MQTT – come usare MQTT in C – pt.1

Charles Xavier: *Due giorni in viaggio, un solo pasto e non abbiamo chiuso occhio. Lei ha 11 anni e io 90, cazzo!*
Logan: *Sono 101 ragioni per continuare a muoverci.*

MQTT, un titolo secco di una sola parola. E qui ci vuole un film con un titolo (e un contenuto) altrettanto secco: **Logan** del bravo **James Mangold** fa proprio al caso nostro. Un film *"crepuscolare"* che celebra la fine della Saga dei Mutanti, con un **Wolverine/H.Jackman** e un **Prof.X/P.Stewart** veramente perfetti. Un film super-raccomandato con frasi come quella sopra che ci introducono al tema di oggi: dopo tanta fatica fatta per studiare e assimilare gli *n-mila* protocolli di rete disponibili, è il caso di imparare a usarne un'altro? E la risposta è si: in questo articolo parleremo di **MQTT**!



…facciamo in fretta, se no ci perdiamo il prossimo MQTT!…
Ok, veniamo al dunque: cosa è, esattamente, MQTT (*Message Queuing Telemetry Transport*)? È un protocollo di messaggistica leggero, basato su standard aperti, di tipo *"pubblica/sottoscrivi"*, progettato per garantire una

comunicazione efficiente tra dispositivi in reti con larghezza di banda ridotta, elevata latenza o inaffidabili. È un protocollo relativamente recente: fu sviluppato nel 1999 per il *monitoring* di oleodotti via satellite, e in seguito, visto che funzionava bene, è diventato lo standard de facto per le applicazioni dell'Internet delle cose (IoT) e dell'IoT industriale (IIoT) grazie al suo impatto minimo sulle risorse e alla sua scalabilità.

Il protocollo lavora usando un'architettura basata su un *broker centrale* (una specie di *server)* in cui i dispositivi (*client publisher*) pubblicano messaggi su argomenti specifici anziché inviare dati direttamente ad altri dispositivi. Gli abbonati (*client subscriber*) registrano il proprio interesse per questi argomenti, consentendo al *broker* di filtrare e distribuire i messaggi solo ai destinatari pertinenti. Le caratteristiche principali includono tre livelli di *Quality of Service* (QoS) per garantire la consegna, messaggi conservati per gli aggiornamenti di stato e supporto per la crittografia e l'autenticazione TLS per garantire la sicurezza. La sua natura binaria e le intestazioni dei messaggi di piccole dimensioni (fino a due byte) lo rendono ideale per microcontrollori con risorse limitate e sensori sensibili al consumo energetico.

Si può anche notare che per pubblicare e sottoscrivere si usa una nomenclatura gerarchica molto simile a quella di un *file-system* POSIX, quindi con alberi di nomi separati dal carattere *"/"*, e questi alberi possono essere molto nidificati, esattamente (come appena detto) come in un *file-system*: un'idea molto interessante e azzeccata. Notare che nella terminologia MQTT questi *pseudo-pathname,* usati per pubblicare e sottoscrivere, vengono chiamati **topic.**

E non possiamo continuare senza mostrare un po' di codice, no? Ok, per scrivere il codice dobbiamo/possiamo appoggiarci a qualche libreria di quelle disponibili: per il nostro amato C abbiamo due scelte principali:

- **Eclipse Paho MQTT C Client library**: è una libreria abbastanza sofisticata e completa, compatibile con i due standard disponibili, MQTT 3.1.1 e MQTT 5.0. È ideale per applicazioni *"senza compromessi"*, su Hardware di buon livello (PC e Server).
- **MQTT-C**: è una libreria semplice e leggerissima, compatibile solo con lo standard MQTT 3.1.1. È ideale per applicazioni *embedded* (con OS o anche di tipo *"bare-metal"*) ma usabile anche su **Linux** o alcuni RTOS.

