浙江工业大学

实验报告

课程：嵌入式系统 A

第六次实验

姓　　名 凌智城

学　　号 201806061211

专业班级 通信工程 1803 班

老　　师 黄国兴

学　　院 信息工程学院

提交日期 2021 年 6 月 9 日

实验 10：以太网传输程序编写实验

一、 实验目的

1. 通过实验了解以太网通讯原理和驱动程序开发方法。
2. 通过实验了解 TCP 和 UDP 协议的功能和作用。
3. 通过实验了解基于 TCP/UDP 的 socket 编程。

二、 实验内容

1. 学习 KSZ8001L 网卡驱动程序。
2. 测试网卡功能，编写基于 TFP/UDP 协议网络聊天程序，并且能够接受键盘输入和彼此之间相互发数据。

三、 实验步骤

步骤 1：硬件连接

虚拟机：打开虚拟机

接着查看串口号，通过 putty 软件使用串口通信方式连接实验箱，如下图所示：

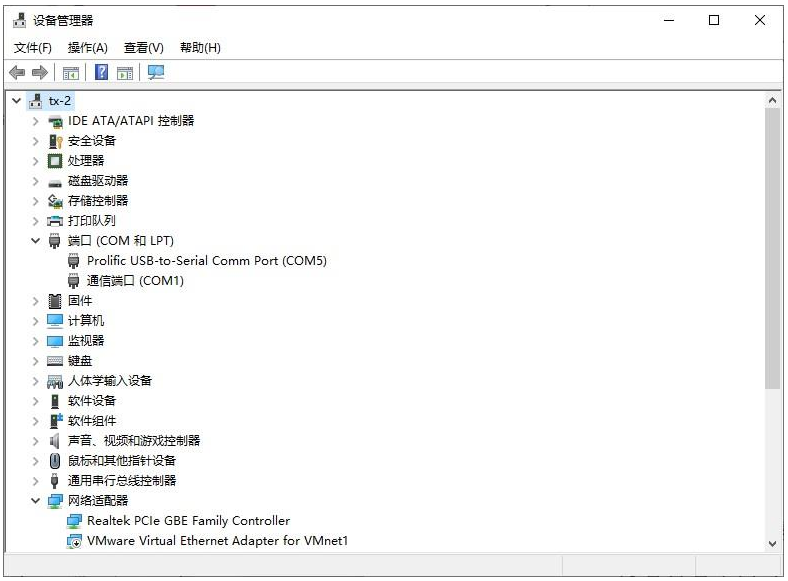

图 10-1 端口号查询

选择 putty 串口连接实验箱:

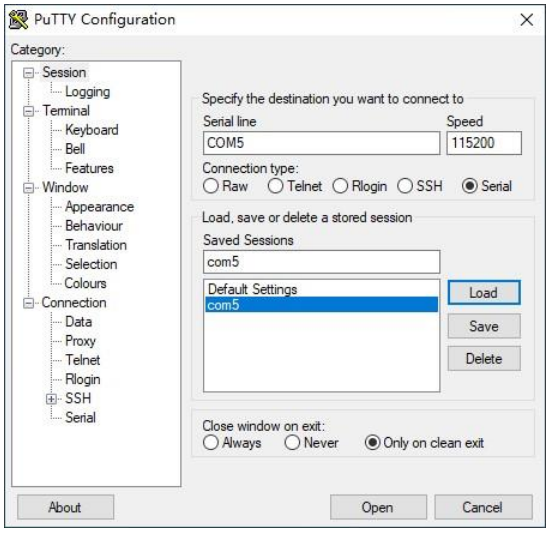

图 10-2 putty 串口连接配置

输入启动参数，接着启动内核

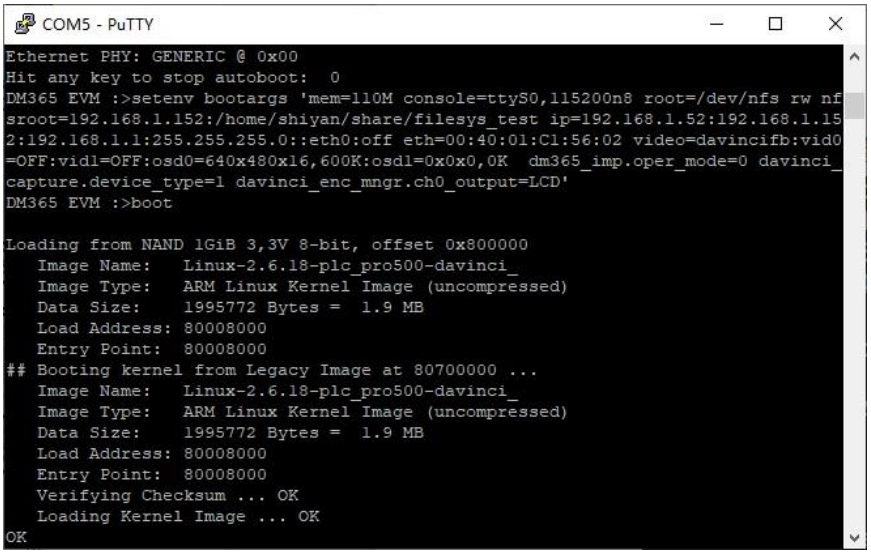

图 10-3 输入启动参数启动

输入用户名 root 登录实验箱：

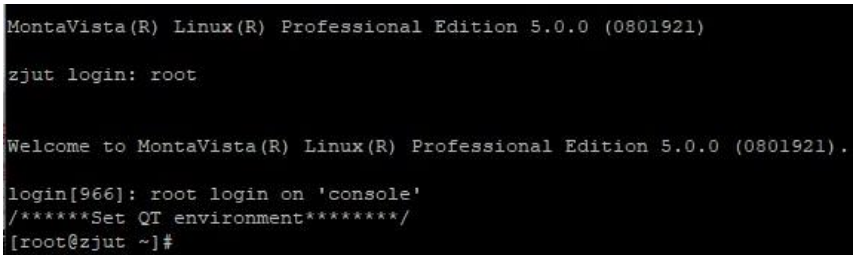

图 10-4 登录实验箱

步骤 2：测试网络连通性（在虚拟机/服务器上）

利用 ping 命令测试，输入命令：

```
$ ping 192.168.1.52
```

(192.168.1.(50+x)为板子设定的 ip，根据实际情况来，此时使用的为第二胎实验箱，x=2，故为 52)

并观察响应时间和丢包率，判断连接是否正常，如果正常，说明 ARP ,IP , ICMP 协议正常：（使用“CTRL+c” 退出 ping 测试）

```
shiyan@ubuntu:~$ ping 192.168.1.52
PING 192.168.1.52 (192.168.1.52) 56(84) bytes of data.
64 bytes from 192.168.1.52: icmp_seq=1 ttl=64 time=0.469 ms
64 bytes from 192.168.1.52: icmp_seq=2 ttl=64 time=0.622 ms
64 bytes from 192.168.1.52: icmp_seq=3 ttl=64 time=0.447 ms
64 bytes from 192.168.1.52: icmp_seq=4 ttl=64 time=0.646 ms
64 bytes from 192.168.1.52: icmp_seq=5 ttl=64 time=0.672 ms
64 bytes from 192.168.1.52: icmp_seq=6 ttl=64 time=0.666 ms
64 bytes from 192.168.1.52: icmp_seq=7 ttl=64 time=0.718 ms
64 bytes from 192.168.1.52: icmp_seq=8 ttl=64 time=0.672 ms
^C
--- 192.168.1.52 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6996ms
rtt min/avg/max/mdev = 0.447/0.614/0.718/0.093 ms
shiyan@ubuntu:~$
```

图 10-5 ping 命令

步骤 3：基于 TCP 的 socket 编程

编写好 TCP 代码，包括服务器端和客户端代码。在用户 home 目录中创建 ethenet 目录，进入该目录创建 tcpclient.c 和 tcpserver.c 文件，代码在附录中。

```
$ mkdir ethenet
$ cd ethenet/
$ vim tcpserver.c
$ vim tcpclient.c
```

