

Bibliotecas de código

Alejandro Furfaro

Setiembre 2017



- 1 ¿Que es una Biblioteca?.
- 2 Clasificación.
- 3 Desarrollo de una Biblioteca Estática.
- 4 Desarrollo de una Biblioteca Compartida
- 5 Desarrollo de una Biblioteca de carga Dinámica
- 6 Conclusiones

Definición

Una biblioteca de código no es otra cosa que un archivo que contiene código y datos compilados y funcionales, que podrán ser incorporados a otros programas.

En otras palabras es una colección de programas objeto, (programas fuente pasados solamente por la fase de compilación), que ponen disponible funciones a ser invocadas desde programas.

Ventajas en su uso

- Facilitan la reutilización de código evitando tener que reescribirlo cada vez que lo necesitamos.
- Fomentan el trabajo en equipo. Nos abstraemos de la implementación y nos enfocamos en su uso.
- El código de una Biblioteca es de uso general. Ejemplo: la función `printf`, nos permite acceder a la pantalla para imprimir en ella con el formato que necesitemos.
- No tenemos que preocuparnos ni siquiera de conocer el código de fuente de `printf`. Solo utilizamos el objeto compilado.

Ventajas en su uso

- Facilitan la reutilización de código evitando tener que reescribirlo cada vez que lo necesitamos.
- Fomentan el trabajo en equipo. Nos abstraemos de la implementación y nos enfocamos en su uso.
- El código de una Biblioteca es de uso general. Ejemplo: la función `printf`, nos permite acceder a la pantalla para imprimir en ella con el formato que necesitemos.
- No tenemos que preocuparnos ni siquiera de conocer el código de fuente de `printf`. Solo utilizamos el objeto compilado.

Ventajas en su uso

- Facilitan la reutilización de código evitando tener que reescribirlo cada vez que lo necesitamos.
- Fomentan el trabajo en equipo. Nos abstraemos de la implementación y nos enfocamos en su uso.
- El código de una Biblioteca es de uso general. Ejemplo: la función `printf`, nos permite acceder a la pantalla para imprimir en ella con el formato que necesitemos.
- No tenemos que preocuparnos ni siquiera de conocer el código de fuente de `printf`. Solo utilizamos el objeto compilado.

Ventajas en su uso

- Facilitan la reutilización de código evitando tener que reescribirlo cada vez que lo necesitamos.
- Fomentan el trabajo en equipo. Nos abstraemos de la implementación y nos enfocamos en su uso.
- El código de una Biblioteca es de uso general. Ejemplo: la función `printf`, nos permite acceder a la pantalla para imprimir en ella con el formato que necesitemos.
- No tenemos que preocuparnos ni siquiera de conocer el código de fuente de `printf`. Solo utilizamos el objeto compilado.

Ventajas en su uso

- Facilitan la reutilización de código evitando tener que reescribirlo cada vez que lo necesitamos.
- Fomentan el trabajo en equipo. Nos abstraemos de la implementación y nos enfocamos en su uso.
- El código de una Biblioteca es de uso general. Ejemplo: la función `printf`, nos permite acceder a la pantalla para imprimir en ella con el formato que necesitemos.
- No tenemos que preocuparnos ni siquiera de conocer el código de fuente de `printf`. Solo utilizamos el objeto compilado.

¿Que necesitamos en el sistema para usar `printf`?

- El archivo header (terminado en `.h`) que contiene el prototipo de la función, en este caso `printf`, y que deberemos incluir en el encabezamiento de nuestro programa para que el compilador sepa interpretar las líneas de nuestro programa que invocan a la función `printf` y verificar que no existen errores de sintaxis en las mismas.
- Debe estar instalado en nuestro sistema el archivo de la biblioteca que contiene el código objeto de `printf` (entre otras funciones contenidas en la misma biblioteca).
- Un modo de indicarle al linker la ubicación en el file system del archivo de la biblioteca, para que pueda resolver las llamadas que hacemos desde nuestro código a las funciones de código empaquetadas en dicha biblioteca.

¿Que necesitamos en el sistema para usar `printf`?

- El archivo header (terminado en `.h`) que contiene el prototipo de la función, en este caso `printf`, y que deberemos incluir en el encabezamiento de nuestro programa para que el compilador sepa interpretar las líneas de nuestro programa que invocan a la función `printf` y verificar que no existen errores de sintaxis en las mismas.
- Debe estar instalado en nuestro sistema el archivo de la biblioteca que contiene el código objeto de `printf` (entre otras funciones contenidas en la misma biblioteca).
- Un modo de indicarle al linker la ubicación en el file system del archivo de la biblioteca, para que pueda resolver las llamadas que hacemos desde nuestro código a las funciones de código empaquetadas en dicha biblioteca.

¿Que necesitamos en el sistema para usar `printf`?

