NEON
Network of Competence on Internet of Things
Co-funded by the Erasmus+ Programme of the European Union

Comunicación Cliente - Servidor

Comunicación Cliente - Servidor

Cliente

- Conexión WiFi en la misma red del servidor
- Recibe y envía mensajes MQTT

Servidor

- Servidor MQTT (Mosquitto)
- Interfaz de control (NodeRED)

WiFi - Modos de Operación

- Estación (STA_IF)
  - Se conecta a redes existentes
- AP (AP_IF)
  - Permite que otros dispositivos se conecten a ella

WiFi - Consulta de redes disponibles

```
import network

wlan = network.WLAN(network.STA_IF)
if not wlan.active():
   wlan.active(True)

redes = wlan.scan()
for red in redes:
   …
```

WiFi - Consulta de redes disponibles

```
import network

wlan = network.WLAN(network.STA_IF)
if not wlan.active():
    wlan.active(True)

redes = wlan.scan()
for red in redes:
    …
```

WiFi - Consulta de redes disponibles

```
import network

wlan = network.WLAN(network.STA_IF)
if not wlan.active():
    wlan.active(True)

redes = wlan.scan()
for red in redes:
    …
```

0. SSID
1. BSSID
2. Canal
3. RSSI
4. Modo de autenticación
5. Si está oculta

WiFi - Conexión a una red conocida

```
...

if not wlan.isconnected():
    wlan.connect("ssid", "contraseña")
    print("Conectando...")
    while not wlan.isconnected():
        sleep_ms(1000)

config = wlan.ifconfig()
print(f"Conectado con ip {config[0]}")
```

WiFi - Conexión a una red conocida

```
...

if not wlan.isconnected():
    wlan.connect("ssid", "contraseña")
    print("Conectando...")
    while not wlan.isconnected():
        sleep_ms(1000)

config = wlan.ifconfig()
print(f"Conectado con ip {config[0]}")
```

WiFi - Conexión a una red conocida

```
...

if not wlan.isconnected():
    wlan.connect("ssid", "contraseña")
    print("Conectando...")
    while not wlan.isconnected():
        sleep_ms(1000)

config = wlan.ifconfig()
print(f"Conectado con ip {config[0]}")
```

WiFi - Conexión a una red conocida

```
...

if not wlan.isconnected():
    wlan.connect("ssid", "contraseña")
    print("Conectando...")
    while not wlan.isconnected():
        sleep_ms(1000)

config = wlan.ifconfig()
print(f"Conectado con ip {config[0]}")
```

Ejemplos WiFi (bajo nivel)

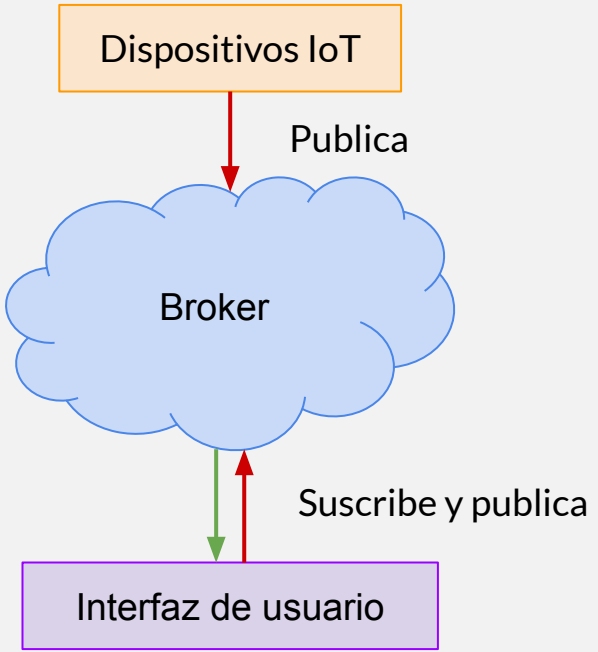
https://github.com/micropython/micropython/tree/master/examples/network

MQTT

- Protocolo de mensajería estandarizado
- Distribuye la información a través de publicación y suscripción
- Es eficiente para situaciones que se transporta poca información ya que consume poco ancho de banda

