– A5: freeRTOS –

Objetivo:

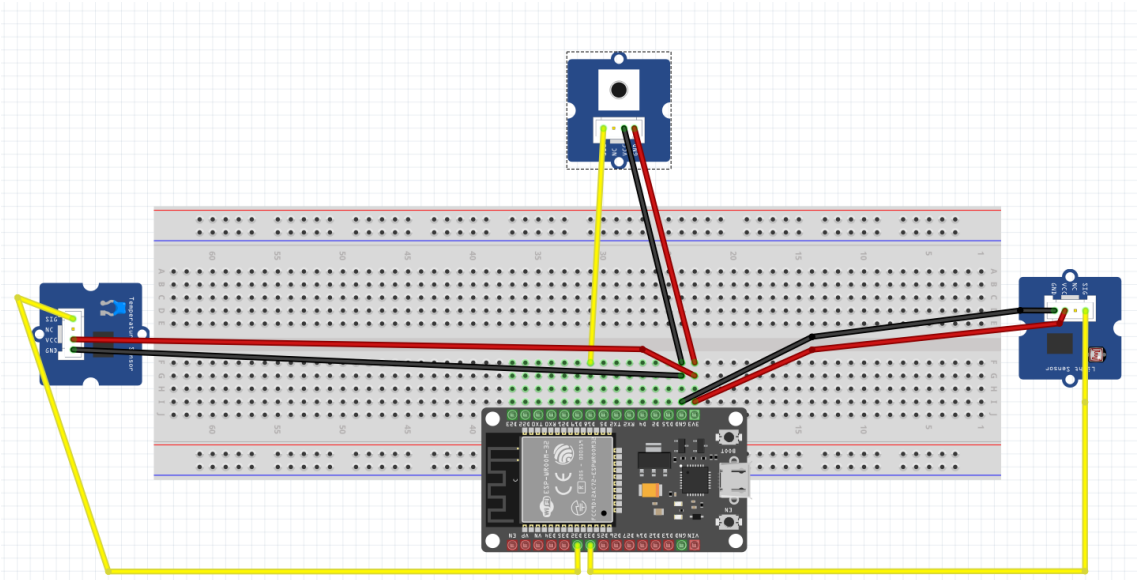
La dirección de la ESI ha vuelto a contactar con nosotros para mejorar el sistema creado para la obtención de la temperatura y luminosidad de las diferentes aulas del edificio Fermín Caballero (Ejercicio A12). En este caso la dirección de la ESI quiere añadir un botón que permita encender y apagar el sistema y cuya funcionalidad debe ser implementada sobre freeRTOS. A continuación, se detallan las características que debe tener el sistema:

• Los sensores de luminosidad y temperatura capturarán un nuevo dato cada 4 y 3 segundos respectivamente (siempre que el sistema esté activo).

• La información se enviará cada 2 segundos mediante un mensaje MQTT.

• Si un sensor no ha actualizado el valor de la medida se deberá enviar el valor anterior. También se deberá indicar si el dato enviado ha sido actualizado o no.

Esquema:



Componentes:

- Placa ESP32
- Light sensor
- Temperature sensor
- Botón

Explicación de la solución:

Antes de nada, debemos mencionar que para el funcionamiento de la práctica hemos hecho uso de colas, tal y como se nos recomendaba en el enunciado.

En el siguiente fragmento de código, quedan definidos los distintos pines que vamos a utilizar de la placa, los manejadores empleados para cada tarea, las dos colas para el envío de datos entre las tareas, constantes, configuración de ssid y contraseña, así como puertos, dirección IP y topic empleados en la conexión mqtt.

```
//PINES que se van a utilizar
#define btn 18 //ESP32 pin GPIO18 (BOTÓN)
#define LIGHT_SENSOR_PIN 32 // ESP32 pin GIOP32
#define TEMPERATURE_SENSOR_PIN 33 //ESP32 pin GPIO33

//manejadores de tareas
TaskHandle_t senderHandle;
TaskHandle_t lightHandle;
TaskHandle_t temperatureHandle;

//colas para el envío de datos entre tareas
QueueHandle_t xQueueLight;
QueueHandle_t xQueueTemp;

const int B = 4275;               // B value of the thermistor
const int R0 = 100000;            // R0 = 100k

//configuración de WI-FI y mqtt
const char* ssid = "AndroidApp";
const char* password = "12345678";

#define TOPIC "A12/NODOA" //TOPIC en el que publicará este nodo
#define BROKER_IP "192.168.43.174" //direccion IP del broker
#define BROKER_PORT 2883 //puerto en el que escucha el broker

WiFiClient espClient;
PubSubClient client(espClient);
```