- El archivo header (terminado en `.h`) que contiene el prototipo de la función, en este caso `printf`, y que deberemos incluir en el encabezamiento de nuestro programa para que el compilador sepa interpretar las líneas de nuestro programa que invocan a la función `printf` y verificar que no existen errores de sintaxis en las mismas.
- Debe estar instalado en nuestro sistema el archivo de la biblioteca que contiene el código objeto de `printf` (entre otras funciones contenidas en la misma biblioteca).
- Un modo de indicarle al linker la ubicación en el file system del archivo de la biblioteca, para que pueda resolver las llamadas que hacemos desde nuestro código a las funciones de código empaquetadas en dicha biblioteca.

¿Que necesitamos en el sistema para usar `printf`?

- El archivo header (terminado en `.h`) que contiene el prototipo de la función, en este caso `printf`, y que deberemos incluir en el encabezamiento de nuestro programa para que el compilador sepa interpretar las líneas de nuestro programa que invocan a la función `printf` y verificar que no existen errores de sintaxis en las mismas.
- Debe estar instalado en nuestro sistema el archivo de la biblioteca que contiene el código objeto de `printf` (entre otras funciones contenidas en la misma biblioteca).
- Un modo de indicarle al linker la ubicación en el file system del archivo de la biblioteca, para que pueda resolver las llamadas que hacemos desde nuestro código a las funciones de código empaquetadas en dicha biblioteca.

Desarrollando nuestra propia Biblioteca

Para desarrollar una Biblioteca necesitamos dos componentes

- Los prototipos de las funciones que tiene incorporadas, que normalmente están en los headers (archivos terminados en .h) que siempre acompañan a la biblioteca, ya que de otro modo no es posible para el compilador resolver la sintaxis de cada llamada a cada función de la biblioteca.
- Código fuente de las funciones, (archivos terminados en .c) que normalmente no acompañan a las bibliotecas, al menos no necesariamente salvo que hayan sido desarrolladas bajo General Public License (GPL).

Desarrollando nuestra propia Biblioteca

Para desarrollar una Biblioteca necesitamos dos componentes

- Los prototipos de las funciones que tiene incorporadas, que normalmente están en los headers (archivos terminados en .h) que siempre acompañan a la biblioteca, ya que de otro modo no es posible para el compilador resolver la sintaxis de cada llamada a cada función de la biblioteca.
- Código fuente de las funciones, (archivos terminados en .c) que normalmente no acompañan a las bibliotecas, al menos no necesariamente salvo que hayan sido desarrolladas bajo General Public License (GPL).

Desarrollando nuestra propia Biblioteca

Para desarrollar una Biblioteca necesitamos dos componentes

- Los prototipos de las funciones que tiene incorporadas, que normalmente están en los headers (archivos terminados en .h) que siempre acompañan a la biblioteca, ya que de otro modo no es posible para el compilador resolver la sintaxis de cada llamada a cada función de la biblioteca.
- Código fuente de las funciones, (archivos terminados en .c) que normalmente no acompañan a las bibliotecas, al menos no necesariamente salvo que hayan sido desarrolladas bajo General Public License (GPL).

Existen tres tipos de Bibliotecas en Linux

- 1 Estáticas
- 2 Compartidas (Shared)
- 3 De carga dinámica (DL, del inglés Dynamic Loaded)

Bibliotecas Estáticas, generalidades

Definición

- 1 Una biblioteca estática no es otra cosa que una simple **colección de programas objeto** agrupados en un único archivo cuyo nombre típicamente finaliza en '.a'.
- 2 Al compilar un programa que utiliza código contenido en una biblioteca estática, en el momento de realizar la fase de enlace (link), se copia en nuestro programa el código objeto de la biblioteca.
- 3 Es decir que no es necesario distribuir nuestro programa con la Biblioteca ya que el código que utilizamos de ella se encuentra incrustado en nuestro nuevo programa.

Código de una función de biblioteca básica

A lo largo de buena parte de la clase trabajaremos con el mismo código independientemente del tipo de biblioteca que vayamos a desarrollar.

```
1  /* Archivo holalib.h:
2  Header para holalib.c
3  */
4  #ifndef HOLA_H
5  #define HOLA_H
6  #include <stdio.h>
7  void hola (void);
8  #endif
9
10 /* Programa holalib.c
11 Ejemplo para generar una Biblioteca.
12 Función: hola
13 Imprime en stdout Hola Mundo!
14 */
15 #include "holalib.h"
16 void hola (void)
17 {
18     printf ("Hola Mundo!\n");
19 }
```

Observar en holalib.c que no hay función main();

Observaciones a holalib.h

- Incluimos una compilación condicional aprovechando los buenos oficios del preprocesador.

¿Porque? ¿Para que?

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).