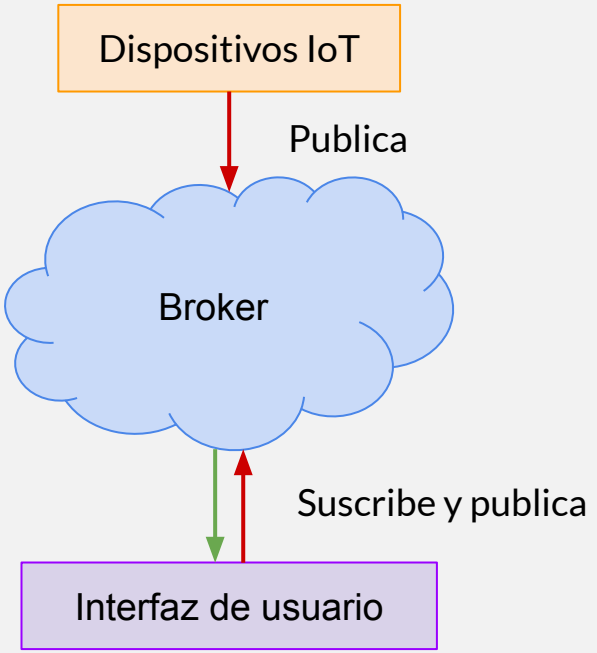
MQTT - Funcionamiento

- La conexión entre el subscriber y el publisher se encuentran desacopladas.
- La comunicación es manejada por un tercer componente llamado broker (servidor)



MQTT - Funcionamiento

- Un cliente puede suscribirse o publicar en diferentes topics en simultáneo
- Una red MQTT responde a una topología estrella



MQTT - Formato de Topics Sugerido

- Único dispositivo: **<parámetro>**
- Múltiples dispositivos: **<dispositivo>/<parámetro>**
- Ejemplos:
  - humedad
  - cocina/temperatura

Mensajes retenidos (retained)

- Bandera que permite que un mensaje sea persistente
- Al suscribirse al topic, se recibe este mensaje
- Solo persiste el último en cada topic

QoS (Quality of Service)

- Nivel 0: no ofrece garantías
- Nivel 1: Se asegura que el paquete llegue a destino al menos una vez
- Nivel 2: Se asegura que el paquete llegue a destino como máximo una vez

Last Will

- Mensaje enviado al perder la conexión con el broker
- Se puede configurar un topic y un mensaje
- Notifica a todos los clientes escuchando el topic que se perdió la conexión de forma no esperada

Cliente MQTT

Se sugiere el uso de un cliente MQTT con soporte de uasyncio

https://github.com/peterhinch/micropython-mqtt

- No es bloqueante
- Recupera la conexión Wi-Fi ante pérdidas
- Recupera la conexión del broker

Instalación de módulos externos

Por el momento, MicroPython carece de un repositorio oficial donde encontrar módulos hechos por usuarios

Existe una colección no oficial en https://awesome-micropython.com/

Por lo general, cada módulo lleva a un repositorio git con las instrucciones de instalación y ejemplos de uso

Instalación de módulos externos

Para mqtt_as:

Instalación de módulos externos

peterhinch mqtt_as/README: Link to Justin's TLS repo. 94b97f5 3 weeks ago History

..

README.md	mqtt_as/README: Link to Justin's TLS repo.	3 weeks ago
clean.py	Rename config file, improve README.	4 months ago
lptest_min.py	Rename config file, improve README.	4 months ago
main.py	1st commit of resilient async driver.	6 years ago
mqtt_as.py	mqtt_as: Improve connect fail error report.	2 months ago
mqtt_as_timeout.py	Support new uasyncio	3 years ago
mqtt_local.py	Tested with ESP32-S3	3 months ago
pubtest	Retained msg flag. Add range_ex demo. Doc changes.	4 years ago
range.py	Merge event interface.	4 months ago
range_ex.py	Merge event interface.	4 months ago
tls.py	Improve TLS docs and code comments.	2 months ago
tls32.py	Improve TLS docs and code comments.	2 months ago
tls8266.py	Rename config file, improve README.	4 months ago
unclean.py	Rename config file, improve README.	4 months ago

