MQTT – come usare MQTT in C – pt.2

Logan: *Dove vado non c'è redenzione. E chi la vuole?*

Dove eravamo rimasti? Ah, si, **MQTT** (*Message Queuing Telemetry Transport*), un bel protocollo di comunicazione da studiare e usare. Nello scorso articolo abbiamo cominciato realizzando una implementazione di un semplice *Client* MQTT usando la *libpaho,* che è la libreria più usata e (probabilmente) più performante, ma non è l'unica: e quindi, come promesso, in questo secondo articolo mostrerò un altro semplice esempio di *Client,* questa volta usando la libreria **MQTT-C**. Ah, il film della frase qui sopra è lo stesso dell'altra volta, lo splendido **Logan** del bravo **James Mangold**. E, direi, sono d'accordo con il buon **Wolverine/H.Jackman**: anche noi programmatori **C** non abbiamo bisogno di redenzione… o forse si? *Beh, chi usa Linux sicuramente no, mentre gli altri… ah ah ah.*



…non ho bisogno di redenzione, ho bisogno di un MQTT funzionante!…

Come già anticipato nello scorso articolo, 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. La libreria consiste in solo due file sorgente (*mqtt.c* e *mqtt_pal.c*) da includere nel progetto e non dipende da librerie di sistema specifiche, rendendola ideale per ambienti con risorse limitate. L’interfaccia di programmazione è relativamente semplice: si può inizializzare un *Client,* connettersi a un *broker,* pubblicare messaggi e sottoscriversi ai *“topic”* con poche chiamate dirette.

Per l’uso, è necessario includere l’*header mqtt.h* (e, indiretamente, l’altro *header mqtt_pal.h*) e istanziare un oggetto *mqtt_client*. La ben più complessa *Eclipse Paho MQTT C Client Library* offre API sincrone e asincrone con supporto nativo per SSL/TLS e threading, mentre la semplice MQTT-C delega la gestione della rete e dell’I/O alle risorse del sistema ospite (Linux, un RTOS embedded o addirittura un *“bare-metal”*), mantenendo lo stack MQTT dedicato e leggero.

Il *Client* di esempio che ho scritto oggi è, più o meno, come quello dello scorso articolo: 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 MQTT-C)
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "include/mqtt.h"
```

```
#include "templates/posix_sockets.h" // prototipi locali
void onMessage(void** unused, struct mqtt_response_publish
*published);
void* clientDaemon(void* client);
void cleanExit(int status, int sockfd, pthread_t
*client_daemon); // 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; } // apro il non-
blocking TCP socket collegato al broker int sockfd; if
((sockfd = open_nb_socket("localhost", "1883")) == -1) {
printf("non posso aprire il socket\n");
cleanExit(EXIT_FAILURE, sockfd, NULL); } // creo un client
struct mqtt_client client; // dimensionare i buffer in base ai
messaggi previsti in uso uint8_t sendbuf[2048]; uint8_t
recvbuf[1024]; mqtt_init(&client, sockfd, sendbuf,
sizeof(sendbuf), recvbuf, sizeof(recvbuf), onMessage); // crea
una sessione anonima/vuota e invio una richiesta di
connessione al broker const char* client_id = NULL; uint8_t
connect_flags = MQTT_CONNECT_CLEAN_SESSION;
mqtt_connect(&client, client_id, NULL, NULL, 0, NULL, NULL,
connect_flags, 400); // check errori if (client.error !=
MQTT_OK) { fprintf(stderr, "error: %s\n",
mqtt_error_str(client.error)); cleanExit(EXIT_FAILURE, sockfd,
NULL); } // avvio un thread per aggiornare il client (gestisce
il traffico I/O del client) pthread_t client_daemon;
if(pthread_create(&client_daemon, NULL, clientDaemon,
&client)) { printf("non posso avviare il client daemon\n");
cleanExit(EXIT_FAILURE, sockfd, NULL); } // mi sottoscrivo a
un topic (topicSub in "uso: %s clientId topicPub topicSub")
printf("%s: mi sottoscrivo al topic %s\n", argv[1], argv[3]);
mqtt_subscribe(&client, argv[3], 0); // pubblico messaggi
sull'altro topic (topicPub in "uso: %s clientId topicPub
topicSub") for (int i = 0; i < 3; i++) { // attendo 2 secondi
prima di pubblicare un messaggio sleep(2); // pubblico char
payload[256]; snprintf(payload, sizeof(payload), "messaggio %d
dal client %s", i, argv[1]); mqtt_publish(&client, argv[2],
payload, strlen(payload) + 1, MQTT_PUBLISH_QOS_0); // check
```

