

# Linux

## 内核源码剖析

### —— TCP/IP 实现

上册

樊东东 莫澜 编著



 机械工业出版社  
CHINA MACHINE PRESS

# Linux 内核源码剖析

## ——TCP/IP 实现

上册

樊东东 莫 澜 编著



机械工业出版社

本书详细论述了 Linux 内核 2.6.20 版本中 TCP/IP 的实现。书中给出了大量的源代码，通过对源代码的详细注释，帮助读者掌握 TCP/IP 的实现。本书根据协议栈层次，从驱动层逐步论述到传输层，包括驱动的实现、接口层的输入输出、IP 层的输入输出以及 IP 选项的处理、邻居子系统、路由、套接口及传输层等内容，全书基本涵盖了网络体系架构全部的知识点。特别是 TCP，包括 TCP 连接的建立和终止、输入与输出，以及拥塞控制的实现。

本书适用于熟悉 Linux 的基本使用方法，对 Linux 内核工作原理以及网络知识有一定的了解，而又极想更深入理解各个机制在 Linux 中的具体实现的用户，包括应用程序员和嵌入式程序员，以及网络管理员等。相关专业的科研人员在工作中遇到问题时，也可以查阅本书，理解相关内核部分的实现。此外，计算机相关专业的本科高年级学生和研究生，在学习相关课程（如操作系统、计算机网络等）时，可将本书作为辅助教程，与理论相结合以便更好地理解相应的知识点。

## 图书在版编目（CIP）数据

Linux 内核源码剖析：TCP/IP 实现 / 樊东东，莫澜编著. —北京：机械工业出版社，2010.12

ISBN 978-7-111-32373-0

I. ①L… II. ①樊…②莫… III. ①Linux 操作系统—机器代码程序—程序分析②计算机网络—通信协议 IV. ①TP316.89②TN915.04

中国版本图书馆 CIP 数据核字（2010）第 212455 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：车 忱

责任编辑：车 忱

责任印制：乔 宇

三河市宏达印刷有限公司印刷

2011 年 1 月第 1 版·第 1 次印刷

184mm×260mm·67.75 印张·1677 千字

0001—3000 册

标准书号：ISBN 978-7-111-32373-0

定价：142.00 元（上、下册）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

网络服务

社服务中心：（010）88361066

门户网：<http://www.cmpbook.com>

销售一部：（010）68326294

教材网：<http://www.cmpedu.com>

销售二部：（010）88379649

读者服务部：（010）68993821

封面无防伪标均为盗版

# 前 言

有人宣称 Linux 人才是未来 20 年 IT 职场中的王者。无论这种说法有多夸张，有一个事实是不可否认的，那就是，近年来 Linux 的市场份额不断增长，Linux 正在受到越来越多的关注乃至推崇。由于 Linux 可广泛地应用到各种系统，包括很多嵌入式系统上，以及其他的诸多优点，如开放、高效、丰富的网络功能等，这种趋势在可预计的未来还将持续。

目前，国内对 Linux 各方面的研究工作没有国外那样广泛和深入，相关的出版物水准参差不齐。特别是在网络的实现方面，有些著作针对性不强，有些则缺少了重要的传输层协议实现的论述，还有一些虽然有比较全面的介绍，却不够深入，选用的 Linux 版本也比较旧。

针对以上情况，本书选择了较新的 2.6.20 版本内核 TCP/IP 实现作详细的论述，在重要的细节处甚至逐行分析，并在此基础上，对代码背后的机制和原理作了深入的阐述，将各关键点连成一个整体，帮助读者理清整个 Linux 网络部分的脉络。作者本着严谨的态度，在写作过程中参阅了大量的中英文资料及相关的文档。相信本书可以成为那些希望深入了解 Linux 的 TCP/IP 协议栈、网络部分实现的人们的有力工具。

本书共有 33 章，通过自底向上的方法来论述 TCP/IP 的实现，从数据链路层开始，然后是网络层（IP、ICMP、IGMP、路由以及邻居子系统和 IP 组播），接下来是套接口，最后是传输层（TCP 和 UDP）。在学习的时候也可以采用自顶向下的方法（从传输层开始向下），或者结合以上两种方法。要理解一个系统的运行机制，对于一个专业人员来说，代码是最为直接也最为可靠的资源。Linux 普及面不断扩展，越来越多的人会想通过研读内核代码来了解 Linux 系统，一来是更好地解决具体工作中的相关问题，二是从 Linux 这个高质量的操作系统中学习到更多的编程、架构等技术。本书的特点如下：

- 选择的内核版本新，内容不会在短期内过时。
- 对代码作了详细的论述，在此基础上进一步分析了代码背后的机制和原理，为读者理清了整个框架的脉络，帮助读者避免迷失在细节中。
- 写作过程中参阅了大量中英文资料和相关的文档，对内核代码更是作了长时间深入细致的研究和分析，在细节处反复推敲，以确保本书的质量。
- 书中有大量的图表，用来帮助读者直观地理解各个数据结构之间的联系及各个函数的调用关系等。

Linux 内核中的网络模块虽然是一个比较独立的模块，但仍会涉及内核中一些常用的技术和算法。因此读者在阅读本书之前需要掌握一些基础知识，对 TCP/IP 应有一定的了解。最好能够深入地阅读过 TCP/IP 相关的 RFC，了解 Linux 内核的运行机制，包括进程管理、内存管理、文件系统、系统调用、netlink、内核中多种锁（自旋锁、读写锁、RCU）等。当然在学习过程中，免不了要做些实验，因此也需要熟悉 Linux 提供的多种工具，包

