Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Departamento de Electrónica

Asignatura: Informática I

Creando y Usando Librerías

(mas bien.... afianzando la existencia del LINKER!!!!)

Autor: Ing. Alejandro Furfaro. Profesor Titular

Índice de contenido

Introducción.

Una librería de programas no es otra cosa que un archivo que contiene código y datos compilados y funcionales, que serán incorporados a otros programas cuando estos los requieran.

Las librerías de código facilitan la reutilización de código evitando tener que re escribirlos cada vez que los necesitamos y a su vez permiten el trabajo en equipo, de manera de abstraernos de la implementación y enfocarnos en su utilización. Normalmente el código de una librería es de uso general, como por ejemplo *printf*. Exactamente. Esta función es muy general, y prácticamente nos permite acceder a la pantalla para imprimir en ella lo que se nos ocurra y con el formato que necesitemos. Y no tenemos que preocuparnos ni siquiera de conocer su código. Todo lo que tenemos que tener es el archivo de la librería que tiene el código de *printf* presente en nuestro sistema e indicarle al linker en donde está ese archivo para que pueda resolver las llamadas que hacemos desde nuestro código a las funciones de código empaquetadas en esa librería y el archivo cabecera (con extensión *.h*) con la definición de la función, en este caso *printf*, que se deberá incluir en nuestro programa.

Resumen, una librería se compone de:

- Prototipos de las funciones (archivo/s cabecera/s .h)
- Definición de las funciones (archivos/s fuente .c)

Ejemplos de trabajo.

A lo largo de esta guía vamos a trabajar con estos archivos fuente.

```
Archivo holalib.h:
/* Header para holalib.c
*/
#ifndef HOLA_H
#define HOLA_H

# include <stdio.h>

void hola (void);

#endif
```

Notar que hemos incluido una compilación condicional aprovechando los buenos oficios del preprocesador.

Normalmente los proyectos informáticos incluyen múltiples archivos fuente, de

modo de compilar por separado y unirlos en un ejecutable por medio del linker.

En esas condiciones si se incluyen archivos headers que duplican includes, podemos arribar a errores de linkeo por re definición de elementos (en especial si se definen estructuras)

Para evitarlo chequeamos si una macro (en este caso HOLA_H), ha sido definida. Si no lo fue , la definimos y luego definimos todos los prototipos y macros que necesitemos. Si otro fuente del proyecto incluye este archivo header, entonces encontrará definida la macro HOLA_H, y salteará la re definición de los prototipos y macros evitándonos los consecuentes errores de compilación.

Archivo *holalib.c*:

```
/* Programa holalib.c
Ejemplo simple para generar una librería.
La función hola que imprime en stdout el viejo y querido Hola Mundo!
*/
#include “holalib.h”
void hola (void)
{
      printf ("Hola Mundo!\n");
}
```

Y *test_holalib.c* que será el programa de prueba de nuestra función de librería

```
/* Programa de prueba para nuestra librería holalib
*/
# include "holalib.h"
int   main (void)
{
  hola ();
  return 0;
}
```

Librerías

En los sistemas UNIX like hay tres formas de implementar una librería de código:

- Estática.

- Compartida (shared)
- De carga Dinámica (DL, por Dynamic Loaded)

En el código que componen los programas objeto de una librería (independientemente del tipo que esta sea), nunca se tiene una función *main*, ya que el código que incluimos en las librerías es para ser invocado desde programas de aplicación los cuales necesitan indispensablemente una función *main*. Por lo tanto, el código que compone una librería, no representa un programa que pueda ejecutarse en forma directa, sino que será invocado desde programas que dispongan de las llamadas adecuadas a ellos.

Librerías estáticas

Son simples colecciones de programas objeto agrupados en un único archivo cuyo nombre típicamente finaliza en '*.a*'.

Al compilar un programa que hace uso de código contenido en una librería estática, en el momento de realizar la fase de enlace (link), se copia en nuestro programa el código objeto de la librería (luego lo verificaremos observando el tamaño en bytes final de nuestro programa), es decir que no es necesario distribuir nuestro programa con la librería ya que ésta se encuentra incrustada de nuestro nuevo programa.

