学习Linux网络编程（1）（2）（3）（4）

这篇教程是用来介绍在Linux下编写网络程序的.

Linux系统的一个主要特点是他的网络功能非常强大。随着网络的日益普及，基于网络的应用也将越来越多。 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，学习Linux的网络编程，可以让我们真正的体会到网络的魅力。 想成为一位真正的hacker，必须掌握网络编程技术。
现在书店里面已经有了许多关于Linux网络编程方面的书籍，网络上也有了许多关于网络编程方面的教材，大家都可以 去看一看的。在这里我会和大家一起来领会Linux网络编程的奥妙，由于我学习Linux的网络编程也开始不久，所以我下面所说的肯定会有错误的， 还请大家指点出来，在这里我先谢谢大家了。

在这一个章节里面，我会和以前的几个章节不同，在前面我都是概括的说了一下， 从现在开始我会尽可能的详细的说明每一个函数及其用法。好了让我们去领会Linux的伟大的魅力吧！

1. Linux网络知识介绍

1.1 客户端程序和服务端程序

网络程序和普通的程序有一个最大的区别是网络程序是由两个部分组成的--客户端和服务器端.

网络程序是先有服务器程序启动,等待客户端的程序运行并建立连接.一般的来说是服务端的程序 在一个端口上监听,直到有一个客户端的程序发来了请求.

1.2 常用的命令

由于网络程序是有两个部分组成,所以在调试的时候比较麻烦,为此我们有必要知道一些常用的网络命令

netstat

命令netstat是用来显示网络的连接,路由表和接口统计等网络的信息.netstat有许多的选项 我们常用的选项是 -an 用来显示详细的网络状态.至于其它的选项我们可以使用帮助手册获得详细的情况.

telnet

telnet是一个用来远程控制的程序,但是我们完全可以用这个程序来调试我们的服务端程序的. 比如我们的服务器程序在监听8888端口,我们可以用telnet localhost 8888来查看服务端的状况.

1.3 TCP/UDP介绍

TCP(Transfer Control Protocol)传输控制协议是一种面向连接的协议,当我们的网络程序使用 这个协议的时候,网络可以保证我们的客户端和服务端的连接是可靠的,安全的.

UDP(User Datagram Protocol)用户数据报协议是一种非面向连接的协议,这种协议并不能保证我们 的网络程序的连接是可靠的,所以我们现在编写的程序一般是采用TCP协议的.

2. 初等网络函数介绍（TCP）

Linux系统是通过提供套接字(socket)来进行网络编程的.网络程序通过socket和其它几个函数的调用,会返回一个 通讯的文件描述符,我们可以将这个描述符看成普通的文件的描述符来操作,这就是linux的设备无关性的 好处.我们可以通过向描述符读写操作实现网络之间的数据交流.

2.1 socket
int socket(int domain, int type,int protocol)

domain:说明我们网络程序所在的主机采用的通讯协族(AF_UNIX和AF_INET等). AF_UNIX只能够用于单一的Unix系统进程间通信,而AF_INET是针对Internet的,因而可以允许在远程 主机之间通信(当我们 man socket时发现 domain可选项是 PF_*而不是AF_*,因为glibc是posix的实现 所以用PF代替了AF,不过我们都可以使用的).

type:我们网络程序所采用的通讯协议(SOCK_STREAM,SOCK_DGRAM等) SOCK_STREAM表明我们用的是TCP协议,这样会提供按顺序的,可靠,双向,面向连接的比特流. SOCK_DGRAM 表明我们用的是UDP协议,这样只会提供定长的,不可靠,无连接的通信.

protocol:由于我们指定了type,所以这个地方我们一般只要用0来代替就可以了 socket为网络通讯做基本的准备.成功时返回文件描述符,失败时返回-1,看errno可知道出错的详细情况.

2.2 bind
int bind(int sockfd, struct sockaddr *my_addr, int addrlen)

sockfd:是由socket调用返回的文件描述符.

addrlen:是sockaddr结构的长度.

my_addr:是一个指向sockaddr的指针. 在中有 sockaddr的定义

```
struct sockaddr{
unisgned short as_family;
char sa_data[14];
};
```

不过由于系统的兼容性,我们一般不用这个头文件,而使用另外一个结构(struct sockaddr_in) 来代替.在中有sockaddr_in的定义

```
struct sockaddr_in{
unsigned short sin_family;
unsigned short int sin_port;
struct in_addr sin_addr;
unsigned char sin_zero[8];
```

我们主要使用Internet所以sin_family一般为AF_INET,sin_addr设置为INADDR_ANY表示可以 和任何的主机通信,sin_port是我们要监听的端口号.sin_zero[8]是用来填充的. bind将本地的端口同socket返回的文件描述符捆绑在一起.成功是返回0,失败的情况和socket一样

2.3 listen
int listen(int sockfd,int backlog)

sockfd:是bind后的文件描述符.

backlog:设置请求排队的最大长度.当有多个客户端程序和服务端相连时, 使用这个表示可以介绍的排队长

度. listen函数将bind的文件描述符变为监听套接字.返回的情况和bind一样.

