

Linux

C

语言实务

- ※ 详尽的 C 语言介绍
- ※ Linux 特有的系统调用函数
- ※ 好用的 Linux 程序开发工具
- ※ 丰富的范例讲解
- ※ GCC 常用参数列表
- ※ 使用 gdb 调试工具



施威铭研究室 著



机械工业出版社
China Machine Press

Linux C 语言实务

施威铭研究室 著



机械工业出版社

本书从基础的 C 语言知识入手, 重点介绍 Linux 下的 C 语言编程, 以及在 Linux 中编写程序应注意的事项及概念。本书分为 2 篇, 第 1 篇是 C 语言入门, 带领读者从无到有, 进入 C 语言的世界。第 2 篇是 C 语言详解, 对 C 语言的构成方面进行详细的讨论; 读者可以详尽了解 C 语言的语法、Linux 特有的系统调用函数、Linux 的程序开发工具、GCC 常用参数列表以及 gdb 调试工具的使用, 并结合丰富的范例讲解进行实际操作。

本书通俗易懂, 理论与实践紧密结合, 可以作为初学者的 C 语言教材, 同时也是相关人员学习和使用 Linux 的参考用书。

版权声明

本书由台湾旗标出版股份有限公司授权机械工业出版社在中国大陆境内独家出版发行, 未经出版者许可, 不得以任何方式抄袭、复制或节录本书中的任何部分。

图书在版编目 (CIP) 数据

Linux C 语言实务/施威铭研究室著.

—北京: 机械工业出版社, 2002.8

ISBN 7-111-10773-X

I L II 施.. III.C 语言—程序设计 IV TP312

中国版本图书 CIP 数据核字 (2002) 第 061217 号

机械工业出版社 (北京市百万庄大街 22 号 邮政编码 100037)

责任编辑: 于希莹

北京忠信诚胶印厂印刷·新华书店北京发行所发行

2002 年 9 月第 1 版·第 1 次印刷

787mm×1092mm 1/16 · 21.75 印张·526 千字

0001-4000 册

定价: 29.00 元

凡购本图书, 如有缺页、倒页、脱页, 由本社发行部调换

本社购书热线电话: (010) 68993821、68326677-2527

封面无防伪标均为盗版

前 言

近年来虽然不断地涌现出许多新的程序语言，但 C 语言仍然占有举足轻重的地位。C 语言的关键字（keyword）大约 30 个左右，但却可以写出功能庞大的程序；而且其运算式也很简洁，由此写出来的程序十分有效率。C 语言也因此成为许多人初次学习程序语言的第一选择。

Linux 是目前主流的计算机操作系统，它保持着自由与开放的精神，使用者只需用很少的钱就能获得软件。当然，若只想在 Linux 中练习 C 语言，也可不需任何金钱便可获得开发工具。

同时 C 语言也是 Linux 的母语，其操作系统本身主要是由 C 语言编写而成的。Linux 的源程序代码大多是开放的，如果想要了解某个程序到底是如何写成的，大多可以顺利取得程序代码进行分析研究。也正是由于这种开放精神，使得 Linux 成为学习 C 语言的最佳平台。

本书针对 C 语言的初学者，从最基本的 C 语言入门、编译器操作，到整个 C 语言的详解都一一作了说明。虽然是以 Linux 操作系统为平台，但书中仍会针对在 Linux 开发程序中应有的概念进行介绍，因此即使对 Linux 没有深入的了解，也能轻松学会。只要循序渐进地学习本书，必定可以顺利地掌握 C 语言的精髓，进而发挥它强大的功能。

施威铭研究室

目 录

第 1 篇 C 语言入门

第 1 章 Linux C 语言的开发环境.....	3
1.1 编译语言的基本认识.....	4
1.1.1 编译器与解释器.....	4
1.1.2 解释性语言与编译语言的比较.....	5
1.2 GCC 简介.....	5
1.3 C 程序的开发过程.....	6
1.4 编辑一个 C 程序.....	7
1.5 C 程序编译与执行.....	9
1.5.1 编译与执行程序.....	9
1.5.2 程序源文件发生错误.....	10
1.6 在 Linux 开发程序的技巧.....	11
1.7 习题.....	15
第 2 章 C 程序初步.....	17
2.1 一个 C 程序的例子.....	18
2.2 C 程序的基本结构.....	18
2.3 函数的概念.....	24
2.4 字符、字符串、数组与指针.....	27
2.5 循环控制.....	34
2.6 if 语句及 == % & 运算符.....	36
2.6.1 if 流程控制语句.....	36
2.6.2 == 与 =.....	37
2.6.3 逻辑运算符.....	37
2.6.4 输入函数 scanf().....	38
2.6.5 运算符 &.....	38
2.7 编写 C 程序的注意事项.....	38
2.7.1 变量的初值赋值.....	38
2.7.2 指针的初始值设置.....	39
2.7.3 “;” 与 C 语言的自由格式.....	40
2.7.4 语句的集合.....	41