Para generar una librería estática utilizamos el utilitario *ar (empaqueta archivos objetos en un único archivo)*. Antes de ello obviamente debemos haber generado el programa objeto, compilando con la opción *-c*.

```
alejandro@notebook:~/InfoI/second$ gcc -c holalib.c
alejandro@notebook:~/InfoI/second$ ls -las
total 24
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 14 12:45 .
4 drwxr-xr-x 6 alejandro alejandro 4096 feb 14 12:23 ..
4 -rw-r--r-- 1 alejandro alejandro  209 feb 14 12:29 holalib.c
4 -rw-r--r-- 1 alejandro alejandro   44 feb 14 12:25 holalib.h
4 -rw-r--r-- 1 alejandro alejandro  836 feb 14 12:45 holalib.o
4 -rw-r--r-- 1 alejandro alejandro  121 feb 14 12:32 test_holalib.c
alejandro@notebook:~/InfoI/second$ ar rcs libhola.a holalib.o
alejandro@notebook:~/InfoI/second$ ls -las
total 28
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 14 12:46 .
4 drwxr-xr-x 6 alejandro alejandro 4096 feb 14 12:23 ..
4 -rw-r--r-- 1 alejandro alejandro  978 feb 14 12:46 libhola.a
4 -rw-r--r-- 1 alejandro alejandro  209 feb 14 12:29 holalib.c
```

```
4 -rw-r--r-- 1 alejandro alejandro   44 feb 14 12:25 holalib.h
4 -rw-r--r-- 1 alejandro alejandro  836 feb 14 12:45 holalib.o
4 -rw-r--r-- 1 alejandro alejandro  121 feb 14 12:32 test_holalib.c
```

El comando *ar* se ejecutó con tres opciones: *rcs*.

r indica que reemplace al elemento previamente existente con el mismo nombre. En nuestro caso si existía dentro de la librería *libhola.a* un elemento llamado *holalib.o*, *ar* lo pisa por este nuevo.

c indica que cree el archivo de librería especificado si no existiere.

s le indica actualizar el índice de la librería una vez inserto el archivo solicitado.

En ambos casos tenemos listados los directorios luego de cada comando para apreciar los resultados.

Si queremos listar los archivos (objetos) que componen la libraría, el comando es

```
alejandro@notebook:~/InfoI/second$ ar -t libhola.a
holalib.o
alejandro@notebook:~/InfoI/second$
```

Tenemos todo en orden. Cabe un solo comentario antes de continuar. Las librerías en general llevan en su nombre el prefijo "lib". Por lo tanto el nombre a los efectos del linker es lo que sigue a "lib" y precede a ".a".

```
alejandro@notebook:~/InfoI/second$ gcc -g -o demo test_holalib.o -L. -lhola
alejandro@notebook:~/InfoI/second$
```

aquí vemos que llamando simplemente *hola* a nuestra librería es suficiente para el linker. Hemos comprobado (no sin sufrir considerablemente) que si no se antepone "lib" al nombre del archivo que contiene la librería, el linker (que tiene su sensibilidad, no vayan a creer) se ofusca y devuelve un mensaje de error abortando el proceso de generación de nuestro ejecutable.

Además es necesario comprender como funciona este proceso a la perfección para evitar dolores de cabeza y sacarle máximo provecho. Al respecto ya vimos en la Guia001, que *gcc* arma una lista de opciones para invocar al linker (que puede ser *ld* o *collect2* cualquiera de estas herramientas funciona perfectamente a los efectos de linker). Cuando queremos enviar en la línea de gcc opciones que son para el linker éstas van al final. Sino sencillamente *gcc* (otro miembro honorario del clan de los utilitarios sensibles) se ofuscará y enviará al linker las cosas que éste no necesita o no comprende con los consiguientes dolores de cabeza para nosotros, los humildes mortales.

En nuestro caso *-L* le indica al linker en donde debe buscar la librería. En

nuestro caso lleva '.' a continuación ya que la librería hola, está en el mismo directorio en el que se invoca a *gcc*. Por su parte *-l*, indica que debe incluirse en la compilación la librería cuyo nombre se indica a continuación, en nuestro caso, *hola*. El linker buscará *libhola.a*, en el directorio de trabajo y sacará de allí los programas objeto que incluirá en el ejecutable.

Tal vez el lector se esté preguntando "¿Y porque cuando uso *printf* no le tengo que decir al *gcc* que librería utilizar y donde se encuentran?". Buena pregunta. Para responderla debemos recordar que el lenguaje C trabaja con un juego de librerías denominadas habitualmente librerías estándar de C. Estas ya le son conocidas, y para buscarlas existe en el sistema una variable de entorno que los programas que anexan librerías al entorno C al momento de su instalación configuran adecuadamente.

