TCP/IP ARCHITECTURE, DESIGN, AND IMPLEMENTATION IN LINUX

Sameer Seth
M. Ajaykumar Venkatesulu

IEEE computer society

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

TCP/IP ARCHITECTURE, DESIGN, AND IMPLEMENTATION IN LINUX

TCP/IP ARCHITECTURE, DESIGN, AND IMPLEMENTATION IN LINUX

Sameer Seth
M. Ajaykumar Venkatesulu

IEEE computer society

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data is available.

ISBN 978-0470-14773-3

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE

For more than a decade, Linux has been the most popular choice for server technology, embedded systems, or research work in the networking domain. It slowly gained momentum beginning with the student community and slowly reaching researchers and the corporate world. Networking, when combined with Linux, gives birth to an innovative product line, be it in the high-end telecom sector, data centers, or embedded systems, and so on.

In 1996, I was introduced to Linux while doing my first assignment on TCP/IP socket programming. At that time, I had a very little knowledge about a server program using a unique port number to register itself with the system or a client program using the same port number to communicate with the server. I also had little knowledge of an IP address that is fed to the client program to identify the host. I then set myself to learn about how all that was made possible.

Much information needed to be explored at that time, such as system calls, protocols, Linux kernel, drivers, and kernel framework that supports the stack, and so on. Slowly, I explored the Linux kernel and user–land program interaction with that kernel by writing new system calls and kernel modules.

This learning process began with the *TCP/IP Illustrated, Volume 1* by the honorable Richard Stevens. But it continued to be really difficult to map the protocol with the implementation on Linux because there was so little documentation, and available books provided hardly any information. So, I decided to dive deep into the jungle of the huge source base to find out how the stack is implemented. Finally, I got hooked to the socket and VFS layer to understand how socket layer is linked to the VFS layer. Then slowly I was pointed to the TCP layer and the first routine that interfaces TCP protocol to send out data. Then the journey of documenting and experimenting with the TCP/IP stack began. When the documentation had grown big enough, the idea of making it available to the Linux community emerged. But writing a book was beyond my strength and it was too much work, requiring a lot of time and dedication. But I was determined to expose the complex topic to the Linux community to whatever extent I could even if it demanded many requirements. The absence of detailed, leveled documentation or a book that would have made the subject easier to understand, forced me to think about the topic. The idea of writing a book was supported when I received acceptance on the subject from IEEE Computer Society Press and John Wiley & Sons.

Working on the book along with office work became difficult so I searched for a co-author who would help cover some of the topics. After a long struggle, I convinced M. Ajaykumar Venkatesulu to be my co-author and work on a giant and most complex routing subsystem and QOS.

This text tries to cover almost all the aspects of TCP/IP stack and supporting kernel framework. The idea is to present the topic in a way that dilutes its complexity so that it can be easily understood. To understand TCP/IP implementation on any OS, we need to understand the kernel frameworks that support the stack. On Linux, these frameworks include VFS layer, socket framework, protocol layer, timers, memory management, interrupt handling, softIRQ, kernel threads, kernel synchronization mechanism, and so on. This is the kernel perspective of the stack. Apart from this, we also need to know the basics of the communication protocol and application interfaces (system calls) to open TCP communication sockets and the client–server program. This knowledge is helpful as a reference for experienced professionals and for students willing to learn the complex subject and contribute to the Linux community.

This book is written for the Linux kernel 2.4.20. The newest kernel version 2.6 does not have much variation as far as the TCP/IP stack is considered. Kernel version 2.4 is the most widely accepted kernel in the Linux world. Version 2.6 specific changes will be discussed in subsequent revisions of the book.

AUDIENCE

The book is targeted for large cross section of audience:

Researchers at Worldwide Premier Institutes. Researchers who work on various aspects of the TCP/IP stack find BSD the most suitable networking OS. But BSD is not a popular choice in the corporate world. So, the next most popular choice of researchers is the Linux OS and improvement of the TCP/IP stack performance on this OS. Networking is currently the most popular field for research because of growing usage and popularity of the Internet. Mostly, researchers prefer an OS with commercial viability that can run on cheap hardware.

