第21章 Linux系统网络编程

本章介绍Linux系统网络编程的内容，如套接口的概念与使用、网络编程的结构等。

21.1 什么是套接口

简单地说，套接口就是一种使用UNIX系统中的文件描述符和系统进程通信的一种方法。

因为在UNIX系统中，所有的I/O操作都是通过读写文件描述符而产生的。文件描述符就是一个和打开的文件相关连的整数。但文件可以是一个网络连接、一个 FIFO、一个管道、一个终端、一个真正存储在磁盘上的文件或者 UNIX系统中的任何其他的东西。所以，如果你希望通过Internet和其他的程序进行通信，你只有通过文件描述符。

使用系统调用socket()，你可以得到socket()描述符。然后你可以使用send() 和 recv()调用而与其他的程序通信。你也可以使用一般的文件操作来调用 read() 和 write()而与其他的程序进行通信，但send() 和 recv()调用可以提供一种更好的数据通信的控制手段。下面我们讨论 Internet套接口的使用方法。

21.2 两种类型的Internet套接口

有两种最常用的 Internet 套接口，“ 数据流套接口 ”和“ 数据报套接口 ”，以后我们用“ SOCK_STREAM ” 和 “ SOCK_DGRAM ”分别代表上面两种套接口。数据报套接口有时也叫做“ 无连接的套接口 ”。

数据流套接口是可靠的双向连接的通信数据流。如果你在套接口中以“ 1, 2 ”的顺序放入两个数据，它们在另一端也会以“ 1, 2 ”的顺序到达。它们也可以被认为是无错误的传输。

经常使用的telnet应用程序就是使用数据流套接口的一个例子。使用 HTTP的WWW浏览器也使用数据流套接口来读取网页。事实上，如果你使用 telnet 登录到一个WWW站点的80端口，然后键入“ GET 网页名 ”，你将可以得到这个HTML页。数据流套接口使用TCP得到这种高质量的数据传输。数据报套接口使用UDP，所以数据报的顺序是没有保障的。数据报是按一种应答的方式进行数据传输的。

21.3 网络协议分层

由于网络中的协议是分层的，所以上层的协议是依赖于下一层所提供的服务的。也就是说，你可以在不同的物理网络中使用同样的套接口程序，因为下层的协议对你来说是透明的。UNIX系统中的网络协议是这样分层的：

- 应用层 (telnet、ftp等)。
- 主机到主机传输层 (TCP、UDP)。
- Internet层 (IP和路由)。
- 网络访问层(网络、数据链路和物理层)。

21.4 数据结构

下面我们要讨论使用套接口编写程序可能要用到的数据结构。

首先是套接口描述符。一个套接口描述符只是一个整型的数值： int。

第一个数据结构是struct sockaddr，这个数据结构中保存着套接口的地址信息。

```
struct sockaddr {
    unsigned short    sa_family;    /* address family, AF_xxx       */
    char              sa_data[14];  /* 14 bytes of protocol address */
};
```

sa_family 中可以是其他的很多值，但在这里我们把它赋值为“ AF_INET ”。sa_data包括一个目的地址和一个端口地址。

你也可以使用另一个数据结构sockaddr_in，如下所示：

```
struct sockaddr_in {
    short int          sin_family;  /* Address family         */
    unsigned short int sin_port;    /* Port number            */
    struct in_addr     sin_addr;    /* Internet address       */
    unsigned char      sin_zero[8]; /* Same size as struct sockaddr */
};
```

这个数据结构使得使用其中的各个元素更为方便。要注意的是 sin_zero应该使用bzero() 或者 memset()而设置为全0。另外，一个指向sockaddr_in数据结构的指针可以投射到一个指向数据结构sockaddr的指针，反之亦然。

