嵌入式系统原理实验报告六

通信 1503 班 201503090323 叶启彬

实验六 以太网传输实验

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、实验设备

1. 硬件：PC 机；串口线；AMR9 系统教学实验系统；网线。
2. 软件：PC 机操作系统（Windows XP）；服务器 Linux 操作系统；超级终端等串口软件；内核等相关软件包；
3. 环境：Ubuntu12.04.4 系统。文件系统版本为 filesys_It、烧写的内核版本为 UImage_It_SDcard1。

三、实验原理

1. 概述

1.1 以太网的工作原理

以太网采用带冲突检测的载波帧听多路访问（CSMA/CD）机制。以太网中节点都可以看到在网络中发送的所有信息，因此，我们说以太网是一种广播网络。

以太网的工作过程如下：

当以太网中的一台主机要传输数据时，它将按如下步骤进行：

1、帧听信道上收否有信号在传输。如果有的话，表明信道处于忙状态，就继续帧听，直到信道空闲为止。

2、若没有帧听到任何信号，就传输数据

3、传输的时候继续帧听，如发现冲突则执行退避算法，随机等待一段时间后，重新执行步骤 1（当冲突发生时，涉及冲突的计算机会发送会返回到帧听信道状态。

注意：每台计算机一次只允许发送一个包，一个拥塞序列，以警告所有的节点）

4、若未发现冲突则发送成功，计算机所有计算机在试图再一次发送数据之前，必须在最近一次发送后等待 9.6 微秒（以 10Mbps 运行）。

1.2 以太网帧传输原理

使用 KSZ8001L 网卡做为以太网的物理层接口。它的基本工作原理是：在收到主机发送的数据后（如图 1 帧格式所示，从目的地址域到数据域），先侦听并判断网络线路。若网络线路繁忙，等到网络线路空闲为止，否则，立即发送数据帧。在其过程中，先填加以太网帧头(如图 1 帧格式所示，包括前导码和帧开始标志)，接着生成 CRC 校验码，最后将以数据帧形式将其发送到以太网上去。

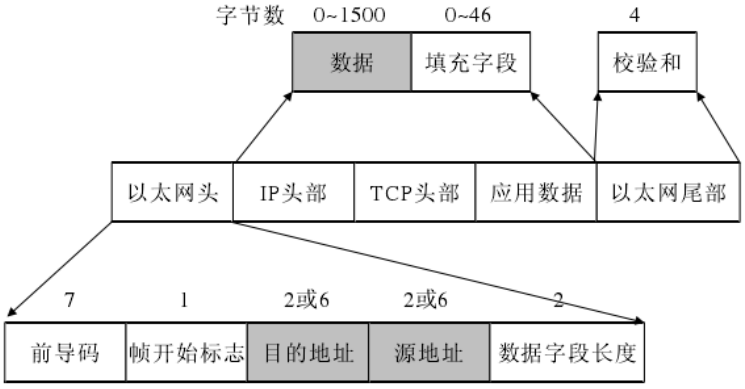

图 1.802.3 帧格式

以太网的数据帧规定，以太网包头为 14byte ，IP 包头为 20byte ，TCP 包头为 20byte ，有些以太网的最大帧长为 1514 字节。

在网卡接收数据过程中，将先把从以太网收到的数据帧通过解码、去帧头和地址检验等相关步骤后缓存在片内；再经过 CRC 校验后，通知网卡 KSZ8001L 接收已经到了数据帧；最后，用某种传输模式传送 ARM 的存储器中。大多数嵌入式系统内嵌一个以太网控制器，用来支持媒体独立接口（MII）和带缓冲的 DMA 接口（BDI）。可在半双工或全双工模式下提供 10M/100Mbps 的以太网接入。在半双工模式下，控制器支持 CSMA/CD 协议，在全双工模式下控制器能够支持 IEEE 802.3 MAC 的控制层协议。