Observaciones a holalib.h

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

Código de prueba para nuestra función

Un simple programa para probar nuestro código.

```
1 /* Programa de prueba para nuestra Biblioteca holalib
2 */
3 #include "holalib.h"
4 int main (void)
5 {
6     hola ();
7     return 0;
8 }
```

Como se genera una Biblioteca Estática

- 1 Compilamos nuestro programa original con la opción -c del gcc (**c** = Compile only)

```
alejandro@notebook:~/Infol/second$ gcc -c holalib.c
alejandro@notebook:~/Infol/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
```

- 2 Para crear la Biblioteca estática utilizamos el comando **ar**, que es el empaquetador clásico de Unix. El comando completo es:

ar rcs libhola.a holalib.o

Comando **ar**

- ③ 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 Biblioteca **libhola.a** un elemento llamado **holalib.o**, **ar** lo pisa por este nuevo.
 - **c** indica que cree el archivo de Biblioteca especificado si no existiere.
 - **s** le indica actualizar el índice de la Biblioteca una vez inserto el archivo solicitado.
- ④ Si queremos listar los archivos (objetos) que componen la Biblioteca, el comando es:

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

Compilando el programa que usa la Biblioteca

- Hasta ahora hemos compilado el código de nuestra Biblioteca.
- Las Bibliotecas en general llevan en su nombre el prefijo “lib”(por el término original del idioma inglés que las identifica: library).
- Por lo tanto, a los efectos del linker, el nombre de la Biblioteca es lo que sigue a “lib” y precede a “.a”
- Lo primero es compilar el programa fuente de prueba utilizando la opción -c del gcc (**c** = Compile only)

```
gcc -c -o test_holalib.o test_holalib.c
```


Compilando el programa que usa la Biblioteca

- De modo que para compilar el programa que invoca funciones de nuestra Biblioteca, tenemos que darle cierta información adicional al gcc, para que éste llame al linker (ld o collect2), pasándole la información adecuada.

```
gcc -o demo test_holalib.o -L. -lhola
```

- l, indica que debe incluirse en la compilación la Biblioteca 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.
- L le indica al linker en donde debe buscar la Biblioteca. En nuestro caso lleva '.' a continuación ya que la Biblioteca hola, está en el mismo directorio en el que se invoca a gcc.

dudo, luego aprendo

¿Porque cuando uso printf no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

El lenguaje C trabaja con un juego de Bibliotecas denominadas habitualmente Bibliotecas estándar de C. Estas ya le son conocidas.

dudo, luego aprendo

¿Porque cuando uso printf no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

El lenguaje C trabaja con un juego de Bibliotecas denominadas habitualmente Bibliotecas estándar de C. Estas ya le son conocidas.

dudo, luego aprendo

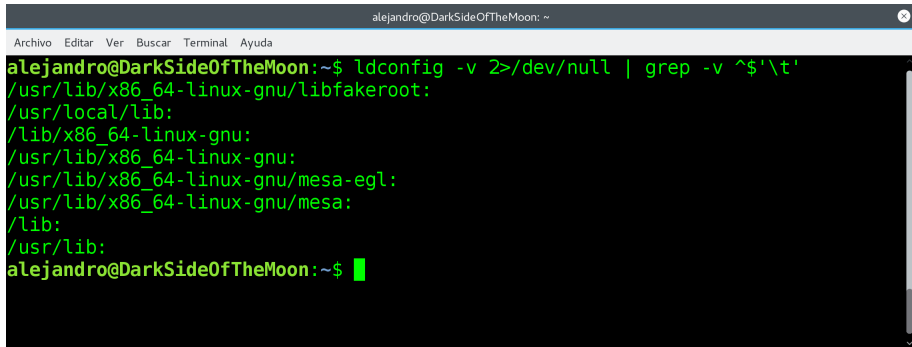
¿Porque cuando uso printf no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

El lenguaje C trabaja con un juego de Bibliotecas denominadas habitualmente Bibliotecas estándar de C. Estas ya le son conocidas.

dudo, luego aprendo

¿Porque cuando uso printf no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

El lenguaje C trabaja con un juego de Bibliotecas denominadas habitualmente Bibliotecas estándar de C. Estas ya le son conocidas.



```
alejandro@DarkSideOfTheMoon: ~  
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  
alejandro@DarkSideOfTheMoon:~$ ldconfig -v 2>/dev/null | grep -v ^$'\t'  
/usr/lib/x86_64-linux-gnu/libfakeroot:  
/usr/local/lib:  
/lib/x86_64-linux-gnu:  
/usr/lib/x86_64-linux-gnu:  
/usr/lib/x86_64-linux-gnu/mesa-egl:  
/usr/lib/x86_64-linux-gnu/mesa:  
/lib:  
/usr/lib:  
alejandro@DarkSideOfTheMoon:~$
```

dudo, luego aprendo

¿Porque cuando uso `printf` no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

Para agregar el resto de las librerías se dispone de una variable de entorno en la cual podemos agregar la ruta absoluta a nuestro archivo de Biblioteca. La variable es `LD_LIBRARY_PATH`, y se maneja con las mismas reglas que la variable de entorno `PATH` : Se incluyen en ella las rutas separadas por el caracter `'.'`. Por default su valor es nulo, pero...

dudo, luego aprendo

¿Porque cuando uso `printf` no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

Para agregar el resto de las librerías se dispone de una variable de entorno en la cual podemos agregar la ruta absoluta a nuestro archivo de Biblioteca. La variable es `LD_LIBRARY_PATH`, y se maneja con las mismas reglas que la variable de entorno `PATH` : Se incluyen en ella las rutas separadas por el caracter `:`. Por default su valor es nulo, pero...

