## 1. HelloWorld背后没那么简单

### 1.1 交叉编译hello.c

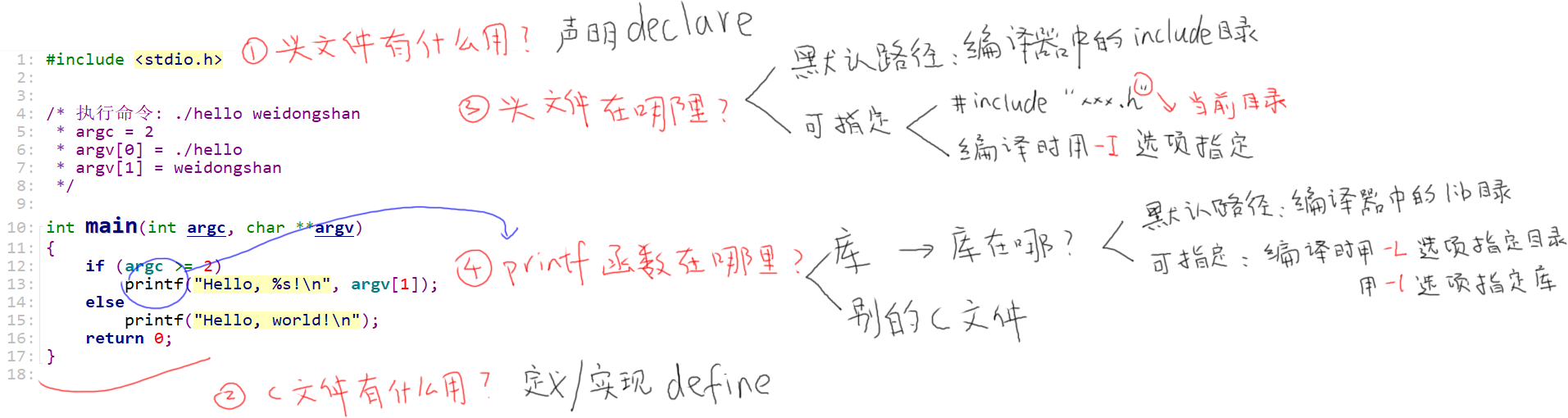
准备工作：

① 解压工具链、设置PATH环境变量，确定编译器名称；然后才可以编译。

② 要在板子上运行，使用NFS会比较方便。

这2点，都可以参考开发板的高级用户使用手册。

### 1.2 请回答这几个问题



① 怎么确定交叉编译器中头文件的默认路径？

进入交叉编译器的目录里，执行：find -name “stdio.h”，它位于一个“include”目录下的根目录里。

这个“include”目录，就是要找的路径。

② 怎么自己指定头文件目录？

编译时，加上“-I <头文件目录>”这样的选项。

③ 怎么确定交叉编译器中库文件的默认路径？

进入交叉编译器的目录里，执行：find -name lib，可以得到xxxx/lib、xxxx/usr/lib，一般来说这2个目录就是要找的路径。

如果有很多类似的lib，进去看看，有很多so文件的目录一般就是要找的路径。

④ 怎么自己指定库文件目录、指定要用的库文件？

编译时，加上“-L <库文件目录>”这样的选项，用来指定库目录；

编译时，加上“-labc”这样的选项，用来指定库文件libabc.so。

### 1.3 演示

## 2. GCC编译器的使用

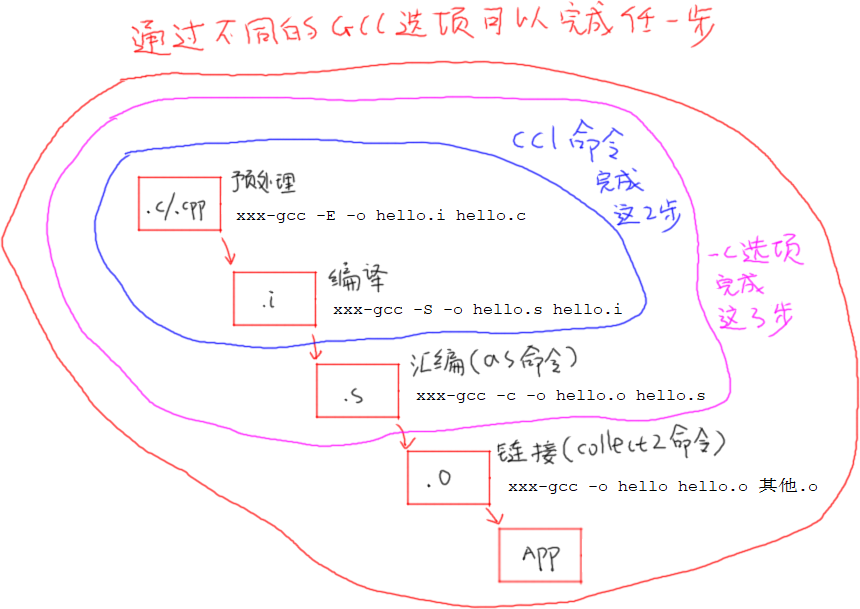
源文件需要经过编译才能生成可执行文件。在Windows下进行开发时，只需要点几个按钮即可编译，集成开发环境(比如Visual studio)已经将各种编译工具的使用封装好了。Linux下也有很优秀的集成开发工具，但是更多的时候是直接使用编译工具；即使使用集成开发工具，也需要掌握一些编译选项。

PC机上的编译工具链为gcc、ld、objcopy、objdump等，它们编译出来的程序在x86平台上运行。要编译出能在ARM平台上运行的程序，必须使用交叉编译工具xxx-gcc、xxx-ld等(不同版本的编译器的前缀不一样，比如arm-linux-gcc)，下面分别介绍。

### 2.0 配套视频内容大纲

#### 2.0.1 GCC编译过程(精简版)

一个C/C++文件要经过预处理(preprocessing)、编译(compilation)、汇编(assembly)和链接(linking)等4步才能变成可执行文件。



在日常交流中通常使用“编译”统称这4个步骤，如果不是特指这4个步骤中的某一个，本教程也依惯例使用“编译”这个统称。

