

广受好评的Linux精品图书全面升级，ChinaUnix社区鼎力推荐
资深程序员15年经验总结，深入探讨Linux应用层和内核层的网络编程
详细讲解HTTP服务器、协议栈和防火墙三个典型案例的实际开发过程



Linux

宋敬彬 等编著

网络编程 (第2版)

本书源代码及PPT下载网址：www.tup.com.cn或www.wanjuanchina.net

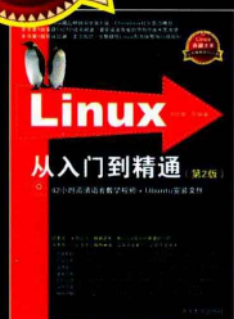
- ◎ 内容全面：涵盖Linux网络编程从基础到高级开发的方方面面知识
- ◎ 内容深入：重点讲解了技术性较强的Linux用户空间网络编程及内核网络编程
- ◎ 注重原理：对每个知识点都从原始概念和基本原理进行了详细和透彻的分析
- ◎ 插图丰富：对比较复杂和难度较高的内容绘制了220余幅原理图进行讲解
- ◎ 代码经典：书中的示例代码大多是从实际项目总结而来，有很强的实用性
- ◎ 实践性强：结合450余个示例、70余个应用实例及3个项目案例进行讲解
- ◎ 案例典型：详细介绍了HTTP网络服务器、协议栈和防火墙的实现过程



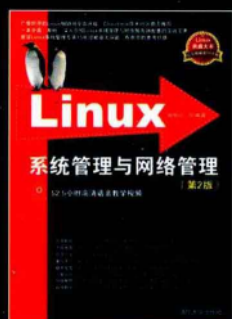
清华大学出版社



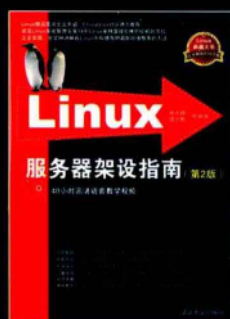
精品畅销丛书，一线人员打造，有口皆碑，Linux爱好者的不二选择！



ISBN 978-7-302-31272-7



ISBN 978-7-302-32018-0



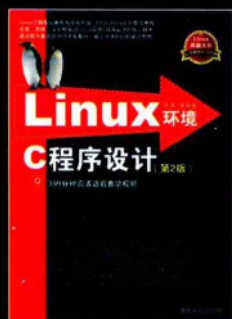
ISBN 978-7-302-31957-3



ISBN 978-7-302-33776-8



ISBN 978-7-302-33528-3



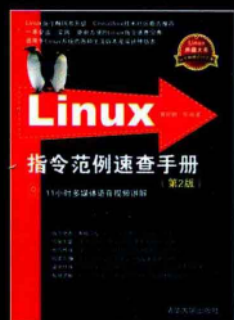
ISBN 978-7-302-34792-7



ISBN 978-7-302-34052-2



ISBN 978-7-302-34426-1



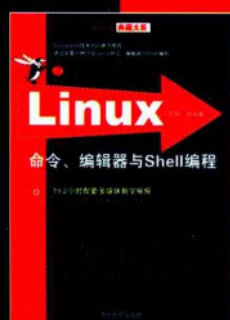
ISBN 978-7-302-34425-4



ISBN 978-7-302-33807-9



ISBN 978-7-302-30735-8



ISBN 978-7-302-27615-9



上架建议：计算机/Linux

ISBN 978-7-302-33528-3

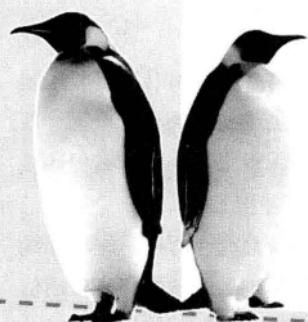


定价：89.00元

清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com



Linux

宋敬彬 等编著

网络编程 (第2版)

清华大学出版社

北 京

内 容 简 介

本书是获得大量读者好评的“Linux 典藏大系”中的《Linux 网络编程》的第2版。本书第1版出版后获得了读者的高度评价。本书循序渐进,从应用层到Linux内核,从基本知识点到综合案例,全面、系统地向读者介绍了如何在Linux下进行网络程序设计。本书涉及面广,从基本的编程工具介绍和编程环境搭建,到高级技术和核心原理,再到项目实战,几乎涉及Linux网络编程的所有重要知识。

本书共分4篇。第1篇介绍Linux操作系统概述、Linux编程环境、文件系统简介、程序、进程和线程;第2篇介绍TCP/IP协议族简介、应用层网络服务程序简介、TCP网络编程基础、服务器和客户端信息的获取、数据的IO和复用、基于UDP协议的接收和发送、高级套接字、套接字选项、原始套接字、服务器模型选择,以及IPv6的简介;第3篇介绍Linux内核中网络部分结构,以及分布和netfilter框架内报文处理;第4篇介绍三个网络编程的实例:Web服务器的例子SHTTPD、网络协议栈的例子SIP、防火墙的例子SIPFW。

本书适合所有想全面学习Linux网络编程的人员阅读,也适合已经从事Linux网络开发的工程技术人员使用。对于广大的Linux平台下的网络程序设计人员,本书更是一本不可多得的参考手册。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

Linux 网络编程/宋敬彬等编著.--2版.--北京:清华大学出版社,2014
(Linux 典藏大系)
ISBN 978-7-302-33528-3

I. ①L… II. ①宋… III. ①Linux 操作系统-程序设计 IV. ①TP316.89

中国版本图书馆CIP数据核字(2013)第199067号

责任编辑:夏兆彦

封面设计:欧振旭

责任校对:胡伟民

责任印制:沈 露

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>
地 址: 北京清华大学学研大厦A座 邮 编: 100084
社 总 机: 010-62770175 邮 购: 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 北京市密云县京文制本装订厂

经 销: 全国新华书店

开 本: 185mm×260mm 印 张: 44 字 数: 1100千字
版 次: 2010年1月第1版 2014年2月第2版 印 次: 2014年2月第1次印刷
印 数: 6501~10500
定 价: 89.00元

产品编号: 050118-01

前 言

Linux 操作系统已经成为目前最流行的开源操作系统，在服务器、嵌入式系统有着广泛的应用，并且逐步走入个人电脑的桌面操作系统。Linux 的网络程序设计在服务器领域、嵌入式领域有着广泛的应用。例如 Web 服务器、P2P 应用、嵌入式网络机顶盒、IPTV 机顶盒、手持设备等，上述产品大部分采用了开源的 Linux 系统。因此，熟悉并且能够编写网络程序代码，构建自己的网络架构程序是十分重要的。

本书是获得了大量读者好评的“Linux 典藏大系”中的《Linux 网络编程》的第 2 版。本书全面、系统地介绍了 Linux 网络编程技术，其中通过实例重点介绍了 Linux 的应用层网络设计、网络协议栈的实现原理和 Linux 内核防火墙的技术。学完本书之后，读者可以有编写比较复杂项目的本领。

关于“Linux 典藏大系”

“Linux 典藏大系”是清华大学出版社自 2010 年 1 月以来陆续推出的一个图书系列，截止 2013 年 1 月，已经出版了 10 余个品种。该系列图书涵盖了 Linux 技术的方方面面，可以满足各个层次和各个领域的读者学习 Linux 技术的需求。该系列图书自出版以来获得了广大读者的好评，已经成为 Linux 图书市场上最耀眼的明星品牌之一，其销量在同类图书中也名列前茅，其中一些图书还获得了“51CTO 读书频道”颁发的“最受读者喜爱的原创 IT 技术图书奖”。该系列图书出版过程中也得到了国内 Linux 领域最知名的技术社区 ChinaUnix（简称 CU）的大力支持和帮助，读者在 CU 社区中就图书的内容与活跃在 CU 社区中的 Linux 技术爱好者进行广泛交流，将会取得了良好的学习效果。

关于本书第 2 版

本书第 1 版出版后深受读者好评，并被 ChinaUNIX 技术社区所推荐。但是随着 Linux 技术的发展，本书第 1 版的内容与 Linux 各个新版本有一定出入，这给读者的学习造成了一些不便。应广大读者的要求，我们结合 Linux 技术的最新发展推出第 2 版图书。相比第 1 版，第 2 版图书在内容上的变化主要体现在以下几个方面：

- (1) 操作系统环境从原有的 Debian 改为更为通用的 Ubuntu。
- (2) Linux 内核介绍增加了 3.* 系列。
- (3) 对 IT 业界的动态进行了更新。
- (4) 对一些专有名词的大小写进行了更正，如 VIM、Emacs。
- (5) 由于 Vim 区分大小写，尤其在快捷键上面。为了避免读者误操作，所以对原有的快捷键大小写进行了重新确认，并更正部分错误的大小写。
- (6) 更正了第 1 版中的部分描述错误，如 Objective-C。

- (7) 对 GCC 软件包进行了更新。
- (8) 为了便于读者阅读和使用代码，对于完整的代码增加了行号。
- (9) 更正了部分调试选项的大小写错误。
- (10) 对部分 Shell 命令进行了更新，如 fdisk。
- (11) 对 Linux 涉及的硬件信息进行了更新，如对 Ext4 的支持。
- (12) 对需要重点注意的关键代码做了加粗。
- (13) 对部分代码缺少的库文件进行了补充。
- (14) 修改了部分函数库的包含关系。
- (15) 修改了部分变量的数据类型。
- (16) 修改了部分代码行号的说明错误。

本书的特点

1. 循序渐进，由浅入深

为了方便读者学习，本书首先介绍 Linux 的开发环境，然后介绍基本的网路程序设计方法，再进行 Linux 内核的网络设计方法。最后，通过 3 个综合案例，综合运用上述知识，让读者更深刻地了解网络程序设计的知识。在每一部分的介绍中都是按照由浅入深的方式进行介绍，先介绍基础知识，再结合高级知识进行介绍。

2. 技术全面，内容充实

本书基本涵盖了 Linux 网络程序设计的所有知识面，特别对于高级网络编程、原始套接字等高级应用层网络程序设计给出了全面的介绍和丰富的例子程序。除了用户界面的网络程序设计外，本书还对内核空间的网络程序设计进行了详细的介绍，针对 netfilter 框架，做了很细致的讲解，并给出了一个全面使用 netfilter 框架的案例，以方便读者深入了解。

3. 对比讲解，理解深刻

由于 Linux 程序设计的知识用于空间和内核空间的代码和模块是相互作用的，在多个主要函数介绍过程中，本书对用户空间和内核空间进行交互式的对比介绍，使读者在了解如何使用的情况下，更深入地了解为什么这样用，所谓“知其然并知其所以然”。

4. 案例精讲，深入剖析

根据本人多年的项目经验，只有实际接触案例和代码才能够对知识点更深入地了解。本书在介绍了 Linux 网络程序设计知识点的基础上，通过具有典型意义的 3 个案例，对各个知识点包括应用层的 HTTP 协议的 Web 服务器、协议栈原理的协议栈案例和内核网络的防火墙案例进行了深入剖析。

本书内容及体系结构

第 1 篇 Linux 网络开发基础（第 1~4 章）

本篇主要内容包括：Linux 操作系统概述、Linux 编程环境、文件系统简介、程序、进程和线程。通过本篇的学习，读者可以掌握 Linux 编程的基础知识，以及编程环境。

第 2 篇 Linux 用户层网络编程（第 5~15 章）

本篇主要包括：TCP/IP 协议族简介、应用层网络服务程序简介、TCP 网络编程基础、服务器和客户端信息的获取、数据的 IO 和复用、基于 UDP 协议的接收和发送、高级套接字、套接字选项、原始套接字、服务器模型选择、IPv6 简介。通过本篇的学习，读者可以掌握 Linux 网络编程的大部分知识。

第 3 篇 Linux 内核网络编程（第 16 章和第 17 章）

本篇主要包括：Linux 内核中网络部分结构，以及分布和 netfilter 框架内报文处理。通过本篇的学习，读者可以初步了解 Linux 内核网络编程的知识。

第 4 篇 综合案例（第 18~20 章）

本篇主要包括：一个简单 Web 服务器的例子 SHTTPD、一个简单网络协议栈的例子 SIP、一个简单防火墙的例子 SIPFW。通过本篇的学习，读者可以全面了解一个完整可用的 Linux 网络程序是如何编写的。

本书学习建议

- ❑ 建议没有基础的读者，从前至后顺次阅读，尽量不要跳跃。
- ❑ 书中的实例和示例建议读者都要亲自上机动手实践，学习效果会更好。
- ❑ 第 4 篇的内容偏重于实战，这部分内容在初期可以不需要全面掌握，只要理解思想即可，等读者有了较多开发经验后可进一步研读。

本书读者对象

- ❑ 想全面学习 Linux 网络编程的人员；
- ❑ Linux 网络编程从业人员；
- ❑ Linux 网络编程爱好者；
- ❑ 大中专院校的学生；
- ❑ 社会培训班的学员；
- ❑ 需要一本案头必备手册的开发人员。

本书作者

本书由宋敬彬主笔编写。其他参与编写的人员有陈超、陈锴、陈佩霞、陈锐、黎华、李鹏钦、李森、李奕辉、李玉莉、刘仲义、卢香清、鲁木应、马向东、麦廷琮、米永刚、欧阳昉、綦彦臣、冉卫华、宋永强、滕科平、王秀丽、王玉芹、魏莹、魏宗寿、温本利。

虽然我们对书中所述的内容都尽量予以核实，并多次进行文字校对，但可能还存在疏漏和不足之处，恳请读者批评指正。

编著者

目 录

第 1 篇 Linux 网络开发基础

第 1 章 Linux 操作系统概述	2
1.1 Linux 发展历史	2
1.1.1 Linux 的诞生和发展	2
1.1.2 Linux 名称的由来	3
1.2 Linux 的发展要素	3
1.2.1 UNIX 操作系统	3
1.2.2 Minix 操作系统	4
1.2.3 POSIX 标准	4
1.3 Linux 与 UNIX 的异同	5
1.4 操作系统类型选择和内核版本的选择	5
1.4.1 常见的不同公司发行的 Linux 异同	5
1.4.2 内核版本的选择	6
1.5 Linux 的系统架构	7
1.5.1 Linux 内核的主要模块	7
1.5.2 Linux 的文件结构	8
1.6 GNU 通用公共许可证	9
1.6.1 GPL 许可证的历史	9
1.6.2 GPL 的自由理念	10
1.6.3 GPL 的基本条款	11
1.6.4 关于 GPL 许可证的争议	12
1.7 Linux 软件开发的可借鉴之处	12
1.8 小结	13
第 2 章 Linux 编程环境	14
2.1 Linux 环境下的编辑器	14
2.1.1 Vim 使用简介	14
2.1.2 使用 Vim 建立文件	15
2.1.3 使用 Vim 编辑文本	16
2.1.4 Vim 的格式设置	18
2.1.5 Vim 配置文件.vimrc	18
2.1.6 使用其他编辑器	19

