IoT Mandatory Assignment 02 - MQTT

This document containsthe answers to the tasks in the IKT520 MQTT Mandatory Assignment (MA-02), using the `paho-mqtt` library in Python. The full source code is available in the following GitHub repository:

https://github.com/yngvemag/ikt520-mqtt-assignment

---

1. Use the `paho-mqtt` library and create two clients (publisher and subscriber)

```
publisher = mqtt.Client(
    client_id=f"publisher-{uuid4().hex[:8]}",
    clean_session=True
)

subscriber = mqtt.Client(
    client_id=f"subscriber-{uuid4().hex[:8]}",
    clean_session=True
)
```

Explanation of Arguments

- **client_id**: A unique identifier for the client. If empty or None, the broker will assign one.
- **clean_session**: Boolean indicating whether the broker should remember previous session data (subscriptions, undelivered messages, etc.).
    - `True`: New session every time; state is not stored.
    - `False`: Persistent session.

Output Example:

```
Publisher client created with ID: publisher-87faeb60
Subscriber client created with ID: subscriber-b7dbd946
```

2. Connect the publishing client to a Broker

```
def on_connect(client: mqtt.Client, userdata: Any, flags: Dict[str, bool], rc:
int):
    if rc == 0:
        print(f"Connected to broker with result code {rc}")
        session_present = flags.get('session_present', False)
        print(f"Session present flag: {session_present}")
    else:
        print(f"Failed to connect: {rc}")
```

on_connect Parameters

- **client**: The client instance that triggered the callback.
- **userdata**: Optional user data set by `client.user_data_set()`.
- **flags**: Dictionary with connection flags returned by the broker.
  - `session_present`: Indicates whether the broker already had session data for this client ID.
- **rc**: Return code from the broker. Possible values:
  - `0`: Connection successful
  - `1`: Connection refused - incorrect protocol version
  - `2`: Connection refused - invalid client identifier
  - `3`: Connection refused - server unavailable
  - `4`: Connection refused - bad username or password
  - `5`: Connection refused - not authorized
  - `6-255`: Reserved for future use

Output Example:

```
Connected to broker with result code 0
Session present flag: False
```

3. Connect the subscribing client to the Broker

```
def on_subscribe(client: mqtt.Client, userdata: Any, mid: int, granted_qos:
List[int]):
    print(f"Subscribed with message ID {mid}, granted QoS: {granted_qos}")
```

- **topic**: `CyberSec/IKT520`
- **on_subscribe** is triggered upon receipt of a SUBACK packet.
- **mid**: Message ID associated with the subscription.
- **granted_qos**: List of QoS levels granted for each topic subscription request (can differ from requested QoS).

Output Example:

```
Subscribed with message ID 1, granted QoS: (1,)
```

4. Publish a message to the topic `CyberSec/IKT520`

```
info = publisher.publish(
    topic="CyberSec/IKT520",
    payload="Hello MQTT!",
    qos=1,
    retain=False
)
```

Explanation of Arguments

- **topic**: The topic to publish to.
- **payload**: The message content (string or bytes).
- **qos**: Quality of Service level (0, 1, or 2)
- **retain**: If True, the broker stores the message as the last known good value on that topic.

Output Example:

```
Received message: 'Hello MQTT World!' on topic 'CyberSec/IKT520'
Published message 'Hello MQTT World!' to topic 'CyberSec/IKT520'
Subscriber received 1 messages:
  - Hello MQTT World!
```

5. Make two subscriptions using wildcards

```
client.subscribe("Sensors/+/Temperature", qos=1)  # Single-level wildcard
client.subscribe("Sensors/#", qos=1)              # Multi-level wildcard
```

Wildcard Explanation

- `+`: Matches **exactly one** topic level.
- `#`: Matches **zero or more** topic levels.

Output Example:

```
Subscribed to: Sensors/+/Temperature
Subscribed to: Sensors/#
Published '22C' to 'Sensors/Living/Temperature'
SINGLE-WILDCARD received: '22C' on 'Sensors/Living/Temperature'
```

```
MULTI-WILDCARD received: '22C' on 'Sensors/Living/Temperature'
...
```

Summary

Single-level wildcard (+) subscription results:

- Sensors/Living/Temperature: 22C
- Sensors/Kitchen/Temperature: 25C
- Sensors/Garden/Temperature: 18C

Multi-level wildcard (#) subscription results:

- Sensors/Living/Temperature: 22C
- Sensors/Kitchen/Temperature: 25C
- Sensors/Living/Humidity: 60%
- Sensors/Garden/Temperature: 18C
- Sensors/Kitchen/Temperature/Indoor: 24C

6. Persistent Session (clean_session=False, QoS 1)

Output Example:

```
Persistent subscriber received: 'Temperature reading 1' on 'Sensor/Temp'
...
Persistent subscriber received: 'Temperature reading 20' on 'Sensor/Temp'

Received 20 messages after reconnection
```

Explanation

- clean_session=False creates a persistent session
- QoS 1 ensures messages are stored for offline clients
- Upon reconnection, the broker delivers stored messages to the same client_id

7. Non-Persistent Session (clean_session=True, QoS 1)

Output Example:

```
Received 0 messages after reconnection
```

Explanation

- clean_session=True deletes all session state upon disconnect
- QoS 1 guarantees delivery only while connected
- Messages published while offline are not stored or delivered

8. Mixed QoS with Persistent Session (QoS 0 subscribe, QoS 2 publish)

Output Example:

```
Received 20 messages after reconnection
```

Explanation

- Subscriber uses clean_session=False (persistent)
- But QoS 0 subscription does not allow message storage
- Despite QoS 2 publishing, subscription level QoS (0) determines behavior
- Messages are only delivered while client is connected