cc1 main.c -o /tmp/ccXCx1YG.s

as -o /tmp/ccZfdaDo.o /tmp/ccXCx1YG.s

cc1 sub.c -o /tmp/ccXCx1YG.s

as -o /tmp/ccn8Cjq6.o /tmp/ccXCx1YG.s

collect2 -o test /tmp/ccZfdaDo.o /tmp/ccn8Cjq6.o ....

#### 2.0.2 常用编译选项

在学习时，我们暂时只需要了解下表中的选项。

|  |  |
| --- | --- |
| 常用选项 | 描述 |
| -E | 预处理，开发过程中想快速确定某个宏可以使用“-E -dM” |
| -c | 把预处理、编译、汇编都做了，但是不链接 |
| -o | 指定输出文件 |
| -I | 指定头文件目录 |
| -L | 指定链接时库文件目录 |
| -l | 指定链接哪一个库文件 |

#### 2.0.3 怎么编译多个文件

① 一起编译、链接：

gcc -o test main.c sub.c

② 分开编译，统一链接：

gcc -c -o main.o main.c

gcc -c -o sub.o sub.c

gcc -o test main.o sub.o

#### 2.0.4 制作、使用动态库

制作、编译：

gcc -c -o main.o main.c

gcc -c -o sub.o sub.c

gcc -shared -o libsub.so sub.o ~~sub2.o sub3.o(~~可以使用多个.o生成动态库~~)~~

gcc -o test main.o -lsub -L /libsub.so/所在目录/

运行：

① 先把libusb.so放到PC或板子上的/lib目录，然后就可以运行test程序。

② 如果不想把libusb.so放到/lib，也可以放在某个目录比如/a，然后如下执行：

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/a

./test

#### 2.0.5 制作、使用静态库

gcc -c -o main.o main.c

gcc -c -o sub.o sub.c

ar crs libsub.a sub.o ~~sub2.o sub3.o(~~可以使用多个.o生成静态库~~)~~

gcc -o test main.o libsub.a (如果.a不在当前目录下，需要指定它的绝对或相对路径)