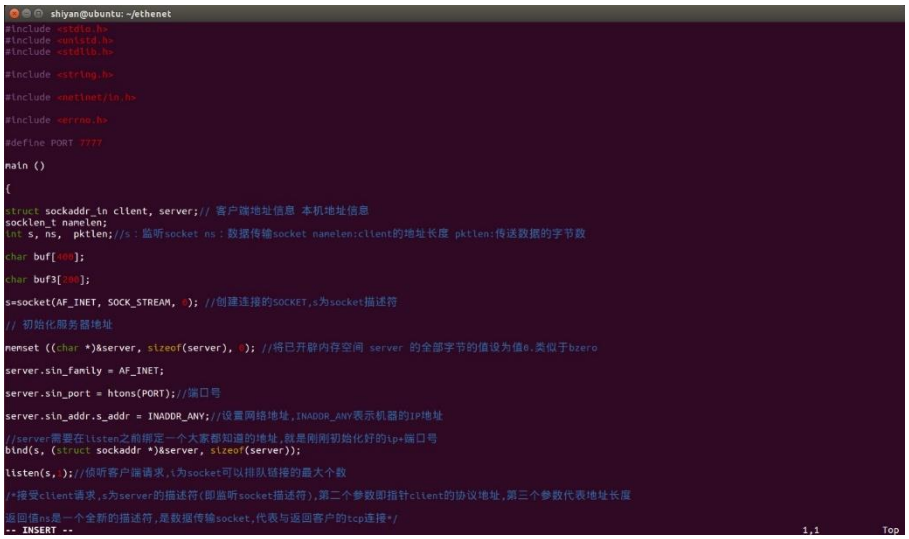

图 10-6 tcpserver.c

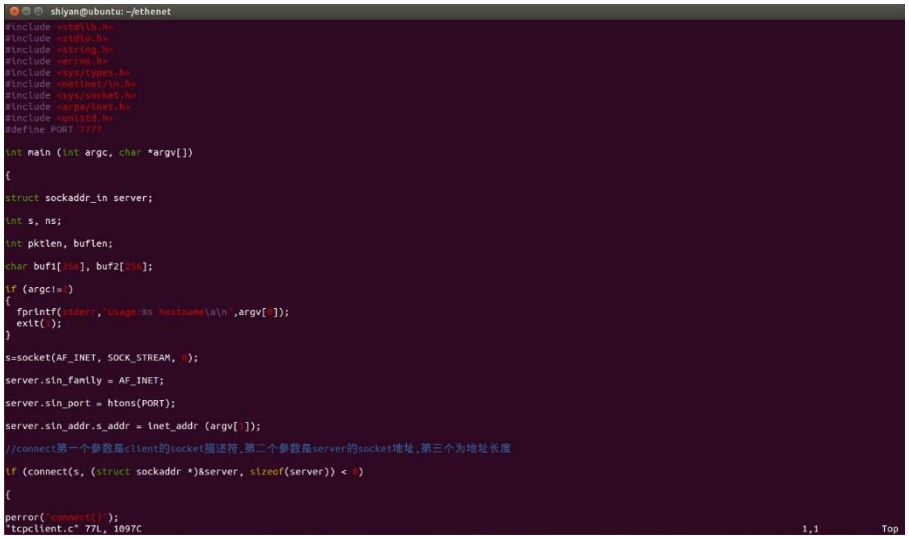

图 10-6 tcpclient.c

编辑完毕客户端和服务器端程序后分别执行:wq 指令保存

退回根目录运行 source /etc/profile 指令使环境变量生效

输入命令编译：

```
$ arm_v5t_le-gcc tcpclient.c -o tcpclient_arm
$ gcc tcpserver.c -o tcpserver-gcc
```

```
shiyan@ubuntu:~/ethenet$ ls
tcpclient_arm  tcpclient.c  tcpserver.c  tcpserver-gcc
shiyan@ubuntu:~/ethenet$
```

图 10-7 生成 TCP 客户端代码和服务端代码

第一行命令将编译生成可以在实验箱上运行的 tcp 客户端代码，第二行编译生成可以在 pc（虚拟机）或 linux 服务器上运行的代码，如果第二行改为如下所示，那么服务器代码也可以在实验箱上运行。

```
$ arm_v5t_le- gcc tcpserver.c -o tcpserver_arm
```