Academia. Advanced academic degree projects, such as MS, M. Tech., B. Tech. and PG, are mostly done on Linux because it was the first UNIX-like OS available with fairly good documentation and stability. In the networking area, students usually choose Linux over TCP/IP for their project work. The project may require modifying the router or TCP performance, implementing some new TCP/IP RFC, network drivers, implementing secured IP layer, or improving scalability factor to handle network traffic.

Corporations. For the most part, the corporate world has widely accepted Linux as the base OS for networking products. Many companies are developing network products, such as IP security, QOS (class-based routing), developing routers, bandwidth management products, cluster servers and many more, which require modifying the TCP/IP stack or writing a new module altogether that fits into Linux TCP/IP stack somewhere. Linux is not only popular as an open system but is also a major choice for embedded products or real-time OS. These embedded products are mostly developed for networking domains such as routers, embedded web servers, web browsers, and so on.

Entrepreneurs. New ideas keep popping up which need to be turned into products. With the Internet gaining popularity, many ideas have been born to develop networking products. Linux is once again the most popular choice for development among entrepreneurs.

The Open Source Community. Because of the growing popularity of Linux and Internet technologies, many fresh college graduates or even software professionals want to contribute to Linux networking capabilities. Their goal is to make Linux more powerful, stable, secure, and full of network capabilities in order to meet corporate requirements in every possible way. Many professionals want to contribute to Linux networking capabilities but don't find enough time to get acquainted with its networking stack and the kernel framework.

Defense Organizations. There is a growing popularity of Linux as network OS in defense organizations with increasing military adoption of Linux IP security with some modifications for secured military network transactions.

All these audiences require a thorough knowledge of Linux TCP/IP stack and kernel framework for networking stacks. To understand TCP, IP, BSD sockets, firewall, IP security, IP forwarding, router network driver, complete knowledge of how networking stack implementation and design work is needed. If IP security or firewall implementation is wanted, then knowledge of how the packet is implemented in Linux, how and where packet is passed to the IP layer, how the IP processes the packets and adds headers, and finally how the IP passes the packet to the device driver for final transmission is needed. Similarly, implementation of the QOS or some modifications in the existing implementation is needed, knowledge of Linux routing table implementation, packet structure, packet scheduling and all related kernel frame work including network soft IRQs is required. So, anything and everything that requires modifying the Linux network stack or adding a new feature to the stack, requires complete knowledge of the design and implementation of Linux TCP/IP stack.

ORGANIZATION OF THIS BOOK

This book completely explains TCP/IP protocol, its design, and implementation in Linux. Basically, the book begins with simple client–server socket programs and ends with complex design and implementation of TCP/IP protocol in Linux. In between, we gradually explain the different aspects of socket programming and major TCP/IP-related algorithms. These are:

Linux Kernel and TCP/IP Application Interfaces: Chapter 1 covers the Linux kernel basics and we kick start with kernel interfaces (system calls) to use TCP/IP protocol stack for communication.

Protocols: Chapter 2 covers TCP/IP protocols and supporting protocols such as ARP and ICMP. We cover some of the major RFCs with illustrations to acquaint the reader with the protocols so that it will be easy to map Linux implementation on Linux in further chapters.

Sockets: Chapter 3 explains the implementation of BSD socket implementation in the Linux kernel. Here we discuss in detail how socket layer is hooked to VFS layer and how various protocols are hooked to BSD socket.

Kernel Implementation of Connection Setup: Chapter 4 explains the client–server application with the help of the C program. We explain the complete process of connection setup with the help of tcp dump output in different chapters. We cover kernel implementation of system calls used by application program to implement client–server interaction. We see how connections are accepted on the server side and at the same time, learn how the server program registers with the kernel to bind to a specific listening port.

Linux Implementation of Network Packet: Chapter 5 explains sk_buff which represents network packet on Linux. We explain important routines that manipulate sk_buff.