2.7.5 关键字	41
2.8 回顾与总结.....	42
2.9 习题	43
第3章 C语言的工作环境支持——函数库及整合服务.....	47
3.1 C语言是没有I/O的语言	48
3.2 I/O转向的概念	48
3.3 标准I/O的函数群	49
3.3.1 函数库	49
3.3.2 I/O函数群.....	49
3.3.3 printf(): 格式化输出函数	53
3.3.4 格式化输入函数.....	56
3.3.5 &与*运算符.....	57
3.4 #define与常量名.....	58
3.4.1 常量的概念.....	58
3.4.2 使用常量名.....	59
3.5 观察预处理的结果.....	61
3.6 #include与stdio.h	62
3.6.1 #include的用法.....	62
3.6.2 包含子程序文件.....	64
3.7 使用make命令	65
3.7.1 makefile的结构.....	66
3.7.2 makefile的简化及宏.....	69
3.7.3 常用的make参数	71
3.8 习题	71

第2篇 C语言详解

第4章 C语言的数据类型	75
4.1 数据类型的变革.....	76
4.1.1 数据类型.....	76
4.1.2 C语言的数据类型.....	76
4.2 C语言的基本数据类型	76
4.2.1 数据的长度.....	77
4.2.2 整型的修饰符.....	77
4.2.3 数据类型的用法.....	79
4.3 数据的表示法.....	79
4.3.1 字符类型	79

4.3.2	int: 整型	81
4.3.3	浮点数与双精度数	82
4.3.4	双精度浮点数类型	83
4.3.5	void 类型	84
4.4	中文字体处理	84
4.4.1	中文字体码的问题	84
4.4.2	如何显示及输入中文字	84
4.5	习题	85
第 5 章	变量、运算符与运算式	87
5.1	变量与标识符	88
5.2	运算符与表达式	89
5.2.1	运算符	89
5.2.2	表达式	89
5.3	四则运算符、=与%	90
5.3.1	=运算符	90
5.3.2	四则运算符与%	91
5.3.3	=与运算符的合并	93
5.3.4	++与--运算符	94
5.3.5	比较用的运算符	96
5.3.6	逻辑运算符	97
5.3.7	位逻辑运算符	98
5.3.8	条件运算符	100
5.3.9	sizeof 运算符	101
5.3.10	其他运算符	101
5.4	运算符的优先级与结合性	102
5.5	表达式中的类型转换	103
5.6	强制的类型转换	106
5.7	习题	107
第 6 章	程序的流程控制	109
6.1	C 语言的流程控制	110
6.1.1	语句的集合	110
6.1.2	条件表达式的真假	111
6.2	条件判断 if...else	111
6.2.1	if 语句	111
6.2.2	多重的 if 语句	112
6.2.3	if...else 语句	112

6.2.4	if...else 的变形一：嵌套式 if...else.....	114
6.2.5	if...else 的变形二：else...if 语句.....	116
6.2.6	if...else 的变形三.....	117
6.2.7	使用 if...else 的注意事项.....	118
6.3	多重选择的 switch...case.....	119
6.3.1	switch...case.....	119
6.3.2	break.....	121
6.3.3	default.....	123
6.4	循环控制：while、do...while、for 与 continue.....	123
6.4.1	预先判断式循环 while.....	123
6.4.2	后设判断式循环 do...while.....	125
6.4.3	for 循环.....	127
6.4.4	各种循环的使用时机.....	130
6.4.5	循环控制中的 break.....	131
6.4.6	continue 语句.....	132
6.5	强制性的流程控制 goto.....	133
6.6	习题.....	136
第 7 章	函数与宏.....	139
7.1	函数简介.....	140
7.2	函数的定义与类型声明.....	141
7.2.1	ANSI 函数定义与声明.....	141
7.2.2	函数的返回值.....	143
7.2.3	函数的类型.....	145
7.2.4	void 类型.....	146
7.3	递归函数.....	147
7.4	预处理器.....	153
7.4.1	宏指令.....	153
7.4.2	带参数的宏定义.....	153
7.4.3	利用宏来调试.....	155
7.4.4	#undef 的使用.....	156
7.4.5	条件性编译指令.....	156
7.5	标准函数库.....	158
7.5.1	math.h.....	159
7.5.2	stdlib.h.....	161
7.5.3	time.h.....	163
7.5.4	其他的标准函数库.....	165
7.6	习题.....	165