2.4 accept

int accept(int sockfd, struct sockaddr *addr,int *addrlen)

sockfd:是listen后的文件描述符.

addr,addrlen是用来给客户端的程序填写的,服务器端只要传递指针就可以了. bind,listen和accept是服务器端用的函数,accept调用时,服务器端的程序会一直阻塞到有一个 客户程序发出了连接. accept成功时返回最后的服务器端的文件描述符,这个时候服务器端可以向该描述符写信息了. 失败时返回-1

2.5 connect

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

sockfd:socket返回的文件描述符.

serv_addr:储存了服务器端的连接信息.其中sin_add是服务端的地址

addrlen:serv_addr的长度

connect函数是客户端用来同服务端连接的.成功时返回0,sockfd是同服务端通讯的文件描述符 失败时返回-1.

2.6 实例

服务器端程序

```
/******* 服务器程序 (server.c) ***********/
#include
#include
#include
#include
#include
#include
#include
#include

int main(int argc, char *argv[])
{
int sockfd,new_fd;
struct sockaddr_in server_addr;
struct sockaddr_in client_addr;
int sin_size,portnumber;
char hello[]="Hello! Are You Fine?\n";
```

```
if(argc!=2)
{
fprintf(stderr,"Usage:%s portnumber\a\n",argv[0]);
exit(1);
}

if((portnumber=atoi(argv[1]))<0)
{
fprintf(stderr,"Usage:%s portnumber\a\n",argv[0]);
exit(1);
}

/* 服务器端开始建立socket描述符 */
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
fprintf(stderr,"Socket error:%s\n\a",strerror(errno));
exit(1);
}

/* 服务器端填充 sockaddr结构 */
bzero(&server_addr,sizeof(struct sockaddr_in));
server_addr.sin_family=AF_INET;
server_addr.sin_addr.s_addr=htonl(INADDR_ANY);
server_addr.sin_port=htons(portnumber);

/* 捆绑sockfd描述符 */
if(bind(sockfd,(struct sockaddr *)(&server_addr),sizeof(struct sockaddr))==-1)
{
fprintf(stderr,"Bind error:%s\n\a",strerror(errno));
exit(1);
}

/* 监听sockfd描述符 */
if(listen(sockfd,5)==-1)
{
fprintf(stderr,"Listen error:%s\n\a",strerror(errno));
exit(1);
}

while(1)
{
/* 服务器阻塞,直到客户程序建立连接 */
sin_size=sizeof(struct sockaddr_in);
if((new_fd=accept(sockfd,(struct sockaddr *)(&client_addr),&sin_size))==-1)
{
fprintf(stderr,"Accept error:%s\n\a",strerror(errno));
exit(1);
```

```
}

fprintf(stderr,"Server get connection from %s\n",
inet_ntoa(client_addr.sin_addr));
if(write(new_fd,hello,strlen(hello))==-1)
{
fprintf(stderr,"Write Error:%s\n",strerror(errno));
exit(1);
}
/* 这个通讯已经结束 */
close(new_fd);
/* 循环下一个 */
}
close(sockfd);
exit(0);
}
```

客户端程序

```
/******* 客户端程序 client.c ***********/
#include
#include
#include
#include
#include
#include
#include
#include

int main(int argc, char *argv[])
{
int sockfd;
char buffer[1024];
struct sockaddr_in server_addr;
struct hostent *host;
int portnumber,nbytes;

if(argc!=3)
{
fprintf(stderr,"Usage:%s hostname portnumber\a\n",argv[0]);
exit(1);
}

if((host=gethostbyname(argv[1]))==NULL)
{
fprintf(stderr,"Gethostname error\n");
exit(1);
```

```
}

if((portnumber=atoi(argv[2]))<0)
{
fprintf(stderr,"Usage:%s hostname portnumber\a\n",argv[0]);
exit(1);
}

/* 客户程序开始建立 sockfd描述符 */
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
fprintf(stderr,"Socket Error:%s\a\n",strerror(errno));
exit(1);
}

/* 客户程序填充服务端的资料 */
bzero(&server_addr,sizeof(server_addr));
server_addr.sin_family=AF_INET;
server_addr.sin_port=htons(portnumber);
server_addr.sin_addr=*((struct in_addr *)host->h_addr);

/* 客户程序发起连接请求 */
if(connect(sockfd,(struct sockaddr *)(&server_addr),sizeof(struct sockaddr))==-1)
{
fprintf(stderr,"Connect Error:%s\a\n",strerror(errno));
exit(1);
}

/* 连接成功了 */
if((nbytes=read(sockfd,buffer,1024))==-1)
{
fprintf(stderr,"Read Error:%s\n",strerror(errno));
exit(1);
}
buffer[nbytes]=\;
printf("I have received:%s\n",buffer);
/* 结束通讯 */
close(sockfd);
exit(0);
}
```

MakeFile
这里我们使用GNU 的make实用程序来编译. 关于make的详细说明见 Make 使用介绍

```
######### Makefile ###########
all:server client
server:server.c
```

```
gcc $^ -o $@
client:client.c
gcc $^ -o $@
```

运行make后会产生两个程序server(服务器端)和client(客户端) 先运行./server portnumber&(portnumber随便取一个大于1204且不在/etc/services中出现的号码 就用8888好了),然后运行 ./client localhost 8888 看看有什么结果. (你也可以用telnet和netstat试一试.) 上面是一个最简单的网络程序,不过是不是也有点烦.上面有许多函数我们还没有解释. 我会在下一章进行的详细的说明.

2.7 总结

总的来说网络程序是由两个部分组成的--客户端和服务器端.它们的建立步骤一般是:

服务器端

socket-->bind-->listen-->accept

客户端

socket-->connect