dudo, luego aprendo

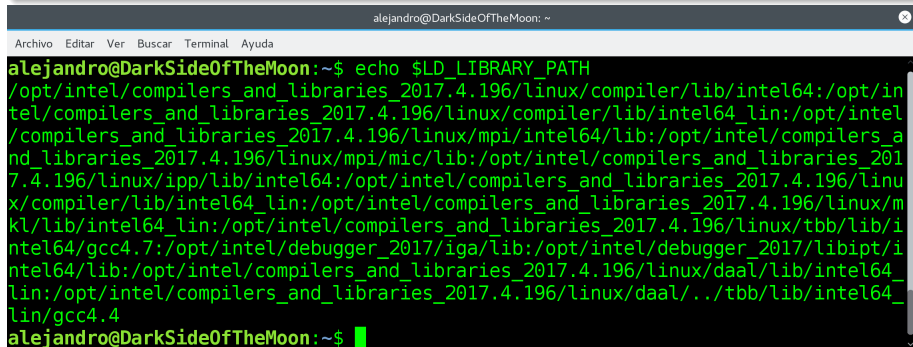
¿Porque cuando uso `printf` no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

Para agregar el resto de las librerías se dispone de una variable de entorno en la cual podemos agregar la ruta absoluta a nuestro archivo de Biblioteca. La variable es `LD_LIBRARY_PATH`, y se maneja con las mismas reglas que la variable de entorno `PATH` : Se incluyen en ella las rutas separadas por el caracter `'.'`. Por default su valor es nulo, pero...

dudo, luego aprendo

¿Porque cuando uso printf no le tengo que decir al gcc que Bibliotecas utilizar y donde se encuentran?

Para agregar el resto de las librerías se dispone de una variable de entorno en la cual podemos agregar la ruta absoluta a nuestro archivo de Biblioteca. La variable es `LD_LIBRARY_PATH`, y se maneja con las mismas reglas que la variable de entorno `PATH`: Se incluyen en ella las rutas separadas por el caracter `:`. Por default su valor es nulo, pero...



```
alejandro@DarkSideOfTheMoon: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
alejandro@DarkSideOfTheMoon:~$ echo $LD_LIBRARY_PATH  
/opt/intel/compiler_and_libraries_2017.4.196/linux/compiler/lib/intel64:/opt/in  
tel/compiler_and_libraries_2017.4.196/linux/compiler/lib/intel64_lin:/opt/in  
tel/compiler_and_libraries_2017.4.196/linux/mpi/intel64/lib:/opt/intel/compiler_a  
nd_libraries_2017.4.196/linux/mpi/mic/lib:/opt/intel/compiler_and_libraries_201  
7.4.196/linux/ipp/lib/intel64:/opt/intel/compiler_and_libraries_2017.4.196/linu  
x/compiler/lib/intel64_lin:/opt/intel/compiler_and_libraries_2017.4.196/linux/m  
kl/lib/intel64_lin:/opt/intel/compiler_and_libraries_2017.4.196/linux/tbb/lib/i  
ntel64/gcc4.7:/opt/intel/debugger_2017/iga/lib:/opt/intel/debugger_2017/libipt/i  
ntel64/lib:/opt/intel/compiler_and_libraries_2017.4.196/linux/daal/lib/intel64_  
lin:/opt/intel/compiler_and_libraries_2017.4.196/linux/daal/./tbb/lib/intel64_  
lin/gcc4.4  
alejandro@DarkSideOfTheMoon:~$
```

Tareas...

- 1 Ejecutar el programa demo para verificar que imprime finalmente "Hola Mundo!"
- 2 Listar el directorio
- 3 Comparar los tamaños de hollib.a, hollib.o, y demo

Tareas...

- 1 Ejecutar el programa demo para verificar que imprime finalmente "Hola Mundo!"
- 2 Listar el directorio
- 3 Comparar los tamaños de hocalib.a, hocalib.o, y demo

Tareas...

- 1 Ejecutar el programa demo para verificar que imprime finalmente "Hola Mundo!"
- 2 Listar el directorio
- 3 Comparar los tamaños de hocalib.a, hocalib.o, y demo

Tareas...

- 1 Ejecutar el programa demo para verificar que imprime finalmente “Hola Mundo!”
- 2 Listar el directorio
- 3 Comparar los tamaños de holalib.a, holalib.o, y demo

Observaciones

¿Que resultados observamos del slide anterior?

El archivo *libhola.a* es ligeramente mayor que *holalib.o*, debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo *libhola.a*.

Observaciones

¿Que resultados observamos del slide anterior?

El archivo *libhola.a* es ligeramente mayor que *holalib.o*, debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo *libhola.a*.

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Observaciones

¿Que resultados observamos del slide anterior?

El archivo [libhola.a](#) es ligeramente mayor que [holalib.o](#), debido a la información que necesita agregar un archivo de Bibliotecas para construir un encabezamiento con:

- El índice de archivos de programas objeto.
- Los nombres de las funciones contenidas en dichos programas objeto.
- El offset relativo dentro del archivo Biblioteca del inicio de cada función.

Tengamos en cuenta que esta simple Biblioteca 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 por su parte contiene el encabezado ELF, y el código que implementa la función hola, extraído del archivo [libhola.a](#).