以太网内部控制器内部结构图如下所示

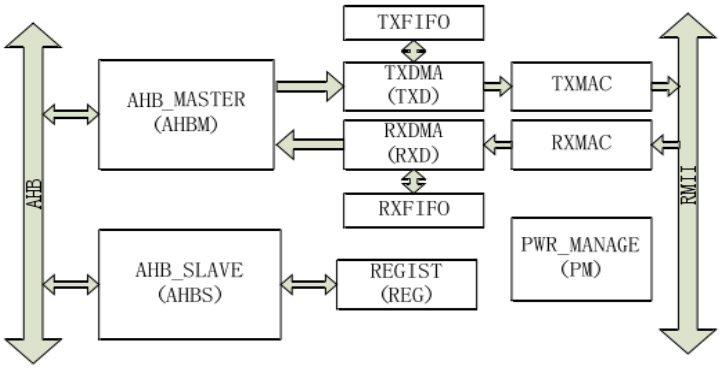

图 2.MAC 控制器的结构框图

由图可以看出模块的组成以及模块中各组成部分的连接方式。MAC 控制器各组成部分的特征或功能有：

1.AHB_MASTER 同时支持大端和小端模式，可以根据需要自行设定。一旦决定其模式，AHB_SLAVE 也将按这种模式工作。

2.TXDMA 主要有读取发送状态描述符或将发送状态写入描述符、将要发送的数据包从发送缓存区中传送到 TXFIFO 中、控制 TXFIFO 的读写状态等三个功能。

3.RXDMA 也有三个作用，分别是：读取接收状态描述符或将接收状态写入描述符、将要发送的数据包从 TXFIFO 中传送到接收缓存区中、控制 RXFIFO 的读写状态。

4.TXMAC 的作用是将数据包从 TXFIFO 中传送到网卡，RXMAC 的作用是将数据包从网卡接收到 RXFIFO 中。不论是接收到的还是要发送出去的数据都包括前缀、校验、发送状态等。

1.2.1 以太网发送和接收的流程

当 TXMAC 去发送一个数据包时，它首先会检测网卡的状态，挂起发送装置直到网卡空闲。然后，TXMAC 给数据包加上前缀和校验信息，将数据包发送到网卡。如果在发送过程中 TXMAC 探测到冲突，就向网卡发送阻塞信号，然后检测冲突源是否是外来的，如果不是，在等待休止时间后，继续发送该数据包。发送的流程如图 3 所示。

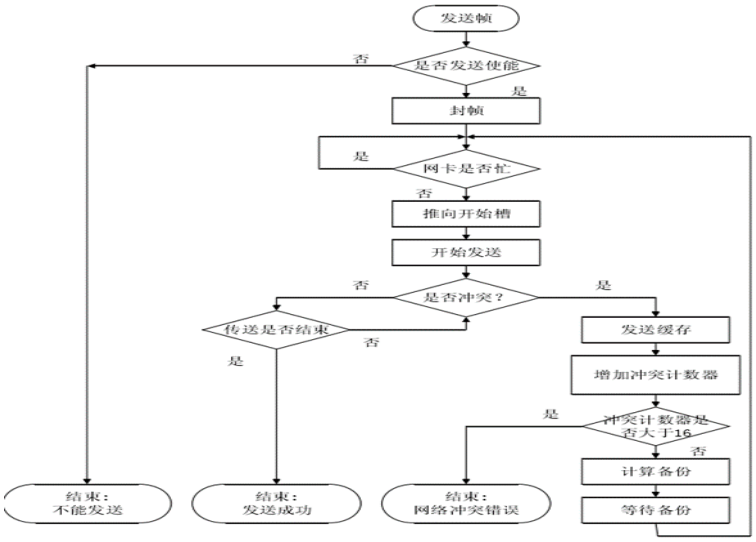

图 3.MAC 发送流程

当网络上有数据包到达，RXDMA 会将数据包从 RXMAC 中拿出来，并传送到 RX FIFO 中。当校验信息和数据包的地址都是正确时，RXDMA 就会将收到的数据包保存在 RXFIFIO 中，否则，忽略该数据包。图 4 描述了 MAC 接收数据包的基本流程。

