

第 3 章 Linux GCC/G++编译器与调试器

编译器是将易于编写、阅读和维护的高级计算机语言翻译为计算机能解读和运行的低级机器语言的程序。调试器是用于查找源代码中的错误，测试源代码和可执行文件的工具。GNU 项目提供了 GCC 编译器、G++编译器和 GDB 调试器，这些程序是在 Linux 系统上使用 C 和 C++语言进行开发的重要工具。本章将介绍这些工具的安装和使用方法。


3.1 GCC/G++编译器

GCC 是 GNU 项目中的一个子项目，最初为用于编译 C 语言的编译器。随着 GNU 项目的发展，GCC 已经成为了能编译 C、C++、Ada、Object C 和 Java 等语言的 GNU 编译器家族，同时还可执行跨硬件平台的交叉编译工作。G++则是专门用来编译 C 和 C++语言的编译器。C 和 C++语言正在不断发展，为了保持兼容程序语言的最新特性，开发者通常选择 GCC 来编译 C 语言编写的源代码，选择 G++来编译 C++源代码。

3.1.1 GCC/G++编译器的安装

安装或更新 GCC 和 G++可在 GNU 项目的官方网站（www.gnu.org）下载相应的安装包，也可以使用 YUM 软件包管理器安装。安装 GCC 和 G++的命令如下：

```
yum install make           //安装 make 程序
yum install gcc            //安装 GCC 编译器
yum install gcc-c++        //安装 G++编译器
```

 **注意：**如果安装过程中提示需要选择编译器版本，可根据当前硬件平台选择最新发布的版本。另外，如果提示需要安装其他相关软件包，请一并安装。

3.1.2 GCC/G++编译命令

GCC/G++编译器没有图形界面，只能在终端上以命令行方式运行。编译命令由命令名、选项和源文件名组成，格式如下：

```
gcc [-选项 1] [-选项 2]...[-选项 n] <源文件名>
g++ [-选项 1] [-选项 2]...[-选项 n] <源文件名>
```

命令名、选项和源文件名之间使用空格分隔，一行命令中可以有多个选项，也可以只有一个选项。文件名可以包含文件的绝对路径，也可以使用相对路径。如果文件名中不包

含路径，那么源文件被视为存在于工作目录中。如果命令中不包含输出的可执行文件名称，那么默认情况下将在工作目录中生成后缀为.out的可执行文件。

3.1.3 GCC/G++编译选项

GCC 拥有一百多个编译选项。对于 C 语言和 C++语言，G++与 GCC 的编译选项基本相同。常用的 GCC 和 G++编译选项见表 3.1。

表 3.1 常用GCC/G++编译选项

编 译 选 项	说 明
-c	只进行预处理、编译和汇编，生成.o 文件
-S	只进行预处理和编译，生成.s 文件
-E	只进行预处理，产生预处理后的结果到标准输出
-C	预处理时不删除注释信息，常与-E 同时使用
-o	指定目标名称，常与-c、-S 同时使用，默认是.out
-include file	插入一个文件，功能等同源代码中的#include
-Dmacro[=defval]	定义一个宏，功能等同源代码中的#define macro [defval]
-Umacro	取消宏的定义，功能等同源代码中的#undef macro
-Idir	优先在选项后的目录中查找包含的头文件
-lname	链接后缀为.so 的动态链接库来编译程序
-Ldir	指定编译搜索库的路径
-O[0-3]	编译器优化，数值越大优化级别越高，0 没有优化
-g	编译器编译时加入 debug 信息
-pg	编译器加入信息给 gprof
-share	使用动态库
-static	禁止使用动态库

3.1.4 GCC/G++编译器的执行过程

GCC 和 G++编译器执行过程可总结为 4 步：预处理、编译、汇编和连接。在预处理过程中，编译器会对源代码中的头文件和预处理语句进行分析，生成以.i 为后缀的预处理文件。编译过程是将输入的源代码编译为以.o 为后缀的目标文件。汇编过程是针对汇编语言的步骤，编译后生成以.o 为后缀的目标文件。最后执行连接过程，所有的目标文件被安排在可执行程序中恰当的位置。同时，该程序所调用到的库函数也从各自所在的档案库中连到合适的地方，如图 3.1 所示。