Bibliotecas Shared: Generalidades

- 1 Estas Bibliotecas no incorporan el código de sus funciones al programa que las invoca, sino que resuelven las referencias en el momento de arranque del programa en cuestión (load time).
- 2 El módulo `/lib/ld-linux.so.2` es el dynamic loader de Linux, que se encarga de cargar un programa ejecutable en la memoria. Es, entonces, el responsable de resolver en ese momento las referencias a funciones contenidas en las Bibliotecas compartidas, realizadas por el programa en proceso de carga.
- 3 Si en el momento de la carga de nuestro programa no están cargadas en memoria las Bibliotecas necesarias, el dynamic loader cargará simultáneamente a memoria el programa en sí que va a componer el proceso disparado por el usuario, junto con las Bibliotecas que necesita. Una vez cargados en memoria todos los módulos, resolverá las referencias, dejando al programa listo para su ejecución.

Bibliotecas Shared: Generalidades

- 1 Estas Bibliotecas no incorporan el código de sus funciones al programa que las invoca, sino que resuelven las referencias en el momento de arranque del programa en cuestión (load time).
- 2 El módulo `/lib/ld-linux.so.2` es el dynamic loader de Linux, que se encarga de cargar un programa ejecutable en la memoria. Es, entonces, el responsable de resolver en ese momento las referencias a funciones contenidas en las Bibliotecas compartidas, realizadas por el programa en proceso de carga.
- 3 Si en el momento de la carga de nuestro programa no están cargadas en memoria las Bibliotecas necesarias, el dynamic loader cargará simultáneamente a memoria el programa en sí que va a componer el proceso disparado por el usuario, junto con las Bibliotecas que necesita. Una vez cargados en memoria todos los módulos, resolverá las referencias, dejando al programa listo para su ejecución.

Bibliotecas Shared: Generalidades

- 1 Estas Bibliotecas no incorporan el código de sus funciones al programa que las invoca, sino que resuelven las referencias en el momento de arranque del programa en cuestión (load time).
- 2 El módulo `/lib/ld-linux.so.2` es el dynamic loader de Linux, que se encarga de cargar un programa ejecutable en la memoria. Es, entonces, el responsable de resolver en ese momento las referencias a funciones contenidas en las Bibliotecas compartidas, realizadas por el programa en proceso de carga.
- 3 Si en el momento de la carga de nuestro programa no están cargadas en memoria las Bibliotecas necesarias, el dynamic loader cargará simultáneamente a memoria el programa en sí que va a componer el proceso disparado por el usuario, junto con las Bibliotecas que necesita. Una vez cargados en memoria todos los módulos, resolverá las referencias, dejando al programa listo para su ejecución.

Generalidades

- 4 Una vez instalada la Biblioteca en la memoria del sistema, si ejecutamos otro programa que invocará algún(os) programa(s) objeto incluido(s) en la Biblioteca, directamente se enlaza con la instancia de la Biblioteca residente en memoria para resolver las direcciones a donde efectuar las llamadas desde el programa invocador.
- 5 Las Bibliotecas compartidas tienen un nombre al que se denomina “**soname**” que en general difiere del nombre del archivo, que la contiene.
- 6 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.
- 7 Ejercicio: ¿Donde está ubicado **/lib/ld-linux.so.2**? ¿que tipo de archivo es? ¿Cual es su “**soname**”?

Generalidades

- 4 Una vez instalada la Biblioteca en la memoria del sistema, si ejecutamos otro programa que invocará algún(os) programa(s) objeto incluido(s) en la Biblioteca, directamente se enlaza con la instancia de la Biblioteca residente en memoria para resolver las direcciones a donde efectuar las llamadas desde el programa invocador.
- 5 Las Bibliotecas compartidas tienen un nombre al que se denomina “**soname**” que en general difiere del nombre del archivo, que la contiene.
- 6 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.
- 7 Ejercicio: ¿Donde está ubicado **/lib/ld-linux.so.2**? ¿que tipo de archivo es? ¿Cual es su “**soname**”?

Generalidades

- 4 Una vez instalada la Biblioteca en la memoria del sistema, si ejecutamos otro programa que invocará algún(os) programa(s) objeto incluido(s) en la Biblioteca, directamente se enlaza con la instancia de la Biblioteca residente en memoria para resolver las direcciones a donde efectuar las llamadas desde el programa invocador.
- 5 Las Bibliotecas compartidas tienen un nombre al que se denomina “**soname**” que en general difiere del nombre del archivo, que la contiene.
- 6 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.
- 7 Ejercicio: ¿Donde está ubicado **/lib/ld-linux.so.2**? ¿que tipo de archivo es? ¿Cual es su “**soname**”?

Generalidades

- 4 Una vez instalada la Biblioteca en la memoria del sistema, si ejecutamos otro programa que invocará algún(os) programa(s) objeto incluido(s) en la Biblioteca, directamente se enlaza con la instancia de la Biblioteca residente en memoria para resolver las direcciones a donde efectuar las llamadas desde el programa invocador.
- 5 Las Bibliotecas compartidas tienen un nombre al que se denomina “**soname**” que en general difiere del nombre del archivo, que la contiene.
- 6 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.
- 7 Ejercicio: ¿Donde está ubicado **/lib/ld-linux.so.2**? ¿que tipo de archivo es? ¿Cual es su “**soname**”?

