Linux/UNIX开发与应用系列

Linux 网络编程

张斌 高波 等 编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

Linux/UNIX 开发与应用系列

Linux 网络编程

张斌　高波　等编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

随着网络的日益普及,网络应用也越来越丰富。掌握网络编程技术,就能从本质上理解网络,真正把握网络的发展趋势。另外,对于一个程序员而言,编写网络程序也是件非常有趣的事情。

本书全面而又深入地介绍了网络编程技术。为了适应不同读者的需要,本书内容分为两个部分:基本部分和提高部分。基本部分包括前 5 章,介绍了基本的网络协议和套接字(socket)编程内容。提高部分介绍了 I/O 多路复用、非阻塞 I/O、Inetd 超级服务器、带外数据、原始套接字和数据链路层编程等专题。对于大多数专题给出了相应的示例,其内容涵盖了 UNIX/Linux 网络编程的大部分内容,既可以作为基本部分的提高,也可以作为参考手册使用。通过阅读本书,读者可以全面掌握 UNIX/Linux 网络编程的知识。

本书适合于任何对 UNIX/Linux 网络编程有兴趣的读者。初级读者可以从本书前 5 章获得入门知识,中、高级读者可以从第 5 章以后的内容中深入了解关于网络编程方面的知识。

书　名:Linux 网络编程
作　者:张斌 高波 等
出版者:清华大学出版社(北京清华大学学研楼,邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者:北京市清华园胶印厂
发行者:新华书店总店北京发行所
开　本:787×1092 1/16　印张:20.5　字数:484 千字
版　次:2000 年 1 月第 1 版　2000 年 1 月第 1 次印刷
书　号:ISBN 7-302-01098-6/TP.2222
印　数:0001~6000
定　价:27.00 元

前　言

Linux是可靠性非常高的系统，实践已经表明，Linux系统可以不停机地稳定运行几个月，甚至几年。而且，因为Linux出色的性能，已经成为中低档次服务器的首选。在服务器市场上，它是Windows NT服务器的最有力竞争者。在桌面系统上，它与Windows 95/98还有一些差距，主要表现在使用上过于复杂，还不适合初级用户。目前已经有专门的人员在为提高它的易用性而努力，相信在不久的将来，Linux系统就能像Windows 95/98一样方便地使用。但对于广大计算机爱好者来说，易用性不会成为主要的障碍，因为它具有异常强大的功能，多花一点时间是值得的，这也是它成为目前最受计算机迷们欢迎的系统的原因。所以我们可以肯定地说，Linux系统的应用前景非常广阔。

Linux系统的一个主要特点是它的网络功能非常强大，因为它自身就是在Internet上诞生的。随着网络的日益普及，基于网络的应用也将日益增多。在这个网络时代，掌握了Linux的网络编程技术，将令每一个人处于不败之地，而且，Linux网络编程是一件非常有趣的事情，学习Linux网络编程，可以让我们真正体会网络的魅力。

但是，很遗憾，目前市场上没有一本全面深入介绍Linux网络编程的书籍。要学习Linux网络编程，只有英文方面的资料可以参考，如在Internet上有一些指导性的文档，但内容很零散，不适合系统地学习。鉴于这种情况，我们决定写这本Linux网络编程书籍，较为全面深入地介绍Linux网络编程技术，希望能对广大Linux爱好者掌握Linux网络编程技术有所帮助。

