Por una universidad de **excelencia** y **solidaria**

Universidad del Cauca

ISO 9001:2015 SC- CER 450832

IQNet: CO- SC-CER450832

Sustentación final de sistemas embebidos y de tiempo real

Presentado por:

David Alejandro Ortega Flórez
Valentina Muñoz Arcos
Luís Miguel Gómez Muñoz

Arquitectura del sistema

Componentes del sistema

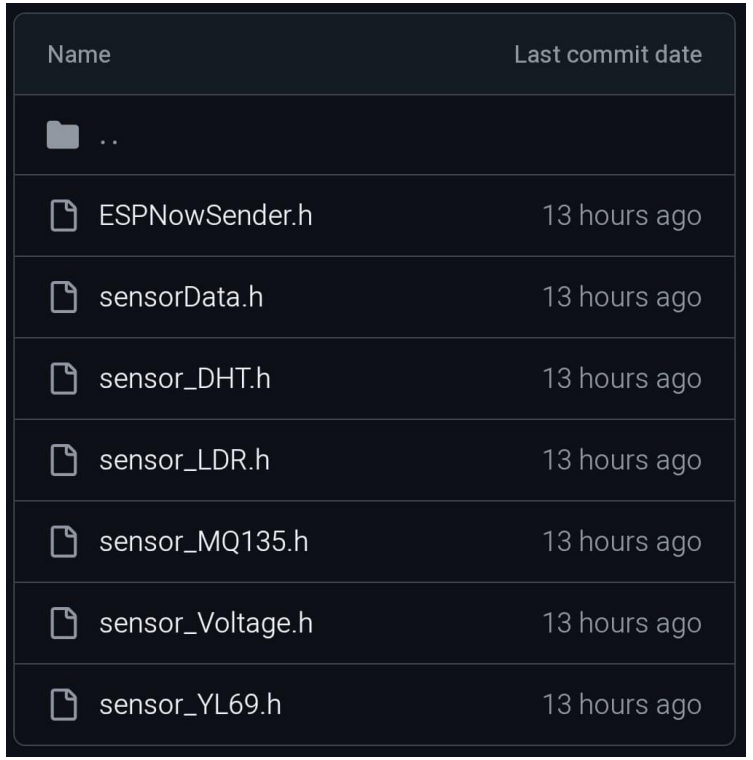
Este sistema está formado por 3 componentes principales:

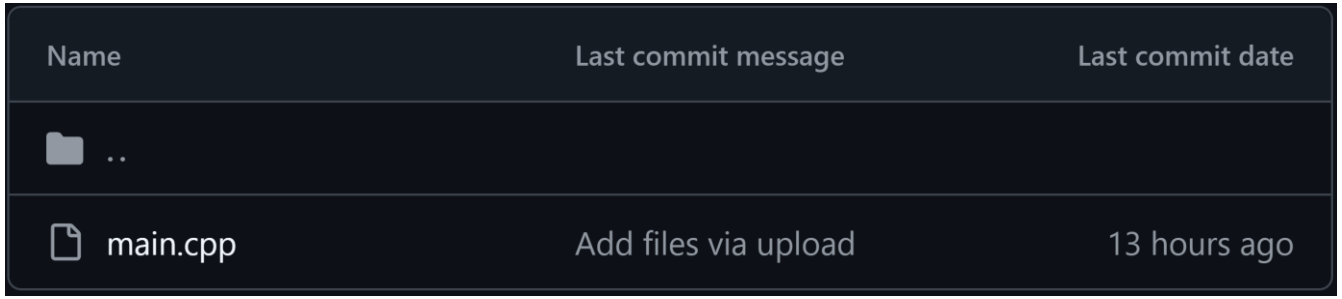
❖ PF-Sensores: Encargada de capturar y enviar los datos ambientales al dispositivo central.

❖ PF-Edge: Recibe la información de los sensores, procesa los datos y envía comandos a los actuadores.

