Static & Shared Libraries

Archives :

An archive (or static library) is simply a collection of object files stored as a single file. (An archive is roughly the equivalent of a Windows .LIB file.) When you provide an archive to the linker, the linker searches the archive for the object files it needs, extracts them, and links them into your program much as if you had provided those object files directly.

You can create an archive using the ar command. Archive files traditionally use a .a extension rather than the .o extension used by ordinary object files. Here's how you would combine test1.o and test2.o into a single libtest.a archive:

$ar cr libtest.a test1.o test2.o

The cr flags tell ar to create the archive. Now you can link with this archive using the -ltest option with gcc or g++.

When the linker encounters an archive on the command line, it searches the archive for all definitions of symbols (functions or variables) that are referenced from the object files that it has already processed but not yet defined.**The object files that define those symbols are extracted from the archive and included in the final executable.** Because the linker searches the archive when it is encountered on the command line, it usually makes sense to put archives at the end of the command line.

Example 1.0 :

$cat test1.c

```
void test1()
{
        printf("test1\n");
}
```

$cat test2.c

```
void test2()
{
        printf("test2\n");
 }
```

$gcc -c test1.c
$gcc -c test2.c

Create an archive or static library using ar.

$ar rcs libtest.a test1.o test2.0

Now create an app.c file that uses your static library.

$cat app.c

```
#include <stdio.h>
int main()
{
```

```
        test1();
        test2();
        return 0;
}
```

Now link your static library and compile your app.

$gcc -o app app.c -L. -ltest

L. tells that your libtest.a is in the current folder.

Shared Libraries :

A shared library (also known as a shared object, or as a dynamically linked library) is similar to a archive in that it is a grouping of object files. However, there are many important differences.**The most fundamental difference is that when a shared library is linked into a program, the final executable does not actually contain the code that is present in the shared library. Instead, the executable merely contains a reference to the shared library**. If several programs on the system are linked against the same shared library, they will all reference the library, but none will actually be included.Thus, the library is "shared" among all the programs that link with it.

A second important difference is that a shared library is not merely a collection of object files, out of which the linker chooses those that are needed to satisfy undefined references. Instead, the object files that compose the shared library are combined into a single object file so that a program that links against a shared library always includes all of the code in the library, rather than just those portions that are needed.

To create a shared library, you must compile the objects that will make up the library using the -fPIC option to the compiler, like this:

```
$gcc -c -fPIC test1.c
```

The -fPIC option tells the compiler that you are going to be using shared object test.o as part of a Position-Independent Code (PIC) .

PIC stands for position-independent code. The functions in a shared library may be loaded at different addresses in different programs, so the code in the shared object must not depend on the address (or position) at which it is loaded. This consideration has no impact on you, as the programmer, except that you must remember to use the -fPIC flag when compiling code that will be used in a shared library.

Writing and Using Libraries

Then you combine the object files into a shared library, like this:

$gcc -shared -fPIC -o libtest.so test1.o test2.o

The -shared option tells the linker to produce a shared library rather than an ordinary executable. Shared libraries use the extension .so, which stands for shared object. Like static archives, the name always begins with lib to indicate that the file is a library. Linking with a shared library is just like linking with a static archive. For example, the following line will link with

libtest.so if it is in the current directory, or one of the standard library search directories on the system:

$ gcc -o app app.o -L. –ltest

Suppose that both libtest.a and libtest.so are available.Then the linker must choose one of the libraries and not the other.The linker searches each directory (first those specified with -L options, and then those in the standard directories).When the linker finds a directory that contains either libtest.a or libtest.so, the linker stops search directories. If only one of the two variants is present in the directory, the linker chooses that variant. Otherwise, the linker chooses the shared library version, unless you explicitly instruct it otherwise.You can use the -static option to demand static archives. For example, the following line will use the libtest.a archive, even if the libtest.so shared library is also available:

$gcc -static -o app app.o -L. –ltest

The ldd command displays the shared libraries that are linked into an executable. These libraries need to be available when the executable is run. Note that ldd will list an additional library called ld-linux.so, which is a part of GNU/Linux's dynamic linking mechanism.

Using -wl,-rpath:

When you link a program with a shared library, the linker does not put the full path to the shared library in the resulting executable. Instead, it places only the name of the shared library.When the program is actually run, the system searches for the shared library and loads it.The system searches only /lib and /usr/lib, by default. If a shared library that is linked into your program is installed outside those directories, it will not be found, and the system will refuse to run the program.

One solution to this problem is to use the -Wl,-rpath option when linking the program. Suppose that you use this:

$gcc -o app app.o -L. -ltest -Wl,-rpath,/usr/local/lib

Example:

Let us consider above **example 1.0** test1.c , test2.c and app.c

$gcc -fPIC -c test1.c test2.c

Create a shared library as follows

$gcc -shared -fPIC -o libtest.so test1.o test2.o

Link it with your application programas follows

$ gcc -o app app.c -L. -ltest -Wl,-rpath,/home/rudra/examples/lib_dynamic

---

Rudrayya Chowdary R
Project Engineer,C DAC Hyderabad
Email : rudrayyar@cdac.in