≡ Menu

- Home
- Free eBook
- Start Here
- Contact
- About

Intro to Linux Shared Libraries (How to Create Shared Libraries)

by Himanshu Arora on June 11, 2012

Like 16 Tweet

A library is a file containing compiled code from various object files stuffed into a single file. It may contain a group of functions that are used in a particular context. For example, the 'pthread' library is used when thread related functions are to be used in the program.

Broadly, a library (or Program Library) can be of two types :

1. Shared Library
2. Static Library

In this article we will discuss specifically about Shared Libraries.

Shared Libraries

Shared Libraries are the libraries that can be linked to any program at run-time. They provide a means to use code that can be loaded anywhere in the memory. Once loaded, the shared library code can be used by any number of programs. So, this way the size of programs(using shared library) and the memory footprint can be kept low as a lot of code is kept common in form of a shared library.

Shared libraries provide modularity to the development environment as the library code can be changed, modified and recompiled without having to re-compile the applications that use this library. For example, for any change in the pthread library code, no change is required in the programs using pthread shared library. A shared library can be accessed through different names :

- Name used by linker ('lib' followed by the library name, followed by '.so' . For example libpthread.so)
- Fully qualified name or soname ('lib' followed by the library name, followed by '.so', followed by '.' and a version number. For example : libpthread.so.1)

- Real name ('lib' followed by the library name, followed by '.so', followed by '.' and a version number, followed by a '.' and a minor number, followed by a '.' and a release number. Release number is optional. For example, libpthread.so.1.1)

A version number is changed for a shared library when the changes done in the code make the shared library incompatible with the previous version. For example, if a function is completely removed then a new version of the library is required.

A minor number is changed in case there is a modification in the code that does not make the shared library incompatible with the previous version being used. For example, a small bug fix won't break the compatibility of the existing shared library so only a minor number is changed while version remains the same.

Now, one may wonder why so many names for a shared library?

Well, these naming conventions help multiple versions of same shared library to co-exist in a system. The programs linking with the shared library do not need to take care about the latest version of the shared library installed in the system. Once the latest version of the shared library is installed successfully, all the programs automatically start linking to the latest version.

The name used by linker is usually a symbolic link to the fully qualified soname which in turn is a symbolic link to the real name.

Placement in File System

There are mainly three standard locations in the filesystem where a library can be placed.

- /lib
- /usr/lib
- /usr/local/lib

We will go by the Filesystem Hierarchy standards(FHS) here. According to the FHS standards, All the libraries which are loaded at start up and running in the root filesystem are kept in /lib. While the libraries that are used by system internally are stored at /usr/lib. These libraries are not meant to be directly used by users or shell scripts. There is a third location /usr/local/lib(though it is not defined in the latest version of FHS). If it exists, it contains all the libraries that are not part of standard distribution. These non-standard libraries are the one's which you download and could be possibly buggy.

Using ldconfig

Once a shared library is created, copy the shared library to directory in which you want the library to reside (for example /usr/local/lib or /usr/lib). Now, run ldconfig command in this directory.

What does ldconfig do?

You remember that we discussed earlier that a linker name for shared library is a symbolic link to the fully qualified soname which in turn is a symbolic link to the real name. Well, this command does exactly the same.

When you run an ELF executable, by default the loader is run first. The loader itself is a shared object file /lib/ld-linux.so.X where 'X' is a version number. This loader in turn finds and loads all the shared libraries on which our program depends.

All the directories that are searched by the loader in order to find the libraries is stored in /etc/ld.so.conf. Searching all the directories specified in /etc/ld.so.conf file can be time consuming so every time ldconfig command is run, it sets up the required symbolic links and then creates a cache in file /etc/ld.so.cache where all the information required for executable is written. Reading information from cache is very less time consuming. The catch here is that ldconfig command needs to be run every-time a shared library is added or removed. So on start-up the program uses /etc/ld.so.cache to load the libraries it requires.