Shared Object name: soname

Tarea: Comprobar con `objdump` como diferentes nodos del File system correspondientes a una librería tienen el mismo **soname**.

Tipear en la consola:

```
alejandro@DarkSideOfTheMoon:~$ cd /usr/lib/x86_64-linux-gnu
alejandro@DarkSideOfTheMoon:/usr/lib/x86_64-linux-gnu$ objdump -p libopencv_core.so | grep
SONAME
SONAME                libopencv_core.so.2.4
alejandro@DarkSideOfTheMoon:/usr/lib/x86_64-linux-gnu$ objdump -p libopencv_core.so.2.4 | grep
SONAME
SONAME                libopencv_core.so.2.4
alejandro@DarkSideOfTheMoon:/usr/lib/x86_64-linux-gnu$ objdump -p libopencv_core.so.2.4.9 |
grep SONAME
SONAME                libopencv_core.so.2.4
```

Como vemos el archivo `libopencv_core.so.2.4` tiene en este caso un **soname** igual a su nombre de archivo, pero `libopencv_core.so` y `libopencv_core.so.2.4.9` tienen el mismo **soname**, que obviamente difiere de su nombre de archivo. El **soname** en general lleva el prefijo “lib” seguido del nombre de la Biblioteca y un sufijo compuesto por la string **so** seguida de un ‘.’ y un número de versión. Este número se incrementa cuando cambia la Biblioteca.

Como toda regla existe alguna excepción

En general las Bibliotecas 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 Biblioteca 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.

soname...mas detalles...

- 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 Bibliotecas está en tener separados **soname** y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los **soname** de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa **/etc/ldconfig**, que se encarga de crear el **soname**, escribirlo en el encabezado ELF, y setear el cache **/etc/ld.so.cache**.

soname...mas detalles...

- 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 Bibliotecas está en tener separados **soname** y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los **soname** de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa **/etc/ldconfig**, que se encarga de crear el **soname**, escribirlo en el encabezado ELF, y setear el cache **/etc/ld.so.cache**.

soname...mas detalles...

- 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 Bibliotecas está en tener separados **soname** y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los **soname** de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa **/etc/ldconfig**, que se encarga de crear el **soname**, escribirlo en el encabezado ELF, y setear el cache **/etc/ld.so.cache**.

soname...mas detalles...

- 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 Bibliotecas está en tener separados **soname** y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los **soname** de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa **/etc/ldconfig**, que se encarga de crear el **soname**, escribirlo en el encabezado ELF, y setear el cache **/etc/ld.so.cache**.

soname...mas detalles...

- 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 Bibliotecas está en tener separados `soname` y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los `soname` de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa `/etc/ldconfig`, que se encarga de crear el `soname`, escribirlo en el encabezado ELF, y setear el cache `/etc/ld.so.cache`.

soname...mas detalles...

- 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 Bibliotecas está en tener separados **soname** y nombre de archivo.
- Los programas cuando listan internamente las Bibliotecas que necesitan, listan directamente los **soname** de c/u.
- En cambio cuando desarrollamos una Biblioteca 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 instalar la Biblioteca, debemos ejecutar el programa **/etc/ldconfig**, que se encarga de crear el **soname**, escribirlo en el encabezado ELF, y setear el cache **/etc/ld.so.cache**.

Enlazando el soname con el nombre de la Biblioteca

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

Comprobémoslo con un ejemplo vivo: libhighgui.so

```
alejandro@DarkSideOfTheMoon :/usr/lib/x86_64-linux-gnu$ ls -las  
libopencv_core*  
4224 -rw-r--r-- 1 root root 4322498 abr 18 06:14 libopencv_core.a  
0 lrwxrwxrwx 1 root root 21 abr 18 06:14 libopencv_core.so ->  
libopencv_core.so.2.4  
0 lrwxrwxrwx 1 root root 23 abr 18 06:14 libopencv_core.so.2.4  
-> libopencv_core.so.2.4.9  
2236 -rw-r--r-- 1 root root 2286744 abr 18 06:14 libopencv_core.so  
.2.4.9
```

- Vemos que el archivo de la Biblioteca contiene el soname el número de versión y el número menor (release).
- Los soft links se crean al instalar la Biblioteca.

Para instalar, basta con un copy

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

y el Linux Loader es... ¡una Biblioteca compartida!

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

listando ld-linux

```
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.
- Libera el espacio de memoria que ocupaba el archivo binario.
- Enlaza el programa con las librerías y carga en memoria las librerías.
- Ejecuta el programa.

y el Linux Loader es... ¡una Biblioteca compartida!

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

listando ld-linux

```
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 Bibliotecas 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 Bibliotecas.

y el Linux Loader es... ¡una Biblioteca compartida!

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

listando ld-linux

```
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 Bibliotecas 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 Bibliotecas.

y el Linux Loader es... ¡una Biblioteca compartida!

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

listando ld-linux

```
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 Bibliotecas 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 Bibliotecas.

