Khalil
Posted on Sep 4, 2020 • Updated on Apr 3, 2022

All you need to know about C Static libraries

#c #linux #tutorial #linker

Introduction

Before starting to talk about the subject matter, let us take a brief look at the compilation phases of a C program.
There are basically four phases:

1. Pre-processing
2. Compilation
3. Assembly
4. Linking

In this article, we will focus on the static libraries and their role in the linking phase of a program. But, first of all, let us define a library.

What is a library and what is it good for?

A library is a collection of code routines (functions, classes, variables, and so on) that can be called upon when building our program, so instead of writing it ourselves, we can go and get it from something that has already been written and optimized. That is where the idea behind libraries comes from. We are reusing blocks of codes that have come from somewhere else.

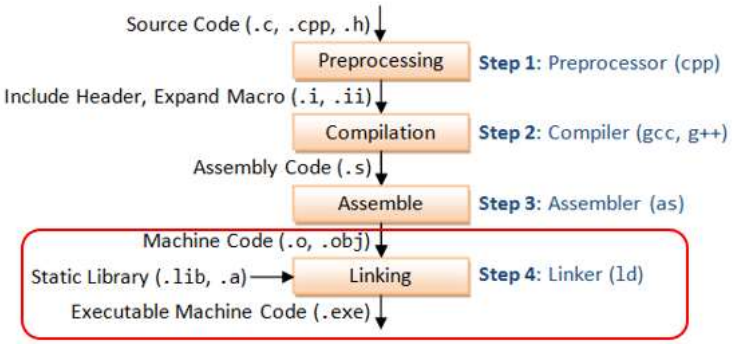
Basically, we have two kinds of libraries:

- static libraries
- shared (or dynamic) libraries

The main reason for writing a library is to allow code reusability, thus save considerable development time.

What is a static library and how does it work?

A static library is a file containing a collection of object files (*.o) that are linked into the program during the linking phase of compilation and are not relevant during runtime.



As shown in the diagram above, when a program is compiled, the compiler generates an object file from a source file. After generating the object file, the compiler also invokes the Linker. The role of the linker, in this case, is to copy the code of the library to our object file.

Basically, static libraries are just a collection of object files that are merged by the linker with another object file to form a final executable.

Conventionally, they start with “lib” and end with “.a” or “.lib” (depending on your platform).

How to create static libraries?

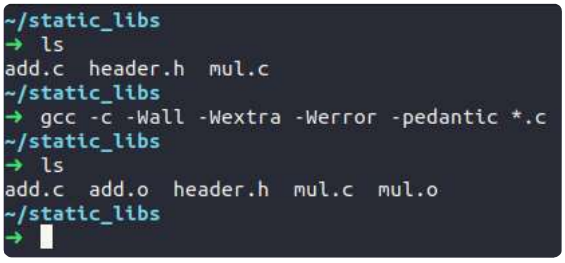
To create a static library, we need to specify to the compiler, which is GCC in our case, that we want to compile all library codes (*.c) into object files (*.o) without linking. To do that we are going to use the command below.

```
$ gcc -c -Wall -Werror -Wextra *.c
```

Flags description:
-c: Compile and assemble, but do not link.
-Wall, -Werro and -Wextra: These aren't necessary but they are recommended to generate better code.

Note that the "*.c" matches all files in the current working directory with the ".c" extension.

For example, let us take two c files, **add.c**, and **mul.c** which make respectively the addition and the multiplication of two integers, and a header file that contains the prototypes of these functions. The picture below shows the output generated after using the command.



Once we have object file(s), we can now bundle all object files into one static library.
To create a static library or to add additional object files to an existing static library, we have to use the GNU ar (archiver) program. We can use a command like this:

```
$ ar -rc libname.a *.o
```

This command creates a static library named "libname.a" and puts copies of the object files "add.o" and "mul.o" in it. The 'c' flag tells ar to create the library if it doesn't already exist. The 'r' flag tells it to insert object files or replace existing object files in the library, with the new object files.

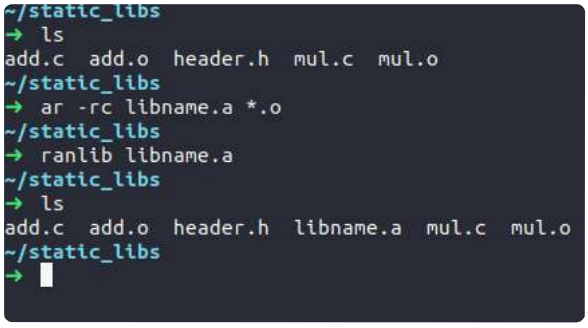
After an archive is created or modified, there is a need to index it. This index is later used by the compiler to speed up symbol-lookup inside the library and to make sure that the order of the symbols in the library will not matter during compilation. There are two ways to create or update the index. The first one is, by using the command ranlib.

```
$ ranlib libname.a
```

or by adding an extra flag (*-s*) to the ar command and it becomes like this:

```
$ ar -rcs libname.a *.o
```

The picture below shows the execution of these commands on our example.



