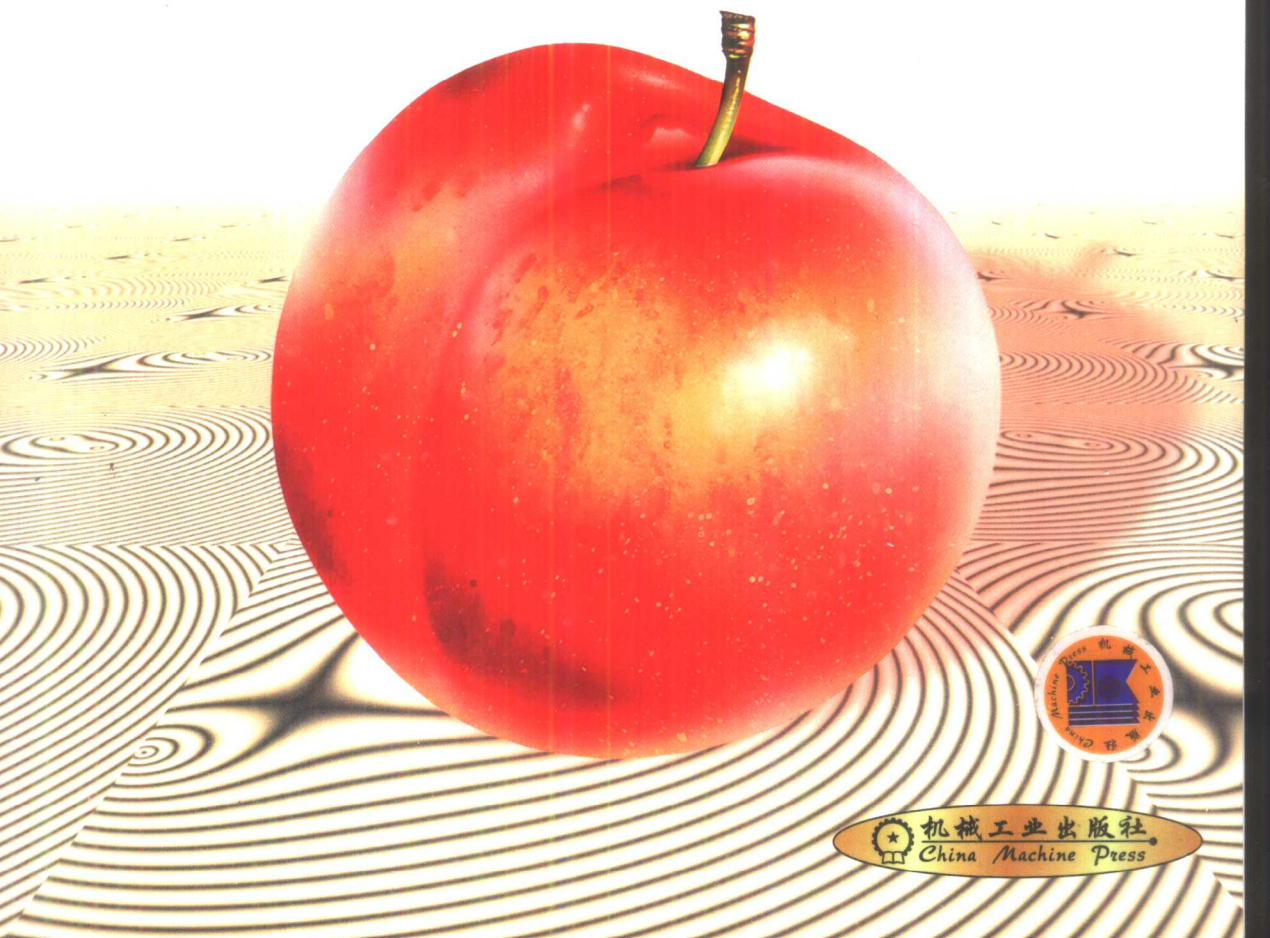
Linux 系统应用丛书

黄 超 等编著

LINUX 高级开发技术



机械工业出版社
China Machine Press

Linux 系统应用丛书

Linux 高级开发技术

黄　超　等编著
肖　薇　审

机械工业出版社

为了让读者对 Linux 的高级应用开发有一个基本的了解，本书首先介绍了 Linux 网络开发技术中的 Linux 设备驱动程序的知识和 Linux 数据库开发方面的基础知识。接着介绍了 MySQL 数据库，MySQL 语言的基础知识，以及 MySQL 的应用开发，最后介绍了 GTK 开发工具及编程实例和 PHP 开发及汉化方面的内容。

本书主要适用于已经对 Linux 的基本性能比较熟悉，希望在 Linux 平台上做一些基础开发的人员。同时由于本书是 Linux 开发技术中的高级部分，对于用户的水平要求相对要高一些。因此应对前面的基础开发比较了解之后再阅读本书，那样才能收到较好的效果。

图书在版编目（CIP）数据

（Linux 系统应用丛书）

ISBN 7-111-10603-2

I. L… II. 黄… III. Linux 操作系统 IV. TP316.89

中国版本图书馆 CIP 数据核字（2002）第 050275 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：胡毓坚　　责任编辑：汪汉友

责任印制：付方敏

北京铭成印刷有限公司印刷·新华书店北京发行所发行

2002 年 8 月第 1 版·第 1 次印刷

1000mm×1400mm·B5·13.125 印张·510 千字

0001-5000 册

定价：39.00 元

前　言