A continuación, mostramos los métodos para la conexión Wi-FI y mqtt (explicados ambos en la tarea A12, por lo que no entraremos en mayor detalle).

```
void wifiConnect()
{
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}
```

```
void mqttConnect() {
  client.setServer(BROKER_IP, BROKER_PORT);
  while (!client.connected()) {
    Serial.print("MQTT connecting ...");

    if (client.connect("ESP32Client1")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, status code =");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");

      delay(5000);  //* Wait 5 seconds before retrying
    }
  }
}
```

Mediante el siguiente método (manejador de interrupciones), somos capaces de manipular el estado de las tareas a partir de la pulsación del botón, así como imprimir por pantalla su estado:

```
void isr(){

  if(eTaskGetState(senderHandle)==eSuspended){
    Serial.println("Resuming tasks");
    vTaskResume(lightHandle);
    vTaskResume(temperatureHandle);
    vTaskResume(senderHandle);
    } else {
      Serial.println("Suspending tasks");
      vTaskSuspend(lightHandle);
      vTaskSuspend(temperatureHandle);
      vTaskSuspend(senderHandle);
    }
}
```

A continuación veremos las tres tareas que hemos utilizado para implementar la solución. Hemos utilizado dos tareas para realizar las mediciones y una tercera tarea para realizar las publicaciones. Cada tarea de medición está conectada con la tarea de envío mediante una cola de tamaño uno para poder transmitir los datos que se van a publicar.

En la siguiente función se define la tarea referente a la medición de luz. Es muy similar al explicado en la anterior entrega, únicamente varía la adición y envío de los datos a la cola. Se añadirá el valor calculado a la cola para que, posteriormente, lo recoja la tarea encargada de publicar el mqtt. Se incorpora además un control de errores y un delay para definir el tiempo que debe transcurrir entre una medición y otra.

```
void taskLight(void * parameters){

  portBASE_TYPE xStatus;
  //const portTickType xTicksToWait = 500 / portTICK_RATE_MS;

  for (;;)
  {
    int lightValue = analogRead(LIGHT_SENSOR_PIN);
    lightValue = (lightValue*100)/4095;
    Serial.print("light percentage: ");
    Serial.println(lightValue);
    xStatus = xQueueSendToBack(xQueueLight,&lightValue,0); //se añade el valor medido a la cola para que lo recoja el método encargado de publicar en mqtt
    if(xStatus !=pdPASS) Serial.println("No se ha podido añadir elemento a la cola light");
    vTaskDelay(4000/ portTICK_PERIOD_MS); //espera de 4 segundos hasta la proxima medición
  }
  vTaskDelete(NULL);

}
```

De igual forma, se ha realizado la tarea propia de la medición de temperatura. La lógica de estas dos tareas es la misma que en la tarea A12 por lo que no entraremos en más detalles de como se han realizado las mediciones.

```
void taskTemperature(void * parameters){

  portBASE_TYPE xStatus;
  //const portTickType xTicksToWait = 500 / portTICK_RATE_MS;

  for(;;){
    int a= analogRead(TEMPERATURE_SENSOR_PIN);
    float R = 4095/a-1.0;
    R = R0*R;
    float temperature = 1.0/(log(R/R0)/B+1/298.15)-273.15;

    Serial.print("Temperatura: ");
    Serial.println(temperature);

    xStatus = xQueueSendToBack(xQueueTemp,&temperature,0);//Se añade el valor medido en la cola para que lo recoja el método encargado de publicar en mqtt
    if(xStatus !=pdPASS) Serial.println("No se ha podido añadir elemento a la cola temperature");
    vTaskDelay(3000/ portTICK_PERIOD_MS);//espera de 3 segundos hasta la proxima medición
  }
  vTaskDelete(NULL);
}
```

