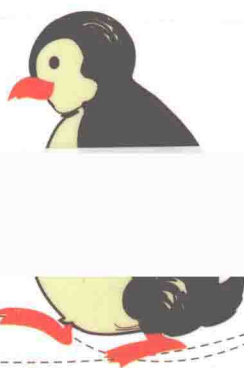


高质量嵌入式 Linux C编程

梁庚 陈明 马小陆 编著



内外兼修

涵盖“内功”和“外功”涉及的知识点
令您成为一个能够写出高质量程序的程序员



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

嵌入式技术与应用丛书

高质量嵌入式 Linux C 编程

梁 庚 陈 明 马小陆 编著

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

本书从嵌入式开发角度出发,以 Linux 操作系统为开发平台,将隐藏在系统开发背后的关于 C 语言、计算机组成原理、计算机操作系统等方面的机制和知识娓娓道来,不仅能让读者知其然,更要让读者知其所以然,揭开嵌入式 Linux C 系统开发背后鲜为人知的秘密,并让这些知识再反作用于编程实践,从而帮助读者写出高质量的嵌入式 Linux C 代码。具体说来,本书主要讨论了包括嵌入式 C 语言高级编程、嵌入式 Linux 系统编程、多任务解决机制、网络编程等多个方面的话内容。

本书既可作为大专院校相关专业师生的教学参考书,也可供计算机及其相关领域的工程技术人员查阅之用。对于普通计算机爱好者,本书也不失为帮助他们掌握高质量嵌入式 Linux C 系统开发的一本深入浅出的嵌入式开发读物。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

高质量嵌入式 Linux C 编程 / 梁庚, 陈明, 马本立编著. —北京: 电子工业出版社, 2015.1

(嵌入式技术与应用丛书)

ISBN 978-7-121-25383-6

I. ①高… II. ①梁… ②陈… ③马… III. ①Linux 操作系统—程序设计 ②C 语言—程序设计
IV. ①TP316.89 ②TP312

中国版本图书馆 CIP 数据核字 (2014) 第 313674 号



责任编辑: 田宏峰

印 刷: 北京京科印刷有限公司

装 订: 三河市华成印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1 092 1/16 印张: 23.5 字数: 600 千字

版 次: 2015 年 1 月第 1 版

印 次: 2015 年 1 月第 1 次印刷

印 数: 3000 册 定价: 68.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zltts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线: (010) 88258888。

前 言

在这几年的项目开发和担任苏嵌教育培训讲师的过程中，作者发现一个能够写出高质量程序的程序员需要“内外兼修”：“内功”就是精通数据结构和算法，并且拥有庞大的代码量，这是决定程序员写出高质量程序的根本因素；而“外功”就是程序员是否明白计算机组织机制和原理，写出的程序是否能符合计算机系统的“口味”，是否能够最大程度地调动和运用系统的资源，是否能够像一件艺术品一样供其他人欣赏。

一些程序员通常都认为具备“内功”就可以写出高效率的程序，常常忽略“外功”的作用，只能说他所写出的代码是“伪高效代码”，并不是一个高质量的程序。就作者个人的经历而言，自己手底下带的人和培训过的学员大多是半路出家，没有系统化地学习过计算机相关的理论知识，写出代码也只是简单地对需求进行模拟，并不能够很好地利用计算机系统资源。

需要指出的是，很多计算机专业科班出身的学生也未必能够领悟“外功”的含义。就目前国内计算机教育来说，大部分学校设置的计算机相关课程仅仅是将各项知识独立地对待，这样对于悟性不是非常高的学生来说，在没有被点化的情况下就没有办法有机地将这么多课程串接起来。一个不能形成完整系统性的知识结构是空洞和脆弱的结构。就拿现在畅销的《C 程序设计语言》来说，好多在校大学生在阅读这本书之后都说看不懂，觉得讲得很难，是因为这本书并不像国内很多 C 语言教材一样去单纯地介绍 C 语言的语法，这本书结合了计算机操作系统和计算机组成原理，以及 Linux 操作系统来阐述 C 语言的开发。