Per questo articolo ho scritto un semplice esempio che usa la **libpaho** (versione 3.1.1), ed è un codice che implementa un *client* che funziona, allo stesso tempo, come *Publisher* e come *Subscriber,* usando opportunamente la linea di comando: in questo modo si possono, ad esempio, lanciare due *client* che scrivono/leggono sui rispettivi topic *"incrociati"* (e cioè: il *client1* pubblica sul *topic2* e legge dal *topic1* e il *client2* pubblica sul *topic1* e legge dal *topic2)…* spero di essere stato chiaro. Comunque, vediamo il codice, compiliamo ed eseguiamo come appena descritto, e sarà tutto più chiaro. Vai col codice!

```
// mqttclient.c - un semplice client mqtt (con libpaho)
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <unistd.h>
#include "MQTTClient.h" #define QOS 1
#define TIMEOUT 10000L // onMessage - una callback per i messaggi che arrivano sul topic sorvegliato
int onMessage(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{ printf("messaggio ricevuto sul topic %s: \"%.*s\"\n", topicName, message->payloadlen, (char *)message->payload); MQTTClient_freeMessage(&message); MQTTClient_free(topicName); return 1;
} // funzione main
int main(int argc, char *argv[])
{ // test del numero di argomenti if (argc != 4) { // errore:
```

```
numero errato di argomenti printf("%s: numero errato di argomenti\n", argv[0]); printf("uso: %s clientId topicPub topicSub" "(e.g.: %s client1 test/topic2 test/topic1)\n", argv[0], argv[0]); return EXIT_FAILURE; } // creo un client MQTTClient client; MQTTClient_create(&client, "tcp://localhost:1883", argv[1], MQTTCLIENT_PERSISTENCE_NONE, NULL); // set delle callback: in questo caso solo la onMessage() MQTTClient_setCallbacks(client, NULL, NULL, onMessage, NULL); // set delle opzioni di connessione MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
conn_opts.keepAliveInterval = 20; conn_opts.cleansession = 1; // mi collego int rc; if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) { printf("errore di connessione: codice %d\n", rc); return EXIT_FAILURE; } // mi sottoscrivo a un topic (topicSub in "uso: %s clientId topicPub topicSub") printf("%s: mi sottoscrivo al topic %s\n", argv[1], argv[3]); MQTTClient_subscribe(client, argv[3], QOS); // pubblico messaggi sull'altro topic (topicPub in "uso: %s clientId topicPub topicSub") MQTTClient_message pubmsg = MQTTClient_message_initializer; char payload[128]; for (int i = 0; i < 3; i++) { // attendo 2 secondi prima di pubblicare un messaggio sleep(2); // pubblico snprintf(payload, sizeof(payload), "messaggio %d dal client %s", i, argv[1]); pubmsg.payload = (void*)payload; pubmsg.payloadlen = strlen(payload); pubmsg.qos = QOS; pubmsg.retained = 0; MQTTClient_deliveryToken token; MQTTClient_publishMessage(client, argv[2], &pubmsg, &token); MQTTClient_waitForCompletion(client, token, TIMEOUT); printf("Messaggio %d publicato sul topic %s\n", i, argv[2]); } // mi disconnetto sleep(2); MQTTClient_disconnect(client, TIMEOUT); MQTTClient_destroy(&client); return EXIT_SUCCESS;
}
```

Come avrete notato, il codice è abbastanza semplice e **super-commentato**, quindi con c'è molto più da spiegare. Faccio solo notare che questo esempio si collega a un *broker* locale (ossia installato sulla macchina stessa di esecuzione), quindi il collegamento è fatto con l'indirizzo: *"tcp://localhost:1883"*. Ma ci sono anche dei broker pubblici disponibili in rete che funzionano esattamente come quello

locale. Nel mio caso (Linux) il *broker* locale che uso è **Mosquitto** (disponibile anche per Windows) e funziona alla grande.

E, come detto sopra, compiliamo:

```
aldo@Linux $ gcc -o mqttclient mqttclient.c -lpaho-mqtt3c
```

ed eseguiamo in due terminali diversi (*possibilmente in quasi-contemporanea, per non perdersi messaggi: se siete troppo lenti, quando il secondo client parte il primo potrebbe già aver pubblicato tutti i suoi messaggi, ah ah ah. Comunque le "strane" sleep(3) che ho aggiunto sono proprio per rendere leggibili i risultati*). Il risultato è questo:

terminale 1:

```
aldo@Linux $ ./mqttclient Client2 test/topic1 test/topic2
Client2: mi sottoscrivo al topic test/topic2
Messaggio 0 publicato sul topic test/topic1
messaggio ricevuto sul topic test/topic2: "messaggio 0 dal client Client1"
Messaggio 1 publicato sul topic test/topic1
messaggio ricevuto sul topic test/topic2: "messaggio 1 dal client Client1"
Messaggio 2 publicato sul topic test/topic1
messaggio ricevuto sul topic test/topic2: "messaggio 2 dal client Client1"
```

terminale 2:

```
aldo@Linux $ ./mqttclient Client1 test/topic2 test/topic1
Client1: mi sottoscrivo al topic test/topic1
messaggio ricevuto sul topic test/topic1: "messaggio 0 dal client Client2"
Messaggio 0 publicato sul topic test/topic2
messaggio ricevuto sul topic test/topic1: "messaggio 1 dal client Client2"
Messaggio 1 publicato sul topic test/topic2
messaggio ricevuto sul topic test/topic1: "messaggio 2 dal client Client2"
Messaggio 2 publicato sul topic test/topic2
```

Visto? Funziona perfettamente!

Ok, per oggi può bastare. Nel prossimo articolo proveremo a usare la leggerissima libreria **MQTT-C**. E, come spesso vi raccomando, *non trattenete il respiro nell'attesa! (può nuocere gravemente alla salute).*

Ciao, e al prossimo post!

https://italiancoders.it/mqtt-come-usare-mqtt-in-c-pt-1/