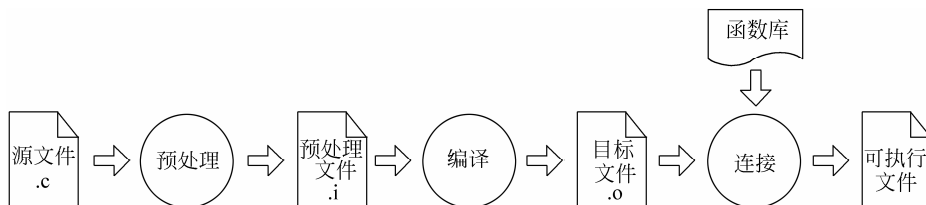


图 3.1 编译器执行过程

3.2 程序和进程

程序和进程是操作系统中的重要概念。程序是可运行的一组指令，以二进制代码的形式保存在存储设备中。操作系统运行程序后，该程序在内存中的映像就是进程，进程是活动的程序。

3.2.1 程序

程序是指一组指示计算机或其他具有信息处理能力设备每一步动作的指令。Linux 系统下的应用程序主要由两种特殊类型的文件代表，分别是可执行文件和脚本程序。可执行文件是能够被计算机直接执行的程序，相当于 Windows 系统中的 exe 文件。使用 C 和 C++ 语言设计的程序编译后即是可执行文件。Linux 系统不要求可执行文件使用特定的扩展名，文件是否能被执行由文件的系统属性来决定。

3.2.2 进程和 PID

进程是一个具有独立功能的程序关于某个数据集合的一次可以并发执行的运行活动，是处于活动状态的程序。进程作为构成系统的基本元件，不仅是系统内部独立运行的实体，而且是独立竞争资源的实体。

在 Linux 系统中，用户创建进程时会先在系统的进程表中为进程创建独一无二的编码，即 PID。PID 是一个正整数，其取值范围是 2~32768。进程创建时会顺序挑选下一个未使用的编号数字作为自己的 PID。如果它们已经经过一圈的循环，新的编码将重新从 2 开始。数字 1 一般是为特殊进程 init 保留的，它负责管理其他的进程。例如，执行下列命令会显示系统内的所有进程。

```
ps -e
```

显示的结果如下：

PID	TTY	TIME	CMD
1	?	00:00:03	init
2	?	00:00:00	kthreadd
3	?	00:00:00	migration/0
4	?	00:00:02	ksoftirqd/0
5	?	00:00:00	watchdog/0
6	?	00:00:02	events/0
7	?	00:00:00	khelper
80	?	00:00:00	kintegrityd/0
:			

3.3 ANSI C 标准

C 语言诞生后的很长一段时期内，并没有针对 C 语言制定严格的标准。不同编译器可能使用不同的语法规则或数据结构，此状况给程序的移植带来很多麻烦。于是，美国国家标准协会（ANSI）决定统一 C 语言的标准，并于 1989 年颁布 *ANSX3.159-1989* 标准文档，这一标准被称为 ANSI C 标准。ANSI C 标准在订立时吸取了很多 C++ 语言的内容，同时促使 C 语言支持多国字符集，其中包括各种中文字符集。ANSI C 标准的推出使 C 语言保持着活力，成为最受开发者欢迎的开发语言。

1999 年，ANSI 推出了 C 语言标准的修订版，该修订版简称 C99。GCC 编译器和 GDB 调试器均以 ANSI C 标准为原则，同时也支持 C99。

3.3.1 函数原型

C 语言设计的程序是由函数所组成的，在函数被详细定义前，可先在头文件定义函数原型，这样函数间可更容易地相互调用。头文件 `<unistd.h>` 包含了许多 Linux 系统服务的函数原型，例如 `read()`、`write()` 和 `getpid()` 函数，它们的原型如下：

```
ssize_t read(int, void *, size_t);           //read() 函数的原型
ssize_t write(int, const void *, size_t);     //write() 函数的原型
pid_t getpid(void);                           //getpid() 函数的原型
```

函数原型由函数的返回类型、函数名和参数 3 部分组成。例如第一行中 `ssize_t` 表示该函数返回值是 `ssize_t` 类型的值。括号中是参数列表，多个参数用逗号分隔，代码中最后一行 `getpid()` 函数的参数是 `void`，表示 `getpid()` 函数没有参数。当源代码编译时，因为编译器已经知道参数的类型，所以会将调用的参数进行强制转换。

3.3.2 类属指针

类属指针是一种能够同时支持所有数据类型的指针。函数原型中常用的“`void *`”类型即是类属指针。ANSI C 标准常用类属指针代替函数参数中的其他指针，使同一个函数能支持多种数据类型。相关内容将在本书的程序实例中多次见到。

