Shared Libraries

A single copy of a shared library can be used by many applications at once; thus, both executable sizes and application load time are reduced.

Shared libraries have the extension **.so**. Typically, the full name is something like **libc.so.N** where **N** is a major version number.

Under Linux, shared libraries are carefully versioned. For example, a shared library might have any of the following names:

- **libmyfuncs.so.1.0** - The actual shared library.
- **libmyfuncs.so.1** - The name included in the **soname** field of the library. Used by the executable at runtime to find the latest revision of the **v. 1myfuncs** library.
- **libmyfuncs.so** - Used by **gcc** to resolve symbol names in the library at link time when the executable is created.

To create a shared library, first you must compile all sources with the -**fPIC** option, which generates so-called Position Independent Code; do not use **-fpic**; it produces somewhat faster code on **m68k**, **m88k**, and **Sparc** chips, but imposes arbitrary size limits on shared libraries.

```
$ gcc -fPIC -c func1.c
$ gcc -fPIC -c func2.c
```

To create a shared library, the option **-shared** must be given during compilation, giving the **soname** of the library, as well as the full library name as the output:

```
$ gcc -fPIC -shared -Wl,-soname=libmyfuncs.so.1 *.o -
o libmyfuncs.so.1.0 -lc
```

where the **-Wl** tells **gcc** to pass the option to the linker. The **-lc** tells the linker that **libc** is also needed, which is generally the case.

You can usually get away ignoring the **-fPIC** and **-Wl**, **soname** options, but it is not a good idea. This is because **gcc** normally emits such code anyway. In fact, giving the option prevents the compiler from ever issuing position-dependent code.

Note that it is really the linker (**ld**) that is doing the work, and the above step could also have been written as:

```
$ ld -shared -soname=libmyfuncs.so.1 *.o -o libmyfuncs.so.1.0 -lc
```

To get the above to link and run properly, you will also have to do:

```
$ ln -s libmyfuncs.so.1.0 libmyfuncs.so
$ ln -s libmyfuncs.so.1.0 libmyfuncs.so.1
```

If you leave out the first symbolic link, you will not be able to compile the program; if you leave out the second, you will not be able to run it. It is at this point that the version check is done, and if you do not use the **-soname** option when compiling the library, no such check will be done.

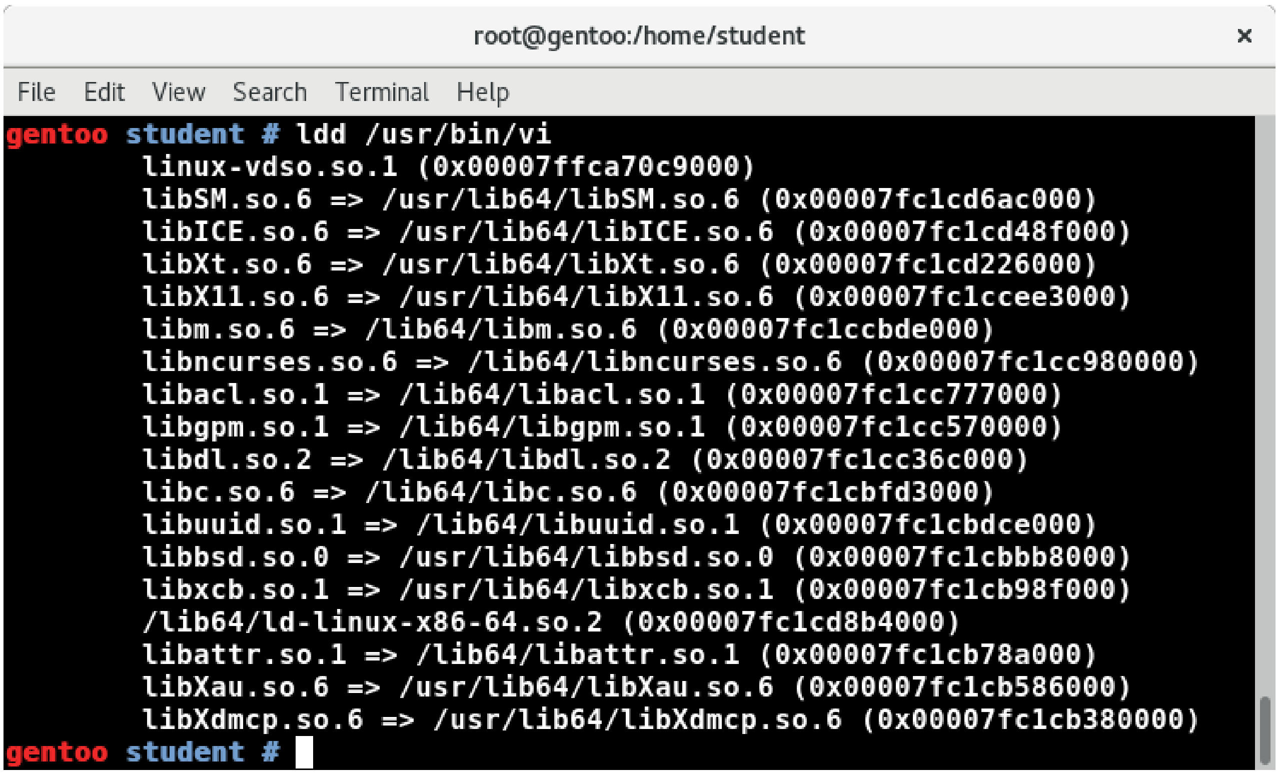
In many cases, both a shared and a static version of the same library may exist on the system, and the linker will choose the shared version by default. This may be overridden with the **-static** option to the linker. However, if you do this, all libraries will be linked in statically, including **libc**, so the resulting executable will be large; you have to be more careful than that.