❖ PF-Actuadores: Recibe los comandos del PF-Edge y controla físicamente los dispositivos (bomba, ventilador, LEDs).

Cada uno de estos 3 componentes cuenta con un archivo “main.cpp” que maneja la lógica del funcionamiento del sistema. Además, cada componente tiene una serie de “Headers” que complementan al código principal.





Por una universidad de excelencia y solidaria

Universidad del Cauca

Arquitectura del sistema

Uso de programación orientada a objetos (POO)

Un ejemplo del uso de POO se observa en el código principal del componente de sensores:

```
// Instancias
SensorDHT dhtSensor(DHT_PIN, DHT_TYPE);
SensorLDR ldrSensor(LDR_PIN);
SensorMQ135 mq135Sensor(MQ135_PIN);
SensorYL69 yl69Sensor(YL69_PIN);
VoltageSensor voltageSensor(VOLTAGE_PIN);
ESPNowSender espNowSender(receiverMac, CHANNEL);
```

- ❖ Se crean instancias de clases que son objetos concretos con estado y comportamiento propio.
- ❖ Cada sensor se representa como un objeto que encapsula sus propiedades y comportamientos.
- ❖ Los detalles internos de cómo cada sensor realiza sus mediciones están ocultos dentro de sus respectivas clases.
- ❖ Cada clase de sensor puede ser reutilizada en otros proyectos sin modificar su código interno.

FreeRTOS

Definición inicial de una tarea

En el código principal del componente de sensores, se define a la siguiente tarea:

- ❖ Se define a una función que será ejecutada como una tarea independiente. El parámetro void *parameter permite pasar datos a la tarea cuando se crea. Gracias al bucle while(true) se ejecuta indefinidamente.
- ❖ Se crea una estructura SensorData data para almacenar las lecturas de los sensores.
- ❖ Se envían los datos registrados y se pausa la tarea por 3 segundos antes de la siguiente ejecución.

```
// Tarea única: Leer sensores y enviar
void taskReadAndSend(void *parameter) {
  while (true) {
    SensorData data;

    // Leer sensores
    data.temperature  = dhtSensor.readTemperature();
    data.humidity     = dhtSensor.readHumidity();
    data.light        = ldrSensor.readLight();
    data.co2ppm       = mq135Sensor.readCO2();
    data.soilMoisture = yl69Sensor.readPercentage();
    data.voltage      = voltageSensor.readVoltage();

    // Enviar datos
    espNowSender.sendData(data);

    vTaskDelay(pdMS_TO_TICKS(3000)); // Ejecutar cada 3 segundos
  }
}
```

FreeRTOS

Ejemplos de creación de tareas

Un ejemplo del uso de FreeRTOS para la creación de una tarea se observa en el código principal del componente de sensores:

```
  // Crear una sola tarea
  xTaskCreatePinnedToCore(taskReadAndSend, "LeerYEnviar", 4096, NULL, 1, NULL, 1);
}
```

❖ La función “xTaskCreatePinnedToCore” crea una nueva tarea y la fija a un núcleo específico del procesador del ESP32. Aquí se implementa a “taskReadAndSend”, la función de tarea definida anteriormente.

❖ Se le da un nombre a la tarea, en este caso “LeerYEnviar”. El tamaño del Stack asignado a la tarea son 4096 Bytes.

❖ El primer NULL significa que no se pasa ningún parámetro a la tarea. El segundo NULL está relacionado al puntero al handle de la tarea, este valor significa que no se guardará este handle.

❖ El primer 1 es la prioridad de la tarea. En FreeRTOS la prioridad mínima para una tarea es 0. 1 es un valor medio bajo. El segundo 1 es el núcleo donde se ejecutará la tarea.