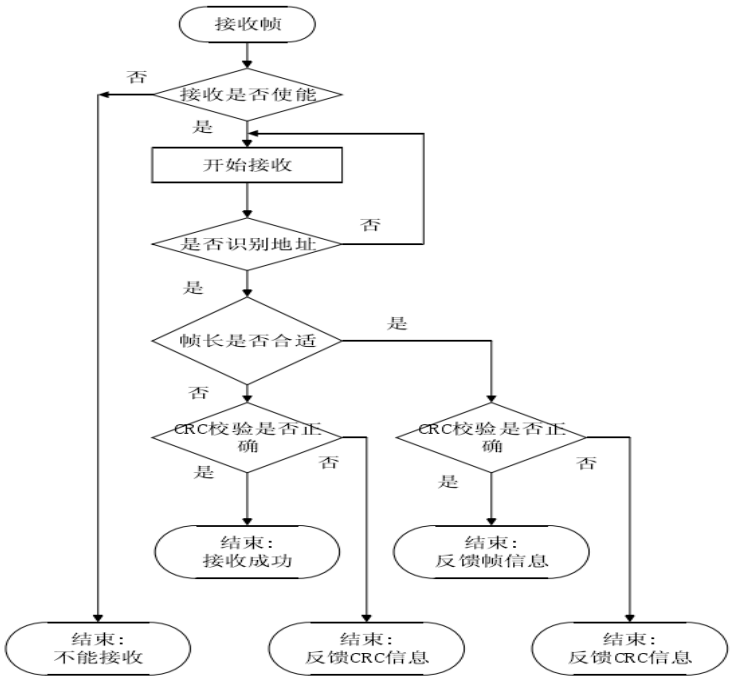

图 4.MAC 接收流程图

2.1 IP 网络协议原理

TCP/IP 协议是一组包括 TCP 协议和 IP 协议，UDP 协议、ICMP 协议和其他一些协议的协议组。

TCP/IP 协议采用分层结构共分为四层，每一层独立完成指定功能，如图：

应用层（第四层）
传输层（第三层）
互联层（第二层）
网络接口层（第一层）

图 5.TCP/IP 协议层

网络接口层：负责接收和发送物理帧。

互联层：负责相邻接点之间的通信。

传输层：负责起点到终点的通信。

应用层：定义了应用程序使用互联网的规程。

2.2 TCP 协议

TCP 是面向连接的通信协议，通过三次握手建立连接，通讯完成时要拆除连接，由于 TCP 是面向连接的所以只能用于端到端的通讯。

TCP 提供的是一种可靠的数据流服务，采用“带重传的肯定确认”技术来实现传输的可靠性。TCP 还采用一种称为“滑动窗口”的方式进行流量控制，所谓窗口实际表示接收能力，用以限制发送方的发送速度。

如果 IP 数据包中有已经封好的 TCP 数据包，那么 IP 将把它们向‘上’传送到 TCP 层。TCP 将包排序并进行错误检查，同时实现虚电路间的连接。TCP 数据包中包括序号和确认，所以未按照顺序收到的包可以被排序，而损坏的包可以被重传。

TCP 将它的信息送到更高层的应用程序，例如 Telnet 的服务程序和客户程序。应用程序轮流将信息送回 TCP 层，TCP 层便将它们向下传送到 IP 层，设备驱动程序和物理介质，最后到接收方。

面向连接的服务（例如 Telnet、FTP、rlogin、X Windows 和 SMTP）需要高度的可靠性，所以它们使用了 TCP。DNS 在某些情况下使用 TCP（发送和接收域名数据库），但使用 UDP 传送有关单个主机的信息。

TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN+ACK，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接，TCP 使用的流量控制协议是可变大小的滑动窗口协议。

TCP 三次握手的过程如下：

（1）客户端发送 SYN（SEQ=x）报文给服务器端，进入 SYN_SEND 状态。

（2）服务器端收到 SYN 报文，回应一个 SYN （SEQ=y）ACK(ACK=x+1）报文，进入 SYN_RECV 状态。

（3）客户端收到服务器端的 SYN 报文，回应一个 ACK(ACK=y+1）报文，进入 Established 状态。

三次握手完成，TCP 客户端和服务器端成功地建立连接，可以开始传输数据了。

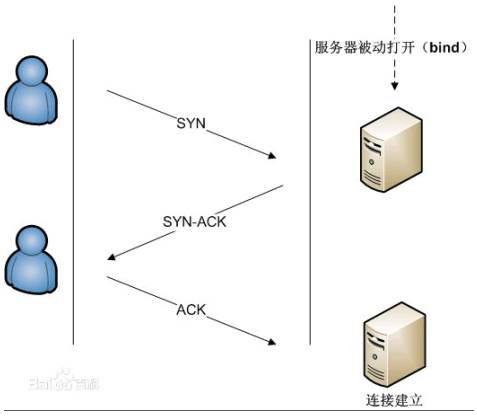

图 6. TCP 的三次握手

2.3 基于 TCP 的 socket 编程

在 server 端，server 首先启动，调用 socket( )创建套接字；然后调用 bind( ) 绑定 server 的地址（IP+port）；再调用 listen( )让 server 做好侦听准备，并规定好请求队列长度，然后 server 进入阻塞状态，等待 client 的连接请求；最后通过 accept( )来接收连接请求，并获得 client 的地址。当 accpet 接收到一个 client 发来的 connet 请求时，将生成一个新的 socket，用于传输数据。