21.5 IP地址和如何使用IP地址

有一系列的程序可以使你处理IP地址。

首先，你可以使用inet_addr()程序把诸如“ 132.241.5.10 ”形式的IP地址转化为无符号的整型数。

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

如果出错， inet_addr()程序将返回-1。

也可以调用inet_ntoa()把地址转换成数字和句点的形式：

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

这将会打印出IP地址。它返回的是一个指向字符串的指针。

21.5.1 socket()

我们使用系统调用socket()来获得文件描述符：

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

第一个参数domain设置为“ AF_INET ”。第二个参数是套接口的类型：SOCK_STREAM 或 SOCK_DGRAM。第三个参数设置为0。

系统调用socket()只返回一个套接口描述符，如果出错，则返回 -1。

21.5.2 bind()

一旦你有了一个套接口以后，下一步就是把套接口绑定到本地计算机的某一个端口上。但如果你只想使用connect()则无此必要。

下面是系统调用bind()的使用方法：

```
#include <sys/types.h>
```

```
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

第一个参数sockfd 是由socket()调用返回的套接口文件描述符。第二个参数my_addr 是指向数据结构sockaddr的指针。数据结构 sockaddr中包括了关于你的地址、端口和 IP地址的信息。第三个参数addrlen可以设置成sizeof(struct sockaddr)。

下面是一个例子：

```
#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); /* do some error checking! */

    my_addr.sin_family = AF_INET;     /* host byte order */
    my_addr.sin_port = htons(MYPORT); /* short, network byte order */
    my_addr.sin_addr.s_addr = inet_addr("132.241.5.10");
    bzero(&(my_addr.sin_zero), 8);    /* zero the rest of the struct */

    /* don't forget your error checking for bind(): */
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
    .
    .
    .
```

如果出错，bind() 也返回-1。

如果你使用connect()系统调用，那么你不必知道你使用的端口号。当你调用 connect()时，它检查套接口是否已经绑定，如果没有，它将会分配一个空闲的端口。

21.5.3 connect()

系统调用connect()的用法如下：

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

第一个参数还是套接口文件描述符，它是由系统调用 socket()返回的。第二个参数是serv_addr是指向数据结构sockaddr的指针，其中包括目的端口和IP地址。第三个参数可以使用sizeof(struct sockaddr)而获得。下面是一个例子：

```
#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;   /* will hold the destination addr */

        sockfd = socket(AF_INET, SOCK_STREAM, 0); /* do some error checking! */

        dest_addr.sin_family = AF_INET;         /* host byte order */
        dest_addr.sin_port = htons(DEST_PORT); /* short, network byte order */
        dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
        bzero(&(dest_addr.sin_zero), 8);        /* zero the rest of the struct */

        /* don't forget to error check the connect()! */
        connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
        .
        .
        .
```

同样，如果出错， connect()将会返回-1。

21.5.4 listen()

如果你希望不连接到远程的主机，也就是说你希望等待一个进入的连接请求，然后再处理它们。这样，你通过首先调用listen()，然后再调用accept()来实现。

系统调用listen()的形式如下：

```
int listen(int sockfd, int backlog);
```

第一个参数是系统调用 socket()返回的套接口文件描述符。第二个参数是进入队列中允许的连接的个数。进入的连接请求在使用系统调用 accept()应答之前要在进入队列中等待。这个值是队列中最多可以拥有的请求的个数。大多数系统的缺省设置为 20。你可以设置为5或者10。

当出错时，listen()将会返回-1值。

当然，在使用系统调用 listen()之前，我们需要调用 bind()绑定到需要的端口，否则系统内核将会让我们监听一个随机的端口。所以，如果你希望监听一个端口，下面是应该使用的系统调用的顺序：

```
socket();
bind();
listen();
/* accept() goes here */
```

21.5.5 accept()

系统调用 accept()比较起来有点复杂。在远程的主机可能试图使用 connect()连接你使用 listen()正在监听的端口。但此连接将会在队列中等待，直到使用 accept()处理它。调用accept()之后，将会返回一个全新的套接口文件描述符来处理这个单个的连接。这样，对于同一个连接来说，你就有了两个文件描述符。原先的一个文件描述符正在监听你指定的端口，新的文件描述符可以用来调用send()和recv()。

调用的例子如下：