为了能适应广大读者的需要，本书内容分为两个部分：基础部分和提高部分。前5章内容是Linux网络编程的基础知识，是初学者的入门知识，内容依次为：Linux网络编程概述、TCP/IP网络协议、套接字基本函数和UDP套接字。TCP/IP协议是Linux网络编程的基础，所以在讨论套接字编程之前，先介绍TCP/IP协议。初学者顺序阅读完这5章内容之后，可以掌握Linux网络编程的基本技术，编写简单的网络程序，为继续学习Linux网络编程的高级技术做好准备。从第6章开始的内容是Linux网络编程的高级技术，这是本书的主要内容。这部分内容依次介绍了：Linux进程概念、I/O多路复用、非阻塞式I/O、Inetd超级服务器、紧急数据、原始套接字编程等知识。这部分有些章之间是相关联的，如在阅读Inetd超级服务器的内容前，需要掌握Linux的进程概念；但是大部分内容之间的联系是很弱的，可以独立阅读。也就是说，提高部分的内容可以作为Linux网络编程参考手册使用，在编程需要时再阅读相关章节的内容。Linux进程、I/O多路复用和非阻塞式I/O是普通Linux网络编程的主要内容，这几章的内容可以满足编写绝大多数Linux网络程序的

需要；Inetd 超级服务器为 Linux 系统提供了很多 Internet 标准服务；紧急数据是 Linux 网络的中断功能；原始套接字提供访问 IP 协议层的功能。总而言之，本书涵盖了 Linux 网络编程的绝大部分技术内容，初学者可以从本书获得入门知识，中、高级读者也可以从本书深入了解 Linux 网络编程的知识。

本书由张斌、高波、李兀编著。在编写过程中，范正刚、张宇贻、贺英、刘丽海、汤博、李正阳做了很多辅助工作，吴鑫对本书的策划和总体结构也提出了不少中肯的建议，在此一并表示感谢。

限于时间和水平，不足之处在所难免，望读者多加批评指正。

作　者

1999 年 9 月

目　录

第 1 章　Linux 网络编程概述

1.1　Linux 网络的发展

Linux 的网络功能是 Linux 系统非常重要的组成部分，在学习 Linux 的网络编程知识之前，了解 Linux 网络功能的发展过程可以帮助我们理解 Linux 的网络功能。现在让我们回顾一下这一激动人心的过程。

Linux 诞生时，UNIX 系统的网络功能已经相当成熟了，当时主要有两个版本：BSD 系统和 System V 系统。但是，Linux 网络的开发者选择了重新开发这种方式。因为那时还不能确定使用这两个版本是否会有版权的限制。另外一个原因是 Linux 的开发者有很多新的想法，他们希望用新的、自己的方法来实现，而不是模仿他人的方法，而且他们相信自己可以做得更好。于是他们就开始工作了。

最早领导 Linux 内核网络代码开发的是 Ross Biro < biro@yggdrasil.com > 。Ross 完成了一个简单的、不完整的实现。他主要实现的部分是从 WD-8003 以太网卡驱动程序扩展出来的一些例程序。这个实现使其他人可以测试和使用这个软件，有的人确实用它实现了网络功能。随着时间的推移，Linux 团体内部对网络功能要求的压力超过了 Ross 的承受能力，于是他放弃了作为开发领导者的身份。Ross 的努力是值得尊敬的，正是他的努力使这个工程启动了，并且他确实创建了一些有用的东西，而所有这些都是未来工作能够继续的动力。他的这部分工作仍是目前版本的一个重要部分。

Orest Zborowski < obz @ Kodak. COM > 为 Linux 网络内核编写了最初的 BSD 套接字(Socket)编程接口。对于编程者，这是一个巨大的飞跃，因为这使得很多已有的网络应用程序无需经过太大的修改就可以移植到 Linux 系统。这就丰富了 Linux 系统的网络应用，并进而促进了 Linux 网络功能的进一步发展。

大约在同时，Laurence Culhane < loz@holmes.demon.co.uk > 开发了最初的支持 SLIP 协议的驱动程序。很多使用电话线上网的用户这时也可以使用新的 Linux 网络软件了。随着用 Linux 上网的人数的增多，更多的人参与到使用和测试网络软件的工作中来。

Ross 辞职了一段时间之后，Fred van Kempen < waltje@uwalt.nl.mugnet.org > 承担起了领导开发的任务。Fred 对 Linux 网络功能的发展方向有一些雄心壮志，他编写了一系列网络代码，被称为 NET-2 内核代码，这是一个用户能很好使用的版本。Fred 在他的开发日程上添加了一些创新，如动态设备接口、业余电台 AX.25 协议支持和一个更加模块化的网络实现。Fred 的 NET-2 代码被很多 Linux 爱好者使用过，但它仍然只是标准内核代码的补丁程序，并未包含在普通发行版本中。Fred 的注意力集中在对标准网络实现的创新，而