FreeRTOS

Ejemplos de creación de tareas

Otros ejemplos de creación de tareas con FreeRTOS son:

```
xTaskCreatePinnedToCore(ReceiveDataTask, "ReceiveData", 4096, NULL, 1, NULL, 0);
xTaskCreatePinnedToCore(TelegramReceiverTask, "Telegram", 4096, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(SDLoggerTask, "SDLogger", 4096, NULL, 1, NULL, 0);
xTaskCreatePinnedToCore(RTCUpdateTask, "RTCUpdate", 4096, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(checkRtcTime, "CheckRTC", 2048, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(DisplayTask, "Display", 4096, NULL, 1, NULL, 1);
```

Siguen la misma estructura ya explicada; Se crea la tarea, y se asigna la función que dicha tarea llamará al ser ejecutada, se le da un nombre, un tamaño a su Stack, se le asigna una prioridad y el núcleo en donde será ejecutada. A ninguna tarea se le pasan parámetros ni guardan el handle.

FreeRTOS

Exclusión mutua

Este sistema utiliza un mecanismo de exclusión mutua (mutex) específico para el ESP32 en entornos FreeRTOS, usado para proteger recursos compartidos entre tareas:

```
// Protecciones contra acceso concurrente
portMUX_TYPE dataMux = portMUX_INITIALIZER_UNLOCKED;
```

Se inicializa este mecanismo con portMUX_TYPE, que es una implementación optimizada para los núcleos Xtensa del ESP32. Y con portMUX_INITIALIZER_UNLOCKED se inicializa el mutex en estado desbloqueado.

Este mecanismo de sincronización previene el acceso concurrente a recursos compartidos, y evita condiciones de carrera cuando múltiples tareas acceden al mismo recurso. El mecanismo protege los datos leídos antes de enviarlos por ESP-NOW si múltiples tareas pueden acceder a ellos.

PF-Sensores

Librerías

Para el correcto funcionamiento de la red de sensores del sistema se utilizan las siguientes librerías:

❖ Arduino: Biblioteca base para programación en plataformas Arduino/ESP32.
❖ sensor_DHT.h: Control para sensores DHT.
❖ sensor_LDR.h: Manejo de fotorresistencias.
❖ sensor_MQ135.h: Es el driver para el sensor MQ135 (Detector de calidad de aire).
❖ sensor_YL69.h: Control para el sensor de humedad de suelo YL-69.

```
#include <Arduino.h>
#include "sensor_DHT.h"
#include "sensor_LDR.h"
#include "sensor_MQ135.h"
#include "sensor_YL69.h"
#include "sensor_Voltage.h"
#include "sensorData.h"
#include "ESPNowSender.h"
```

❖ sensor_Voltage.h: Control para la lectura de tensión eléctrica.
❖ sensorData.h: Definición de estructuras de datos para almacenar y transmitir las lecturas de los sensores.
❖ ESPNowSender.h: Implementación del protocolo ESP-NOW de Espressif para comunicación inalámbrica directa entre dispositivos ESP32 sin necesidad de WiFi.

PF-Sensores

Instancias

Las instancias creadas para la red de sensores son:

```
// Instancias
SensorDHT dhtSensor(DHT_PIN, DHT_TYPE);
SensorLDR ldrSensor(LDR_PIN);
SensorMQ135 mq135Sensor(MQ135_PIN);
SensorYL69 yl69Sensor(YL69_PIN);
VoltageSensor voltageSensor(VOLTAGE_PIN);
ESPNowSender espNowSender(receiverMac, CHANNEL);
```

Estas instancias representan cada componente hardware y software del sistema, cada una tiene su propio estado (Valores internos) y puede ejecutar operaciones específicas (métodos).

Para las instancias de los sensores, se requiere como parámetro el pin en donde están operando (El DHT adicionalmente requiere el tipo de sensor utilizado, el cual puede ser dht11 o dht22). Y para la instancia del ESPNowSender, se requiere la dirección MAC del dispositivo receptor y el canal de comunicación.