Using Non Standard Library Locations

When using non standard library locations. One of the following three steps could be carried out :

Add the path to /etc/ld.so.conf file. This file contains paths to all the directories in which the library is searched by the loader. This file could sometime contain a single line like :

```
include /etc/ld.so.conf.d/*.conf
```

In that case, just create a conf file in the same directory. You can directly add a directory to cache by using the following command :

```
ldconfig -n [non standard directory path containing shared library]
```

Note that this is a temporary change and will be lost once the system is rebooted. Update the environment variable LD_LIBRARY_PATH to point to your directory containing the shared library. Loader will use the paths mentioned in this environment variable to resolve dependencies.

Note that on some Unix systems the name of the environment variable could differ.

Note: On a related topic, as we explained earlier, there are four main stages through which a source code passes in order to finally become an executable.

Example (How to Create a Shared Library)

Lets take a simple practical example to see how we can create and use shared libraries. The following is the piece of code (shared.c) that we want to put in a shared library :

```
#include "shared.h"
unsigned int add(unsigned int a, unsigned int b)
{
    printf("\n Inside add()\n");
    return (a+b);
}
```

shared.h looks like :

```
#include<stdio.h>
extern unsigned int add(unsigned int a, unsigned int b);
```

Lets first make shared.c as a shared library.

1. Run the following two commands to create a shared library :

```
gcc -c -Wall -Werror -fPIC shared.c
gcc -shared -o libshared.so shared.o
```

The first command compiles the code shared.c into position independent code which is required for a shared library.
The second command actually creates a shared library with name 'libshared.so'.

2. Here is the code of the program that uses the shared library function 'add()'

```
#include<stdio.h>
#include"shared.h"
int main(void)
{
    unsigned int a = 1;
    unsigned int b = 2;
    unsigned int result = 0;

    result = add(a,b);

    printf("\n The result is [%u]\n",result);
    return 0;
}
```

3. Next, run the following command :

```
gcc -L/home/himanshu/practice/ -Wall main.c -o main -lshared
```

This command compiles the main.c code and tells gcc to link the code with shared library libshared.so (by using flag -l) and also tells the location of shared file(by using flag -L).

4. Now, export the path where the newly created shared library is kept by using the following command :

```
export LD_LIBRARY_PATH=/home/himanshu/practice:$LD_LIBRARY_PATH
```

The above command exports the path to the environment variable 'LD_LIBRARY_PATH'.

5. Now run the executable 'main' :

```
# ./main

Inside add()

The result is [3]
```

So we see that shared library was loaded and the add function inside it was executed.

Tweet Like 16 > Add your comment

If you enjoyed this article, you might also like..

1. 50 Linux Sysadmin Tutorials
2. 50 Most Frequently Used Linux Commands (With Examples)
3. Top 25 Best Linux Performance Monitoring and Debugging Tools
4. Mommy, I found it! – 15 Practical Linux Find Command Examples
5. Linux 101 Hacks 2nd Edition eBook Free

- Awk Introduction – 7 Awk Print Examples
- Advanced Sed Substitution Examples
- 8 Essential Vim Editor Navigation Fundamentals
- 25 Most Frequently Used Linux IPTables Rules Examples
- Turbocharge PuTTY with 12 Powerful Add-Ons

Tagged as: /etc/ld.so.conf

{ 23 comments… add one }

- Jalal Hajigholamali June 11, 2012, 2:37 am

  Hi,
  Thanks, very nice article…

  Link
- Kuldeep June 11, 2012, 2:44 am

  Very helpful article !!!
  thanks a lot….

  Link
- Himanshu June 11, 2012, 2:53 am

  Thanks!!

  Link
- bob June 11, 2012, 7:13 am

  Greate article. Learned something new today. I like the clear explanation and simple example. Keep up the good work!!!

  Link
- Mark June 12, 2012, 7:48 pm

  Great post! An equally great 'part 2' to this post would be how to dynamically load and unload shared libraries during run-time.

  Link
- Ishu June 14, 2012, 1:25 pm

  G8t artical and most importantly well explained. Keep up the good work.

