Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de la Matanza
ISSN: 2525-1333
Vol.: 5 - Nro. 1 (AGOSTO-2020)

Comunicación en Congreso

ACCESO REMOTO A DISPOSITIVOS IOT MEDIANTE TÉCNICAS DE MENSAJERÍA EMPLEANDO BOTS Y FREERTOS

REMOTE ACCESS TO IOT DEVICES THROUGH MESSAGING TECHNIQUES USING BOTS AND FREERTOS

Carlos Alberto BINKER [(1)], Hugo TANTIGNONE [(1)], Eliseo Alfredo ZURDO [(1)], Guillermo BURANITS[(1)]

[(1)]Departamento de Ingeniería e Investigaciones Tecnológicas. UNLAM

cbinker@unlam.edu.ar
htantignone@unlam.edu.ar
ezurdo@alumno.unlam.edu.ar
gburanits@unlam.edu.ar

Resumen:

Este trabajo tiene como eje principal gestionar un hardware IOT remotamente mediante mensajería instantánea de texto, empleando Telegram. Telegram permite definir bots (aféresis de Robots) y administrarlos a través de un token que la misma aplicación genera en forma aleatoria a través de su bot principal, denominado BotFather (el padre de todos los bots). Telegram brinda una API , que permite que los bots interactúen con nuestro sistema. En nuestro caso, esa interacción se dará para controlar un microcontrolador que opera con dispositivos IOT, el destacado chip ESP32. El ESP32 es un SOC (System on chip). Este SOC incorpora WIFI, bluetooth, sensores, conversores AD y DA, etc. Como caso de estudio se propone controlar un conjunto de leds, empleando el ESP32 usando un bot, por lo que se requerirá el uso de Telegram (versión web o móvil) y el entorno de programación Arduino. Además como valor agregado se hará uso de multitarea empleando ambos núcleos del SOC utilizando técnicas FreeRTOS.

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

Abstract:

The focus of this work is to manage an IOT hardware remotely through instant text messaging, using Telegram. Telegram allows you to define bots (apheresis of Robots) and manage them through a token that the same application randomly generates through its main bot, called BotFather (the father of all bots). Telegram provides an API, which allows bots to interact with our system. In our case, that interaction will occur to control a microcontroller that operates with IOT devices, the outstanding ESP32 chip. The ESP32 is a SOC (System on chip). This SOC incorporates WIFI, Bluetooth, sensors, AD and DA converters, etc. As a case study, it is proposed to control a set of LEDs, using ESP32 using a bot, so the use of Telegram (web or mobile version) and the Arduino programming environment will be required. In addition, as an added value, multitasking will be used using both SOC cores using FreeRTOS techniques.

Palabras Clave: *Mensajería, Robots, Sensores, Internet de las cosas, API*

Key Words: *Telegram, Bots, Sensors, IOT, API*

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

1. Introducción

Cuando se quiere acceder a un host en internet se lo puede hacer a través de su dirección IP o de su nombre, que se extrae de una tabla donde se encuentran mapeadas las direcciones IP con sus respectivos nombres de Host (El Domain Net System o DNS, se encarga de actualizar las tablas de direcciones y nombres). Con el crecimiento de internet y la proliferación de computadoras personales y dispositivos dentro de las redes privadas con servicios de comunicación y acceso a internet brindados por ISPs se hizo necesario el desarrollo de un método para acceder desde internet a los hosts de la red local conectada al router. La asignación de la dirección IP para los clientes se puede hacer en forma estática (se mantiene la dirección IP a lo largo del tiempo) o en forma dinámica (la dirección IP puede variar en el tiempo). La asignación dinámica le impediría a un cliente externo tener la dirección actualizada a la cual comunicarse, por lo cual será necesario un servicio de publicación de las direcciones IP actualizadas. Ver FIGURA. 1. Para evitar esto, ya que no siempre es fácil tener una dirección IP pública se propone la solución de mensajería por medio de Telegram mediante el empleo de bots. En nuestro caso nuestro host es un SOC ESP32, que a través de su IP accedemos a él a través de una red WIFI conectada a Internet. El SOC estará configurado en modo STATION y tomará su dirección IP mediante DHCP.

2. Descripción de la tecnología de mensajería Telegram y del SOC ESP 32

2.1 Esquema de la plataforma de mensajería instantánea Telegram

Telegram está creado sobre una serie de servidores distribuidos. Dichos servidores, para comunicarse entre sí, utilizan un protocolo propio llamado MTProto. La razón del uso de este protocolo propietario es la mejora en seguridad como también el envío de mensajes, sobre todo vídeo e imagen. Haciendo un poco de historia podemos ver que hay dos versiones de MTProto. La primera versión se utilizó en 2014. Los mensajes en MTProto eran cifrados con el algoritmo SHA-1. Por un reporte en enero de 2015, en donde el investigador Juliano Rizzo reveló un error en el funcionamiento de SHA-1 que originó una vulnerabilidad al interceptar los mensajes, en 2016 se señaló un posible reemplazo del SHA-1 por el SHA-2. Por lo tanto en 2017, se lanza la segunda versión. El cifrado se reemplazó a SHA-256 con mayor cantidad de bytes de carga útil. Para complementar lo dicho observar la FIGURA 2.