PF-Sensores

Emparejamiento con ESP-NOW

Dentro del Header “ESPNowSender.h”, se configura un dispositivo peer (par) para la comunicación ESP-NOW:

```
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, _peerAddress, 6);
peerInfo.channel = _channel;
peerInfo.encrypt = false;
peerInfo.ifidx = WIFI_IF_STA;
```

❖ Se define una estructura “esp_now_peer_info_t” que contiene toda la información necesaria para agregar un dispositivo receptor, se inicializa vacía para almacenar la configuración del peer.

❖ Para la configuración de los campos, primero se tiene a la dirección MAC, en donde se copian los 6 bytes de la dirección MAC del dispositivo receptor (_peerAddress) al campo peer_addr.

❖ Se establece el canal WiFi (0-14) donde ocurrirá la comunicación.

❖ Para esta implementación no se usa encriptación, por lo que se marca este campo como “False”.

❖ Para la interfaz WiFi, se especifica que usará en modo Station (STA).

PF-Sensores

Método lectura nivel de batería

Dentro del Header “sensor_Voltage.h”, se implementa un voltímetro digital:

```
float readVoltage() {
    int adcValue = analogRead(_adcPin);
    float vOut = (adcValue / (float)_adcMax) * _vRef;
    float vIn = (vOut * (_R1 + _R2) / _R2)*1.02266;
    return vIn;
}
```

❖ Se define una función “readVoltage” la cual devuelve el voltaje medido como un valor float.
❖ Se hace la lectura del ADC; Se lee el valor crudo del pin analógico y se guarda como un entero.
❖ El valor digital obtenido de la anterior lectura se convierte en un valor de voltaje, _vRef es el voltaje de referencia del ADC.
❖ Aplicando la fórmula de división de voltaje se calcula el voltaje real.
❖ Por último, se devuelve el voltaje calculado de la batería.

PF-Sensores

Estructura de datos

El Header “sensorData.h”, define una estructura de datos que actúa como un contenedor organizado para almacenar todas las lecturas de los sensores. Permite manejar todos los datos del sensor como una sola unidad.

```
struct SensorData {
    float temperature = 0.0f;
    float humidity = 0.0f;
    uint16_t light = 0;
    float co2ppm = 0.0f;
    float soilMoisture = 0.0f;
    float voltage = 0.0f;
};
```

Se tienen campos para todos los valores registrados, con su respectivo tipo de dato, y un valor predeterminado fijado a cero.

PF-Actuadores

Librerías

Para el correcto funcionamiento de la red de actuadores del sistema se utilizan las siguientes librerías:

```
#include <Arduino.h>
#include "ESPNowReceiver.h"
#include "Rele.h"
#include "LedRGB.h"
```

❖ Arduino.h: Es la biblioteca esencial, Proporciona las funciones básicas para programar en plataformas Arduino/ESP32.

❖ ESPNowReceiver.h: Biblioteca para la comunicación inalámbrica, implementa el receptor del protocolo ESP-NOW. Maneja direcciones MAC y deserialización de datos.

❖ Rele.h: Biblioteca para el control de actuadores eléctricos, gestiona relés o módulos de salida digital. Permite activar/desactivar cargas eléctricas, como bombas o luces.

❖ LedRGB.h: Biblioteca para el control de LEDs RGB.

PF-Actuadores

Instancias

Las instancias creadas para la red de actuadores son:

```
// Instancia del receptor
ESPNowActuatorReceiver receiver(1);  // Canal 1

// Actuadores
Rele bomba(PIN_BOMBA);
Rele ventilador(PIN_VENTILADOR);
LedRGB led1(PIN_LED_1);
LedRGB led2(PIN_LED_2);
LedRGB led3(PIN_LED_3);
LedRGB led4(PIN_LED_4);
```

Estas instancias Crean a los objetos que representan los componentes físicos del sistema de actuación.

❖ Se define al Receptor ESP-NOW, que recibe los datos de los sensores y como parámetro tiene al canal de escucha.

❖ Se definen a actuadores – relés para controlar a los dispositivos de potencia, que en este caso son la bomba y el ventilador, sus parámetros son los pines a los que están conectados.