```
#include <sys/socket.h>

int accept(int sockfd, void *addr, int *addrlen);
```

第一个参数是正在监听端口的套接口文件描述符。第二个参数 addr是指向本地的数据结构sockaddr_in的指针。调用connect()中的信息将存储在这里。通过它你可以了解哪个主机在哪个端口呼叫你。第三个参数同样可以使用sizeof(struct sockaddr_in)来获得。

如果出错，accept()也将返回-1。下面是一个简单的例子：

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>

#define MYPORT 3490    /* the port users will be connecting to */

#define BACKLOG 10     /* how many pending connections queue will hold */

main()
{
        int sockfd, new_fd;  /* listen on sock_fd, new connection on new_fd */
        struct sockaddr_in my_addr;    /* my address information */
        struct sockaddr_in their_addr; /* connector's address information */
        int sin_size;

        sockfd = socket(AF_INET, SOCK_STREAM, 0); /* do some error checking! */

        my_addr.sin_family = AF_INET;              /* host byte order */
        my_addr.sin_port = htons(MYPORT);          /* short, network byte order */
        my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
        bzero(&(my_addr.sin_zero), 8);             /* zero the rest of the struct */

        /* don't forget your error checking for these calls: */
        bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

        listen(sockfd, BACKLOG);

 sin_size = sizeof(struct sockaddr_in);
 new_fd = accept(sockfd, &their_addr, &sin_size);
.
.
.
```

下面，我们将可以使用新创建的套接口文件描述符 new_fd来调用send()和recv()。

21.5.6 send() 和 recv()

系统调用send()的用法如下：

```
int send(int sockfd, const void *msg, int len, int flags);
```

第一个参数是你希望给发送数据的套接口文件描述符。它可以是你通过 socket()系统调用返回的，也可以是通过accept()系统调用得到的。第二个参数是指向你希望发送的数据的指针。第三个参数是数据的字节长度。第四个参数标志设置为 0。

下面是一个简单的例子：

```
char *msg = "Beej was here!";
int len, bytes_sent;
.
.
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.
.
.
```

系统调用send()返回实际发送的字节数，这可能比你实际想要发送的字节数少。如果返回的字节数比要发送的字节数少，你在以后必须发送剩下的数据。当 send()出错时，将返回 -1。系统调用recv()的使用方法和send()类似：

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

第一个参数是要读取的套接口文件描述符。第二个参数是保存读入信息的地址。第三个参数是缓冲区的最大长度。第四个参数设置为0。

系统调用recv()返回实际读取到缓冲区的字节数，如果出错则返回 -1。

这样使用上面的系统调用，你可以通过数据流套接口来发送和接受信息。

21.5.7 sendto() 和 recvfrom()

因为数据报套接口并不连接到远程的主机上，所以在发送数据包之前，我们必须首先给出目的地址，请看：

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
const struct sockaddr *to, int tolen);
```

除了两个参数以外，其他的参数和系统调用 send()时相同。参数to是指向包含目的IP地址和端口号的数据结构sockaddr的指针。参数tolen可以设置为sizeof(struct sockaddr)。

系统调用sendto()返回实际发送的字节数，如果出错则返回 -1。

系统调用recvfrom()的使用方法也和recv()的十分近似：

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags
        struct sockaddr *from, int *fromlen);
```

参数from是指向本地计算机中包含源 IP地址和端口号的数据结构 sockaddr的指针。参数fromlen设置为sizeof(struct sockaddr)。

系统调用recvfrom()返回接收到的字节数，如果出错则返回 -1。

21.5.8 close() 和 shutdown()