运行：

不需要把静态库libsub.a放到板子上。

#### 2.0.6 很有用的选项

gcc -E main.c // 查看预处理结果，比如头文件是哪个

gcc -E -dM main.c > 1.txt // 把所有的宏展开，存在1.txt里

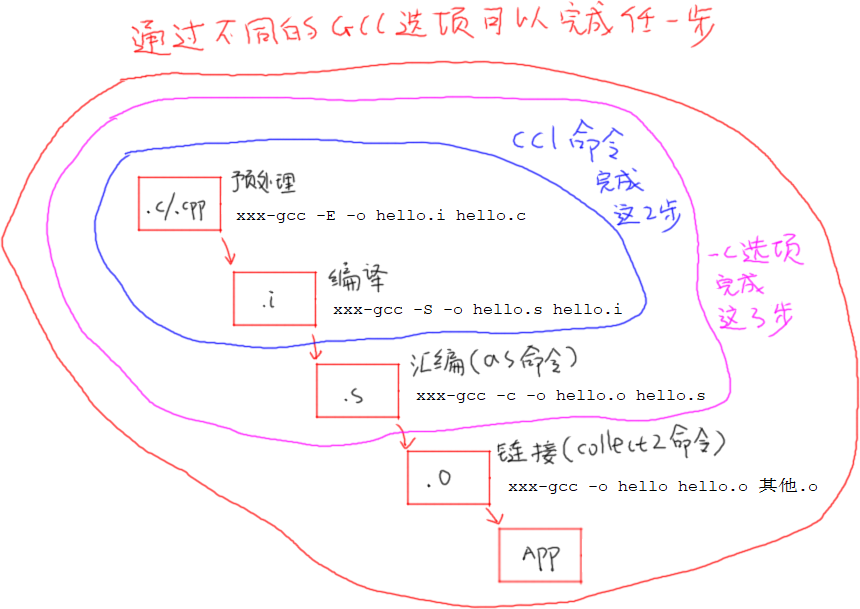
gcc -Wp,-MD,abc.dep -c -o main.o main.c // 生成依赖文件abc.dep，后面Makefile会用

我们的视频会配套写成书：《嵌入式Linux应用开发完全手册 升级版》。下面的资料会写进书里，我会写得详细一点。

下面的资料来自GCC官方文档及一些中文资料，没必要逐一研读，用到时再翻翻。

### 2.1 GCC编译过程

一个C/C++文件要经过预处理(preprocessing)、编译(compilation)、汇编(assembly)和链接(linking)等4步才能变成可执行文件。



在日常交流中通常使用“编译”统称这4个步骤，如果不是特指这4个步骤中的某一个，本教程也依惯例使用“编译”这个统称。

本节文档使用x86上的gcc来试验，使用ARM板的交叉编译工具链做实验时效果也是类似的。不同的交叉编译器工具链前缀可能不同，比如arm-linux-gcc。

（1）预处理

C/C++源文件中，以“#”开头的命令被称为预处理命令，如包含命令“#include”、宏定义命令“#define”、条件编译命令“#if”、“#ifdef”等。预处理就是将要包含(include)的文件插入原文件中、将宏定义展开、根据条件编译命令选择要使用的代码，最后将这些东西输出到一个“.i”文件中等待进一步处理。

（2）编译

编译就是把C/C++代码(比如上述的“.i”文件)“翻译”成汇编代码，所用到的工具为cc1(它的名字就是cc1，x86有自己的cc1命令，ARM板也有自己的cc1命令)。

（3）汇编

汇编就是将第二步输出的汇编代码翻译成符合一定格式的机器代码，在Linux系统上一般表现为ELF目标文件(OBJ文件)，用到的工具为as。x86有自己的as命令，ARM版也有自己的as命令，也可能是xxxx-as（比如arm-linux-as）。

