```
import RPi.GPIO as GPIO

import threading

import time

import paho.mqtt.client as mqtt


# Pin setup

red = 17

green = 27

blue = 22

button = 18

red2 = 9


# GPIO setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(red, GPIO.OUT)

GPIO.setup(green, GPIO.OUT)

GPIO.setup(blue, GPIO.OUT)

GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(red2, GPIO.OUT)


# PWM setup for Green LED

green_pwm = GPIO.PWM(green, 500)  # 500 Hz PWM frequency

green_pwm.start(0)


# LED color states

color_states = [
    (1, 0, 0),  # Red
    (0, 1, 0),  # Green
    (0, 0, 1),  # Blue
    (1, 1, 0),  # Red + Green
    (1, 0, 1),  # Red + Blue
    (0, 1, 1),  # Green + Blue
    (1, 1, 1)   # All on (White)
]

# MQTT Settings
MQTT_BROKER = "iot.kmitlnext.com"  # Replace with your broker address
MQTT_PORT = 9001                   # Port number
MQTT_TOPIC = "led/control"         # Topic for LED control
MQTT_USERNAME = "kmitliot"         # Username (if required)
MQTT_PASSWORD = "KMITL@iot1234"    # Password (if required)

# Global variable for button state
button_pressed = False

# Function to set RGB LED colors
def set_rgb_color(r, g, b):
    GPIO.output(red, r)
    GPIO.output(green, g)
    GPIO.output(blue, b)
    print(f"RGB LEDs set to: Red={r}, Green={g}, Blue={b}")
```

```
# Main task: RGB LED changes color every second
def main_task():
    print("Main task (RGB LED color cycle) started.")
    while True:
        for state in color_states:
            set_rgb_color(*state)
            time.sleep(1)

# Sub-thread: Green LED dimming
def dimming_task():
    print("Dimming task (Green LED) started.")
    while True:
        for duty in range(10, 101, 10):  # Increase brightness
            green_pwm.ChangeDutyCycle(duty)
            time.sleep(2)
        for duty in range(100, 9, -10):  # Decrease brightness
            green_pwm.ChangeDutyCycle(duty)
            time.sleep(2)

# Event task: Toggle Red2 LED on button press
def button_event(channel):
    global button_pressed
    button_pressed = not button_pressed
    print(f"Button {'pressed' if button_pressed else 'released'}")
    GPIO.output(red2, button_pressed)
```

```
# MQTT Callback: When a message is received
def on_message(client, userdata, msg):
    print(f"Message received: {msg.payload.decode()}")
    command = msg.payload.decode().lower()

    # Control RGB LEDs
    if command == "cyan":
        set_rgb_color(1, 0, 0)
    elif command == "yellow":
        set_rgb_color(0, 1, 0)
    elif command == "magenta":
        set_rgb_color(0, 0, 1)
    elif command == "green":
        set_rgb_color(1, 1, 0)
    elif command == "blue":
        set_rgb_color(1, 0, 1)
    elif command == "red":
        set_rgb_color(0, 1, 1)
    elif command == "off":
        set_rgb_color(1, 1, 1)
    elif command == "white":
        set_rgb_color(0, 0, 0)  # Turn off all LEDs

    # Control Red2 LED on/off
    if command == "on":
        GPIO.output(red2, 1)
    elif command == "off":
        GPIO.output(red2, 0)

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
        client.subscribe(MQTT_TOPIC)
        print(f"Subscribed to {MQTT_TOPIC} topic!")
    else:
        print(f"Failed to connect, return code {rc}")

# Add button press event detection
GPIO.add_event_detect(button, GPIO.RISING, callback=button_event, bouncetime=300)

def mqtt_loop():
    client = mqtt.Client(transport="websockets")
    client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
    client.on_connect = on_connect
    client.on_message = on_message

    client.connect(MQTT_BROKER, MQTT_PORT)
    client.loop_forever()

def main():
    try:
        # Create threads
        dimming_thread = threading.Thread(target=dimming_task)
        mqtt_thread = threading.Thread(target=mqtt_loop)

        # Start threads
        dimming_thread.start()
        mqtt_thread.start()

        # Join threads to the main thread
        dimming_thread.join()
        mqtt_thread.join()

    except KeyboardInterrupt:
        print("Exiting program")
    finally:
        GPIO.cleanup()

if __name__ == "__main__":
    main()
```