现在开发人员和学员间流行着这样一句话：“大学老师教的企业不用，企业用的大学老师不教”。其实我个人并不是很赞同这句话，毕竟大学让大学生们掌握了做开发所需要的一些基础理论知识，这为以后继续学习和工作都奠定了很好基础。现在的大学生急需去解决自己“内功”和“外功”修炼的问题，而本书可以帮助你，让你成为一个“内功”深厚，“外功”强大的程序员。

本书从嵌入式开发角度出发，以 Linux 操作系统为开发平台，将隐藏在系统开发背后的关于 C 语言、计算机组成原理、计算机操作系统等方面的机制和知识娓娓道来，不仅能让读者知其然，更要让读者知其所以然，揭开嵌入式 Linux C 系统开发背后鲜为人知的秘密，并让这些知识再反作用于编程实践，从而帮助读者写出高质量的嵌入式 Linux C 代码。具体说来，本书主要讨论了包括嵌入式 C 语言高级编程、嵌入式 Linux 系统编程、多任务解决机制、网络编程等多个方面的内容。本书大量案例都在苏嵌课堂教学实践过多年，既可作为大专院校相关专业师生的教学参考书，也可供计算机及其相关领域的工程技术人员查阅之用，对于普通计算机爱好者，本书也不失为帮助他们掌握高质量嵌入式 Linux C 系统开发的一本深入浅出的嵌入式开发读物。

作 者

2014 年 12 月

目 录

第 1 章 嵌入式 Linux C 语言开发工具	1
1.1 嵌入式 Linux C 语言开发概述	1
1.2 嵌入式 Linux C 开发环境	1
1.3 嵌入式文本编辑器	2
1.3.1 基本模式	2
1.3.2 基本操作	3
1.3.3 实训操作	5
1.4 嵌入式编译器	6
1.4.1 初识 GCC 编译器	6
1.4.2 gcc 命令常用选项及工作流程	6
1.4.3 库的使用	10
1.5 嵌入式调试器	12
1.6 工程管理器	15
1.6.1 Makefile	16
1.6.2 Makefile 特性介绍	18
1.7 Eclipse 程序开发	26
1.7.1 Eclipse 环境安装	26
1.7.2 Eclipse C 程序开发	28
第 2 章 数据类型	35
2.1 变量与常量	35
2.2 变量	35
2.2.1 什么是变量	35
2.2.2 变量名和变量值	36
2.2.3 局部变量和全局变量	38
2.3 常量	40
2.4 基本内置类型	41
2.4.1 数据类型大小	42
2.4.2 陷阱之有符号与无符号	42
2.5 声明与定义	43
2.5.1 定义	43
2.5.2 声明	43
2.6 乱世枭雄: static 与 extern	44
2.6.1 政权旗帜 static	44
2.6.2 外来的和尚会念经 extern	45
2.7 铁布衫: const	47

2.8	隐形刺客: auto	48
2.9	闪电飞刀: register	49
2.10	专一王子: volatile	50
2.11	typedef 详解	51
2.11.1	typedef 与结构的问题	51
2.11.2	typedef 与#define 的问题	53
2.11.3	typedef 与#define 的另一例	53
2.11.4	typedef 与复杂的变量声明	54
2.12	枚举	55
2.12.1	枚举类型的使用方法	55
2.12.2	枚举与#define 宏的区别	56
2.13	联合体	56
2.13.1	联合体的定义	56
2.13.2	从两道经典试题谈联合体 (union) 的使用	57
第 3 章	运算符、表达式	59
3.1	运算符简介	59
3.1.1	运算符优先级	59
3.1.2	一些容易出错的优先级问题	61
3.1.3	逻辑运算符	61
3.2	条件运算符和条件表达式	62
3.3	++、--操作符	63
3.4	位运算	64
3.4.1	按位与运算及应用	64
3.4.2	按位或运算及应用	64
3.4.3	按位异或运算及应用	65
3.4.4	左移和右移	65
3.5	C 语言性能优化: 使用位操作	65
第 4 章	语句	67
4.1	空语句	67
4.2	基础语句	68
4.2.1	表达式语句	68
4.2.2	函数调用语句	68
4.3	语句 if	68
4.3.1	布尔变量与零值比较	69
4.3.2	整型变量与零值比较	69
4.3.3	浮点变量与零值比较	69
4.3.4	指针变量与零值比较	70
4.3.5	对 if 语句的补充说明	70
4.4	跳转语句: goto	70
4.5	循环语句	71