2.2	Linux 下的 GCC 编译器工具集.....	19
2.2.1	GCC 简介.....	19
2.2.2	编译程序的基本知识.....	21
2.2.3	单个文件编译成执行文件.....	21
2.2.4	编译生成目标文件.....	22
2.2.5	多文件编译.....	22
2.2.6	预处理.....	24
2.2.7	编译成汇编语言.....	24
2.2.8	生成和使用静态链接库.....	25
2.2.9	生成动态链接库.....	26
2.2.10	动态加载库.....	29
2.2.11	GCC 常用选项.....	31
2.2.12	编译环境的搭建.....	33
2.3	Makefile 文件简介.....	33
2.3.1	一个多文件的工程例子.....	33
2.3.2	多文件工程的编译.....	35
2.3.3	Makefile 的规则.....	37
2.3.4	Makefile 中使用变量.....	39
2.3.5	搜索路径.....	42
2.3.6	自动推导规则.....	43
2.3.7	递归 make.....	44
2.3.8	Makefile 中的函数.....	46
2.4	用 GDB 调试程序.....	47
2.4.1	编译可调试程序.....	48
2.4.2	使用 GDB 调试程序.....	49
2.4.3	GDB 常用命令.....	52
2.4.4	其他的 GDB.....	59
2.5	小结.....	60
第 3 章	文件系统简介.....	61
3.1	Linux 下的文件系统.....	61
3.1.1	Linux 下文件的内涵.....	61
3.1.2	文件系统的创建.....	62
3.1.3	挂接文件系统.....	65
3.1.4	索引节点 inode.....	65
3.1.5	普通文件.....	66
3.1.6	设备文件.....	66
3.1.7	虚拟文件系统 VFS.....	68
3.2	文件的通用操作方法.....	72
3.2.1	文件描述符.....	72

6.4	NFS 协议和服务	175
6.4.1	安装 NFS 服务器和客户端	175
6.4.2	服务器端的设定	176
6.4.3	客户端的操作	176
6.4.4	showmount 命令	177
6.5	自定义网络服务	177
6.5.1	xinetd/inetd	177
6.5.2	xinetd 服务配置	178
6.5.3	自定义网络服务	179
6.6	小结	180
第 7 章	TCP 网络编程基础	181
7.1	套接字编程基础知识	181
7.1.1	套接字地址结构	181
7.1.2	用户层和内核层交互过程	183
7.2	TCP 网络编程流程	184
7.2.1	TCP 网络编程架构	184
7.2.2	创建网络插口函数 socket()	186
7.2.3	绑定一个地址端口对函数 bind()	189
7.2.4	监听本地端口 listen	192
7.2.5	接受一个网络请求函数 accept()	194
7.2.6	连接目标网络服务器函数 connect()	199
7.2.7	写入数据函数 write()	200
7.2.8	读取数据函数 read()	201
7.2.9	关闭套接字函数	202
7.3	服务器/客户端的简单例子	202
7.3.1	例子功能描述	202
7.3.2	服务器网络程序	202
7.3.3	服务器读取和显示字符串	205
7.3.4	客户端的网络程序	205
7.3.5	客户端读取和显示字符串	206
7.3.6	编译运行程序	206
7.4	截取信号的例子	207
7.4.1	信号处理	207
7.4.2	信号 SIGPIPE	207
7.4.3	信号 SIGINT	208
7.5	小结	208
第 8 章	服务器和客户端信息的获取	209
8.1	字节序	209
8.1.1	大端字节序和小端字节序	209
8.1.2	字节序转换函数	211

8.1.3	一个字节序转换的例子.....	213
8.2	字符串 IP 地址和二进制 IP 地址的转换	216
8.2.1	inet_xxx()函数	216
8.2.2	inet_pton()和 inet_ntop()函数	218
8.2.3	使用 8.2.1 节地址转换函数的例子	219
8.2.4	使用函数 inet_pton()和函数 inet_ntop()的例子.....	221
8.3	套接字描述符判定函数 issockettype()	222
8.3.1	进行文件描述符判定的函数 issockettype().....	222
8.3.2	main()函数	223
8.4	IP 地址与域名之间的相互转换	223
8.4.1	DNS 原理	223
8.4.2	获取主机信息的函数.....	224
8.4.3	使用主机名获取主机信息的例子	227
8.4.4	函数 gethostbyname()不可重入的例子	229
8.5	协议名称处理函数	230
8.5.1	xxxprotoxxx()函数.....	231
8.5.2	使用协议族函数的例子.....	232
8.6	小结	235
第 9 章	数据的 IO 和复用	236
9.1	IO 函数.....	236
9.1.1	使用 recv()函数接收数据	236
9.1.2	使用 send()函数发送数据.....	238
9.1.3	使用 readv()函数接收数据	239
9.1.4	使用 writev()函数发送数据.....	239
9.1.5	使用 recvmsg()函数接收数据	241
9.1.6	使用 sendmsg()函数发送数据	243
9.1.7	IO 函数的比较.....	245
9.2	使用 IO 函数的例子.....	245
9.2.1	客户端处理框架的例子.....	245
9.2.2	服务器端程序框架.....	247
9.2.3	使用 recv()和 send()函数	248
9.2.4	使用 readv()和 write()函数	250
9.2.5	使用 recvmsg()和 sendmsg()函数.....	252
9.3	IO 模型.....	255
9.3.1	阻塞 IO 模型.....	255
9.3.2	非阻塞 IO 模型.....	255
9.3.3	IO 复用.....	256
9.3.4	信号驱动 IO 模型	256
9.3.5	异步 IO 模型.....	257

9.4	select()函数和 pselect()函数	258
9.4.1	select()函数	258
9.4.2	pselect()函数	260
9.5	poll()函数和 ppoll()函数	261
9.5.1	poll()函数	261
9.5.2	ppoll()函数	262
9.6	非阻塞编程	263
9.6.1	非阻塞方式程序设计介绍	263
9.6.2	非阻塞程序设计的例子	263
9.7	小结	264
第 10 章	基于 UDP 协议的接收和发送	265
10.1	UDP 编程框架	265
10.1.1	UDP 编程框图	265
10.1.2	UDP 服务器编程框架	267
10.1.3	UDP 客户端编程框架	267
10.2	UDP 协议程序设计的常用函数	267
10.2.1	建立套接字 socket()和绑定套接字 bind()	268
10.2.2	接收数据 recvfrom()/recv()	268
10.2.3	发送数据 sendto()/send()	273
10.3	UDP 接收和发送数据的例子	277
10.3.1	UDP 服务器端	277
10.3.2	UDP 服务器端数据处理	278
10.3.3	UDP 客户端	279
10.3.4	UDP 客户端数据处理	279
10.3.5	测试 UDP 程序	280
10.4	UDP 协议程序设计中的几个问题	280
10.4.1	UDP 报文丢失数据	280
10.4.2	UDP 数据发送中的乱序	282
10.4.3	UDP 协议中的 connect()函数	284
10.4.4	UDP 缺乏流量控制	285
10.4.5	UDP 协议中的外出网络接口	287
10.4.6	UDP 协议中的数据报文截断	288
10.5	小结	289
第 11 章	高级套接字	290
11.1	UNIX 域函数	290
11.1.1	UNIX 域函数的地址结构	290
11.1.2	套接字函数	291
11.1.3	使用 UNIX 域函数进行套接字编程	291
11.1.4	传递文件描述符	293
11.1.5	socketpair()函数	294

11.1.6	传递文件描述符的例子.....	295
11.2	广播.....	299
11.2.1	广播的 IP 地址.....	300
11.2.2	广播与单播的比较.....	300
11.2.3	广播的示例.....	301
11.3	多播.....	307
11.3.1	多播的概念.....	308
11.3.2	广域网的多播.....	308
11.3.3	多播的编程.....	308
11.3.4	内核中的多播.....	310
11.3.5	一个多播例子的服务器端.....	313
11.3.6	一个多播例子的客户端.....	315
11.4	数据链路层访问.....	317
11.4.1	SOCK_PACKET 类型.....	317
11.4.2	设置套接口以捕获链路帧的编程方法.....	317
11.4.3	从套接口读取链路帧的编程方法.....	318
11.4.4	定位 IP 包头的编程方法.....	319
11.4.5	定位 TCP 报头的编程方法.....	321
11.4.6	定位 UDP 报头的编程方法.....	322
11.4.7	定位应用层报文数据的编程方法.....	323
11.4.8	使用 SOCK_PACKET 编写 ARP 请求程序的例子.....	323
11.5	小结.....	326
第 12 章	套接字选项.....	328
12.1	获取和设置套接字选项 getsockopt()/setsockopt().....	328
12.1.1	getsockopt()函数和 setsockopt()函数的介绍.....	328
12.1.2	套接字选项.....	329
12.1.3	套接字选项简单示例.....	330
12.2	SOL_SOCKET 协议族选项.....	334
12.2.1	SO_BROADCAST 广播选项.....	334
12.2.2	SO_DEBUG 调试选项.....	335
12.2.3	SO_DONTROUTE 不经过路由选项.....	335
12.2.4	SO_ERROR 错误选项.....	335
12.2.5	SO_KEEPALIVE 保持连接选项.....	336
12.2.6	SO_LINGER 缓冲区处理方式选项.....	337
12.2.7	SO_OOBINLINE 带外数据处理方式选项.....	339
12.2.8	SO_RCVBUF 和 SO_SNDBUF 缓冲区大小选项.....	340
12.2.9	SO_RCVLOWAT 和 SO_SNDLOWAT 缓冲区下限选项.....	340
12.2.10	SO_RCVTIMEO 和 SO_SNDTIMEO 收发超时选项.....	341
12.2.11	SO_REUSEADDR 地址重用选项.....	341
12.2.12	SO_EXCLUSIVEADDRUSE 端口独占选项.....	342

12.2.13	SO_TYPE 套接字类型选项	342
12.2.14	SO_BSDCOMPAT 与 BSD 套接字兼容选项	342
12.2.15	SO_BINDTODEVICE 套接字网络接口绑定选项	343
12.2.16	SO_PRIORITY 套接字优先级选项	344
12.3	IPPROTO_IP 选项	344
12.3.1	IP_HDRINCL 选项	344
12.3.2	IP_OPTNOS 选项	344
12.3.3	IP_TOS 选项	344
12.3.4	IP_TTL 选项	345
12.4	IPPROTO_TCP 选项	345
12.4.1	TCP_KEEPAIVE 选项	345
12.4.2	TCP_MAXRT 选项	346
12.4.3	TCP_MAXSEG 选项	346
12.4.4	TCP_NODELAY 和 TCP_CORK 选项	346
12.5	使用套接字选项	348
12.5.1	设置和获取缓冲区大小	348
12.5.2	获取套接字类型的例子	353
12.5.3	使用套接字选项的综合例子	353
12.6	ioctl()函数	358
12.6.1	ioctl()函数的命令选项	358
12.6.2	ioctl()函数的 IO 请求	360
12.6.3	ioctl()函数的文件请求	362
12.6.4	ioctl()函数的网络接口请求	362
12.6.5	使用 ioctl()函数对 ARP 高速缓存操作	369
12.6.6	使用 ioctl()函数发送路由表请求	371
12.7	fcntl()函数	371
12.7.1	fcntl()函数的选项	372
12.7.2	使用 fcntl()函数修改套接字非阻塞属性	372
12.7.3	使用 fcntl()函数设置信号属主	372
12.8	小结	373
第 13 章	原始套接字	374
13.1	概述	374
13.2	原始套接字的创建	375
13.2.1	SOCK_RAW 选项	375
13.2.2	IP_HDRINCL 套接字选项	376
13.2.3	不需要 bind()函数	376
13.3	原始套接字发送报文	376
13.4	原始套接字接收报文	377
13.5	原始套接字报文处理时的结构	377
13.5.1	IP 头部的结构	377

13.5.2	ICMP 头部结构	378
13.5.3	UDP 头部结构	381
13.5.4	TCP 头部结构	382
13.6	ping 的例子	384
13.6.1	协议格式	384
13.6.2	校验和函数	385
13.6.3	设置 ICMP 发送报文的头部	386
13.6.4	剥离 ICMP 接受报文的头部	387
13.6.5	计算时间差	388
13.6.6	发送报文	389
13.6.7	接收报文	390
13.6.8	主函数过程	391
13.6.9	主函数 main()	393
13.6.10	编译测试	396
13.7	洪水攻击	396
13.8	ICMP 洪水攻击	397
13.8.1	ICMP 洪水攻击的原理	397
13.8.2	ICMP 洪水攻击的例子	397
13.9	UDP 洪水攻击	401
13.10	SYN 洪水攻击	405
13.10.1	SYN 洪水攻击的原理	405
13.10.2	SYN 洪水攻击的例子	405
13.11	小结	409
第 14 章	服务器模型选择	410
14.1	循环服务器	410
14.1.1	UDP 循环服务器	410
14.1.2	TCP 循环服务器	413
14.2	简单并发服务器	415
14.2.1	并发服务器的模型	416
14.2.2	UDP 并发服务器	416
14.2.3	TCP 并发服务器	419
14.3	TCP 的高级并发服务器模型	421
14.3.1	单客户端单进程，统一 accept()	422
14.3.2	单客户端单线程，统一 accept()	425
14.3.3	单客户端单线程，各线程独自 accept()，使用互斥锁	427
14.4	IO 复用循环服务器	431
14.4.1	IO 复用循环服务器模型介绍	431
14.4.2	IO 复用循环服务器模型例子	432
14.5	小结	436