Linus Torvalds 先生开创了 Linux 操作系统的历史。虽然他在因特网上最早发布的版本已经受到比较严格的许可证的保护，但他还是很快转向通用公共许可证 GPL。这一步对 Linux 发展十分重要。它在早期赢得了许多专业人员的支持，这些人将 GNU 项目的许多成果移植到 Linux 系统上来；后来它赢得了许多公司的支持，其中包括提供技术支持，开发 Linux 的应用软件，他们并将 Linux 系统的应用推向各个方面；近期 Linux 赢得了大型数据库软件公司的支持，从而为 Linux 进入大型企业网的应用领域奠定了基础。

Linux 内核的功能以及它和 GPL 的结合，使许多软件开发人员可以参加到内核的开发工作，并将 GNU 项目的 C 库、gcc、Emacs、bash 等很快移植到 Linux 内核上来。Linux 项目一开始就和 GNU 项目紧密结合在一起。Linux 操作系统的另一些重要组成部分则来自加利福尼亚大学 Berkeley 分校的 BSD UNIX 和麻省理工学院的 X Window 系统项目。正是 Linux 内核与 GNU 项目、BSD UNIX 以及 MIT 的 X11 的结合，才使整个 Linux 操作系统得以很快形成，而且建立在稳固的基础上。

由于可以得到 Linux 的源代码，用户可以按照自己的意愿来对其进行改造，以满足应用方面的特殊需要。使用 Linux 是安全可靠的，因为用户可以自己用源代码来生成可执行程序，而不必担心软件中的陷阱。这对于安全性方面要求比较高的特殊用户来说尤为重要。使用 Linux 的另外一个好处是不受商品化的制约，不会跟着软件公司产品的不断升级而造成投资上的不断增加。

如何利用 Linux 的源代码公开的优势，在 Linux 上进行自己的开发的问题就需要您学习 Linux 应用开发方面的知识。与其姊妹篇《Linux 应用开发基础》一书相比，本书主要讲述 Linux 应用开发中的高级部分。

本书首先在《Linux 应用开发基础》的基础上介绍了 Linux 网络开发方面中比较高级的知识；随后介绍了 Linux 设备驱动程序方面的知识；然后又介绍了 Linux 数据库开发方面的基础知识，主要包括 MySQL 语言的基础知识、MySQL 数据库和 MySQL 的应用开发；最后介绍了 GTK 开发工具及编程实例、PHP 开发等内容。