4.5.1	do-while 语句	72
4.5.2	for 语句	72
4.5.3	循环语句的效率	74
4.6	break 和 continue	75
4.6.1	break 语句	75
4.6.2	continue 语句	75
4.7	switch 语句	77
第 5 章	数组与指针	79
5.1	数组认知	79
5.2	使用数组之常见问题	80
5.2.1	数组的下标总是从 0 开始吗	80
5.2.2	可以使用数组后面第一个元素的地址吗	81
5.2.3	为什么要小心对待位于数组后面的那些元素的地址呢	82
5.2.4	数组作为参数传递给函数时, 可以通过 sizeof 得到数组的大小吗	82
5.2.5	指针或带下标的数组名都可以访问元素, 哪一种更好呢	83
5.2.6	可以把另外一个地址赋给一个数组名吗	85
5.2.7	array_name 和 &array_name 有什么不同	86
5.2.8	为什么用 const 说明的常量不能用来定义一个数组的初始大小	87
5.2.9	字符串和数组有什么不同	87
5.3	指针	89
5.3.1	指针是变量	90
5.3.2	指针的类型和指针所指向的类型	90
5.3.3	指针的值	91
5.3.4	指针本身所占据的内存区	91
5.4	指针的运算	92
5.4.1	指针的算术运算	92
5.4.2	指针的关系运算	92
5.4.3	间接引用	93
5.4.4	最多可以使用几层指针	93
5.5	常量指针和指针常量	95
5.5.1	常量指针与指针常量的实例	95
5.5.2	常量指针的应用	96
5.6	空指针及其使用	97
5.6.1	NULL 总是被定义为 0 吗	97
5.6.2	NULL 总是等于 0 吗	97
5.6.3	空指针的使用	98
5.7	指针 void: 万能指针	99
5.8	指针数组与数组指针	100
5.9	字符串函数详解	101
5.10	函数指针与指针函数	105

5.11	复杂指针声明: “int * (* (*fp1) (int)) [10];”	106
5.11.1	基础	106
5.11.2	const 修饰符	107
5.11.3	typedef 的妙用	108
5.11.4	函数指针	109
5.11.5	右左法则	109
第 6 章	内存管理	111
6.1	你的数据放在哪里	111
6.1.1	未初始化的全局变量 (.bss 段)	111
6.1.2	初始化过的全局变量 (.data 段)	112
6.1.3	常量数据 (.rodata 段)	112
6.1.4	代码 (.text 段)	113
6.1.5	栈 (stack)	113
6.1.6	堆(heap)	113
6.2	内存分配方式	114
6.3	野指针	115
6.4	常见的内存错误及对策	115
6.5	段错误以及调试方法	116
6.5.1	方法一: 利用 gdb 逐步查找段错误	117
6.5.2	方法二: 分析 core 文件	118
6.5.3	方法三: 段错误时启动调试	119
6.5.4	方法四: 利用 backtrace 和 objdump 进行分析	120
6.6	指针与数组的对比	121
第 7 章	预处理、结构体	125
7.1	宏定义: #define	125
7.1.1	无参宏定义	125
7.1.2	带参宏定义	127
7.2	文件包含	128
7.3	条件编译	129
7.4	宏定义使用技巧	131
7.5	关于#和##	132
7.6	结构体	133
7.6.1	内存字节对齐	135
7.6.2	内存对齐正式原则	138
7.7	#define 和 typedef 的区别	139
7.8	结构体和联合体的区别	139
7.9	浅谈 C 语言中的位段	139
7.9.1	位段的使用	140
7.9.2	位段结构在内存中的存储方式	140