第 15 章 IPv6 简介	437
15.1 IPv4 的缺陷	437
15.2 IPv6 的特点	438
15.3 IPv6 的地址	439
15.3.1 IPv6 的单播地址	439
15.3.2 可聚集全球单播地址	439
15.3.3 本地使用单播地址	440
15.3.4 兼容性地址	441
15.3.5 IPv6 多播地址	441
15.3.6 IPv6 任播地址	442
15.3.7 主机的多个 IPv6 地址	442
15.4 IPv6 的头部	443
15.4.1 IPv6 头部格式	443
15.4.2 与 IPv4 头部的对比	444
15.4.3 IPv6 的 TCP 头部	444
15.4.4 IPv6 的 UDP 头部	444
15.4.5 IPv6 的 ICMP 头部	445
15.5 IPv6 运行环境	447
15.5.1 加载 IPv6 模块	447
15.5.2 查看是否支持 IPv6	447
15.6 IPv6 的结构定义	448
15.6.1 IPv6 的地址族和协议族	448
15.6.2 套接字地址结构	448
15.6.3 地址兼容考虑	450
15.6.4 IPv6 通用地址	450
15.7 IPv6 的套接字函数	451
15.7.1 socket() 函数	451
15.7.2 没有发生改变的函数	451
15.7.3 发生改变的函数	452
15.8 IPv6 的套接字选项	452
15.8.1 IPv6 的套接字选项	452
15.8.2 单播跳限 IPV6_UNICAST_HOPS	453
15.8.3 发送和接收多播包	454
15.8.4 IPv6 中获得时间戳的 ioctl 命令	455
15.9 IPv6 的库函数	455
15.9.1 地址转换函数的差异	455
15.9.2 域名解析函数的差异	455
15.9.3 测试宏	458
15.10 IPv6 的编程的一个简单例子	458
15.10.1 服务器程序	458

15.10.2	客户端程序.....	460
15.10.3	编译调试.....	461
15.11	小结.....	462

第 3 篇 Linux 内核网络编程

第 16 章	Linux 内核中网络部分结构以及分布.....	464
16.1	概述.....	464
16.1.1	代码目录分布.....	464
16.1.2	内核中网络部分流程简介.....	466
16.1.3	系统提供修改网络流程点.....	468
16.1.4	sk_buff 结构.....	469
16.1.5	网络协议数据结构 inet_protosw.....	471
16.2	软中断 CPU 报文队列及其处理.....	473
16.2.1	Linux 内核网络协议层的层间传递手段——软中断.....	473
16.2.2	网络收发处理软中断的实现机制.....	475
16.3	socket 数据如何在内核中接收和发送.....	476
16.3.1	socket()的初始化.....	476
16.3.2	接收网络数据 recv().....	476
16.3.3	发送网络数据 send().....	477
16.4	小结.....	477
第 17 章	netfilter 框架内报文处理.....	478
17.1	netfilter.....	478
17.1.1	netfilter 简介.....	478
17.1.2	netfilter 框架.....	479
17.1.3	netfilter 检查时的表格.....	480
17.1.4	netfilter 的规则.....	480
17.2	iptables 和 netfilter.....	481
17.2.1	iptables 简介.....	481
17.2.2	iptables 的表和链.....	481
17.2.3	使用 iptables 设置过滤规则.....	483
17.3	内核模块编程.....	485
17.3.1	内核“Hello World!”程序.....	485
17.3.2	内核模块的基本架构.....	487
17.3.3	内核模块加载和卸载过程.....	489
17.3.4	内核模块初始化和清理函数.....	490
17.3.5	内核模块初始化和清理过程的容错处理.....	490
17.3.6	内核模块编译所需的 Makefile.....	491
17.4	5 个钩子点.....	492
17.4.1	netfilter 的 5 个钩子点.....	492

17.4.2	NF_HOOK 宏	493
17.4.3	钩子的处理规则	494
17.5	注册/注销钩子	494
17.5.1	结构 nf_hook_ops	494
17.5.2	注册钩子	495
17.5.3	注销钩子	496
17.5.4	注册注销函数	497
17.6	钩子的简单处理例子	498
17.6.1	功能描述	498
17.6.2	需求分析	498
17.6.3	ping 回显屏蔽实现	498
17.6.4	禁止向目的 IP 地址发送数据的实现	499
17.6.5	端口关闭实现	499
17.6.6	动态配置实现	499
17.6.7	可加载内核实现代码	501
17.6.8	应用层测试代码实现	508
17.6.9	编译运行	508
17.7	一点多个钩子的优先级	508
17.8	校验和问题	509
17.9	小结	510

第 4 篇 综合案例

第 18 章	一个简单 Web 服务器的例子 SHTTPD	512
18.1	SHTTPD 的需求分析	512
18.1.1	SHTTPD 启动参数可动态配置的需求	513
18.1.2	SHTTPD 的多客户端支持的需求	515
18.1.3	SHTTPD 支持方法的需求	515
18.1.4	SHTTPD 支持的 HTTP 协议版本的需求	516
18.1.5	SHTTPD 支持头部的需求	517
18.1.6	SHTTPD 定位 URI 的需求	517
18.1.7	SHTTPD 支持 CGI 的需求	518
18.1.8	SHTTPD 错误代码的需求	519
18.2	SHTTPD 的模块分析和设计	519
18.2.1	SHTTPD 的主函数	520
18.2.2	SHTTPD 命令行解析的分析设计	521
18.2.3	SHTTPD 配置文件解析的分析设计	523
18.2.4	SHTTPD 的多客户端支持的分析设计	523
18.2.5	SHTTPD 头部解析的分析设计	526
18.2.6	SHTTPD 对 URI 的分析设计	526

18.2.7	SHTTPD 支持方法的分析设计	527
18.2.8	SHTTPD 支持 CGI 的分析设计	527
18.2.9	SHTTPD 错误处理的分析设计	530
18.3	SHTTPD 各模块的实现.....	532
18.3.1	SHTTPD 命令行解析的实现	532
18.3.2	SHTTPD 文件配置解析的实现	535
18.3.3	SHTTPD 的多客户端支持的实现	536
18.3.4	SHTTPD 所请求 URI 解析的实现	540
18.3.5	SHTTPD 方法解析的实现	541
18.3.6	SHTTPD 响应方法的实现	541
18.3.7	SHTTPD 支持 CGI 的实现.....	545
18.3.8	SHTTPD 支持 HTTP 协议版本的实现	548
18.3.9	SHTTPD 内容类型的实现	548
18.3.10	SHTTPD 错误处理的实现	550
18.3.11	SHTTPD 生成目录下文件列表文件的实现	552
18.3.12	SHTTPD 主函数的实现	554
18.4	SHTTPD 的编译、调试和测试	555
18.4.1	建立源文件.....	555
18.4.2	制作 Makefile	555
18.4.3	制作执行文件.....	555
18.4.4	使用不同的浏览器测试服务器程序	556
18.5	小结	557
第 19 章	一个简单网络协议栈的例子 SIP	558
19.1	SIP 网络协议栈的功能描述	558
19.1.1	SIP 网络协议栈的基本功能描述.....	558
19.1.2	SIP 网络协议栈的分层功能描述.....	559
19.1.3	SIP 网络协议栈的用户接口功能描述.....	559
19.2	SIP 网络协议栈的架构.....	560
19.3	SIP 网络协议栈的存储区缓存.....	561
19.3.1	SIP 存储缓冲的结构定义.....	561
19.3.2	SIP 存储缓冲的处理函数.....	565
19.4	SIP 网络协议栈的网络接口层.....	567
19.4.1	SIP 网络接口层的架构.....	568
19.4.2	SIP 网络接口层的数据结构.....	568
19.4.3	SIP 网络接口层的初始化函数.....	570
19.4.4	SIP 网络接口层的输入函数.....	571
19.4.5	SIP 网络接口层的输出函数.....	574
19.5	SIP 网络协议栈的 ARP 层	577
19.5.1	SIP 地址解析层的架构.....	577
19.5.2	SIP 地址解析层的数据结构.....	577

19.5.3	SIP 地址解析层的映射表.....	579
19.5.4	SIP 地址解析层的 ARP 映射表维护函数.....	580
19.5.5	SIP 地址解析层的 ARP 网络报文构建函数.....	581
19.5.6	SIP 地址解析层的 ARP 网络报文收发处理函数.....	583
19.6	SIP 网络协议栈的 IP 层	586
19.6.1	SIP 网际协议层的架构.....	586
19.6.2	SIP 网际协议层的数据结构.....	587
19.6.3	SIP 网际协议层的输入函数.....	589
19.6.4	SIP 网际协议层的输出函数.....	593
19.6.5	SIP 网际协议层的分片函数.....	594
19.6.6	SIP 网际协议层的分片组装函数.....	595
19.7	SIP 网络协议栈的 ICMP 层	599
19.7.1	SIP 控制报文协议的数据结构.....	599
19.7.2	SIP 控制报文协议的协议支持.....	600
19.7.3	SIP 控制报文协议的输入函数.....	601
19.7.4	SIP 控制报文协议的回显应答函数.....	602
19.8	SIP 网络协议栈的 UDP 层	603
19.8.1	SIP 数据报文层的数据结构.....	603
19.8.2	SIP 数据报文层的控制单元.....	603
19.8.3	SIP 数据报文层的输入函数.....	605
19.8.4	SIP 数据报文层的输出函数.....	606
19.8.5	SIP 数据报文层的建立函数.....	606
19.8.6	SIP 数据报文层的释放函数.....	607
19.8.7	SIP 数据报文层的绑定函数.....	607
19.8.8	SIP 数据报文层的发送数据函数.....	608
19.8.9	SIP 数据报文层的校验和计算.....	609
19.9	SIP 网络协议栈的协议无关层	610
19.9.1	SIP 协议无关层的系统架构.....	611
19.9.2	SIP 协议无关层的函数形式.....	611
19.9.3	SIP 协议无关层的接收数据函数.....	612
19.10	SIP 网络协议栈的 BSD 接口层	613
19.10.1	SIP 用户接口层的架构.....	613
19.10.2	SIP 用户接口层的套接字建立函数.....	613
19.10.3	SIP 用户接口层的套接字关闭函数.....	614
19.10.4	SIP 用户接口层的套接字绑定函数.....	614
19.10.5	SIP 用户接口层的套接字连接函数.....	615
19.10.6	SIP 用户接口层的套接字接收数据函数	615
19.10.7	SIP 用户接口层的发送数据函数.....	616
19.11	SIP 网络协议栈的编译.....	617
19.11.1	SIP 的文件结构.....	617

19.11.2	SIP 的 Makefile	618
19.11.3	SIP 的编译运行	618
19.12	小结	618
第 20 章	一个简单防火墙的例子 SIPFW	620
20.1	SIPFW 防火墙的功能描述	620
20.1.1	SIPFW 防火墙对主机进行网络数据过滤的功能描述	620
20.1.2	SIPFW 防火墙用户设置防火墙规则的功能描述	621
20.1.3	SIPFW 防火墙配置文件等附加功能的功能描述	621
20.2	SIPFW 需求分析	621
20.2.1	SIPFW 防火墙条件和动作	621
20.2.2	SIPFW 防火墙支持过滤的类型和内容	622
20.2.3	SIPFW 防火墙过滤的方式和动作	625
20.2.4	SIPFW 防火墙的配置文件	626
20.2.5	SIPFW 防火墙命令行配置格式	627
20.2.6	SIPFW 防火墙的规则文件格式	628
20.2.7	SIPFW 防火墙的日志文件数据格式	630
20.2.8	SIPFW 防火墙构建所采用的技术方案	630
20.3	使用 netlink 进行用户空间和内核空间数据交互	631
20.3.1	netlink 的用户空间程序设计	632
20.3.2	netlink 的内核空间 API	635
20.4	使用 proc 进行内存数据用户空间映射	637
20.4.1	proc 虚拟文件系统的结构	637
20.4.2	创建 proc 虚拟文件	638
20.4.3	删除 proc 虚拟文件	639
20.4.4	proc 文件的写函数	639
20.4.5	proc 文件的读函数	640
20.5	内核空间的文件操作函数	641
20.5.1	内核空间的文件结构	641
20.5.2	内核空间的文件建立操作	641
20.5.3	内核空间的文件读写操作	642
20.5.4	内核空间的文件关闭操作	643
20.6	SIPFW 防火墙的模块分析和设计	644
20.6.1	SIPFW 防火墙的总体架构	644
20.6.2	SIPFW 防火墙的用户命令解析	645
20.6.3	SIPFW 用户空间与内核空间的交互	649
20.6.4	SIPFW 防火墙内核链上的规则处理	651
20.6.5	SIPFW 防火墙的 PROC 虚拟文件系统	654
20.6.6	SIPFW 防火墙的配置文件和日志文件处理	655
20.6.7	SIPFW 防火墙的过滤模块设计	657

20.7	SIPFW 防火墙各功能模块的实现.....	660
20.7.1	SIPFW 防火墙的命令解析代码.....	660
20.7.2	SIPFW 防火墙的过滤规则解析模块代码	664
20.7.3	SIPFW 防火墙的网络数据拦截模块代码	666
20.7.4	SIPFW 防火墙的 PROC 虚拟文件系统	668
20.7.5	SIPFW 防火墙对配置文件的解析.....	670
20.7.6	SIPFW 防火墙内核模块初始化和退出	671
20.7.7	用户空间处理主函数.....	672
20.8	编译、调试和测试.....	673
20.8.1	用户程序和内核程序的 Makefile.....	673
20.8.2	编译及运行	674
20.8.3	下发过滤规则，测试过滤结果	674
20.9	小结	676

第 1 篇 *Linux* 网络开发基础

- » 第 1 章 Linux 操作系统概述
- » 第 2 章 Linux 编程环境
- » 第 3 章 文件系统简介
- » 第 4 章 程序、进程和线程

第 1 章 Linux 操作系统概述

Linux 操作系统是目前发展最快的操作系统，自 1991 年诞生到现在的二十多年间，Linux 逐步完善和发展。Linux 操作系统在服务器、嵌入式等方面获得了长足的发展，并在个人操作系统方面有着广范的应用，这主要得益于其开放性。本章将对 Linux 的发展进行介绍，主要包括如下内容：

- ❑ Linux 发展的历史，以时间为主线对 Linux 的诞生进行介绍；
- ❑ 分析 Linux 和 UNIX 操作系统的异同；
- ❑ 介绍常用的几种 Linux 发行版本的特点；
- ❑ 对 Linux 操作系统的系统架构进行简单的介绍；
- ❑ 介绍 GNU 通用公共许可证及其特点。

通过本章的阅读，读者可以对 Linux 的发展历史和 Linux 操作系统的基本特点有一个简单的认识。

1.1 Linux 发展历史

Linux 操作系统于 1991 年诞生，目前已经成为主流的操作系统之一。其版本从开始的 0.01 版本到目前的 3.9.4 版本经历了二十多年的发展，从最初的蹒跚学步的“婴儿”成长为目前在服务器、嵌入式系统和个人计算机等多个方面得到广泛应用的操作系统。