Vamos a imprimirla para ver su valor

```
alejandro@notebook:~/InfoI/second$echo $LD_LIBRARY_PATH
/lib:/usr/lib
alejandro@notebook:~/InfoI/second$
```

No es raro. Queda a cargo del lector echar un vistazo a los directorios contenidos por la variable *LD_LIBRARY_PATH*, y ver los archivos allí contenidos para entender que buscando en ese par de directorios el linker encontrará lo necesario para resolver llamadas a las librerías estándar.

Listemos el directorio para observar los tamaños de los diferentes archivos y de paso ejecutemos nuestro programa *demo*, linkeado estáticamente a la librería *hola*.

```
alejandro@notebook:~/InfoI/second$ ls -las
total 40
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 16 18:02 .
4 drwxr-xr-x 6 alejandro alejandro 4096 feb 14 12:23 ..
8 -rwxr-xr-x 1 alejandro alejandro 6367 feb 16 18:02 demo
4 -rw-r--r-- 1 alejandro alejandro   21 feb 16 11:30 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 16 07:58 holalib.h
4 -rw-r--r-- 1 alejandro alejandro  836 feb 16 15:47 holalib.o
4 -rw-r--r-- 1 alejandro alejandro  978 feb 16 15:47 libhola.a
4 -rw-r--r-- 1 alejandro alejandro  121 feb 14 12:32 test_holalib.c
4 -rw-r--r-- 1 alejandro alejandro  772 feb 16 15:47 test_holalib.o
alejandro@notebook:~/InfoI/second$ demo
Hola Mundo!
alejandro@notebook:~/InfoI/second$
```

Observaciones: El archivo *libhola.a* es ligeramente mayor que *holalib.o*. Esto se debe a la información que necesita agregar un archivo de librerías para albergar el encabezamiento y el índice de archivos de programas objeto, las funciones que estos contienen y su offset relativo dentro del archivo librería. De todos modos tengamos en cuenta que esta simple librería contiene por ahora un solo programa objeto, de modo que a medida que se le agreguen mas programas objeto su tamaño será la suma de los tamaños de los programas objeto mas un pequeño plus debido a este encabezado que necesita.

El archivo demo contiene el encabezado *ELF*, y el código que implementa la función *hola*, extraído del archivo l*ibhola.a*.

Finalmente comprobamos que implementa nuestra valiosa función de imprimir "Hola Mundo!" por *stdout*.

Librerías compartidas (Shared)

Estas librerías no se linkean al programa que llama a funciones empaquetadas en ellas, sino que se resuelven las referencias en el momento de arranque del programa en cuestión (load time o tiempo de carga).

Recordemos de la Guía001, que el módulo */lib/ld-linux.so.2* es el dynamic loader de Linux por default que se encarga de cargar un programa ejecutable en la memoria, y que a la luz de estos nuevos conceptos, ahora podemos decir, que es el responsable de resolver en ese momento las referencias del programa en proceso de carga con funciones contenidas en las librerías shared.

Si en el momento de la carga de nuestro programa, las librerías necesarias no están ya cargadas en memoria, el dynamic loader efectuará la carga simultánea a memoria del programa en sí que va a componer el proceso disparado por el usuario, junto con las librerías que necesita. Acto seguido con todos los módulos cargados en memoria, resolverá las referencias, dejando al programa listo para su ejecución.

Una vez instalada la librería en la memoria del sistema, si ejecutamos otro programa que invocará algún(os) programa(s) objeto incluido(s) en la librería, directamente se enlaza con la instancia de la librería residente en memoria para resolver las direcciones a donde efectuar las llamadas desde el programa invocador.

En la implementaciones de Linux tienen algunas prestaciones adicionales que las hacen mas cómodas y atractivas, como por ejemplo, actualizar la librería y seguir soportando a los programas que usan versiones mas viejas de éstas.

Sin embargo al ser tan versátiles nos hacen "pagar" sus ventajas con algunas complejidades. Es así. Bienvenidos a la Ingeniería.....

Este apartado trata acerca de esas complejidades. Manos a la obra.

