Il Logger – come scrivere un logger in C – pt.1

Don Vito Corleone: *Gli farò un'offerta che non potrà rifiutare.*

Il Logger è un po' come **Il Padrino**, il protagonista di uno dei capolavori del Maestro **Francis Ford Coppola**: lui lavora dietro le quinte e registra tutto, non gli sfugge niente. E quando è necessario bisogna ricorrere a lui per risolvere svariati problemi (*uff… e, ahimè, in questo caso la finzione cinematografica è molto vicina a cose che succedono nella realtà…*). Come avrete già capito, in questo articolo parleremo di **logger** e di **syslog**!



…gli farò un logger che non potrà rifiutare…

Quando si scrive Software bisogna sempre cercare di essere professionali, qualunque sia l'uso del Software che stiamo scrivendo. Ad esempio non ci si può dimenticare di prevedere un buon sistema di *log* (un *logger,* per gli amici). *Ci sono, poi, alcuni professionisti della programmazione che non sanno neanche cos'è un buon sistema di log… ma questa è un altra*

storia…

Per aggiungere un *logger* al nostro Software possiamo aiutarci col sistema operativo, ad esempio su **UNIX** e **Linux** abbiamo il **syslog** (*system logger*) che è molto flessibile e ci semplifica la vita. Nel caso di non potere (o non volere) usare i mezzi forniti dal sistema si può pensare di scriversi un proprio sistema di *log* (l'ho fatto alcune volte, ma erano esigenze particolari di progetto), ma il *syslog* dei sistemi **POSIX** è così pratico e flessibile che non si può rinunciare a usarlo. Comunque in un **mio vecchio articolo** potete trovare un buon esempio di *"logger indipendente".*

Sia perché non voglio fare un post troppo lungo (diventerebbe noioso), sia per seguire una specie di approccio specifica+implementazione (adattato a un blog di programmazione), ho diviso il post in due puntate: nella prima descriverò, a mo' di specifica funzionale, l'*header file* (*mylog.h*) e un esempio di uso (*main.c*). Nella seconda puntata descriverò la vera e propria implementazione.

Vediamo l'*header file*, *mylog.h*:

```
// prototipi globali
void openLog(int level);
void closeLog();
void traceLog(int level, const char* format, ...)
__attribute__ ((format(printf, 2, 3)));
// NOTA: il __attribute__ serve a mostrare warning (a compile-time) sui parametri di traceLog()
```

Semplicissimo e auto-esplicativo, no? Il nostro prodotto è formato da tre funzioni canoniche (*apri, chiudi, traccia*). L'argomento level di **openLog()** è il livello di *log* richiesto, e corrisponde a uno degli **otto livelli di gravità** previsti dal *syslog* standard (dal più grave al meno grave: LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, LOG_DEBUG). Quasi inutile spiegare che, settando un livello, nel file del *syslog* appariranno tutti i messaggi di

uguale o maggiore gravità del livello selezionato.

Merita invece un po' di attenzione il prototipo della funzione **traceLog()**: è la funzione per scrivere i messaggi nel *syslog,* ed è una funzione ***printf-like***, quindi con argomenti variabili che si *formattano* usando l'argomento *format*. Notare che ho aggiunto, dopo il prototipo, questo attributo:

__attribute__ ((format(printf, 2, 3)));

A cosa serve? Come ben saprete le funzioni della famiglia **printf(3)** sono conosciute dal compilatore, e quindi se sbagliamo a formattare un argomento (ad esempio se assegniamo un **int** a un campo *%s* che si aspetta una **stringa)** il compilatore emette un *warning* che ci aiuta a correggere il problema a *compile-time* (che, nell'esempio appena descritto, si trasformerebbe in un *problemone a run-time*).

Il fatto è che la nostra funzione **traceLog()** è sconosciuta al compilatore, e quindi per generare eventuali *warning* bisogna presentargliela: e questo lo possiamo fare con l'apposito attributo **format(archetype, string-index, first-to-check)** che, nel nostro caso, specifica che è una **printf** (*archetype=printf*) con la stringa come secondo argomento (*string-index=2*) e gli argomenti variabili dalla terza posizione (*first-to-check=3*). Sfortunatamente tutto questo è valido solo per **GCC**, ma visto che è il compilatore più usato su **UNIX** e **Linux** mi sembra una informazione utilissima, no?

Vediamo ora il *main* dell'esempio d'uso: vai col codice!