❖ Las instancias para los LED’s tienen como parámetro el pin a donde están conectados.

PF-Actuadores

Estructura de datos

En el Header “dataActuator.h”, se define la estructura de datos para la red de actuadores, define el estado de los actuadores en el sistema, y funciona como un "interruptor maestro" digital para cada componente.

```
struct ActuatorState {
    uint8_t waterPump = 0;    // 0 = OFF, 1 = ON
    uint8_t fan = 0;          // 0 = OFF, 1 = ON
    uint8_t leds = 0;         // 0 = OFF, 1 = ON
};
```

Todos los actuadores inician con un valor cero (Apagado) por defecto. El tipo de dato “uint8_t” usa exactamente 1 byte por campo, esto para la optimización de memoria.

PF-Actuadores

Conexión

En el Header “ESPNowReceiver.h”, se prepara el módulo WiFi del ESP32 para establecer conexiones ESP-NOW:

```
WiFi.mode(WIFI_STA);
esp_wifi_set_channel(_channel, WIFI_SECOND_CHAN_NONE);
```

❖ Primero se realiza la configuración del WiFi; Se establece el ESP32 en modo Station (STA), lo que implica que el dispositivo actuará como cliente WiFi (aunque no se conectará a un router).

❖ En segundo lugar, se realiza la configuración del canal WiFi. Respecto a los parámetros de la función “esp_wifi_set_channel”, “_channel” es el número del canal, y “WIFI_SECOND_CHAN_NONE” desactiva cualquier canal secundario.

PF-Actuadores

Conexión recibiendo datos

En el Header “ESPNowReceiver.h”, se implementa el callback de recepción para datos ESP-NOW:

```
esp_now_register_recv_cb([](const uint8_t *mac, const uint8_t *incomingDataRaw, int len) {
    ActuatorState state;
    memcpy(&state, incomingDataRaw, sizeof(ActuatorState));
```

❖ “esp_now_register_recv_cb()” registra una función que se ejecutará automáticamente cuando lleguen datos. Sus parámetros son la dirección MAC del dispositivo que envió los datos, el puntero a los datos recibidos (incomingDataRaw) y la longitud de los datos en bytes (len).

❖ Se crea la variable local “ActuatorState state” para almacenar los datos decodificados.

❖ “memcpy()” convierte los bytes crudos (incomingDataRaw) a la estructura ActuatorState. State es el destino, incomingDataRaw es la fuente de datos, y sizeof(ActuatorState) es la cantidad de bytes a copiar.

PF-Edge

Librerías

❖ Arduino: Biblioteca base para programación en plataformas Arduino/ESP32.
❖ WiFiConnector.h: Manejo de conexiones WiFi
❖ ESPNowReceiver.h: Librería para recepción de datos mediante protocolo ESP-NOW.
❖ ESPNowSender.h: Envío de datos via ESP-NOW sin necesidad de Router WiFi.
❖ TelegramBot.h: Integración con Telegram para notificaciones/control remoto.
❖ time.h: Manejo de tiempo/fechas.

```
#include <Arduino.h>
#include "WiFiConnector.h"
#include "ESPNowReceiver.h"
#include "ESPNowSender.h"
#include "TelegramBot.h"
#include "ThresholdsController.h"
#include "dataSensor.h"
#include "dataActuator.h"
#include "SDLogger.h"
#include "RtcDS1302Helper.h"
#include "DisplayManager.h"
#include <time.h>
```

❖ ThresholdsController.h: Gestión de umbrales para activación automática de actuadores.
❖ dataSensor.h: Definición de estructuras para datos de sensores.
❖ dataActuator.h: Estructuras para control de actuadores.
❖ SDLogger.h: Registro de datos en tarjeta SD.
❖ RtcDS1302Helper.h: Manejo de reloj en tiempo real (RTC).
❖ DisplayManager.h: Control de pantalla local (OLED/LCD).