3.3.3 原始系统数据类型

在函数原型中以“`_t`”结尾的类型被称为原始系统数据类型。原始系统数据类型定义在头文件 `sys/types.h` 中，以 `typedef` 操作符加以定义。原始系统数据类型是目标系统数据结构的接口，在不同的操作系统中，其字长会有变化。

3.4 编译 hello world

hello world 程序作为程序员学习的第一个程序已成为有趣的惯例。本节将讲述如何使用 Linux 系统中默认的编辑器编写该程序的源代码, 并使用 GCC 编译器将该程序编译为可执行文件。

3.4.1 使用 VI 编写源代码

VI 是 Linux 系统中最常用的文本编辑器, 几乎所有 Linux 发行版中都已包含 VI 程序。它工作在控制台或终端中, 通过 shell 调用, 全部操作均由命令完成, 对于初学者来说并不容易掌握。下列命令在用户主目录 “/home/用户名” 中创建一个名为 helloworld 的目录, 并在该目录中使用 VI 新建并打开 helloworld.c 文件。

```
cd ~                //进入 “/home/用户名” 目录, 使之成为工作目录
mkdir helloworld    //新建 helloworld 目录
cd helloworld       //进入 helloworld 目录, 工作目录是 “/home/用户名/helloworld”
vi helloworld.c      //使用 vi 新建并打开 helloworld.c 文件
```

“.c” 结尾的文件表示该文件是 C 语言源代码。执行完以上命令, 终端即进入 VI 程序。这时先按 Esc 键进入命令输入状态, 再输入命令 a 并按 Enter 键, VI 可从光标所在位置开始录入文本。输入 helloworld.c 文件的内容如下:

```
01 #include <stdio.h>           //这个头文件包含基本的输入输出函数
02 int main()                   //主函数, 程序将从这里开始执行
03 {
04     char *c;                  //声明一个字符串变量 c
05     c = "hello world!";       //为字符串变量赋值
06     printf("%s\n", c);        //输出该变量, 并输出换行符
07     return 0;                 //程序结束时向操作系统返回 0, 表示正常退出
08 }
```

录入结束后, 先按 Esc 退出输入状态, 再输入命令 Q 进入 Ex 模式, 在 Ex 模式下输入 wq 并按 Enter 键, 该文件被保存并退出 VI 程序。VI 的命令非常丰富, 如果输入有误或需修改文件, 可参照表 3.2 对文本进行修改。

表 3.2 常用 VI 命令及解释

命 令	解 释	命 令	解 释
Esc	进入或退出命令模式	i/I	插入
h/j/k/l 或方向键	移动光标位置	a/A	在光标后输入
/关键字	向下查找关键字	o/O	插入新行
x/X	向前, 向后删除一个字符	r/R	在光标后改写
D/d	删除整行	w	保存文件
Y/y	复制整行	q	退出 VI
p/P	在上一行, 下一行粘贴	wq	保存文件并退出
U	还原前一动作	set nu	显示行号

3.4.2 程序的编译与连接

程序经过编译器的编译与连接后，即可生成可执行文件。如果源代码有语法错误，则会在终端上显示错误信息。有些时候，编译器会出现警告提示，但程序依然被编译成功。这表明源代码没有严格按照标准编写，可能会在运行时出现意外的结果。继续前面的操作不改变工作目录，编译并连接 `helloworld.c` 程序，可在终端上输入下列命令：

```
gcc -o helloworld helloworld.c           //编译并连接程序
                                           //-o helloworld 表示使用 helloworld 作为目标文件名
```

3.4.3 使用终端运行程序

要运行 `helloworld` 程序，继续前面的操作不改变工作目录，只需要在终端上输入下列命令：

```
./helloworld                             //运行当前目录下的 helloworld 程序
```

程序的输出结果为：

```
hello world!
```

要在 Linux 系统上运行程序，必须给出该程序完整的路径。前面的 `helloworld.c` 文件建立在“/home/用户名/helloworld”目录中，编译和连接后所生成的可执行文件也在该目录中。运行程序则需输入“/home/用户名/helloworld/helloworld”。但当前的工作目录已经是“/home/用户名/helloworld”了，所以可用“./”替代工作目录的路径。

3.5 GDB 调试器

程序编写后难免会出现各种错误。当程序完成编译时，隐藏的错误可能会使程序无法正常运行，或者不能实现预期的功能。简单的程序或浅显的错误可依赖程序员的经验判断出故障点，但现在的软件规模越来越大，调试起来也就越来越困难。调试器是帮助程序员修改错误的得力工具，常用的断点、单步跟踪等功能可帮助程序员快速找到故障点。