“反汇编”是指将机器代码转换为汇编代码，这在调试程序时常常用到。

（4）链接

链接就是将上步生成的OBJ文件和系统库的OBJ文件、库文件链接起来，最终生成了可以在特定平台运行的可执行文件，用到的工具为ld或collect2。

编译程序时，加上-v选项就可以看到这几个步骤。比如：

gcc -o hello hello.c -v

可以看到很多输出结果，我们把其中的主要信息摘出来：

cc1 hello.c -o /tmp/cctETob7.s

as -o /tmp/ccvv2KbL.o /tmp/cctETob7.s

collect2 -o hello crt1.o crti.o crtbegin.o /tmp/ccvv2KbL.o crtend.o crtn.o

以上3个命令分别对应于编译步骤中的预处理+编译、汇编和链接，ld被collect2调用来链接程序。预处理和编译被放在了一个命令（cc1）中进行的，可以把它再次拆分为以下两步：

cpp -o hello.i hello.c

cc1 hello.i -o /tmp/cctETob7.s

我们不需要手工去执行cpp、cc1、collect2等命令，我们直接执行gcc并指定不同的参数就可以了。

可以通过各种选项来控制gcc的动作，下面介绍一些常用的选项。

|  |  |
| --- | --- |
| 常用选项 | 描述 |
| -E | 预处理，开发过程中想快速确定某个宏可以使用“-E -dM” |
| -c | 把预处理、编译、汇编都做了，但是不链接 |
| -o | 指定输出文件 |
| -I | 指定头文件目录 |
| -L | 指定链接时库文件目录 |
| -l | 指定链接哪一个库文件 |

### 2.2 GCC总体选项(Overall Option)

（1）-c

预处理、编译和汇编源文件，但是不作链接，编译器根据源文件生成OBJ文件。缺省情况下，GCC通过用`.o'替换源文件名的后缀`.c'，`.i'，`.s'等，产生OBJ文件名。可以使用-o选项选择其他名字。GCC忽略-c选项后面任何无法识别的输入文件。

（2）-S

编译后即停止，不进行汇编。对于每个输入的非汇编语言文件，输出结果是汇编语言文件。缺省情况下，GCC通过用`.s'替换源文件名后缀`.c'，`.i'等等，产生汇编文件名。可以使用-o选项选择其他名字。GCC忽略任何不需要汇编的输入文件。

（3）-E

预处理后即停止，不进行编译。预处理后的代码送往标准输出。

（4）-o file

指定输出文件为file。无论是预处理、编译、汇编还是链接，这个选项都可以使用。如果没有使用`-o'选项，默认的输出结果是：可执行文件为`a.out'；修改输入文件的名称是`source.suffix'，则它的OBJ文件是`source.o'，汇编文件是 `source.s'，而预处理后的C源代码送往标准输出。

（5）-v

显示制作GCC工具自身时的配置命令；同时显示编译器驱动程序、预处理器、编译器的版本号。

以一个程序为例，它包含三个文件，代码在02\_options目录下。下面列出源码：

File: main.c

01 #include <stdio.h>

02 #include "sub.h"

03

04 int main(int argc, char \*argv[])

05 {

06 int i;

07 printf("Main fun!\n");

08 sub\_fun();

09 return 0;

10 }

11

File: sub.h

01 void sub\_fun(void);

02

File: sub.c

01 void sub\_fun(void)

02 {

03 printf("Sub fun!\n");

04 }

05

ARM版本的编译工具与gcc、ld等工具的使用方法相似，很多选项是一样的。本节使用gcc、ld等工具进行编译、链接，这样可以在PC上直接看到运行结果。使用上面介绍的选项进行编译，命令如下：

$ gcc -c -o main.o main.c

$ gcc -c -o sub.o sub.c

$ gcc -o test main.o sub.o

其中，main.o、sub.o是经过了预处理、编译、汇编后生成的OBJ文件，它们还没有被链接成可执行文件；最后一步将它们链接成可执行文件test，可以直接运行以下命令：

$ ./test

Main fun!

Sub fun!