PF-Edge

Instancias

Las instancias creadas para la PF-Edge son:

```
// 🤖 Telegram
TelegramBot bot(botToken, chatId); ///< Instancia del bot de Telegram
int lastUpdateId = 0; ///< ID de la última actualización del bot

// 🧠 RTC y SD
RtcDS1302Helper rtc(15, 14, 2); ///< Instancia del RTC DS1302
SDLogger logger(5); ///< Instancia del logger de SD

// 🌡️ Datos compartidos
SensorData lastReceivedData; ///< Últimos datos recibidos de sensores
ActuatorState actuatorState; ///< Estado de los actuadores
bool hasData = false; ///< Indica si se han recibido datos

// Protecciones contra acceso concurrente
portMUX_TYPE dataMux = portMUX_INITIALIZER_UNLOCKED; ///< Mutex para proteger acceso a datos compartidos

// 📡 ESP-NOW
ESPNowReceiver receiver(2); ///< Receptor ESP-NOW en el canal 2

// 🟢 Actuadores
const uint8_t actuatorMAC[] = {0xEC, 0xE3, 0x34, 0x8A, 0x55, 0xA0}; ///< Dirección MAC del actuador
ESPNowActuatorSender actuatorSender(actuatorMAC, 2); ///< Instancia del sender ESP-NOW para actuadores

// Control de pantalla OLED
#define BUTTON_PIN 27 ///< Pin del botón para cambiar páginas
DisplayManager display(false); ///< Instancia del manejador de pantalla

// 🔢 Umbrales
Thresholds thresholds; ///< Instanci
```

PF-Edge

Instancias

❖ La instancia de TelegramBot “bot” está dedicada para la comunicación con Telegram; enviar y recibir mensajes. Sus parámetros son el Token del bot y la ID del último mensaje procesado.
❖ La instancia de RTC y SD se usa para registrar datos en la tarjeta SD.
❖ La instancia de datos compartidos almacena la última lectura de sensores y el estado actual de los actuadores, además, se cuenta con una bandera que indica si hay datos nuevos.
❖ La instancia de protección de datos usa el mecanismo Mutex ya explicado para acceso seguro a datos compartidos entre tareas/núcleos.
❖ Las instancias de ESP-NOW y de actuadores cumplen la función de recibir los datos en el canal 2 y enviar comandos a los actuadores.
❖ En la instancia de control de pantalla OLED se define un botón para navegar en la pantalla (GPIO 27), y con “Display” se controla.
❖ La instancia de umbrales almacena los umbrales para activación automática del sistema.

PF-Edge

Comandos de Telegram

```
void TelegramReceiverTask(void* pvParameters) {
    while (true) {
        if (thresholds.hasAlertChanged()) ...
        if (cmd == "/datos") ...
        else if (cmd == "/estado") ...
        else if (cmd.startsWith("/umbral ")) ...
        else if (cmd == "/umbrales") ...
        else if (cmd.startsWith("/activar ") || cmd.startsWith("/desactivar ")) ...
        else if (cmd == "/actuadores") ...
        else if (cmd == "/guia") ...
    }
}
```

El manejo de comandos recibidos desde Telegram para controlar y monitorear la red de sensores/actuadores se realiza como una tarea FreeRTOS que ejecuta un bucle infinito. Reacciona a cambios automáticos de los sensores o comandos de usuario, estos comandos son:

- /datos: Muestra las últimas lecturas de todos los sensores.
- /estado: Reporta estado del sistema y alertas activas.
- /umbral: Configura umbrales automáticos (3 parámetros requeridos).
- /umbrales: Lista todos los umbrales configurados.
- /activar: Activa un actuador específico.
- /desactivar: Desactiva un actuador específico.
- /actuadores: Muestra estado actual de todos los actuadores.

PF-Edge

Actualizar RTC con servidor de internet

Aquí se define una tarea encargada de mantener el Reloj en Tiempo Real (RTC) sincronizado con un servidor de tiempo de Internet (NTP).

```
void RTCUpdateTask(void* pvParameters) {
    while (true) {
        syncRtcWithNTP();
        vTaskDelay(1800000 / portTICK_PERIOD_MS);
    }
}
```

❖ Se conecta a Internet vía WiFi, se consulta un servidor NTP y se actualiza el RTC con la hora exacta.