第 8 章	函数	141
8.1	函数声明与定义	141
8.1.1	定义	141
8.1.2	声明与定义不同	142
8.2	形式参数和实际参数	143
8.3	参数传递	143
8.3.1	简单变量或数组元素作为函数参数	143
8.3.2	指针变量或数组名作为函数参数	144
8.3.3	数组名作函数参数	145
8.3.4	结构体数组作函数参数	146
8.4	如何编写有多个返回值的 C 语言函数	146
8.4.1	利用全局变量	146
8.4.2	传递数组指针	148
8.4.3	传递结构体指针	148
8.5	回调函数	149
8.6	变参函数详解: printf 的实现	151
8.7	可变参数问题	152
第 9 章	编码规范	155
9.1	排版	155
9.2	注释	158
9.3	标示符名称	163
第 10 章	shell 编程	165
10.1	什么是 shell	165
10.2	几种流行的 shell	165
10.3	shell 程序设计 (基础部分)	166
10.3.1	shell 基本语法	166
10.3.2	shell 程序的变量和参数	167
10.4	shell 程序设计的流程控制	169
10.4.1	test 测试命令	169
10.4.2	if 条件语句	170
10.4.3	for 循环	171
10.4.4	while 和 until 循环	171
10.4.5	case 条件选择	172
10.4.6	无条件控制语句 break 和 continue	173
10.4.7	函数定义	173
10.5	命令分组	174
10.6	信号	174
10.7	运行 shell 程序的方法	175
10.8	bash 程序的调试	175
10.9	bash 的内部命令	176

第 11 章	文件操作	179
11.1	Linux 文件结构	179
11.1.1	Linux 文件系统	179
11.1.2	Linux 目录结构	180
11.1.3	Linux 文件分类	182
11.1.4	常见文件类型	183
11.1.5	Linux 文件属性	183
11.2	系统调用	184
11.3	Linux 文件描述符	184
11.4	不带缓存的 I/O 操作	185
11.4.1	creat 函数	185
11.4.2	open 函数	186
11.4.3	read 函数	188
11.4.4	write 函数	189
11.4.5	lseek 函数	189
11.4.6	close 函数	189
11.4.7	经典范例：文件复制	190
11.5	带缓存的 I/O 操作	191
11.5.1	三种类型的缓冲	191
11.5.2	fopen 函数	193
11.5.3	fclose 函数	194
11.5.4	fdopen 函数	194
11.5.5	fread 函数	195
11.5.6	fwrite 函数	195
11.5.7	fseek 函数	196
11.5.8	fgetc 函数、getc 函数和 getchar 函数	197
11.5.9	fputc 函数、putc 函数和 putchar 函数	198
11.6	fgets 函数与 gets 函数比较分析	199
11.7	输出与输入	201
11.7.1	printf 函数、fprintf 函数和 sprintf 函数	201
11.7.2	scanf 函数、fscanf 函数和 sscanf 函数	203
第 12 章	进程控制编程	207
12.1	为何需要多进程（或者多线程），为何需要并发	207
12.1.1	进程	207
12.1.2	进程分类	208
12.1.3	进程的属性	208
12.1.4	父进程和子进程	208
12.2	Linux 进程管理	209
12.2.1	ps 监视进程工具	209
12.2.2	pgrep 查询进程工具	211

12.2.3	终止进程的工具 kill、killall、pkill、xkill	211
12.2.4	top 监视系统任务的工具	213
12.2.5	进程的优先级: nice 和 renice	214
12.3	Linux 进程的三态	215
12.3.1	三种基本状态	215
12.3.2	三种状态间的转换	215
12.4	Linux 进程结构	216
12.5	Linux 进程控制块 PCB	216
12.6	Linux 进程调度	218
12.6.1	调度的目标	218
12.6.2	调度算法	218
12.6.3	优先级反转	220
12.7	进程创建	221
12.7.1	获取进程	221
12.7.2	启动进程: fork()	222
12.7.3	启动进程: vfork()	224
12.7.4	启动进程: exec 族	225
12.7.5	启动进程: system	228
12.8	进程等待	229
12.8.1	僵尸进程的产生	229
12.8.2	如何避免僵尸进程	231
12.8.3	wait 函数和 waitpid 函数	231
12.9	进程退出	235
12.9.1	退出方式的不同点	236
12.9.2	exit() 和 _exit() 函数	236
12.9.3	exit() 和 _exit() 的区别	237
第 13 章	进程间通信方式	239
13.1	进程间通信方式概述	239
13.1.1	进程间通信的目的	239
13.1.2	Linux 进程间通信方式简介	240
13.2	管道通信	241
13.2.1	建立无名管道	241
13.2.2	读写无名管道	242
13.2.3	无名管道应用实例	246
13.2.4	创建有名管道	248
13.2.5	读写有名管道	250
13.3	管道通信方式的应用场景	253
13.4	信号	254
13.4.1	信号及信号来源	254
13.4.2	信号种类	254