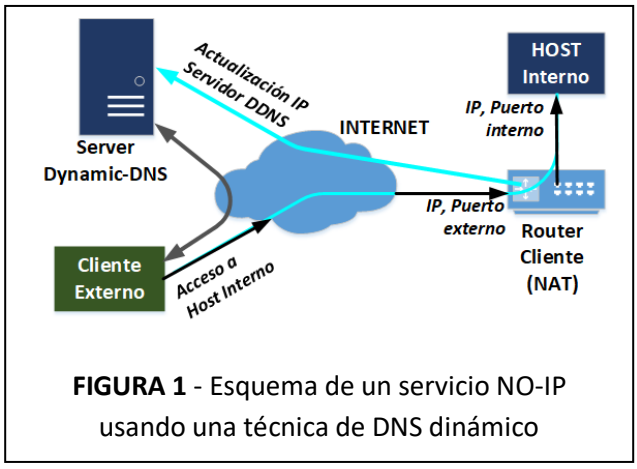

FIGURA 1 - Esquema de un servicio NO-IP usando una técnica de DNS dinámico

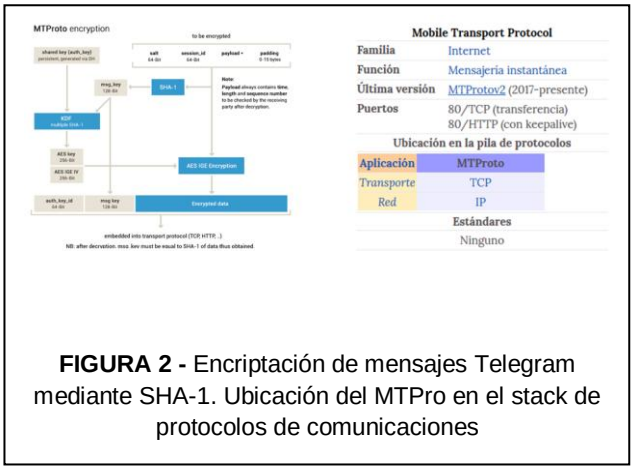

FIGURA 2 - Encriptación de mensajes Telegram mediante SHA-1. Ubicación del MTPro en el stack de protocolos de comunicaciones

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

2.2 API de Telegram asociada para el manejo de bots

La API de Telegram permite la interacción de los bots con un sistema. En nuestro caso ese sistema será un hardware a controlar remotamente a través de una interfaz de usuario, que es en concreto el bot. Los bots de Telegram son cuentas especiales que no están ligadas a un número de teléfono. Al igual que la cuenta de un usuario, el acceso a la misma es por medio del Alias, que se accede anteponiendo @ al nombre del bot, o bien se accede por el propio nombre. Estas cuentas sirven como una interface para albergar código que puede estar ejecutándose en cualquier lugar del mundo en un servidor. Su uso es totalmente transparente, ya que los usuarios no necesitan conocer nada absolutamente sobre como el protocolo de encriptación MTProto funciona. Los servidores de Telegram manejarán todo lo referente a la encriptación de los mensajes y la comunicación con la API de una manera muy sencilla. Esta comunicación con los servidores de Telegram a través de la API se da vía una simple conexión https. Todas las consultas a la API Telegram Bot deberán realizarse de esta manera:

https://api.telegram.org/bot<token>/METHOD_NAME

Se creará un bot que suministra los comandos anteponiendo el símbolo "/". Dicho bot se denomina CONAIISI_2019. A título de ejemplo con el token suministrado para CONAIISI_2019, la forma de acceder desde la web sería:

https://api.telegram.org/botxxxx32320:AAGgmhH6_H69CiQcbr4hPS-CZs0M18oxxxx/getme

La API soporta los métodos http GET y POST. Ante una petición (request), la respuesta de la API bot es un objeto JSON como el siguiente:

{"ok":true,"result":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"}}

Tal como se observa, la API siempre devuelve un campo de tipo Boolean denominado "ok" y otro campo denominado "result", en donde puede tener un campo opcional String denominado "description" con una descripción del resultado. Si la petición hacia el bot fue satisfactoria, el campo "ok" recibido en el objeto JSON es igual a true y si además el resultado de la consulta fue localizado se nos devolverá el campo "result" con todos los resultados, tal como se indica en el ejemplo precedente En cambio, si la respuesta del campo "ok" es igual a false, se devolverá un código de error, tal como el siguiente ejemplo.