本书除黄超外，陈华立、李清、高鹏程、徐光捷、高强、何百磊也参加了编写工作。由于编者水平有限，错误在所难免，希望读者能给予批评指正。

作　者

目　录

第1章

深入网络开发

本章要点:

- 监听连接
- 调用 Socket
- 通过 Socket 交谈
- IP 地址及其处理

本章深入地讲述了网络开发方面的知识。希望通过本章的学习，对网络开发有一个深入的了解。由于 socket 使用的广泛性，本章将主要介绍 socket 编程方面的知识。

1.1 监听连接

一个人想收到别人打给他的电话，首先要拥有一部电话机。同样的道理，在网络中，必须先建立 socket 对象才能监听线路。通过 socket 函数可以建立所需的 socket 对象，调用 socket 函数需要提供 3 个参数。

1）用来设定 socket 对象的形式的参数。主要有两个比较重要的选项，分别为 AF_UNIX 和 AF_INET。AF_UNIX 的格式形如 UNIX 路径名，这种形式对于在同一台机器上的 IPC 很有用；而 AF_INET 使用形如 192.9.200.10 的 IP 地址格式。将 IP 地址同端口号相结合，则可以在每台机器上创建多个 AF_INET。

2）设定 socket 对象类型的参数。socket 对象有多种类型，因此在建立 socket 对象时还需要注明所建立的 socket 对象的类型。主要有两个重要的类型，即 SOCK_STREAM 和 SOCK_DGRAM。SOCK_STREAM 表明数据以字符流方式传输，socket 对象能把所有连接请求组成一个队列，在前面的连接处理完之后才能接着处理排在后面的连接；而 SOCK_DGRAM 则表明数据将采用数据报的形式。

3）第三个参数通常设为 0 即可。

在建立 socket 对象后，就需要提供 socket 的监听地址了，这好比需要一个电话号码来接电话一样，它可通过 bind 函数进行处理。同时还有另外一个重要的函数是 listen()，它用来设置最大允许的请求数（一般为 5 个）。

下面的例子将说明如何利用 socket、bind 和 listen 函数建立连接并接受数据。

```
int establish (unsigned short portnum)   /* 建立 socket 的代码*/
{
    char myname[MAXHOSTNAME+1];
    int s;
    struct sockaddr_in sa;
    struct hostent *hp;
    memset (&sa, 0, sizeof (struct sockaddr_in)); /*清除地址结构内容 */
    gethostname (myname, MAXHOSTNAME); /* 得到主机名*/
    hp= gethostbyname (myname);   /* 得到地址信息 */
    if (hp == NULL)   /* 如果地址不存在*/
    return (-1);
    sa.sin_family= hp->h_addrtype;   /* 主机地址 */
    sa.sin_port= htons (portnum); /* 端口号 */
```

```
    if ((s= socket (AF_INET, SOCK_STREAM, 0)) < 0) return (-1);
    /* 如果创建 socket 不成功，返回-1 */
    if (bind (s, &sa, sizeof (struct sockaddr_in)) < 0)
    /* 把地址捆绑到 socket 对象上 */
    {
        close (s);
        return (-1);
    }
    listen (s, 3); /* 设置请求队列中最大请求数为 3*/
    return (s);
}
```

在建立完 socket 后，需要等待对该 socket 对象的调用。该功能通过 accept 函数实现。调用 accept 函数如同在电话铃响后提起电话一样，该函数返回一个新的到调用方 socket 对象的连接。下面的代码行就是这一过程的演示。

```
int get_connection (int s) /* establish()创建的 socket 等待连接请求 */

    int t; /*所连接的 socket 对象*/
    if ((t = accept (s, NULL, NULL)) < 0) return (-1);
    return (t); /*如果存在连接，就接受连接，并且返回所连接的 socket 对象*/

}
```