在 client 端，client 在创建套接字并指定 client 的 socket 地址，然后就调用 connect( )和 server 建立连接。一旦连接建立成功，client 和 server 之间就可以通过调用 recv 和 send 来接收和发送数据。一旦数据传输结束，server 和 client 通过调用 close( )来关闭套接字。原理图如下：

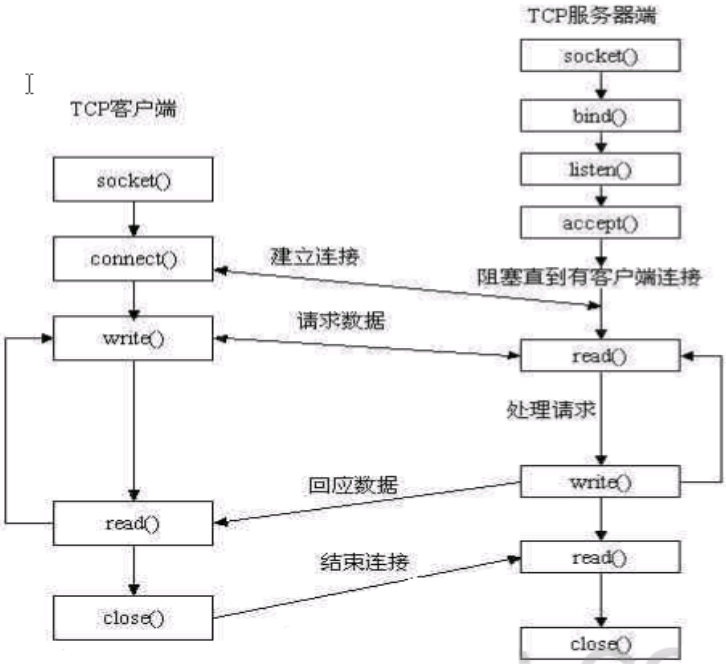

图 7. TCP 通信原理图

2.4 UDP 协议

UDP 是面向无连接的通讯协议，UDP 数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。

UDP 通讯时不需要接收方确认，属于不可靠的传输，可能会出现丢包现象，实际应用中要求程序员编程验证。

UDP 与 TCP 位于同一层，但它不管数据包的顺序、错误或重发。因此，UDP 不被应用于那些使用虚电路的面向连接的服务，UDP 主要用于那些面向查询---应答的服务，例如 NFS。相对于 FTP 或 Telnet，这些服务需要交换的信息量较小。使用 UDP 的服务包括 NTP（网络时间协议）和 DNS（DNS 也使用 TCP）。

UDP 是 OSI 参考模型中一种无连接的传输层协议，它主要用于不要求分组顺序到达的传输中，分组传输顺序的检查与排序由应用层完成，提供面向事务的简单不可靠信息传送服务。UDP 协议基本上是 IP 协议与上层协议的接口。UDP 协议适用端口分别运行在同一台设备上的多个应用程序。

UDP 提供了无连接通信，且不对传送数据包进行可靠性保证，适合于一次传输少量数据，UDP 传输的可靠性由应用层负责。常用的 UDP 端口号有：

应用协议	端口号
DNS	53
TFTP	69
SNMP	161

UDP 在 IP 报文中的位置如图所示：

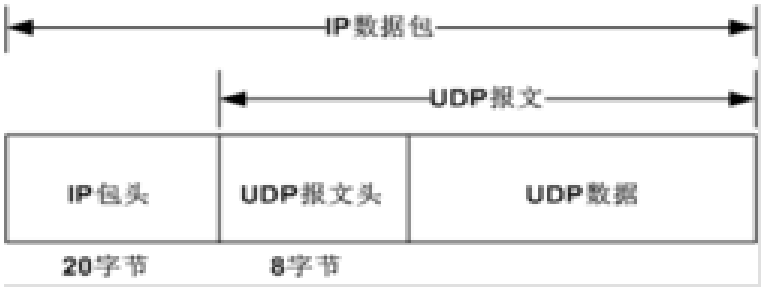

图 8.UDP 在 IP 报文位置

2.5 基于 UDP 的 socket 编程

基于 UDP 协议的无连接 C/S 的工作流程 在 server 端，server 首先启动，调用 socket()创建套接字，然后调用 bind( )绑定 server 的地址（IP+port），调用 recvfrom( )等待接收数据。