第 8 章 指针与数组	167
8.1 指针与&、*运算符	168
8.1.1 指针的声明	168
8.1.2 &运算符的使用方法	168
8.1.3 运算符的使用方法	168
8.1.4 指针的初始化	170
8.1.5 指针的转型	172
8.2 以指针来传递参数	173
8.2.1 以指针传递变量地址	174
8.2.2 以指针返回字符串地址	177
8.3 指针与数组的关系	177
8.3.1 以指针传递数组地址	181
8.3.2 概念的澄清	183
8.3.3 指针的运算	184
8.4 多维数组及其设置	185
8.4.1 多维数组	186
8.4.2 数组指针	189
8.4.3 把多维数组传入函数	189
8.5 指针数组、指针的指针	190
8.5.1 指针数组	190
8.5.2 指针的指针	192
8.6 命令参数的引入: argc、argv	195
8.7 习题	199
第 9 章 变量等级	203
9.1 C 程序的结构与变量等级	204
9.2 内部变量	204
9.2.1 内部变量的范围	205
9.2.2 内部变量的生命周期	206
9.2.3 内部变量的优缺点	206
9.2.4 区段的构造与范围	206
9.3 静态内部变量	208
9.4 外部变量	209
9.4.1 外部变量的范围	209
9.4.2 外部变量的生命周期	210
9.4.3 外部变量的优缺点	210
9.5 静态外部变量	214

9.6 函数是外部个体.....	216
9.7 register 变量.....	217
9.8 习题	217
第 10 章 用户自定义数据类型——结构体与共用体	219
10.1 结构体	220
10.1.1 可由用户自定义的数据类型	220
10.1.2 结构体的声明	220
10.1.3 结构体的数组	222
10.1.4 结构体的初始设置	223
10.2 存取结构体数据	224
10.3 结构体指针	225
10.3.1 ->运算符	226
10.3.2 结构体的结构体	228
10.4 结构体与函数	229
10.4.1 把结构体指针传入函数	229
10.4.2 把结构体直接传入函数	231
10.5 共用体的使用	232
10.6 位段的使用	235
10.7 枚举型	236
10.8 类型名称定义 typedef	238
10.9 习题	240
第 11 章 文件处理	245
11.1 文件 I/O 与操作系统的关系	246
11.1.1 标准 I/O 函数库	246
11.1.2 Linux 的 I/O 函数群	246
11.2 C 语言的文件概念	247
11.2.1 stream	247
11.2.2 FILE 类型	247
11.2.3 文件的打开和关闭	248
11.2.4 缓冲式 I/O	251
11.3 C 语言的文件 I/O 函数群	251
11.3.1 读写文件字符: fgetc()、fputc()	251
11.3.2 字符串 I/O: fgets()、fputs()	254
11.3.3 格式化 I/O: fscanf()、fprintf()	256
11.4 顺序与随机读写	258
11.4.1 文件位置指针	258

11.4.2	fseek()函数.....	258
11.4.3	ftell()函数	260
11.4.4	fopen()的“+”更新模式.....	261
11.5	文件的格式分类.....	261
11.5.1	文本文件.....	262
11.5.2	二进制文件.....	263
11.5.3	设备文件.....	263
11.6	使用二进制文件.....	263
11.6.1	fopen()的另外3种文件设置模式	263
11.6.2	读写二进制文件.....	265
11.6.3	与结构体配合	266
11.7	习题	269
第 12 章	系统调用函数	271
12.1	认识系统调用.....	272
12.1.1	判断某个函数是属于系统调用或标准函数库函数	272
12.1.2	系统调用使用上的限制.....	273
12.2	取得与设置系统信息.....	273
12.2.1	取得操作系统名称、版本、网址及计算机类型	273
12.2.2	取得与设置网址.....	274
12.2.3	取得系统运行的状态.....	277
12.2.4	取得文件系统的状态.....	278
12.3	文件与目录管理.....	280
12.3.1	取得文件的状态.....	280
12.3.2	更改文件的权限与拥有者.....	283
12.3.3	创建文件的链接或符号链接.....	285
12.3.4	删除文件.....	287
12.3.5	创建与删除目录.....	287
12.3.6	取得与切换工作目录.....	288
12.4	习题	290
附 录		291
附录 A	GCC 参数说明.....	292
附录 B	利用 gdb 调试	300
附录 C	集成开发环境	317
附录 D	VIM 文本编辑器	329