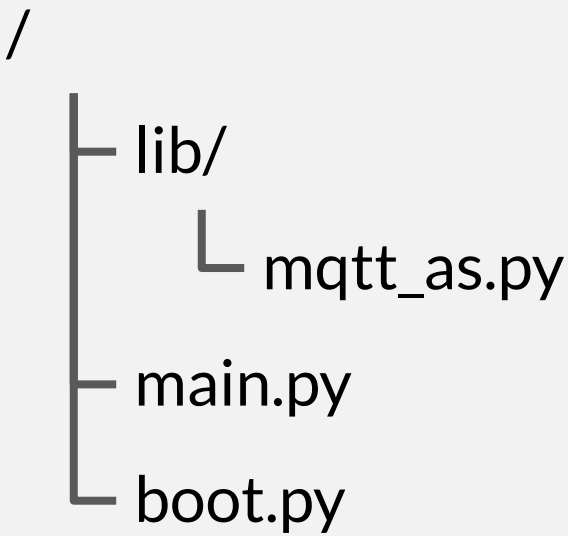
Instalación de módulos externos

785 lines (704 sloc) | 28 KB — Raw | Blame

```
# mqtt_as.py Asynchronous version of umqtt.robust
# (C) Copyright Peter Hinch 2017-2022.
# Released under the MIT licence.

# Pyboard D support added also RP2/default
# Various improvements contributed by Kevin Köck.

import gc
import usocket as socket
import ustruct as struct

gc.collect()
from ubinascii import hexlify
import uasyncio as asyncio

gc.collect()
from utime import ticks_ms, ticks_diff
from uerrno import EINPROGRESS, ETIMEDOUT

gc.collect()
from micropython import const
from machine import unique_id
```

Guardar Link como...

Estructura de archivos



Ejemplo mqtt_as

```
from mqtt_as import MQTTClient, config
import uasyncio

config["ssid"] = "Hands On IoT"
config["wifi_pw"] = "handsoniot"
config["server"] = "10.2.13.97"
config["port"] = 1884
```

Fuente:
https://github.com/peterhinch/micropython-mqtt/blob/master/mqtt_as/README.md#24-usage-with-callbacks

Ejemplo mqtt_as

```
def callback(topic, payload, retained):
    print((topic, payload, retained))

async def conn_handler(cliente):
    await cliente.subscribe("prueba/#", qos=0)
```

Fuente:
https://github.com/peterhinch/micropython-mqtt/blob/master/mqtt_as/README.md#24-usage-with-callbacks

Ejemplo mqtt_as

```
def callback(topic, payload, retained):
    print((topic, payload, retained))

async def conn_handler(cliente):
    await cliente.subscribe("prueba/#", qos=0)
```

Fuente:
https://github.com/peterhinch/micropython-mqtt/blob/master/mqtt_as/README.md#24-usage-with-callbacks

Ejemplo mqtt_as

```
config["subs_cb"] = callback
config["connect_coro"] = conn_handler

async def main(cliente):
    await cliente.connect()
    while True:
        await uasyncio.sleep(5)
        await cliente.publish("prueba/esp32",
                              "Mensaje de prueba", retain=False, qos=0)
```

Fuente:
https://github.com/peterhinch/micropython-mqtt/blob/master/mqtt_as/README.md#24-usage-with-callbacks

Ejemplo mqtt_as

```
config["subs_cb"] = callback
config["connect_coro"] = conn_handler

async def main(cliente):
    await cliente.connect()
    while True:
        await uasyncio.sleep(5)
        await cliente.publish("prueba/esp32",
                              "Mensaje de prueba", retain=False, qos=0)
```

Fuente:
https://github.com/peterhinch/micropython-mqtt/blob/master/mqtt_as/README.md#24-usage-with-callbacks

Responder comandos desde el callback

Las variables *topic* y *payload* son de tipo *bytes*

Para manipularlas como *strings* se pueden convertir usando el método .decode()

Luego se puede analizar la jerarquía con el método .split()

Responder comandos desde el callback

Los mensajes publicables deben ser *iterables*, es decir, ser una lista, cadena de caracteres u otro tipo que pueda contener más de un elemento

Para enviar números, primero puede convertirse usando la función *str(x)* que convierte a su representación en cadena de caracteres.

