**题目：以太网数据帧的发送与接收**

**姓名: 王翰墨**

**学号：2019302683**

**班号：10011902**

**课程代号：网络试点班**

**计算机学院**

**时间:2021年10月4日**

**目 录**

摘 要

[1 目的 1](#_Toc341302685)

[2 要求 1](#_Toc341302686)

[3 相关知识 2](#_Toc341302687)

[4 实现原理 2](#_Toc341302688)

[5 程序代码 5](#_Toc341302689)

[6 运行结果与分析 19](#_Toc341302690)

[7 参考文献 22](#_Toc341302690)

# 题目：以太网数据帧的发送与接收

# 目的

编写程序，实现以太网数据帧的发送与接收；在此基础上，尝试改用双线程发送接收的思想优化程序。

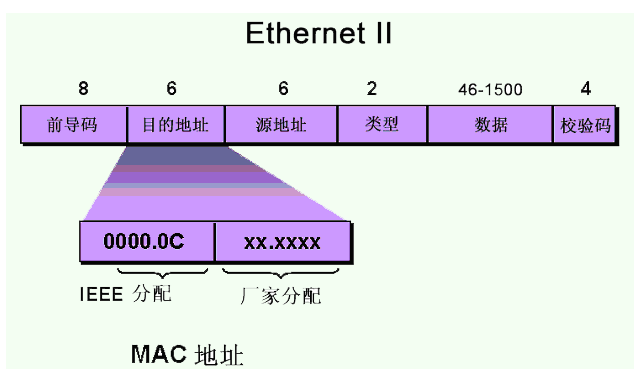
# 2、要求

**发送方：**从指定文件读取数据并按照数据部分46-1500字节的大小要求封装数据帧：若超过1500字节则拆分分组，若不满46字节则补0。对数据生成CRC32校验码并封装成帧，然后进行发送，直到发送完成为止。

**接收方：**对数据帧根据目的MAC地址进行接收并进行CRC32校验，对以太网头部进行解析并输出信息，然后将数据保存到指定文件中，直到接收完成为止。

**对发送方和接收方进行优化，采用多线程收发以太网数据帧。**

# 3、相关知识

****

前导码：01111110（逢5加0， 逢5删0）；

目的MAC地址：6字节，例如00-09-73-07-74-73；

源MAC地址： 6字节，例如00-09-73-07-73-f9；

类型： ICMPv4 ，IGMP, IPv4(0x0800)，ARP; ICMP v4, IPv6等；

数据: 最小46字节，最大1500字节；

校验字段: 32比特位CRC校验。

# 4、实现原理

**发送方：**

1. **输入输出**

**输入：**命令行输入4个参数，分别是①本机MAC地址（源MAC地址），②目的MAC地址，③发送的网口号，④数据文件名。此外还需要一个存放数据的文件用于读取。

**输出：**执行过程中，每封装完成一个数据帧并放入Pool后会输出“生成第%d个数据帧”；每成功发送一个数据帧会输出“发送第%d个数据帧”。

1. **双线程任务**

**线程read\_from\_file：**打开指定数据文件，用fread读出不超过1500字节的数据块，对不满46字节的情况进行补0操作，然后将以太网头部、数据部分载入buffer中，根据数据生成CRC32校验码并添加到buffer尾部。将buffer按ReadIndex顺序标记送入发送池中，循环直到数据全部读完为止，置起ReadEndFlag。

**线程send：**使用pcap\_findalldevs\_ex()寻找网络设备并连接指定的网口，然后用SendIndex（<ReadIndex）索引发送池中的数据帧，通过pcap\_sendpacket()发送数据帧，循环直到ReadEndFlag置起后的所有ReadIndex全部发送完。

1. **实现细节**

进程的互斥和同步由P操作和V操作完成，设置三个信号量Mutex=1，Full=0，Empty=100。其中Mutex用于保护临界区的发送池，Full和Empty用于进程同步，即Empty为0则不能读取（Pool中没有剩余空间），Full为0则不能发送（Pool中没有可发数据帧）。