挂载文件系统：

```
mount -t nfs -o nolock 192.168.1.152:/home/shiyan/share/filesys_test /mnt/mtd/
```

```
[root@zjut ~]# df
Filesystem           1k-blocks      Used Available Use% Mounted on
tmpfs                    53868        92     53776   0% /tmp
tmpfs                    10240        32     10208   0% /dev
tmpfs                    53868         0     53868   0% /dev/shm
192.168.1.152:/home/shiyan/share/filesys_test  43218944  10363904  30637056  25%
 /mnt/mtd
[root@zjut ~]#
```

图 10-8 挂载文件系统到实验箱

然后将生成的可执行文件拷贝到文件系统的 /home/shiyan/share/filesys_test/opt/dm365 下：

服务器：$ cp tcpclient_arm /home/stx/filesys_test/opt/dm365
虚拟机：$ cp tcpclient_arm /home/shiyan/share/filesys_test/opt/dm365

```
shiyan@ubuntu:~/ethenet$ cp tcpclient_arm /home/shiyan/share/filesys_test/opt/dm365
shiyan@ubuntu:~/ethenet$
```

图 10-9 将 tcpclient 复制到挂载系统文件夹

在服务端运行./tcpserver-gcc 代码等待客户端连接，客户端运行./tcpserver_arm 192.168.1.152 连接成功（192.168.1.152 是服务端 IP 地址）。

```
shiyan@ubuntu:~/ethenet$ ./tcpserver-gcc
Received line: hello
Enter a line: hello 'client'
```

图 10-10 tcp 服务端成功接收客户端发送的 hello

```
[root@zjut dm365]# ./tcpclient_arm 192.168.1.152
Enter a line: hello
Received line: hello 'client'
Enter a line: hi,'server',i'm client

shiyan@ubuntu:~/ethenet$ ./tcpserver-gcc
Received line: hello
Enter a line: hello 'client'
Received line: hi,'server',i'm client
Enter a line:
```

图 10-10 tcp 客户端成功接收服务端发送的 hello ‘client’并且继续发送消息

步骤 4：基于 UDP 的 socket 编程

编写好 TCP 代码，包括服务器端和客户端代码。在 ethenet 目录，进入该目录创建 udpclient.c 和 udpserver.c 文件，代码在附录中。

$ vim udpserver.c
$ vim udpclient.c

```
shiyan@ubuntu: ~/ethenet
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#define PORT 7050
int main(void)
{
int sockfd,pktlen;
char buf[300],buf1[300];
struct sockaddr_in server;
struct sockaddr_in client;
sockfd=socket(AF_INET,SOCK_DGRAM,0);
memset ((char *)&server, sizeof(server), 0); //将已开辟内存空间 server 的全部字>
节的值设为值0.类似于bzero
server.sin_family = AF_INET;
server.sin_port = htons(PORT);//端口号
server.sin_addr.s_addr = INADDR_ANY;//设置网络地址,INADDR_ANY表示机器的IP地址
"udpserver.c" 41L, 1616C                                      1,1           Top
```

图 10-11 udpserver.c

图 10-11 udpclient.c

编辑完毕客户端和服务器端程序后分别执行:wq 指令保存

退回根目录运行 source /etc/profile 指令使环境变量生效

输入命令编译：

```
$ arm_v5t_le-gcc udpclient.c -o udpclient_arm
$ gcc udpserver.c -o udpserver-gcc
```

图 10-12 生成 UDP 客户端代码和服务端代码

第一行命令将编译生成可以在实验箱上运行的 udp 客户端代码，第二行编译生成可以在 pc（虚拟机）或 linux 服务器上运行的代码，如果第二行改为如下所示，那么服务器代码也可以在实验箱上运行。

```
$ arm_v5t_le- gcc udpserver.c -o udpserver_arm
```

(步骤三中已经挂载文件系统)然后将生成的可执行文件拷贝到文件系统的/home/shiyan/share/filesys_test/opt/dm365 下：

```
服务器：$ cp udpclient_arm /home/stx/filesys_test/opt/dm365
虚拟机：$ cp udpclient_arm /home/shiyan/share/filesys_test/opt/dm365
```

