

# Cenni su Librerie

Prof. Vittorio Ghini, Appunti delle lezioni di Sistemi Operativi,  
Corso di Laurea in Ingegneria e Scienze Informatiche, Università di Bologna

Le librerie sono dei file che contengono le implementazioni, in codice macchina, di alcune funzioni. Possono implementare funzioni di base messe a disposizione dal sistema operativo oppure funzioni definite da utenti.

Solitamente le librerie sono mantenute su disco in directory predefinite, in dei file il cui nome comincia per **lib**. L'estensione del nome del file che contiene la libreria è diverso a seconda che la libreria sia statica (.a) o condivisa (.so).

Escludendo il "lib" iniziale ed escludendo l'estensione, ciò che resta del nome del file della libreria rappresenta il **nome ufficiale** della libreria. Tale nome ufficiale è quello che si usa per indicare al linker quali librerie dobbiamo linkare, con l'opzione `-lnomelibreria`

Ad esempio, la libreria matematica che fornisce cos sin etc etc si chiama **m**, è contenuta nei file `/usr/lib/libm.a` (versione statica, se esiste) e `/lib/x86_64-linux-gnu/libm.so.6` (versione condivisa) e si passa al linker specificando il flag `-lm`

Le librerie si dividono in 2 categorie:

- librerie statiche (**static library**),
- librerie condivise o linkate dinamicamente (impropriamente, dinamiche) (**shared library** o **dynamically linked library**).

Le due categorie di librerie si distinguono in funzione di come vengono collegate all'eseguibile che le deve utilizzare. Vanno distinti due momenti:

- link-time (in cui si genera l'eseguibile, collegando le librerie ed i diversi moduli oggetto)
- run-time (il momento in cui si esegue l'eseguibile).

## Librerie Statiche

A link-time le librerie statiche sono fisicamente accorpate ai moduli e inserite assieme a queste nell'eseguibile generato. Il linker deve sapere dove cercare su disco le librerie statiche per copiarne il contenuto e inserirlo nell'eseguibile. Quindi l'eseguibile conterrà le librerie statiche. Da ciò deriva che a run-time l'eseguibile caricato in memoria avrà da subito in memoria pronto per l'esecuzione anche il codice macchina delle librerie.

Generalmente il linker cerca le librerie nella directory `/usr/lib`, ma si può ordinare al linker di cercare, a link-time, le librerie statiche anche in una o più directory diverse specificandole al linker con uno o più parametri `-Llinktimepath` (ad esempio `-L/home/studente/lib` `-L/nuovo/percorso/assoluto/fino/alla/directory/delle/librerie`)

## Librerie Condivise.

A link-time le librerie condivise NON sono fisicamente accorpate ai moduli e NON sono inserite nell'eseguibile generato. Invece, per ciascuna chiamata ad una funzione della libreria condivisa, il linker inserisce nell'eseguibile una porzione di codice macchina (stub) e la posizione a run-time su disco della libreria.

A run-time, al momento dell'esecuzione della chiamata a funzione, lo stub cercherà la libreria condivisa sul disco del computer in cui avviene l'esecuzione del programma, la caricherà in memoria rendendola disponibile per l'esecuzione e poi effettuerà la chiamata alla funzione saltando nel codice della libreria appena caricata in memoria.

Se le librerie non sono nelle directory predefinite del sistema, il linker deve perciò sapere due cose:

- 1) dove si trova a link-time la libreria condivisa,

## 2) dove si troverà a run-time la libreria su disco,

Approfondiamo le motivazioni delle due informazioni necessarie al linker per linkare librerie caricate dinamicamente:

1) Il linker deve sapere dove si trova a link-time la libreria condivisa, poiché deve i) verificare se questa libreria fornisce veramente la funzione richiesta e ii) copiare dalla libreria il solo stub che effettua il caricamento. Con l'opzione **-Llinktimepath** diciamo al linker dove sono a link-time le librerie condivise.

2) Il linker deve sapere dove si troverà a run-time la libreria su disco, poiché deve inserire questa posizione nel codice assieme allo stub. Con l'opzione **-Wl,-rpath,runtimepath** diciamo al linker dove saranno le librerie condivise a run-time.

Ad esempio, se devo generare un eseguibile prog.exe, linkando un modulo oggetto prog.o che necessita della libreria libmia.so la quale si trova, a link-time in /home/studente/temp, e a run-time si troverà nella directory /home/studente/lib, allora devo linkare l'eseguibile eseguendo il comando:

```
gcc altriflag -o prog.exe prog.o -L/home/studente/temp -Wl,-rpath,/home/studente/lib -lmia
```

## Utilities per le librerie

Le librerie ed i file binari eseguibili sono dei file strutturati secondo un formato standard, in ambito Linux il più usato è il formato elf. Esistono alcune utilities che permettono di verificare il contenuto delle librerie.