Las librerías compartidas tienen un nombre al que se denomina "*soname*" que en general difiere del nombre del archivo, que la contiene. Este nombre se asigna mediante el parámetro *-soname* que se pasa al linker *ld* en el momento de su construcción. Este nombre es parte del header *ELF* del archivo. Lo podemos comprobar con el conocido comando *objdump*, usando la opción -p para que presente información privada o adicional del archivo. En algunos casos no hay nada mas que presentar y *-p* no agrega información al listado, en otros casos si la hay.

```
alejandro@notebook:/usr/lib$ objdump -p libhighgui.so.1 | grep SONAME
  SONAME      libhighgui.so.1
alejandro@notebook:/usr/lib$ objdump -p libhighgui.so.1.0.0 | grep SONAME
  SONAME      libhighgui.so.1
alejandro@notebook:/usr/lib$
```

Como vemos el archivo *libhighgui.so.1* tiene en este caso un *soname* igual a su nombre de archivo, pero el *libhighgui.so.1.0.0* tiene el mismo *soname* que la anterior, que obviamente difiere de su nombre de archivo.

El *soname* en general lleva el prefijo "*lib*" seguido del nombre de la librería y un sufijo compuesto por la string "*so*" seguida de un '.' y un número de versión. Este número se incrementa cuando cambia la librería.

Como toda regla existe alguna excepción. En general las librerías de mas bajo nivel del C vienen sin el prefijo "*lib*". Sin embargo es de práctica que el prefijo del *soname*, sea el nombre del directorio en el que irá a parar la librería una vez instalada en el *file system*.

En general el nombre del archivo (también llamado *real name*) se compone del *soname*, seguida de '.', un número de versión, y opcionalmente '.' y un número de *release*.

El *soname* es el nombre que utiliza el linker. Si no tiene incluido el número de versión, se lo incluye.

La clave del manejo de estas librerías está en tener separados estos dos nombres: los programas cuando listan internamente las librerías que necesitan, listan directamente los *sonames* de cada una.

En cambio cuando desarrollamos una librería solo nos remitimos a crear un archivo con el nombre de acuerdo a las reglas establecidas y a lo sumo incluir el número de versión. Nada mas. Al instalarla, debemos ejecutar el programa */etc/ldconfig*, que se encarga de crear el *soname*, escribirlo en el encabezado, y setear el cahe */etc/ld.so.cache*.

Pero ¿como se enlaza el *soname* con el nombre de la librería?

El programa */etc/ldconfig*, se encarga de crear los enlaces simbólicos correspondientes.

```
alejandro@notebook:/usr/lib$ ls -las libhighgui*
260 -rw-r--r-- 1 root root 258414 sep 12  2008 libhighgui.a
  4 -rw-r--r-- 1 root root   1536 sep 12  2008 libhighgui.la
  0 lrwxrwxrwx 1 root root     19 jun 18  2009 libhighgui.so ->
libhighgui.so.1.0.0
  0 lrwxrwxrwx 1 root root     19 jun 18  2009 libhighgui.so.1 ->
libhighgui.so.1.0.0
172 -rw-r--r--1 root root 170024 sep 12  2008 libhighgui.so.1.0.0
```

Vemos que el archivo de la librería contiene el *soname* el número de versión y el número menor (*release*). Luego se crean los *soft links* en el momento de instalar la librería.

Por lo general las librerías se instalan en */lib* o */usr/lib*. Si no son parte del sistema pueden ir en */usr/local/lib*.

Cada vez que se carga un programa el módulo *loader* del sistema operativo (en general */lib/ld-linux.so.[versión]*) en nuestro caso:

```
alejandro@notebook:/usr/lib$ ls -las /lib/ld-linux.so.2
0 lrwxrwxrwx 1 root root 9 ene 23 2009 /lib/ld-linux.so.2 ->
ld-2.7.so
alejandro@notebook:/usr/lib$
```

Baja desde disco a memoria el contenido del archivo binario ejecutable interpretando el formato *ELF* y demás delicias ya conocidas.

Una vez hecho esto, se encarga de buscar las librerías compartidas invocadas en el programa (si las hubiera), y cargar en memoria aquellas que aún no estuviesen presentes, para finalmente resolver en forma dinámica (es decir en tiempo de carga para su ejecución del programa) las referencias hechas a las librerías.

Para saber en que directorios buscar las librerías, lee el siguiente archivo:

```
alejandro@notebook:/usr/lib$ cat /etc/ld.so.conf
include /etc/ld.so.conf.d/*.conf
/usr/local/cuda/lib
alejandro@notebook:/usr/lib$
```