图 10-13 将 udpclient 复制到挂载系统文件夹

在服务端直接运行./udpserver-gcc 指令等待客户端连接：

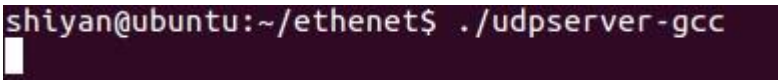

图 10-14 udp 等待客户端连接

在实验箱执行./udpclient_arm 192.168.1.152 命令连接成功：

图 10-15 udp 客户端连接成功

client 向 server 发送：hello,'udpserver'，server 成功接收

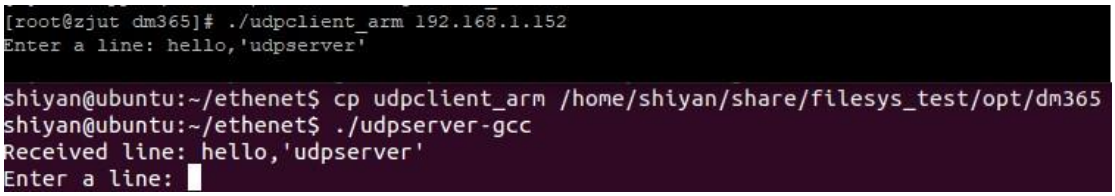

图 10-16 client 发送，server 接收

server 向 client 回复：hi,'udpclient',I'm udpserver，client 成功接收

图 10-17 server 发送，client 接收

四、 心得与体会

通过这次以太网传输实验，对之前学习的 TCP/UDP 协议网络通信协议有了更加清晰的理解，参考已有 TCP 和 UDP 的服务端与客户端程序实现服务器和客户端简单的网络双机通信，接下来要进一步理解相应程序代码，对网络编程的基本原理和常用的函数充分理解，以为之后网络编程的继续学习打下良好基础。

五、 附录

tcpserver.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h
#include <errno.h>
#define PORT 7777

main ()
{
struct sockaddr_in client, server;// 客户端地址信息 本机地址信息
socklen_t namelen;
int s, ns,  pktlen;//s：监听 socket ns：数据传输 socket namelen:client 的地址长度 pktlen:传送数据的字节数
char buf[400];
char buf3[200];
s=socket(AF_INET, SOCK_STREAM, 0); //创建连接的 SOCKET,s 为 socket 描述符

// 初始化服务器地址
memset ((char *)&server, sizeof(server), 0); //将已开辟内存空间 server 的全部字节的值设为值 0.类似于 bzero
server.sin_family = AF_INET;
server.sin_port = htons(PORT);//端口号
server.sin_addr.s_addr = INADDR_ANY;//设置网络地址,INADDR_ANY 表示机器的 IP 地址
//server 需要在 listen 之前绑定一个大家都知道的地址,就是刚刚初始化好的 ip+端口号
bind(s, (struct sockaddr *)&server, sizeof(server));
listen(s,1);//侦听客户端请求,i 为 socket 可以排队链接的最大个数
/*接受 client 请求,s 为 server 的描述符(即监听 socket 描述符),第二个参数即指针 client 的协议地址,第三个参数代表地址长度
返回值 ns 是一个全新的描述符,是数据传输 socket,代表与返回客户的 tcp 连接*/

namelen = sizeof (client);
ns = accept (s, (struct sockaddr *)&client, &namelen);
//开始进行网络 I/O
for (;;) {
/*recv 接受 client 发送的数据,recv 函数仅仅是 copy 数据，真正的接收数据是协议来完成的）， 第一个参数指定接收端套接字描述符；第二个参数指明一个缓冲区，该缓冲区用来存放 recv 函数接收到的数据；第三个参数指明 buf 的长度
recv 函数返回其实际 copy 的字节数*/
pktlen = recv (ns, buf, sizeof (buf), 0);
```

```
if (pktlen == 0)
break;
printf ("Received line: %s\n", buf);
printf ("Enter a line: ");
gets(buf3);