13.4.3	信号处理方式	256
13.4.4	信号发送	256
13.4.5	自定义处理信号方式	258
13.4.6	信号集操作	262
13.4.7	使用信号注意事项	264
13.5	消息队列	265
13.5.1	消息队列基础理论	266
13.5.2	使用消息队列	266
13.5.3	消息队列 API	267
13.5.4	消息队列的限制	269
13.5.5	消息队列的应用实例	270
13.6	信号灯	273
13.6.1	信号灯概述	273
13.6.2	内核实现原理	274
13.6.3	使用信号灯	274
13.6.4	信号灯 API	275
13.6.5	信号灯的	277
13.6.6	竞争问题	277
13.6.7	信号灯应用实例	277
13.7	共享内存方式一	281
13.7.1	内核实现原理	281
13.7.2	mmap()及其相关系统调用	282
13.7.3	mmap()范例	283
13.7.4	对 mmap()返回地址的访问	287
13.8	共享内存方式二	289
13.8.1	系统 V 共享内存原理	289
13.8.2	系统 V 共享内存 API	290
13.8.3	系统 V 共享内存范例	291
第 14 章	多线程编程	295
14.1	线程概述	295
14.1.1	为什么有了进程的概念后,还要再引入线程呢	295
14.1.2	多线程的优点	296
14.1.3	多线程的缺点	296
14.2	多线程的实现	297
14.2.1	线程的创建	297
14.2.2	终止线程	299
14.2.3	等待线程终止	300
14.3	线程属性	300
14.3.1	线程属性初始化	301
14.3.2	线程分离	301

14.3.3	线程的继承性	302
14.3.4	线程的调度策略	303
14.3.5	线程的调度参数	304
14.3.6	实例分析	305
14.4	线程同步机制	306
14.4.1	互斥锁 Mutex	306
14.4.2	互斥锁使用实例	308
14.4.3	条件变量 Conditions	310
14.4.4	条件变量使用实例	311
第 15 章	网络编程	313
15.1	TCP/IP 协议概述	313
15.1.1	TCP/IP 起源	313
15.1.2	TCP/IP 的特性与应用	315
15.1.3	互联网地址	315
15.1.4	域名系统	316
15.1.5	封装	317
15.1.6	TCP/IP 工作模型	318
15.1.7	TCP/IP 协议层	318
15.1.8	TCP/IP 应用	320
15.1.9	网桥、路由器和网关	321
15.2	TCP 和 UDP	322
15.2.1	TCP 协议	322
15.2.2	三次握手协议	322
15.2.3	TCP 数据报头	323
15.2.4	UDP 协议	324
15.2.5	协议的选择	324
15.2.6	IP 和端口号	324
15.3	套接字	325
15.3.1	Socket 概念	325
15.3.2	Socket 类型	325
15.3.3	Socket 信息数据结构	325
15.3.4	数据存储优先顺序的转换	326
15.3.5	地址格式转化	327
15.3.6	名字地址转化	328
15.4	网络编程	330
15.4.1	建立 Socket	331
15.4.2	绑定地址	332
15.4.3	监听	333
15.4.4	接受请求	334
15.4.5	连接服务器	335