在 client 端，先调用 socket()创建套接字，调用 sendto( )向 server 发送数据。

server 接收到 client 发来数据后，调用 sendto( )向 client 发送应答数据，client 调用 recv 接收 server 发来的应答数据。数据传输结束，server 和 client 通过调用 close()关闭套接字。原理图如下图

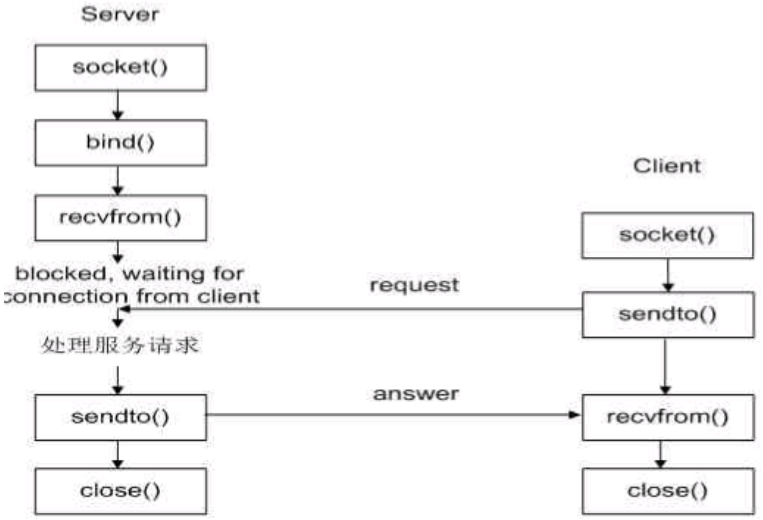

图 9.UDP 通信原理图

3.驱动开发的基本流程

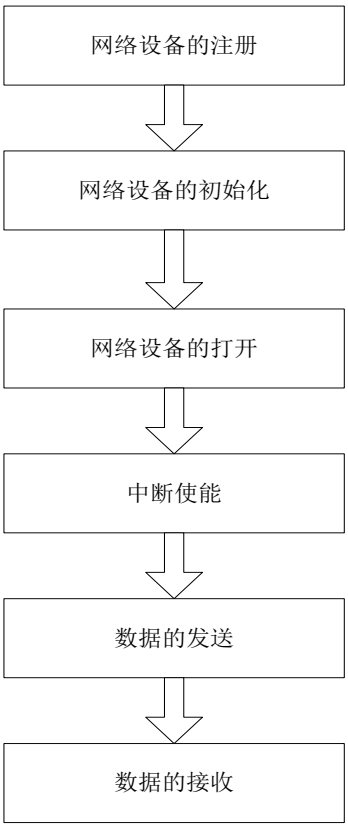

图 10. 驱动开发基本流程图

3.1 驱动程序中重要函数

```
static int __init davinci_emac_init(void)//网络设备驱动的注册
static void __exit davinci_emac_exit(void)//网络设备的注销
static int __init davinci_emac_probe(struct platform_device *pdev)
//网络设备的检测，包括以太网的分配，时钟使能等
static void emac_enable_interrupt(struct emac_dev *dev, int ack_eoi)//使能中断
static int emac_open(struct emac_dev *dev, void *param)
//设备打开，包括使能设备的硬件资源，申请中断、DMA，激活发送列队等
static int emac_send(struct emac_dev *dev, net_pkt_obj *pkt, int channel, bool send_args)
//数据包的发送
static int emac_net_rx_cb(struct emac_dev *dev, net_pkt_obj *net_pkt_list,void *rx_args)
//数据包的接收
```

四、实验内容（代码注释及步骤）

1. 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 TCP/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。