你可以使用close()调用关闭连接的套接口文件描述符：

```
close(sockfd);
```

这样就不能再对此套接口做任何的读写操作了。

使用系统调用shutdown()，可有更多的控制权。它允许你在某一个方向切断通信，或者切断双方的通信：

```
int shutdown(int sockfd, int how);
```

第一个参数是你希望切断通信的套接口文件描述符。第二个参数 how值如下：

0 —— Further receives are disallowed

1 —— Further sends are disallowed

2 —— Further sends and receives are disallowed (like close())

shutdown() 如果成功则返回0，如果失败则返回-1。

21.5.9 getpeername()

这个系统的调用十分简单。它将告诉你是谁在连接的另一端：

```
#include <sys/socket.h>

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

第一个参数是连接的数据流套接口文件描述符。第二个参数是指向包含另一端的信息的数据结构sockaddr的指针。第三个参数可以设置为sizeof(struct sockaddr)。

如果出错，系统调用将返回-1。

一旦你获得了它们的地址，你可以使用inet_ntoa() 或者 gethostbyaddr()来得到更多的信息。

21.5.10 gethostname()

系统调用gethostname()比系统调用getpeername()还简单。它返回程序正在运行的计算机的名字。系统调用gethostbyname()可以使用这个名字来决定你的机器的IP地址。

下面是一个例子：

```
#include <unistd.h>

int gethostname(char *hostname, size_t size);
```

如果成功，gethostname将返回0。如果失败，它将返回-1。

21.6 DNS

DNS 代表“ Domain Name Service ”，即域名服务器。它可以把域名翻译成相应的 IP地址。你可以使用此IP地址调用bind()、connect()、sendto()或者用于其他的地方。

系统调用gethostbyname()可以完成这个函数：

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

它返回一个指向数据结构hostent的指针，数据结构hostent如下：

```
struct hostent {
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
};
#define h_addr h_addr_list[0]
```

h_name —— 主机的正式名称。

h_aliases —— 主机的别名。

h_addrtype —— 将要返回的地址的类型，一般是 AF_INET。

h_length —— 地址的字节长度。

h_addr_list —— 主机的网络地址。

h_addr ——h_addr_list中的第一个地址。

系统调用gethostbyname()返回一个指向填充好的数据结构 hostent的指针。当发生错误时，则返回一个NULL指针。下面是一个实际例子：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
    struct hostent *h;

    if (argc != 2) {  /* error check the command line */
        fprintf(stderr,"usage: getip address\n");
        exit(1);
    }

    if ((h=gethostbyname(argv[1])) == NULL) {  /* get the host info */
        herror("gethostbyname");
        exit(1);
    }

    printf("Host name  : %s\n", h->h_name);
    printf("IP Address : %s\n",inet_ntoa(*((struct in_addr *)h->h_addr)));

    return 0;
}
```

在使用gethostbyname()时，你不能使用perror()来打印错误信息。你应该使用的是系统调用herror()。

21.7 客户机/服务器模式

在网络上大部分的通信都是在客户机/服务器模式下进行的。例如telnet。当你使用telnet连接到远程主机的端口 23时，主机上的一个叫做 telnetd的程序就开始运行。它处理所有进入的telnet连接，为你设置登录提示符等。

应当注意的是客户机/服务器模式可以使用SOCK_STREAM、SOCK_DGRAM或者任何其他的方式。例如telnet/telnetd、ftp/ftpd和 bootp/bootpd。每当你使用ftp时，远程计算机都在运行一个ftpd为你服务。

一般情况下，一台机器上只有一个服务器程序，它通过使用 fork()来处理多个客户端程序的请求。最基本的处理方法是：服务器等待连接，使用 accept()接受连接，调用fork()生成一个子进程处理连接。

21.8 简单的数据流服务器程序

此服务器程序所作的事情就是通过一个数据流连接发送字符串“ Hello, World!\n ”。你可以在一个窗口上运行此程序，然后在另一个窗口使用 telnet：

```
$ telnet remotehostname 3490
```