括 ip、ifconfig 等命令。

本书使用的 Linux 内核版本为 2.6.20。也许读者想知道为什么选择这个版本。实际上在作者开始分析 TCP/IP 代码时，该版本是最新版本。只是因为 Linux 内核的更新很快，经过两年多的分析后，最新版本已经是 2.6.3x 了。即便如此，如果读者能够理解 2.6.20 版本的代码，则再读最新版本的代码时，应该不会太难。

由于本书内容繁多，漏洞和不足之处难免，请读者谅解并提出修改建议。作者的博客是 <http://blog.chinaunix.net/u3/112243/>。

在此，还要特别感谢网友衫秋南对本书提出宝贵的意见和建议。

# 上册目录

## 前言

<b>第 1 章 预备知识</b> .....	1	3.2.3 数据存储相关的变量	20
1.1 应用层配置诊断工具	2	3.2.4 通用的成员变量	21
1.1.1 iputils	2	3.2.5 标志性变量	24
1.1.2 net-tools	2	3.2.6 特性相关的成员变量	25
1.1.3 iproute2	2	<b>3.3 skb_shared_info 结构</b>	25
1.2 内核空间与用户空间的接口	2	3.3.1 “零拷贝”技术	25
1.2.1 procfs	2	3.3.2 对聚合分散 I/O 数据的支持	27
1.2.2 sysctl(/proc/sys 目录)	4	3.3.3 对 GSO 的支持	30
1.2.3 sysfs(/sys 文件系统)	5	3.3.4 访问 skb_shared_info 结构	31
1.2.4 ioctl 系统调用	6	<b>3.4 管理函数</b>	31
1.2.5 netlink 套接口	6	3.4.1 SKB 的缓存池	31
1.3 网络 I/O 加速	6	3.4.2 分配 SKB	32
1.3.1 TSO/GSO	7	3.4.3 释放 SKB	34
1.3.2 I/O AT	8	3.4.4 数据预留和对齐	36
1.4 其他	8	3.4.5 克隆和复制 SKB	38
1.4.1 slab 分配器	9	3.4.6 链表管理函数	42
1.4.2 RCU	9	3.4.7 添加或删除尾部数据	42
<b>第 2 章 网络体系结构概述</b> .....	10	3.4.8 拆分数据: skb_split()	44
2.1 引言	10	3.4.9 重新分配 SKB 的线性数据区:	
2.2 协议简介	10	pskb_expand_head()	46
2.3 网络架构	11	3.4.10 其他函数	46
2.4 系统调用接口	11	<b>第 4 章 网络模块初始化</b>	48
2.5 协议无关接口	12	4.1 引言	48
2.6 传输层协议	12	4.2 网络模块初始化顺序	48
2.7 套接口缓存	13	4.3 优化基于宏的标记	49
2.8 设备无关接口	14	4.4 网络设备处理层初始化	52
2.9 设备驱动程序	14	<b>第 5 章 网络设备</b>	55
2.10 网络模块源代码组织	14	5.1 PCI 设备	55
<b>第 3 章 套接口缓存</b> .....	15	5.1.1 PCI 驱动程序相关结构	55
3.1 引言	15	5.1.2 注册 PCI 驱动程序	57
3.2 sk_buff 结构	15	5.2 与网络设备有关的数据结构	59
3.2.1 网络参数和内核数据结构	16	5.2.1 net_device 结构	59
3.2.2 SKB 组织相关的变量	19	5.2.2 网络设备有关结构的组织	71

5.2.3 相关函数	72	6.3 IP 地址的设置	109
5.3 网络设备的注册	73	6.3.1 netlink 接口	109
5.3.1 设备注册的时机	73	6.3.2 inet_insert_ifa()	111
5.3.2 分配 net_device 结构空间	73	6.3.3 inet_del_ifa()	112
5.3.3 网络设备注册过程	75	6.4 ioctl	115
5.3.4 注册设备的状态迁移	79	6.5 inetaddr_chain 通知链	121
5.3.5 设备注册状态通知	79	<b>第 7 章 接口层的输入</b>	122
5.3.6 引用计数	80	7.1 系统参数	122
5.4 网络设备的注销	80	7.2 接口层的 ioctl	123
5.4.1 设备注销的时机	80	7.2.1 SIOCxIFxxx 类命令	123
5.4.2 网络设备注销过程	81	7.2.2 SIOCETHTOOL	126
5.5 网络设备的启用	86	7.2.3 私有命令	127
5.6 网络设备的禁用	88	7.3 初始化	127
5.7 与电源管理交互	89	7.4 softnet_data 结构	128
5.7.1 挂起设备	90	7.5 NAPI 方式	130
5.7.2 唤醒设备	90	7.5.1 网络设备中断例程	131
5.8 侦测连接状态改变	91	7.5.2 网络输入软中断	131
5.8.1 调度处理连接状态改变事件	91	7.5.3 轮询处理	133
5.8.2 linkwatch 标志	95	7.6 非 NAPI 方式	134
5.9 从用户空间配置设备相关 信息	95	7.7 接口层输入报文的处理	137
5.9.1 ethtool	95	7.7.1 报文接收例程	137
5.9.2 媒体独立接口	97	7.7.2 netif_receive_skb()	138
5.10 虚拟网络设备	97	7.7.3 dev_queue_xmit_nit()	141
<b>第 6 章 IP 编址</b>	99	7.8 响应 CPU 状态的变化	142
6.1 接口和 IP 地址	99	7.9 netpoll	143
6.1.1 主 IP 地址、从属 IP 地址和 IP 别名	99	7.9.1 netpoll 相关结构	143
6.1.2 IP 地址的组织	99	7.9.2 注册 netpoll 实例	145
6.1.3 in_device 结构	100	7.9.3 netpoll 的输入	148
6.1.4 in_ifaddr 结构	101	7.9.4 netpoll 的输出	156
6.2 函数	102	7.9.5 tx_work 工作队列	159
6.2.1 inetdev_init()	102	7.9.6 netpoll 实例: netconsole	160
6.2.2 inetdev_destroy()	104	<b>第 8 章 接口层的输出</b>	163
6.2.3 inet_select_addr()	104	8.1 输出接口	163
6.2.4 inet_confirm_addr()	106	8.1.1 dev_queue_xmit()	163
6.2.5 inet_addr_onlink()	107	8.1.2 dev_hard_start_xmit()	167
6.2.6 inetdev_by_index()	107	8.1.3 e100 的输出接口: e100_xmit_frame()	168
6.2.7 inet_ifa_byprefix()	108	8.2 网络输出软中断	168
6.2.8 inet_abc_len()	108	8.2.1 netif_schedule()	168
		8.2.2 net_tx_action()	169