```
errori if (client.error != MQTT_OK) { fprintf(stderr, "error: %s\n", mqtt_error_str(client.error)); cleanExit(EXIT_FAILURE, sockfd, &client_daemon); } printf("Messaggio %d publicato sul topic %s\n", i, argv[2]); } // mi disconnetto sleep(2); cleanExit(EXIT_SUCCESS, sockfd, &client_daemon);
} // onMessage - una callback per i messaggi che arrivano sul topic sorvegliato
void onMessage(void** unused, struct mqtt_response_publish *published)
{ char *topic_name = malloc(published->topic_name_size + 1); memcpy(topic_name, published->topic_name, published->topic_name_size);
topic_name[published->topic_name_size] = '\0'; printf("messaggio ricevuto sul topic %s: \"%s\"\n", topic_name, (const char*)published->application_message); free(topic_name);
} // clientDaemon - un thread che sincronizza l'I/O mqtt
void* clientDaemon(void* client)
{ // loop infinito del thread for (;;) { // sincronizzo mqtt_sync((struct mqtt_client*) client); // thread sleep (NOTA: in produzione NON usare usleep(3). Usare nanosleep(2)) usleep(100000U); } return NULL;
} // cleanExit - uscita controllata
void cleanExit(int status, int sockfd, pthread_t *client_daemon)
{ // chiudo il socket if (sockfd != -1) close(sockfd); // cancello il thread (NOTA: in produzione NON usare pthread_cancel(3) ma uno stop-flag) if (client_daemon != NULL) pthread_cancel(*client_daemon); // chiudo l'applicazione exit(status);
}
```

Come avrete notato, il codice è abbastanza semplice e **super-commentato**, quindi con c'è molto più da spiegare. Se fate un confronto con l'analogo *Client* scritto per la *libpaho* noterete che il codice è veramente molto simile. Faccio solo notare che anche 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.

L'esempio che ho scritto è ispirato dagli esempi contenuti nella pagina **Github di MQTT-C**, ho fatto un po' di cambi ma l'ispirazione proviene da lì. Per cui faccio anche notare, come da apposite NOTE che ho aggiunto al codice, che sono presenti alcune cose da cambiare per una eventuale versione di produzione di questo Client. Le NOTE sono:

- *NOTA: in produzione NON usare usleep(3). Usare nanosleep(2):* come già visto in un **mio vecchio articolo**, **usleep(3)** è deprecata da tempo, e bisogna usare **nanosleep(2)**, anche se è probabile (ma non sicuro) che la **usleep(3)** della *libc* sia implementata internamente usando **nanosleep(2)**.
- *NOTA: in produzione NON usare pthread_cancel(3) ma uno stop-flag):* come già visto in un **mio vecchio articolo**, **pthread_cancel(3)** è una funzione molto problematica di cui sconsiglio l'uso: meglio passare uno *stop-flag* al *thread* per forzarne una uscita pulita.

Ed è giunta l'ora di compilare ed eseguire. Compiliamo:

```
aldo@Linux $ gcc -o mqttclient mqttclient.c mqtt_pal.c mqtt.c
```

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 ricevuto sul topic test/topic2: "messaggio 0 dal client Client1"
Messaggio 0 publicato sul topic test/topic1
```

```
messaggio ricevuto sul topic test/topic2: "messaggio 1 dal client Client1"
Messaggio 1 publicato sul topic test/topic1
messaggio ricevuto sul topic test/topic2: "messaggio 2 dal client Client1"
Messaggio 2 publicato sul topic test/topic1
```

TERMINALE 2:

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

Visto? Anche questo funziona perfettamente!

Ok, per oggi può bastare. Nel prossimo articolo proveremo (*udite! udite!*) a cambiare linguaggio e implementeremo un *Client* MQTT in **Go:** non è la prima volta che affrontiamo il gran linguaggio **Go** su queste pagine eh! Non dovrebbe sorprendervi! E, come già vi ho raccomandato l'altra volta, 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-2/