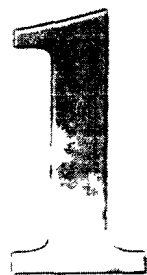
第 1 篇 C 语言入门

本篇 3 章的内容，将带领读者进入 C 语言的世界。第 1 章将介绍在 Linux 中开发程序的基本概念，并试着编辑、编译、产生第 1 个程序；第 2 章将对 C 语言做一个入门性的介绍，这才是真正接触 C 程序的一章；第 3 章介绍如何使用函数库及整体开发环境的支持。

第 1 章：本章主要说明开发 C 程序的基本概念。首先说明什么是编译语言及 GCC 编译器，并介绍开发程序所需经过的步骤，使读者对程序的产生流程有个基本的认识；接着实际编写一个 C 程序，并产生可执行文件；最后介绍几个在 Linux 中编写程序的技巧

第 2 章：本章将介绍 C 程序的基本格式、语句的定义、类型的声明等基本知识，另外对于一些重要的 C 语言概念如函数、指针，也都做了初步的介绍，目的是使读者早日熟悉并巩固这些概念。读完本章后读者应该对 C 程序有进一步的了解，并且已经能够编写简单的 C 程序了。

第 3 章：本章讲解 C 语言的 I/O 函数库，包含标准 I/O 的概念、I/O 的转向，以及字符、字符串函数的使用方法。使用 I/O 的转向，使较短的 C 程序也能发挥很强大的功能。另外，本章还介绍 C 语言的操作环境支持，包括 Preprocessor 指令及 make 的使用方法。



原书空白页

第 1 章

Linux C 语言的开发环境

1.1 编译语言的基本认识

计算机是以一种与人类完全不同的形式工作的，亦即其内部的语言与人类的语言完全不同。如果要驱使计算机工作，就必须以计算机认识的语言（即计算机语言）来下达命令，否则会造成“对牛弹琴”的窘境。

然而，要人类说计算机的语言是很不方便的。为了把人的语言编译成计算机的语言，计算机科学家设计了所谓的计算机程序语言（Computer Programming Language），其主要的方式就是把人类的语言在词汇、文法上均加以约束简化，使编译的工作变得简单，以达到实用、迅速的目的。这就是为什么计算机的程序语言总是有种似曾相识（如 if、for、while... 等）的感觉，但其语法（文法）又显得奇怪（多半限制重重）的原因了。

计算机语言的种类很多，有些语言较接近人类的日常用语，称为高级语言，这种语言较易学习使用。有些语言形式上较接近计算机内部语言，称为低级语言，这种语言不易学习，但效率较高。

1.1.1 编译器与解释器

常见的高级语言可分成两种，一种叫编译语言，一种叫解释性语言。无论是编译或解释性的语言，都需要一个编译程序把人类所下达的命令编译成计算机内部的语言。差异在于编译方式的不同。编译语言的编译程序叫编译器（Compiler），解释性语言的编译程序叫解释器（Interpreter）。

编译器是以整批作业的方式进行编译工作。首先必须完成源程序编写，并以编辑程序输入存为文件后，才能由编译器读入、编译，并且把编译的结果存为文件。这种经编译器编译成的文件，称为目标文件（Object File）。必须在整个编译工作完成，并产生可执行的（计算机看得懂的）目标文件后，才能把此目标文件交给计算机执行（目前大多数的系统还要经过链接处理才能执行）。如图 1-1 所示。常见的编译语言有 C/C++、Delphi、Kylix、Pascal 等。