❖ El intervalo de actualización es de 30 minutos para mantener un consumo energético adecuado.

PF-Edge

Tarea guardado en SD

Esta tarea es el módulo de data logging del sistema, responsable de guardar de forma segura todos los datos de sensores junto con metadatos críticos.

```
void SDLoggerTask(void* pvParameters) {
    int systemState = 0; // Estado del sistema (0: Todo OK, 1: Fallo conectividad, 2: Sensor desbordado, 3: Falla crítica)
    while (true) {
        portENTER_CRITICAL(&dataMux);
        bool available = hasData;
        SensorData copy = lastReceivedData;
        portEXIT_CRITICAL(&dataMux);

        if (thresholds.alertESPActuator || thresholds.alertWifi || thresholds.alertESPSensor) {
            systemState = 1; // Fallo de conectividad
        } else if (thresholds.anySensorAlertActive()) {
            systemState = 2; // Sensor desbordado
        } else if (thresholds.hasCriticalFailure()) {
            systemState = 3; // Falla crítica
        } else {
            systemState = 0; // Todo OK
        }

        if (available) {
            String timestamp = rtc.getTimestamp();
            logger.logSensorData(timestamp, "NODE_1", wifi.getRSSI(), copy, systemState);
        }

        vTaskDelay(10000 / portTICK_PERIOD_MS);

        thresholds.RSSIWiFi = WiFi.RSSI();
        receiver.update();
        thresholds.alertESPSensor = !receiver.isConnected();
    }
}
```

PF-Edge

Tarea guardado en SD

❖ Para la protección de datos se usa mutex (dataMux) para acceder a lastReceivedData de forma segura. Y se crea una copia local (copy) para evitar bloqueos prolongados.

❖ Para el sistema de datos, se clasifica el estado en 4 niveles: 0 para operación normal, 1 para problemas de conectividad, 2 para valores se sensores fuera del rango, y 3 para fallas críticas.

❖ El formato para el registro de los datos es: (timestamp, nodo, RSSI, temp, humedad, CO2, hum_suelo, voltaje, estado).

❖ Los metadatos incluidos son “WiFi.RSSI()” para la calidad de la señal WiFi, y “receiver.isConnected()” que es el estado de la conexión ESP-NOW.

PF-Edge

Tareas

Las tareas creadas con FreeRTOS para el PF-Edge son:

```
xTaskCreatePinnedToCore(ReceiveDataTask, "ReceiveData", 4096, NULL, 1, NULL, 0);
xTaskCreatePinnedToCore(TelegramReceiverTask, "Telegram", 4096, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(SDLoggerTask, "SDLogger", 4096, NULL, 1, NULL, 0);
xTaskCreatePinnedToCore(RTCUpdateTask, "RTCUpdate", 4096, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(checkRtcTime, "CheckRTC", 2048, NULL, 1, NULL, 1);
xTaskCreatePinnedToCore(DisplayTask, "Display", 4096, NULL, 1, NULL, 1);
```

Y las funciones de estas tareas son:

- ReceiveDataTask: Recibe datos de sensores via ESP-NOW y actualiza variables compartidas
- SDLoggerTask: Registra datos en tarjeta SD cada 10 segundos con timestamp y estado del sistema
- TelegramReceiver: Maneja comandos de Telegram y envía notificaciones.
- RTCUpdateTask: Sincroniza el reloj RTC con servidores NTP cada 30 minutos.
- checkRtcTime: Verifica validez de la hora RTC cada minuto.
- DisplayTask: Actualiza la pantalla OLED y maneja el botón de navegación.

Por una universidad de **excelencia** y **solidaria**

Universidad del Cauca

¡Gracias por su atención!

www.unicauca.edu.co