In order to list the names of the object files in our library, we can use the ar command with *-t* flag.

```
~/static_libs
→ ar -t libname.a
add.o
mul.o
~/static_libs
→
```

How to use them?

Now our static library "libname.a" is ready to be used. we can use it in a program. This is done by adding the library's name to the object file(s) given to the linker. First, let us create a C source file that uses the above created static library.

```
~/static_libs
→ ls
add.c  add.o  header.h  libname.a  main.c  mul.c  mul.o
~/static_libs
→ cat main.c
#include "header.h"
#include <stdio.h>

int main(void)
{
        printf("add(40, 2) = %d\n", add(40, 2));
        printf("mul(40, 2) = %d\n", mul(40, 2));

        return 0;
}
```

Now we can use the command below to create our final executable program:

```
$ gcc main.c -L. -lname -o main
```

This will create a program using the object file "main.o", and any symbols it requires from the "name" static library.

Flags description:
-L : Specifies the path to the given libraries ('.' referring to the current directory)
-l : Specifies the library name without the "lib" prefix and the ".a" suffix, because the linker attaches these parts back to the name of the library to create a name of a file to look for.

All we have to do now is to run our program.

```
~/static_libs
→ gcc main.c -L. -lname -o main
~/static_libs
→ ls
add.c  add.o  header.h  libname.a  main  main.c  mul.c  mul.o
~/static_libs
→ ./main
add(40, 2) = 42
mul(40, 2) = 80
~/static_libs
→
```

That was my summary of static libraries!
Hopefully, you found this useful. In the next blog post, we will cover dynamic libraries and the difference between them and static libraries.

Happy learning!

Top comments (26)

Dohou Daniel • May 1

I read this in 2022, and came back for it in 2023.
That is to make you understand that this was really helpful, and also a well-detailed explanation.
Thank you 💚.

Gabriel Effangha • Sep 6

I understand static libraries now, thanks to you.

Endurance Edward • Aug 4

Succinct 👌

ONDO ABAGHE Rooly Marvin • Oct 10 '22

The content is so clear and concise, thanks a lot

Hamzat Abdul-muizz • Sep 30 '22

Thanks so much well detailed explanation

yacoubou • Oct 2 '22

Good content thank

Olimpio • Oct 2 '22

Thanks for this amazing article

Onwuka Chukwuebuka Nathan • Nov 8 '22

Nice 👍

pweety_aries • Oct 1 '22

thanks .... it well explanatory

Chukwuemeka Ejiofor • Sep 30 '22

This was really insightful. Thanks 👍

Demmythetechie • Nov 4 '22

Nice article bro, shows you are very good at what you do.
Thanks for the help, I understood in a Jiffy.

Otieno Onyango • Apr 9

Could you please say something about the content of the add.c and the mul.c, for a beginner like me, that is. Otherwise, great content here

Khalil • Apr 9

As mentioned above, add.c and mul.c are files that contain a single function that makes the addition or the multiplication of two integer values. You can think of them as follow:

```
/* add.c */
int add(int a, int b) { return a + b; }

/* mul.c */
int mul(int a, int b) { return a * b; }
```

I hope this answers your question :)

Code warrior • Apr 10

Thank you for this amazing post.May you please do one for Dynamic libraires?

Itumeleng-Malgas • Apr 7

Awesome stuff

David-ngozichukwuka • Apr 9

This article covers every important detail on static libraries and simplifies these concepts. Impressive material

Bereket-Melese • Mar 17

Concise and to the Point. Keep up the good work.

Hossam Ayman • Jan 6

Thank you. That was an amazing explanation.

Esogibe Chidubem Andrew • Jan 6

Great explanation. Just cleared my little confusion with ease. Thanks Dev.

XNFTS_Chuks💎 • Jul 23 '22

I really learned and enjoyed a lot of ideas shared here thanks so much

Hayatudeen Abdulrahman • Jun 27 '22

Thank you for this. I really needed it for my project

pentacular • Sep 4 '20

I think it is worth noting that libraries are not part of the C language.

Which means that they are all implementation features and vary between compiler and system.

The description of libraries here seems to be for gcc libraries under linux?

Khalil • Sep 4 '20

Yes, you're right. The C libraries aren't part of the language.
It slipped my mind. Maybe I'll add it. Thanks for your feedback.

And yes I've used the GCC compiler on Linux. but the same principle applies to other platforms.

pentacular • Sep 4 '20

You're welcome.

Valeryio • Oct 2

Succint, but, what is the difference between libname, and the module add.c and mul.c ?
Here you include "header.h", and without libname.a, it will also compile and work.

Could you give me more explanations, please?

Idowu • Sep 18

Well explained

Code of Conduct • Report abuse

Khalil

Computer engineering student. 🧢 lover. He/Him.

LOCATION
Tunisia

JOINED
Aug 23, 2020

Trending on DEV Community 🔥

AWS ECS: How does it work?
#aws #tutorial #cloud #docker

Music Monday — What are you listening to? (Video Game Soundtrack Edition 👾)
#watercooler #music #discuss