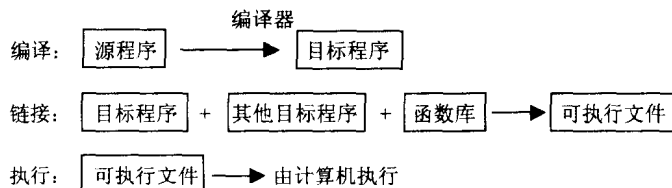


图 1-1 编译语言的处理过程

编译语言的另一项优点是，编译过的程序可以多次使用。通常，都将一些常用的程序

片段写成独立的函数（即子程序），并把这些模块集结成一个文件，在使用时就从此文件中抽取需要的函数模块来链接。这种由常用的函数模块所组成的文件称为函数库（Function Library）。函数库是编译语言的重要概念，将会在后文中详述。

解释器的工作方式与编译器不同。解释器不是采用事前整批作业的方式，而是在程序执行时，以实时的方式进行编译。也就是说，在执行时才把源程序一行一行地读进来进行编译，每读一行编译一行，并送给计算机执行，然后再继续读入下一行。如图 1-2 所示。

解释器与编译器的最大区别是，在每行命令读入、编译、送给计算机执行的过程中，解释器随时控制着整个进程。解释器这种兼具监督执行状况的功能，是编译器所没有的。常见的解释性语言有 Shell script、Python 等。

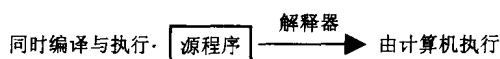


图 1-2 解释性语言的处理过程

1.1.2 解释性语言与编译语言的比较

由于解释性语言不做事先的编译，因此没有编译（及链接）的过程，操作过程简便、迅速。而且由于它具有监督执行状况的功能，所以可以在程序执行的过程中随时指出程序的错误。因此它通常容易学习，且十分友好（Friendly），所以如 Shell script 等这样的解释性语言，特别适合作为初学者的学习语言。

编译语言的优点是事先整批编译，而在执行过程中不必再次编译，所以执行速度很快，通常是解释性语言的几倍到十几倍，特别适于讲求实效性的应用。编译语言的主要缺点是操作过程过于复杂，从程序的编辑、编译、链接到执行，再加上来回调试，无论是方法或时间上都很费力费时。不过近年来出现的集成开发环境（Integrated Development Enviroment, IDE）已弥补了这方面的缺点，它将编辑和编译的功能整合在一起，使编译语言的开发过程，和解释性语言一样方便。

目前在 Linux 上最广泛使用的 C 语言编译器是 GCC，而这也是本书中所使用的编译器。

1.2 GCC 简介

GCC（GNU Compiler Collection）是一套由自由软件基金会（FSF, Free Software Foundation）开发的 C 语言编译器，问世于 1987 年 3 月 22 日，版本编号为具有测试意义的 0.9 版，紧接着在第二天（23 日）立即推出了正式的 1.0 版。经过版本的不断更新，目前最新版本为 3.x 版。其标志如图 1-3 所示。



图 1-3 GCC 的标志

在 Linux 中，GCC 被广泛采用并不是没有理由的，因为它具有以下几个特点：

- 软件获取方便：GCC 已包括在各 Linux 发行版的光盘中。只要在安装操作系统时，同时选择 GCC，那么在装好 Linux 系统后就可以马上使用了。
- 良好的执行效果：GCC 同时整合了编译器与链接器，因此可以将编译与链接同时执行，而不需要再分成 2 个步骤。同时 GCC 产生的可执行文件，在文件大小与执行效率上也颇为理想。
- 具有众多的参数：GCC 执行时可选择的参数很多，借助这些丰富的参数，使编译很有弹性。虽然 GCC 的参数众多，但这并不表示难以使用，一般的程序代码如果不加任何参数，也可顺利编译成功。
- 跨多种操作平台：凭借着 GNU 自由与开放的精神，除了 Linux 外，GCC 还被移植到大多数热门的操作平台上，例如 Solaris、BSD、OS/2，甚至连 Windows、DOS 也可以见到它的踪迹。
- 良好的兼容性：GCC 除了符合标准的 ANSI C 规范之外，还与早期 C 编译器的规范兼容，因此具有绝佳的兼容性。

注意

如果想进一步了解移植到 Windows/DOS 平台的 GCC 相关信息，请参考 cygwin (www.cygwin.com)、MinGW (www.mingw.org) 及 Dev-C++ (www.bloodshed.net/devcpp.html) 等网站的介绍。

1.3 C 程序的开发过程

对 GCC 编译器有了基本的概念之后，接着便可以编写程序了。一个程序从最初的构思到最后完成的过程称为“开发过程”。C 语言程序的开发过程如下：