/*并不是 send 把 ns 的发送缓冲中的数据传到连接的另一端的，而是协议传的，send 仅仅是把 buf 中的
数据 copy 到 ns 的发送缓冲区的剩余空间里
返回实际 copy 的字节数*/

send (ns, buf3,sizeof(buf3), 0);
}
close(ns);
close(s);
}
```

tcpclient.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 7777

int main (int argc, char *argv[])
{
struct sockaddr_in server;
int s, ns;
int pktlen, buflen;
char buf1[256], buf2[256];

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

s=socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(PORT);
server.sin_addr.s_addr = inet_addr (argv[1]);
```

```
//connect 第一个参数是 client 的 socket 描述符,第二个参数是 server 的 socket 地址,第三个为地址长度

if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
perror("connect()");
exit(1);
}

//进行网络 I/O
for (;;) {
printf ("Enter a line: ");
gets (buf1);//从 stdin 流中读取字符串，直至接受到换行符
buflen = strlen (buf1);
if (buflen == 0)
break;
send(s, buf1, buflen + 1, 0);
recv(s, buf2, sizeof (buf2), 0);
printf("Received line: %s\n", buf2);
}
close(s);
return 0;
}
```

udpserver.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#define PORT 7050
int main(void)
{
int sockfd,pktlen;
char buf[300],buf1[300];
struct sockaddr_in server;
struct sockaddr_in client;
sockfd=socket(AF_INET,SOCK_DGRAM,0);
memset ((char *)&server, sizeof(server), 0); //将已开辟内存空间 server 的全部字节的值设为值 0.类似于
bzero
server.sin_family = AF_INET;
```

```
server.sin_port = htons(PORT);//端口号
server.sin_addr.s_addr = INADDR_ANY;//设置网络地址,INADDR_ANY 表示机器的 IP 地址
bind(sockfd,(struct sockaddr *)&server,sizeof(struct sockaddr_in));
for (;;) {
/*recv 接受 client 发送的数据,recv 函数仅仅是 copy 数据，真正的接收数据是协议来完成的）， 第一个参数指定接收端套接字描述符；第二个参数指明一个缓冲区，该缓冲区用来存放 recv 函数接收到的数据；第三个参数指明 buf 的长度
recv 函数返回其实际 copy 的字节数*/
socklen_t len=sizeof(struct sockaddr_in);
pktlen = recvfrom (sockfd, buf, sizeof (buf), 0,(struct sockaddr *)&client, &len);
if (pktlen == 0)
break;
printf ("Received line: %s\n", buf);
printf ("Enter a line: ");
fgets(buf1,300,stdin);
/*并不是 send 把 ns 的发送缓冲中的数据传到连接的另一端的，而是协议传的，send 仅仅是把 buf 中的数据 copy 到 ns 的发送缓冲区的剩余空间里
返回实际 copy 的字节数*/
sendto (sockfd, buf1,sizeof(buf1), 0,(struct sockaddr*)&client,len);
}
close(sockfd);
}
```

udpclient.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#define MAX_SIZE 1024
#define PORT 7050
#define HOST_ADDR "192.168.1.51"            //根据组队的服务器地址更改
int main(int argc,char **argv)
{
int sockfd,buflen;
char buf1[300],buf2[300];
struct sockaddr_in server,client;
socklen_t client_length=sizeof(client);
if(argc!=2)
{
   fprintf(stderr,"Usage:%s hostname\a\n",argv[0]);
```

```
    exit(1);
}
sockfd=socket(AF_INET,SOCK_DGRAM,0);
server.sin_family=AF_INET;
server.sin_port = htons(PORT);
server.sin_addr.s_addr = inet_addr(argv[1]);
for(;;)
{
printf ("Enter a line: ");
gets (buf1);//从 stdin 流中读取字符串，直至接受到换行符
buflen = strlen (buf1);
if (buflen == 0)
break;
sendto(sockfd, buf1, buflen + 1, 0,(struct sockaddr *)&server,sizeof(server));
if(recvfrom(sockfd, buf2, sizeof (buf2), 0,(struct sockaddr *)&client,&client_length)==-1)
{
        printf("recvfrom() error\n");
        exit(1);
}
printf("Received line: %s\n", buf2);
}
close(sockfd);
return 0;
}
```