  Link
- Praveen kumar January 17, 2013, 8:54 am

  Thanks…

  Link
- Dev June 15, 2013, 10:29 pm

  Wonder full post …. Helped me a lot in understanding static and dynamic Lib.

Thanks buddies 🙂

Link

- Oviya October 4, 2013, 4:47 am

  it may compile in simple line as gcc -o main main.c -o shared shared.c. you will get shared obj file then ./shared to compile or gcc -o shared shared.c -o main main.c you will get main obj file then ./main to execute

  Link

- Oviya October 4, 2013, 4:48 am

  it may compile in simple line as gcc -o main main.c -o shared shared.c. you will get shared obj file then ./shared to compile

  Link

- vidya October 4, 2013, 4:49 am

  give example using dlsym() it will usefull for big project

  Link

- jannu December 12, 2013, 5:34 am

  where is satic library… insufficient infromation… abt shared its good… understandable

  Link

- Karthick May 18, 2014, 12:44 am

  A very clear article. Very clearly explains Shared Libraries and how to create one to a beginner. ldconfig and shared library names are covered to the level necesary for a noob. I've gone through the gnu gcc manual and tried to understand it before, but it was difficult and It was still not clear, until I read this one. Thank you very much.

  Link

- makouda May 27, 2014, 4:13 am

  in the example above, what's the use of telling gcc the location of shared library if we need to export the path with export LD_CONFIG_PATH?? it'nt the same thing?????

  Link

- Ravi June 3, 2014, 3:45 am

  sir can you provide the makefile for same operation
  thanks and regards
  Ravi

  Link

- Anonymous June 3, 2014, 3:48 am

  How to use dlsym() and dlopen()??

  Link

- Nick DeCoursin January 28, 2015, 9:56 pm

  Great read. Thanks so much. I had a similar question as makouda. I feel like these two steps are repetitive, and there should some way to do one or the other and not have to do both.

  Link

- geee May 1, 2015, 1:13 pm

  Nice tutorial. Did try the stuff and also found out:

  you can also ("hardwire") compile into the main an additional search path:

  if you make a subdir lib and put libshared.so into that and compile like this:
  gcc -L./lib -Wall main.c -o main -lshared -Wl,-rpath=lib
  then with the option -Wl,-rpath=lib you compiled into the main EXEC an additinal search path that is looked into first before searching libs on the system if not found in subfolder lib.
  then you can start the main without the env variable LD_LIBRARY_PATH: ./main

  if LD_LIBRARY_PATH is set, it got top priority. if the .so is found in LD_LIBRARY_PATH then it will take that and not the one in lib and not the one in the system…

  good that the steam game "don't starve" did compile such an rpath for subdir lib32 so i could copy a lib there because the newer systemlib on manjaro was incompatible (did copy libcurl-gnutls.so.4.2.0 there)
  see also: here

  Link
- Kelly September 30, 2015, 10:15 am

  Just a note that it seems you only need to run ldconfig once in the directory where you place your shared/dynamic .so file. Updating/adding new objects and copying the .so back to the directory is good enough for code to pick up the new modules.

  Link
- lakshmikanth October 25, 2015, 8:54 am

  Nice .. it is very useful .Thank you

  Link
- Manjunath December 14, 2015, 4:53 am

  Hi,
  When we click on the image link on Login screen, we need to create a shortcut of the website on user desktop. i am able to do this is windows using a jar file. But same jar file not supporting in Linux environment. dll files are not supporting.

  I am using JShortcut library (http://alumnus.caltech.edu/~jimmc/jshortcut/jshortcut/).

  Please can you able to help me on this.

  Link
- Praveen Badiger April 28, 2016, 6:00 am

  Hi,
  I have situation in my project where i have to copy *.so(primary) to some other project folder.
  since these *.so built in library of some apps(Oracle client) and these library
  depend upon other *.so library, i have copied these libraries also, but primary *.s0 is looking secondary library in different folder(folder referred when built).

  Is there any way we can modify primary *.s0 to lookout into project folder for secondary dependency?