{"ok":false,"error_code":401,"description":"Unauthorized"}

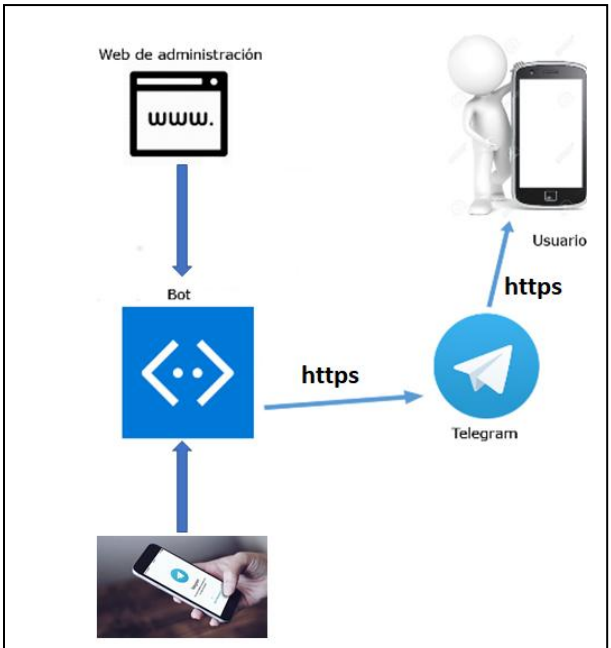

FIGURA 3 - Interacción entre usuario (cliente), el bot, la API de Telegram y el hardware remoto

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

En la FIGURA 3 se puede observar claramente la interacción entre todos los componentes, es decir entre el cliente, el bot, la API de Telegram y el hardware remoto a controlar

2.3 Descripción del SOC ESP32

Este dispositivo es un SOC que está adquiriendo una enorme popularidad debido a su bajo costo y la implementación de diferentes entornos de programación, tales como LUA y Python. Pero su enorme potencial radica en el hecho de que soporta también el IDE de Arduino y trabajar con él es como si realmente estuviéramos trabajando sobre alguna de las placas Arduino, que como ya sabemos goza de una enorme comunidad abocada al desarrollo de código y de librerías. Todo esto puede ser utilizado en forma transparente para este dispositivo y así evitamos la utilización de un doble microcontrolador. Este dispositivo se destaca por sobre todo porque ya resuelve todo lo concerniente a la comunicación de dispositivos a través de redes, ya que soporta WIFI, Bluetooth, Hardware criptográfico, SPI, I2C, I2S, Ethernet, etc. Además, incorpora un sensor que mide la temperatura interna del dispositivo y sensores Touch capacitivos y de efecto Hall, además de conversores DAC y ADC. Posee un microprocesador con dos núcleos paralelo. Cualquier pin del ESP32 puede configurarse para el control de interrupciones. Para más detalle se expone el diagrama en bloques para dar una idea general de su potencialidad, como así también el correspondiente PIN-OUT, (ver FIGURA 4 y FIGURA 5 respectivamente).

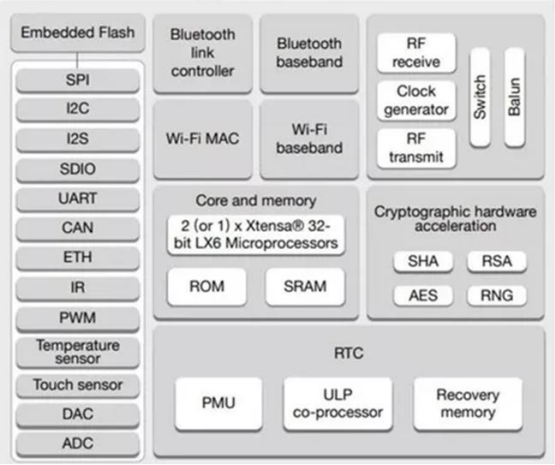

FIGURA 4 - Diagrama en bloques del ESP32

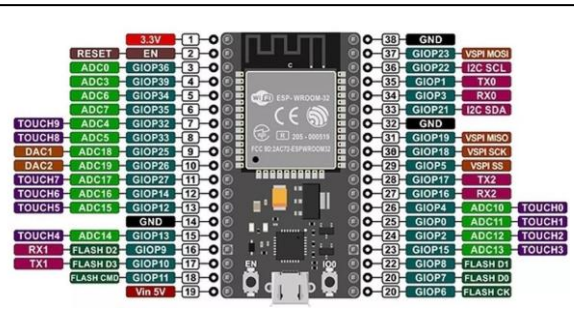

FIGURA 5 – Pin out ESP32. Placa Nodemcu ESP 32 DEV Module

3. Descripción de un Sistema RTOS

Un Sistema RTOS es un sistema operativo de tiempo real. Existen muchas aplicaciones en las cuales los tiempos de respuesta son un factor decisivo para el correcto funcionamiento, como por ejemplo los mandos de un avión o el sistema de accionamiento ABS de frenado en un automóvil son ejemplos más que ilustrativos de la exigencia requerida. El concepto es simple, dos tareas que pueden correr o no en un mismo núcleo realizan operaciones diferentes. Para conectar dichas tareas es necesario la existencia de una cola que

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

transfiera los datos de una tarea a la otra. Por ejemplo una tarea A es productora de datos y otra tarea B es consumidora de los datos que genera la tarea A. Dichos datos se envían a través de una cola que deberá crearse previamente. A continuación se ilustra el procedimiento descripto en la FIGURA 6.