The GNU **libtool** script can assist in providing shared library support, helping with compiling and linking libraries and executables, and is particularly useful for distributing applications and packages. Full documentation can be obtained by doing **info libtool**.

Finding shared libraries

A program which uses shared libraries has to be able to find them at run time.

ldd can be used to ascertain what shared libraries an executable requires. It shows the **soname** of the library and what file it actually points to:


root@gentoo:/home/student
File Edit View Search Terminal Help

```
gentoo student # ldd /usr/bin/vi
        linux-vdso.so.1 (0x00007ffca70c9000)
        libSM.so.6 => /usr/lib64/libSM.so.6 (0x00007fc1cd6ac000)
        libICE.so.6 => /usr/lib64/libICE.so.6 (0x00007fc1cd48f000)
        libXt.so.6 => /usr/lib64/libXt.so.6 (0x00007fc1cd226000)
        libX11.so.6 => /usr/lib64/libX11.so.6 (0x00007fc1ccee3000)
        libm.so.6 => /lib64/libm.so.6 (0x00007fc1ccbde000)
        libncurses.so.6 => /lib64/libncurses.so.6 (0x00007fc1cc980000)
        libacl.so.1 => /lib64/libacl.so.1 (0x00007fc1cc777000)
        libgpm.so.1 => /lib64/libgpm.so.1 (0x00007fc1cc570000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007fc1cc36c000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fc1cbfd3000)
        libuuid.so.1 => /lib64/libuuid.so.1 (0x00007fc1cbdce000)
        libbsd.so.0 => /usr/lib64/libbsd.so.0 (0x00007fc1cbbb8000)
        libxcb.so.1 => /usr/lib64/libxcb.so.1 (0x00007fc1cb98f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc1cd8b4000)
        libattr.so.1 => /lib64/libattr.so.1 (0x00007fc1cb78a000)
        libXau.so.6 => /usr/lib64/libXau.so.6 (0x00007fc1cb586000)
        libXdmcp.so.6 => /usr/lib64/libXdmcp.so.6 (0x00007fc1cb380000)
gentoo student #
```

ldconfig is generally run at boot time (but can be run anytime), and uses the file **/etc/ld.so.conf**, which lists the directories that will be searched for shared libraries. **ldconfig** must be run as root and shared libraries should only be stored in system directories when they are stable and useful.

Besides searching the data base built up by **ldconfig**, the linker will first search any directories specified in the environment variable **LD_LIBRARY_PATH**, a colon separated list of directories, as in the **PATH** variable. So, you can do:

```
$ LD_LIBRARY_PATH=$HOME/foo/lib
$ foo [args]
```

or

```
$ LD_LIBRARY_PATH=$HOME/foo/lib  foo [args]
```