和电话不同的是，在处理已有的连接时还可以接受新的连接调用。通常调用 fork 函数创建一个新的进程来处理新的连接请求。下面的代码演示了如何使用 establish 和 get_connection 函数处理多个连接。

```
#include <errno.h>      /* 引入所需库文件*/
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORTNUM 50000   /* 随机的端口号*/
void fireman (void);
void do_something (int);
main ()
{
    int s,  t;
```

```
    if ( s= establish (PORTNUM)) < 0)
    {
        /* 建立 socket */
        perror ( " establish" );
        exit (1);
    }
    signal (SIGCHLD, fireman); /* 使用信号机制消除僵尸进程 */
    for (;;) /* 循环等待呼叫 */
    {
        ……
        if ((t= get_connection (s)) < 0) /* 建立一个连接 */
        {
            if (errno == EINTR)  /* accept()可能遇到 EINTR*/
            continue;  /* 重试 */
            perror ( " accept" ); /* 错误处理 */
            exit (1);
        }
        switch (fork ()) /* 处理新的连接 */
        {
        case -1 : /* 没有创建成功的分支 */
            perror ( " fork" );
            close (s);
            close (t);
            exit (1);
        case 0 :  /* 子进程 */
            close (s);
            do_something (t);
            exit (0);
        default : /* 父进程 */
            close (t); /* 处理另一个连接 */
            continue;
        }
    }
}
void fireman (void)
{
    while (waitpid (-1, NULL, WNOHANG) > 0);
}
void do_something (int s) /*处理 socket 的函数，在获得连接之后被调用*/
{
    /*通过 socket 做需要的事情*/
}
```

1.2　调用 socket

前面一节介绍了如何建立 socket 对象，本节将介绍如何调用 socket 对象并建立连接。下面是调用 socket 对象的 call_socket 函数，如果调用 socket 成功，call_socket 函数将返回 socket 对象。

```
int call_socket (char *hostname, unsigned short portnum)
{
    struct sockaddr_in sa;
    struct hostent *hp;
    int a, s;
    if (hp= gethostbyname (hostname)) == NULL)
    {
        errno= ECONNREFUSED;
        return (-1);
    }
    memset (&sa, 0, sizeof (sa));
    memcpy ((char *) &sa.sin_addr, hp->h_addr, hp->h_length);
    /*设置地址*/
    sa.sin_family= hp->h_addrtype;
    sa.sin_port= htons ((u_short) portnum);
    if ((s= socket (hp->h_addrtype, SOCK_STREAM, 0)) < 0) return (-1);
    /*如果不能获得 socket 对象，返回-1*/
    if (connect (s, &sa, sizeof sa) < 0) /*如果连接不成功*/
    {
        close (s);
        return (-1);
    }
    return (s);
}
```

1.3　通过 socket 交谈

在连接建立好以后就该传输数据了。数据传输主要用到 read 和 write 函数。socket 对象的读写操作与一般文件的读写操作是一样的，只是它不能一下子得到所要数目的数据，必须一直循环下去，直到需要的数据全部到来。下面是一个简单的例子，演示了如何将一定数目的数据读到缓存区。

```
int read_data (int s, char * buf,int n);
/* s 为连接 socket; buf 为指向 buffer 的指针; n 为想要得到的字符的数目*/
{
```

```
    int bcount; /*计算读取的字节数*/
    int br; /* 一次读取的字节数 */
    bcount= 0;
    br= 0;
    while (bcount < n)
    {
        /*循环至读入的数据充满 buffer*/
        if ((br= read (s, buf, n-bcount)) > 0)
        {
            bcount += br; /* 增加字节计数器 */
            buf += br; /* 移动缓冲区指针，进行下一次读取 */
        }
        else if (br < 0) return (-1); /*向调用者返回错误*/
    }
    return (bcount);
}
```

