The D-Bus – come usare D-Bus in C (e Go) – pt.1

regista (H.Ellis): *Tu.*
Hrundi V.Bakshi (P.Sellers): *Io?*
regista: *Sì, tu. Scendi dal mio set ed esci dalla mia foto. Spegni, spegni! Sei finito, hai finito! Farò in modo che tu non faccia mai più un altro film!*
Hrundi V.Bakshi: *Include la televisione, signore?*

Nel bellissimo film ***The Party*** (***Hollywood Party*** in Italia) il gran **Peter Sellers** si intrufola (da invitato per sbaglio) in un party del *jet set* hollywoodiano, e scombussola un po' l'ambiente… Ecco, il tema di questo articolo si riferisce a un sistema di IPC (i*nter-process communication*) che si è intrufolato un po' tardi (nel 2003) nel mondo IPC, quando già ne esistevano mille tipi ben consolidati, eppure si è fatto strada e oggi è usatissimo. Preparatevi, in questo articolo (e nei prossimi) parleremo di **D-Bus**!



…ma da dove è arrivato 'sto D-Bus?…
Visto che l'argomento è vasto questo sarà il primo articolo di

una serie, e per introdurre l'argomento mi sembra il caso di iniziare con:

Un po' di storia…

D-Bus (Desktop Bus) è un sistema di comunicazione tra processi (IPC) utilizzato principalmente in ambienti **Linux** e **Unix-like** (e quindi POSIX). È stato sviluppato da **Red Hat** e dalla community **freedesktop.org** per offrire alle applicazioni un modo semplice e standardizzato per scambiarsi messaggi e servizi senza bisogno di supportarsi direttamente tra loro. E, come dice il nome stesso, è un sistema nato per facilitare la comunicazione tra le applicazioni dello stesso *desktop* (ambiente grafico) e tra l'ambiente *desktop* e il sistema operativo. Interessante, no?

D-Bus si basa su un *daemon* (un *demone* Unix, un classico) che gestisce il traffico dei messaggi e permette:

- La comunicazione tra applicazioni dello stesso ambiente *desktop* (*Session Bus*).
- La comunicazione tra il sistema operativo e l'ambiente *desktop* o tra processi di sistema (*System Bus*).

L'architettura di D-Bus è strutturata su tre livelli: la libreria di base **libdbus**, il *daemon* di gestione dei messaggi e le librerie *wrapper* specifiche per diversi ambienti applicativi, facilitando l'integrazione e la gestione del ciclo di vita dei processi in modo affidabile. E aggiungo un ultimo dettaglio: anche se D-Bus è basato nativamente, come visto sopra, su un *daemon,* è comunque anche un *IPC classico*, e quindi supporta la connessione diretta (*peer-to-peer*) che non usa il *daemon*: ma questa opzione si usa solo per casi particolari.

Ah, prima che mi dimentico: D-Bus è anche multi-piattaforma, quindi è disponibile (ma poco usato) anche in quel sistema operativo che comincia con W e che, di solito, *preferisco non nominare nemmeno (ah ah ah).*

E come si scrive il codice C per usare D-Bus?

Ecco, *or incomincian le dolenti note*: la libreria base storica, quella nata insieme al protocollo è la **libdbus**: è una libreria *low-level* completissima, ben fatta, super-testata e ben funzionante ma… *praticamente inutilizzabile (ah ah ah)* perché criptica e complicatissima da usare così com'è. Infatti, nella documentazione ufficiale di **freedesktop.org se ne sconsiglia l'uso**!

"È opportuno sottolineare che l'implementazione di basso livello non è pensata principalmente per essere utilizzata dagli autori delle applicazioni. Si tratta piuttosto di una base per gli autori di binding e di un riferimento per le reimplementazioni. Se possibile, si raccomanda di utilizzare uno dei binding o delle implementazioni di livello superiore. Un elenco di questi è disponibile nella pagina dedicata ai binding." [cit. freedesktop.org - What is D-bus?]

e lo stesso consiglio si trova in altre fonti, **tra qui questa**:

"NOTA: si sconsiglia di utilizzare questa API a meno che non sia assolutamente necessario. Gli sviluppatori di applicazioni dovrebbero utilizzare uno dei binding, se possibile. Se si scrive in in C, si consigliano i binding GLib." [cit. Matthew Johnson - Using the DBUS C API]

E allora che si fa?