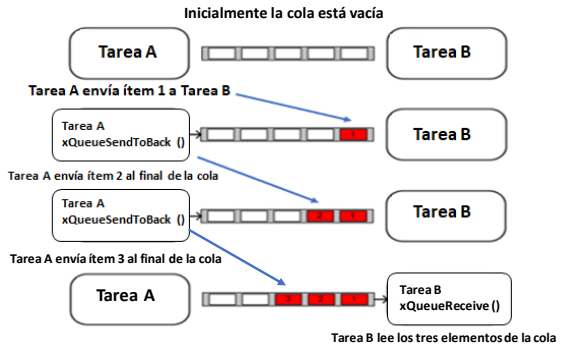

FIGURA 6 – Envío de datos de una tarea a otra a través de una cola

xQueueCreate
[Queue management]

queue.h

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLenght,
                            UBaseType_t uxITemSize ) ;
```

xQueueSend
[Queue Management]

queue.h

```
BaseType_t xQueueSend( QueueHandle_t xQueue,
                       const void * pvItemToQueue,
                       TickType_t xTicksToWait
                     );
```

xQueueReceive
[Queue Management]

queue.h

```
Base_Type xQueueReceive(
                          QueueHandle_t xQueue,
                          void * pvBuffer,
                          TickType_t xTicksToWait
                        ) ;
```

FIGURA 7 – Funciones xQueueCreate, xQueueSend y xQueueReceive, elementos básicos de RTOS.

Existen 3 funciones principales básicas que son las que se utilizarán en toda experiencia de RTOS: Las funciones son: *xQueueCreate*, *xQueueSend* y *xQueueReceive*. La descripción de las mismas se detalla en la FIGURA 7.

Finalmente y yendo al caso concreto del ESP32, el framework de Arduino ya viene montado sobre un sistema operativo RTOS en donde las funciones setup() y loop() se implementan de la siguiente manera:

main.cpp

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp-task-wdt.h"
#include "Arduino.h"
   TaskHandle-t loopTaskHandle = NULL;
   #if CONFIG_AUTOSTART_ARDUINO
   bool loopTaskWDTEnabled;
   void loopTask(void *pvParameters)
   {
      setup();
      for(;;) {
         if(loopTaskWDTEnabled){
            esp-task-wdt-reset();
         }
         loop();
      }
   }
   extern "C" void app-main()
   {
      loopTaskWDTEnabled = false;
      initArduino();
     xTaskCreateUniversal(loopTask, "loopTask", 8192,
NULL, 1, &loopTaskHandle,
CONFIG_ARDUINO_RUNNING_CORE);
   }
```

#endif

Recordemos que para invocar al bot se usa el alias y este se corresponde con el nombre del bot. Para acceder en forma web debe escribirse el siguiente comando:

t.me/c2ing59_bot

(acceso al bot por comandos utilizando "/"), donde en este caso c2ing59_bot es el username del bot y se corresponde con el Alias (@c2ing59_bot).

4. Caso de estudio empleando el SOC ESP32, Telegram y el IDE de Arduino con FreeRTOS

4.1 Prototipo a emplear en el caso de estudio

El prototipo de hardware que se utilizará es el mostrado a continuación en la FIGURA 8:

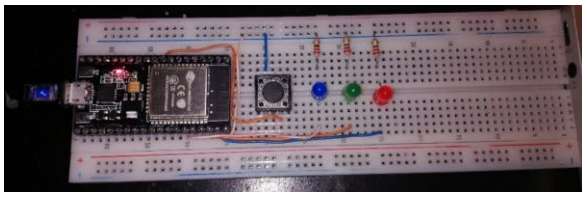

FIGURA 8 – Hardware a controlar por el bot de Telegram empleando el NodeMCU ESP32

FIGURA 9 – Bot que controlará el hardware (CONAIISI_2019), version móvil por comandos

FIGURA 10 – Bot que controlará el hardware (CONAIISI_2019), version web por comandos

El bot que lo controlará tiene el siguiente aspecto (ver FIGURA 9 y FIGURA 10), el mismo fue configurado a través de los comandos del BotFather, y lo apreciamos tanto en su versión web cómo móvil.

5. Desarrollo de la experiencia

El software que controlará remotamente el hardware por medio del bot es el siguiente (se exponen las partes más sobresalientes y se omiten algunos saltos de líneas), dada su extensión:

5.1 Las librerías a emplear son las siguientes:

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <TelegramBotClient.h>
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

Para el correcto funcionamiento, debe estar instalada la librería ArduinoJson version 5.13.5, ya que es una dependencia del resto de las librerías intervinientes.

5.2 Variable de entorno para el sensor de temperatura interno del ESP32

```
#ifdef __cplusplus
extern "C" { uint8_t temprature_sens_read();}
#endif
```

5.3 Definición de variables globales y constantes:

```
// Defino un PULSADOR en el GIOP 25, el cual se
// corresponde con el pin 9 del ESP32 Dev Module
const int pulsador=25;
// Defino los leds en los pines 23, 22 y 21, los cuales se
// corresponden con los pines 37, 36 y 33 del ESP32 Dev
// Module
const int led_rojo=23;
const int led_verde=22;
const int led_azul=21;
// Defino las constantes a emplear en el switch (ver
Tarea2)
#define ROJO 23
#define VERDE 22
#define AZUL 21
#define PRENDER  1
#define APAGAR  -1
//Estas variables determinarán el status del sistema a
//controlar, es decir qué leds están encendidos o
apagados
byte rojo = 0;
byte verde = 0;
byte azul = 0;
//Defino el Descriptor para la Cola de Mensajes
QueueHandle_t xqueue;
//Defino el Descriptor para Tarea2
TaskHandle_t TaskHandle2 = NULL;
```