3.5.1 GDB 调试器概述

Linux 程序员中最常用的调试工具是 GDB。GDB 调试器是 GNU 项目的子项目。该程序提供了所有常用调试功能，是 Linux 系统中最为简单快捷的调试工具。由于当前图形用户界面 (GUI) 普及，大量基于 GUI 的调试器被开发和运行在 Linux 上。它们大多是以 GDB 为核心配上 GUI，即用户通过 GUI 发出命令，这些命令依次被传送给 GDB。其中一个为 DDD，意为数据显示调试器。在一些集成开发环境如 Eclipse 中，也提供了调试功能，并且以 GDB 为核心。

3.5.2 GDB 调试器安装

通常在 Linux 桌面版的软件开发包集合中已包含 GDB 调试器。如果需要安装或更新 GDB 调试器，可使用 YUM 软件包管理器完成。操作方法如下：

```
yum install gdb //YUM 安装 GDB 调试器
```

3.5.3 GDB 常用调试命令

GDB 调试器调试的对象是可执行文件，使用 GCC 或 G++编译器编译源代码时，必须加上选项-g 才能使目标可执行文件包含可被调试的信息。以 3.4 节中的 helloworld 程序为例，编译连接程序，并使用 GDB 调试器打开目标可执行文件的命令如下：

```
gcc -g -o helloworld helloworld.c //编译并连接程序，使之包含可被调试信息
gdb helloworld //使用 GDB 调试器打开 helloworld 可执行文件
```

完成以上操作后，系统将显示 GDB 调试器的版本、使用的函数库信息，并显示 (gdb) 命令提示符。这时可输入命令对程序进行调试，常用的命令参见表 3.3。

表 3.3 常用GDB命令及解释

命 令	解 释
file <文件名>	在 GDB 中打开执行文件
break	设置断点，支持的形式有 break 行号、break 函数名称、break 行号/函数名称 if 条件
info	查看和可执行程序相关的各种信息
kill	终止正在调试的程序
print	显示变量或表达式的值
set args	设置调试程序的运行参数
delete	删除设置的某个断点或观测点
clear	删除设置在指定行号或函数上的断点
continue	从断点处继续执行程序
list	列出 GDB 中打开的可执行文件代码
watch	在程序中设置观测点
run	运行打开的可执行文件
next	单步执行程序
step	进入所调用的函数内部，查看执行情况
whatis	查看变量或函数类型，调用格式为“whatis 变量名/函数名”
ptype	显示数据结构定义情况
make	编译程序
quit	退出 GDB

3.5.4 在 GDB 下运行程序

打开可执行文件后，可根据需要在程序中加入断点或观察点，并运行程序。以 helloworld 程序为例，可在为变量赋值前加入断点，并运行程序。继续 3.5.3 小节的操作，方法如下：

```
(gdb) break 5           //在源代码第 5 行，即变量 c 赋值处加入断点
(gdb) run                //运行程序
```

3.5.5 检查数据

在程序中加入断点后，程序运行时会在断点处暂时停止，以便检查程序中的数据。通过检查数据可判断出许多种错误的所在。helloworld 程序在第 5 行加入了断点，这时第 5 行的代码并未执行。检查变量 c 的值可输入如下命令：

```
(gdb) print c           //显示变量 c 的值
```

命令执行后可见输出结果为：

```
$1 = 0x4e54eff4 "\355TNv~;N"
```

该结果表明变量 c 所指向的地址为 0x4e54eff4。继续执行程序，使用单步执行方式，再检查变量 c 的值，输入下列命令：

```
(gdb) next              //单步执行程序
(gdb) print c           //显示变量 c 的值
```

执行后可见输出结果为：

```
$2 = 0x80484c4 "hello world!"
```

表明变量 c 指向了地址 0x80484c4，该地址的内容转换为 ASCII 码的结果为 hello world!。如果还要继续运行程序，输入 continue 命令将运行到下一个断点或者程序结束。

3.6 小 结

本章介绍 Linux 系统下编译 C 和 C++ 语言的编译器 GCC 和 G++ 的基本概念及操作。它们与文本编辑器 VI 的结合，组成了最简单的程序开发环境。另外，本章讲解了程序和进程的概念。程序是编程工作的结果，进程是程序运行时在系统上的映射。GCC 和 G++ 编译器是遵循 ANSI C 标准所设计的，所以本章也简单介绍了 ANSI C 的概念。当深入学习 C 语言时，对 ANSI C 标准的了解会避免许多程序错误的产生。本章最后介绍了 GDB 调试器，读者还需要在学习中不断摸索该工具的操作方法。