Rimanendo sul C si potrebbe/dovrebbe usare uno dei *wrapper* disponibili, ad esempio **GDBus**, ma questo è molto legato all'ambiente grafico **GNOME/GTK**, quindi volendo scrivere del Software un po' più generico (è il nostro caso) non è adatto. Oppure si potrebbero usare i *wrapper* in C++ (*ehm…*): ad esempio **QtDBus**, ma per questo valgono le stesse considerazioni fatte per **GDBus** (e in questo caso si dipenderebbe da **Qt**). Oppure usare **dbus-c++**, che è, effettivamente, un buon *wrapper* diretto di **libdbus:** facilita molto la scrittura del codice, ma è un po' obsoleto e credo che non sia più un progetto attivo. E poi, la domanda iniziale era: *"E come si scrive il codice C per usare D-Bus?"* Quindi dimentichiamoci del C++ (*bene! Anzi,*

benissimo!).

Ripetiamo: *“E allora che si fa?”* La risposta sarà nel prossimo articolo! Lì vedremo l’evoluzione Software che ha avuto il nostro D-Bus e le nuove librerie disponibili (in C!).

Comunque non posso terminare un articolo senza codice, no? Quindi ho scritto due applicazioni in C (un *Server* e un *Client)* funzionanti (parlano tra loro) e che usano **libdbus***.* Vai col codice! Prima il *Server:*

```
// server.c - un server D-Bus con libdbus (usa dbus-daemon)
#include <stdio.h>
#include <stdlib.h>
#include <dbus/dbus.h> // nomi condivisai tra server e client
#define SRV_BUS_NAME "com.esempio.server" // nome del bus del server
#define SRV_OBJ_PATH "/com/esempio/object" // percorso dell'oggetto
#define IFACE_NAME "com.esempio.interface" // nome dell'interfaccia
#define METHOD_NAME "SendAndRecv" // nome del metodo // funzione main()
int main()
{ // init gestione errori DBusError err; dbus_error_init(&err); // mi connetto al bus di sessione DBusConnection *conn = dbus_bus_get(DBUS_BUS_SESSION, &err); if (dbus_error_is_set(&err)) { // errore di connessione: esco fprintf(stderr, "errore di connessione: %s\n", err.message); dbus_error_free(&err); return EXIT_FAILURE; } // richiedo un nome noto sul bus int ret = dbus_bus_request_name(conn, SRV_BUS_NAME, DBUS_NAME_FLAG_REPLACE_EXISTING, &err); if (dbus_error_is_set(&err)) { // errore sul nome: esco fprintf(stderr, "errore nel nome del bus: %s\n", err.message); dbus_error_free(&err); return EXIT_FAILURE; } // check dei privilegi if (ret != DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER) { // errore di privilegi: esco fprintf(stderr, "non sei il proprietario primario del nome del bus\n"); return EXIT_FAILURE; } // loop di comunicazione col client for (;;) { // Leggo i dati dal socket e li inserisco nella coda dei messaggi dbus_connection_read_write_dispatch(conn, -1); // -1
```

per blocking mode DBusMessage *msg = dbus_connection_pop_message(conn); if (msg == NULL) { // continuo fino a quando non c'è un messaggio continue; } // controllo se il messaggio è una chiamata al metodo corretto if (dbus_message_is_method_call(msg, IFACE_NAME, METHOD_NAME)) { // estraggo gli argomenti (di tipo stringa) char *param = NULL; if (dbus_message_get_args(msg, &err, DBUS_TYPE_STRING, &param, DBUS_TYPE_INVALID)) { // tutto Ok: mostro il messaggio printf("messaggio ricevuto dal client: %s", param); // creo il messaggio di risposta (method return) char *reply_str = malloc(256); snprintf(reply_str, 256, "mi hai scritto: %s", param); DBusMessage *reply = dbus_message_new_method_return(msg); // accodo l'argomento alla risposta DBusMessageIter args; dbus_message_iter_init_append(reply, &args); if (!dbus_message_iter_append_basic(&args, DBUS_TYPE_STRING, &reply_str)) { // errore di accodamento fprintf(stderr, "errore di memoria nell'accodare la risposta\n"); } // invio la risposta e faccio il flush della coda if (!dbus_connection_send(conn, reply, NULL)) { // errore di invio fprintf(stderr, "errore nell'invio della risposta\n"); } dbus_connection_flush(conn); // libero la memoria del messaggio di risposta dbus_message_unref(reply); free(reply_str); } else { // errore sui parametri fprintf(stderr, "errore nella lettura dei parametri: %s\n", err.message); dbus_error_free(&err); } } // Libero il messaggio elaborato dbus_message_unref(msg); } // esco con Ok return EXIT_SUCCESS;
}