不是如何更方便地配置网络。这时候的网络配置过程非常复杂,只能满足80%左右的用户的要求。用户们逐渐对这个复杂的网络内核不耐烦了,如同Ross面临的一样,Fred作为开发领导者所承受的压力不断增大。

Alan Cox < iialan@www.uk.linux.org > 提出了一种解决办法。他建议由他来承担Fred的NET-2代码的改进,他将调试这部分代码,使它变得可靠和稳定,满足大多数用户的要求,这样便可以减轻Fred的压力,让他能专注于创新的工作。Alan开始了他的工作,并获得了一定程度的成功,他的第一个版本的Linux网络代码被称为NET-2D(ebugged)。该代码在很多典型配置的情况下工作得很可靠,用户们对它很满意。随着工作的深入,Alan对于这个工程也产生了他自己的想法和实现技巧,这便带来了一个关于NET-2代码发展方向的问题。在Linux网络团体内部产生了两个不同的派别:一派的原则是"先使它工作起来,然后使它变得更好",另一派的原则是"先使它更好"。Linus最后作出了裁决,他给Alan的开发工作提供了支持,并在标准内核代码中包含了Alan的代码。这使Fred处于一个困难的境地,他的继续努力将缺乏广大的用户基础来使用和测试,这意味着发展将变得缓慢,而困难更大。在继续了一段时间后,Fred最后还是放弃了。Alan于是成为了新的Linux网络代码开发的领导者。

Donald Becker < becker@cesdis.gsfc.nasa.gov > 在编写低层网络代码方面做出了很大的贡献,他编写了很多以太网卡驱动程序,现在内核所包含的以太网卡驱动程序几乎都是他开发的。

到Linux 1.3.*内核代码出现时,核心网络代码已经变成NET-3了,这是当前版本的基础。Alan在网络代码的很多方面都有贡献,并且在一大批有天赋的人员的帮助下,这些方面都得到了很大的发展。Alan编写了动态网络设备和第一个标准AX.25和IPX的实现。他一直致力于网络代码的开发,慢慢地重新组织了网络代码的结构,并且将它扩展到了现在的状态。到编写本书时,Linux的网络代码已经升级到NET-4了。

在Linux的网络代码开发过程中,成百上千的程序员贡献了力量。可以说,Linux内核网络代码的开发是这种无组织的Linux开发模式的经典范例。我们可以从Linux系统的开发(包括网络代码的开发)过程中体会到这种开放模式的力量。

1.2 客户机 – 服务器编程模型

网络应用的标准模型是客户机 – 服务器模型,这是一个不对称的编程模型,通信的双方扮演不同的角色:客户机和服务器。一般发起通信请求的应用程序被称为客户机。用户一般是通过客户机软件来访问某种服务。客户机应用程序通过与服务器建立联系,发送请求,然后等待服务器返回所请求的内容。服务器一般是等待接收并处理客户机请求的应用程序。服务器通常由系统执行,在系统生存期间一直存在,等待客户机请求,并且在接收到客户机的请求之后,根据请求的内容,向客户机返回合适内容。它们之间的通信过程如图1-1所示。

网络程序的主要执行过程如下:

(1) 系统启动服务器执行。服务器完成一些初始化操作,然后进入睡眠状态,等待客

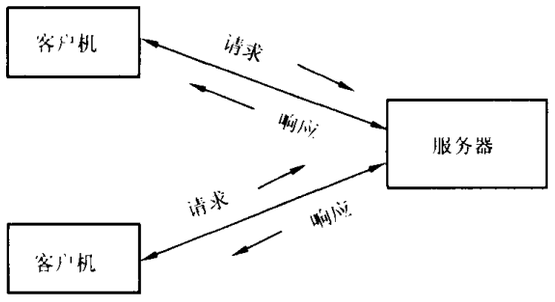

图 1-1　客户机 - 服务器通信模型

户机请求。

(2) 在网络的某台机器上,用户执行客户机程序。

(3) 客户机进程与服务器进程建立一条连接。

(4) 连接建立之后,客户机通过网络向服务器发出请求,请求某种服务。

(5) 服务器接收到客户机请求后,根据客户机请求的内容进行相应的处理,然后将处理结果返回。

(6) 服务器断开与客户机的连接,继续睡眠,等待其他客户机的请求。

Linux 系统中的很多服务器是在系统初启时启动的,如时间服务器、打印服务器、文件传输服务器和电子邮件服务器等。大多数时间这些服务器进程处于睡眠状态,等待客户机的请求。

根据处理请求方式的不同,服务器可以分为两种类型:

• 循环服务器(iterative server),

循环服务器在同一时刻只能处理一个客户机请求。服务器在接收到一个客户机请求之后,自己处理这个请求,在处理完这个请求之后再继续下一个请求。在处理一个请求的过程中,下一个请求将等待。通常,这种服务器只用于处理耗时比较短的服务,如 Linux 系统中的时间服务器(daytime)。这个服务器向客户机返回系统当前的日期和时间。

• 并发服务器(concurrent server)

并发服务器在同一时刻能够处理多个客户机请求。服务器一般使用多个进程来提供并发服务,其中的一个进程完成接收客户机连接的任务,而其他进程则完成具体的处理请求任务。通常,服务器在接收到一个客户机请求之后,创建一个子进程来处理这个客户机请求,而它自身则继续等待接收客户机请求。使用这种服务器,多个客户机请求可以同时由不同的子进程来完成。通常,使用这种服务器处理比较耗时的或要求快速的服务。Linux 系统中的很多服务器都是这种类型的服务器,如文件传输服务(FTP)。这个服务器返回客户机请求的文件。

这两种服务器模型有各自的优缺点。当服务耗时较长时,如果采用循环服务器,客户机请求将不能得到快速响应,甚至可能出现客户机请求被拒绝的情况,但是这种服务器消耗的系统资源很少;并发服务器为每一个请求创建一个子进程,所以可以保证同时处理多个客户机请求,使用这种服务器一般不会造成拒绝客户机请求的情况,但是创建子进程是很耗系统资源的操作,如果频繁创建子进程,将会加重服务器的负担。在设计服务器时,需要根据所提供的服务类型和服务质量来灵活选择服务器的模型。

1.3 一个客户机程序

下面两节介绍一个简单的客户机－服务器程序，使读者对网络客户机程序有个大致印象。这个客户机－服务器的工作过程非常简单：客户机与服务器建立连接之后，服务器向客户机返回一条消息。

客户机程序的源代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define PORT 3490
int main(int argc, char *argv[])
{
    int sockfd, nbytes;
    char buf[1024];
    struct hostent *he;
    struct sockaddr_in srvaddr;
    if (argc != 2) {
        perror("usage: client hostname \n");
        exit(1);
    }
    if ((he = gethostbyname(argv[1])) == NULL) {
        perror("gethostbyname");
        exit(1);
    }
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("create socket error");
        exit(1);
    }
    bzero(&srvaddr, sizeof(srvaddr));
    srvaddr.sin_family = AF_INET;
    srvaddr.sin_port = htons(PORT);
    srvaddr.sin_addr = *((struct in_addr *)he->h_addr);
    if (connect(sockfd, (struct sockaddr *)&srvaddr, \
                    sizeof(struct sockaddr)) == -1) {
        perror("connect error");
```

```
        exit(1);
    }
    if ((nbytes = read(sockfd, buf, MAXDATASIZE)) == -1) {
        perror("read error");
        exit(1);
    }
    buf[nbytes] = '\0';
    printf("read: %s",buf);
    close(sockfd);
}
```

执行这个客户机程序的方法是在命令行参数中指定服务器的域名地址,如:

```
bash $ netclient localhost
Hello, Network!
```

服务器返回"Hello, Network!"。

程序的第一部分是包含必要的头文件,有几个头文件几乎是每个网络程序都必需的,如 < sys/socket.h > 和 < netinet/in.h > 等。

主程序的第一个操作是获得服务器的地址。所有的 Internet 网络操作都是针对 IP 地址而言的,域名地址只是为人的记忆方便而引入的,对网络程序是没有意义的,所以在进行网络操作之前需要将用户指定的域名地址转换成网络函数使用的 IP 地址。函数 gethostbyname获得指定域名地址所对应的 IP 地址。

下一步是创建套接字(socket)。套接字是访问低层网络协议的接口,在进行网络操作之前,必须首先创建一个套接字,以便访问低层的网络协议。函数 socket 返回一个套接字的描述符。函数 socket 和文件操作函数 open 功能相似,都是返回一个描述符。在文件操作中,用文件描述符来标识打开的文件;在网络操作中,用套接字描述符来标识打开的套接字。

进行网络连接之前需要知道远程服务器的地址。这需要用远程服务器进程的 IP 地址和端口号来填充一个 Internet 套接字地址结构(结构 sockaddr_in)。地址的协议簇为 AF_INET,表示是 Internet 协议簇;端口号为 3499,这是服务器进程接受请求的端口号;最后用函数 gethostbyname 返回的远程服务器的 IP 地址来填充 IP 地址部分。

网络传送中使用统一的网络字节顺序,这不同于主机字节顺序。套接字地址结构中的 IP 地址和端口号必须是网络字节顺序。函数 htons 将主机字节顺序的短整数转换成网络字节顺序的短整数,所以可用这个函数将端口号转换成网络字节顺序。IP 地址是由函数 gethostbyname 得到的,已经是网络字节顺序了,不用再进行转换。

设置了远程服务器的地址之后,客户机调用函数 connect 与这个远程服务器建立一个 Internet 连接。

连接建立之后,客户机使用这个套接字与远程服务器进行通信。客户机和服务器根据预先定义的协议进行通信。网络通信的读、写操作与文件的读、写操作类似,也是使用

函数 read 和 write 来读、写数据。这个客户机的通信过程非常简单，调用函数 read 从服务器读取一些数据，然后客户机将这些数据显示在标准输出上。

最后，在完成通信过程之后，客户机用函数 close 关闭连接。

1.4 一个服务器程序

上一节已经介绍了一个简单的客户机程序，这一节将给出为这个客户机程序提供服务的服务器程序。服务器程序的源代码如下：

```
#include <stdio.h>;
#include <stdlib.h>;
#include <errno.h>;
#include <string.h>;
#include <sys/types.h>;
#include <netinet/in.h>;
#include <sys/socket.h>;
#include <sys/wait.h>;

#define MYPORT 3490
#define BACKLOG 5

main()
{
    int sockfd, new_fd;
    struct sockaddr_in srvaddr;
    struct sockaddr_in cliaddr;
    int sin_size;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket error");
        exit(1);
    }
        bzero(&srvaddr, sizeof(srvaddr));
        srvaddr.sin_family = AF_INET;
    srvaddr.sin_port = htons(MYPORT);
    srvaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sockfd, (struct sockaddr *)&srvaddr,
            sizeof(struct sockaddr)) == -1) {
        perror("bind error");
        exit(1);
    }
    if (listen(sockfd, BACKLOG) == -1) {
```

