

[首页](#) [资讯](#) [精华](#) [论坛](#) [问答](#) [博客](#) [专栏](#) [群组](#) [更多](#) ▼

[您还未登录！](#) [登录](#) [注册](#)

## lveyo的BCNDYL

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)

### linux下GCC编译C程序(一)

博客分类：

- [C/C++](#)

#### LinuxGCC#C++C

#### GNU编译器集

GNU编译器集（其前身为GNU C编译器）诞生于1987年。当时Richard Stallman（GNU项目的创办人）想要创建一个编译器，它可以满足他定义的“自由软件”概念，并可用来编译GNU项目发布的其他软件。GNU C编译器迅速在自由软件社区中流行开来，而且以其健壮性和可移植性而闻名。它已成为许多集成开发工具的基础，被世界各地的发行商应用在Linux和其他操作系统之上。

GCC已不再是主要针对GNU项目自身的软件的小型C语言编译器了。如今，它已支持了许多不同的语言，包括C、C++、Ada、Fortran、Objective C，甚至还有Java。事实上，现代Linux系统除了可以自豪地炫耀那些由GNU工具直接支持的语言以外，它还支持大量其他语言。日益流行的脚本语言Perl、Python和Ruby，以及正在不断发展的mono 可移植C#实现的确有助于冲淡人们对Linux编程的传统看法，但这完全是另外一个问题了。

Linux内核和许多其他自由软件以及开放源码应用程序都是用C语言编写并使用GCC编译的。

#### 1. 编译单个源文件

为了进行测试，你可以创建“Hello World”程序：

C代码  

1. `#include <stdio.h>`
2. `#include <stdlib.h>`
- 3.

```
4. int main(int argc, char **argv)
5. {
6.     printf("Hello world!\n");
7.     exit(0);
8. }
```

使用如下命令编译并测试这个代码：

引用

```
# gcc -o hello hello.c
```

```
# ./hello
```

```
Hello wordl!
```

在默认情况下产生的可执行程序名为a.out，但你通常可以通过gcc的“-o”选项来指定自己的可执行程序名称。

## 2. 编译多个源文件

源文件message.c包含一个简单的消息打印函数：

C代码  

```
1. #include <stdio.h>
2.
3. void goodbye_world(void)
4. {
5.     printf("Goodbye, world!\n");
6. }
```

使用gcc的“-c”标记来编译支持库代码：

引用

```
# gcc -c message.c
```

这一过程的输出结果是一个名为message.o的文件，它包含适合连接到一个较大程序的已编译目标代码。

创建一个简单的示例程序，它包含一个调用goodbye\_world的main函数

C代码  

```
1. #include <stdlib.h>
2.
3. void goodbye_world(void):
```

```
4.  
5. int main(int argc, char **argv)  
6. {  
7.     goodbye_world();  
8.     exit(0);  
9. }
```

使用GCC编译这个程序：

引用

```
# gcc -c main.c
```

现在有了两个目标文件：message.o和main.o。它们包含能够被Linux执行的目标代码。要从这个目标代码创建Linux可执行程序，需要再一次调用GCC来执行连接阶段的工作：

引用

```
# gcc -o goodbye message.o main.o
```

运行编译结果：

引用

```
# ./goodbye
```

Goodbye, world!

前面这些单独的步骤也可以简化为一个命令，这是因为GCC对如何将多个源文件编译为一个可执行程序有内置的规则。

引用

```
# gcc -o goodbye message.c main.c
```

```
# ./goodbye
```

Goodbye, world!

### 3. 使用外部函数库

GCC常常与包含标准例程的外部软件库结合使用，几乎每一个Linux应用程序都依赖于由GNU C函数库GLIBC。

应用外部函数库的例子：

C代码  

```
1. #include <stdio.h>  
2. #include <stdlib.h>  
3. #include <math.h>  
4.  
5. #define MAX_INPUT 25  
6.  
7. int main(int argc, char **argv)
```

```
8. {
9.     char input[MAX_INPUT];
10.    double angle;
11.
12.    printf("Give me an angle (in radians) ==>");
13.    if(!fgets(input, MAX_INPUT, stdin)){
14.        perror("an error occurred.\n");
15.    }
16.    angle = strtod(input, NULL);
17.
18.    printf("sin(%e) = %e\n", angle, sin(angle));
19.
20.    return 0;
21. }
```

编译命令：

引用

```
# gcc -o trig -lm trig.c
```

GCC的"-lm"选项，它告诉GCC查看系统提供的数学库（libm）。因为Linux和UNIX的系统函数库通常以"lib"为前缀，所以我们假设它存在。真正的函数库位置随系统的不同而不同，但它一般会位于目录/lib或/usr/lib中，在这些目录中还有数以百计的其他必需的系统函数库。

#### 4. 共享函数库与静态函数库

Linux系统上的函数库分为两种不同的类型：共享的和静态的

静态函数库：每次当应用程序和静态连接的函数库一起编译时，任何引用的库函数中的代码都会被直接包含进最终的二进制程序。

共享函数库：包含每个库函数的单一全局版本，它在所有应用程序之间共享。这一过程背后所涉及的机制相当复杂，但主要依靠的是现代计算机的虚拟内存能力，它允许包含库函数的物理内存安全地在多个独立用户程序之间共享。

使用共享函数库不仅减少了文件的容量和Linux应用程序在内存中覆盖的区域，而且它还提高了系统的安全性。一个被许多不同程序同时调用的共享函数库很可能会驻留在内存中，以在需要使用它时被立即使用，而不是位于磁盘的交换分区中。这有助于进一步减少一些大型Linux应用程序的装载时间。

将上面的message.c作为共享库函数使用的例子：

引用

```
# gcc -fPIC -c message.c
```

“PIC”命令行标记告诉GCC产生的代码不要包含对函数和变量具体内存位置的引用，这是因为现在

还无法知道使用该消息代码的应用程序会将它连接到哪一段内存地址空间。这样编译输出的文件message.o可以被用于建立共享函数库，我们只需使用gcc的“-shared”标记即可：

引用

```
# gcc -shared -o libmessage.so message.o
```

将上面的mian.c使用共享库函数libmessage.so编译：

引用

```
# gcc -o goodbye -lmessage -L. message.o
```

“-lmessage”标记来告诉GCC在连接阶段引用共享函数库libmessage.so。“-L.”标记告诉GCC函数库可能位于当前目录中，否则GNU的连接器会查找标准系统函数库目录，在本例的情况下，就找不到可用的函数库了。

此时运行编译好的goodbye会提示找不到共享函数库：

引用

```
#!/goodbye
```

```
./goodbye: error while loading shared libraries: libmessage.so: cannot open shared object file: No such file or directory
```

可以使用命令ldd来发现一个特定应用程序需要使用的函数库。ldd搜索标准系统函数库路径并显示一个特定程序使用的函数库版本。

引用

```
#ldd goodbye
linux-gate.so.1 => (0x00493000)
libmessage.so => not found
libc.so.6 => /lib/libc.so.6 (0x0097c000)
/lib/ld-linux.so.2 (0x0095a000)
```

库文件libmessage.so不能在任何一个标准搜索路径中找到，而且系统提供的配置文件/etc/ld.so.conf也没有包含一个额外的条目来指定包含该库文件的目录。

需要设置一个环境变量LD\_LIBRARY\_PATH来制定额外的共享函数库搜索路径，

引用

```
# export LD_LIBRARY_PATH=`pwd`
# ldd goodbye
linux-gate.so.1 => (0x002ce000)
libmessage.so => /tmp/cpro/libmessage.so (0x00b0f000)
libc.so.6 => /lib/libc.so.6 (0x0097c000)
/lib/ld-linux.so.2 (0x0095a000)
```

运行程序

引用



```
# ./goodbye
Goodbye, world!
```

3

顶

0

踩

分享到:  [Spring调用spymemcached客户端的例子](#) | [memcached运行情况监测](#)

- 2008-09-11 13:54
- 浏览 23794
- [评论\(0\)](#)
- 论坛回复 / [浏览](#) (0 / 9899)
- 分类:[编程语言](#)
- [相关推荐](#)

评论

发表评论

[您还没有登录,请您登录后再发表评论](#)

lveyo

- 浏览: 592628 次
- 性别: 
- 来自: 北京
-  我现在离线

最近访客

[更多访客>>](#)[pretty\\_kiddy](#)



[Mr.cool](#)



[walk影](#)



[轻风无言](#)

## 文章分类

- [全部博客 \(135\)](#)
- [JAVA \(12\)](#)
- [C/C++ \(3\)](#)
- [AJAX \(7\)](#)
- [ORACLE \(3\)](#)
- [MYSQL \(2\)](#)
- [TOMCAT \(1\)](#)
- [PHP \(6\)](#)
- [ANDROID \(4\)](#)
- [UBUNTU \(23\)](#)
- [MG4J \(2\)](#)
- [WEBSERVER \(4\)](#)
- [MEMCACHED \(6\)](#)
- [MAC \(1\)](#)
- [OTHER \(19\)](#)

## 社区版块

- [我的资讯 \(0\)](#)
- [我的论坛 \(10\)](#)
- [我的问答 \(1\)](#)

## 存档分类

- [2011-04 \(1\)](#)
- [2011-03 \(1\)](#)
- [2010-12 \(2\)](#)
- [更多存档...](#)

## 最新评论

- [kernel288](#): 谢谢.解决了.  
[WARN No appenders could be found for logger的解决方法](#)

- [chou91225](#):  
[WARN No appenders could be found for logger的解决方法](#)
  - [往事也加](#): 还是没有解决问题啊  
[WARN No appenders could be found for logger的解决方法](#)
  - [905766491](#): 很不错,我看着你写的。五分钟搞定。不知道能否加下qq,我的qq ...  
[memcached安装和启动](#)
  - [style2013](#): 哈哈,太牛比了。。  
[WARN No appenders could be found for logger的解决方法](#)
- 

声明: ITeye文章版权属于作者, 受法律保护。没有作者书面许可不得转载。若作者同意转载, 必须以超链接形式标明文章原始出处和作者。

© 2003-2016 ITeye.com. All rights reserved. [ 京ICP证110151号 京公网安备110105010620 ]