```python
# -*- coding: utf-8 -*-

from paho.mqtt.client import Client

def on_message(mqttc, userdata, msg):
    print("MESSAGE:", userdata, msg.topic, msg.qos, msg.payload, msg.retain)
    mqttc.publish('clients/test', msg.payload)


def main(broker, topic):
    mqttc = Client()

    mqttc.on_message = on_message
    mqttc.connect(broker)

    mqttc.subscribe(topic)
    mqttc.loop_forever()


if __name__ == "__main__":
    import sys
    if len(sys.argv)<3:
        print(f"Usage: {sys.argv[0]} broker topic")
        sys.exit(1)
    broker = sys.argv[1]
    topic = sys.argv[2]
    main(broker, topic)
```

```
from paho.mqtt.client import Client
from multiprocessing import Process, Manager
from time import sleep
import random

NUMBERS = 'numbers'
CLIENTS = 'clients'
TIMER_STOP = f'{CLIENTS}/timerstop'
HUMIDITY = 'humidity'


def is_prime(n):
    i = 2
    while i*i < n and n % i != 0:
        i += 1
    return i*i > n



"""
Esto de aquÃ- no funciona. El parÃ¡metro mqtcc no funciona en
un proceso.
"""
# def timer(time, mqttc):
#     msg = f'timer working. timeout: {time}'
#     print(msg)
#     mqttc.publish(TIMER_STOP, msg)#, hostname=BROKER)
#     sleep(time)
#     msg = f'timer working. timeout: {time}'
#     mqttc.publish(TIMER_STOP, msg)#, hostname=BROKER)
#     print('timer end working')

# es necesario poner publish.single
# def timer(time, data):
#     msg = f'timer working. timeout: {time}'
#     print(msg)
#     publish.single(TIMER_STOP, msg, hostname=BROKER)
#     sleep(time)
#     msg = f'timer working. timeout: {time}'
#     publish.single(TIMER_STOP, msg, hostname=BROKER)
#     print('timer end working')


# o crear in cliente nuevo
def timer(time, data):
    mqttc = Client()
    mqttc.connect(data['broker'])
    msg = f'timer working. timeout: {time}'
    print(msg)
    mqttc.publish(TIMER_STOP, msg)
    sleep(time)
    msg = f'timer working. timeout: {time}'
    mqttc.publish(TIMER_STOP, msg)
    print('timer end working')
    mqttc.disconnect()


def on_message(mqttc, data, msg):
    print(f"MESSAGE:data:{data}, msg.topic:{msg.topic}, payload:{msg.payload}")
    try:
        #if is_prime(int(msg.payload)):
        if int(msg.payload) % 2 == 0:
            worker = Process(target=timer,
                             args=(random.random()*20, data))
            worker.start()
    except ValueError as e:
```

```
        print(e)
        pass


def on_log(mqttc, userdata, level, string):
    print("LOG", userdata, level, string)


def main(broker):
    data = {'client':None,
            'broker': broker}
    mqttc = Client(client_id="combine_numbers", userdata=data)
    data['client'] = mqttc
    mqttc.enable_logger()
    mqttc.on_message = on_message
    mqttc.on_log = on_log
    mqttc.connect(broker)
    mqttc.subscribe(NUMBERS)
    mqttc.loop_forever()


if __name__ == "__main__":
    import sys
    if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} broker")
        sys.exit(1)
    broker = sys.argv[1]
    main(broker)
```

```
# -*- coding: utf-8 -*-

