Hello,

This video explains some useful tools for writing C programs in Linux.

This lecture will cover the following topics:

Why is C a good language to learn?

An overview programming tools in Linux.

Then, this lecture focuses on three tools: G. C. C. .

G D. B. .

and G. C. O. V. .

C was created in the early 1970s and it is indeed quite old, compared with many other languages. The Institute of Electrical and Electronics Engineers, or I. triple E., publishes a survey of programming languages every year. C is always near the top. This is the top programming languages in 2019.

As you can see, 3 of the top 5 languages are C + +, Java, C. .

All of them are based on C. .

Based on this report, C is still a very important programming language.

Linux has many programming tools for C. .

Here are a few examples.

G C. C. is a compiler and its function is to convert source code in the format readable by humans to executable in the format readable by machines.

G D. B. is an interactive debugger. It allows you to check the states of a program during execution. It allows you to inspect the flow of a program.

The data display debugger provides a graphical user interface for G. D. B. . It allows you to visualize data structures.

Valgrind can detect memory errors, for example, when your program intends to read from or write to invalid memory locations. It also detects memory that has been allocated but not released.

Another tool is the integrated development environment called eclipse.

Here are only a small sample of the tools for writing C programs. There are many more other tools.

Now, let me show you how these tools can be useful in writing C programs.

First, let's consider a very simple C program. This program prints 0, 1, 2, to 9.

This is the source code of the program.

Source code is a text file. A text file is generated by human programmer typing in a text editor. This format is not understood by computers as a computer program. The text file has to go through a process called compilation to generate an executable. The executable can run as a program.

A popular compiler is called G. C. C. It means G. N. U. C compiler.

G C. C. needs several inputs. The first input is the name of file. It is the source code. Suppose it is called my program dot c.

minus O specifies the name of the output file. This is the executable file. In this example, the executable's name is P. R. O. G. .

By convention, the name of an executable has no extension, no dot E. X. E. .

It is advised not to call the executable test because test is also a Linux command. If you call your program test, it is confusing which test you are running. Is it your program, or the Linux command?

There is one way to specify which program to run by adding dot slash in front. This means running the program in this particular directory.

The next example prints the 5 times 5 multiplication table. This program is fairly easy. You need two for loops with different variables. Print the product of these two variables.

Suppose you write a program and it prints only five zeros. If you put the correct and the wrong program side by side, can you easily tell what is wrong?

What's wrong is that the two for loops use the same control variable. The intention is to use i and j. However, i is used twice. This is not easy to detect by eyes.

One solution is to take advantage of the warning function of G. C. C. .

By adding minus W. all after G.C. C., G.C.C. tries to find code that is likely to be problematic. In this example, G.C.C. finds that the value of j is used without initialization and issues a warning message.

This warning message is helpful because it directs attention to the code that is likely wrong.

It is very important to take advantage of G. C. C. 's warning messages. A major advantage of these warning messages is that they do not depend on test inputs. You can use G. C. C. to detect problems before you complete the entire program.

A general rule in writing computer programs is the following: It is better if you can detect problems earlier. If you wait until the program has been written and detect problems by testing, usually much more effort is needed to fix the problems.

The next topic is G. D. B. .

It is an interactive debugger. It has many functions. The most commonly used functions are breakpoints, printing, and seeing the stack memory.

Let's consider this program as the example. Let's start from the main function. It takes two arguments for two numbers called a and b. Then, the main function calls function f two.

Function two takes two integers a and b and then calls function one. If function one returns a value greater than zero, function two returns a minus b. If function one returns a value not greater than zero, function two returns a plus b.

To run this program, we first need to compile it. This time we add minus G. to enable debugging.

G.D. B. takes the name of an executable, not the source file. The reason is that an executable may be created by using multiple source files. G.D.B. automatically finds the right source files, if you have added minus G. after G. C. C. .

You can set a breakpoint by using the command B. followed by the name of a function. In this example, a breakpoint is set at the main function.

You can also set a breakpoint by using the command B. followed by the line number of the source file. In this example, a breakpoint is set at the eleventh line.

To run the program, use the R. command. If the program takes input arguments, add the arguments after R. In this example, 3 and 5 are the input arguments.

The program will start and stop at the first line of the main function.

G.D. B. tells you that the program has stopped at line 21. G.D.B. stops before executing this line.

The list command lists 10 lines of the code around the breakpoint.

Let us review the three G. D.B. commands:

B sets a breakpoint. You can set a breakpoint by giving the name of a function or the line number.

R starts running the program. If the program needs arguments, add the arguments after R. .

list prints 10 lines of the program.

Another way to set a breakpoint is by using the file name followed by colon and line number. In this example, a breakpoint is set to the third line of the source file my program dot c.

C is the command to continue the program from the current location to the next breakpoint. This program stops at line 13 because we set a breakpoint there earlier.

The print command shows the value of a variable.

If you want to see the call stack, use the B.T. command. It means back trace.

The currently running function is always frame zero.

The call stack is useful when you want to know how to get to this place in the program.

It shows the line numbers.

Currently, the program stops at the thirteenth line of function F. two.

Function F. two is called by the main function at line 27.

The C command continues the program to the next breakpoint.

The B. T. command now shows three functions: at line 27, main function calls F. two. At line 13, function two calls function one. The program stops at the fifth line of function one.

When main function calls function two, the arguments a and b are 3 and 5 respectively.

When function two calls function one, the arguments a and b are also 3 and 5 respectively.

Let's review three new commands in G.D.B.:

print: it prints the value of a variables.

c: the command continues the program to the next breakpoint.

bt: means backtrace and it shows the call stack.

The last topic in this video is test coverage.

What does test coverage mean? When testing a program, it is very common that the test cases do not cover every line of code. As the program becomes more complex, it becomes increasingly difficult to know which lines of code have not been tested. Test coverage tells you which lines of code have been tested and which lines have not been tested.

To get the information about test coverage, add minus F test coverage and minus F. profile arcs after G.C.C. . They enable test coverage.

Then, execute the program.

Run G. C. O.V. followed by the file name of the source code.

Only 20% of the code has been tested.

Three additional files will be generated. They have extensions of G.C.O.V.

G.C.D.A

and

G.C.N.O.

.

The file endings with G.C.O.V. shows which lines have been tested. Some lines start with the # . These lines have not been tested.

A few lines start with one, meaning that these lines have been tested once.

Since no argument is given to the program, the condition at line 21 is true and the program stops at line 23.

Next, run the program with two arguments 6 and 2.

Use G.C.O.V. again. This time, 86.67% code has been tested.

Again, # still means the lines have not been tested. As you can see many lines have been tested with numbers 1 or 2 in front.

Number 1 means the line has been tested once.

Number 2 means the line has been tested twice.

These numbers accumulate as you run multiple test cases.

After you run several test cases, you can check which lines have not been tested. Test coverage helps you identify the code that has not been tested.

This is the review of the video. This video explains why C is still an important language to learn. Three useful Linux commands are introduced. They are G.C.C. .

G.D.B.

and G.C.O.V. .

Enjoy C Programming!