Aquí encuentra un *path* concreto y un *include donde* se especifica leer todos los archivos terminados en .”*conf*” del directorio */etc/ld.so.conf.d/*.

```
alejandro@notebook:/usr/lib$ ls -las /etc/ld.so.conf.d/
total 28
```

```
 4 drwxr-xr-x   2 root root  4096 ene 23  2009 .
12 drwxr-xr-x 137 root root 12288 feb 18 12:59 ..
 4 -rw-r--r--    1 root root    64 oct 13  2008 i486-linux-gnu.conf
 4 -rw-r--r--   1 root root    44 oct 13  2008 libc.conf
alejandro@notebook:/usr/lib$ cat /etc/ld.so.conf.d/i486-linux-gnu.conf
# Multiarch support
/lib/i486-linux-gnu
/usr/lib/i486-linux-gnu
alejandro@notebook:/usr/lib$ cat /etc/ld.so.conf.d/libc.conf
# libc default configuration
/usr/local/lib
alejandro@notebook:/usr/lib$
```

Si al menos una de las librerías incluidas en el código del programa no se encuentran en estos *paths* no se carga del programa y se presenta el correspondiente mensaje por *stderr*.

Queda por cuenta del lector el comando *cat /etc/ld.so.cache*.

El contenido de este archivo depende de la cantidad de librerías compartidas cargadas en memoria. De este modo se evita recorrer todos los directorios en busca de las librerías cargadas en memoria.

Al tratar con librerías estáticas hemos listado el contenido de la variable de entorno *LD_LIBRARY_PATH.*

Esta variable, además de para saber en donde buscar librerías, sirve para probar una nueva versión agregando allí el *path* para que sea nuestra versión de prueba la que primero se cargue. Una vez finalizada la prueba podemos limpiar la variable a su estado original e instalar la librería o corregirla de acuerdo con el estado de las pruebas efectuadas.

La variable *LD_PRELOAD* es específicamente para ello, aunque no es una variable estándar.

Para compilar la librería compartida, procedemos del siguiente modo.

```
alejandro@notebook:~/InfoI/second$ gcc -fPIC -g -c -Wall holalib.c
alejandro@notebook:~/InfoI/shared$
```

Con el comando *gcc* compilamos el fuente de la librería. La opción *fPIC* indica al compilador que genere código independiente de la posición que ocupe en memoria.

Este concepto es central ya que si esa opción no se incluye el código no va a ser apto para ser enlazado dinámicamente con los programas que puedan requerir sus servicios, y pueden además existir límites en la tabla global de offsets que puede ocupar ese código dentro de una de las secciones del programa. Requiere cierto soporte a nivel de hardware que convierte a este tipo de código en procesador dependiente, pudiendo por lo tanto no ser factible bajo ciertas arquitecturas. La opción *-g* habilita la inclusión de toda la información simbólica para soporte al *debuging*. Es recomendable su inclusión. Ahora, con este comando solo hemos generado el programa objeto:

```
alejandro@notebook:~/InfoI/shared$ ls -las
total 32
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 19 02:22 .
4 drwxr-xr-x 3 alejandro alejandro 4096 feb 19 02:14 ..
4 -rw-r--r-- 1 alejandro alejandro  210 feb 19 02:14 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 19 02:14 holalib.h
4 -rw-r--r-- 1 alejandro alejandro 2744 feb 19 02:16 holalib.o
4 -rw-r--r-- 1 alejandro alejandro  121 feb 19 02:14 test_holalib.c
alejandro@notebook:~/InfoI/shared$
```

Resta generar la librería. Esta vez no recurrimos a un archivador sino al propio linker. Claro, como ya hemos visto con anterioridad a esta guía el tratamiento con el *ld* conviene dejarlo en manos del *gcc*. Sin embargo los parámetros que le vamos a pasar son todos para el linker. El *gcc* armará los argumentos de linkeo e invocará al linker. Recordemos que siempre podemos usar la opción *verbose* (*-v*) del *gcc* para satisfacer nuestra curiosidad de conocer los argumentos que *gcc* le arma a *ld*.

```
alejandro@notebook:~/InfoI/shared$  gcc -shared -Wl,-soname,libhola.so.1
-o libhola.so.1.0.1 holalib.o -lc
alejandro@notebook:~/InfoI/shared$ ls -las
total 32
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 19 02:22 .
4 drwxr-xr-x 3 alejandro alejandro 4096 feb 19 02:14 ..
4 -rw-r--r-- 1 alejandro alejandro  210 feb 19 02:14 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 19 02:14 holalib.h
4 -rw-r--r-- 1 alejandro alejandro 2744 feb 19 02:16 holalib.o
8 -rw-r--r-- 1 alejandro alejandro 6074 feb 19 02:22 libhola.so.1.0.1
4 -rw-r--r-- 1 alejandro alejandro  121 feb 19 02:14 test_holalib.
alejandro@notebook:~/InfoI/shared$
```

El argumento *-shared* es de tan obvia explicación que huelgan las palabras. *-Wl* indica que lo que sigue es pasado directamente al linker. Si lo que sigue a *-Wl* contiene una o mas comas, se trata de argumentos que deberán ser enviados al linker uno a uno.