from threading import Lock #No estamos usando multiprocessing
# en realidad aquÃ- no hace falta el lock, solo hay 1 hebra en
# ejecuciÃ³n.
from paho.mqtt.client import Client
from time import sleep

def on_message(mqttc, data, msg):
    print ('on_message', msg.topic, msg.payload)
    n = len('temperature/')
    lock = data['lock']
    lock.acquire()
    try:
        key = msg.topic[n:]
        if key in data:
            data['temp'][key].append(msg.payload)
        else:
            data['temp'][key]=[msg.payload]
    finally:
        lock.release()
    print ('on_message', data)


def main(broker):
    data  = {'lock':Lock(), 'temp':{}}
    mqttc = Client(userdata=data)
    mqttc.on_message = on_message
    mqttc.connect(broker)
    mqttc.subscribe('temperature/#')
    mqttc.loop_start()

    while True:
        sleep(8)
        for key,temp  in data['temp'].items():
            mean = sum(map(lambda x: int(x), temp))/len(temp)
            print(f'mean {key}: {mean}')
            data[key]=[]



if __name__ == "__main__":
    import sys
    if len(sys.argv)<2:
        print(f"Usage: {sys.argv[0]} broker")
        sys.exit(1)
    broker = sys.argv[1]
    main(broker)
```

```
# -*- coding: utf-8 -*-

from paho.mqtt.client import Client

TEMP = 'temperature'
HUMIDITY = 'humidity'

def on_message(mqttc, data, msg):
    print (f'message:{msg.topic}:{msg.payload}:{data}')
    if data['status'] == 0:
        temp = int(msg.payload) # we are only susbribed in temperature
        if temp>data['temp_threshold']:
            print(f'umbral superado {temp}, suscribiendo a humidity')
            mqttc.subscribe(HUMIDITY)
            data['status'] = 1
    elif data['status'] == 1:
        if msg.topic==HUMIDITY:
            humidity = int(msg.payload)
            if humidity>data['humidity_threshold']:
                print(f'umbral humedad {humidity} superado, cancelando suscripciÃ³n')
                mqttc.unsubscribe(HUMIDITY) # Esto debe ser lo Ãºltimo
                data['status'] = 0
        elif TEMP in msg.topic:
            temp = int(msg.payload)
            if temp<=data['temp_threshold']:
                print(f'temperatura {temp} por debajo de umbral, cancelando suscripciÃ³n')
                data['status']=0
                mqttc.unsubscribe(HUMIDITY)

def on_log(mqttc, data, level, buf):
    print(f'LOG: {data}:{msg}')


def main(broker):
    data = {'temp_threshold':20,
            'humidity_threshold':80,
            'status': 0}
    mqttc = Client(userdata=data)
    mqttc.on_message = on_message
    mqttc.enable_logger()

    mqttc.connect(broker)
    mqttc.subscribe(f'{TEMP}/t1')
    mqttc.loop_forever()


if __name__ == "__main__":
    import sys
    if len(sys.argv)<2:
        print(f"Usage: {sys.argv[0]} broker")
        sys.exit(1)
    broker = sys.argv[1]
    main(broker)
```

```
from paho.mqtt.client import Client
from multiprocessing import Process, Manager
from time import sleep
import paho.mqtt.publish as publish
import time

def on_message(mqttc, data, msg):
    print(f"MESSAGE:data:{data}, msg.topic:{msg.topic}, payload:{msg.payload}")


def on_log(mqttc, userdata, level, string):
    print("LOG", userdata, level, string)

def main(broker):
    data = {'status':0}
    mqttc = Client(userdata=data)
    mqttc.enable_logger()
    mqttc.on_message = on_message
    mqttc.on_log = on_log
    mqttc.connect(broker)

    res_topics = ['clients/a', 'clients/b']
    for t in res_topics:
        mqttc.subscribe(t)
    mqttc.loop_start()
    tests = [
        (res_topics[0], 4, 'uno'),
        (res_topics[1], 1, 'dos'),
        (res_topics[0], 2, 'tres'),
        (res_topics[1], 5, 'tres')
    ]
    topic = 'clients/timeout'
    for test in tests:
        mqttc.publish(topic, f'{test[0]},{test[1]},{test[2]}')
    time.sleep(10)

if __name__ == "__main__":
    import sys
    if len(sys.argv)<2:
        print(f"Usage: {sys.argv[0]} broker")
        sys.exit(1)
    broker = sys.argv[1]
    main(broker)
```

```
# -*- coding: utf-8 -*-

from paho.mqtt.client import Client
import paho.mqtt.publish as publish
from multiprocessing import Process
from time import sleep

def work_on_message(message, broker):
    print('process body', message)
    topic, timeout, text = message[2:-1].split(',')
    print('process body', timeout, topic, text)
    sleep(int(timeout))
    publish.single(topic, payload=text, hostname=broker)
    print('end process body',message)

def on_message(mqttc, userdata, msg):
    print('on_message', msg.topic, msg.payload)
    worker = Process(target=work_on_message, args=(str(msg.payload), userdata['b
roker']))
    worker.start()
    print('end on_message', msg.payload)

def on_log(mqttc, userdata, level, string):
    print("LOG", userdata, level, string)

def on_connect(mqttc, userdata, flags, rc):
    print("CONNECT:", userdata, flags, rc)

def main(broker):
    userdata = {
        'broker': broker
    }
    mqttc = Client(userdata=userdata)
    mqttc.enable_logger()
    mqttc.on_message = on_message
    mqttc.on_connect = on_connect
    mqttc.connect(broker)

    topic = 'clients/timeout'
    mqttc.subscribe(topic)

    mqttc.loop_forever()


if __name__ == "__main__":
    import sys
    if len(sys.argv)<2:
        print(f"Usage: {sys.argv[0]} broker")
        sys.exit(1)
    broker = sys.argv[1]
    main(broker)
```