8.3 网络设备不支持 GSO 时的处理 .....	170	11.7.3 读取错误信息 .....	233
8.3.1 dev_gso_cb 私有控制块 .....	171	11.8 报文控制信息 .....	235
8.3.2 dev_gso_segment() .....	171	11.8.1 IP 控制信息块 .....	235
8.3.3 skb_gso_segment() .....	172	11.8.2 报文控制信息的输出 .....	235
<b>第 9 章 流量控制</b> .....	174	11.8.3 报文控制信息的输入 .....	236
9.1 通过流量控制后输出 .....	174	11.9 对端信息块 .....	237
9.1.1 dev_queue_xmit() .....	175	11.9.1 系统参数 .....	239
9.1.2 qdisc_restart() .....	176	11.9.2 对端信息块的创建和查找 .....	239
9.2 构成流量控制的三种元素 .....	178	11.9.3 对端信息块的删除 .....	241
9.2.1 排队规则 .....	179	11.9.4 垃圾回收 .....	242
9.2.2 类 .....	186	11.10 IP 数据报的输入处理 .....	244
9.2.3 过滤器 .....	189	11.10.1 IP 数据报输入到本地 .....	247
9.3 默认的 FIFO 排队规则 .....	192	11.10.2 IP 数据报的转发 .....	249
9.3.1 pfifo_fast_init() .....	194	11.11 IP 数据报的输出处理 .....	253
9.3.2 pfifo_fast_reset() .....	194	11.11.1 IP 数据报输出到设备 .....	253
9.3.3 pfifo_fast_enqueue() .....	194	11.11.2 TCP 输出的接口 .....	255
9.3.4 pfifo_fast_dequeue() .....	195	11.11.3 UDP 输出的接口 .....	261
9.3.5 pfifo_fast_requeue() .....	195	11.12 IP 层对 GSO 的支持 .....	275
9.4 netlink 的 tc 接口 .....	195	11.12.1 inet_gso_segment() .....	275
9.5 排队规则的创建接口 .....	197	11.12.2 inet_gso_send_check() .....	277
9.5.1 类的创建接口 .....	201	<b>第 12 章 IP 选项处理</b> .....	278
9.5.2 过滤器的创建接口 .....	204	12.1 IP 选项 .....	278
<b>第 10 章 Internet 协议族</b> .....	209	12.1.1 选项列表的结束符 .....	279
10.1 net_proto_family 结构 .....	209	12.1.2 空操作 .....	279
10.2 inet_protosw 结构 .....	210	12.1.3 安全选项 .....	279
10.3 net_protocol 结构 .....	212	12.1.4 严格源路由选项 .....	280
10.4 Internet 协议族的初始化 .....	214	12.1.5 宽松源路由选项 .....	281
<b>第 11 章 IP: 网际协议</b> .....	217	12.1.6 记录路由选项 .....	282
11.1 引言 .....	217	12.1.7 流标识选项 .....	282
11.1.1 IP 首部 .....	218	12.1.8 时间戳选项 .....	283
11.1.2 IP 数据报的输入与输出 .....	219	12.1.9 路由器警告选项 .....	283
11.2 IP 的私有信息控制块 .....	220	12.2 ip_options 结构 .....	284
11.3 系统参数 .....	220	12.3 在 IP 数据报中构建 IP 选项 .....	285
11.4 初始化 .....	223	12.4 复制 IP 数据报中选项到指定的 ip_options 结构 .....	286
11.5 IP 层套接口选项 .....	223	12.5 处理待发送 IP 分片中的选项 .....	290
11.6 ipv4_devconf 结构 .....	227	12.6 解析 IP 选项 .....	291
11.7 套接口的错误队列 .....	229	12.7 还原在校验 IP 选项时修改的 IP 选项 .....	297
11.7.1 添加 ICMP 差错信息 .....	231		
11.7.2 添加由本地产生的差错信息 .....	232		



12.8 处理转发 IP 数据报中的 IP 选项 .....	298	15.2 虚拟接口 .....	354
12.9 处理 IP 数据报的源路由选项 .....	299	15.2.1 虚拟接口的添加 .....	355
12.10 解析并处理 IP 首部中的 IP 选项 .....	300	15.2.2 虚拟接口的删除: vif_delete() .....	358
12.11 路由警告选项的处理 .....	301	15.2.3 查找虚拟接口: ipmr_find_vif() ..	358
12.12 由控制信息生成 IP 选项信息块 .....	302	15.3 组播转发缓存 .....	358
<b>第 13 章 IP 的分片与组装</b> .....	303	15.3.1 组播转发缓存的创建 .....	361
13.1 系统参数 .....	303	15.3.2 组播转发缓存的删除 .....	361
13.2 分片 .....	303	15.3.3 组播转发缓存的查找 .....	361
13.2.1 快速分片 .....	306	15.3.4 向组播路由守护进程发送 报告 .....	362
13.2.2 慢速分片 .....	309	15.4 临时组播转发缓存 .....	364
13.3 组装 .....	312	15.4.1 临时组播转发缓存队列 .....	365
13.3.1 ipq 结构 .....	312	15.4.2 创建临时组播转发缓存 .....	365
13.3.2 ipq 散列表和链表的维护 .....	315	15.4.3 用于超时而删除临时组播 转发缓存的定时器 .....	367
13.3.3 ipq 散列表的重组 .....	316	15.4.4 释放临时组播缓存项中保存的 临时组播报文 .....	368
13.3.4 超时 IP 分片的清除 .....	317	15.5 外部事件 .....	369
13.3.5 垃圾收集 .....	318	15.6 组播套接口选项 .....	369
13.3.6 相关分片组装函数 .....	319	15.6.1 IP_MULTICAST_TTL .....	369
13.3.7 分片组装 .....	327	15.6.2 IP_MULTICAST_LOOP .....	370
<b>第 14 章 ICMP: Internet 控制 报文协议</b> .....	330	15.6.3 IP_MULTICAST_IF .....	370
14.1 ICMP 报文结构 .....	330	15.6.4 IP_ADD_MEMBERSHIP .....	372
14.2 注册 ICMP 报文类型 .....	330	15.6.5 IP_DROP_MEMBERSHIP .....	372
14.3 系统参数 .....	330	15.6.6 IP_MSFILTER .....	373
14.4 ICMP 的初始化 .....	332	15.6.7 IP_BLOCK_SOURCE 和 IP_UNBLOCK_SOURCE .....	375
14.5 输入处理 .....	333	15.6.8 IP_ADD_SOURCE_MEMBERSHIP 和 IP_DROP_SOURCE_ MEMBERSHIP .....	375
14.5.1 差错处理 .....	337	15.6.9 MCAST_JOIN_GROUP .....	376
14.5.2 重定向处理 .....	342	15.6.10 MCAST_LEAVE_GROUP .....	377
14.5.3 请求回显 .....	343	15.6.11 MCAST_BLOCK_SOURCE 和 MCAST_UNBLOCK_SOURCE ..	377
14.5.4 时间戳请求 .....	345	15.6.12 MCAST_JOIN_SOURCE_GROUP 和 MCAST_LEAVE_SOURCE_ GROUP .....	377
14.5.5 地址掩码请求和应答 .....	346	15.6.13 MCAST_MSFILTER .....	378
14.6 输出处理 .....	346	15.7 组播选路套接口选项 .....	378
14.6.1 发送 ICMP 报文 .....	346		
14.6.2 发送回显应答和时间戳 应答报文 .....	350		
<b>第 15 章 IP 组播</b> .....	353		
15.1 初始化 .....	353		

15.7.1	MRT_INIT	379	16.9.1	套接口加入组播组	417
15.7.2	MRT_DONE	379	16.9.2	套接口离开组播组	418
15.7.3	MRT_ADD_VIF 和 MRT_		16.10	维护网络设备组播状态	419
	DEL_VIF	380	16.10.1	被阻止的组播源列表的维护	421
15.7.4	MRT_ADD_MFC 和 MRT_		16.10.2	网络设备加入组播组	421
	DEL_MFC	380	16.10.3	网络设备离开组播组	425
15.7.5	MRT_ASSERT	380	16.11	ip_mc_source()	430
15.8	组播的 ioctl	380	16.12	ip_mc_msfilter()	434
15.8.1	SIOCGETVIFCNT	380	16.13	网络设备组播硬件地址的	
15.8.2	SIOCGETSGCNT	380		管理	436
15.9	组播报文的输入	381	<b>第 17 章</b>	<b>邻居子系统</b>	437
15.10	组播报文的转发	383	17.1	什么是邻居子系统	437
15.10.1	ip_mr_forward()	383	17.2	系统参数	437
15.10.2	ipmr_queue_xmit()	385	17.3	邻居子系统的结构	438
15.11	组播报文的输出	388	17.3.1	neigh_table 结构	438
<b>第 16 章</b>	<b>IGMP: Internet 组</b>		17.3.2	neighbour 结构	441
	<b>管理协议</b>	390	17.3.3	neigh_ops 结构	444
16.1	in_device 结构中的组播参数	390	17.3.4	neigh_parms 结构	445
16.2	ip_mc_list 结构	391	17.3.5	pneigh_entry 结构	447
16.3	系统参数	393	17.3.6	neigh_statistics 结构	447
16.4	IGMP 的版本与协议结构	393	17.3.7	hh_cache 结构	448
16.4.1	IGMP 的版本	393	17.4	邻居表的初始化	449
16.4.2	第一版和第二版的 IGMP		17.5	邻居项的状态机	450
	报文结构	395	17.6	邻居项的添加与删除	452
16.4.3	第三版的 IGMP 查询报文结构	395	17.6.1	netlink 接口	452
16.4.4	第三版的 IGMP 报告结构	396	17.6.2	ioctl	456
16.5	IGMP 报文的输入	398	17.6.3	路由表项与邻居项的绑定	456
16.6	函数	399	17.6.4	接收到的并非请求的应答	456
16.6.1	ip_mc_find_dev()	399	17.7	邻居项的创建与初始化	456
16.6.2	ip_check_mc()	400	17.7.1	neigh_alloc()	456
16.7	成员关系查询	400	17.7.2	neigh_create()	457
16.8	成员关系报告	404	17.8	邻居项散列表的扩容	459
16.8.1	最近离开组播组列表的维护	404	17.9	邻居项的查找	460
16.8.2	is_in()	404	17.9.1	neigh_lookup()	460
16.8.3	add_grec()	406	17.9.2	neigh_lookup_nodev()	461
16.8.4	普通查询的报告	409	17.9.3	__neigh_lookup()和	
16.8.5	V1 和 V2 的报告以及 V3 的			neigh_lookup_errno()	461
	当前状态记录报告	410	17.10	邻居项的更新	461
16.8.6	主动发送组关系报告	413	17.11	垃圾回收	465
16.9	维护套接口组播状态	416	17.11.1	同步回收	465

17.11.2 异步回收 .....	466	19.1.1 路由的要素 .....	503
17.12 外部事件 .....	468	19.1.2 特殊路由 .....	505
17.13 邻居项状态处理定时器 .....	469	19.1.3 路由缓存 .....	505
17.14 代理项 .....	472	19.2 系统参数 .....	506
17.14.1 代理项的查找、添加和删除 .....	472	19.3 路由表组成结构 .....	506
17.14.2 延时处理代理的请求报文 .....	472	19.3.1 fib_table 结构 .....	508
17.15 输出函数 .....	474	19.3.2 fn_zone 结构 .....	510
17.15.1 丢弃 .....	474	19.3.3 fib_node 结构 .....	511
17.15.2 慢速发送 .....	474	19.3.4 fib_alias 结构 .....	511
17.15.3 快速发送 .....	477	19.3.5 fib_info 结构 .....	512
<b>第 18 章 ARP: 地址解析协议</b> .....	<b>480</b>	19.3.6 fib_nh 结构 .....	515
18.1 ARP 报文格式 .....	480	19.4 路由表的初始化 .....	516
18.2 系统参数 .....	481	19.5 netlink 接口 .....	517
18.3 注册 ARP 报文类型 .....	483	19.5.1 netlink 路由表项消息结构 .....	517
18.4 ARP 初始化 .....	483	19.5.2 inet_rtm_newroute() .....	519
18.5 ARP 的邻居项函数指针表 .....	483	19.5.3 inet_rtm_delroute() .....	520
18.6 ARP 表 .....	484	19.6 获取指定的路由表 .....	520
18.7 函数 .....	485	19.7 路由表项的添加 .....	520
18.7.1 arp_error_report() .....	485	19.8 路由表项的删除 .....	526
18.7.2 arp_solicit() .....	485	19.9 外部事件 .....	528
18.7.3 arp_ignore() .....	486	19.9.1 网络设备状态变化事件 .....	528
18.7.4 arp_filter() .....	488	19.9.2 IP 地址变化事件 .....	529
18.8 IPv4 中邻居项的初始化 .....	488	19.9.3 fib_add_ifaddr() .....	529
18.9 ARP 报文的创建 .....	490	19.9.4 fib_del_ifaddr() .....	531
18.10 ARP 的输出 .....	490	19.9.5 fib_disable_ip() .....	534
18.11 ARP 的输入 .....	491	19.9.6 fib_magic() .....	534
18.11.1 arp_rcv() .....	491	19.10 选路 .....	535
18.11.2 arp_process() .....	492	19.10.1 输入选路:	
18.12 ARP 代理 .....	497	ip_route_input_slow() .....	535
18.12.1 arp_process() .....	498	19.10.2 组播输入选路:	
18.12.2 arp_fwd_proxy() .....	499	ip_route_input_mc() .....	539
18.12.3 parp_redo() .....	500	19.10.3 输出选路:	
18.13 ARP 的 ioctl .....	500	ip_route_output_slow() .....	541
18.14 外部事件 .....	501	19.10.4 fib_lookup() .....	546
18.15 路由表项与邻居项的绑定 .....	502	19.10.5 fn_hash_lookup() .....	548
<b>第 19 章 路由表</b> .....	<b>503</b>	19.11 ICMP 重定向消息的发送 .....	548
19.1 什么是路由表 .....	503		

# 第1章 预备知识

随着 Linux 的蓬勃发展和普及，想深入了解内核实现的技术人员也越来越多。而要真正深入了解内核，就需要阅读和分析内核源码。

Linux 内核源码可以从包括网络在内的很多途径得到，例如从 <http://www.kernel.org> 下载。当然，在很多发行版中，`/usr/src/linux` 目录也存放内核源码。目前最新的稳定版是 2.6.36.2。

很多技术人员畏惧阅读 Linux 内核源码，因为这样庞大而复杂的系统代码，阅读起来确实有很多困难。由于操作系统由多个模块组成，包括进程管理、内存管理、文件系统、驱动程序、网络等，而 Linux 内核也采用了面向对象技术进行设计和编码，因此实际上阅读源码也没有想象的那么高不可攀。只要有恒心，不必担心水平不够，困难都是可以克服的，很多知识可以边阅读源码边学习。

做任何事情都需要有方法和工具，同样，阅读 Linux 内核源码也是如此。由于内核源码非常庞大，因此不能全面铺开，而是要按照模块一个一个去攻克，本书的目的就是指导和帮助读者学习网络模块。要想比较顺利地阅读内核网络源码，事先最好对源码的知识背景有一定的了解。对于内核网络源码来讲，基本要求是：熟悉 C 语言，最好了解 GNU 对标准 C 的扩展；熟悉 GCC 编译器以及使用方法；熟悉操作系统的基本知识；熟悉 Linux 内核通用技术，包括内存管理、下半部、锁等；熟悉 TCP/IP 的原理。

本书讲述的代码来自 Linux-2.6.20，下载网址是 <http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.20.tar.bz2>。

俗话说：“工欲善其事，必先利其器”。像 Linux 内核源码这样的复杂程序，阅读时遇到的问题肯定会像一个越滚越大的雪球。例如一个函数经常进行多层次的调用，一个结构会有多个结构多层次的嵌套，而且会涉及多个其他的相关文件，对代码进行多层次跟踪后，说不定一下子还回不到原先阅读的函数或结构。因此需要一个好的阅读工具。

由于大部分技术人员和爱好者都比较熟悉 Windows 或 Linux 平台，所以在此介绍两个工具。

一个是 Windows 下的 Source Insight，它是一个共享软件，有 30 天免费期，可以从 <http://www.sourcelynx.com> 下载。安装和使用这个软件都非常简单，安装完成并运行后，先选择“Project”菜单下的“new”命令，新建一个工程，输入工程名，接着把欲读的源码加入（可以添加整个目录）后，软件就会分析所加的源码。分析完后，就可以进行阅读了。对于打开的阅读文件，如果想看某一变量的定义，可先把光标定位于该变量，然后点击工具条上的相应选项，该变量的定义就会显示出来，对于函数的定义与实现也可以同样操作。

另一个是 Windows 和 Linux 下的 Source-Navigator，可以从 <http://sourceforge.net/projects/sourcenav/files/> 下载。它是 Red Hat 开发的一个 IDE，很适合用来阅读源代码——因为它能很好地解决文件定位和跳转问题，功能与 Source Insight 类似。运行 Source-Navigator 之后，该软件会建立源代码的 project 并扫描源代码文件，自动建立文件之间的索引，之后便可以进行阅读了。

读者也可在线阅读源代码，网址是 <http://lxr.linux.no/#linux+v2.6.20/>。

## 1.1 应用层配置诊断工具

在学习 Linux 网络模块实现的过程中，免不了要做些实验，来验证自己的想法或查看代码流程。Linux 提供了多种配置工具用于配置内核以及网络特性。

### 1.1.1 iputils

除了经常使用的 ping 外，iputils 包还包含 arping（用来生成 arp 请求），网络路由发现服务器 rdisc，以及其他一些程序（tracepath、clockdiff、tftpd 和 rarpd）。

### 1.1.2 net-tools

net-tools 网络工具包，最常用的就是 ifconfig, route, netstat 和 arp，还包括 hostname, nameif, plipconfig, rarp, slattach, ipmaddr, iptunnel 和 mii-tool 等工具。

### 1.1.3 iproute2

iproute2 是 Linux 特有的、新一代的网络配置工具包。其中 ip 是最常用的命令，它可以用来配置网络设备 (ip link)、IP 地址 (ip address)、路由 (ip route) 以及其他功能。tc 也是 IPROUTE2 工具包中常用的、用来配置流量控制的工具。除了以上两个工具，还包括 ss, nstat, ifstat, rtacct, arpd, lstat, maketable, normal, pareto 和 paretonormal。

## 1.2 内核空间与用户空间的接口

内核提供接口给用户空间程序，以使用户进程进行信息的读取或配置。除了经典的系统调用接口，Linux 还可通过几个特殊的接口，包括虚拟文件系统和 netlink 套接口，获取相关信息。

procfs 和 sysctl 都可以导出内核内部信息，但是 procfs 主要用于导出只读数据，而 sysctl 导出的信息是可写的（只有超级用户可写）。如果导出的是只读数据，那么选择 procfs 还是 sysctl 就与导出数据的数量有关。如果导出的是简单变量，那么应该使用 sysctl。反之，如果导出大量复杂的数据，并且要求导出的数据有固定格式，就应该使用 procfs。

### 1.2.1 procfs

procfs 是一个虚拟文件系统，通常挂接在 /proc 目录下，内核通过文件的形式将内部信息展现给用户空间程序。这些文件都不是磁盘文件，但是可以使用 cat 命令读取它们，或者通过重定向符号 (>) 写它们，甚至可以像普通文件一样设定它们的读写权限。

网络代码注册的文件一般位于 /proc/net 目录下，在该目录下存在不少文件，有一种比较特殊的文件，比如 tcp、udp 等。这种文件的格式比较固定，Linux 称之为综合文件 (synthetic files)，它们最大的特点就是由一系列记录组成，类似于数据库表中的一条条记录。用这种格式来描述系统中的一些统计或状态是比较适合的，因此 proc 文件系统特地增加了对这种文件类型的支持。因为它只存在于 proc 文件系统中，文件系统提供了对此文件框架的支持，而在使用此类型文件时，只需关心数据的处理即可。

大多数网络功能在初始化时，都会在 /proc/net 中注册一个或多个这样的文件，无论初始化

动作发生在系统启动时还是模块加载时。当用户读取某个文件时，内核会调用一组内核函数来输出相应的信息。

创建和删除文件可以分别调用 `proc_net_fops_create()` 和 `proc_net_remove()`，这两个函数分别包装了函数 `create_proc_entry()` 和 `remove_proc_entry()`。需要注意的是 `proc_net_fops_create()` 在创建文件的同时也初始化了这个文件的操作函数。来看下面的例子。

这是 IP 组播在 `/proc/net` 目录下注册 `ip_mr_vif` 文件的例子：

```
1717 static struct file_operations ipmr_vif_fops = {
1718     .owner      = THIS_MODULE,
1719     .open       = ipmr_vif_open,
1720     .read       = seq_read,
1721     .llseek     = seq_lseek,
1722     .release    = seq_release_private,
1723 };

1899 void __init ip_mr_init(void)
1900 {
1901     ....
1902     #ifdef CONFIG_PROC_FS
1903         proc_net_fops_create("ip_mr_vif", 0, &ipmr_vif_fops);
1904     ....
1905     #endif
1906 }
```