5.4 Código para manejo de interrupciones:

```
portMUX-TYPE mux =
portMUX_INITIALIZER_UNLOCKED;
// Cuando se presiona el pulsador esta variable pasa a
true
bool pirTriggered = false;
//Código de la rutina de interrupción del pulsador por
//flanco descendente “Rising”
void IRAM_ATTR handleInterruptRising()
{
Serial.println("handleInterruptRising->"+
String(xPortGetCoreID()));
  portENTER_CRITICAL_ISR(&mux);
  Serial.println("Rising");
// Al ingresar a este rutina pirTriggered se pone a true
  pirTriggered = true;
  portEXIT_CRITICAL_ISR(&mux); }
// El chatId correspondiente al alias del usuario que
// interactúa con el bot
long chatId;
// Mensaje a enviar al bot
String msg = "Pulsador presionado";
```

5.5 Credenciales WIFI e instanciación de objetos

```
  //Instanciación de las credenciales para la conexión
WIFI //(ssid y password)
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
const char* ssid     = " IPLAN-306232 ";
const char* password = "xxxxxxxxx";

//Instanciación del Bot de Telegram dando el token
//suministrado por BotFather

//Nombre de Bot: CONAIISI_2019
const String botToken = "xxxx32320:AAGgmhH6_H69CiQcbr4hPS-CZs0M18oxxxx";

//Instanciación del cliente ssl utilizado para comunicarse //con la web API de Telegram

WiFiClientSecure sslPollClient;

//Instanciación de la clase client pasando como
//argumentos el Token del Bot y un cliente seguro ssl

static TelegramBotClient client(botToken, sslPollClient);
```

5.6 Declaración de función destinada a recibir mensajes desde el Bot de Telegram

```
void onReceive (TelegramProcessError tbcErr, JwcProcessError jwcErr, Message* msg)
```

Esta función corresponde a un evento que se disparará ante un comando suministrado al bot. Dicho evento funcionará en el núcleo 1. A tal efecto se imprime en el monitor serie dicho suceso con el siguiente comando:

```
Serial.println("onReceive En núcleo -> " + String(xPortGetCoreID()));
```

La variable xPortGetCoreID(), suministra el núcleo en donde se está ejecutando una tarea determinada. A continuación se expone como ejemplo el encendido del led verde:

```
if (msg->Text == "/verde-on" )
{
  client.postMessage(msg->ChatId, String("Encendiendo led verde"));
  SendMessage(VERDE * PRENDER);
  verde = 1;
 }
```

La función SendMessage() envía un mensaje desde una tarea que opera en un determinado núcleo hacia otra tarea que puede o no operar en un núcleo diferente. Para ello, se emplea una cola que opera como interface entre ambas tareas. En este caso la tarea a realizar es la de prender o apagar un led (ya que el manejo del hardware se realiza en el núcleo 0 mientras que el manejo de Telegram y la funcionalidad concerniente a WiFi se procesa en el núcleo 1.

Otras tareas a realizar en onReceive()

```
// Control de status acerca de los leds
 if (msg->Text == "/status" )  {
  if (!rojo && !verde && !azul)
   client.postMessage(msg->ChatId, String("Led rojo apagado\nLed verde apagado\nLed azul apagado"));
  else if (!rojo && !verde && azul)
   client.postMessage(msg->ChatId, String("Led rojo apagado\nLed verde apagado\nLed azul encendido\n"));
  else if (!rojo && verde && !azul)
   client.postMessage(msg->ChatId, String("Led rojo apagado\nLed verde encendido\nLed azul apagado\n"));
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
  else if (!rojo && verde && azul)
   client.postMessage(msg->ChatId, String("Led rojo apagado\nLed verde encendido\nLed azul encendido\n"));
  else if (rojo && !verde && !azul)
   client.postMessage(msg->ChatId, String("Led rojo encendido\nLed verde apagado\nLed azul apagado\n"));
  else if (rojo && !verde && azul)
   client.postMessage(msg->ChatId, String("Led rojo encendido\nLed verde apagado\nLed azul encendido\n"));
  else if (rojo && verde && !azul)
   client.postMessage(msg->ChatId, String("Led rojo encendido\nLed verde encendido\nLed azul apagado\n"));
  else
   client.postMessage(msg->ChatId, String("Led rojo encendido\nLed verde encendido\nLed azul encendido\n"));
 }
//Envío de un mensaje al bot en donde indica la //temperatura interna del ESP32 expresada en °C y °F ante //la petición /tempC o /tempF

if (msg->Text == "/tempc" ) client.postMessage(msg->ChatId, String((temprature_sens_read() - 32) / 1.8) + String(" °C"));

 if (msg->Text == "/tempf" ) client.postMessage(msg->ChatId, String((temprature_sens_read()) + String(" °F")));
```

5.7 Declaración de función invocada en el caso de que sucedan errores

```
Void onError (TelegramProcessError tbcErr, JwcProcessError jwcErr)
```

