

Dentro de setup declaramos los pines como salida y creamos las tareas

```
void setup(){
    pinMode(led11, OUTPUT);
    pinMode(led12, OUTPUT);

    xTaskCreate(TaskBlinkLed11, "BlinkLed", 128, NULL, 2, NULL);
    xTaskCreate(TaskStartAlarm, "StartAlarm", 128, NULL, 2, NULL);
}
```

Las tareas declaradas realizan lo siguiente

```
//Tarea para hacer parpadear el led 11
void TaskBlinkLed11(void *pvParameters) {
  for (;;) {
    if (lecturaHabilitada) { //Si se presionó el botón para activar la
lectura
      digitalWrite(led11, HIGH); //Se prende el foco
      vTaskDelay(pdMS_TO_TICKS(1000)); //Se espera un segundo
      digitalWrite(led11, LOW); //Se apaga el foco
      vTaskDelay(pdMS_TO_TICKS(1000));
    } else {
      digitalWrite(led11, LOW);
      vTaskDelay(pdMS_TO_TICKS(100));
    }
  }
}

//Tarea que enciende la alarma cuando la intesidad es mayor a 800
void TaskStartAlarm(void *pvParameters) {
  static bool alarmaActiva = false;
  for (;;) {
    if (lecturaHabilitada) { //Si se presionó el botón para activar la
lectura
      int actualIntensity;
      if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) { //Se toma el
semáforo
        actualIntensity = sensorValue; //Se guarda el valor y se libera el
semáforo
        xSemaphoreGive(xMutex);
      }
      if (actualIntensity > 800) { //Se chequea si el valor
        alarmaActiva = true;
      }
      if (alarmaActiva) { //Si la alarma está activa se enciende y apaga el
led 12
        digitalWrite(led12, HIGH);
        vTaskDelay(pdMS_TO_TICKS(100));
        digitalWrite(led12, LOW);
        vTaskDelay(pdMS_TO_TICKS(100));
      } else {
        vTaskDelay(pdMS_TO_TICKS(50));
      }
```

```
      } else {
        alarmaActiva = false;
        digitalWrite(led12, LOW);
        vTaskDelay(pdMS_TO_TICKS(100));
      }
    }
  }
```

La página web fue realizada con Flask para el BackEnd, y HTML, CSS y JavaScript para el FrontEnd.

- `app.py`: BackEnd de la página web.
- `static/js/script.js`: Script encargado de la comunicación entre la vista web y el BackEnd.
- `static/css/styles.css`: Hoja de estilos para la página.
- `templates/index.html`: Estructura de la página.

La información fue enviada utilizando JSON y para eso se creo una tarea específica

```
void TaskSerialCommand(void *pvParameters);

void setup(){
    xTaskCreate(TaskSerialCommand, "SerialCommand", 128, NULL, 2, NULL);
}

void TaskSerialCommand(void *pvParameters){
  for (;;) {
    if (Serial.available() > 0) {
      String command = Serial.readStringUntil('\n');
      command.trim();
      if (command == "ON") {
        iniciarLectura();
      } else if (command == "OFF") {
        detenerLectura();
      }
    }
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

Vista previa: