第 5 章　gcc 及 gdb 使用

本章学习目标

- 了解 C 语言编译的一般流程。
- 熟悉 gcc 的使用方法。
- 掌握 gdb 调试程序的一般步骤。
- 掌握 makefile 文件编写规则。

在 Windows 下进行 C 或 C++ 程序设计时，一般使用集成开发环境 Visual Studio，所有环节均可在该环境下完成，且比较方便。与之相对，在 Linux 系统中，特别是在终端窗口中开发程序，为了提高开发效率及使工具尽量少占用空间的目的，一般均使用单独的工具编辑程序、编译和调试，这就不可避免地要使用编辑软件、gcc 编译工具、gdb 调试工具以及为大型项目开发而使用的 make 工具。本章简单介绍 Linux 系统下 C 程序开发的一般流程，重点对 gcc 编译工具和 gdb 调试工具进行详细讲解说明 gcc 编译过程都完成了哪些工作，在程序发生逻辑性错误时，如何使用 gdb 工具进行断点设置、跟踪调试。最后，通过一个复杂的项目实例演示 Makefile 制定编译规则给程序开发带来的好处。

5.1　Linux 下 C 编程概述

C 语言是贝尔实验室的 Dennis Ritchie 于 1973 年为了 UNIX 的辅助开发设计而编写的一种程序设计语言，是在 B 语言的基础上开发出来的。由于它的硬件具有无关性和可移植性，逐渐成为世界上使用最广泛的计算机语言之一。1987 年，美国国家标准协会(ANSI)根据 C 语言问世以来各种版本对 C 语言的发展和扩充，制定了新的标准，称为 ANSIC。

总体来讲，C 语言具备如下特点：

(1) 把高级语言的基本结构和语句与低级语言的实用性结合起来，像汇编语言一样，对位、字节和地址进行操作，更接近硬件设备，执行效率很高。

(2) 具有强大的图形处理功能，支持多种显示器和驱动器，且计算与逻辑判断功能均非常强大，可实现决策目的。

(3) 具有多种数据类型，并引入指针概念，使程序效率更高。

(4) 是结构化的语言，采用代码及数据分隔，除必要的信息交流外，程序的各个部分彼此独立，层次清晰，便于使用、维护及调试。

(5) 可移植性强，适合诸如 DOS、Windows 和 Linux 等多种操作系统及体系结构，尤其适合在嵌入式领域的开发。

当前 Linux 系统下进行的 C 语言编程，一般遵循 ISO/IEC 9899 标准，简称 C99 标准，该标准于 1999 年出台，经过 2001 年和 2004 年两次技术修正，是目前最新的 C 语言标准。

5.1.1 程序执行一般过程

从编写一个C语言程序到最终运行需要多个步骤。首先要使用合适的编译器(如vi或Emacs工具)编写程序源代码,一般得到一个后缀名为.c的文件;然后使用编译器(如gcc工具)对源程序文件进行编译,这需要完成词法分析、语法分析、中间代码生成、代码优化以及最终目标代码的生成,得到后缀名为.o的文件;最后要经过链接的过程,将多个目标代码文件(包括用到的静态库文件)合成最终的可执行文件(动态库的链接是在程序执行的时候被链接的)。

在Linux编程过程中,对于完成简单任务的程序,一般单个文件就可实现功能,可以使用gcc工具的不同参数功能将上述执行过程中的编译和链接工作合并完成。但是要完成的功能比较复杂,则需要编写多个源文件,还可能用到多个静态和动态库文件,这时就比较适合编写makefile文件,来描述整个工程的编译和链接规则,最终使用Make工具解释规则并执行。

当然,上述编译的过程中会出现一些语法错误,可根据提示进行修改,比较容易解决。而有些情况下发现程序能顺利编译通过并执行,但程序的运行结果与预期的不一致,这种情况表示程序中出现了逻辑性错误,此时就可利用gdb工具进行调试,通过设置断点、单步执行等手段进行跟踪调试,发现问题并加以解决。

上面程序执行的一般过程如图5.1所示。

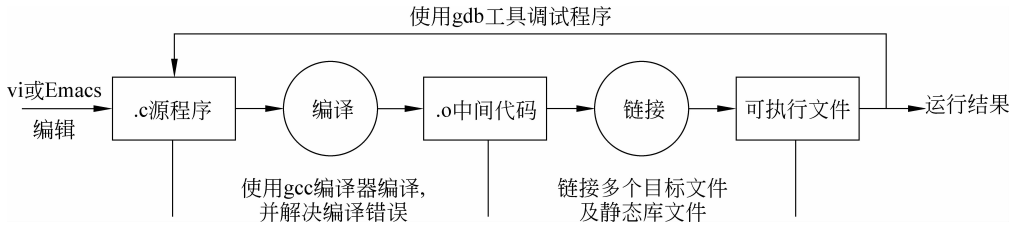

图5.1 程序一般执行过程示意图

5.1.2 编译过程描述

从编写源文件一直到程序成功运行,编译阶段需要完成词法分析、语法分析、中间代码生成、代码优化、目标代码生成5项工作,下面简要描述每个阶段的作用及工作原理。

(1) 词法分析。从左至右逐个字符地对源程序进行扫描,将源代码改造成单词符号串的组合,然后对每一个单词进行检查分析,发现错误就停止并显示错误提示。

(2) 语法分析。以单词符号为输入,分析单词符号串是否形成符合语法规则的语句,其间需要检查表达式、赋值、循环等结构是否完整和符合使用规则。一旦发现错误就提示,终止编译任务。

(3) 中间代码生成。该阶段将词法分析和语法分析无误的内容转换成中间代码,这样可使程序结构更加简单、规范,该过程与用户无关。

(4) 代码优化。对程序进行多种等价变换,使其成为更有效的目标代码。此过程中可以设置代码优化的参数,针对不同的环境和设置优化,提高程序运行效率。

(5) 目标代码生成。目标代码是二进制的机器语言,用户只能运行这个程序,而不能打开这个文件查看程序的代码。

编译过程的一般流程如图 5.2 所示。

图 5.2 编译过程框图

5.1.3 Emacs 编辑工具使用方法

编辑工具用来录入文字及修改文字,如 Windows 环境中的记事本和 Word。在 Linux 系统下,同样有多种编辑工具可供选用,如 vi(Vim)、Gedit 及 Emacs 等。下面介绍另一种常用并且强大的编辑工具——Emacs。

该工具是一款集编辑、编译、调试于一体的集成开发环境,可以在没有图形显示的终端下出色地工作,功能强大且效率高。在 Emacs 环境中,没有类似于 vi 的 3 种"模式",只是处于编辑状态,要实现的命令全靠功能键完成。

编写程序时,可以根据不同的程序类型给出相应的关键字颜色提示。例如,会根据编写的程序是 C 语言程序还是 Shell 脚本改变显示效果,更加方便编程人员。例如,在编写 C 语言程序时,用户拥有"自动缩进"、"注释"、"预处理扩展"、"自动状态"等强大功能,并且可以用 Tab 键自动地将当前行的代码产生适当的缩进,使代码结构清晰、美观,也可以指定缩进的规则。

1. 安装 Emacs 工具

在终端命令提示符下输入以下命令进行安装,该命令在联网情况下安装软件非常便利,会自动检测软件间的依赖关系,效率较高。安装过程及提示信息如下。

```
[root@bogon backup]#yum install emacs
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.neowiz.com
 * extras: ftp.neowiz.com
 * updates: ftp.neowiz.com
Setting up Install Process
Resolving Dependencies
-->Running transaction check
--->Package emacs.i386 0:21.4-24.el5 set to be updated
-->Processing Dependency: emacs-common =21.4-24.el5 for package: emacs
-->Processing Dependency: libXaw3d.so.7 for package: emacs
-->Running transaction check
--->Package Xaw3d.i386 0:1.5E-10.1 set to be updated
--->Package emacs-common.i386 0:21.4-24.el5 set to be updated
-->Finished Dependency Resolution
```

```
Dependencies Resolved

================================================================================
Package             Arch       Version           Repository     Size
================================================================================
Installing:
emacs               i386       21.4-24.el5       base           1.6 M
Installing for dependencies:
Xaw3d               i386       1.5E-10.1         base           152 k
emacs-common        i386       21.4-24.el5       base           10 M

Transaction Summary
================================================================================
Install     3 Package(s)
Upgrade     0 Package(s)

Total download size: 12 M
Is this ok [y/N]: y
Downloading Packages:
(1/3): Xaw3d-1.5E-10.1.i386.rpm                          | 152 kB     00:01
(2/3): emacs-21.4-24.el5.i386.rpm                        | 1.6 MB     00:05
(3/3): emacs-common-21.4-24.el5.i386.rpm                 | 10 MB      00:36
--------------------------------------------------------------------------------
Total                                           273 kB/s | 12 MB     00:45
warning: rpmts_HdrFromFdno: Header V3 DSA signature: NOKEY, key ID e8562897
base/gpgkey                                               | 1.5 kB     00:00
Importing GPG key 0xE8562897 "CentOS-5 Key (CentOS 5 Official Signing Key) <
centos-5-key@centos.org>" from /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
Is this ok [y/N]: y
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
Installing     : Xaw3d                                                    1/3
Installing     : emacs-common                                             2/3
Installing     : emacs                                                    3/3

Installed:
emacs.i386 0:21.4-24.el5

Dependency Installed:
Xaw3d.i386 0:1.5E-10.1           emacs-common.i386 0:21.4-24.el5

Complete!
[root@bogon backup]#
```

安装过程主要包括检查软件包依赖关系、要求用户下载相应软件包、询问是否导入GPG文件以及最终的安装过程显示。这时就可以在终端输入Emacs命令，打开这个编辑工具，欢迎界面如图5.3所示。

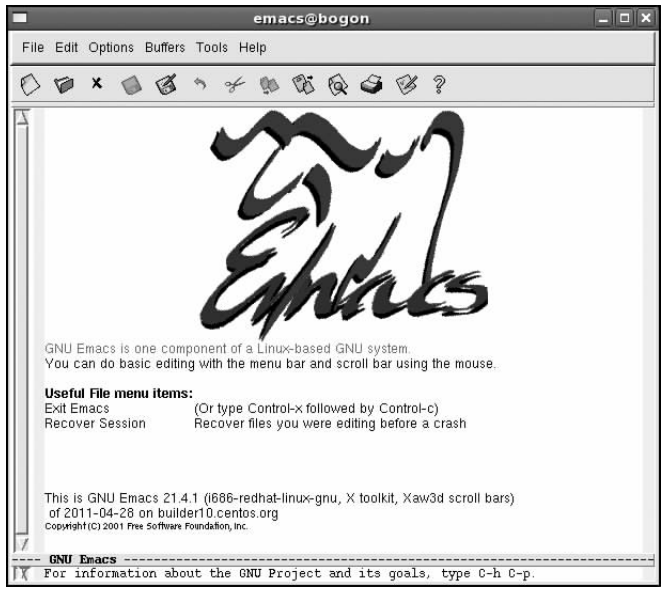

图5.3 emacs工具运行的界面

2. 使用方法

Emacs的工作窗口分为上、下两部分，上部为编辑窗口，下部为命令显示窗口，按功能键会在底部有相应的显示，有时也需要在下部窗口输入相应的命令，如查找字符串等。常用的编辑操作包括定位、复制、剪切、粘贴及保存文件等，它们都有对应的功能键，表5.1至表5.3列举了组合按键及其对应的作用。

表5.1 定位组合键及作用表

功 能 键	作 用	功 能 键	作 用
Ctrl+F	向前移动一个字符	Ctrl+B	向后移动一个字符
Ctrl+P	移动到上一行	Ctrl+N	移动到下一行
Ctrl+A	移动到行首	Ctrl+E	移动到行尾
Alt+F	向前移动一个单词	Alt+B	向后移动一个单词
Alt+<	移动到整篇文本开头	Alt+>	移动到整篇文本结尾
Ctrl+V	向上翻页	Alt+V	向下翻页

表5.2 剪切粘贴组合键及作用表

功 能 键	作 用
Delete	删除光标后的字符
Alt+Delete	剪切光标前面的单词

续表

功 能 键	作 用
Ctrl+K	剪切从光标到行尾内容
Backspace	删除光标前的字符
Alt+D	剪切光标前面的单词
Alt+K	剪切从光标到句尾的内容
Ctrl+Y	将缓冲区内容粘贴到当前光标所在位置
Ctrl+X U	先按 Ctrl+X,然后按 U 键,撤销操作

需要注意的是,如果希望复制一段内容到目标位置,可以首先把光标定位到复制内容的起始位置,然后按 Ctrl+Space 组合键,再将光标移动到结束位置,按 Alt+W 组合键,这样就可以将起点位置到终点位置之间的文本复制到缓冲区,最后定位到目标位置,按 Ctrl+Y 组合键实施粘贴。

表 5.3 查找等其他操作组合键及作用表

功 能 键	作 用
Ctrl+R	查找光标以前的内容,在对话框的"I-search backward:"后输入查找字符串
Ctrl+S	查找光标以后的内容,在对话框的"I-search:"后输入查找字符串
Ctrl+X Ctrl+S	先按 Ctrl+X,然后按 Ctrl+U,保存文档
Ctrl+X Ctrl+C	先按 Ctrl+X,然后按 Ctrl+U,退出文档

编辑时,Emacs 提供了"自动保存"(auto save)机制,该机制在系统发生异常时非常有用。自动保存是当用户编辑一个文档时,系统会生成一个与编辑文档同名且文件名前后都有一个"#"的文件。例如,编辑名为"hello.c"的文件,其自动保存的文件名就叫"#hello.c#",只有当用户正常地保存文件后,Emacs 才会删除该文件。

3. 编译和调试

Emacs 环境下可直接进行编译,此时编辑器把编译器的输出和程序代码连接起来。程序员可以像在 Windows 的其他开发工具一样,将出错位置和代码定位联系起来,以快捷地发现错位处并进行修改。Emacs 支持大量的工程项目,方便开发。它还为 gdb 调试器提供了功能齐全的接口。Emacs 中使用 gdb 时,不仅能够获得 gdb 用其他方式运行时所具有的全部标准特性,还可以通过接口增强而获得其他性能。

5.2 gcc 编译工具

在 Linux 系统下,C 语言程序一般需要通过 gcc 来编译成可执行程序。gcc(GNU C Compiler)编译器是一个功能强大、性能优越的编译器,是 GNU 项目中符合 ANSI C 标准的编译系统,能够编译用 C、C++ 和 Object C 等语言编写的程序(具体支持语言详见表 5.4)。gcc 同时也是一个交叉平台编译器,它能够在当前 CPU 平台上为多种不同体系结构的硬件平台开发软件,尤其适合在嵌入式领域的开发编译。

表 5.4　gcc 支持后缀名文件表

后　缀　名	对应的语言
.c	C 语言源代码文件
.a、.so	由目标文件构成的静态和动态库文件
.C、.cc 或.cxx	C++ 源代码文件
.h	程序所包含的头文件
.i	已经预处理过的 C 源代码文件
.ii	已经预处理过的 C++ 源代码文件
.m	Object-C 原始程序
.s	汇编语言源代码文件
.S	经过预编译的汇编语言源代码文件

5.2.1　gcc 使用方法简介

gcc 使用时有很多选项可供选择，它们以不同参数形式附加在 gcc 命令后，并且可以直接形成可执行文件，使编辑到执行的过程更加快捷。gcc 命令的基本用法如下。

```
gcc [参数...] [文件名...]
```

从命令格式可以看到，gcc 命令后可跟多个参数和文件名，附加的参数可作用在每一个文件上，常用参数及其作用如表 5.5 所示。更多参数及其作用可以输入“gcc --help”命令进行查看。

表 5.5　gcc 常用参数及其作用表

参　数	作　　用
-E	只运行 C 预编译器，配合-o 可指定得到预处理过的.i 文件
-S	配合-o，将预处理输出.i 文件汇编成扩展名为.s 的汇编语言源代码文件
-c	只编译并生成后缀名为.o 的目标文件，不连接成为可执行文件
-o	指定可执行文件的名称，如果不加该参数，可执行文件默认名为 a.out
-g	产生调试工具 gdb 必需的符号信息，要调试程序，必须加入该选项
-O	编译、链接时优化，产生效率更高的可执行文件，编译链接速度相应减慢
-O2	比-O 效果更好的优化选项，同时对应的编译链接速度会更慢
-l	将该参数后跟的目录加入到程序头文件列表中
-L	首先到该参数后跟的目录中寻找所需要的库文件
-w	不生成任何警告信息
-Wall	生成所有警告信息
-MM	自动生成源文件和目标文件的依赖关系

为了演示这些参数的作用，使用Vim编辑最为熟悉的helloworld.c文件，该文件内容如下所示。

```
#include< stdio.h>
int main()
{
printf("hello world!\n");
return 0;
}
```

命令终端窗口中输入gcc命令编译该文件，得到可执行文件，一般情况下如果不加入-o，则可执行文件名默认为a.out，代码如下。

```
[jacky@bogon ~]$mkdir c
[jacky@bogon ~]$cd c
[jacky@bogon c]$vim helloworld.c
[jacky@bogon c]$ls
helloworld.c
[jacky@bogon c]$gcc helloworld.c
[jacky@bogon c]$ls
a.out helloworld.c
```

要执行该文件，可以输入“./a.out”查看执行结果，如果该文件不具备执行权限，需要使用“chmod +x a.out”命令为文件添加执行属性。

```
[jacky@bogon c]$ll a.out
-rwxrwxr-x 1 jacky jacky 4950 May 1 05:28 a.out
[jacky@bogon c]$./a.out
hello world!
[jacky@bogon c]$
```

可见，执行a.out时需要在其前面加上“./”，原因是可执行文件所在目录没有包含在环境变量PATH值中。这时必须给出可执行文件的完整路径名，“.”表示当前目录。要解决这个问题，可将文件所在的目录添加到环境变量PATH中，方法如下。

```
[jacky@bogon c]$export PATH=$PATH:.
[jacky@bogon c]$a.out
hello world!
```

如果需要修改可执行文件的名字，可以使用-o参数直接指定，效果如下。

```
[jacky@bogon c]$gcc helloworld.c-o helloworld
[jacky@bogon c]$ls
a.out helloworld helloworld.c
```

5.2.2 gcc编译流程

使用gcc进行的编译过程是一个复杂过程，可分为预处理(Pre-Processing)、编译

(Compiling)、汇编(Assembling)和链接(Linking)4 个阶段,整个流程如图 5.4 所示。

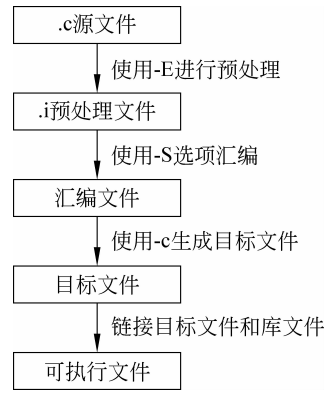

图 5.4 gcc 编译流程

下面结合 helloworld.c 程序实例展示这 4 个阶段都完成了哪些工作。

1. 预处理阶段

根据程序中以字符 # 开头的命令,修改原始的 C 程序,如 helloworld.c 中有语句 #include <stdio.h>,预处理器就读系统头文件 stdio.h 的内容,并把它直接插入到程序文本中去。完成后,会生成一个完整的 C 程序源文件。为了清楚这个阶段完成了什么工作,可以使用-E 参数进行预处理,并终止编译,得到扩展名为.i 的预处理文件,然后使用 cat 命令查看文件内容。

```
[jacky@bogon c]$gcc -E helloworld.c o helloworld.i
[jacky@bogon c]$cat helloworld.i
#1 "helloworld.c"
#1 "< built-in>"
#1 "< command line>"
#1 "helloworld.c"
#1 "/usr/include/stdio.h" 1 3 4
#28 "/usr/include/stdio.h" 3 4
#1 "/usr/include/features.h" 1 3 4
#329 "/usr/include/features.h" 3 4
#1 "/usr/include/sys/cdefs.h" 1 3 4
#313 "/usr/include/sys/cdefs.h" 3 4
#1 "/usr/include/bits/wordsize.h" 1 3 4
#314 "/usr/include/sys/cdefs.h" 2 3 4
#330 "/usr/include/features.h" 2 3 4
#352 "/usr/include/features.h" 3 4
#1 "/usr/include/gnu/stubs.h" 1 3 4

......

extern void funlockfile (FILE * __stream) __attribute__ ((__nothrow__));
#844 "/usr/include/stdio.h" 3 4

#2 "helloworld.c" 2
int main()
{
  printf("hello world!\n");
  return 0;
}
```

该文件共有 800 多行代码。可见,在程序编译时,需要调用很多的头文件和系统库函数。

2. 编译阶段

gcc对预处理后的文件进行编译，生成以.s为后缀的汇编语言文件。该阶段首先要检查代码的规范性、是否有语法错误等，以确定代码实际要做的工作，代码如下。检查无误后，会把代码翻译成汇编语言。这里使用-S参数来查看该过程的执行效果。

```
[jacky@bogon c]$gcc -S helloworld.i -o helloworld.s
[jacky@bogon c]$cat helloworld.s
    .file    "helloworld.c"
    .section    .rodata
.LC0:
    .string  "hello world!"
    .text
.globl main
    .type    main, @function
main:
    leal     4(%esp), %ecx
    andl     $-16, %esp
    pushl    -4(%ecx)
    pushl    %ebp
    movl     %esp, %ebp
    pushl    %ecx
    subl     $4, %esp
    movl     $.LC0, (%esp)
    call    puts
    movl     $0, %eax
    addl     $4, %esp
    popl     %ecx
    popl     %ebp
    leal     -4(%ecx), %esp
    ret
    .size    main, .-main
    .ident   "gcc: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-52)"
    .section    .note.GNU-stack,"",@progbits
```

3. 汇编阶段

汇编是处理汇编语言的阶段，主要调用汇编处理程序，将汇编语言汇编成二进制机器代码。该过程将扩展名为.s的汇编语言代码文件汇编为扩展名为.o的目标文件。所生成的目标文件作为下一步链接过程的输入文件，代码如下。

```
[jacky@bogon c]$gcc -c helloworld.s -o helloworld.o
```

查看.o文件内容时，全是乱码，这是因为已经得到了与机器语言对应的程序文件。

4. 链接阶段

链接就是将多个汇编生成的目标文件以及引用的库文件进行模块链接，生成一个完整