5.8 Empleo de las Funciones SendMessage() y ReceiveMessage()

5.8.1 SendMessage()

```
  void SendMessage( int message ) {

  //Envío al monitor serie para ver en que núcleo se está //ejecutando la función

  Serial.println("SendMessage(" + String(message) + ") En núcleo -> " + String(xPortGetCoreID()));

  if ( xqueue != 0 )
    {
  //Envía una tarea a la cola xqueue (ver 5.3), en este caso //corresponde al encendido o apagado de los leds.

  xQueueSend( xqueue, ( void * ) &message, ( TickType_t ) 0 );  }
  }
```

5.8.2 ReceiveMessage()

```
int ReceiveMessage()
{
 int msg = 0;
 if ( xqueue != 0 )
 {
//Receive a message on the created queue.  Block for 10 //ticks if a message is not immediately available.
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
  if ( xQueueReceive( xqueue, &( msg ), ( TickType_t )
10 ) )
   {
   }
 }
 Serial.println("ReceiveMessage(" + String(msg) + ") En
núcleo -> " + String(xPortGetCoreID()));
 return msg;
}
```

5.9 Declaración del setup() y del setup2()

En el setup() se configurarán los valores iniciales correspondientes al núcleo 1, estas tareas corresponderán al manejo del WiFi del dispositivo ESP32, y al llamado al evento onReceive() que es el encargado de procesar las peticiones del bot de Telegram. También acá se indican las tareas a desarrollarse en los distintos núcleos del dispositivo.

5.9.1 setup()

```
   void setup() {

   //Arranco el puerto serial ante todo, para que al
//imprimir ya esté activo

   Serial.begin(115200);

   Serial.println("setup En núcleo -> " +
String(xPortGetCoreID()));
   setupWiFi(); // Enciendo el WIFI.
   //Creo Cola de Mensajes para la comunicación entre
   //procesos con 100 lugares

   xqueue = xQueueCreate(100, sizeof( int ));

    if (xqueue == NULL) {
     Serial.println("Error creating the queue");
     exit(-1);
    }

   //Implementación del método begin del objeto client,
para //las denominadas “callback functions” a las que el
cliente //llamará ante la recepción de datos o de un error.

   client.begin(onReceive, onError);

   //Declaración de la tarea a realizarse en el núcleo 0

   //Task2 - Loop en donde se ejecutará la tarea.
//"loopTask2" - Descripción de la tarea, se ejecutará en
un //nuevo loop infinito dado precisamente por for (;;)
   //4096 - Tamaño del stack
   //NULL - Parámetro a pasarle a la tarea
   //1 - Prioridad
   //&TaskHandle2 - Definida como variable global del
tipo //TaskHandle-t, ver 5.3)
   //0 - Núcleo en el que se ejecutará la tarea

   xTaskCreateUniversal(Task2, "loopTask2", 4096,
NULL, 1, &TaskHandle2, 0);
   }
```

5.9.2 setup2()

```
void setup2() {
Serial.println("setup2 En núcleo -> " +
String(xPortGetCoreID()));
 setupLEDs();
 setupPULL();
 }
```

Las funciones setupLEDs() y setupPULL() aparecen declaradas previamente en el archivo fuente. Expondremos sólo la función setupPULL() por tratarse del manejo de una interrupción.

```
void setupPULL()
{
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
Serial.println("setupPULL En núcleo -> " + String(xPortGetCoreID()));
pinMode(pulsador, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(pulsador), handleInterruptRising, RISING);
}
```

6. Funciones loop() y Task2()

6.1 loop()

```
//Loop Principal de la Arduino. Se encargará del manejo
//de Wifi y Telegram.
void loop()
{
Serial.println("loop En núcleo -> " + String(xPortGetCoreID()));
delay(300);
//Pregunto si se disparó un evento, en este caso si se
//oprimió un pulsador.
if (pirTriggered) {
Serial.println("Se presionó el pulsador En núcleo -> " + String(xPortGetCoreID()));
// Le paso el ChatId del usuario que interactúa con el bot
client.postMessage(chatId, msg);
pirTriggered = false;
//pirTriggered en true significa que se oprimió el pulsador
  }
// Para procesar los datos recibidos, este método deberá ser //invocado en forma continua dentro del main loop.
  client.loop();
}
```

6.2 Task2()

```
//Se encarga del control del Hardware sobre el Núcleo 0,
//esto es sobre los actuadores, que en este caso de
//simulación son los leds, y el control del pulsador.
void Task2(void *pvParameters)
{
  setup2(); // NOTAR que se ejecuta solo una vez
//A continuación forzamos el loop infinito con un for
for (;;) {
Serial.println("loop En núcleo -> " + String(xPortGetCoreID()));
    delay(300);

    int msj = 0;
    msj = ReceiveMessage();
switch ( msj )
    {
      case (ROJO):
        digitalWrite(led_rojo, HIGH);
Serial.println("Pender Rojo en Núcleo -> " + String(xPortGetCoreID()));
        break;
      case (ROJO*APAGAR):
        digitalWrite(led_rojo, LOW);
Serial.println("Apagar Rojo en Núcleo -> " + String(xPortGetCoreID()));
        break;
      case (VERDE):
        digitalWrite(led_verde, HIGH);
Serial.println("Prender Verde en Núcleo -> " + String(xPortGetCoreID()));
        break;
      case (VERDE*APAGAR):
        digitalWrite(led_verde, LOW);
```