Per sapere di quale librerie condivise necessita un file binario eseguibile o una libreria, lanciare:

```
ldd nome_eseguibile (oppure ldd percorsolibreria)
```

Ad esempio, vediamo l'output ottenuto eseguendo il seguente comando:

```
ldd prog.exe
```

```
linux-vdso.so.1 => (0x00007ffc844d7000)
libmia.so => /home/studente/lib/libmia.so (0x00007f4504dd3000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f45049f7000)
/lib64/ld-linux-x86-64.so.2 (0x0000558aeabc0000)
```

Per sapere quali informazioni di linking dinamico sono state inserite in una libreria condivisa oppure in un eseguibile, si può usare l'eseguibile readelf, ad esempio lanciando il comando:

```
readelf -d percorsolibreria (oppure percorsoeseguibile)
```

Ad esempio, vediamo l'output ottenuto eseguendo il seguente comando:

```
readelf -d prog.exe (analogamente readelf -d libmia.so)
```

```
Dynamic section at offset 0xe08 contains 26 entries:
```

Tag	Type	Name/Value
0x0000000000000001	(NEEDED)	Shared library: [libmia.so]
0x0000000000000001	(NEEDED)	Shared library: [libc.so.6]
0x000000000000000f	(RPATH)	Library rpath: [/home/studente/lib]
0x000000000000000c	(INIT)	0x400788

Anche readelf riconosce le dipendenze annidate, e visualizza anche le librerie utilizzate indirettamente, cioè quelle usate dalle librerie direttamente usate nell'eseguibile o libreria.

Aggiungere due parole sull'utility **file**

file prog.exe

prog.exe: **ELF 64-bit LSB executable, x86-64**, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=df0c3fc103bc4c91e3709c53fa35ff45cb5b8e97, **not stripped**

file libmia.so

libmia.so: **ELF 64-bit LSB shared object, x86-64**, version 1 (SYSV), dynamically linked, BuildID[sha1]=2a187da98f0a66cb254f340c7249301bd93a7b40, **not stripped**

file mia.o

mia.o: **ELF 64-bit LSB relocatable**, x86-64, version 1 (SYSV), not stripped

file mia.c

mia.c: C source, ASCII text

file mia.h

mia.h: C source, ASCII text

**AGGIUNGERE spiegazione su utility nm per visualizzare le tabelle dei simboli dei file oggetto, librerie ed eseguibili.**

Quando faccio il linking dei moduli oggetto per generare un eseguibile, il linker cerca di utilizzare PRIMA le librerie CONDIVISE, e solo se non le trova, usa le librerie statiche.

Generalmente il linker cerca le librerie nella directory `/usr/lib`, ma si puo' ordinarli di cercarle in una directory diversa specificandola al linker con il parametro `-L/nuovo/percorso`

Se voglio che il linker usi solo le librerie statiche devo specificare al gcc il flag **-static**

Domanda da approfondire: come posso dire al linker di usare solo le versioni condivise, e se non ci sono di dare un errore?

-----  
Come fare a sapere di quali librerie condivise un eseguibile `a.out` ha bisogno?  
Si utilizza l'eseguibile **ldd** (List Dynamic Dependencies)

```
ldd /usr/bin/lynx
    libncurses.so.1 => /usr/lib/libncurses.so.1.9.6
    libc.so.5 => /lib/libc.so.5.2.18
```

Domanda da approfondire: ldd riconosce le dipendenze annidate? Ovvero, se `a.out` usa la libreria condivisa `libncurses.so` e la libreria `libncurses` utilizza a sua volta la libreria condivisa `libncurses.so`, allora ldd ci dice che `a.out` utilizza la libreria `libncurses.so`? SI, l'ho verificato !

-----  
L'eseguibile **nm** elenca tutti i simboli per i quali esistono dei riferimenti nella libreria che viene passata come argomento:

```
nm libc.so.5
00010fe8 U getname
00010fe8 W tcgetattr
00068718 T tcgetpgrp
00010fe8 T __tcgetattr
```

**U** getname            significa che il simbolo `getname` e' usato ma NON definito (unknown) ovvero e' contenuto in una diversa libreria.

**T** tcgetpgrp        significa che il simbolo `tcgetpgrp` e' definito all'interno della libreria (il modo normale).

**W** tcgetattr        significa che il simbolo `tcgetattr` e' definito all'interno della libreria, ma in una forma detta weak (W) ovvero in modo tale da poter essere sostituito da un'altra definizione in una libreria diversa. (Domanda da approfondire: anche nella stessa libreria? Credo di si).

A	The symbol's value is absolute, and will not be changed by further linking.
B	Un-initialized data section
D	Initialized data section
T	Normal code section
U	Undefined symbol used but not defined. Dependency on another library.
W	Doubly defined symbol. If found, allow definition in another library to resolve dependenc

l'eseguibile **nm -D**

elenca i simboli per i quali esistono dei riferimenti nella libreria che viene passata come argomento:

-----

per elencare tutte le librerie dinamiche presenti in memoria, si puo' utilizzare il comando

**lsdf** listing open files

specificando alcuni parametri

**lsdf | grep mem | grep "\.so" | awk '{print \$9}' | sort | uniq**

## ESEMPIO DI LIBRERIE ANNIDATE

### albero delle dipendenze