其中，remotehostname是你运行的机器名。下面是此程序的代码：

```
#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    /* the port users will be connecting to */

#define BACKLOG 10     /* how many pending connections queue will hold */

main()
{
    int sockfd, new_fd;  /* listen on sock_fd, new connection on new_fd */
    struct sockaddr_in my_addr;    /* my address information */
    struct sockaddr_in their_addr; /* connector's address information */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;         /* host byte order */
    my_addr.sin_port = htons(MYPORT);     /* short, network byte order */
    my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
    bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the struct */

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) \
                                                  == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

while(1) {  /* main accept() loop */
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, \
                                        &sin_size)) == -1) {
            perror("accept");
            continue;
        }
```

```
            printf("server: got connection from %s\n", \
                                    inet_ntoa(their_addr.sin_addr));
            if (!fork()) { /* this is the child process */
                if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
                    perror("send");
                close(new_fd);
                exit(0);
            }
            close(new_fd);  /* parent doesn't need this */

            while(waitpid(-1,NULL,WNOHANG) > 0); /* clean up child processes */
        }
    }
```

你也可以使用下面的客户机程序从服务器上得到字符串。

21.9 简单的数据流客户机程序

客户机所做的是连接到你在命令行中指定的主机的 3490端口。它读取服务器发送的字符串。

下面是客户机程序的代码：

```
#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    /* the port client will be connecting to */

#define MAXDATASIZE 100 /* max number of bytes we can get at once */

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr; /* connector's address information */

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  /* get the host info */
        herror("gethostbyname");
        exit(1);
    }
```

```
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;      /* host byte order */
    their_addr.sin_port = htons(PORT);    /* short, network byte order */
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    bzero(&(their_addr.sin_zero), 8);     /* zero the rest of the struct */

    if (connect(sockfd, (struct sockaddr *)&their_addr, \
                                sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    if ((numbytes=recv(sockfd, buf, MAXDATASIZE, 0)) == -1) {
        perror("recv");
        exit(1);
    }

    buf[numbytes] = '\0';

    printf("Received: %s",buf);

    close(sockfd);

    return 0;
}
```

如果你在运行服务器程序之前运行客户机程序，则将会得到一个“ Connection refused ”的信息。

21.10 数据报套接口

程序listener在机器中等待端口4950到来的数据包。程序talker向指定的机器的4950端口发送数据包。

下面是listener.c的代码：

```
#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 4950    /* the port users will be sending to */

#define MAXBUFLEN 100
```

```
main()
{
        int sockfd;
        struct sockaddr_in my_addr;    /* my address information */
        struct sockaddr_in their_addr; /* connector's address information */
        int addr_len, numbytes;
        char buf[MAXBUFLEN];

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

        my_addr.sin_family = AF_INET;         /* host byte order */
        my_addr.sin_port = htons(MYPORT);     /* short, network byte order */
        my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
        bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the struct */

        if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) \
                                                    == -1) {
            perror("bind");
            exit(1);
        }

        addr_len = sizeof(struct sockaddr);
        if ((numbytes=recvfrom(sockfd, buf, MAXBUFLEN, 0, \
                    (struct sockaddr *)&their_addr, &addr_len)) == -1) {
            perror("recvfrom");
            exit(1);
        }

        printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
        printf("packet is %d bytes long\n",numbytes);
        buf[numbytes] = '\0';
        printf("packet contains \"%s\"\n",buf);

        close(sockfd);
}
```