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

```
Serial.println("Apagar Verde en Núcleo -> " + String(xPortGetCoreID()));
      break;
    case (AZUL):
      digitalWrite(led_azul, HIGH);
Serial.println("Prender Azul en Núcleo -> " + String(xPortGetCoreID()));
      break;
    case (AZUL*APAGAR):
      digitalWrite(led_azul, LOW);
Serial.println("Apagar Verde en Núcleo -> " + String(xPortGetCoreID()));
      break;
    default:
      msj = 0;
   };
  }
}
```

FIGURA 11 – Estado inicial al invocar al bot desde Telegram

FIGURA 12– Estado del bot al apretar /start

7. Resultados obtenidos

7.1 Capturas del bot

Se indica a continuación en las figuras de la 11 a la 18, los resultados obtenidos desde el bot para el control del hardware. Este bot emplea comandos anteponiendo el símbolo de "/", como ya se hizo mención anteriormente.

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

FIGURA 13 – Comandos disponibles al apretar **/**

FIGURA 15 - Indicación de la temperatura interna del ESP32 en °C y en ° F

FIGURA 14– Respuesta del bot al encender y apagar los leds

FIGURA 16 – Estado del bot al apretar **/help**

FIGURA17 - Encendiendo todos los leds y verificando el status de los 3 leds

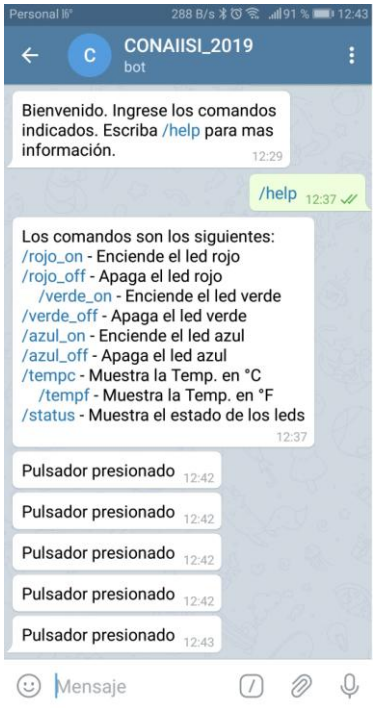

FIGURA 18 - Envío de alarma de Pulsador

7.2 Capturas Monitor serie

7.2.1 Configuración setup() y setup2()

14:02:25.584 -> setup En núcleo -> 1

14:02:25.584 -> SetupWiFi En núcleo -> 1

14:02:25.584 -> Intentando conectar a la red IPLAN-306232

14:02:26.193 -> ..

14:02:26.709 -> OK

14:02:26.709 -> IP address: 192.168.1.11

14:02:26.709 -> Strength: -44 dbm

14:02:26.709 ->

14:02:26.709 -> loop En núcleo -> 1

14:02:26.709 -> setup2 En núcleo -> 0

14:02:26.709 -> setupLEDs En núcleo -> 0

14:02:26.709 -> setupPULL En núcleo -> 0

7.2.2 Capturas de comandos del bot con /

a) /rojo_on

14:02:31.475 -> onReceive En núcleo -> 1

14:02:33.678 -> SendMessage(23) En núcleo -> 1

14:02:33.678 -> loop En núcleo -> 1

14:02:33.725 -> ReceiveMessage(23) En núcleo -> 0

14:02:33.725 -> Prender Rojo en núcleo -> 0

14:02:34.287 ->

{"ok":true,"result":{"message_id":272,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566955,"text":"Encendiendo led rojo"}}ReceiveMessage(0) En núcleo -> 0

b) /verde_on

14:02:36.631 -> onReceive En núcleo -> 1

14:02:38.694 -> SendMessage(22) En núcleo -> 1

14:02:38.694 -> loop En núcleo -> 1

14:02:38.787 -> ReceiveMessage(22) En núcleo -> 0

14:02:38.787 -> Prender Verde en núcleo -> 0

14:02:39.303 ->

{"ok":true,"result":{"message_id":275,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566960,"text":"Encendiendo led verde"}}ReceiveMessage(0) En núcleo -> 0

c) /azul_on

14:02:41.881 -> onReceive En núcleo -> 1

14:02:43.944 -> SendMessage(21) En núcleo -> 1

14:02:43.944 -> loop En núcleo -> 1

14:02:43.944 -> ReceiveMessage(21) En núcleo -> 0

14:02:43.944 -> Prender Azul en núcleo -> 0

14:02:44.553 ->

{"ok":true,"result":{"message_id":276,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566965,"text":"***Encendiendo led azul***"}}ReceiveMessage(0) En núcleo -> 0

d) /temp_C

14:02:46.897 -> onReceive En núcleo -> 1

14:02:49.475 -> loop En núcleo -> 0

14:02:49.522 ->

{"ok":true,"result":{"message_id":278,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566970,"text":"***53.33 u00b0C***"}}ReceiveMessage(0) En núcleo -> 0

