Compile, Run and Debug

周嘉鑫, 李阳

March 10, 2020

Contents

What's a compiler?

- Simply stated, a compiler is a program that can read a program in one language - the *source* language - and translate it into an equivalent program in another language - the *target* language;
- An important role of the compiler is to report any errors in the source program that it detects during the translation process.

What's a compiler?

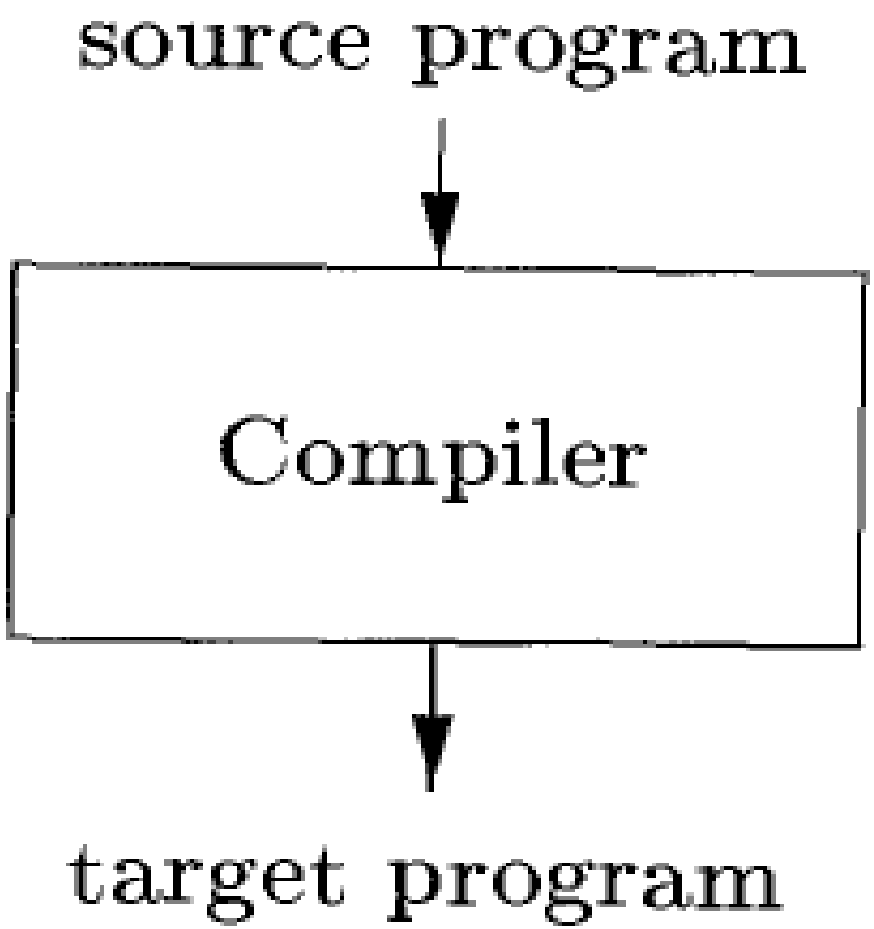

图: Compiler

The compiling processes

There are usually four steps, preprocessing, compilation, assembly and linking.

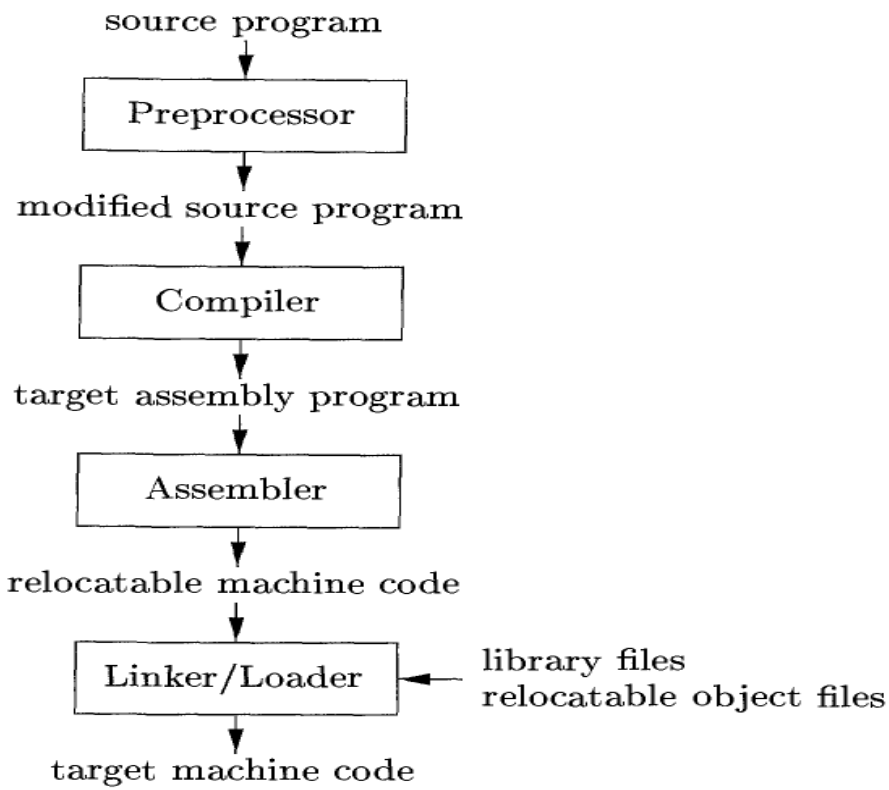

图: The compiling processes

GCC, the GNU Compiler Collection

- We run gcc to compile C language programs;
- We run g++ to compile C++ language programs.