15.4.6	发送数据	335
15.4.7	接收数据	336
15.5	采用 TCP 协议的 C/S 架构实现	338
15.5.1	模块封装	338
15.5.2	服务器的实现	340
15.5.3	客户端的实现	341
15.6	并发服务器模型	342
15.6.1	多进程解决方案	342
15.6.2	多线程解决方案	342
15.6.3	调用 fcntl 将 sockfd 设置为非阻塞模式	348
15.7	多路转接模型	348
15.7.1	服务器的实现	349
15.7.2	客户端的实现	354
15.8	采用 UDP 协议 C/S 架构的实现	355
15.8.1	服务器的实现	355
15.8.2	客户端的实现	356
15.8.3	UDP 协议传输文件的实现	357
参考文献		360

嵌入式 Linux C 语言开发工具

开发工具和操作系统之间是相互促进、相互发展的，操作系统离不开软件开发工具的支持，软件开发工具也离不开操作系统这个平台。Linux 操作系统下 Linux 开发工具的开源方式，可以让大家拥有更多的资源，得到更多的信息，对软件工具的发展起到了更大的促进作用。在这其中开发工具起到了至关重要的作用，开发工具作为生产软件的工具，有如神兵利器一般为 Linux 的发展保驾护航。



1.1 嵌入式 Linux C 语言开发概述

在自然界中不论多复杂的问题都是由两部分组成的，一是问题处理的对象；二是处理问题的具体方法。例如，用布匹做衣服的问题，衣服就是问题的对象，具体的剪裁工艺就是做衣服的方法。由于现实生活中的问题是无穷无尽的，所以人们发明了很多计算工具来帮助处理问题，如古老的沙漏计时器、算盘等。但直到电子计算机的出现，人们才真正从繁重的计算任务中解脱出来。

电子计算机之所以具有强大的计算能力，除了运算速度快之外，根本原因在于计算机具有自动运行程序的能力。因此，计算机能否正确高效地处理问题取决于程序能否客观正确地描述问题。而程序要把问题客观正确地描述清楚，最基本的要求是使用具有一套正确合理的语法规则的编程语言。这也就说明，学习编程语言的主要内容之一就是学习其语法规则和运行机制。

在众多的编程语言中，C 语言是一门历史悠久但生命力很强的高级语言。据最新的调查数据显示，目前 C 语言的使用率依旧保持在 30% 以上。C 语言之所以能够久盛不衰，主要原因有以下几点。

- C 语言具有出色的可移植性，能在多种不同体系结构的软/硬件平台上运行。
- C 语言具有简洁紧凑、使用灵活的语法规则，并能直接访问硬件。
- C 语言具有很高的运行效率。

鉴于以上原因，很多操作系统的内核、系统软件等都是使用 C 语言编写的。在嵌入式 Linux 开发领域，C 语言同样是最广泛的语言之一。



1.2 嵌入式 Linux C 开发环境

编辑工具：在 Linux 下编程，你不再拥有集成化环境，你可以使用类似于 EDIT 的工具——经典的 vi 来编辑源程序。当然，还有更高档一些的，如 joe、emacs 等。总之，编辑程序与编译工作是分开的。

编译工具：在 Linux 下支持大量的语言，有 C、C++、Java、Pascal、Fortran、COBOL 等，本书以 C/C++ 语言为主。在使用这些编译工具时，是使用命令行方式的，也就是说，先用编辑工具输入源程序，然后执行一长串的命令（参数比较复杂）进行编译。例如，“`gcc-o hello hello.c`”就是将 `hello.c` 编译为 `hello`，然后还需要为其赋予可执行的权限，这样才能完成整个工作。

调试工具：GDB 是 GNU 开源组织发布的一个强大的 UNIX 下调试程序工具。或许大家比较喜欢那种图形界面方式的，像 VC、BCB 等 IDE 的调试，但如果是在 UNIX 平台下编写软件，就会发现 GDB 这个调试工具有比 VC、BCB 的图形化调试器更强大的功能。所谓“寸有所长，尺有所短”，就是这个道理。

软件工具：一个大型软件总是由多个源程序组成的，为了能够将大量的编译命令作一次完成，Linux 中提供了 `make` 各 `autoconf` 的工具，分别用于大型软件的编译，以及编译前根据机器当前状态作相应配置。