e) /temp_F

14:02:53.694 -> onReceive En núcleo -> 1

14:02:56.366 -> loop En núcleo -> 0

14:02:56.413 ->

{"ok":true,"result":{"message_id":280,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566977,"text":"***128 u00b0F***"}}ReceiveMessage(0) En núcleo -> 0

f) /status

14:03:01.185 -> onReceive En núcleo -> 1

14:03:03.857 -> loop En núcleo -> 0

14:03:03.904 ->

{"ok":true,"result":{"message_id":282,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568566984,"text":"**Led rojo encendido\nLed verde encendido\nLed azul encendido**"}}ReceiveMessage(0) En núcleo -> 0

g) /help

14:26:51.812 -> onReceive En núcleo -> 1

ISSN: 2525-1333. Vol.: 5 - Nro. 1 (AGOSTO-2020)

14:26:54.466 -> loop En núcleo -> 0

14:26:54.513 ->

{"ok":true,"result":{"message_id":284,"from":{"id":859432320,"is_bot":true,"first_name":"CONAIISI_2019","username":"c2ing59_bot"},"chat":{"id":464201191,"first_name":"Carlos","username":"Pqj23","type":"private"},"date":1568568415,"text":"Los comandos son los siguientes:\n/rojo_on - Enciende el led rojo\n/rojo_off - Apaga el led rojo\n /verde_on - Enciende el led verde\n/verde_off - Apaga el led verde\n/azul_on - Enciende el led azul\n/azul_off - Apaga el led azul\n/tempc - Muestra la Temp. en u00b0C\n /tempf - Muestra la Temp. en u00b0F\n/status - Muestra el estado de los leds","entities":[{"offset":33,"length":8,"type":"bot_command"},{"offset":65,"length":9,"type":"bot_command"},{"offset":99,"length":9,"type":"bot_command"},{"offset":133,"length":10,"type":"bot_command"},{"offset":165,"length":8,"type":"bot_command"},{"offset":197,"length":9,"type":"bot_command"},{"offset":227,"length":6,"type":"bot_command"},{"offset":263,"length":6,"type":"bot_command"},{"offset":295,"length":7,"type":"bot_command"}]}}ReceiveMessage(0) En núcleo -> 0

(2.203 + 2.063 + 2.063)/3 = 2.11 S

El valor promedio que se tarda en procesar la cola, es decir desde que se envía una tarea con SendMessage() desde el núcleo 1 y hasta que esa misma tarea se recibe con ReceiveMessage() en el núcleo 0 es:

(47 + 93 + 0)/3 = 46.67 ms

Finalmente el valor promedio ponderado desde que se envía la leyenda de led prendido o apagado en el monitor serie del ESP32 y hasta que llega ese mensaje al bot a través de la API de Telegram es:

(564 + 516 + 609)/3 = 563 ms

8. Conclusiones y trabajo futuro

1. Observando las marcas de tiempo obtenidas en las capturas del monitor serie podemos establecer los siguientes valores:

El valor promedio desde que se ingresa al evento onReceive() y hasta que se envía la tarea desde el núcleo 1 con SendMessage() de aprender o apagar un led es:

2. La mensajería constituye un excelente método de control a través de Internet sin la necesidad de caer en un esquema de un servicio NO-IP como lo explicado en la introducción.

3. Cada vez es mucho más el interés por el empleo de bots y se desarrollan constantemente nuevos frameworks a fin de facilitar la escritura del código.

4. Gracias al sistema de mensajería instantánea de Telegram se da una excelente solución al tema de seguridad ya que se brinda un mecanismo de cifrado de canal extremo a extremo y se resuelve de manera simultánea la autenticación con intercambio de credenciales entre el bot y los distintos usuarios que componen Telegram, como así también entre los distintos usuarios y Telegram.

5. En un futuro trabajo se prevé la utilización de bots pero para que se comuniquen directamente entre dispositivos sin intervención humana (Sistemas machine to machine M2M).

6. También se prevé en un futuro trabajo el empleo de una Raspberry Pi para poder manejar más núcleos y así poder ejecutar mayor cantidad de tareas, aumentando de manera notable la performance del sistema.

9. Referencias y Bibliografía

[1] https://www.freertos.org/a00116.html

[2] https://www.freertos.org/a00117.html

[3] https://www.freertos.org/a00118.html

[4] https://telegram.org/

[5] Designing bots. 1st Edition. Amir Shevat. Publicación: mayo del 2017. Editorial OREILLY. ISBN-13: 978-1491974827

[6] https://core.telegram.org/bots#6-botfather

[7] https://core.telegram.org/bots/api

[8] Internet of Things Projects with ESP32. Agus Kurniawan. Publicación: 30 de marzo de 2019. Editorial Packt. ISBN-13: 978-1789956870

[9] https://core.telegram.org/mtproto

[10] INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY. (Paper). CHAITYA B. SHAH, DRASHTI R. PANCHAL. Volume 2, Issue X, Oct 2014. ISSN 2320-6802.

Recibido: 2020-07-30
Aprobado: 2020-08-07
Hipervínculo Permanente: http://www.reddi.unlam.edu.ar
Datos de edición: Vol. 5-Nro. 1-Art. 5
Fecha de edición: 2020-08-15