Static and Dynamic Libraries | Set 1

When a C program is compiled, the compiler generates object code. After generating the object code, the compiler also invokes linker. One of the main tasks for linker is to make code of library functions (eg printf(), scanf(), sqrt(), ..etc) available to your program. A linker can accomplish this task in two ways, by copying the code of library function to your object code, or by making some arrangements so that the complete code of library functions is not copied, but made available at run-time.

Static Linking and Static Libraries is the result of the linker making copy of all used library functions to the executable file. Static Linking creates larger binary files, and need more space on disk and main memory. Examples of static libraries (libraries which are statically linked) are, ***.a*** files in Linux and ***.lib*** files in Windows.

Steps to create a static library Let us create and use a Static Library in UNIX or UNIX like OS.

1. Create a C file that contains functions in your library.

```
/* Filename: lib_mylib.c */
#include <stdio.h>
void fun(void)
{
  printf("fun() called from a static library");
}
```

Run on IDE

We have created only one file for simplicity. We can also create multiple files in a library.

2. Create a header file for the library

```
/* Filename: lib_mylib.h */
void fun(void);
```

Run on IDE

3. Compile library files.

```
gcc -c lib_mylib.c -o lib_mylib.o
```

4. Create static library. This step is to bundle multiple object files in one static library (see ar for details). The output of this step is static library.

```
ar rcs lib_mylib.a lib_mylib.o
```

5. Now our static library is ready to use. At this point we could just copy lib_hello_static.a somewhere else to use it. For demo purposes, let us keep the library in the current directory.

Let us create a driver program that uses above created static library.

1. Create a C file with main function

```
/* filename: driver.c  */
#include "lib_mylib.h"
void main()
{
  fun();
}
```

Run on IDE

2. Compile the driver program.

```
gcc -c driver.c -o driver.o
```

3. Link the compiled driver program to the static library. Note that -L**.** is used to tell that the static library is in current folder (See this for details of -L and -l options).

```
gcc -o driver driver.o -L. -l_mylib
```

4. Run the driver program

```
./driver
fun() called from a static library
```

Following are some important points about static libraries.

1. For a static library, the actual code is extracted from the library by the linker and used to build the final executable at the point you compile/build your application.

2. Each process gets its own copy of the code and data. Where as in case of dynamic libraries it is only code shared, data is specific to each process. For static libraries memory footprints are larger. For example, if all the window system tools were statically linked, several tens of megabytes of RAM would be wasted for a typical user, and the user would be slowed down by a lot of paging.

3. Since library code is connected at compile time, the final executable has no dependencies on the the library at run time i.e. no additional run-time loading costs, it means that you don't need to carry along a copy of the library that is being used and you have everything under your control and there is no dependency.

4. In static libraries, once everything is bundled into your application, you don't have to worry that the client will have the right library (and version) available on their system.

5. One drawback of static libraries is, for any change(up-gradation) in the static libraries, you have to recompile the main program every time.

6. One major advantage of static libraries being preferred even now “is speed”. There will be no dynamic querying of symbols in static libraries. Many production line software use static libraries even today.

Dynamic linking and Dynamic Libraries Dynamic Linking doesn’t require the code to be copied, it is done by just placing name of the library in the binary file. The actual linking happens when the program is run, when both the binary file and the library are in memory. Examples of Dynamic libraries (libraries which are linked at run-time) are, ***.so*** in Linux and ***.dll*** in Windows.

We will soon be covering more points on Dynamic Libraries and steps to create them.

This article is compiled by **Abhijit Saha** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

32 Comments Category: Misc

Related Posts:

- Combinatorial Game Theory | Set 4 (Sprague – Grundy Theorem)
- Combinatorial Game Theory | Set 3 (Grundy Numbers/Nimbers and Mex)
- Combinatorial Game Theory | Set 2 (Game of Nim)
- Combinatorial Game Theory | Set 1 (Introduction)
- Decorator Pattern | Set 3 (Coding the Design)
- The Decorator Pattern | Set 2 (Introduction and Design)
- How to add articles to “To Do” and “Done” lists on GeeksforGeeks?
- Testimonials – Words that keep us going

(Login to Rate and Mark)

2.5 Average Difficulty : **2.5**/**5.0**
Based on **2** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

32 Comments GeeksforGeeks Login

Recommend Share Sort by Newest

Join the discussion…

Arun Kp • 10 months ago
Hi, if someone asks, whether stdio.h is linked statically or dynamically. what will be the answer?
△ | ▽ • Reply • Share ›

Nitish > Arun Kp • 20 days ago
Have you got the answer?
△ | ▽ • Reply • Share ›

Sachin Sundar • a year ago
"3. Link the compiled driver program to the static library. Note that -L. is used to tell that the static library is in current folder (See this for details of -L and -l options).
gcc -o driver driver.o -L. -l_mylib"

The above won't work.

There's a change in this. Its:
gcc -o driver -driver.o -L. lib_mylib.a
1 △ | ▽ • Reply • Share ›

Saurabh mehta > Sachin Sundar • 20 days ago
Thanks. A little mistake though. Its not '-driver.o' its 'driver.o'
△ | ▽ • Reply • Share ›

Rashi • a year ago
good explanation... easy to understand
△ | ▽ • Reply • Share ›

Pratyush • a year ago
It would be good to mention the difference in the usage of linker flags like -Wl,-dy and -Wl,-dn since they're not obvious and can sometimes cause compilation to fail. Also it's good to understand the prefixing that happens when you say '-lsomething' since the compiler looks for a library with name 'libsomething.so'.
△ | ▽ • Reply • Share ›

venu • a year ago
good sir....a small doubt.....