`proc_net_fops_create` 的三个参数所代表的含义是：文件名是 `ip_mr_vif`，文件属性是只读，文件的操作函数句柄是 `ipmr_vif_fops`。当用户读文件时，`file_operations` 结构里面的函数会以数据块的方式将数据返回给用户。这在读取大量相同类型数据的时候很有用。例如，可以一次读取组播虚拟接口。

`ipmr_vif_open()` 会进行一系列初始化：它会注册一组函数指针，这些函数指针在 `procfs` 的例程中被调用，用于遍历返回给用户的数据：一个函数用于初始化打印动作，另一个用于向前移动一次指针，还有一个用于打印一个数据项。这些函数会在内部保存必要的上下文信息以便了解当前的打印项在哪里，以及可以从哪个地方重新开始打印等。

```
1686 static struct seq_operations ipmr_vif_seq_ops = {
1687     .start = ipmr_vif_seq_start,
1688     .next  = ipmr_vif_seq_next,
1689     .stop  = ipmr_vif_seq_stop,
1690     .show  = ipmr_vif_seq_show,
1691 };

1693 static int ipmr_vif_open(struct inode *inode, struct file *file)
1694 {
1695     ....
1696     rc = seq_open(file, &ipmr_vif_seq_ops);
1697     if (rc)
1698         goto out_kfree;
1699     ....
1700 }
```

## 1.2.2 sysctl(/proc/sys 目录)

sysctl 接口允许用户读取或者修改内核参数。在用户空间可以通过两种方法访问 sysctl 导出的变量。一种方法是通过 sysctl 系统调用，另一种是通过 procfs。如果内核支持 procfs，它会在 /proc 目录下增加一个特殊的目录 (/proc/sys)，这个目录里包含了 sysctl 导出变量的列表。

procfs 包里的 sysctl 命令可以用于配置 sysctl 接口导出的内核参数，实际上这个命令是通过 /proc/sys 目录下的文件与内核通信的。大多数 Linux 发行版的内核都默认包含了 sysctl 的支持。

通过 sysctl 系统调用和命令来设置和获得运行时内核的配置参数是一种有效的方式，通过这种方式，用户可以在内核运行的任何时刻修改和获取内核的配置参数。例如，通过 `cat /proc/sys/net/ipv4/ip_forward` 来获取内核网络层是否允许转发 IP 数据报。通过 `echo 1 > /proc/sys/net/ipv4/ip_forward` 将内核网络层设置为允许转发 IP 数据报。通过 `cat /proc/sys/kernel/ostype` 可以获取操作系统的类型，其实也可以通过 sysctl 系统调用达到同样的效果。

```
#include <unistd.h>
#include <sys/syscall.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/sysctl.h>

int _sysctl(struct __sysctl_args *args );

#define OSNAMESZ 100

int main(void)
{
    struct __sysctl_args args;
    char osname[OSNAMESZ];
    size_t osnamelth;
    int name[] = { CTL_KERN, KERN_OSTYPE };

    memset(&args, 0, sizeof(struct __sysctl_args));
    args.name = name;
    args.nlen = sizeof(name)/sizeof(name[0]);
    args.oldval = osname;
    args.oldlenp = &osnamelth;

    osnamelth = sizeof(osname);

    if (syscall(SYS_sysctl, &args) == -1) {
        perror("_sysctl");
        exit(EXIT_FAILURE);
    }
    printf("This machine is running %*s\n", osnamelth, osname);
    exit(EXIT_SUCCESS);
}
```