1.1.1 Linux 的诞生和发展


Linux 的诞生和发展与个人计算机的发展历程是紧密相关的，特别是随着 Intel 的 i386 个人计算机的发展而逐步成熟。在 1981 年之前没有个人计算机，计算机是大型企业和政府部门才能使用的昂贵设备。IBM 公司在 1981 年推出了个人计算机 IBM PC，从而造成个人计算机的发展和普及。刚开始的时候，微软帮助 IBM 公司开发的 MS-DOS 操作系统在个人计算机中占有统治地位。随着 IT 行业的发展，个人计算机的硬件价格虽然逐年下降，但是软件特别是操作系统的价格一直居高不下。

与个人计算机对应，在大型机上的主流操作系统是 UNIX，而 UNIX 操作系统对操作系统的发展有诸多障碍：

- ❑ UNIX 的经销商为了寻求高利率，将价格抬得很高，个人计算机的用户根本不可能靠近它，不利于操作系统的普及。
- ❑ UNIX 操作系统的源代码具有版权，虽然贝尔实验室许可可以在大学的教学中使用 UNIX 源代码，但是因为版权问题源代码一直不能公开。对于广大的 PC 用户，软件行业的供应商一直没有一个很好的办法来解决 UNIX 操作系统普及性问题的

方法。

在操作系统的发展受到版权限制的时候，出现了 Minix 操作系统，这个操作系统由一本书来详细地描述它的实现原理。由于书中对 Minix 操作系统的描述非常详细，并且很有条理性，当时几乎全世界的计算机爱好者都在看这本书来理解操作系统的原理，其中包括 Linux 系统的创始者 Linus Torvalds。

 **注意：**当时苹果公司的 Mac 系列操作系统，不论从性能方面还是从用户的易用性方面来说都是最好的，但是其价格也是最高的。

1.1.2 Linux 名称的由来

Linux 操作系统的名称最初并没有被称做 Linux。Linus 给他的操作系统取的名字是 Freax，这个单词的含义是怪诞的、怪物、异想天开的意思。当 Torvalds 将他的操作系统上传到服务器 `ftp.funet.fi` 上的时候，这个服务器的管理员 Ari Lemke 对 Freax 这个名称很不赞成，所以将操作系统的名称改为了 Linus 的谐音 Linux，于是这个操作系统的名称就以 Linux 流传下来。

在 Linus 的自传《Just for Fun》一书中，Linus 解释说：“Ari Lemke，他十分不喜欢 Freax 这个名字。倒喜欢我当时正在使用的另一个名字 Linux，并把我的邮件路径命名为 pub OS/Linux。我承认我并没有太坚持，但这一切都是他搞的，所以我既可以不惭愧地说自己不是那么以个人为中心，但是也有一点个人的荣誉感。而且个人认为，Linux 是个不错的名字。”实际上，在早期的源文件中仍然使用 Freax 作为操作系统的名字，可以从 Makefile 文件中看出此名称的一些蛛丝马迹。

关于 Linux 的发音有各种说法，例如[ˈlinʌks]，但是按照 Torvalds 的说法，Linux 中 Li 中 i 的发音类似于 Minix 中 i 的发音，而 nux 中 u 的发音类似于英文单词 pronounce 中第一个 o 的发音。根据 Torvalds 对此的解释，依照国际音标其发音为[ˈlinəks]，与“喱呐科斯”类似。在网络上有一份 Torvalds 本人说话的音频，音频中的内容为“Hello, this is Linus Torvalds, and I pronounce Linux as Linux”，其下载网络地址为 <http://www.paul.sladen.org/pronunciation/torvalds-says-linux.wav>。

对于 Linux 发音的解释，还有一份 Torvalds 本人的解说片段，这一片段发音的视频可以从如下的 URL 下载：<http://www.Linuxweblog.com/Linux-pronunciation>。

1.2 Linux 的发展要素

Linux 操作系统是 UNIX 的一种典型的克隆系统。在 Linux 诞生之后，借助于 Internet 网络，在全世界计算机爱好者的共同努力下，成为目前世界上使用者最多的一种类似 UNIX 的操作系统。在 Linux 操作系统的诞生、成长和发展过程中，以下 5 个方面起到了重要的作用：UNIX 操作系统、Minix 操作系统、GNU 计划、POSIX 标准和 Internet 网络。

1.2.1 UNIX 操作系统

UNIX 操作系统于 1969 年在 Bell 实验室诞生，它是美国贝尔实验室的 Ken.Thompson

和 Dennis Ritchie 在 DEC PDP-7 小型计算机系统上开发的一种分时操作系统。

Ken Thompson 开发 UNIX 操作系统的初衷是为了能在一台闲置的 PDP-7 计算机上运行星际旅行游戏。他在 1969 年夏天花费一个月的时间开发出了 UNIX 操作系统的原型。最开始,开发 UNIX 操作系统使用的是 BCPL 语言(即通常所说的 B 语言),后来 Dennis Ritchie 于 1972 年使用 C 语言对 UNIX 操作系统进行了改写。同时 UNIX 操作系统在大学中得到广泛的推广,并将 UNIX 的授权分发给多个商业公司。

自从 UNIX 操作系统从实验室走出来之后,得到了长足的发展。目前已经成为大型系统的主流操作系统,现在几乎每个主要的计算机厂商都有其自有版本的 UNIX。UNIX 是一个功能强大、性能全面的、多用户、多任务的分时操作系统,在从巨型计算机到普通 PC 等多种不同的平台上,都有着十分广泛的应用。

通常情况下,比较大型的系统应用,例如银行和电信部门,一般都采用固定机型的 UNIX 解决方案:在电信系统中以 SUN(SUN 公司已经被 Oracle 公司收购)的 UNIX 系统方案居多,在民航里以 HP 的系统方案居多,在银行里以 IBM 的系统方案居多。

Linux 是一种 UNIX 的克隆系统,采用了几乎一致的系统 API 接口。特别是网络方面,二者接口的应用程序几乎完全一致。

1.2.2 Minix 操作系统

Minix 操作系统也是 UNIX 操作系统的一种克隆系统,它由荷兰 Amsterdam 的 Vrije 大学著名教授 Andrew S.Tanenbaum 于 1987 年开发完成。Minix 操作系统主要用于学生学习操作系统原理时的教学。在当时 Minix 操作系统在大学中是免费使用的,但是其他用途则需要收费。目前 Minix 操作系统已经全部是免费的,可以从许多 FTP 上下载,目前 Minix 3 是主流版本。

由于 Minix 操作系统提供源代码,并且与操作系统相结合,有一本高质量的书籍介绍其实现原理,在当时全世界的大学中形成了学习 Minix 操作系统的风气,Linus 刚开始就是参照此系统在 1991 年开始开发 Linux 的。

实际上,Minix 操作系统并不是很优秀,但是这个操作系统提供了 C 语言和汇编语言的源代码。而当时的 UNIX 操作系统源代码除了极少的范围外一直是保密的,Minix 操作系统对程序员来说是一个福音。为了可以让学生在一個学期内能够学完操作系统的课程,AST 保持了 Minix 操作系统的小型化,没有接受各界对 Minix 扩展的要求,而正是这个原因激发了 Linus 编写 Linux 操作系统。

1.2.3 POSIX 标准

POSIX (Portable Operating System Interface for Computing Systems) 是由 IEEE 和 ISO/IEC 开发的一套标准。POSIX 标准是对 UNIX 操作系统的经验和实践的总结,对操作系统调用的服务接口进行了标准化,保证所编制的应用程序在源代码一级可以在多种操作系统上进行移植。

在 20 世纪 90 年代初,POSIX 标准的制定处于最后确定的投票阶段,而 Linux 正处于开始的诞生时期。作为一个指导性的纲领性标准,Linux 的接口与 POSIX 相兼容。

1.3 Linux 与 UNIX 的异同

Linux 是 UNIX 操作系统的一个克隆系统, 没有 UNIX 就没有 Linux。但是, Linux 和传统的 UNIX 有很大的不同, 两者之间的最大区别是关于版权方面的: Linux 是开放源代码的自由软件, 而 UNIX 是对源代码实行知识产权保护的传统商业软件。两者之间还存在如下的区别:

- ❑ UNIX 操作系统大多数是与硬件配套的, 操作系统与硬件进行了绑定; 而 Linux 则可运行在多种硬件平台上。
- ❑ UNIX 操作系统是一种商业软件(授权费大约为 5 万美元); 而 Linux 操作提供则是一种自由软件, 是免费的, 并且公开源代码。
- ❑ UNIX 的历史要比 Linux 悠久, 但是 Linux 操作系统由于吸取了其他操作系统的经验, 其设计思想虽然源于 UNIX 但是要优于 UNIX。
- ❑ 虽然 UNIX 和 Linux 都是操作系统的名称, 但 UNIX 除了是一种操作系统的名称外, 作为商标, 它归 SCO 所有。
- ❑ Linux 的商业化版本有 Red Hat Linux、SuSe Linux、slakeware Linux、国内的红旗 Linux 等, 还有 Turbo Linux; UNIX 主要有 Oracle 的 Solaris, IBM 的 AIX, HP 的 HP-UX, 以及基于 x86 平台的 SCO UNIX/UNIXware。
- ❑ Linux 操作系统的内核是免费的; 而 UNIX 的内核并不公开。
- ❑ 在对硬件的要求上, Linux 操作系统要比 UNIX 要求低, 并且没有 UNIX 对硬件要求的那么苛刻; 在对系统的安装难易度上, Linux 比 UNIX 容易得多; 在使用上, Linux 相对没有 UNIX 那么复杂。

总体来说, Linux 操作系统无论在外观上还是在性能上都与 UNIX 相同或者比 UNIX 更好, 但是 Linux 操作系统不同于 UNIX 的源代码。在功能上, Linux 仿制了 UNIX 的一部分, 与 UNIX 的 System V 和 BSD UNIX 相兼容。在 UNIX 上可以运行的源代码, 一般情况下在 Linux 上重新进行编译后就可以运行, 甚至 BSD UNIX 的执行文件可以在 Linux 操作系统上直接运行。

1.4 操作系统类型选择和内核版本的选择

要在 Linux 环境下进行程序设计, 首先要选择一款适合自己的 Linux 操作系统。本节对常用的发行版本和 Linux 内核进行了介绍, 并简要讲解了如何定制自己的 Linux 操作系统。

1.4.1 常见的不同公司发行的 Linux 异同

Linux 的发行版本众多, 曾有人收集过超过 300 种的发行版本。当然, 不能在本书中介绍众多的发行版特点, 这超出了本书的范围。本小节将对最常用的发行版本进行简单的介绍, 表 1.1 为用户经常使用的版本。读者可以去相关网址查找, 选择适合的版本使用。本书所使用的 Linux 为 Ubuntu。

表 1.1 常用 Linux 发行版本特点

版本名称	网 址	特 点	软件包管理器
Debian Linux	www.debian.org	开放的开发模式，并且易于进行软件包升级	apt
Fedora Core	www.redhat.com	拥有数量庞大的用户，优秀的社区技术支持，并且有许多创新	up2date (rpm), yum (rpm)
CentOS	www.centos.org	CentOS 是一种对 RHEL (Red Hat Enterprise Linux) 源代码再编译的产物，由于 Linux 是开发源代码的操作系统，并不排斥基于源代码的再分发，CentOS 就是将商业的 Linux 操作系统 RHEL 进行源代码再编译后分发，并在 RHEL 的基础上修正了不少已知的漏洞	rpm
SUSE Linux	www.suse.com	专业的操作系统，易用的 YaST 软件包管理系统	YaST (rpm)，第 三方 apt (rpm) 软 件库 (repository)
Mandriva	www.mandriva.com	操作界面友好，使用图形配置工具，有庞大的社区进行技术支持，支持 NTFS 分区的大小变更	rpm
KNOPPIX	www.knoppix.com	可以直接在 CD 上运行，具有优秀的硬件检测和适配能力，可作为系统的急救盘使用	apt
Gentoo Linux	www.gentoo.org	高度的可定制性，使用手册完整	portage
Ubuntu	www.ubuntu.com	优秀易用的桌面环境，基于 Debian 构建	apt

1.4.2 内核版本的选择

内核是 Linux 操作系统的最重要的部分，从最初的 0.95 版本到目前的 3.9.4 版本，Linux 内核开发经过了 20 多年的时间，其架构已经十分稳定。Linux 内核的编号采用如下编号形式：


主版本号.次版本号.主补丁号.次补丁号

说明：在 2011 年，Linux Kernel 3.0 发布。随后，一系列以 3 开头的内核版本被发布更新。

例如 2.6.34.14 各数字的含义如下：

- ❑ 第 1 个数字 (2) 是主版本号，表示第 2 大版本；
- ❑ 第 2 个数字 (6) 是次版本号，有两个含义：既表示是 Linux 内核大版本的第 6 个小版本，同时因为 6 是偶数表示为发布版本（奇数表示测试版）；
- ❑ 第 3 个数字 (34) 是主版本补丁号，表示指定小版本的第 34 个补丁包；
- ❑ 第 4 个数字 (14) 是次版本补丁号，表示次补丁号的第 3 个小补丁。

在安装 Linux 操作系统的时候，最好不要采用发行版本号中的小版本号是奇数的内核，因为开发中的版本没有经过比较完善的测试，有一些漏洞是未知的，有可能造成使用中不必要的麻烦。

 **注意：**Debian Linux 内核的版本稍有不同，如 2.6.18-3，可以发现多了一组数字 (3)，该数字是构建号。每个构建号可以增加少量新的驱动程序或缺陷修复。

Linux 内核版本的开发源代码树目前比较通用的是 2.6.xx 的版本,当然,有部分 2.4 的版本仍在使用。与 2.4 版本的内核相比较,2.6 版本内核具有如下优势:

- ❑ 支持绝大多数的嵌入式系统,加入了之前嵌入式系统经常使用的 μ Clinux 的大部分代码,并且子系统的支持更加细化,可以支持硬件体系结构的多样性,可抢占内核的调度方式支持实时系统,可定制内核。
- ❑ 支持目前最新的 CPU,例如 Intel 的超线程、可扩展的地址空间访问。
- ❑ 驱动程序框架变更,例如用 .ko 替代了原来的 .o 方式,消除内核竞争,更加透明的子模块方式。
- ❑ 增加了更多的内核级的硬件支持。

本书中的环境对 Linux 的内核没有特殊要求,因此读者在选择内核版本的时候不需要重新编译内核,使用操作系统自带的内核就可以满足需要。本书作者的操作系统内核为 Linux-3.2.0-44。

1.5 Linux 的系统架构

Linux 系统从应用角度来看,分为内核空间和用户空间两个部分。内核空间是 Linux 操作系统的主要部分,但是仅有内核的操作系统是不能完成用户任务的。丰富并且功能强大的应用程序包是一个操作系统成功的必要条件。