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include "mylog.h" // prototipi locali
static int getLoglevel(const char* level); // main - funzione main
int main(int argc, char* argv[])
{ // test del numero di argomenti if (argc != 2) { // errore:
```

```
numero errato di argomenti fprintf(stderr, "%s: numero errato di argomenti\n", argv[0]); fprintf(stderr, "uso: %s loglevel (e.g.: %s debug)\n", argv[0], argv[0]); return EXIT_FAILURE; } // test del argomento loglevel int loglevel; if ((loglevel = getLoglevel(argv[1])) == -1) { // errore: livello di log sconosciuto fprintf(stderr, "%s: livello di log sconosciuto (%s)\n", argv[0], argv[1]); fprintf(stderr, "uso: %s loglevel (e.g.: %s debug)\n", argv[0], argv[0]); return EXIT_FAILURE; } // apro il log su syslog openLog(loglevel); // eseguo un test di traceLog() traceLog(LOG_EMERG, "log di tipo LOG_EMERG (il livello impostato è %s)", argv[1]); traceLog(LOG_ALERT, "log di tipo LOG_ALERT (il livello impostato è %s)", argv[1]); traceLog(LOG_CRIT, "log di tipo LOG_CRIT (il livello impostato è %s)", argv[1]); traceLog(LOG_ERR, "log di tipo LOG_ERR (il livello impostato è %s)", argv[1]); traceLog(LOG_WARNING, "log di tipo LOG_WARNING (il livello impostato è %s)", argv[1]); traceLog(LOG_NOTICE, "log di tipo LOG_NOTICE (il livello impostato è %s)", argv[1]); traceLog(LOG_INFO, "log di tipo LOG_INFO (il livello impostato è %s)", argv[1]); traceLog(LOG_DEBUG, "log di tipo LOG_DEBUG (il livello impostato è %s)", argv[1]); // chiudo il log closeLog(); // exit Ok return EXIT_SUCCESS;
} // getLoglevel - ritorna il loglevel come numero definito in syslog.h
static int getLoglevel( const char* level) // la stringa corrispondente al numero definito
{ if (!strcmp(level, "emerg")) // sistema non usabile return LOG_EMERG; else if (!strcmp(level, "alert")) // necessità di azione immediata return LOG_ALERT; else if (!strcmp(level, "crit")) // condizione critica return LOG_CRIT; else if (!strcmp(level, "err")) // condizione di errore return LOG_ERR; else if (!strcmp(level, "warning")) // condizione di warning return LOG_WARNING; else if (!strcmp(level, "notice")) // condizione normale, ma significativa return LOG_NOTICE; else if (!strcmp(level, "info")) // messaggio di informazione return LOG_INFO; else if (!strcmp(level, "debug")) // messaggio di debug return LOG_DEBUG; // livello di log sconosciuto return -1;
}
```

Semplice, no? Eseguo un test degli argomenti e, se vanno bene, apro il *logger,* lo uso, lo chiudo ed esco. Nel caso di una

errata linea di comando mostro l'errore e un esempio di lancio corretto: *"uso: ./mylog loglevel (e.g.: ./mylog debug)".* Il resto del codice è, come sempre, **stra-commentato**, quindi non credo che ci sia altro da aggiungere.

Ma, in pratica, usando una linea di comando del tipo *"./mylog err"* quale sara il risultato? Nel file del *syslog* verranno aggiunte le seguenti linee (delle quali spiegherò il contenuto nel prossimo articolo):

```
2025-10-27T19:12:48.035208+01:00 Linux mylog[8030]: [8030]: log di tipo LOG_EMERG (il livello impostato è err)
2025-10-27T19:12:48.036163+01:00 Linux mylog[8030]: [8030]: log di tipo LOG_ALERT (il livello impostato è err)
2025-10-27T19:12:48.036390+01:00 Linux mylog[8030]: [8030]: log di tipo LOG_CRIT (il livello impostato è err)
2025-10-27T19:12:48.036625+01:00 Linux mylog[8030]: [8030]: log di tipo LOG_ERR (il livello impostato è err)
```

Notare che, essendo il livello scelto nella linea di comando quello di LOG_ERR, non sono apparsi i messaggi di LOG_NOTICE, LOG_INFO e LOG_DEBUG. Aggiungo che le scritte di *log* possono apparire anche su terminale se (come vedremo nel prossimo articolo) il *syslog* viene aperto con questa modalità (che è opzionale).

Per oggi abbiamo finito. I più volenterosi potranno, nell'attesa della seconda parte, scrivere una propria implementazione, e poi confrontarla con la mia, *ma la mia sarà sicuramente migliore… (si allontana sghignazzando).*

Ciao e al prossimo post!

https://italiancoders.it/il-logger-come-scrivere-un-logger-in-c-pt-1/