现在试试其他选项，以下命令生成的main.s是main.c的汇编语言文件：

$ gcc -S -o main.s main.c

以下命令对main.c进行预处理，并将得到的结果打印出来。里面扩展了所有包含的文件、所有定义的宏。在编写程序时，有时候查找某个宏定义是非常繁琐的事，可以使用`-dM –E’选项来查看。命令如下：

$ gcc -E main.c

### 2.3 警告选项(Warning Option)

（1）-Wall

这个选项基本打开了所有需要注意的警告信息，比如没有指定类型的声明、在声明之前就使用的函数、局部变量除了声明就没再使用等。

上面的main.c文件中，第6行定义的变量i没有被使用，但是使用“gcc –c –o main.o main.c”进行编译时并没有出现提示。

可以加上-Wall选项，例子如下：

$ gcc -Wall -c main.c

执行上述命令后，得到如下警告信息：

main.c: In function `main':

main.c:6: warning: unused variable `i'

这个警告虽然对程序没有坏的影响，但是有些警告需要加以关注，比如类型匹配的警告等。

### 2.4 调试选项(Debugging Option)

（1）-g

以操作系统的本地格式(stabs，COFF，XCOFF，或DWARF)产生调试信息，GDB能够使用这些调试信息。在大多数使用stabs格式的系统上，`-g'选项加入只有GDB才使用的额外调试信息。可以使用下面的选项来生成额外的信息：`-gstabs+'，`-gstabs'，`-gxcoff+'，`-gxcoff'，`-gdwarf+'或`-gdwarf'，具体用法请读者参考GCC手册。

### 2.5 优化选项(Optimization Option)

（1）-O或-O1

优化：对于大函数，优化编译的过程将占用稍微多的时间和相当大的内存。不使用`-O'或`-O1'选项的目的是减少编译的开销，使编译结果能够调试、语句是独立的：如果在两条语句之间用断点中止程序，可以对任何变量重新赋值，或者在函数体内把程序计数器指到其他语句，以及从源程序中精确地获取你所期待的结果。

不使用`-O'或`-O1'选项时，只有声明了register的变量才分配使用寄存器。

使用了`-O'或`-O1'选项，编译器会试图减少目标码的大小和执行时间。如果指定了`-O'或`-O1'选项,，`-fthread-jumps'和`-fdefer-pop'选项将被打开。在有delay slot的机器上，`-fdelayed-branch'选项将被打开。在即使没有帧指针 (frame pointer)也支持调试的机器上，`-fomit-frame-pointer'选项将被打开。某些机器上还可能会打开其他选项。

（2）-O2

多优化一些。除了涉及空间和速度交换的优化选项，执行几乎所有的优化工作。例如不进行循环展开(loop unrolling)和函数内嵌(inlining)。和`-O'或`-O1'选项比较，这个选项既增加了编译时间，也提高了生成代码的运行效果。

（3）-O3

优化的更多。除了打开-O2所做的一切，它还打开了-finline-functions选项。

（4）-O0

不优化。

如果指定了多个-O选项，不管带不带数字，生效的是最后一个选项。

在一般应用中，经常使用-O2选项，比如对于options程序：

$ gcc -O2 -c -o main.o main.c

$ gcc -O2 -c -o sub.o sub.c

$ gcc -o test main.o sub.o

### 2.6 链接器选项(Linker Option)

下面的选项用于链接OBJ文件，输出可执行文件或库文件。

（1）object-file-name

如果某些文件没有特别明确的后缀(a special recognized suffix)，GCC就认为他们是OBJ文件或库文件(根据文件内容,链接器能够区分OBJ文件和库文件)。如果GCC执行链接操作，这些OBJ文件将成为链接器的输入文件。

比如上面的“gcc -o test main.o sub.o”中，main.o、sub.o就是输入的文件。

（2）-llibrary

链接名为library的库文件。

链接器在标准搜索目录中寻找这个库文件，库文件的真正名字是`liblibrary.a'。搜索目录除了一些系统标准目录外，还包括用户以`-L'选项指定的路径。一般说来用这个方法找到的文件是库文件──即由OBJ文件组成的归档文件(archive file)。链接器处理归档文件的方法是：扫描归档文件，寻找某些成员，这些成员的符号目前已被引用，不过还没有被定义。但是，如果链接器找到普通的OBJ文件，而不是库文件，就把这个OBJ文件按平常方式链接进来。指定`-l'选项和指定文件名的唯一区别是，`-l’选项用`lib'和`.a'把library包裹起来，而且搜索一些目录。

即使不明显地使用-llibrary选项，一些默认的库也被链接进去，可以使用-v选项看到这点：

$ gcc -v -o test main.o sub.o

输出的信息如下：

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/collect2 --eh-frame-hdr -m elf\_i386 -dynamic-linker /lib/ld-linux.so.2

-o test

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../../crt1.o /usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../../crti.o

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/crtbegin.o

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../..

main.o

sub.o

-lgcc -lgcc\_eh -lc -lgcc -lgcc\_eh

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/crtend.o

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../../crtn.o

可以看见，除了main.o、sub.o两个文件外，还链接了启动文件crt1.o、crti.o、crtend.o 、crtn.o，还有一些库文件(-lgcc -lgcc\_eh -lc -lgcc -lgcc\_eh)。

（3）-nostartfiles

不链接系统标准启动文件，而标准库文件仍然正常使用：

$ gcc -v -nostartfiles -o test main.o sub.o

输出的信息如下：

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/collect2 --eh-frame-hdr -m elf\_i386 -dynamic-linker

/lib/ld-linux.so.2

-o test

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../..

main.o

sub.o

-lgcc -lgcc\_eh -lc -lgcc -lgcc\_eh

/usr/bin/ld: warning: cannot find entry symbol \_start; defaulting to 08048184

可以看见启动文件crt1.o、crti.o、crtend.o 、crtn.o没有被链接进去。需要说明的是，对于一般应用程序，这些启动文件是必需的，这里仅是作为例子(这样编译出来的test文件无法执行)。在编译bootloader、内核时，将用到这个选项。

（4）-nostdlib

不链接系统标准启动文件和标准库文件，只把指定的文件传递给链接器。这个选项常用于编译内核、bootloader等程序，它们不需要启动文件、标准库文件。

仍以options程序作为例子：

$ gcc -v -nostdlib -o test main.o sub.o

输出的信息如下：

/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/collect2 --eh-frame-hdr -m elf\_i386 -dynamic-linker /lib/ld-linux.so.2

-o test

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2

-L/usr/lib/gcc-lib/i386-redhat-linux/3.2.2/../../..

main.o

sub.o

/usr/bin/ld: warning: cannot find entry symbol \_start; defaulting to 08048074

main.o(.text+0x19): In function `main':

: undefined reference to `printf'

sub.o(.text+0xf): In function `sub\_fun':

: undefined reference to `printf'

collect2: ld returned 1 exit status

出现了一大堆错误，因为printf等函数是在库文件中实现的。在编译bootloader、内核时，用到这个选项──它们用到的很多函数是自包含的。

（5）-static

在支持动态链接(dynamic linking)的系统上，阻止链接共享库。

仍以options程序为例，是否使用-static选项编译出来的可执行程序大小相差巨大：

$ gcc -c -o main.c

$ gcc -c -o sub.c

$ gcc -o test main.o sub.o

$ gcc -o test\_static main.o sub.o –static

$ ls -l test test\_static

-rwxr-xr-x 1 book book 6591 Jan 16 23:51 test

-rwxr-xr-x 1 book book 546479 Jan 16 23:51 test\_static

其中test文件为6591字节，test\_static文件为546479字节。当不使用-static编译文件时，程序执行前要链接共享库文件，所以还需要将共享库文件放入文件系统中。

（6）-shared

生成一个共享OBJ文件，它可以和其他OBJ文件链接产生可执行文件。只有部分系统支持该选项。

当不想以源代码发布程序时，可以使用-shared选项生成库文件，比如对于options程序，可以如下制作库文件：

$ gcc -c -o sub.o sub.c

$ gcc -shared -o libsub.so sub.o

以后要使用sub.c中的函数sub\_fun时，在链接程序时，指定引脚libsub.so即可，比如：

$ gcc -o test main.o -lsub -L /libsub.so/所在的目录/

可以将多个文件制作为一个库文件，比如：

$ gcc -shared -o libsub.so sub.o sub2.o sub3.o

（7）-Xlinker option

把选项option传递给链接器。可以用来传递系统特定的链接选项，GCC无法识别这些选项。如果需要传递携带参数的选项，必须使用两次`-Xlinker'，一次传递选项，另一次传递其参数。例如，如果传递`-assert definitions'，要成`-Xlinker -assert -Xlinker definitions'，而不能写成`-Xlinker "-assert definitions"'，因为这样会把整个字符串当做一个参数传递，显然这不是链接器期待的。