1.5.1 Linux 内核的主要模块

Linux 的内核主要由 5 个子系统组成:进程调度、内存管理、虚拟文件系统、网络接口、进程间通信。下面将依次讲解这 5 个子系统。

1. 进程调度 SCHED

进程调度指的是系统对进程的多种状态之间转换的策略。Linux 下的进程调度有 3 种策略: SCHED_OTHER、SCHED_FIFO 和 SCHED_RR。

- ❑ SCHED_OTHER 是用于针对普通进程的时间片轮转调度策略。这种策略中,系统给所有的运行状态的进程分配时间片。在当前进程的时间片用完之后,系统从进程中优先级最高的进程中选择进程运行。
- ❑ SCHED_FIFO 是针对运行的实时性要求比较高、运行时间短的进程调度策略。这种策略中,系统按照进入队列的先后进行进程的调度,在没有更高优先级进程到来或者当前进程没有因为等待资源而阻塞的情况下,会一直运行。
- ❑ SCHED_RR 是针对实时性要求比较高、运行时间比较长的进程调度策略。这种策略与 SCHED_OTHER 的策略类似,只不过 SCHED_RR 进程的优先级要高得多。系统分配给 SCHED_RR 进程时间片,然后轮循运行这些进程,将时间片用完的进程放入队列的末尾。

由于存在多种调度方式, Linux 进程调度采用的是“有条件可剥夺”的调度方式。普通进程中采用的是 SCHED_OTHER 的时间片轮循方式,实时进程可以剥夺普通进程。如果普通进程在用户空间运行,则普通进程立即停止运行,将资源让给实时进程;如果普通

进程运行在内存空间，需要等系统调用返回用户空间后方可剥夺资源。

2. 内存管理 MMU

内存管理是多个进程间的内存共享策略。在 Linux 系统中，内存管理的主要概念是虚拟内存。

虚拟内存可以让进程拥有比实际物理内存更大的内存，可以是实际内存的很多倍。每个进程的虚拟内存有不同的地址空间，多个进程的虚拟内存不会冲突。

虚拟内存的分配策略是每个进程都可以公平地使用虚拟内存。虚拟内存的大小通常设置为物理内存的两倍。

3. 虚拟文件系统 VFS

在 Linux 下支持多种文件系统，如 ext、ext2、minix、umsdos、msdos、vfat、ntfs、proc、smb、ncp、iso9660、sysv、hpfs、affs 等。目前 Linux 下最常用的文件格式是 ext2 和 ext3。ext2 文件系统用于固定文件系统和可活动文件系统，是 ext 文件系统的扩展。ext3 文件系统是在 ext2 上增加日志功能后的扩展，它兼容 ext2。两种文件系统之间可以互相转换，ext2 不用格式化就可以转换为 ext3 文件系统，而 ext3 文件系统转换为 ext2 文件系统也不会丢失数据。

4. 网络接口

Linux 是在 Internet 飞速发展的时期成长起来的，所以 Linux 支持多种网络接口和协议。网络接口分为网络协议和驱动程序，网络协议是一种网络传输的通信标准，而网络驱动则是对硬件设备的驱动程序。Linux 支持的网络设备多种多样，几乎目前所有网络设备都有驱动程序。

5. 进程间通信

Linux 操作系统支持多进程，进程之间需要进行数据的交流才能完成控制、协同工作等功能，Linux 的进程间通信是从 UNIX 系统继承过来的。Linux 下的进程间的通信方式主要有管道方式、信号方式、消息队列方式、共享内存和套接字等方法。

1.5.2 Linux 的文件结构

与 Windows 下的文件组织结构不同，Linux 不使用磁盘分区符号来访问文件系统，而是将整个文件系统表示成树状的结构，Linux 系统每增加一个文件系统都会将其加入到这个树中。

操作系统文件结构的开始，只有一个单独的顶级目录结构，叫做根目录。所有一切都从“根”开始，用“/”代表，并且延伸到子目录。DOS/Windows 下文件系统按照磁盘分区概念分类，目录都存于分区上。Linux 则通过“挂接”的方式把所有分区都放置在“根”下各个目录里。一个 Linux 系统的文件结构如图 1.1 所示。

不同的 Linux 发行版本的目录结构和具体的实现功能存在一些细微的差别。但是主要的功能都是一致的。一些常用目录的作用如下所述。

- ❑ `/etc`: 包括绝大多数 Linux 系统引导所需要的配置文件, 系统引导时读取配置文件, 按照配置文件的选项进行不同情况的启动, 例如 `fstab`、`host.conf` 等;
- ❑ `/lib`: 包含 C 编译程序需要的函数库, 是一组二进制文件, 例如 `glibc` 等;
- ❑ `/usr`: 包括所有其他内容, 如 `src`、`local`。Linux 的内核就在 `/usr/src` 中。其下有子目录 `/bin`, 存放所有安装语言的命令, 如 `gcc`、`perl` 等;
- ❑ `/var`: 包含系统定义表, 以便在系统运行改变时可以只备份该目录, 如 `cache`;
- ❑ `/tmp`: 用于临时性的存储;
- ❑ `/bin`: 大多数命令存放在这里;
- ❑ `/home`: 主要存放用户账号, 并且可以支持 `ftp` 的用户管理。系统管理员增加用户时, 系统在 `home` 目录下创建与用户同名的目录, 此目录下一般默认有 `Desktop` 目录;
- ❑ `/dev`: 这个目录下存放一种设备文件的特殊文件, 如 `fd0`、`had` 等;
- ❑ `/mnt`: 在 Linux 系统中, 它是专门给外挂的文件系统使用的, 里面有两个文件 `cdrom`、`floopy`, 登录光驱、软驱时要用到。

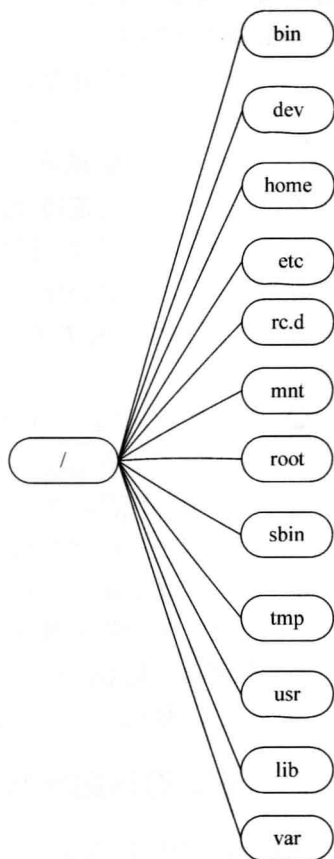


图 1.1 Linux 文件系统结构示意图

刚开始使用 Linux 的人比较容易混淆的是 Linux 下使用斜杠“/”，而在 DOS/Windows 下使用的是反斜杠“\”。例如在 Linux 中，由于从 UNIX 集成的关系，路径用“`/usr/src/Linux`”表示，而在 Windows 下则用“`\usr\src\Linux`”表示。在 Linux 下更加普遍的问题是字母大小写敏感，例如文件 `Hello.c` 和文件 `hello.c` 在 Linux 下不是一个文件，而在 Windows 下则表示同一个文件。

1.6 GNU 通用公共许可证

GNU 通用公共许可证（简称为 GPL）是由自由软件基金会发行的用于计算机软件的一种许可证制度。GPL 最初是由 Richard Stallman 为 GNU 计划而撰写。目前，GNU 通行证被绝大多数的 GNU 程序和超过半数的自由软件采用。此许可证最新版本为“版本 3”，于 2007 年发布。GNU 宽通用公共许可证（简称 LGPL）是由 GPL 衍生出的许可证，被用于一些 GNU 程序库。

1.6.1 GPL 许可证的历史

GNU 通用公共许可证是由 Richard Stallman 为了 GNU 计划而撰写的，它以 GNU 的 Emacs、GDB、GCC 的早期许可证为蓝本。上述的这些许可证都包含了一些 GPL 中的版权

思想，但是仅仅针对特定的某个程序。Richard Stallman 的目标是创造出一种通用的软件许可证制度，来为所有的开源软件代码计划使用。

GPL 的“版本 1”，在 1989 年 1 月诞生。在 1990 年时，因为一些共享库的使用而出现了对于 GPL 许可证制度更为宽松的需求，在 GPL “版本 2”于 1991 年 6 月发布时，另一许可证——库通用许可证（Library General Public License，简称 LGPL）也随之发布，并记做“版本 2”以示对 GPL 的补充。在 LGPL 版本 2.1 发布时与 GPL 版本不再对应，而 LGPL 也被重命名为 GNU 宽通用公共许可证（Lesser General Public License）。

GPLv3 在 2007 年 6 月份开始使用，由于对专利权和数字版权限制的问题造成了自由软件阵营的一次很大的争论。Stallman 于 2006 年 2 月 25 日在自由开源软件开发者欧洲会议上发表的演讲中，对 GPLv3 的特点作了解释，相对于 GPLv2，主要有如下 4 个不同的方面：

- ❑ 数字版权问题。在 GPLv3 中禁止使用 GPLv3 本身作为数字版权的一部分，同时消费类电子设备上使用 GPLv3 代码必须开放源代码，而且允许用户自己重新构建。
- ❑ 专利扩散许可。在 GPLv3 中如果具有专利的代码加入之后，此专利会自动向整个应用程序授权此专利。
- ❑ 衍生产品的定义。在 GPLv3 中定义了衍生产品，即如果某个模块采用了 GPLv3 协议，某个产品使用此模块动态链接后，如果此模块可以被其他模块代替，则这个产品不是 GPLv3 协议，否则需要采用 GPLv3 协议。
- ❑ GPLv3 协议与其他协议的兼容问题。

1.6.2 GPL 的自由理念

软件的版权保护机制在保护发明人权益的同时，对软件的技术进步造成了影响。版权所有软件的最终用户几乎不能从所购买的软件中得到任何软件设计相关的权利（除了使用的权利），甚至可能限制像逆向工程等法律允许范围内的行为。与此对应，GPL 授予程序的接受方下述的权利，即 GPL 所倡导的“自由”：

- ❑ 可以以任何目的运行所购买的程序；
- ❑ 在得到程序代码的前提下，可以以学习为目的，对源程序进行修改；
- ❑ 可以对复制件进行再发行；
- ❑ 对所购买的程序进行改进，并进行公开发布。

自由软件许可证除了 GPL 许可证之外，还有一些其他的许可证，如 BSD、APACHE 等许可证。一些许可证比 GPL 的许可证的限制要少得多，例如 BSD 许可证并不禁止其演绎作品变成版权所有软件。它们之间的最主要区别是 GPL 提供一种软件复制和演绎产品的许可证继承保证。Stallman 发明了一种叫做 Copyleft 的法律机制，要求所有 GPL 程序的演绎作品也要在 GPL 许可证之下。

目前，GPL 许可证是自由软件和开源软件的最流行许可证。到 2004 年 4 月为止，GPL 许可证已占 freshmeat（最大的 UNIX 平台和跨平台软件网络发布平台）上所列的自由软件的 75%，SourceForge 上所列软件的 68%。GNU 软件中最著名的 GPL 自由软件包括 Linux 内核和 GCC 编译器包。

1.6.3 GPL 的基本条款

GPL 许可证作为 Linux 平台软件的主要许可证，有很多独特的地方。GPL 授权的软件并不是说使用者在得到此软件后可以无限制地使用，而是要遵循一定的规则，其中主要的一点就是开放源代码。使用 GPL 授权发布的商业软件，也并不是不要钱，其盈利模式是采用收取服务费用的方式来获取利益。GPL 中的主要条款包括权利授予、Copyleft。

1. 授予的权利

采用 GPL 条款的软件分发给使用人，不管是收费还是免费，其作品符合 GPL 授权，获得 GPL 作品的人成为许可证接受人。许可证接受人有修改、复制、再发行此作品或者此作品的演绎版本的权利，许可证接受人可以由上述的行为收取费用而获利。与一般禁止商业用途的软件不同，GPL 授权的软件不禁止商业用途，例如 Stallman 最初的 Emacs 就是收取费用的，每份 150 美元。

GPL 的授权通常被人理解为免费，其实这是两种完全不同的概念：GPL 在出售产品的同时需要提供源代码，同时允许获得软件的产品进行再次发布。一般的 GPL 分发软件的盈利模式是采用服务的方式，即如果想更好地使用此软件，需要向分发者提供报酬，分发者对使用者的软件进行优化或者进行人员培训等工作。例如 IBM 提供的软件中就有 GPL 协议的，但是 IBM 是典型的服务获利的公司。

GPL 授权的另一层含义是要求分发者提供源代码，防止软件开发商对软件进行锁定，限制用户的某些行为。如果用户获得源代码，在分析源代码的基础上，可以修改某些设置，对源软件进行功能开放。

2. Copyleft

GPL 许可证不是授予许可证接受人无限制的权利，接受人在因为 GPL 而获益的时候（获得软件产品的源代码）必须遵守一定的要求。GPL 协议要求许可证的接受人在进行软件再次发布的时候必须要公开源代码，同时允许对再发行软件进行的复制、发行、修改等的权利，即再发行的软件必须为 GPL 许可证。

上述的这种要求称为 Copyleft，GPL 由此而被称为“被黑的版权法”。因为 GPL 的法理基础是承认软件是拥有版权的，即作品在法律上归版权所有。由于软件的版权由发行者所有，所以发行者可以对软件的发行规定进行设置，GPL 就是发行者对版权进行上述规定，放弃一定的版权。如果某个再发行版本不遵循 GPL 许可证，因为原作者对作品拥有版权，这样就有可能被原作者起诉。

GPL 的 copyleft 仅仅在程序的再发行时发生作用，如果受权人对软件进行修改后没有进行发行，是可以不开放源代码的。Copyleft 只对发行的软件本身起作用，对于软件的输出或者工作成果不起作用。

GPL 软件的发行方法都是把源代码和可执行程序一同发行，一般提供例如 CD 等。目前通行的发行 GPL 软件的方法是将软件放置到互联网上，由用户来下载，例如 HTTP、FTP 等方式。

1.6.4 关于 GPL 许可证的争议

使用 GPL 的许可证造成了很多争议，主要是对软件版权方面的界定、GPL 的软件传染性、商业开发方面的困扰等。比较有代表性的是对 GPL 软件产品的链接库使用的产品版权界定，即非 GPL 软件是否可以链接到 GPL 的库程序。

对于 GPL 开放源代码进行修改的产品遵循 GPL 的授权规定是很明确的，但是对于使用 GPL 链接库的产品是否需要遵循 GPL 存在很大分歧，但是其他专家并不认同这种观点，分成了自由和开放源代码社区两派。这个问题其实不是技术问题，这是一个法律界定的问题，需要法律的案例来例证。

