Linux Academy

Hands-on Lab

Manage Shared Libraries

Contents

Lab Connection Information

- Labs may take up to five minutes to build
- The IP address of your server is located on the Live! Lab page
- Username: linuxacademy
- Password: 123456
- Root Password: 123456

Related Courses

LPIC-1 Sysem Administrator - Exam 101

Related Videos

Manage Shared Libraries

Need Help?

Linux Academy Community

... and you can always send in a support ticket on our website to talk to an instructor!

Shared libraries are files that contain reusable functions for use by other applications. Library files use the `.so` extension and often contain version information – `libraryname-version.so`. In this lab, we learn about libraries, then explore how to best manage and configure libraries and the dynamic linker.

Log in to the server using the credentials provided on the Hands-on Lab page. Use `sudo su -` to switch to the *root* user.

Library Locations

Shared libraries are traditionally contained in the `/lib`, `/lib64`, `/usr/lib`, and `/usr/lib64` directories. Occasionally they are also found in `/usr/local/lib`, typically when installed with a third-party application.

Let's view the libraries available in our `/lib64` file:

```
[root@ip] ls -al /lib64/*.so
-rwxr-xr-x. 1 root root  154664 May 10  2016 /lib64/ld-2.12.so
-rwxr-xr-x. 1 root root   19368 May 10  2016 /lib64/libanl-2.12.so
-rwxr-xr-x. 1 root root    8488 May 10  2016 /lib64/
libBrokenLocale-2.12.so
-rwxr-xr-x. 1 root root 1923352 May 10  2016 /lib64/libc-2.12.so
-rwxr-xr-x. 1 root root  197064 May 10  2016 /lib64/libcidn-2.12.so
-rwxr-xr-x. 1 root root   40400 May 10  2016 /lib64/libcrypt-2.12.so
-rwxr-xr-x. 1 root root 1525560 Jan 11 18:40 /lib64/libdb-4.7.so
-rwxr-xr-x. 1 root root   19536 May 10  2016 /lib64/libdl-2.12.so
-rwxr-xr-x. 1 root root   10312 Jul 12  2016 /lib64/libfreebl3.so
-rwxr-xr-x. 1 root root  477712 Jul 12  2016 /lib64/libfreeblpriv3.so
-rwxr-xr-x. 1 root root  596360 May 10  2016 /lib64/libm-2.12.so
-rwxr-xr-x. 1 root root  113432 May 10  2016 /lib64/libnsl-2.12.so
-rwxr-xr-x. 1 root root  244624 May 11  2016 /lib64/libnspr4.so
-rwxr-xr-x. 1 root root   42808 May 10  2016 /lib64/libnss_compat-2.12.
so
-rwxr-xr-x. 1 root root   27424 May 10  2016 /lib64/libnss_dns-2.12.so
-rwxr-xr-x. 1 root root   65960 May 10  2016 /lib64/libnss_files-2.12.so
-rwxr-xr-x. 1 root root   24152 May 10  2016 /lib64/libnss_hesiod-2.12.
so
-rwxr-xr-x. 1 root root   52560 May 10  2016 /lib64/libnss_nis-2.12.so
-rwxr-xr-x. 1 root root   61712 May 10  2016 /lib64/libnss_nisplus-2.12.
so
-rwxr-xr-x. 1 root root   18720 May 11  2016 /lib64/libplc4.so
-rwxr-xr-x. 1 root root   14560 May 11  2016 /lib64/libplds4.so
-rwxr-xr-x. 1 root root   63200 May 10  2016 /lib64/libproc-3.2.8.so
-rwxr-xr-x. 1 root root  142688 May 10  2016 /lib64/libpthread-2.12.so
-rwxr-xr-x. 1 root root  110960 May 10  2016 /lib64/libresolv-2.12.so
-rwxr-xr-x. 1 root root   43944 May 10  2016 /lib64/librt-2.12.so
-rwxr-xr-x. 1 root root   21928 May 10  2016 /lib64/libSegFault.so
-rwxr-xr-x. 1 root root   34008 May 10  2016 /lib64/libthread_db-1.0.so
-rwxr-xr-x. 1 root root   14584 May 10  2016 /lib64/libutil-2.12.so
```

As we can see, the majority of these libraries contain specific version numbers at the end of their filenames, following the libraryname-version.so style. To accommodate changing versions, libraries are commonly symlinked to files with simpler names – so a file libcrypt.so would be symlinked to the libcrypt-2.12.so file. This provides the application a standard name to use when calling the application.

Linking to Shared Objects

There are two ways in which applications can link to a shared object: static and dynamic.

Static Linking

Static linking is when an application packages a full version of the library within it. By using static linking, we can ensure that the application always has the appropriate version of the library available; although, this process increases the size of the application. Additionally, any feature upgrades and bug fixes added to that library later will not to included in the application-packaged library unless the application maintainers upgrade the library and repackage the application with the included changes.

Dynamic Linking

Dynamic linking is when an application uses the version of the shared library already on the computer. Generally, this is where we see the symlinking of library files with standard names to the name, versioned file.

When calling to these libraries, the dynamic linker, ld.so, is used. This is a background service that copies the library to the computer's RAM for use by the application.

