Lab7: MQTT Message loop

Introduction

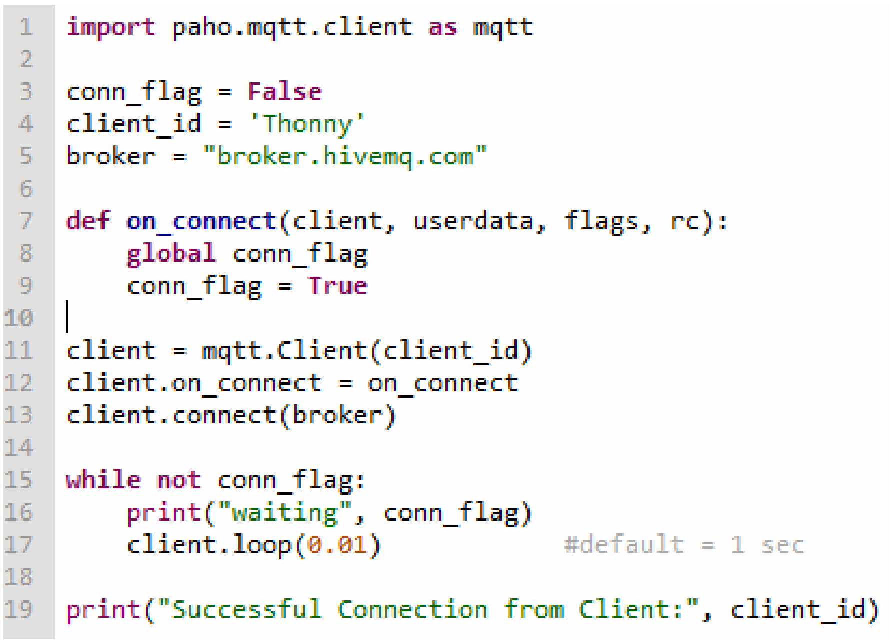
This lab introduces the use of a message loop in the Paho client to maintain a connection and respond to messages. The loop enables a client to run its own code whilst responding to (unpredictable) incoming messages.

Learning outcomes

1. Explore the different types of paho client message loops
2. Use of Callback functions
3. Build responsive MQTT applications

Task 1: Blocking loop

1. The following program defines a client called 'Thonny' which attempts to connect to the 'Hive' broker at broker.hivemq.com. It defines an *on_connect* callback method which sets a *conn_flag* variable to True once a successful connection is made. Whilst waiting for a connection, the message loop is executed repeatedly to allow the callback to run – the number of times it is called can be determined by the number of 'waiting' messages. Enter the code and run it:



```python
import paho.mqtt.client as mqtt

conn_flag = False
client_id = 'Thonny'
broker = "broker.hivemq.com"

def on_connect(client, userdata, flags, rc):
    global conn_flag
    conn_flag = True

client = mqtt.Client(client_id)
client.on_connect = on_connect
client.connect(broker)

while not conn_flag:
    print("waiting", conn_flag)
    client.loop(0.01)             #default = 1 sec

print("Successful Connection from Client:", client_id)
```

The client ID has been set to 'Thonny'
The loop runs every 0.01 s, so an estimate of connection time can be made:

Time to connect =

2. The exchange is very simple, and can be examined using Wireshark. Use Wireshark to determine the following MQTT parameters and write them down in the space below:

   Message Length =

   Protocol Name and Length =

   MQTT Version =

   QoS level =

   Keep Alive =

   Connect Flag = (clean: True or False?)

   Each of the 'bits' in the Connect Flags byte means something – Wireshark can list them, along with their values. List the 8 flag names below:

   ___________________ ______________________ ___________________ ______________

   ___________________ ______________________ ___________________ ______________

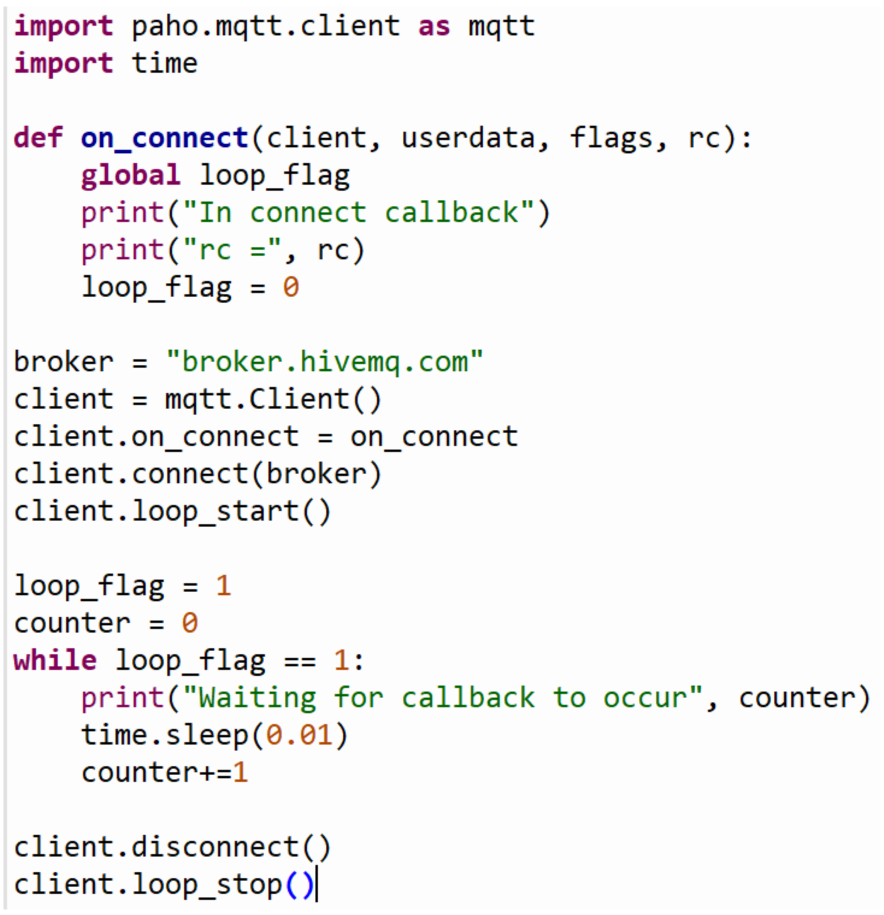
Task 2: Loop start/stop

1. The code overleaf was discussed in the lecture – it uses the Loop start/stop functions to run the message loop on another thread. It also uses a flag (loop_flag) to check the state of the connection. In Task 1 we blocked program execution to run the loop – in this task we start an additional thread which checks for the CONNACK message and fires the on_connect callback which resets the flag.

   The code finally disconnects and terminates (stops) the loop thread.

   Make sure that you appreciate (can explain) the difference between the two approaches to checking for the arrival of the CONNACK message.

   Code is overleaf . . .



```python
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
    global loop_flag
    print("In connect callback")
    print("rc =", rc)
    loop_flag = 0

broker = "broker.hivemq.com"
client = mqtt.Client()
client.on_connect = on_connect
client.connect(broker)
client.loop_start()

loop_flag = 1
counter = 0
while loop_flag == 1:
    print("Waiting for callback to occur", counter)
    time.sleep(0.01)
    counter+=1

client.disconnect()
client.loop_stop()
```

2. The counter will count the number of 0.01 s periods which pass before a connection is made. The time taken depends upon how quickly the connection to a remote broker can be made.

   The connection to a local broker should be much quicker. Change the broker to "localhost" and try to determine the time to connect.

   Time to connect =

   Note that you will need a running broker on your machine for this to work:

   Is there one?

   Did you start it or not?

   Explain what is happening

Task 3: Loop_Forever()

1. Go to https://github.com/eclipse/paho.mqtt.python and copy the code given under the 'Getting Started' heading (we discussed it in class). Run it in Thonny, and (briefly) note down what happens:

2. Most brokers regularly emit ‘diagnostic/information’ messages under the ‘$SYS’ topic. Since we subscribed to the topic `$SYS/#` we saw all the messages

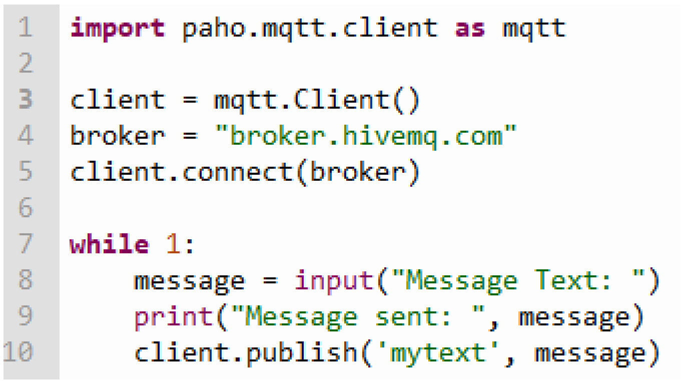
   Try running the local Mosquitto broker and subscribing to the $SYS topics. What is the publish period for these messages?

   $SYS publish period =

3. Change the topic to “CM125/*yourname*” and then use **either** the command line mosquito_pub application **OR** the MQTT-Explorer application to publish some messages that your application can receive and print out.

Task 4: MQTT Chat Application

1. The code below forms the start of a simple MQTT ‘chat’ application:



```python
import paho.mqtt.client as mqtt

client = mqtt.Client()
broker = "broker.hivemq.com"
client.connect(broker)

while 1:
    message = input("Message Text: ")
    print("Message sent: ", message)
    client.publish('mytext', message)
```

   Using your unique topic to publish messages on, use **either** the command line mosquito_sub application **OR** the MQTT-Explorer application to receive your messages.

2. Work with your neighbour to set up a system where you can use the above Chat application to send messages to clients running on each other’s PC (you will need to exchange publish topics to do this).

Challenge:

The application above can send messages (no loop is required because nothing else is happening). A full chat application would be able to send and receive messages without the need for an additional client.

Using pre-shared topics, design an application which can allow you and your neighbour to ‘chat’ using MQTT. You will need one each (slightly different topics).

Advice: _pub and _sub topics will need to be swapped on the second version of the app.
Use the Loop start/stop message loop to access the *on_message* callback
Remove the input ‘Message Text’ string – use `message = input()`

(it can be done in 18 lines of code)