下面是talker.c的代码：

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

#define MYPORT 4950    /* the port users will be sending to */
```

```
int main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in their_addr; /* connector's address information */
    struct hostent *he;
    int numbytes;

    if (argc != 3) {
        fprintf(stderr,"usage: talker hostname message\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  /* get the host info */
        herror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;      /* host byte order */
    their_addr.sin_port = htons(MYPORT);  /* short, network byte order */
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    bzero(&(their_addr.sin_zero), 8);     /* zero the rest of the struct */

    if ((numbytes=sendto(sockfd, argv[2], strlen(argv[2]), 0, \
         (struct sockaddr *)&their_addr, sizeof(struct sockaddr))) == -1) {
        perror("sendto");
        exit(1);
    }

    printf("sent %d bytes to %s\n",numbytes,inet_ntoa(their_addr.sin_addr));

    close(sockfd);

    return 0;
}
```

你可以在一台机器上运行 listener程序，在另一台机器上运行 talker程序，然后观察它们之间的通信。

21.11 阻塞

当使用上面的 listener程序时，此程序在等待直到一个数据包到来。这是因为它调用了 recvform()，如果没有数据，recvform()就一直阻塞，直到有数据到来。

很多函数都有阻塞。系统调用 accept()阻塞，所有的类似 recv*()的函数也可以阻塞。它们之所以可以阻塞是因为系统内核允许它们阻塞。当你第一次创建一个套接口文件描述符时，系

统内核将它设置为可以阻塞。如果你不希望套接口阻塞，你可以使用系统调用 fcntl()：

```
#include <unistd.h>
#include <fcntl.h>
.
.
sockfd = socket(AF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
.
.
```

如果你设置为不阻塞，那么就得频繁地询问套接口以便检查有无信息到来。如果你试图读取一个没有阻塞的套接口，同时它又没有数据，那么你将得到 -1。

询问套接口以检查有无信息得到来可能会占用太多的 CPU时间。另一个可以使用的方法是 select()。

select()用于同步 I/O多路复用。这个系统调用十分有用。考虑一下下面的情况：你是一个服务器，你希望监听进入的连接，同时还一直从已有的连接中读取信息。

也许你认为可以使用一个 accept()调用和几个recv()调用。但如果调用 accept()阻塞了怎么办？如果在这时你希望调用recv()接受数据呢？

系统调用select()使得你可以同时监视几个套接口。它可以告诉你哪一个套接口已经准备好了以供读取，哪一个套接口已经可以写入。

下面是select()的用法：

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

此函数监视几个文件描述符，特别是 readfds、writefds和 exceptfds。如果你希望检查是否可以从标准输入中和一些其他的套接口文件描述符 sockfd中读取数据，只需把文件描述符 0和sockfd添加到readfds中。参数numfds应该设置为最高的文件描述符的值加1。

当select()返回时，readfds将会被修改以便反映你选择的那一个文件描述符已经准备好了以供读取。你可以使用FD_ISSET()测试。

FD_ZERO(fd_set *set) —— 清除文件描述符集。

FD_SET(int fd, fd_set *set) —— 把fd 添加到文件描述符集中 。

FD_CLR(int fd, fd_set *set) —— 把fd 从文件描述符中移走 。

FD_ISSET(int fd, fd_set *set) —— 检测fd 是否在文件描述符集中。

数据结构timeval包含下面的字段：

```
struct timeval {
    int tv_sec;     /* seconds */
    int tv_usec;    /* microseconds */
};
```

把tv_sec设置成需要等待的时间秒数， tv_usec设置成需要等待的微秒数。一秒中包括1 000 000μs 。下面的程序等待2.5s：

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

```
#define STDIN 0  /* file descriptor for standard input */

main()
{
        struct timeval tv;
        fd_set readfds;

        tv.tv_sec = 2;
        tv.tv_usec = 500000;

        FD_ZERO(&readfds);
        FD_SET(STDIN, &readfds);

        /* don't care about writefds and exceptfds: */
        select(STDIN+1, &readfds, NULL, NULL, &tv);

        if (FD_ISSET(STDIN, &readfds))
            printf("A key was pressed!\n");
        else
            printf("Timed out.\n");
}
```