El siguiente método se encarga del envío y publicación de los valores medidos anteriormente:

```
void taskSend(void * parameters){

  portBASE_TYPE xStatusL;
  portBASE_TYPE xStatusT;

  //const portTickType xTicksToWait = 10000 / portTICK_RATE_MS

  //buffer con los ultimos valores obtenidos para saber si se
  int bufferLight=0;
  float bufferTemp=0;

  float temperature=0;
  int lightValue=0;

  for(;;){

    //timeout = 0 para que no se espere en caso de que no haya
    xStatusL = xQueueReceive( xQueueLight, &bufferLight, 0 );
    xStatusT = xQueueReceive( xQueueTemp, &bufferTemp, 0 );

    char chtemp[8];
    dtostrf(temperature,4,2,chtemp);
    char chlight[4];
    dtostrf(lightValue,3,0,chlight);

    char message[100] = "Temperature value = ";
    strcat(message,chtemp);
    strcat(message," C - Light percentage(%) = ");
    strcat(message, chlight);

    //Si no es pdPASS no se han actualizado los datos de los s
    if (xStatusT == pdPASS)
    {
      strcat(message," Temperature updated ");
      temperature = bufferTemp;
    }
    if (xStatusL == pdPASS)
    {
      strcat(message," Light updated ");
      lightValue = bufferLight;
    }

    client.publish(TOPIC,message);
    vTaskDelay(2000/portTICK_PERIOD_MS); //espera de 2 segundo
  }
  vTaskDelete(NULL);
}
```

Definimos los estados, buffers, y variables a utilizar.

En el bucle definimos el estado como la respuesta de la operación de recepción de la cola (estableciendo un timeout de 0 para evitar espera en caso de que no haya elementos disponibles en la cola).

Comprobamos que se hayan actualizado los datos, comparando el valor del estado de ambas variables con pdPass (que, básicamente, indica que la operación de recepción de datos de la cola ha sido un exito), añadiéndolo al mensaje a enviar.

Tras esto se publica el mensaje y se realiza la espera hasta la siguiente publicación.

En el método app_main se pone en funcionamiento el programa. Para ello, creamos las dos colas de paso de datos con un tamaño 1. En el caso de que la creación de ambas colas se haya producido sin errores; crearemos las tareas. Para las dos tareas de medición es suficiente con asignar 1000 de memoria. Sin embargo, para la tarea de envío de datos es necesario asignar más memoria porque al usarse conexión WI-FI se pueden producir excepciones como un stack overflow.

