C Program Compilation Steps

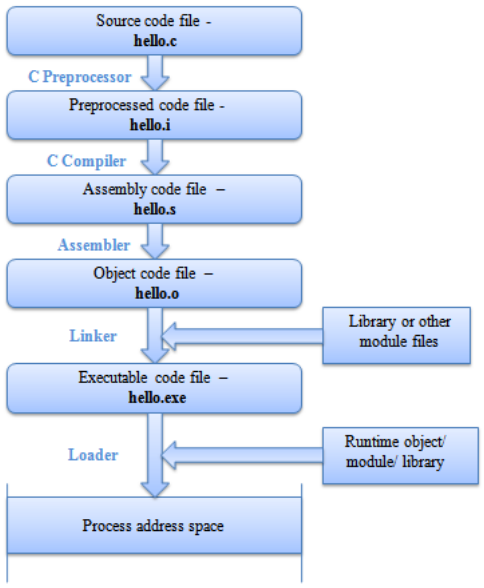
You compile c program and get executables. Have you ever wondered what happens during compilation process and how c program gets converted to executable?

In this module we will learn what are the stages involved in c program compilation using gcc on Linux.

Normally C program building process involves four stages to get executable (.exe)

1. Preprocessing
2. Compilation
3. Assembly
4. Linking

The following Figure shows the steps involved in the process of building the C program starting from the preprocessing until the loading of the executable image into the memory for program running.



Compilation with gcc with different options

-E Preprocess only; do not compile, assemble or link

-S Compile only; do not assemble or link

-c Compile and assemble, but do not link

-o <file> Place the output into <file>

We will use below **hello.c** program to expain all the 4 phases

```
#include<stdio.h>        //Line 1
#define MAX_AGE  21   //Line 2
int main()
{
 printf( "Maximum age : %d ",MAX_AGE); //Line 5
}
```

1. Preprocessing

This is the very first stage through which a source code passes. In this stage the following tasks are done:

1. Macro substitution
2. Comments are stripped off
3. Expansion of the included files

To understand preprocessing better, you can compile the above 'hello.c' program using flag –E with gcc. This will generate the preprocessed hello.i

Example:

>gcc -E hello.c -o hello.i

//hello.i file content

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 28 "/usr/include/stdio.h" 3 4
……
……
Truncated some text…
……
……
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
# 918 "/usr/include/stdio.h" 3 4

# 2 "hello.c" 2

int main()
{
 printf( "Maximum age : %d ",21);
}
```

In above code (hello.i) you can see macros are substituted with its value (MA_AGE with 21 in printf statement), comments are stripped off (//Line 1, //Line 2 and //Line 5)and libraries are expanded(<stdio.h>)

2. Compilation

Compilation is the second pass. It takes the output of the preprocessor (hello.i) and generates assembler source code (hello.s)

> gcc -S hello.i -o hello.s

//hello.s file content

```
.file   "hello.c"
        .section        .rodata
.LC0:
        .string "Maximum age : %d "
        .text
.globl main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        movq    %rsp, %rbp
        .cfi_offset 6, -16
        .cfi_def_cfa_register 6
        movl    $.LC0, %eax
        movl    $21, %esi
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.2 20091027 (Red Hat 4.4.2-7)"
        .section        .note.GNU-stack,"",@progbits
```

Above code is assembly code which assembler can understand and generate machine code.

3. Assembly

Assembly is the third stage of compilation. It takes the assembly source code (hello.s) and produces an assembly listing with offsets. The assembler output is stored in an object file (hello.o)

>gcc -c hello.s -o hello.o

Since the output of this stage is a machine level file (hello.o). So we cannot view the content of it. If you still try to open the hello.o and view it, you'll see something that is totally not readable

//hello.o file content

```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@@^A^@^@^@^@^@^@^@^@^@^@@^@^@^@^@^@@^@^M^@^@UH<89>å¸
^@^@^@^@¾^U^@^@^@H<89>ç¸^@^@^@^@è^@^@^@^@éã^@^@^@Maximum age :%d
 ^@^@GCC:GNU)4.4.220091027(RedHat4.4.2-7)^@^@^T^@^@^@^@^@^@^@^AzR^
@^Ax^P^A^[^L^G^H<90>^A^@^@^\^@^@^@^\^@^@^@^@^@^@^]^@^@^@^@A^N^PC
<86>^B^M^FX^L^G^H^@^@^@^@.symtab^@.strtab^@.shstrtab^@.rela.text^@
.data^@.bss^@.rodata^@.comment^@.note.GNU-stack^@.rela.eh_frame^@
^@^@^@^@^@^@^@^@^@^@^
```

By looking at above code only thing we can explain is ELF (executable and linkable format). This is a relatively new format for machine level object files and executable that are produced by gcc.

4. Linking

Linking is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single executable file (hello.exe). In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

> gcc hello.o -o hello

./hello

Maximum age : 21

Now you know c program compilation steps (Preprocessing, Compiling, Assembly, and Linking). There is lot more things to explain in liking phase.

Hide Page Information

c program compilation steps or phases or process in detail.