（8）-Wl,option

把选项option传递给链接器。如果option中含有逗号，就在逗号处分割成多个选项。链接器通常是通过gcc、arm-linux-gcc等命令间接启动的，要向它传入参数时，参数前面加上`-Wl,’。

（9）-u symbol

使链接器认为取消了symbol的符号定义，从而链接库模块以取得定义。可以使用多个 `-u'选项，各自跟上不同的符号，使得链接器调入附加的库模块。

### 2.7 目录选项(Directory Option)

下列选项指定搜索路径，用于查找头文件，库文件，或编译器的某些成员。

（1）-Idir

在头文件的搜索路径列表中添加dir 目录。

头文件的搜索方法为：如果以“#include < >”包含文件，则只在标准库目录开始搜索(包括使用-Idir选项定义的目录)；如果以“#include “ ””包含文件，则先从用户的工作目录开始搜索，再搜索标准库目录。

（2）-I-

任何在`-I-'前面用`-I'选项指定的搜索路径只适用于`#include "file"'这种情况；它们不能用来搜索`#include <file>'包含的头文件。如果用`-I'选项指定的搜索路径位于`-I-'选项后面，就可以在这些路径中搜索所有的`#include'指令(一般说来-I选项就是这么用的)。还有，`-I-'选项能够阻止当前目录(存放当前输入文件的地方)成为搜索`#include "file"'的第一选择。

`-I-'不影响使用系统标准目录，因此，`-I-'和`-nostdinc'是不同的选项。

（3）-Ldir

在`-l'选项的搜索路径列表中添加dir目录。

仍使用options程序进行说明，先制作库文件libsub.a：

$ gcc -c -o sub.o sub.c

$ gcc -shared -o libsub.a sub.o

编译main.c：

$ gcc -c -o main.o main.c

链接程序，下面的指令将出错，提示找不到库文件：

$ gcc -o test main.o -lsub

/usr/bin/ld: cannot find -lsub

collect2: ld returned 1 exit status

可以使用-Ldir选项将当前目录加入搜索路径，如下则链接成功：

$ gcc -L. -o test main.o -lsub

（4）-Bprefix

这个选项指出在何处寻找可执行文件，库文件，以及编译器自己的数据文件。编译器驱动程序需要使用某些工具，比如：`cpp'，`cc1' (或C++的`cc1plus')，`as'和`ld'。它把prefix当作欲执行的工具的前缀，这个前缀可以用来指定目录，也可以用来修改工具名字。

对于要运行的工具，编译器驱动程序首先试着加上`-B'前缀(如果存在)，如果没有找到文件，或没有指定`-B'选项，编译器接着会试验两个标准前缀`/usr/lib/gcc/'和`/usr/local/lib/gcc-lib/'。如果仍然没能够找到所需文件，编译器就在`PATH'环境变量指定的路径中寻找没加任何前缀的文件名。如果有需要，运行时(run-time)支持文件`libgcc.a'也在`-B'前缀的搜索范围之内。如果这里没有找到，就在上面提到的两个标准前缀中寻找，仅此而已。如果上述方法没有找到这个文件，就不链接它了。多数情况的多数机器上，`libgcc.a'并非必不可少。

可以通过环境变量GCC\_EXEC\_PREFIX获得近似的效果；如果定义了这个变量，其值就和上面说的一样被用作前缀。如果同时指定了`-B'选项和GCC\_EXEC\_PREFIX变量，编译器首先使用`-B'选项，然后才尝试环境变量值。

### 2.8 ld/objdump/objcopy选项

我们在开发APP时，一般不需要直接调用这3个命令；在开发裸机、bootloader时，或是调试APP时会涉及，到时再讲。