Movement of Packet Across the Layers: Chapter 6 covers the complete TCP/IP stack framework, showing how the packet is generated and trickles down the network stack until it is out of the system. Similarly we explain the complete path taken by a packet received from the device to reach the owning socket, covering complete kernel framework that implements TCP/IP stack on Linux.

TCP recv/send: Chapters 7 and 8 address TCP receive/send implementation and cover all the aspects related to TCP receiving and sending data. We also explain the TCP segmentation unit when an ICMP error (mss change for the route) is received by the TCP. There is a small description of how urgent data are processed.

TCP Socket Timers and Memory Management: The kernel keeps track of memory consumed by a connection at the socket layer so that a single-socket connection is not able to hog all the system memory because of a misbehaving application. We also try to collapse sequential buffers in the receive queue when the application is not reading enough fast and socket has exhausted its quota. This aspect of memory management is covered in Chapter 9. TCP is an event-driven protocol. TCP implements timers to track loss of data, to send delayed ACKs, to send out zero window probes, and so on. Chapter 10 addresses all these aspects.

TCP State Machine: Chapter 11 covers TCP core processing, such as reception of packets, sending ACKs, sliding window protocol, Nagle's algorithms, scheduling of delayed ACK's, processing of out-of-order segments, processing SACK, D-SACK, and so on. The tcp_opt object represents state machine implementation on Linux. Chapter 12 covers TCP congestion control algorithms implementation.

Netlink Sockets: User–land applications, such as netstat and iproute, and routing protocol daemons use special netlink sockets to update/read routes and configure QOS in the kernel. We cover netlink sockets in Chapter 13.

IP Layer and Routing Table Implementation: Chapter 14 covers implementation of routing table (FIB) on Linux. We also explain different aspects associated with routing, such as multipathing, policy routing, and so on. This chapter also explains the different kernel control paths that update kernel routing tables and route cache management.

IP QOS: IP in today's network is an advanced topic and is used for different services in the public network. Linux implements QOS very cleanly and we discuss PFIFO and CBQ queuing discipline implementation in Chapter 15.

Netfilter Framework: Linux provides extensions to the TCP/IP stack by way of the netfilter framework. These extensions can be firewall, masquerading, IP security, and so on. Chapter 16 covers netfilter hooks at different layers in the stack and also netfilter implementation.

SoftIRQ Implementation for Scalability: Network frames are received in the kernel memory in the interrupt handler code but complete processing of the packets can't be done in the interrupt handler. Linux associates softIRQ, one each for reception and transmission of packets for processing of packets. Chapter 17 explains net softIRQ framework with the help of illustrations. This chapter completely explains the high scalability of Linux on SMP architecture in handling network traffic.

Link Layer and DMA Ring Buffers: Chapter 18 covers link layer(device driver) processing of packets. Design and working of DMA ring buffer for reception and transmission are also addressed and are explained with the help of a device driver and interrupt routines for a real device.

Debug TCP/IP Stack: Debugging the TCP/IP stack is discussed in Chapter 19. The lkcd (linux kernel crash dump) debugger is used to illustrate the debugging technique, peeking into different kernel data-structures associated with TCP/IP stack.

LEVEL OF DISCRIPTION

As outlined here, we have touched upon critical portions of the implementation that are required to understand core TCP/IP stack and kernel framework. Each chapter begins with a chapter outline and ends with a summary that highlights important points. Source-level explanations with diagrams are provided where ever required. Important routines are explained line-by-line. Code snippets are provided for all those routines with line numbers and files of code snippet. Sometimes routines are so big that they are split into different code snippets. Routines that are called from the main routines are explained in different sections. If the called routine is a couple of lines long, there is no separate section for those routines. Line number and code-snippet number (cs-) are provided with the explanation to assist understanding. When the routines are very big in size, notification is provided at the beginning of the section stating, *see cs ••.••, unless mentioned*; this means that where ever line numbers are mentioned, we need to see the code snippet mentioned at the start of the section.