En nuestro caso *-soname* y *libhola.so.1* son esos argumentos, que indican entre ambos cual es el *soname* de nuestra librería. A continuación se indican el nombre del archivo en el que se empaquetarán el(los) programa(s) objeto y la lista de archivos objeto (terminados en "*.o*") que se incluirá en el archivo de librería.

Allí está nuestro archivo de librería compartida. Note el lector los permisos que le dio el compilador por default.

Ahora llega el momento de ponerla disponible para los programas de aplicación que la quieran utilizar. Para ello vamos a ejecutar los pasos explicados con anterioridad:

```
alejandro@notebook:~/InfoI/shared$ sudo ldconfig -n .
alejandro@notebook:~/InfoI/shared$ ls -las
total 32
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 19 17:05 .
4 drwxr-xr-x 3 alejandro alejandro 4096 feb 19 02:14 ..
4 -rw-r--r-- 1 alejandro alejandro  210 feb 19 02:14 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 19 02:14 holalib.h
4 -rw-r--r-- 1 alejandro alejandro 2744 feb 19 02:16 holalib.o
0 lrwxrwxrwx 1 root      root        16 feb 19 17:05 libhola.so.1 ->
libhola.so.1.0.1
8  -rw-r--r-- 1 alejandro alejandro 6074 feb 19 02:22 libhola.so.1.0.1
4  -rw-r--r-- 1 alejandro alejandro  121 feb 19 02:14 test_holalib.c
alejandro@notebook:~/InfoI/shared$
```

Como vemos el comando *ldconfig* debió ejecutarse como superusuario. ¿Porque?. Es bastante obvio. Publicar una librería compartida es una operación algo delicada y requiere que la realice solo quien administra el equipo ya que puede afectar al resto de los usuarios[1].

Luego del comando el único rastro visible es el *soft link* creado por el comando. Pero recordemos que además trabaja en el encabezado del archivo de la librería registrando allí el *soname*, entre otras cosas

Continuemos con el proceso de generación de nuestra librería compartida.

1 Si bien a esta altura puede resultar obvio, es conveniente recordar al lector que los sistemas UNIX han sido concebidos para que en un mismo computador trabajen muchos usuarios a la vez. Sus descendientes como Linux mantienen este principio, raro de entender para quienes hicieron sus primeros pasos con un computador que lleva instalado Windows.

```
alejandro@notebook:~/InfoI/shared$ ln -sf libhola.so.1 libhola.so
alejandro@notebook:~/InfoI/shared$ ls -las
total 32
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 19 17:06 .
4 drwxr-xr-x 3 alejandro alejandro 4096 feb 19 02:14 ..
4 -rw-r--r-- 1 alejandro alejandro  210 feb 19 02:14 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 19 02:14 holalib.h
4 -rw-r--r-- 1 alejandro alejandro 2744 feb 19 02:16 holalib.o
0 lrwxrwxrwx 1 alejandro alejandro   12 feb 19 17:06 libhola.so ->
libhola.so.1
0 lrwxrwxrwx 1 root      root        16 feb 19 17:05 libhola.so.1 ->
libhola.so.1.0.1
8 -rwxr-xr-x 1 alejandro alejandro 6074 feb 19 02:22 libhola.so.1.0.1
4 -rw-r--r-- 1 alejandro alejandro  121 feb 19 02:14 test_holalib.c
alejandro@notebook:~/InfoI/shared$
```

Creamos un *soft link* adicional con el *soname* sin la versión de modo que se pueda invocar en forma genérica en las líneas de compilación independientemente de la versión instalada en el sistema.