E poi il *Client:*

```
// client.c - un client D-Bus con libdbus (usa dbus-daemon)
#include <stdio.h>
#include <stdlib.h>
#include <dbus/dbus.h> // nomi condivisai tra server e client
#define SRV_BUS_NAME "com.esempio.server" // nome del bus del server
#define SRV_OBJ_PATH "/com/esempio/object" // percorso dell'oggetto
#define IFACE_NAME "com.esempio.interface" // nome dell'interfaccia
```

```
#define METHOD_NAME "SendAndRecv" // nome del metodo // funzione main()
int main()
{   //   init   gestione   errori   DBusError   err;   dbus_error_init(&err); // mi connetto al bus di sessione DBusConnection *conn = dbus_bus_get(DBUS_BUS_SESSION, &err); if (dbus_error_is_set(&err)) { // errore di connessione: esco fprintf(stderr, "errore di connessione: %s\n", err.message); dbus_error_free(&err); return EXIT_FAILURE; } // loop di comunicazione col server char *input_str = malloc(128); for (;;) { // compongo messaggio per il server printf("scrivi un messaggio per il server: "); fgets(input_str, 128, stdin); // creo la chiamata al metodo (destinatario, percorso oggetto, interfaccia, metodo) DBusMessage *msg; if ((msg = dbus_message_new_method_call(  SRV_BUS_NAME,  SRV_OBJ_PATH, IFACE_NAME, METHOD_NAME)) == NULL) { // errore di creazione fprintf(stderr, "errore nella creazione del messaggio\n"); return EXIT_FAILURE; } // aggiungo la stringa di input come argomento             DBusMessageIter             args;   dbus_message_iter_init_append(msg,    &args);    if (!dbus_message_iter_append_basic(&args,  DBUS_TYPE_STRING, &input_str)) { // errore: esco fprintf(stderr, "errore nell'inserimento dell'argomento\n"); return EXIT_FAILURE; } // invio il messaggio e attendo la risposta (asincrona e bloccante)     DBusPendingCall     *pending;     if (!dbus_connection_send_with_reply(conn, msg, &pending, -1)) { // errore di invio: esco fprintf(stderr, "errore nell'invio del messaggio\n"); return EXIT_FAILURE; } // check dela risposta pendente if (pending == NULL) { // errore: esco fprintf(stderr, "la connessione è chiusa o inattiva\n"); return EXIT_FAILURE; } // flush della coda per forzare l'invio immediato sul socket dbus_connection_flush(conn); // libero il messaggio di richiesta originale dbus_message_unref(msg); // blocco il thread finché la risposta non è pronta dbus_pending_call_block(pending); // estraggo il messaggio di risposta         DBusMessage         *reply         = dbus_pending_call_steal_reply(pending); if (reply == NULL) { fprintf(stderr, "risposta nulla ricevuta\n"); return EXIT_FAILURE; } // libero la memoria della risposta pendente dbus_pending_call_unref(pending); // leggo il testo restituito dal server char *reply_str; if (dbus_message_get_args(reply,
```

```
&err, DBUS_TYPE_STRING, &reply_str, DBUS_TYPE_INVALID)) { // tutto Ok: mostro la risposta printf("risposta del server: %s", reply_str); } else { // errore di lettura fprintf(stderr, "errore nella lettura della risposta: %s\n", err.message); dbus_error_free(&err); } // libero la memoria del messaggio di risposta dbus_message_unref(reply); } free(input_str); // esco con Ok return EXIT_SUCCESS;
}
```

Che ve ne pare? Come al solito è tutto **stra-commentato**, quindi non c'è bisogno di aggiungere molti dettagli: la struttura del codice è, più o meno, simile a quella dei classici *Socket Server* e *Client* come quelli che ho già mostrato **qui** e **qui…** ma il problema è che il risultato finale è molto più complicato! È codice abbastanza incomprensibile, ed è l'esempio più semplice che mi è venuto in mente! Direi che le citazioni che ho fatto sopra sono veritiere!

Ah, se non vi fidate e volete volete vedere se funziona veramente, vi passo le linee di installazione della libreria (su Linux) e delle compilazioni di *Server* e *Client:*

```
sudo apt install libdbus-1-dev
gcc -Wall -I/usr/include/dbus-1.0 -I/usr/lib/x86_64-linux-gnu/dbus-1.0/include -o server server.c -ldbus-1
gcc -Wall -I/usr/include/dbus-1.0 -I/usr/lib/x86_64-linux-gnu/dbus-1.0/include -o client client.c -ldbus-1
```

dopodiché, se eseguite i due programmi in terminali separati vedrete la magia dello scambio di messaggi (interattivo) tra Client e Server. Provare per credere!

Ok, per oggi può bastare. Nel prossimo articolo vi parlerò, come anticipato sopra, del D-Bus con le librerie C! Restate sintonizzati!

Ciao, e al prossimo post!

P.S.

Ricordatevi di visitare il mio nuovissimo cineblog! Lo trovate

qui: **Polvere di Cinema**. Buona lettura!

https://italiancoders.it/the-d-bus-come-usare-d-bus-in-c-e-go-pt-1/