Responder comandos desde el callback

```
def callback(topic, payload, retained):
    topic_str = topic.decode()
    payload_str = payload.decode()
    jerarquia = topic_str.split("/")
    # [0]/[1]/[2]...

    # jerarquia[0] es prueba
    if jerarquia[1] == "habitacion":
        # verificar si el topic es prueba/habitacion
```

Responder comandos desde el callback

```
def callback(topic, payload, retained):
    topic_str = topic.decode()
    payload_str = payload.decode()
    jerarquia = topic_str.split("/")
    # [0]/[1]/[2]...

    # jerarquia[0] es prueba
    if jerarquia[1] == "habitacion":
        # verificar si el topic es prueba/habitacion
```

Responder comandos desde el callback

```
def callback(topic, payload, retained):
    topic_str = topic.decode()
    payload_str = payload.decode()
    jerarquia = topic_str.split("/")
    # [0]/[1]/[2]...

    # jerarquia[0] es prueba
    if jerarquia[1] == "habitacion":
        # verificar si el topic es prueba/habitacion
```

Responder comandos desde el callback

Advertencia:

Los mensajes recibidos no necesariamente van a cumplir con la estructura esperada.

Se debe verificar que tanto los *topics* como los *payloads* tengan el formato adecuado

Ejemplos:

Verificación de rangos:

- El dispositivo debe responder dentro de un rango adecuado

Responder comandos desde el callback

Ejemplos:

Verificación de tipos:

- Los *payloads* llegan en formato byte y son convertidos en *strings*
- El dato recibido debe tener formato numérico:
  - Interpretar los bytes como números en vez de ASCII
  - Convertir a entero o flotante

Responder comandos desde el callback

Conversión de tipos

```
>>> n = "123"
>>> int(n)
123
>>> float(n)
123.0
```

Responder comandos desde el callback

Conversión de tipos

```
>>> n = "123a"
>>> int(n)
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
'123a'
```

Responder comandos desde el callback

Conversión de tipos

```
def callback(topic, payload, retained):
    topic_str = topic.decode()
    payload_str = payload.decode()
    …
    if payload_str.isdigit():
        # es número entero
    else:
        # manejar error
        return
```

Responder comandos desde el callback

Conversión de tipos

```
def callback(topic, payload, retained):
    topic_str = topic.decode()
    payload_str = payload.decode()
    …
    try:
        payload_int = int(payload_str)
    except ValueError:
        print(f"Valor {payload_str} no es un número")
        return
```

MQTT - Ejemplo

Escribir un programa que se conecte a un servidor MQTT y emita un mensaje cada un segundo en un topic arbitrario dentro de la jerarquía *salida/<nombre>* y muestre los mensajes que escucha desde el topic *entrada* a través de puerto serie.

Usar MQTT-Explorer para enviar y recibir mensajes.

MQTT - Ejercicio

Reutilizando el programa anterior, agregar el manejo de los siguientes mensajes:

- Suscripciones
  - **<nombre>/salidas/led_<color>**: encender el LED del color correspondiente si se recibe *on*, sino apagar
  - **<nombre>/salidas/neopixel/<num>**: se reciben tres valores separados por comas (R,G,B) y encender el NeoPixel como corresponda. Deben interpretarse qué píxel iluminar siendo el topic <num> el índice.

MQTT - Ejercicio

Reutilizando el programa anterior, agregar el manejo de los siguientes mensajes:

- Publicaciones
  - **<nombre>/entradas/nivel**: al pulsar uno de los botones, enviar la posición del potenciómetro
  - **<nombre>/entradas/alarma**: al pulsar el otro botón, enviar un mensaje de alarma
  - **<nombre>/entradas/temperatura**: periódicamente enviar la temperatura actual
  - **<nombre>/entradas/humedad**: periódicamente enviar la humedad actual