Ya está el código del programa en memoria RAM

- Para saber en que directorios buscar las Bibliotecas, lee el archivo `/etc/ld.so.conf`:

¿Que hay en `ld.so.conf`?

```
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/`

Ya está el código del programa en memoria RAM

- Para saber en que directorios buscar las Bibliotecas, lee el archivo `/etc/ld.so.conf`:

¿Que hay en `ld.so.conf`?

```
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/`

Ya está el código del programa en memoria RAM

- Para saber en que directorios buscar las Bibliotecas, lee el archivo `/etc/ld.so.conf`:

¿Que hay en `ld.so.conf`?

```
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/`

Directorio /etc/ld.so.conf.d/

En la consola listar el contenido de /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
```

En la consola listar el contenido de cada archivo del directorio

```
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$
```

Directorio /etc/ld.so.conf.d/

En la consola listar el contenido de /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
```

En la consola listar el contenido de cada archivo del directorio

```
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$
```

Directorio /etc/ld.so.conf.d/

En la consola listar el contenido de /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
```

En la consola listar el contenido de cada archivo del directorio

```
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$
```


Todos los archivos de biblioteca deben estar allí

- Si al menos uno de los archivos de las Bibliotecas incluidas en el código del programa no se encuentra en estos paths no se carga el programa y se informa con un mensaje por stderr.
- Tarea: Ejecutar el comando `cat /etc/ld.so.cache`.
- El contenido de este archivo depende de la cantidad de Bibliotecas compartidas cargadas en memoria. De este modo se evita recorrer todos los directorios en busca de las Bibliotecas cargadas en memoria.

Todos los archivos de biblioteca deben estar allí

- Si al menos uno de los archivos de las Bibliotecas incluidas en el código del programa no se encuentra en estos paths no se carga el programa y se informa con un mensaje por stderr.
- Tarea: Ejecutar el comando `cat /etc/ld.so.cache`.
- El contenido de este archivo depende de la cantidad de Bibliotecas compartidas cargadas en memoria. De este modo se evita recorrer todos los directorios en busca de las Bibliotecas cargadas en memoria.

Todos los archivos de biblioteca deben estar allí

- Si al menos uno de los archivos de las Bibliotecas incluidas en el código del programa no se encuentra en estos paths no se carga el programa y se informa con un mensaje por stderr.
- Tarea: Ejecutar el comando `cat /etc/ld.so.cache`.
- El contenido de este archivo depende de la cantidad de Bibliotecas compartidas cargadas en memoria. De este modo se evita recorrer todos los directorios en busca de las Bibliotecas cargadas en memoria.

LD_LIBRARY_PATH, o linkear el ejecutable con -rpath

- Al tratar con Bibliotecas estáticas hemos listado el contenido de la variable de entorno **LD_LIBRARY_PATH**, para saber donde buscar archivos de Bibliotecas.
- En el caso de las Bibliotecas compartidas, 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 Biblioteca 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.
- Otra opción para no trabajar con variables de entorno es enviar al linker la opción **-rpath [dir]**, en la que se le especifica el directorio en el que está la Biblioteca shared. Cuando [/lib/ld-linux.so.2](http://lib/ld-linux.so.2) cargue el ejecutable encontrará en el encabezado ELF la ruta de la Biblioteca.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

A trabajar..

- 1 Para compilar una librería shared, utilizaremos el mismo código fuente que en el caso de las estáticas.
- 2 Los comandos son diferentes.
- 3 Para compilar: `gcc -fPIC -g -c -Wall holalib.c`
- 4 La opción `fPIC` indica al compilador que genere código independiente de la posición que ocupe en memoria.
- 5 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.

Llamamos al gcc pero el trabajo es para el linker

Resta generar la Biblioteca

Esta vez no recurrimos a un archivador sino al propio linker. Como ya hemos visto el conviene dejar la interfaz con el linker 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. La línea de compilación es:

```
gcc -shared -Wl,-soname,libhola.so.1 -o libhola.so.1.0.1 holalib.o -lc
```

Parámetros de la línea de linkeo

- 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 Biblioteca.
- 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 Biblioteca.

Poniendo la Biblioteca disponible

Ejecutar desde la consola:

```

alejandro@notebook:~/Infol/shared$ sudo ldconfig -n .
alejandro@notebook:~/Infol/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

```

- El comando `ldconfig` se ejecuta como superusuario.
- Publicar una Biblioteca compartida es una operación que requiere ser realizada solo por quien administra el equipo ya que puede afectar al resto de los usuarios.
- Luego del comando el único rastro visible es el soft link creado.
- Pero recordemos que además trabaja en el encabezado del archivo de la Biblioteca registrando allí el soname, entre otras cosas.

Continuemos generando nuestra Biblioteca compartida

Ejecutar desde la consola:

```
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
```

Creamos un soft link adicional (libhola.so) con el [soname](#) sin la versión de modo que se pueda invocar en forma genérica en las líneas de compilación

1,2,3,.. probando

- Para compilar el programa de prueba utilizamos las mismas líneas de compilación y linkeo que utilizáramos para probar la librería estática.