和通过电话和某人交谈一样，通话完毕之后应挂断电话，在通过 socket 读写数据以后也应关闭连接。通常调用 close 函数来分别关闭连接各方的 socket 连接。如果一边的 socket 已经关闭，而另外一端却还在向该 socket 发送数据，发送数据方将返回错误消息。

1.4 IP 地址及其处理

现在有很多的函数能方便地操作 IP 地址，而不再需要手工计算处理，也没有必要用操作符“<<”来操作长整型地址。下面分别介绍这些专用函数。

1. inet_addr 函数

首先，假设定义了数据结构 struct sockaddr_in ina，并想将 IP 地址“132.241.5.10”保存到其中。要用的函数是 inet_addr，该函数把由小数点分开的十进制的 IP 地址转换到 unsigned long 类型。这个工作可以用以下代码。

```
ina.sin_addr.s_addr = inet_addr ("132.241.5.10");
```

> 注意：
>
> inet_addr 函数返回的地址已经按照网络字节顺序排列，没有必要再去调用 htonl 函数。上面的代码由于没有错误检查，不是很安全。inet_addr 函数在发生错误的时候返回-1。

如果有一个数据结构 struct in_addr，那么该如何按照由小数点分开的数字格式打印呢？这时，要用到函数 inet_ntoa 函数，“ntoa”的意思是“network to ascii”，

它将网络 IP 地址转换为点分的格式。

```
printf ("%s", inet_ntoa (ina.sin_addr));
```

该函数将打印 IP 地址。

> 注意：
>
> 函数 inet_ntoa 的参数是 struct in_addr，而不是 long。同时要注意它返回的是一个指向字符的指针。inet_ntoa 内部的指针静态地储存字符数组，因此每次调用 inet_ntoa 函数时，它将覆盖以前的内容。见下例。

```
char *a1, *a2;
a1 = inet_ntoa (ina1.sin_addr);   /* 第 1 个地址是 198.92.129.1 */
a2 = inet_ntoa (ina2.sin_addr);   /* 第 2 个地址是 132.241.5.10 */
printf ("address 1:  %s\n", a1);
printf ("address 2:  %s\n", a2);
```

运行结果是：

```
address 1:  132.241.5.10
address 2:  132.241.5.10
```

如果你想保存地址，那么可用函数 strcpy()将地址保存到另外的字符数组中。

2. socket 函数

该函数会得到文件描述符，下面是它们的详细定义。

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
```

其中 domain 应该设置成 "AF_INET"，就像上面的数据结构 struct sockaddr_in 中一样；参数 type 通知内核 socket 是 SOCK_STREAM 类型是 SOCK_DGRAM 类型；protocol 设置为“0”。

> 注意：
>
> domain、type 参数有很多种选项，不可能一一列出了，可以参考手册中关于 socket 函数的说明。当然，还有一个更好的方式去得到 protocol 参数，就是参考 getprotobyname 函数的手册。

socket 函数只能返回以后在系统调用中可能用到的 socket 对象描述符，或者在错误的时候返回-1。全局变量 errno 中储存错误值，详情请参考 perror 函数的相关手册。

3. bind 函数

一旦得到 socket 对象，就有必要将 socket 和机器上的相应的端口关联起来。如果想用 listen 函数来监听一定端口的数据，这是必要的一步。例如连接 MUD

要使用命令“telnet x.y.z 6969”，其中的6969就是所用的端口。如果只想调用connect函数，那么这个步骤没有多大必要。下面是系统调用 bind 函数的大致方法。

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int sockfd, struct sockaddr *my_addr, int addrlen);
```

其中 sockfd 是调用 socket 对象返回的文件描述符；my_addr 是指向数据结构 struct sockaddr 的指针，它用来保存地址（即端口和 IP 地址）信息；addrlen 设置为 sizeof（struct sockaddr）。再看看下面的例子。