Para poder probar nuestra librería vamos a compilar el programa de test.

```
alejandro@notebook:~/InfoI/shared$ gcc -Wall -c -o test_holalib.o
test_holalib.c
alejandro@notebook:~/InfoI/shared$ gcc -o demo test_holalib.o -L. -lhola
alejandro@notebook:~/InfoI/shared$ ls -las
total 44
4 drwxr-xr-x 2 alejandro alejandro 4096 feb 19 17:08 .
4 drwxr-xr-x 3 alejandro alejandro 4096 feb 19 02:14 ..
8 -rwxr-xr-x 1 alejandro alejandro 6485 feb 19 17:08 demo
4 -rw-r--r-- 1 alejandro alejandro  210 feb 19 02:14 holalib.c
4 -rw-r--r-- 1 alejandro alejandro  106 feb 19 02:14 holalib.h
4 -rw-r--r-- 1 alejandro alejandro 2744 feb 19 02:16 holalib.o
0 lrwxrwxrwx 1 alejandro alejandro   12 feb 19 17:06 libhola.so ->
libhola.so.1
0 lrwxrwxrwx 1 root      root        16 feb 19 17:05 libhola.so.1 ->
libhola.so.1.0.1
8 -rwxr-xr-x 1 alejandro alejandro 6074 feb 19 02:22 libhola.so.1.0.1
```

```
4 -rw-r--r-- 1 alejandro alejandro  121 feb 19 02:14 test_holalib.c
4 -rw-r--r-- 1 alejandro alejandro  772 feb 19 17:07 test_holalib.o
alejandro@notebook:~/InfoI/shared$
```

La primer línea de comando generan el programa objeto a partir del programa fuente de prueba y la segunda lo linkea con la librería.

Si las comparamos con las utilizadas para idénticos fines cuando utilizamos librerías estáticas, vemos que, como no puede ser de otro modo, no existe ni un punto ni una coma de diferencia. Este concepto es básico. Una vez generada la librería la forma de compilar aplicaciones y linkearlas a la librería es independiente del tipo de librería utilizada.

Con ejemplos tan triviales no alcanzamos a observar una diferencia notable en el tamaño del archivo binario *demo*, que es en definitiva nuestra aplicación. Al usar librerías tan pequeñas no es notoria la diferencia pero claramente al utilizar una librería estática el código se incrusta en la aplicación y en las compartidas se enlaza en tiempo de ejecución. Trabajando con programas y librerías reales cuyos códigos son menos triviales que los que estamos trabajando aquí, la diferencia de tamaños de las dos versiones de *demo*, sería notoria.

¿Ejecutamos nuestra aplicación?

```
alejandro@notebook:~/InfoI/shared$ demo
demo: error while loading shared libraries: libhola.so.1: cannot
open shared object file: No such file or directory
alejandro@notebook:~/InfoI/shared$
```

¿Que es lo que ocurrió?.

Olvidamos poner nuestra librería disponible para el sistema. Y además como nunca se utilizó hasta ahora, tampoco está en el cache.

```
alejandro@notebook:~/InfoI/shared$ echo $LD_LIBRARY_PATH
/lib:/usr/lib
alejandro@notebook:~/InfoI/shared$
```

Los *paths* en los que el sistema buscará las librerías que utiliza el programa en el momento de su carga en memoria para ejecución son los contenidos en la variable de entorno *LD_LIBRARY_PATH*, que no tiene registrado nuestro directorio de trabajo, que es en donde está físicamente nuestra librería.

Solucionamos este pequeño detalle y....

```
alejandro@notebook:~/InfoI/shared$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
alejandro@notebook:~/InfoI/shared$ echo $LD_LIBRARY_PATH
```

```
/lib:/usr/lib:.
alejandro@notebook:~/InfoI/shared$ demo
Hola Mundo!
alejandro@notebook:~/InfoI/shared$
```

Voila!. Ahí está funcionando.

La forma real y profesional es copiar nuestra librería y los *soft links* al directorio */usr/lib*. Puede ser al */lib* pero no es lo mas conveniente, desde el punto de vista de lo que contiene cada directorio.

Librerías de Carga Dinámica (DL por Dynamic Loaded)

Estas librerías se cargan en momentos diferentes de la carga y ejecución del programa. Su principal utilidad es la implementación de módulos, o *plug-ins*, ya que estos elementos de software se cargan cuando se invocan durante la ejecución de un programa.

Desde el punto de vista de su formato no tienen diferencias en Linux con respecto a como se construyen librerías compartidas o programas objeto.

Sin embargo hay diferencias en el código que se necesita escribir en la aplicación para trabajar con estas librerías.

Es decir que el programador de la aplicación debe incluir funciones específicas que hasta ahora en los modelos analizados no se requieren.

Básicamente necesitamos invocar cuatro funciones:

dlopen ()

```
void * dlopen(const char *filename, int flag);
```

Carga una librería y la prepara para su uso. Devuelve un identificador para esa librería que se utiliza en el resto del programa para las referencias posteriores.