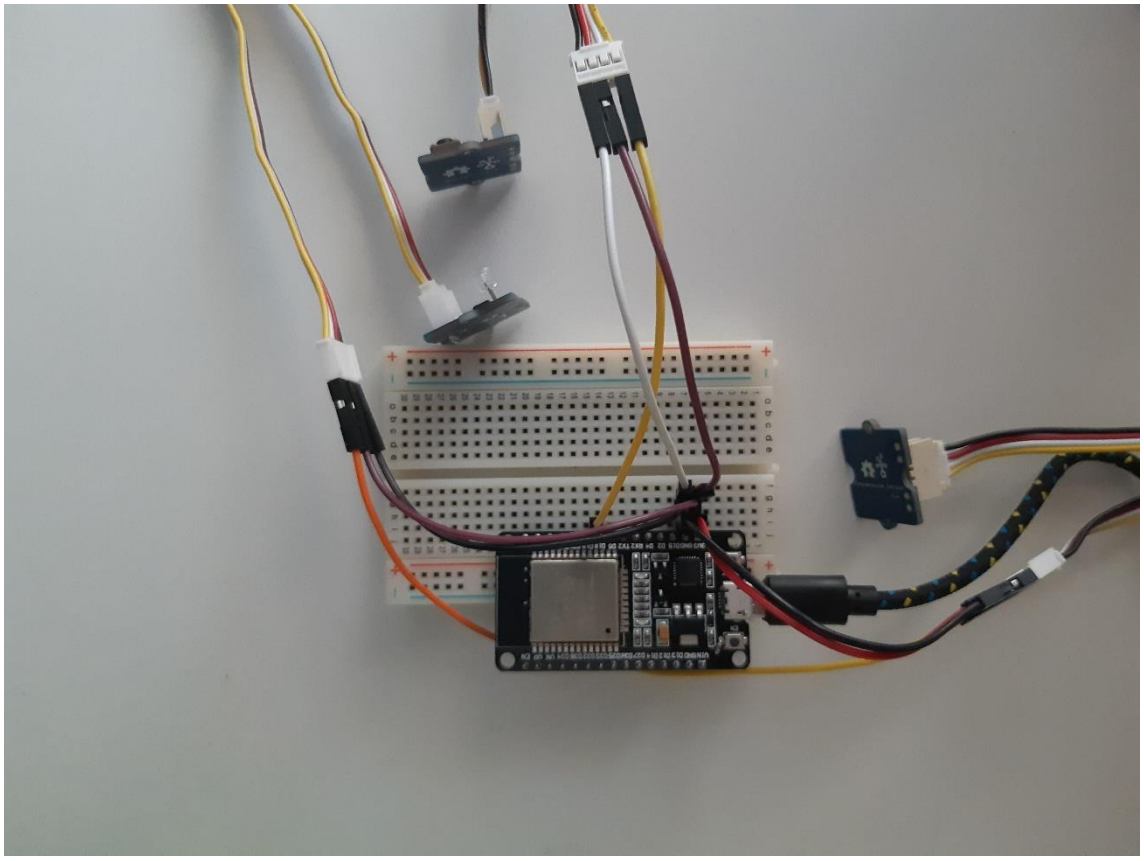
```
//método utilizado para la creacion de las tareas y las colas
void app_main(){

  //Creamos las colas con una capacidad máxima de 1
  xQueueLight = xQueueCreate( 1, sizeof( int ) );
  xQueueTemp = xQueueCreate( 1, sizeof( float ) );

  //Si no ha habido problemas en la creación de las colas creamos las tareas
  if (xQueueLight !=NULL && xQueueTemp!=NULL)
  {
    xTaskCreate(taskLight,"task light",1000,NULL,1,&lightHandle);
    xTaskCreate(taskTemperature,"task temperature",1000,NULL,1,&temperatureHandle);

    /*En esta tarea es necesario asignar más cantidad de memoria porque el uso del WI-F
    si no se le asigna memoria suficiente*/
    xTaskCreate(taskSend,"task Send",3000,NULL,1,&senderHandle);
  }

}
```

En el void setup implementamos el funcionamiento del programa mediante la declaración de los distintos métodos ya explicados.

```
void setup() {

  Serial.begin(9600);
  pinMode(btn,INPUT_PULLDOWN);
  attachInterrupt(btn,isr,RISING);

  wifiConnect();
  mqttConnect();
  app_main();

}
```

En esta ocasión no hacemos uso del void loop ya que los propios métodos hacen uso de bucles infinitos.

Imagen del circuito:



Salida del programa en el suscriptor

```
Temperature value = 25.00 C - Light percentage(%) =  56 Temperature updated
Temperature value = 25.00 C - Light percentage(%) =  56 Temperature updated  Light updated
Temperature value = 25.00 C - Light percentage(%) =  56
Temperature value = 25.00 C - Light percentage(%) =  56 Temperature updated
Temperature value = 25.00 C - Light percentage(%) =  56 Temperature updated  Light updated
Temperature value = 25.00 C - Light percentage(%) =  30 Temperature updated  Light updated
```