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#define MYPORT 3490
main ()
{
    int sockfd;
    struct sockaddr_in my_addr;
    sockfd = socket (AF_INET, SOCK_STREAM, 0); /* 进行错误检查 */
    my_addr.sin_family = AF_INET; /* 主机字节次序 */
    my_addr.sin_port = htons (MYPORT); /* 短的、网络字节次序 */
    my_addr.sin_addr.s_addr = inet_addr ("132.241.5.10");
    bzero (& (my_addr.sin_zero), 8);
    /* 把结构体其他部分置空，不要忘记 bind()的错误检测 */
    bind (sockfd, (struct sockaddr *) &my_addr, sizeof (struct
sockaddr));
}
```

其中 my_addr.sin_port 和 my_addr.sin_addr.s_addr 都是网络字节顺序。另外要注意到的是因系统的不同，包含的头文件也不尽相同，请查阅自己的手册。在处理自己的 IP 地址和端口的时候，有些工作是可以自动处理的。例如：

```
my_addr.sin_port = 0; /* 随机选择未使用的端口 */
my_addr.sin_addr.s_addr = INADDR_ANY; /* 使用自己的 IP 地址 */
```

通过将 0 赋给 my_addr.sin_port 告诉 bind 函数自己选择合适的端口。同样，将 my_addr.sin_addr.s_addr 设置为 INADDR_ANY，告诉它自动填上它所运行的机器的 IP 地址。这里没有将 INADDR_ANY 转换为网络字节顺序，这是因为 INADDR_ANY·实际上就是 0。不过为了安全，也可以采用下面的代码。

```
my_addr.sin_port = htons (0); /* 随机选择一个没有使用的端口*/
my_addr.sin_addr.s_addr = htonl (INADDR_ANY); /*使用我的 IP 地址*/
```

上面的代码可以随便移植。bind 函数在错误时依然是返回-1，并且设置全局变量 errno。在调用 bind 函数的时候，还需要注意不要采用小于 1024 的端口号。所有小于 1024 的端口号都被系统保留使用。可以选择从 1024 到 65535 的没有被其他程序占用的端口。

另外如果使用 connect 函数来和远程机器通讯，不必考虑本地端口号，就像在使用 telnet 的时候，只要简单地调用 connect 函数即可，它会检查 socket 是否绑定，如果没有，它会自动绑定一个没有使用的本地端口。

4．connect 函数

现在假设要通过端口 23（标准 telnet 端口）用 telnet 方式连接到某个服务器“132.241.5.10”，这就需要用到函数 connect 了，connect 函数定义如下。

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (int sockfd,  struct sockaddr *serv_addr,  int addrlen);
```

其中 sockfd 是系统调用 socket 函数返回的 socket 文件描述符；serv_addr 是保存着目的地端口和 IP 地址的数据结构 struct sockaddr；addrlen 设置为 sizeof（struct sockaddr）。

现在来看下面的例子。

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#define DEST_IP  " 132.241.5.10"
#define DEST_PORT 23
main ()
{
int sockfd;
struct sockaddr_in DEST_addr; /* 用以保存目的地址 */
sockfd = socket (AF_INET,  SOCK_STREAM,  0); /* 进行错误检查 */
DEST_addr.sin_family = AF_INET; /* 主机字节顺序 */
DEST_addr.sin_port = htons (DEST_PORT); /* 短的、网络字节顺序 */
DEST_addr.sin_addr.s_addr = inet_addr (DEST_IP);
bzero (& (DEST_addr.sin_zero), 8); /* 清空结构体其他部分*/
connect ( sockfd, ( struct  sockaddr  *) &DEST_addr, sizeof ( struct
sockaddr));
}
```

应该每次都检查 connect 函数的返回值，它在错误的时候返回-1，并设置全局变量 errno。同时可能看到，这里并没有调用 bind 函数。这里不需要关心本地的端口号，而只需关心是否在连接。内核将自动选择合适的端口号，而所连接的