El primer argumento es una string con el nombre del archivo (o del *soft link*). Si proveemos el *path* absoluto (es decir desde */* en adelante), la función busca exactamente ese archivo. Si se provee solo el nombre del archivo, lo buscará en: los directorios contenidos en la variable de entorno *LD_LIBRARY_PATH*, sino la encuentra busca en */etc/ld.so.cache*, si no está busca en */lib*, y sino en */ usr/lib*.

El segundo argumento *flag*, toma los siguientes valores:

RTLD_LAZY: para resolver los símbolos no definidos como código de la librería dinámica cuando se está ejecutando.

RTLD_NOW: para resolver los símbolos no definidos antes de que *dlopen()* retorne y generar un error si no es posible resolverlos.

RTLD_GLOBAL: puede componerse mediante un operador OR con los otros valores y hace que los símbolos externos contenidos en la librería estén disponibles para las librerías cargadas consecuentemente.

dlerror ()

Retorna un string que describe el error generado por las demás funciones de manejo de librerías dinámicas.

dlsym ()

```
void * dlsym(void *handle, char *symbol);
```

Busca el valor de un símbolo presente en una librería ya abierta con *dlopen ()*. Devuelve un puntero a la función o al elemento direccionado.

Los argumentos son el identificador de la librería devuelto por *dlopen()*, y un string con el nombre del símbolo o función cuyo puntero se desea conseguir.

dlclose ()

Cierra la librería abierta con *dlopen()* permitiendo liberar el descriptor o *handle* que se tenía hasta el momento para identificar a nuestra librería dinámica.

```
int dlclose(void *handle);
```

La aplicación

Para implementar una librería dinámica utilizamos los mismos archivos de una librería compartida. Sin cambios de ninguna índole en su procedimiento de generación, ni en su código fuente.

Lo que sí cambia es el programa invocador a las funciones de las librerías ya que para estas se ha definido un grupo de funciones específicas para el acceso.

El programa siguiente:

```
/* Programa de prueba para nuestra librería holalib
*/


# include "holalib.h"
# include <dlfcn.h>
# include <stdlib.h>


int     main (int argc, char **argv)
```

```
{
        void *handle;
        void (*hello) (void);
        char *error;
/*Se abre la librería y se obtiene en handle un puntero a la
instancia de la misma */
        handle = dlopen("../shared/libhola.so.1",RTLD_LAZY);
/*Si handle es NULL, entonces se imprime el mensaje devuelto por
dlerror en stderr*/
        if (!handle) {
                fprintf (stderr, "%s\n", dlerror());
                exit(EXIT_FAILURE);
        }
//En hello se guarda el puntero al label llamado hola en la
librería.
        hello = dlsym (handle, "hola");
        if ((error = dlerror()) != NULL) {
                fputs(error, stderr);
                 exit(EXIT_FAILURE);
        }
// Se invoca la función por medio de su puntero. Aquí es donde
aparece el mensaje característico de nuestra librería.
        (*hello) ();
        dlclose(handle);
        return 0;
}
```

En principio vemos que el programa de uso de la librería ya es mas complejo por mas simple que sea nuestra librería dinámica. Prueba de ello es que se necesite incluir un *header* específico contrariamente a lo que sucede con los restantes ejemplos: # include <dlfcn.h>

Los comentarios del fuente orientan bastante sobre el uso de la librería.

Se accede a la función hola por medio de un puntero a la misma que fue devuelto de *dlsym*, en nuestro caso definido como *hello*.

Para compilar se puede resolver en una sola línea ya que la librería es la misma que desarrollamos como compartida. La variación en este caso es la forma de linkeo contra ella y la forma de acceso por parte del código de la

aplicación.

```
alejandro@notebook:~/InfoI/dynamic$ gcc -Wall -o demo test_holalib.c -ldl
alejandro@notebook:~/InfoI/dynamic$ demo
Hola Mundo!
alejandro@notebook:~/InfoI/dynamic$
```

Como vemos la compilación del programa usa una opción para el linker *-ldl*, indicando el carácter de librería *DL*, es decir de carga dinámica para el tratamiento del archivo *libhola.so.1* al momento de la carga y ejecución de nuestro programa de prueba.

Conclusiones

Hemos podido mediante un ejemplo sumamente trivial desarrollar las librerías de uso estándar en Linux, y observar sus características y ventajas comparativas entre los tres tipos posibles.

Las ideas fuerza que deben quedar tienen que ver con la importancia de tener "paquetes" de código escrito de tal forma que pueda ser reusado por las aplicaciones que desarrollemos. Esto aumenta nuestra productividad y hace que nuestra programación esté mas libre de errores.