图 1-1 显示了以 root\_table 为顶点形成的树状结构。

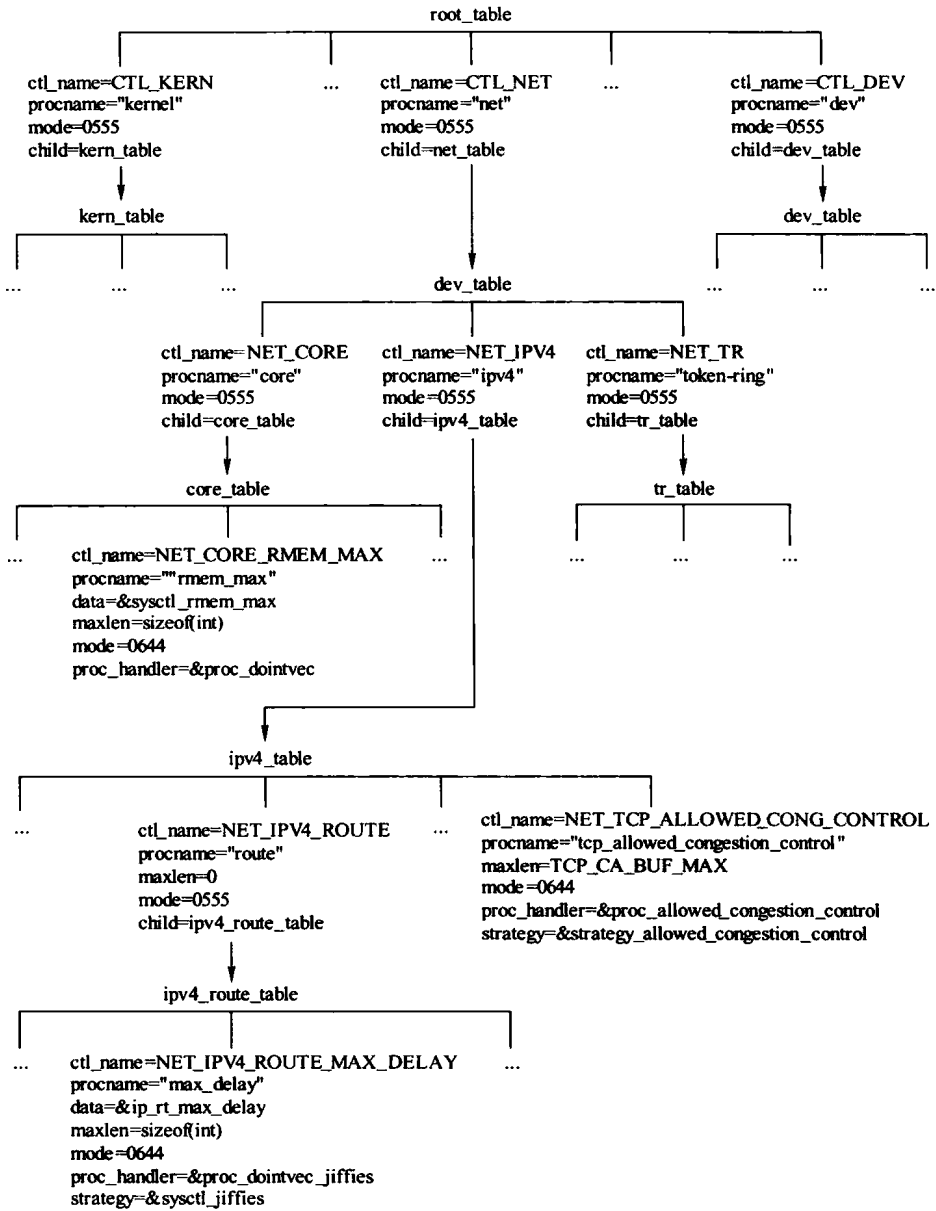


图 1-1 sysctl 的组织

### 1.2.3 sysfs(/sys 文件系统)

sysfs 是 Linux 2.6 提供的一种虚拟文件系统，这个文件系统不仅可以把设备和驱动程序的信息从内核空间导出到用户空间，也可以对设备和驱动进行配置。

sysfs 的目的是把一些原本在 procfs 中的设备独立出来，以设备树的形式呈现给用户。sysfs 最初名称为“driverfs”，当初只是为了要对新的驱动程序模型除错而开发出来的，然而随着代码的更新，新的“驱动程序模型”和“driverfs”证明了对内核中的其他子系统也有用处。Kobjects 开发出来之后，用于核心部件的管理机制，因此 driverfs 也被更名为 sysfs。



## 1.2.4 ioctl 系统调用

ioctl 系统调用可以操作一个文件，它通常用于实现特殊设备的操作，而这些操作在标准的文件系统调用中没有提供。ioctl 也可以操作套接口描述符，net-tools 工具包就是通过 ioctl 系统调用与内核交互的。

## 1.2.5 netlink 套接口

netlink 套接口是网络应用程序与内核通信最新、最常用的接口，IPROUTE2 包中的大多数命令都使用这个接口。netlink 套接口的描述在 RFC 3549 中。

netlink 套接口使用起来非常简单，通过套接口标准的 API 来打开、关闭，或者发送和接收信息。要在用户态创建一个 netlink 套接口，代码如下所示：

```
socket(PF_NETLINK, SOCK_DGRAM, NETLINK_ROUTE)
```

在内核中定义了多种协议，每一种都被协议栈中的一个或一组内核组件使用。例如上述代码中的 NETLINK\_ROUTE 协议就会被多个模块使用，如路由和邻居协议，而 NETLINK\_FIREWALL 用于防火墙模块。

## 1.3 网络 I/O 加速

尽管技术有了巨大的进步，但是 TCP/IP 协议栈的处理方式却几乎没有变化。也就是说，即使用户使用最先进的 CPU，依然要处理那些未经优化的 TCP/IP 协议，由此产生巨大的系统开销。例如，TCP/IP 的传输过程中需要封装、解包，这些动作对于处理器而言并不是一个复杂的过程，但是会占用处理器周期，而且网络带宽越高，这个问题越严重。系统开销的增大不仅仅表现在占用较多的处理器周期，还会导致处理网络相关数据时的内存访问效率降低。这又会进一步降低 CPU 效能和网络效率。

过去，网络流量较低，处理网络相关数据所产生的开销，远远低于用于执行正常任务的开销，所以并未引起重视。现在，随着网络流量大幅度提升，处理网络相关数据所产生的系统开销越来越不能忽视，甚至已经影响到了正常应用。现有几种解决方案：

### (1) TSO (TCP Segmentation Offload)

通过网络设备上的专用处理器处理部分或者全部的封包，借此来降低对于系统处理器资源的占用，不过这种解决方案只对具有某些特征的数据包有效。

### (2) RDMA (Remote Direct Memory Access, 远程直接内存访问)

发送端系统直接将有效数据送至目的系统指定的内存中，无需移动数据包的时间消耗，因此大大提升了网络传输的效率。但是这种技术需要专用的网络设备，应用程序也需要进行修改，甚至还增加了一个 RDMA 层的封装过程，而且这种操作风险较高，因此目前看来还不是一个吸引人的解决方案。

### (3) Onloading 技术

将系统处理器作为处理网络流量的第一引擎，尽可能地提升 CPU 处理网络数据包的效率，这种思想已经被英特尔借鉴。

目前，Linux 已经支持 TSO 和英特尔 I/O 加速技术，对于网络设备的 DMA 本书不作论述。要支持英特尔 I/O 加速选项 CONFIG\_INTEL\_IOATDMA，必须在编译前选择支持“Intel I/OAT