开发工具包：在 Linux 下提供了优秀的 GNU C 库函数、Motif 函数库、GTK 函数库、QT 函数库等工具包，为你的编程提供大量的支持。

项目管理工具：在 Linux 下还有 CVS 这样优秀的用于版本控制、管理的软件配置管理工具。

Linux 作为一个自由软件，同时来提供了大量的自由软件，这些自由软件不仅可执行文件自由，而且源程序也自由。你可以通过研习这些优秀的源码来提高自己的编程技艺。



1.3 嵌入式文本编辑器

Linux 上最常用的文本编辑器是 `vi`（或 `vim`）。文本编辑器是所有计算机系统中最常使用的一种工具。用户在使用计算机的时候，往往需要创建自己的文件，无论是一般的文字文件、资料文件，还是编写源程序，这些工作都离不开编辑器。

`vi` 是 Visual Interface 的简称，它在 Linux 中的地位就像 `Edit` 程序在 DOS 上一样。它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制，这是其他编辑程序所没有的。

`vi` 不是一个排版程序，它不像 Word 或 WPS 那样可以对字体、格式、段落等其他属性进行编排，它只是一个文本编辑程序。`vi` 没有菜单，只有命令，且命令繁多。`vi` 有 3 种基本工作模式：命令行模式、插入模式和底行模式。

`vi` 命令可以说是 UNIX/Linux 世界里最常用的编辑文档的命令了，很多人不喜欢 `vi` 就因为它有太多的命令集，但是我们只需要掌握基本的命令然后灵活地加以运用，相信您一定会喜欢它的。

1.3.1 基本模式

基本上 `vi` 可以分为三种状态，分别是命令行模式、插入模式和底行模式，各模式的功能区分如下。

(1) 命令行模式。控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入插入模式或者底行模式。

(2) 插入模式。只有在插入模式下，才可以输入文字，按「ESC」键可返回命令行模式。

(3) 底行模式。将文件保存或退出 vi，也可以设置编辑环境，如寻找字符串、列出行号等。不过一般我们在使用时把 vi 简化成两个模式，就是将底行模式也算入命令行模式。

1.3.2 基本操作

1. vi 的进入与退出

(1) 在系统提示符号输入 vi 及文件名称后，就进入 vi 全屏幕编辑画面。

```
$ vi myfile
```

不过有一点要特别注意，就是进入 vi 之后，是处于「命令行模式」，要切换到「插入模式」才能够输入文字。初次使用 vi 的人都会想先用上下左右键移动光标，结果计算机一直哔哔叫，把自己气个半死，所以进入 vi 后，先不要乱动，转换到「插入模式」再说吧！

(2) 切换至插入模式编辑文件。在「命令行模式」下按一下字母「i」就可以进入「插入模式」，这时候你就可以开始输入文字了。

(3) 插入模式的切换。您目前处于「插入模式」，你就只能一直输入文字，如果你发现输错了字！想用光标键来回移动，将该字删除，就要先按一下「ESC」键转到「命令行模式」再删除文字。

(4) 退出 vi 及保存文件。在「命令行模式」下，按一下「:」冒号键进入「底行模式」。例如：

```
: w filename      (输入「w filename」将文章以指定的文件名 filename 保存)
: wq              (输入「wq」，存盘并退出 vi)
: q!              (输入 q!，不存盘强制退出 vi)
```

2. vi 的复制、删除

(1) 删除。

「x」：每按一次，删除光标所在位置的“后面”一个字符。

「#x」：例如，「6x」表示删除光标所在位置的“后面”6 个字符。

「X」：大写的 X，每按一次，删除光标所在位置的“前面”一个字符。

「#X」：例如，「20X」表示删除光标所在位置的“前面”20 个字符。

「dd」：删除光标所在行。

「#dd」：从光标所在行开始删除#行

(2) 复制。

「yw」：将光标所在之处到字尾的字符复制到缓冲区中。

「#yw」：复制#个字到缓冲区。