In the explanation if we encounter some concept that is already explained in some other section, a cross reference to that section is provided, as *see Section* ••. ••. Cross references are provided because the subject is interrelated, for example while explaining queuing of incoming TCP packet, we refer to sockets receive buffer. If we have exhausted the receive socket buffer, we need to call routines to collapse receive queue to make space for the new TCP data segment. For this we may need to refer to a section from the TCP memory management chapter. We have explained major data structures with significance separately. Where ever that has not been done, fields of those data-structures are explained as and when they appear in the routines.

Examples and illustrations are provided where ever it is required to make subject easier to understand. For example, diagrams to link various kernel data structures are drawn to illustrate connection requests in the SYN queue. Then we illustrate shifting of connection requests from SYN queue to accept queue when a three-way handshake is over with the help of diagrams. All these illustrations assist in visualizing the complex data structures and scenarios.

Sameer Seth

Bangalore, India
September 2008

ACKNOWLEDGMENTS

For me, this is the heaviest section of the book that carries the most weight. First of all, I'm very thankful to my family for being so supportive and patient when I was working on the title, with little time left for them. My wife, Sumam, provided selfless support to the work right from day one. She provided me with confidence to convert my hard work into a book on the day she provided me with the list of publishers. When submitting my book proposal, only 20% of the work was done and that too was not organized.

I thank my co-author, M. Ajaykumar Venkatesulu, who agreed to join hands with me at the much-needed hour. His commitment eased the load on my shoulders and he worked very hard with all dedication to make this possible. He had a really tough time setting up QOS on Linux, with a couple of Linux boxes, and modifying the kernel for his illustrations.

I'd like to thank the very first person at the IEEE Computer Society with whom I interfaced, Deborah Plummer, who worked on the proposal until it was finished. She helped me in many ways to understand the publication process and was very patient all through, clarifying my doubts. IEEE Staffers, Janet Wilson and Dante David, were so nice and prompt throughout the review process. Even a small communication gap caused serious concerns because this was the first time I was working on such a big project. But Janet and Dante were patient and always prompt in their replies to make sure that all my concerns were addressed. I was introduced to Lisa Van Horn from Wiley much later, when the book had entered the production phase. It is a great experience working with her because she spent time educating me at every point. At times I would be very irritating to her by asking silly doubts but she tackled them all with grace. She has worked very hard editing the book because there were grammatical corrections in almost every line. Through the production process, she was very helpful, cooperative, and prompt in the same way.

There are a few names without which this book would look incomplete. I thank Richard McDougall, the respectable author of *Solaris Internals,* for time spent educating me on the publication process. His inputs helped me achieve the most from my hard work. The respectable senior engineer from SGI and owner of the dwarf extract utility for lkcd, Cliff Wickman, is owed thanks for without him this book would have looked quite dry. He provided a tool to generate a kernel-type database (kerntypes) because the basic lkcd utility does not come with all the stubs for kernel data-structures in kerntypes. Without this tool, the debug chapter would not have been possible. He not only provided the tool but also helped get the kernel-type database built for the kernel 2.4 when the tool was compatible only with kernel 2.6.

S. S.

Writing or co-authoring a book was never even in my wildest dreams. The opportunity came by chance and then it became my choice. God has been kind enough to give me such an amazing opportunity. I have a couple of people to thank with whom my words fall short. First of all I would like to thank the author of the book who had faith in me that I could write on this subject. He gave me a lot of trust when he gave me an opportunity to work on this book. It was solely his brainchild which he shared with me selflessly. He gave me guidance whenever I faced any difficulty in any subject matter. His valuable suggestions and most importantly his inspirations have made it possible for me to finish this assignment.

I thank my family for all their support: My father who stood beside me through all the odds and evens of life so that I could concentrate on this project; my newly wedded wife, Priyanka, who never complained when I had less or sometimes no time left for her; and lastly, my brother-in-law Balaji who has been a great source of inspiration in my life.

Last but not least, I thank Deborah Plummer, Janet Wilson, and Dante David from IEEE for being so cooperative and nice.

The book is not a result of any inspiration but the need of the day. When you have the strong desire to achieve something, then the whole of creation conspires to accomplish your goal.

M. A. V.