由于 GPL 许可证需要授权人对再发行产品按照 GPL 许可证发行，所以在使用许可证软件的时候需要注意。有很多协议是 GPL 兼容的，即这种协议和 GPL 协议的软件共同使用，并且将开发完毕的软件产品作为 GPL 来发行是没有问题的，例如 MIT/X 许可证、BSD 许可证、LGPL，它们和 GPL 许可证兼容；有一些许可证是 GPL 不兼容的，例如某些自由软件的许可证。开发者在开发的时候，要使用 GPL 兼容的许可证，以免引起法律问题。

GPL 再发行软件必须采取 GPL 许可证的问题，被微软的首席执行官 Steve Ballmer 称为“癌症”，认为 GPL 是有“传染性”的“病毒”。由于包含 GPL 代码或动态链接到 GPL 库被理解为“演绎作品”，必须按照 GPL 许可证的强制继承来使用 GPL 分发。微软已经以 GPL 为许可证发行了 SFU (Microsoft Windows Services for UNIX) 中所包含的部分组件，例如 GCC 编译器。

1.7 Linux 软件开发的可借鉴之处

在 Linux 的发展过程中，形成了一种独特的成功模式，包含软件的开发模式。Linux 操作系统的成功从一个系统的角度看有很多值得项目管理人员学习的东西，例如《大教堂与集市》一书中对 Linux 的开发模式进行了比较详细的分析，它主要包含如下几个方面。

- ❑ 使用集市模式进行软件开发应该有一个基本成型的软件原型，这样后来的参与者能够对此进行改进，更重要的是能够看到成功的曙光、可以看到不久的将来能够成功，获得参与的动力。
- ❑ 集市模式的开发把软件的使用者作为开发的协作者而不仅仅是一个简单的用户，这样开发者和使用者能够共同对作品进行快速的代码改进和高效率的调试。
- ❑ 集市模式开发使用早发布、常发布的方法，来方便听取客户的建议，对软件进行改进。项目的开发者想出好主意是件好事，而从使用者那里获得的建议是比前者更好的事情。因为从使用者那里提出的建议是有的放矢，更加切合实际的。
- ❑ 集市的开发模式验证了如下一个成功的假设：如果参与软件 Beta 版测试的人员足够多，几乎软件中所有存在的问题都能够被迅速地找出并进行纠正。
- ❑ 对于集市开发模式的项目来说，比技能和设计能力更为重要的是项目协调人员必须具有良好的人际和交流能力。因为为了建造一个成功的开发小组，需要项目的领导人员所作所为必须让参与者感兴趣并能够有参与的动力，使得参与者感到他们正在做的工作十分有趣（这是因为一般的项目是没有报酬的，大家按照兴趣参

加)，这不仅仅是项目的本身，与领导者的个人素质有很大的关系。

从 Linux 社区中还可以获得更多睿智的经验或者知识，例如 Linus 所持的一种观点：使用聪明的数据结构和笨拙的代码的搭配方式要比相反的搭配方式更好，可以作为软件开发的一种基本的常识。

1.8 小 结

本章对 Linux 的形成历史进行了简单的介绍，并对其发展历程中起重要作用的 5 个要素进行了解释。与 UNIX 系统相比较，Linux 操作系统有很多不同之处，特别是在版权方面。

Linux 的发行版本数以百计，其中的 Debian、Fedora Core、openSUSE，以及 Ubuntu 是比较有代表性的集中。本书中的例子均以 Ubuntu 为例进行介绍。本章还介绍了 Linux 的系统架构和 Linux 内核模块之间的关系，对 GNU 的通用公共许可证进行了介绍，特别是 GNU 的 Copyleft 概念。最后介绍了 Linux 开发模式的成功之处，对集市开发模式进行了简单的介绍。

第 2 章 Linux 编程环境

在第 1 章中对 Linux 的发展历史和特点进行了简单的介绍，要在 Linux 环境下进行程序设计，还需要对 Linux 的环境有所了解。本章对 Linux 的编程环境进行介绍，通过本章的学习，读者将能够在 Linux 环境下编写、编译和调试程序。在 Linux 环境下进行程序开发时，除了需要有一个可运行的 Linux 环境，还需要具有如下的基本知识：Linux 命令行的环境和登录方式；Bash Shell 的使用。

在具有以上条件的基础上，对编程基本知识有了基本的了解后，就可以进行编程工作了。本章对 Linux 操作系统下的编程环境进行介绍，主要包括如下内容：

- ❑ 了解如何使用 Linux 下常用的编辑器，主要对 Vim 进行介绍；
- ❑ 了解 GCC 编译、链接，懂得如何进行编译程序，并能根据情况进行优化和修改代码；
- ❑ Makefile 的编写；
- ❑ 理解程序的编译链接和执行的过程，这对程序的编写有很大的帮助；
- ❑ 了解在 Linux 环境下如何调试程序，能够使用 GDB 调试程序。

2.1 Linux 环境下的编辑器

在 Linux 环境下有很多编译器，例如基于行的编辑器 `ed` 和 `ex`，基于文本的编辑器 `Vim`、`Emacs` 等。使用文本编辑器可以帮助用户翻页、移动光标、查找字符、替换字符、删除等操作。本节中对 `Vim` 编辑器进行详细的介绍，并简单介绍其他的编辑器。

2.1.1 Vim 使用简介

`Vi` 是 `visual editor` 的简写，发音为[vi'ai]，是 UNIX 系统下最通用的文本编辑器。`Vi` 不是一个所见即所得的编辑器，如果要进行复制和格式化文本需要手动输入命令进行操作。安装好 Linux 操作系统后，一般已经默认安装了 `Vi` 编辑器。为了使用方便，建议安装 `Vi` 的扩展版本 `Vim`，它比 `Vi` 更强大，更加适合初学者使用。

1. Vim 的安装

在介绍如何使用 `Vim` 编译器之前需要先安装 `Vim` 软件包，如果没有安装 `Vim`，可以使用如下命令进行安装。

```
#apt-get install vim
Reading package lists... Done
Building dependency tree... Done
Suggested packages:
```

(使用 apt-get 命令安装 vi)
(检查软件包列表)
(建立依赖)

```

ctags vim-doc vim-scripts
The following NEW packages will be installed:      (如下软件包将被安装)
vim                                              (vim 软件包)
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
                                                    (软件包变更统计)

Need to get 0B/745kB of archives.
After unpacking 1438kB of additional disk space will be used.
Selecting previously deselected package vim.
(Reading database ... 83502 files and directories currently installed.)
Unpacking vim (from .../vim_7.0-122+1etch3_i386.deb) ...
                                                    (解压 vim)
Setting up vim(7.0-122+1etch3) ...                (设置 vim)

```

在 Ubuntu 下, `apt-get` 工具对系统的软件包进行管理。`install` 选项安装会自动查找和安装指定的软件包, 其命令格式为:

```
apt-get install 软件包的名字
```

2. Vim 编辑器的模式

Vim 主要分为普通模式和插入模式。普通模式是命令模式, 插入模式是编辑模式。

在插入模式下可以进行字符的输入, 输入的键值显示在编辑框中, 这些文本可以用于编辑。普通界面是进行命令操作的, 输入的值代表一个命令。例如在普通模式下按 `h` 键, 光标会向左移动一个字符的位置。

插入模式和普通模式的切换分别为按 `i` 键和 `Esc` 键。在普通模式下按 `i` 键, 会转入插入模式; 在插入模式下按 `Esc` 键进入普通模式。在用户进入 Vim 还没有进行其他操作时, 操作模式是普通模式。

2.1.2 使用 Vim 建立文件

Vim 的命令行为格式为“`vim 文件名`”, 文件名是所要编辑的文件名。例如要编辑一个“`hello.c`”的 C 文件, 按照如下所述的步骤进行操作。

1. 建立文件

使用 Vim 建立一个新文件的命令行为“`vim 文件名`”。使用如下命令建立一个 `hello.c` 的 C 语言源文件, 并同时打开文件。

```
$vim hello.c
```

2. 进入插入模式

打开文件后, 默认情况下进入普通模式。按 `i` 键, 进入插入模式, vim 会在窗口的底部显示“`--INSERT--`”(中文模式显示下显示的是“`--插入--`”), 这表示当前模式为插入模式。

在输入文本的时候, 最下边有一个指示框, 告诉用户正在编辑文件的一些信息, 例如:

```
-- INSERT --                                6,11          All
```

表示当前模式为插入模式, 光标在第 6 行第 11 个字符位置上。

新接触 Vim 的读者常常会不知道自己在什么模式下, 或者不小心输入了错误的指令和

错误的操作。如果遇到这种情况，无论在什么模式下，要回到普通模式只需按 Esc 键就可以了。有时会按两次 Esc 键，当 Vim 发出“嘀”的一声，就表示 Vim 已经在普通模式了。

3. 文本输入

在编辑区输入如下文本：

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("Hello World!\n");
6     return 0;
7 }
```

输入第一行后，按 Enter 键开始一个新行。

4. 退出 Vim

当编译完成后，按 Esc 键退出插入模式回到普通模式，输入“: wq”退出 Vim 编辑器。运行命令 ls：

```
$ls -l                                (查看当前目录下的文件)
hello.c
```

会发现当前目录下已经存在一个名为 hello.c 的文件。输入的 wq 是“保存后退出”的意思：q 表示退出，w 表示保存。当不想保存所作的修改时，输入“:”键后，再输入“q!”，Vim 会直接退出，不保存所作修改。q! 是强制退出的意思。

Vim 有一个学习的帮助工具——vimtutor，它是很有帮助的一个工具，初学者可以使用它进行 Vim 的练习。输入“vimtutor”后，可以按照它的指示由浅入深地进行学习。

2.1.3 使用 Vim 编辑文本

Vim 的编辑命令有很多，本节将选取经常使用的几个命令进行介绍。介绍如何在 Vim 下移动光标，进行删除字符、复制、查找、转跳等操作。

1. 移动光标 h、j、k、l

Vim 在普通模式下移动光标需要按特定的键，进行左、下、上、右光标移动操作的字符分别为 h、j、k、l 这 4 个字符，其含义如下：

```
h      左 ←
j      下 ↓
k      上 ↑
l      右 →
```

按 h 键，光标左移一个字符的位置；按 l 键，光标右移一个字符的位置；按 k 键，光标上移一行；按 j 键，光标下移一行。

当然还可以用方向键移动光标，但必须将手从字母键位置上移动到方向键上，这样会减慢输入速度。而且，有一些键盘是没有方向键的，或者需要特殊的操作才能使用方向键（例如必须按组合键）。所以，知道用 h、j、k、l 字符移动光标的用法是很有帮助的。

2. 删除字符 x、dd、u、Ctrl+R

要删除一个字符可以使用 **x**，在普通模式下，将光标移到需要删除的字符上面然后按 **x** 键。例如 `hello.c` 的第一行输入有一个错误：

```
#Include <stdio.h>
```

将光标移动到 **I** 上，然后按 **x** 键，切换输入模式到插入模式，输入 **i**，对 **include** 的修正完成了。

要删除一整行可以使用 **dd** 命令，删除一行后，它后面的一行内容会自动向上移动一行。使用这个命令的时候要注意输入 **d** 的个数，两个 **d** 才是一个命令，在实际使用过程中经常将 **d** 的输入个数弄混淆。

恢复删除可以使用 **u**。当删除了不应该删除的东西后，**u** 命令可以取消之前的删除。例如，用 **dd** 命令删除一行，再按 **u** 键，恢复被删除的该行字符。

Ctrl+r 是一个特殊的命令，它为取消一个命令，可以使用它对 **u** 命令造成的后果进行弥补。例如使用 **u** 命令撤销了之前的输入，重新输入字符是很麻烦的，而使用 **Ctrl+r** 可以十分方便地将之前使用 **u** 命令撤销输入的字符重新找回。

3. 复制粘贴 p、y


Vim 下的粘贴命令是字符 **p**，它的作用是将内存中的字符复制到当前光标的后面。使用 **p** 时的前提是内存中有合适的字符串复制，例如要将一行复制到某个地方，可以用 **dd** 命令删除它，然后使用 **u** 命令恢复，这时候内存中是 **dd** 命令删除的字符串。将光标移动到需要插入的行之前，使用 **p** 命令可以把内存中的字符串复制后放置在选定的位置。

y 命令（即 **yank**）是复制命令，将指定的字符串复制到内存中，**yw** 命令（即 **yank words**）用于复制单词，可以指定复制的单词数量，**y2w** 复制两个单词。例如下一行代码：

```
1 #include <stdio.h>
```

光标位于此行的头部，当输入 **y2w** 时字符串 `#include` 就复制到内存中，按 **p** 键后，此行如下：

```
1 ##include include <stdio.h>
```

 **注意：**命令 **y** 在进行字符串复制的时候包含末尾的空格。按行进行字符串的复制时，使用 **dd** 命令复制的方式比较麻烦，可以使用 **yy** 命令进行复制。

4. 查找字符串 “/”

查找字符串的命令是 `/xxx`，其中的 **xxx** 代表要查找的字符串。例如查找当前文件的 `printf` 字符串，可以输入以下命令进行查找：

```
"/:printf"
```

按 **Enter** 键后，如果找到匹配的字符串，光标就停在第一个合适的字符串光标上。查找其他的匹配字符串可以输入字符 **n** 向下移到一个匹配的字符串上，输入字符 **N** 则会向上移到一个匹配的字符串上。

5. 跳到某一行 g

在编写程序或者修改程序的过程中，经常需要转跳到某一行（这在编译程序出错，进行修改程序的时候是经常遇到的，因为 GCC 编译器的报错信息会提示某行出错）。命令“:n”可以让光标转到某一行，其中 n 代表要转跳到的行数。例如要跳到第 5 行，可以输入“:5”，然后按 Enter 键，光标会跳到第 5 行的头部。还有一种实现方式，即 nG，n 为要转跳的行数，5G 是转跳到第 5 行的命令，注意其中的 G 为大写。

2.1.4 Vim 的格式设置

Vim 下可以进行很多方式的格式设置，这里仅对经常使用的格式进行介绍，例如设置缩进，设置 Tab 键对应空格的长度，设置行号等。

1. 设置缩进

合理的缩进会使程序更加清晰，Vim 提供了多种方法来简化这项工作。要对 C 语言程序缩进，需要设定 cindent 选项；如果需要设置下一行的缩进长度可以设置 shiftwidth 选项。例如如下命令实现 4 个空格的缩进。