Este concepto es básico

Una vez generada la Biblioteca la forma de compilar aplicaciones y linkearlas a la Biblioteca es independiente del tipo de Biblioteca utilizada

- Ponemos nuestra librería disponible para el sistema.

Nunca te olvides de LD_LIBRARY_PATH

```
alejandro@notebook : ~/Info1/shared$ echo $LD_LIBRARY_PATH
/lib : /usr/lib
alejandro@notebook : ~/Info1/shared$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH :.
alejandro@notebook : ~/Info1/shared$ echo $LD_LIBRARY_PATH
/lib : /usr/lib :.
```

1,2,3,.. probando (alternativa 2)

- Si queremos independizarnos de las variables de entorno, como ya se dijo, la opción `-rpath` del linker es la alternativa

```
alejandro@notebook: gcc test_holalib.c -o hola -Ihola -L. -Wl,-  
rpath ,/home/alejandro/work/facu/Info1/Programming/second/  
shared -g -Wall
```

- De este modo el encabezado ELF del ejecutable tiene la ruta en la que se encuentra la Biblioteca shared.
- Por lo tanto cuando [/lib/ld-linux.so.2](#) cargue el ejecutable encontrará en el encabezado ELF la ruta de la Biblioteca, sin necesidad de recurrir a `LD_LIBRARY_PATH`

Bibliotecas de carga dinámica: Conceptos preliminares

- Estas Bibliotecas 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 Bibliotecas compartidas o programas objeto.
- Sin embargo hay diferencias en el código que se necesita escribir en la aplicación para trabajar con estas Bibliotecas.
- Es decir que el programador de la aplicación debe incluir funciones específicas que hasta ahora en los modelos analizados no se requieren.

dlopen ()

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

- Carga una Biblioteca y la prepara para su uso. Devuelve un identificador para esa Biblioteca que se utiliza en el resto del programa para las referencias posteriores.
- **const char *filename** 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`, si no la encuentra busca en `/etc/ld.so.cache`, si no está busca en `/lib`, y sino en `/usr/lib`.

dlopen ()

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

- **int flag** toma los siguientes valores:
 - 1 **RTLD_LAZY**: para resolver los símbolos no definidos como código de la Biblioteca dinámica cuando se está ejecutando.
 - 2 **RTLD_NOW**: para resolver los símbolos no definidos antes de que **dlopen()** retorne y generar un error si no es posible resolverlos.
 - 3 **RTLD_GLOBAL**: puede componerse mediante un operador OR con los otros valores y hace que los símbolos externos contenidos en la Biblioteca estén disponibles para las Bibliotecas cargadas consecuentemente.

dlerror(), dlsym(), y dlclose()

- **dlerror()**

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

- **dlsym()**

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

Busca el valor de un símbolo presente en una Biblioteca ya abierta con **dlopen()**. Devuelve un puntero a la función o al elemento direccionado. Los argumentos son el identificador de la Biblioteca devuelto por **dlopen()**, y un string con el nombre del símbolo o función cuyo puntero se desea conseguir.

- **dlclose()**

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

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

La aplicación

- 1 Para implementar una Biblioteca dinámica utilizamos los mismos archivos de una Biblioteca compartida. Sin cambios de ninguna índole en su procedimiento de generación, ni en su código fuente.
- 2 Lo que sí cambia es el programa invocador a las funciones de las Bibliotecas ya que para estas se ha definido un grupo de funciones específicas para el acceso.

Así queda una aplicación

```
1 /* Programa de prueba para nuestra Biblioteca holaLib */
2 #include "holalib.h"
3 #include <dlfcn.h>
4 #include <stdlib.h>
5 int main (int argc, char **argv)
6 {
7     void *handle;
8     void (*hello) (void);
9     char *error;
10    /*Se abre la Biblioteca y se obtiene en handle un puntero a la instancia de la misma */
11    handle = dlopen("../shared/libhola.so.1", RTLD.LAZY);
12    /*Si handle es NULL, entonces se imprime el mensaje devuelto por dlerror en stderr*/
13    if (!handle) {
14        fprintf(stderr, "%s\n", dlerror());
15        exit(EXIT.FAILURE);
16    }
17    //En hello se guarda el puntero al label llamado hola en la Biblioteca.
18    hello = dlsym(handle, "hola");
19    if ((error = dlerror()) != NULL) {
20        fputs(error, stderr);
21        exit(EXIT.FAILURE);
22    }
23    // Se invoca la función por medio de su puntero.
24    (*hello) ();
25    dlclose(handle);
26    return 0;
27 }
```

Como compilar la aplicación cuando la Biblioteca que usamos es DL

- En este caso la aplicación no se linkea con la Biblioteca de código desarrollada ya que ésta no será cargada por [/lib/ld-linux.so.2](#).
- En lugar de ellos debimos incluir en el código del programa usuario de la Biblioteca funciones de otra librería.
- Es esa librería la que hay que vincular con el Inker al programa: `libdl` en este caso

```
alejandro@notebook: gcc test_holalib.c -o hola -ldl
```

Listo!

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

Las ideas fuerza que deben quedar de esta clase, 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.