  Please if anybody knows reply

  Link

- Pramod May 12, 2017, 2:34 am

  There are not much resources available in the web for Shared libraries, this has provided good insight on Shared libraries, thanks Ramesh.

  Link

Leave a Comment

Name

Email

Website

Comment

Submit

☐ Notify me of followup comments via e-mail

Next post: How to Install Apache CouchDB on CentOS 6 (from Source and EPEL)

Previous post: C Constant Pointers and Pointer to Constants Examples

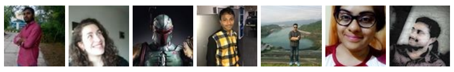
RSS | Email | Twitter | Facebook | Google+

Custom Search Search

The Geek Stuff
17,184 likes

Like Page Share

2 friends like this



POPULAR POSTS

- 15 Essential Accessories for Your Nikon or Canon DSLR Camera
- 12 Amazing and Essential Linux Books To Enrich Your Brain and Library
- 50 UNIX / Linux Sysadmin Tutorials
- 50 Most Frequently Used UNIX / Linux Commands (With Examples)
- How To Be Productive and Get Things Done Using GTD
- 30 Things To Do When you are Bored and have a Computer
- Linux Directory Structure (File System Structure) Explained with Examples
- Linux Crontab: 15 Awesome Cron Job Examples
- Get a Grip on the Grep! – 15 Practical Grep Command Examples
- Unix LS Command: 15 Practical Examples
- 15 Examples To Master Linux Command Line History
- Top 10 Open Source Bug Tracking System
- Vi and Vim Macro Tutorial: How To Record and Play
- Mommy, I found it! -- 15 Practical Linux Find Command Examples
- 15 Awesome Gmail Tips and Tricks
- 15 Awesome Google Search Tips and Tricks
- RAID 0, RAID 1, RAID 5, RAID 10 Explained with Diagrams
- Can You Top This? 15 Practical Linux Top Command Examples
- Top 5 Best System Monitoring Tools
- Top 5 Best Linux OS Distributions
- How To Monitor Remote Linux Host using Nagios 3.0
- Awk Introduction Tutorial – 7 Awk Print Examples
- How to Backup Linux? 15 rsync Command Examples
- The Ultimate Wget Download Guide With 15 Awesome Examples
- Top 5 Best Linux Text Editors
- Packet Analyzer: 15 TCPDUMP Command Examples
- The Ultimate Bash Array Tutorial with 15 Examples
- 3 Steps to Perform SSH Login Without Password Using ssh-keygen & ssh-copy-id
- Unix Sed Tutorial: Advanced Sed Substitution Examples
- UNIX / Linux: 10 Netstat Command Examples
- The Ultimate Guide for Creating Strong Passwords
- 6 Steps to Secure Your Home Wireless Network
- Turbocharge PuTTY with 12 Powerful Add-Ons

CATEGORIES

- Linux Tutorials
- Vim Editor
- Sed Scripting

- Awk Scripting
- Bash Shell Scripting
- Nagios Monitoring
- OpenSSH
- IPTables Firewall
- Apache Web Server
- MySQL Database
- Perl Programming
- Google Tutorials
- Ubuntu Tutorials
- PostgreSQL DB
- Hello World Examples
- C Programming
- C++ Programming
- DELL Server Tutorials
- Oracle Database
- VMware Tutorials

Ramesh Natarajan

G+ Theo dõi

About The Geek Stuff

My name is **Ramesh Natarajan**. I will be posting instruction guides, how-to, troubleshooting tips and tricks on Linux, database, hardware, security and web. My focus is to write articles that will either teach you or help you resolve a problem. Read more about Ramesh Natarajan and the blog.

Contact Us

Email Me : Use this Contact Form to get in touch me with your comments, questions or suggestions about this site. You can also simply drop me a line to say hello!.

Follow us on Google+

Follow us on Twitter

Become a fan on Facebook

Support Us

Support this blog by purchasing one of my ebooks.

Bash 101 Hacks eBook