```
:set cindent shiftwidth=4
```

设定了这一选项之后，当输入一行语句后，Vim 会自动在下一行进行缩进。例如在 if(x) 一行代码后面的内容，行的开头会自动向下一级缩进。例如：

```

                                if (a==b)
自动缩进      --->                do equal();
自动取消缩进  <--      if (a>b) {
自动缩进      --->                do lt();
自动取消缩进  <--      }
```

自动缩进还能提前发现代码中的错误。例如当输入了一个“}”后，如果发现比预想中的缩进多，那可能缺少了一个“}”。用 % 命令可以查找与“}”相匹配的“{”。

2. 设置 Tab 键的空格数量

进行文本编辑的时候 Tab 键可以移动一块较大的距离，不同的文本编辑器对 Tab 键移动距离的解释是不同的。Vim 编辑器 Tab 键的默认移动距离为 8 个空格，当需要对这个值进行更改的时候，需要设置 tabstop 选项的值。命令“:set tabstop=n”可以设置 Tab 键对应空格的量，例如“:set tabstop=2”将 Tab 键的宽度设置为 2 个空格。

3. 设置行号

在程序中设置行号使程序更加一目了然，设置行号的命令是“:set number”，按下 Enter 键后程序每行代码的头部会有一个行号的数值。

2.1.5 Vim 配置文件.vimrc

Vim 启动的时候会根据 ~/.vimrc 文件配置 vi 的设置，可以修改文件.vimrc 来定制 Vim。例如可以使用 shiftwidth 设置缩进宽度、使用 tabstop 设置 Tab 键的宽度、使用 number 设置

行号等格式来定义 Vim 的使用环境。例如按照如下的情况对.vimrc 文件进行修改:

```
set shiftwidth=2           #设置缩进宽度为 2 个空格
set tabstop=2              #设置 Tab 键宽度为 2 个空格
set number                 #显示行号
```

再次启动 Vim, 对缩进宽度、Tab 键的宽度都进行了设定, 并且会自动显示行号。

2.1.6 使用其他编辑器

在 Linux 下还有一些其他的编辑器, 例如 Gvim (Gvim 是 Vim 的 GNOME 版本)、codeblocks (严格来说是一个 IDE 开发环境)。

在 Linux 下进行开发并不排斥使用 Windows 环境下的编辑器, 例如写字板、UltraEdit、VC 的 IDE 开发环境等, 在保存的时候要注意保存为 UNIX 格式, 这主要是换行符造成的。在 Windows 下的换行为“回车+换行”, 而 UNIX 环境下的换行为单个的换行, 在 Linux 下用 Vim 查看会发现每行的末尾有一个很奇怪的“~”。如果没有保存为 UNIX 格式, 在 Linux 下可以用 dos2UNIX 转换。例如, 文件 hello.c 使用 Windows 编辑器, 默认保存, 将其转换为 UNIX 格式:

```
$dos2UNIX hello.c
```

再次查看文件 hello.c, “~”符号已经消失了。

2.2 Linux 下的 GCC 编译器工具集

在 2.1 节中, 介绍了如何使用 Linux 环境下的编辑器编写程序, 并编写了一个 hello.c 的程序。要使编写的程序能够运行, 需要进行程序的编译。本节将介绍 Linux 环境下采用的编译器 GCC 的选项和使用方式。

2.2.1 GCC 简介

GCC 是 Linux 下的编译工具集, 是 GNU Compiler Collection 的缩写, 包含 gcc、g++ 等编译器。这个工具集不仅包含编译器, 还包含其他工具集, 例如 ar、nm 等。

GCC 工具集不仅能编译 C/C++ 语言, 其他例如 Objective-C、Pascal、Fortran、Java、Ada 等语言均能进行编译。GCC 在可以根据不同的硬件平台进行编译, 即能进行交叉编译, 在 A 平台上编译 B 平台的程序, 支持常见的 X86、ARM、PowerPC、mips 等, 以及 Linux、Windows 等软件平台。在本书中仅介绍对 C 语言进行编译, 其他语言的编译请读者查阅相关资料。

GCC 在各种平台下都被广泛地采用, 特别是嵌入式平台下, 这得益于它的目标机定义规则。当对目标机的硬件进行了合适的定义后, 可以生成目标机能够正确解析的文件格式。另外, GCC 的强大前端是定义新语言的好方法, 例如可以重新定义语法解析规则, 定义自己使用的专有编程语言。查看 www.gnu.org 上关于 GCC 介绍的 gcc internel 可以得到更多更详细的信息。

GCC 的 C 编译器是 gcc, 其命令格式为:

Usage: gcc [options] file...

GCC 支持默认扩展名策略，表 2.1 是 GCC 下默认文件扩展名的含义。

表 2.1 文件扩展名含义

文件扩展名	GCC 所理解的含义
*.c	该类文件为 C 语言的源文件
*.h	该类文件为 C 语言的头文件
*.i	该类文件为预处理后的 C 文件
*.C	该类文件为 C++语言的源文件
*.cc	该类文件为 C++语言的源文件
*.cxx	该类文件为 C++语言的源文件
*.m	该类文件为 Objective-C 语言的源文件
*.s	该类文件为汇编语言的源文件
*.o	该类文件为汇编后的目标文件
*.a	该类文件为静态库
*.so	该类文件为共享库
a.out	该类文件为链接后的输出文件

GCC 下有很多编译器，可以支持 C 语言、C++语言等多种语言，表 2.2 是常用的几个编译器。

表 2.2 GCC 编译器含义

GCC 编译器命令	含 义	GCC 编译器命令	含 义
cc	指的是 C 语言编译器	gcc	指的是 C 语言编译器
cpp	指的是预处理编译器	g++	指的是 C++语言编译器

进行程序编译的时候，头文件路径和库文件路径是编译器默认查找的地方，参见表 2.3。

表 2.3 默认路径

类 型	存 放 路 径
头文件	按照先后顺序查找如下目录： /usr/local/include /usr/lib/gcc/i686-Linux-gnu/4.6.3/include /usr/include
库文件	按照先后顺序查找如下路径： /usr/lib/gcc/i686-Linux-gnu/4.6.3/ /usr/lib/gcc/i686-Linux-gnu/4.6.3/ /usr/lib/gcc/i686-Linux-gnu/4.6.3/../../../../i686-Linux-gnu/lib/i686-Linux-gnu/4.6.3/ /usr/lib/gcc/i686-Linux-gnu/4.6.3/../../../../i686-Linux-gnu/lib/ /usr/lib/gcc/i686-Linux-gnu/4.6.3/../../../../i686-Linux-gnu/4.6.3/ /usr/lib/gcc/i686-Linux-gnu/4.6.3/../../../../ /lib/i686-Linux-gnu/4.6.3/ /lib/ /usr/lib/i686-Linux-gnu/4.6.3/ /usr/lib/

2.2.2 编译程序的基本知识

GCC 编译器对程序的编译如图 2.1 所示，分为 4 个阶段：预编译、编译和优化、汇编和链接。GCC 的编译器可以将这 4 个步骤合并成一个。

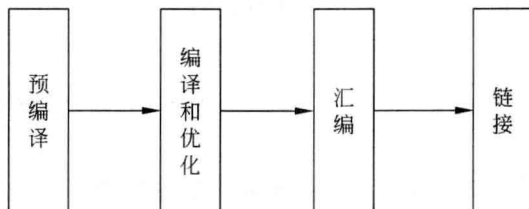


图 2.1 GCC 对程序的编译过程

源文件、目标文件和可执行文件是编译过程中经常用到的名词。源文件通常指存放可编辑代码的文件，如存放 C、C++ 和汇编语言的文件。目标文件是指经过编译器的编译生成的 CPU 可识别的二进制代码，但是目标文件一般不能执行，因为其中的一些函数过程没有相关的指示和说明。可执行文件就是目标文件与相关的库链接后的文件，它是可以执行的。

预编译过程将程序中引用的头文件包含进源代码中，并对一些宏进行替换。

编译过程将用户可识别的语言翻译成一组处理器可识别的操作码，生成目标文件，通常翻译成汇编语言，而汇编语言通常和机器操作码之间是一种一对一的关系。GNU 中有 C/C++ 编译器 GCC 和汇编器 as。

所有的目标文件必须用某种方式组合起来才能运行，这就是链接的作用。目标文件中通常仅解析了文件内部的变量和函数，对于引用的函数和变量还没有解析，这需要将其他已经编写好的目标文件引用进来，将没有解析的变量和函数进行解析，通常引用的目标是库。链接完成后会生成可执行文件。

2.2.3 单个文件编译成执行文件

在 Linux 下使用 GCC 编译器编译单个文件十分简单，直接使用 gcc 命令后面加上要编译的 C 语言的源文件，GCC 会自动生成文件名为 a.out 的可执行文件。自动编译的过程包括头文件扩展、目标文件编译，以及链接默认的系统库生成可执行文件，最后生成系统默认的可执行程序 a.out。

下面是一个程序的源代码，代码的作用是在控制台输出“Hello World!”字符串。

```
/*hello.c*/
#include <stdio.h>                                /*头文件包含*/
int main(void)
{
    printf("Hello World!\n");                      /*打印"Hello World!"*/
    return 0;
}
```

将代码存入 hello.c 文件中，运行如下命令将代码直接编译成可以执行文件：

```
$gcc hello.c
```

在 2.2.1 节中列出了 GCC 编译器可以识别的默认文件扩展名，通过检查 hello.c 文件的

扩展名，GCC 知道这是一个 C 文件。

使用上面的编译命令进行编译的时候，GCC 先进行扩展名判断，选择编译器。由于 `hello.c` 的扩展名为 `.c`，GCC 认为这是一个 C 文件，会选择 `gcc` 编译器来编译 `hello.c` 文件。

GCC 将采取默认步骤，先将 C 文件编译成目标文件，然后将目标文件链接成可执行文件，最后删除目标文件。上述命令没有指定生成执行文件的名称，GCC 将生成默认的文件名 `a.out`。运行结果如下：

```
$/a.out (执行 a.out 可执行文件)
Hello World!
```

如果希望生成指定的可执行文件名，选项 `-o` 可以使编译程序生成指定文件名，例如将上述程序编译输出一个名称为 `test` 的执行程序：

```
$gcc -o test hello.c
```

上述命令把 `hello.c` 源文件编译成可执行文件 `test`。运行可执行文件 `test`，向终端输出“Hello World!”字符串。运行结果如下：

```
$/test
Hello World!
```

2.2.4 编译生成目标文件

目标文件是指经过编译器的编译生成的 CPU 可识别的二进制代码，因为其中一些函数过程没有相关的指示和说明，目标文件不能执行。

在 2.2.3 节中介绍了直接生成可执行文件的编译方法，在这种编译方法中，中间文件作为临时文件存在，在可执行文件生成后，会删除中间文件。在很多情况下需要生成中间的目标文件，用于不同的编译目标。

GCC 的 `-c` 选项用于生成目标文件，这一选项将源文件生成目标文件，而不是生成可执行文件。默认情况下生成的目标文件的文件名和源文件的名称一样，只是扩展名为 `.o`。例如，下面的命令会生成一个名字为 `hello.o` 的目标文件：

```
$gcc -c hello.c
```

如果需要生成指定的文件名，可以使用 `-o` 选项。下面的命令将源文件 `hello.c` 编译成目标文件，文件名为 `test.o`：

```
$gcc -c -o test.o hello.c
```

可以用一条命令编译多个源文件，生成目标文件，这通常用于编写库文件或者一个项目中包含多个源文件。例如一个项目包含 `file1.c`、`file2.c` 和 `file3.c`，下面的命令可以将源文件生成 3 个目标文件：`file1.o`、`file2.o` 和 `file3.o`：

```
$gcc -c file1.c file2.c file3.c
```

2.2.5 多文件编译

GCC 可以自动编译链接多个文件，不管是目标文件还是源文件，都可以使用同一个命

令编译到一个可执行文件中。例如一个项目包含两个文件，文件 `string.c` 中有一个函数 `StrLen` 用于计算字符串的长度，而在 `main.c` 中调用这个函数将计算的结果显示出来。

1. 源文件 `string.c`

文件 `string.c` 的内容如下。文件中主要包含了用于计算字符串长度的函数 `StrLen()`。`StrLen()`函数的作用是计算字符串的长度，输入参数为字符串的指针，输出数值为计算字符串长度的计算结果。`StrLen()`函数将字符串中的字符与`'\0'`进行比较并进行字符长度计数，获得字符串的长度。

```
01 /*string.c*/
02 #define ENDSTRING '\0'           /*定义字符串*/
03 int StrLen(char *string)
04 {
05     int len = 0;
06
07     while(*string++ != ENDSTRING) /*当*string 的值为'\0'时,停止计算*/
08         len++;
09     return len;                  /*返回此值*/
10 }
```

2. 源文件 `main.c`

在文件 `main.c` 中是 `main()`函数的代码，如下代码所示。`main()`函数调用 `Strlen()`函数计算字符串 `Hello Dymatic` 的长度，并将字符串的长度打印出来。

```
01 /*main.c*/
02 #include <stdio.h>
03 extern int StrLen(char* str);    /*声明 Strlen 函数*/
04 int main(void)
05 {
06     char src[]="Hello Dymatic"; /*字符串*/
07     printf("string length is:%d\n",StrLen(src)); /*计算 src 的长度,将结果打印出来*/
08     return 0;
09 }
```

3. 编译运行

下面的命令将两个源文件中的程序编译成一个执行文件，文件名为 `test`。

```
$gcc -o test string.c main.c
```

执行编译出来的可执行文件 `test`，程序的运行结果如下：

```
$/test
String length is:13
```

当然可以先将源文件编成目标文件，然后进行链接。例如，下面的过程先将 `string.c` 和 `main.c` 源文件编译成目标文件 `string.o` 和 `main.o`，然后将 `string.o` 和 `main.o` 链接生成 `test`：

```
$gcc -c string.c main.c
$gcc -o test string.o main.o
```

2.2.6 预处理

在 C 语言程序中，通常需要包含头文件并会定义一些宏。预处理过程将源文件中的头文件包含进源文件中，并且将文件中定义的宏进行扩展。

编译程序时选项-E 告诉编译器进行预编译操作。例如如下命令将文件 `string.c` 的预处理结果显示在计算机屏幕上：

```
$gcc -E string.c
```

如果需要指定源文件预编译后生成的中间结果文件名，需要使用选项-o。例如，下面的代码将文件 `string.c` 进行预编译，生成文件 `string.i`。`string.i` 内容如下：

