Introduction to Programming

Digression - I
Compiler, Linker and Loader

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

There is a hint there after “error”… there is something called `ld` that has returned some exit status

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

There is a hint there after “error”… there is something called `ld` that has returned some exit status

Remember that the last statement of our program is `return 0;`
Here, `0` is the “exit status” of our program

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

There is a hint there after “error”… there is something called `ld` that has returned some exit status

Remember that the last statement of our program is `return 0;`
Here, `0` is the “exit status” of our program

By convention, programs return an exit status of `0` to signify a successful completion

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

There is a hint there after “error”… there is something called `ld` that has returned some exit status

Remember that the last statement of our program is
`return 0;`
Here, `0` is the “exit status” of our program

By convention, programs return an exit status of `0` to signify a successful completion

Anything other than `0`, means that the program ended with some problem

Let us revisit ”that” error !!

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 2$ gcc QuadraticEquationWithRealRootsSolver.c
/tmp/ccqSQGjX.o: In function `main':
QuadraticEquationWithRealRootsSolver.c:(.text+0x3d): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

There is a hint there after “error”… there is something called `ld` that has returned some exit status

Remember that the last statement of our program is `return 0;`
Here, `0` is the “exit status” of our program

By convention, programs return an exit status of `0` to signify a successful completion

Anything other than `0`, means that the program ended with some problem

Developers may use different exit status values to indicate different types of error conditions !!

So what is `ld`?

`ld` is a *Linker* used by gcc

So what is `ld`?

`ld` is a *Linker* used by gcc

A Linker is a software tool that “links” different “object codes” into one executable

So what is `ld`?

`ld` is a *Linker* used by gcc

A Linker is a software tool that “links” different “object codes” into one executable

Object code is machine-dependent code, that can be executed

So what is `ld`?

`ld` is a *Linker* used by gcc

A Linker is a software tool that “links” different “object codes” into one executable

Object code is machine-dependent code, that can be executed

The argument `-lm` is actually meant for the Linker, not the Compiler

So what is `ld`?

`ld` is a *Linker* used by `gcc`

A Linker is a software tool that “links” different “object codes” into one executable

Object code is machine-dependent code, that can be executed

The argument `-lm` is actually meant for the Linker, not the Compiler

Here, `-l` is the switch, and `m` stands for the Math library

- This library contains the object code for the `sqrt()` library function

So what is `ld`?

`ld` is a *Linker* used by gcc

A Linker is a software tool that “links” different “object codes” into one executable

Object code is machine-dependent code, that can be executed

The argument `-lm` is actually meant for the Linker, not the Compiler

Here, `-l` is the switch, and `m` stands for the Math library

- This library contains the object code for the `sqrt()` library function

Remember the term “build”?

- This is basically what happens during a build – collecting “object codes” from different sources
- … and producing one executable program

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

First, your program is “compiled” using a Compiler

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

First, your program is “compiled” using a Compiler

The Compiler is a software tool, that “understands” a particular programming language

- Therefore, Compiler for C, is different than Compiler, for say, C++, or any other language

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

First, your program is “compiled” using a Compiler

The Compiler is a software tool, that “understands” a particular programming language

- Therefore, Compiler for C, is different than Compiler, for say, C++, or any other language

Compiler’s job is to produce object code from the source code (aka program)

- However, almost always, your program may depend on “someone else’s” library

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

First, your program is “compiled” using a Compiler

The Compiler is a software tool, that “understands” a particular programming language

- Therefore, Compiler for C, is different than Compiler, for say, C++, or any other language

Compiler’s job is to produce object code from the source code (aka program)

- However, almost always, your program may depend on “someone else’s” library

After your program’s object code is available, the Linker is invoked

Compilers vs Linkers

So what exactly happens when we invoked `gcc`?

In the background, two distinct steps take place

First, your program is “compiled” using a Compiler

The Compiler is a software tool, that “understands” a particular programming language

- Therefore, Compiler for C, is different than Compiler, for say, C++, or any other language

Compiler’s job is to produce object code from the source code (aka program)

- However, almost always, your program may depend on “someone else’s” library

After your program’s object code is available, the Linker is invoked

The Linker is a software tool, that “puts together”, object codes from various source

- At this step, you will have to provide the libraries, that your code needs

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

What we saw till now, is an example of *static linking*

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

What we saw till now, is an example of *static linking*

In static linking, the Linker finds out the “external” sources of code that your program needs

- In the example, that would be the code for the `sqrt()` function

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

What we saw till now, is an example of *static linking*

In static linking, the Linker finds out the “external” sources of code that your program needs

- In the example, that would be the code for the `sqrt()` function

Then, the Linker produces an executable, by copying the code of external sources

- … as if “you” wrote the code for the `sqrt()` function !!

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

What we saw till now, is an example of *static linking*

In static linking, the Linker finds out the "external" sources of code that your program needs

- In the example, that would be the code for the `sqrt()` function

Then, the Linker produces an executable, by copying the code of external sources

- … as if "you" wrote the code for the `sqrt()` function !!

In dynamic linking, the Linker adds some "information" in the executable about the external code

- … and the library is "loaded" by the Operating System, when the program is to be executed

Static vs Dynamic Linking

The Linker can behave in two different ways, based on its configuration parameters

What we saw till now, is an example of *static linking*

In static linking, the Linker finds out the "external" sources of code that your program needs

- In the example, that would be the code for the `sqrt()` function

Then, the Linker produces an executable, by copying the code of external sources

- … as if "you" wrote the code for the `sqrt()` function !!

In dynamic linking, the Linker adds some "information" in the executable about the external code

- … and the library is "loaded" by the Operating System, when the program is to be executed

This means that the size of the executable is smaller, but the overall execution process is complex

… and what is a Loader then?

The job of a Loader, is to “load” an executable into the main memory

… and what is a Loader then?

The job of a Loader, is to “load” an executable into the main memory

The Loader copies a program’s puts the object code into “appropriate locations” in the main memory

… and what is a Loader then?

The job of a Loader, is to “load” an executable into the main memory

The Loader copies a program’s puts the object code into “appropriate locations” in the main memory

A Loader may also have to perform the “dynamic linking”

- It uses the information stored in the executable, to load any external libraries too at “appropriate locations”

… and what is a Loader then?

The job of a Loader, is to “load” an executable into the main memory

The Loader copies a program’s puts the object code into “appropriate locations” in the main memory

A Loader may also have to perform the “dynamic linking”

- It uses the information stored in the executable, to load any external libraries too at “appropriate locations”

The Loader is a part of the Operating System

- Thus, it needs information in a format it can “understand”
- This is why, an executable is “platform dependent”
- … e.g. the `a.out` that was produced on a Linux system, cannot be executed on Windows

Additional Reading

Although it may be premature, reading Chapter 2 and 3 on this guide may be insightful
https://www.linuxtopia.org/online_books/an_introduction_to_gcc/index.html