2. 实验步骤：

步骤 1：硬件连接

登陆 PuTTY 的 COM 口端（可能不是 COM4，在设备管理器查看连接的是哪个 COM 口,Speed 注意要设为 115200）

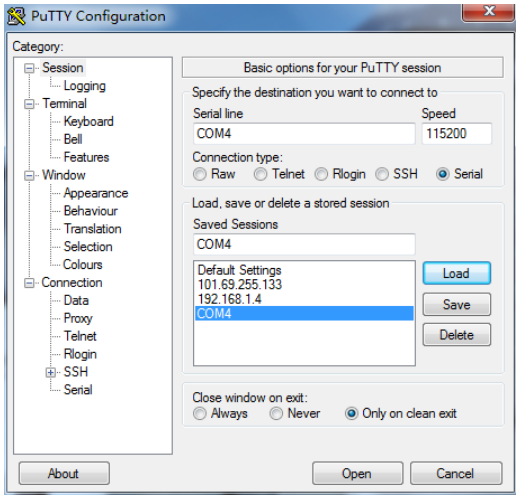

图 11. PuTTY 配置界面

打开设备开关，在 PuTTY 的 COM 口端进行操作，当实验板有打印消息时，按 enter 键使系统停止启动，输入启动参数：

setenv bootargs 'mem=110M console=ttyS0,115200n8 root=/dev/nfsrw nfsroot=192.168.1.188:/home/st17/filesys_test ip=192.168.1.67:192.168.1.88:192.168.1.1:255.255.255.0::eth0:off eth=00:40:06:2B:64:67 video=davincifb:vid0=OFF:vid1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0Kdm365_imp.oper_mode=0davinci_capture.device_type=1davinci_enc_mngr.ch0_output=LCD'

保存启动参数：

#saveenv

在 PuTTY 的 COM 口端进行操作，输入 boot，启动实验板：

#boot

启动之后输入 root 登陆用户

步骤 2：测试网络连通性

利用 ping 命令测试 ,输入命令：

ping 192.168.1.188

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。如下图所示

```
[root@zjut /]# ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4): 56 data bytes
64 bytes from 192.168.1.4: seq=0 ttl=64 time=1.1 ms
64 bytes from 192.168.1.4: seq=1 ttl=64 time=0.5 ms
64 bytes from 192.168.1.4: seq=2 ttl=64 time=0.5 ms
64 bytes from 192.168.1.4: seq=3 ttl=64 time=0.5 ms

--- 192.168.1.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.5/0.6/1.1 ms
```

图 12. 测试网络连通性

步骤 3：基于 TCP 的 socket 编程

在了解 TCP 的通信原理后，编写基于该协议的聊天程序，客户端和服务器端相互发送消息，熟悉协议原理。请注意在编写程序时注意主机的 IP 应当和你板子的 ip 一致。相关程序如下：

服务器程序代码注释：

```
#include <stdio.h>                    //头文件
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
```

```
#include <errno.h>
#define PORT 7777                                   //端口号
main ()                                             //主函数
{
    struct sockaddr_in client, server;              //客户端地址信息 本机地址信息
    socklen_t namelen;                              //定义数据的长度类型
    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];                                  //创建缓冲区
    char buf3[200];                                 //创建缓冲区
    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.
    server.sin_family = AF_INET;                    //sin_family 表示协议簇；在 socket 编程中只能是 AF_INET
    server.sin_port = htons(PORT);                  //端口号 7777，将主机字节顺序转换成网络字节顺序
    server.sin_addr.s_addr = INADDR_ANY;            //设置网络地址,INADDR_ANY 表示机器的 IP 地址
    bind(s, (struct sockaddr *)&server, sizeof(server));
    //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 函数）
    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);                      //提取 client 发送的数据的长度
    ns = accept (s, (struct sockaddr *)&client, &namelen);//接收客户端的服务请求。
    //int accept(int sockfd,struct sockaddr *addr,socklen_t *addrlen)为 accept 的原型函数
    //开始进行网络 I/O
    for (;;)                                        //进行了 for 无限循环；
    {
        /*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)                            //如果 pktlen 为 0；
            break;                                  //跳出
        printf ("Received line: %s\n", buf);        //接收数据；
        printf ("Enter a line: ");                  //打印出“Enter a line:”；
        gets(buf3);                                 //从 stdin 流中读取字符串，直至接受到换行符；
        /*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);            //发送数据；
    }
    close(ns);
    close(s);
}
```