```
$gcc -o string.i -E string.c
# 1 "string.c"
# 1 "<built-in>"
# 1 "<命令行>"
# 1 "string.c"

int StrLen(char *string)
{
    int len = 0;

    while(*string++ != '\0')
        len++;
    return len;
}
```

可以发现之前定义的宏 `ENDSTRING`，已经被替换成了“`\0`”。

2.2.7 编译成汇编语言

编译过程将用户可识别的语言翻译成一组处理器可识别的操作码，通常翻译成汇编语言。汇编语言通常和机器操作码之间是一一对应的关系。

生成汇编语言的 GCC 选项是-S，默认情况下生成的文件名和源文件一致，扩展名为.s。例如，下面的命令将 C 语言源文件 `string.c` 编译成汇编语言，文件名为 `string.s`。

```
$gcc -S string.c
```

下面是编译后的汇编语言文件 `string.s` 的内容。其中，第 1 行内容是 C 语言的文件名，第 3 行和第 4 行是文件中的函数描述，标签 `StrLen` 之后的代码用于实现字符串长度的计算。

```
01      .file      "string.c"
02      .text
03      .globl    StrLen
04      .type     StrLen, @function
05 StrLen:
06      .LFB0:
07      .cfi_startproc
08      pushl     %ebp
09      .cfi_def_cfa_offset 8
10      .cfi_offset 5, -8
11      movl      %esp, %ebp
12      .cfi_def_cfa_register 5
```

```

13     subl    $16, %esp
14     movl    $0, -4(%ebp)
15     jmp     .L2
16 .L3:
17     addl     $1, -4(%ebp)
18 .L2:
19     movl     8(%ebp), %eax
20     movzbl   (%eax), %eax
21     testb    %al, %al
22     setne    %al
23     addl     $1, 8(%ebp)
24     testb    %al, %al
25     jne     .L3
26     movl     -4(%ebp), %eax
27     leave
28     .cfi_restore 5
29     .cfi_def_cfa 4, 4
30     ret
31     .cfi_endproc
32 .LFE0:
33     .size     StrLen, .-StrLen
34     .ident    "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
35     .section   .note.GNU-stack,"",@progbits

```

2.2.8 生成和使用静态链接库

静态库是 obj 文件的一个集合，通常静态库以“.a”为后缀。静态库由程序 ar 生成，现在静态库已经不像之前那么普遍了，这主要是由于程序都在使用动态库。

静态库的优点是可以在不用重新编译程序库代码的情况下，进行程序的重新链接，这种方法节省了编译过程的时间（在编译大型程序的时候，需要花费很长时间）。但是由于现在系统的强大，编译的时间已经不是问题。静态库的另一个优势是开发者可以提供库文件给使用的人员，不用开放源代码，这是库函数提供者经常采用的手段。当然这也是程序模块化开发的一种手段，使每个软件开发人员的精力集中在自己的部分。在理论上，静态库的执行速度比共享库和动态库要快（1%~5%）。

1. 生成静态链接库

生成静态库，或者将一个 obj 文件加到已经存在的静态库的命令为“ar 库文件 obj 文件 1 obj 文件 2”。创建静态库的最基本步骤是生成目标文件，这点前面已经介绍过。然后使用工具 ar 对目标文件进行归档。工具 ar 的 -r 选项，可以创建库，并把目标文件插入到指定库中。例如，将 string.o 打包为库文件 libstr.a 的命令为：

```
$ar -rcs libstr.a string.o
```

2. 使用静态链接库

在编译程序的时候经常需要使用函数库，例如经常使用的 C 标准库等。GCC 链接时使用库函数和一般的 obj 文件的形式是一致的，例如对 main.c 进行链接的时候，需要使用之前已经编译好的静态链接库 libstr.a，命令格式如下：


```
$gcc -o test main.c libstr.a
```


也可以使用命令“-l 库名”进行，库名是不包含函数库和扩展名的字符串。例如编译 main.c 链接静态库 libstr.a 的命令可以修改为：

```
$gcc -o test main.c -lstr
```

上面的命令将在系统默认的路径下查找 str 函数库，并把它链接到要生成的目标程序上。可能系统会提示无法找到库文件 str，这是由于 str 库函数没有在系统默认的查找路径下，需要显示指定库函数的路径，例如库文件和当前编译文件在同一目录下：

```
$gcc -o test main.c -L./ -lstr
```

 **注意：**在使用-l 选项时，-o 选项的目的名称要在-l 链接的库名称之前，否则 gcc 会认为-l 是生成的目标而出错。

2.2.9 生成动态链接库

动态链接库是程序运行时加载的库，当动态链接库正确安装后，所有的程序都可以使用动态库来运行程序。动态链接库是目标文件的集合，目标文件在动态链接库中的组织方式是按照特殊方式形成的。库中函数和变量的地址是相对地址，不是绝对地址，其真实地址在调用动态库的程序加载时形成。

动态链接库的名称有别名 (soname)、真名 (realname) 和链接名 (linker name)。别名由一个前缀 lib，然后是库的名字，再加上一个后缀“.so”构成。真名是动态链接库的真实名称，一般总是在别名的基础上加上一个小版本号、发布版本等构成。除此之外，还有一个链接名，即程序链接时使用的库的名字。在动态链接库安装的时候，总是复制库文件到某个目录下，然后用一个软链接生成别名，在库文件进行更新的时候，仅仅更新软链接即可。

1. 生成动态链接库

生成动态链接库的命令很简单，使用-fPIC 选项或者-fpic 选项。-fPIC 和-fpic 选项的作用是使得 gcc 生成的代码是位置无关的，例如下面的命令将 string.c 编译生成动态链接库：

```
$gcc -shared -Wl,-soname,libstr.so -o libstr.so.1 string.c
```

其中，选项“-soname,libstr.so”表示生成动态库时的别名是 libstr.so；“-o libstr.so.1”选项则表示是生成名字为 libstr.so.1 的实际动态链接库文件；-shared 告诉编译器生成一个动态链接库。

生成动态链接库之后一个很重要的问题就是安装，一般情况下将生成的动态链接库复制到系统默认的动态链接库的搜索路径下，通常有/lib、/usr/lib、/usr/local/lib，放到以上任何一个目录下都可以。

2. 动态链接库的配置

动态链接库不能随意使用，要在运行的程序中使用动态链接库，需要指定系统的动态链接库搜索的路径，让系统找到运行所需的动态链接库才可以。系统中的配置文件/etc/ld.so.conf 是动态链接库的搜索路径配置文件。在这个文件内，存放着可被 Linux 共享

的动态链接库所在目录的名字（系统目录/lib、/usr/lib 除外），多个目录名间以空白字符（空格、换行等）或冒号或逗号分隔。查看系统中的动态链接库配置文件的内容：

```
$ cat /etc/ld.so.conf
include /etc/ld.so.conf.d/*.conf
```

Ubuntu 的配置文件将目录/etc/ld.so.conf.d 中的配置文件包含进来，对这个目录下的文件进行查看：

```
$ ls /etc/ld.so.conf.d/
i386-linux-gnu_GL.conf  libc.conf
i686-linux-gnu.conf      vmware-tools-libraries.conf
$ cat /etc/ld.so.conf.d/i686-linux-gnu.conf
#查看配置文件 i486-linux-gnu.conf

# Multiarch support
/lib/i386-linux-gnu
/usr/lib/i386-linux-gnu
/lib/i686-linux-gnu
/usr/lib/i686-linux-gnu
```

从上面的配置文件可以看出，在系统的动态链接库配置中，包含了该动态库/lib/i386-linux-gnu、/usr/lib/i386-linux-gnu 和/lib/i686-linux-gnu、/usr/lib/i686-linux-gnu 四个目录。

3. 动态链接库管理命令

为了让新增加的动态链接库能够被系统共享，需要运行动态链接库的管理命令 ldconfig。ldconfig 命令的作用是在系统的默认搜索路径，和动态链接库配置文件中所列出的目录里搜索动态链接库，创建动态链接装入程序需要的链接和缓存文件。搜索完毕后，将结果写入缓存文件/etc/ld.so.cache 中，文件中保存的是已经排好序的动态链接库名字列表。ldconfig 命令行的用法如下，其中选项的含义参见表 2.4。

```
ldconfig [-v|--verbose] [-n] [-N] [-X] [-f CONF] [-C CACHE] [-r ROOT] [-l]
[-p|--print-cache] [-c FORMAT] [--format=FORMAT] [-V] [-?|--help|--usage]
path...
```

表 2.4 ldconfig 的选项含义

选 项	含 义
-v	此选项打印 ldconfig 的当前版本号，显示所扫描的每一个目录和动态链接库
-n	此选项处理命令行指定的目录，不对系统的默认目录/lib、/usr/lib 进行扫描，也不对配置文件/etc/ld.so.conf 中所指定的目录进行扫描
-N	此选项 ldconfig 不会重建缓存文件
-X	此选项 ldconfig 不更新链接
-f CONF	此选项使用用户指定的配置文件代替默认文件/etc/ld.so.conf
-C CACHE	此选项使用用户指定的缓存文件代替系统默认的缓存文件/etc/ld.so.cache
-r ROOT	此选项改变当前应用程序的根目录
-l	此选项用于手动链接单个动态链接库
-p 或--print-cache	此选项用于打印出缓存文件中共享库的名字

如果想知道系统中有哪些动态链接库，可以使用 `ldconfig` 的 `-p` 选项来列出缓存文件中的动态链接库列表。下面的命令中表明在系统缓存中共有 682 个动态链接库。

```
$ ldconfig -p                                (列出当前系统中的动态链接库)
在缓冲区"/etc/ld.so.cache"中找到 809 个库 (缓存中的动态链接库的数目)
libzephyr.so.4 (libc6) => /usr/lib/libzephyr.so.4
libzeitgeist-1.0.so.1 (libc6) => /usr/lib/libzeitgeist-1.0.so.1
libz.so.1 (libc6) => /lib/i386-linux-gnu/libz.so.1
libyelp.so.0 (libc6) => /usr/lib/libyelp.so.0
libyajl.so.1 (libc6) => /usr/lib/i386-linux-gnu/libyajl.so.1
...
```

使用 `ldconfig` 命令，默认情况下并不将扫描的结果输出。使用 `-v` 选项会将 `ldconfig` 在运行过程中扫描到的目录和共享库信息输出到终端，用户可以看到运行的结果和中间的信息。在执行 `ldconfig` 后，将刷新缓存文件 `/etc/ld.so.cache`。


```
$ ldconfig -v
/usr/lib:                                     (扫描/usr/lib目录中的动态链接库)
libdb-4.3.so -> libdb-4.3.so
libXcursor.so.1 -> libXcursor.so.1.0.2
...
/usr/lib/i686: (hwcap: 0x20000000000000)      (扫描/usr/lib/i486目录中的动态链接库)
libssl.so.0.9.8 -> libssl.so.0.9.8
libcrypto.so.0.9.8 -> libcrypto.so.0.9.8
...
```

当用户的目录并不在系统动态链接库配置文件 `/etc/ld.so.conf` 中指定的时候，可以使用 `ldconfig` 命令显示指定要扫描的目录，将用户指定目录中的动态链接库放入系统中进行共享。命令格式的形式为：

`ldconfig 目录名`

这个命令将 `ldconfig` 指定的目录名中的动态链接库放入系统的缓存 `/etc/ld.so.cache` 中，从而可以被系统共享使用。下面的代码将扫描当前用户的 `lib` 目录，将其中的动态链接库加入系统：

```
$ ldconfig ~/lib
```

 **注意：**如果在运行上述命令后，再次运行 `ldconfig` 而没有加参数，系统会将 `/lib`、`/usr/lib` 及 `/etc/ld.so.conf` 中指定目录中的动态库加入缓存，这时候上述代码中的动态链接库可能不被系统共享了。

4. 使用动态链接库

在编译程序时，使用动态链接库和静态链接库是一致的，使用“-l 库名”的方式，在生成可执行文件的时候会链接库文件。例如下面的命令将源文件 `main.c` 编译成可执行文件 `test`，并链接库文件 `libstr.a` 或者 `libstr.so`：

```
$gcc -o test main.c -L./ -lstr
```

`-L` 指定链接动态链接库的路径，`-lstr` 链接库函数 `str`。但是运行 `test` 一般会出现如下

问题:

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

这是由于程序运行时没有找到动态链接库造成的。程序编译时链接动态链接库和运行时使用动态链接库的概念是不同的,在运行时,程序链接的动态链接库需要在系统目录下才行。有以下几种办法可以解决此问题。

- ❑ 将动态链接库的目录放到程序搜索路径中,可以将库的路径加到环境变量 `LD_LIBRARY_PATH` 中实现,例如:

```
$export LD_LIBRARY_PATH=/example/ex02: $LD_LIBRARY_PATH
```

将存放库文件 `libstr.so` 的路径 `/example/ex02` 加入到搜索路径中,再运行程序就没有之前的警告了。

- ❑ 另一种方法是使用 `ld-Linux.so.2` 来加载程序,命令格式为:

```
/lib/ld-Linux.so.2 --library-path 路径 程序名
```

加载 `test` 程序的命令为:

```
/lib/ld-Linux.so.2 --library-path /example/ex02 test
```

⚠注意:如果系统的搜索路径下同时存在静态链接库和动态链接库,默认情况下会链接动态链接库。如果需要强制链接静态链接库,需要加上“-static”选项,即上述的编译方法改为如下的方式:

```
$gcc -o test main.c -static -lstr
```

2.2.10 动态加载库

动态加载库和一般的动态链接库所不同的是,一般动态链接库在程序启动的时候就要寻找动态库,找到库函数;而动态加载库可以用程序的方法来控制什么时候加载。动态加载库主要有函数 `dlopen()`、`dlerror()`、`dlsym()`和 `dlclose()`。

1. 打开动态库 `dlopen()`函数

函数 `dlopen()`按照用户指定的方式打开动态链接库,其中参数 `filename` 为动态链接库的文件名, `flag` 为打开方式,一般为 `RTLD_LAZY`,函数的返回值为库的指针。其函数原型如下:

```
void * dlopen(const char *filename, int flag);
```

例如,下面的代码使用 `dlopen` 打开当前目录下的动态库 `libstr.so`。

```
void *phandle = dlopen("./libstr.so", RTLD_LAZY);
```

2. 获得函数指针 `dlsym()`

使用动态链接库的目的是调用其中的函数,完成特定的功能。函数 `dlsym()`可以获得动

关于此电子书的说明

本人由于一些便利条件，可以为您提供各种中文图书的PDF电子版，保证质量清晰。只要图书不是太新，文学、法律、计算机、经济、医学、工业、学术等方面的图书，都可以帮您制作，如果您有这方面的需求，可以通过QQ联系我，我的QQ号是 [3330972307](#)。