The input file suffixes

For any given input file, the file name suffix determines what kind of compilation is done:

- **file.c** C source code that must be preprocessed.
- **file.i** C source code that should not be preprocessed.
- **file.ii** C++ source code that should not be preprocessed.
- **file.h** C, C++ header file to be turned into a precompiled header.

The input file suffixes

- **file.cc,file.cp,file.cxx,file.cpp,file.CPP,file.c++,file.C** C++ source code that must be preprocessed.
- **file.hh,file.H,file.hp,file.hxx,file.hpp,file.HPP,file.h++,file.tcc** C++ header file to be turned into a precompiled header.
- **file.o ...** The linking object file.

Some frequently used options

You can mix options and other arguments. For the most part, the order you use doesn't matter.

- **-c** Stop after the stage of assembly; do not link.
- **-S** Stop after the stage of compilation proper; do not assemble.
- **-E** Stop after the preprocessing stage; do not run the compiler proper.

Some frequently used options

-o file *Place output in file* ***file***.
If **-o file** is not specified, the default outputs are

- **a.out** for an executable file;
- **source.o** for an object file;
- **source.s** for an assembler file;
- **source.suffix.gch** for a precompiled header file.
- All preprocessed C source are put in the standard output.

Some frequently used options

- **-v** Print the commands executed to run the stages of compilation.
- **-g** Generates debug information to be used by GDB debugger.
- **-O*level*** Set the compilers optimization level.
- **-D *name[=value]*** Define a macro to be used by the preprocessor.

Some frequently used options

- **-I*dir*** Add include directory of header files.
- **-L*dir*** Add include directory of libraries.
- **-l*library*** Links the library file.

Debug with GDB

GDB

GDB, the GNU Project debugger, allows you to see what is going on 'inside' another program while it executes – or what another program was doing at the moment it crashed.

Debug with GDB

GDB can do four main kinds of things.

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

Codes with bugs

```c
//buggy.c

#include <stdio.h>
#include <stdlib.h>

int* buggy_fn();
int main(int argc, char *argv[]){
    // atoi() does not check for errors.
    // Watch what happens when we give an error of omission.
    int array_size = atoi(argv[1]);
    int *array;
    array = buggy_fn(array_size);
    int index;
    for(index=0; index<array_size; index++){
        printf("array[index] = %d\n", array[index]);
    }
    return 0;
}
```

图: buggy.c

Codes with bugs

```c
// buggy_fn.c
#include <stdio.h>

//This function returns a pointer to an array
//that was allocated within the function.
//Guess what happens to the array
//when the function returns!

int *buggy_fn(int array_size){

    int index, array[array_size];
    for (index=0; index<array_size; index++){
        array[index] = index;
    }

    int *retval;
    retval = array;
    for (index=0; index<array_size; index++){
        printf("retval[index] = %d\n", retval[index]);
    }
    return retval;
}
```

图: buggyf.c

Run the program

Run

Input these commands at the location of the files.

- *gcc -g -O0 buggy.c buggyf.c -o buggy.ex*
- *./buggy.ex*

We get *segmentation fault*.

Debug with gdb

Input these commands.

- *gdb buggy.ex*
- *run*

We could get more information than the *Segmentation fault*.

Debug with gdb

Input these commands.

- *backtrace*

This command detects where the fault was occurred in our code.

Debug with gdb

Input these commands.

- *break buggy.c:10*
- *finish*
- *run*

Set a breakpoint and run the program.

Debug with gdb

Input these commands.

- *print argv[1]*

We get *$1 = 0x0*.

- *set argv[1]="4"*

We see the value of the variable *argv[1]*, and the value is stored in the *$1*.

Debug with gdb

Input these commands.

- *break buggy.c:11*
- *continue*

Set a breakpoint after the function *atoi*.

Debug with gdb

Input these commands.

- *p array_size*

We see the value of the variable *array_size* equals to 4.

Debug with gdb

Input these commands.

- *continue*
- *delete breakpoints*
- *run 4*

We may see different outputs. Try *run 4*, *run 10*, *run 100*.

Debug with gdb

Input these commands.

- *break 15*
- *run 100*
- *print array[0]@100*
- *continue*
- *print array[0]@100*
- *continue*
- *display array[0]@100*
- *continue*

We will see the values of *array* at every breakpoint. *delete display* could be used to delete the *display* expressions.

Frequently used commands

- *gdb program* Start GDB with program.
- *quit(q)* End GDB.
- *run [arglist]* Run the program with *arglist*.
- *start [arglist]* Run the program with *arglist* and stop at the first line.
- *backtrace(bt)* Display the program stack.

Frequently used commands

- *set [arg=][value]* Set the value of an argument.
- *list(l) [file:][linum]* Display the text of the program.
- *next(n)* Execute next program line; step over any function calls.
- *step(s)* Execute next program line; step into any function calls.
- *print(p) [expr]* Display the value of an expression.
- *display [expr]* Display the value of an expression at every breakpoint.

Frequently used commands

- *break(b) [file:][linum]* Set a breakpoint in the file.
- *break(b) [file:][linum] [if expr]* Break if a condition is satisfied.
- *info(i) [var]* Print the information of the variable.
- *continue(c)* Continue running your program.
- *delete(d) expr* Delete the breakpoints or display expressions, etc.
- *help [name]* Show information about GDB command.

References

- GNU GCC Manual
- GNU GDB Manual
- Beginning Linux Programming
- Scientific Computing with Machine
- Compilers principles, techniques ,tools