```
        perror("listen error");
        exit(1);
    }
    for (;;) {
        sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&cliaddr,
                        &sin_size)) == -1) {
            perror("accept error");
            continue;
        }
        printf("server: got connection from %s\n",
                inet_ntoa(cliaddr.sin_addr));
        if (write(new_fd, "Hello, Network! \n", 14) == -1)
                perror("write error");
        close(new_fd);
    }

    close(sockfd);
}
```

和客户机程序一样,服务器程序的第一部分是必要的头文件,如 < sys/socket.h > 等。

服务器程序的第一个操作也是创建一个套接字。函数 socket 返回一个标识这个套接字的描述符。

服务器的下一步操作是公开自己的地址。函数 bind 将服务器的地址和套接字绑定在一起。为了接收客户机连接,服务器必须公开自己的地址。(这一步操作不同于客户机程序,客户机一般不用绑定自己的地址。)程序首先填充服务器的套接字地址结构,地址仍使用 AF_INET 协议簇,端口号为 3490(注意要转换成网络字节顺序),服务器指定自己的 IP 地址为 INADDR_ANY,表示希望接收来自任何网络设备接口的客户机请求。然后服务器调用函数 bind 将这个 IP 地址与套接字绑定在一起。

然后服务器将套接字转换成倾听套接字(listening socket),这是只有服务器才需要执行的操作,函数 listen 完成这个操作。函数 listen 的作用是告诉内核,这个套接字可以接收来自客户机的请求。函数 listen 的参数 BACKLOG 指定这个套接字可以接收的最大未接收(unaccepted)客户机请求的数目。

以上 3 个函数调用 socket,bind 和 listen,是 TCP 服务器一般都需要执行的步骤。在成功执行了这 3 个函数之后,TCP 服务器能够使用这个套接字接收客户机连接。

然后服务器进入一个无限循环,处理客户机请求。服务器首先调用函数 accept 来获得一个客户机连接。当没有客户机连接时,服务器进程将在函数 accept 处睡眠。当客户机请求到达时,函数 accept 返回一个新的套接字描述符(new_fd)。这个新的套接字描述符标识与这个客户机连接。服务器用这个新的套接字来与客户机进行通信。服务器向客

户机返回一条消息。然后,服务器关闭这个连接,继续循环等待新的客户机请求。

程序最后调用函数 close 来关闭倾听套接字描述符。在这个程序中,因为服务器执行一个无限循环,所以这个函数将无法被执行到,在此列出这个函数,只是表示进程退出之前将关闭倾听套接字描述符。

这个简单的服务器程序是循环服务器,它依次处理客户机的请求,每次一个。

1.5 网络调试方法

在开始学习网络编程之前,有必要掌握一些常用的网络工具的使用方法。它们不仅在平时上网时有用,而且对于网络程序的调试也是必不可少的。希望读者在以后的学习过程中经常使用,并且阅读它们的帮助手册,以便掌握其详细功能。下面简单介绍一下它们的使用方法。

1. 命令 ifconfig

命令 ifconfig 是 Linux 系统配置网络设备接口的工具,也可以用它来查看已经配置好的网络设备接口信息。如:

```
bash $ ifconfig eth0
eth0  Linux  encap:10Mbps  Ethernet  Hwaddr  00:A0:24:9C:43:34
      inet addr:206.62.226.40  Bcast:206.62.226.63  Mask:255.255.255.224
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX  packets:484461  errors:0  dropped:0  overruns:0
      TX  packets:450113  errors:0  dropped:0  overruns:0
      Interrupt: 10  Base address:0x300
```

这个命令显示了第一块以太网卡的硬件地址为 00:A0:24:9C:43:34,这个网络设备接口的 IP 地址为 206.62.226.40,广播地址为 206.62.226.63,子网掩码为 255.255.255.224,标志 MULTICAST 表示这台主机支持多点传送。此命令还显示了这个网络接口的一些其他信息。

2. 命令 netstat

命令 netstat 显示网络连接、路由表和接口统计等网络信息。命令 netstat 有很多命令行选项。在此仅列出一些常用的选项,其他选项请参阅帮助手册。

- 无选项时显示网络连接状态,列出打开的套接字。
- "-a"选项显示所有套接字的状态。
- "-r"选项显示路由表的内容。一般同时指定"-n"选项,这样可以得到数字格式的地址,也显示默认路由器的 IP 地址。如:

```
bash $ netstat -rn
Routing tables
```