Managing Libraries

Determine Needed Libraries

To determine which application uses what libraries, use the ldd command. This can help if we are uncertain if an application has all its needed libraries. We need to know the full path to the application to use this. First, determine the application path, then use it to determine the libraries needed for shhd:

```
[root@ip] which sshd
/usr/sbin/sshd
[root@ip] ldd /usr/sbin/sshd
    linux-vdso.so.1 =>  (0x00007ffc33318000)
    libfipscheck.so.1 => /lib64/libfipscheck.so.1 (0x00007fca7ea14000)
    libwrap.so.0 => /lib64/libwrap.so.0 (0x00007fca7e809000)
    libaudit.so.1 => /lib64/libaudit.so.1 (0x00007fca7e5e4000)
    libpam.so.0 => /lib64/libpam.so.0 (0x00007fca7e3d6000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007fca7e1d2000)
    libselinux.so.1 => /lib64/libselinux.so.1 (0x00007fca7dfb2000)
    libcrypto.so.10 => /usr/lib64/libcrypto.so.10 (0x00007fca7dbce000)
    libutil.so.1 => /lib64/libutil.so.1 (0x00007fca7d9cb000)
    libz.so.1 => /lib64/libz.so.1 (0x00007fca7d7b4000)
    libnsl.so.1 => /lib64/libnsl.so.1 (0x00007fca7d59b000)
    libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007fca7d364000)
    libresolv.so.2 => /lib64/libresolv.so.2 (0x00007fca7d149000)
    libgssapi_krb5.so.2 => /lib64/libgssapi_krb5.so.2
(0x00007fca7cf05000)
    libkrb5.so.3 => /lib64/libkrb5.so.3 (0x00007fca7cc1e000)
    libk5crypto.so.3 => /lib64/libk5crypto.so.3 (0x00007fca7c9f1000)
    libcom_err.so.2 => /lib64/libcom_err.so.2 (0x00007fca7c7ed000)
    libnss3.so => /usr/lib64/libnss3.so (0x00007fca7c4ad000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fca7c118000)
    /lib64/ld-linux-x86-64.so.2 (0x000055fa65d9f000)
    libfreebl3.so => /lib64/libfreebl3.so (0x00007fca7bf15000)
    libkrb5support.so.0 => /lib64/libkrb5support.so.0
(0x00007fca7bd09000)
    libkeyutils.so.1 => /lib64/libkeyutils.so.1 (0x00007fca7bb06000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fca7b8e9000)
    libnssutil3.so => /usr/lib64/libnssutil3.so (0x00007fca7b6bc000)
    libplc4.so => /lib64/libplc4.so (0x00007fca7b4b7000)
    libplds4.so => /lib64/libplds4.so (0x00007fca7b2b3000)
    libnspr4.so => /lib64/libnspr4.so (0x00007fca7b074000)
    librt.so.1 => /lib64/librt.so.1 (0x00007fca7ae6c000)
```

`sshd` requires a number of shared objects. If we were using a bare installation and needed to install the `sshd` package, we could then expect that these libraries already exist on the system, and, if they do not, that they will be downloaded at the time `sshd` is acquired.

Configure Dynamic Linker Runtime Bindings

`ldconfig` allows us to configure the runtime bindings of the dynamic linker. This creates links and caches the most recent of the shared libraries used. This does this by referencing the `ls.so.conf` file, which determines which directories will be scanned for shared objects.

The `ldconfig` command is used as part of the exit script at the end of an application installation and normally does not have to be run on its own. However, if you have any doubts about any shared objects being updated or you made manual changes to the `ld.so.conf` file and need to update the cache, this command should be used. This will update the `ld.so.cache` file:

```
[root@ip] cd /etc/
[root@ip] ls -al ld.so.cache
-rw-r--r--. 1 root root 17917 Mar  7 04:22 ld.so.cache
```

Now, run `ldconfig` and compare the timestamps:

```
[root@ip] ldconfig
[root@ip] ls -al ld.so.cache
-rw-r--r-- 1 root root 17917 Apr 28 19:15 ld.so.cache
```

Additionally, if we want to force the dynamic linker to look in other directories, we can alter the `ld.so.conf` file. `cat` out this file:

```
[root@ip] cat ld.so.conf
include ld.so.conf.d/*.conf
```

Currently, this is just an `include` statement referencing files in `ld.so.conf.d`.

```
[root@ip] cd ld.so.conf.d
```

```
[root@ip] ls -al
total 24
drwxr-xr-x.  2 root root 4096 Mar  7 04:55 .
drwxr-xr-x. 70 root root 4096 Apr 28 19:15 ..
-r--r--r--.  1 root root  324 May 31  2016 kernel-2.6.32-642.1.1.el6.
x86_64.conf
-r--r--r--.  1 root root  324 Feb 24 14:37 kernel-2.6.32-642.15.1.el6.
x86_64.conf
-r--r--r--.  1 root root  324 Feb 27 03:48 kernel-ml-4.10.1-1.el6.
elrepo.x86_64.conf
-rw-r--r--.  1 root root   17 Jan 26 22:25 mysql-x86_64.conf
```

Here we see a number of configurations for third-party application and the kernel. Should we view one of these, we can see which directory is being scanned:

```
[root@ip] cat mysql-x86_64.conf
/usr/lib64/mysql
```

We can create a new file here to add more directories if we so desired.

Alternatively, we can alter the `LD_LIBRARY_PATH` environmental variable to determine which directories are looked at for shared objects. To see what populates this variable currently, use:

```
[root@ip] echo $LD_LIBRARY_PATH
```

We can customize this by overriding the path or appending new ones to the existing path. This process is explained later in the course.

Review

Libraries provide applications with reusable functions. We now know how to find the directories containing these functions and how applications link to these libraries, as well as how to determine needed libraries and configure the behavior of the dynamic linker. This lab has been completed!