客户端代码注释：

```
#include <stdlib.h>                                 //头文件
#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                                   //端口号
int main (int argc, char *argv[])                   //主函数，需要获得形参 argc，和*argv[]
{
    struct sockaddr_in server;                      //服务器地址信息
    int s, ns;                                      // s：监听 socket；ns：数据传输 socket namelen:client 的地址长度；
```

```
    int pktlen, buflen;                          // pktlen：传送数据的字节数，buflen：定义缓冲区长度
    char buf1[256], buf2[256];                   //创建缓冲区
    if (argc!=2)                                 //在 argc 不等于 2 时，输入服务器的地址
    {
        fprintf(stderr,"Usage:%s hostname\a\n",argv[0]); //stderr 标准输出文件，在打印界面上需要输入服务器的地址；
        exit(1);                                 //非正常运行导致退出程序；
    }
    s=socket(AF_INET, SOCK_STREAM, 0);           //创建连接 socket
    // 初始化客户端地址
    server.sin_family = AF_INET;                 //创建套接字时，用该字段指定地址家族，对于 TCP/IP 协议的，
                                                 //必须设置为 AF_INET；
    server.sin_port = htons(PORT);               //端口号 7777
    server.sin_addr.s_addr = inet_addr (argv[1]); //设置网络地址,为服务器的地址
    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
                                                 //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
                                                 //客户端发送的服务连接。
        {
            perror("connect()");                 //如果失败将错误信息返回到操作系统
            exit(1);                             //非正常运行导致退出程序；
        }
    //进行网络 I/O
    for (;;)                                     //进行了 for 无限循环；
    {
        printf ("Enter a line: ");               //打印出“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);                  //取字符长度
        if (buflen == 0)                         //如果 buflen 为 0；
            break;                               //跳出；
        send(s, buf1, buflen + 1, 0);            //输出字符串；
        recv(s, buf2, sizeof (buf2), 0);         //接收字符串；
        printf("Received line: %s\n", buf2);     //打印出来；
    }
    close(s);
    return 0;
}
```

步骤 4：交叉编译并执行命令

编辑完毕客户端和服务器端程序后分别保存为 tcplient.c 和 tcpsever.c，输入命令编译：

```
arm_v5t_le-gcc tcpclient.c -o tcpc
arm_v5t_le-gcc tcpserver.c -o tcps
```

然后将生成的可执行文件拷贝到文件系统的 st2 下：

```
cp tcpc /home/st2/filesys_test/st2
cp tcps /home/st2/filesys_test/st2
```

图 12. 交叉编译生成 tcpc 和 tcps

测试通话需要启动两个实验箱，一个实验箱先执行服务端的应用程序 tcps，

命令：./tcps

另个实验箱再执行客户端的应用程序 tcpc，命令:./tcpc 192.168.1.XX（服务端 ip）

然后客户端向服务端发消息 a，服务端回 b，客户端发 c，服务端回 d。

服务端：

```
dir.c                       sys
dir1                        tcpc
dm365                       tcpclient.c
etc                         tcps
filesys_clwxl.tar.gz        tcpserver.c
fm1188_i2c.ko               test.sh
gpio                        tftp
gpio1                       tmp
helloworld                  uImage_lt_SDcard1.SDcard1
helloworld1                 udpclient
hy0.wav                     udps_1.c
init                        udpserver
lib                         udpserver2.c
linuxrc                     usb
lll.wav                     usr
lost+found                  var
lxq                         ver.txt
lxq.wav                     wav
makesh
[root@zjut /]# ./tcps
Received line: 2
Enter a line: 7
Received line: 3
Enter a line: 5
Received line: a
Enter a line: 5
```

图 13.服务器

客户端：

```
[root@zjut /]# ./tcpc 192.168.1.52
TCPclient@st0 initialized...Please enter a message here: 2
Received line: 7
TCPclient@st0 initialized...Please enter a message here: 3
Received line: 5
TCPclient@st0 initialized...Please enter a message here: a
Received line: 5
TCPclient@st0 initialized...Please enter a message here:
```

图 14.客户端

三、实验总结及心得体会

在本次实验开始前，先复习了关于 TCP 和 UDP 的协议，了解以太网通讯原理和驱动程序开发方法，以及基于 TCP/UDP 的 socket 编程。课前没有及时将 TCP 程序理解完全，导致在实验过程中有部分错误发生，通过检查实验代码发现错误，更正后完成了实验，完成两台实验箱之间的通讯。通过本次实验，深入了解了 TCP 协议的作用，并且应用到了实际的例子中，局域网使我们生活中经常接触的概念，工业中，局域网对于一个系统来说是相当重要的，元器件间的通信都是通过局域网来完成的。聊天程序使我们生活中经常要用到的，通过本次实验了解了此类程序的基本构成。