发送方的read\_from\_file线程调用load\_ethernet\_data()、load\_ethernet\_header()和calculate\_crc()来构造完整的以太网数据帧，并用全局变量size\_of\_packet[]保存每一个数据帧的长度。在send线程中pcap\_sendpacket()使用size\_of\_packet[SendIndex]即可完成定长发送，但改参数为1518时会提示发送失败，因此只能将数据部分压缩为最大1496字节，保证整个帧长1514字节，能够成功发送。此外，两个线程的进度分别对应了ReadIndex和SendIndex，可以保证发送的数据不是乱序的。

**接收方：**

1. **输入输出**

**输入：**命令行输入4个参数，分别是①本机MAC地址（目的MAC地址），②认可接收的MAC地址（源MAC地址），③发送的网口号，④用于保存数据的文件名。

**输出：**执行过程中，每收到一个数据帧并检验正确后会打印该数据帧的信息，包括源MAC地址、目的MAC地址、上层协议等。同时还会将数据部分写入指定的文件中去。

1. **双线程任务**

**线程receive：**使用pcap\_findalldevs\_ex()寻找网络设备并连接指定的网口，然后用 pcap\_loop()不停地检测网口上收到的数据帧，在ethernet\_protocol\_packet\_callback()中对数据帧的目的MAC地址进行检测，如果是广播地址全F或者本机MAC地址则接收，核对源MAC地址和CRC32校验码，无误后对以太网头部进行解析并输出，将数据部分按ReceiveIndex顺序标记放入接收池中，循环直到数全部接收完毕为止，置起ReceiveEndFlag。

**线程write\_to\_file：**打开或创建指定名称的文件，用WriteIndex（<ReceiveIndex）索引接收池中的数据部分，通过fwrite()写入文件，循环直到ReceiveEndFlag置起后的所有ReceiveIndex全部写入完。

1. **实现细节**

进程的互斥和同步由P操作和V操作完成，设置三个信号量Mutex=1，Full=0，Empty=100。其中Mutex用于保护临界区的接收池，Full和Empty用于进程同步，即Empty为0则不能接收（Pool中没有剩余空间），Full为0则不能写入（Pool中没有可写入数据）。

接收方的receive线程调用pcap\_loop()来持续接收数据帧，调用ethernet\_protocol\_packet\_callback()来解析数据帧，但目前没有设置一个表示结束的帧，因而会不停地等待接收数据帧。在接收过程中，除了检查目的MAC地址和CRC校验码，还对源MAC地址做检查，确认是本设备认可的设备后才会接收数据。此外，两个线程的进度分别对应了ReceiveIndex和WriteIndex，可以保证接收后写入的数据不是乱序的。

# 5、程序代码

发送方datalink\_send.cpp：

#include<stdio.h>

#include<stdlib.h>

#define HAVE\_REMOTE

#include<pcap.h>

#pragma comment(lib,"ws2\_32.lib")

#pragma comment(lib,"Packet.lib")

#pragma warning(disable:4996)

#define ETHERNET\_IP 0x0800

#define MAX\_PACKAGE\_SIZE 1514

#define NUM\_QUE 100

int size\_of\_packet[NUM\_QUE];

u\_int32\_t crc32\_table[256];

int INum = 1;

char filename[MAX\_PACKAGE\_SIZE] = "data.txt";

//ethernet header

struct ethernet\_header

{

    u\_int8\_t dest\_mac[6];

    u\_int8\_t src\_mac[6];

    u\_int16\_t ethernet\_type;

};

void P(int\* s);

void V(int\* s);

DWORD WINAPI read\_from\_file(LPVOID pM);

DWORD WINAPI send(LPVOID pM);

void load\_ethernet\_header(u\_int8\_t\* buffer);

int load\_ethernet\_data(u\_int8\_t\* buffer, FILE\* fp);

void generate\_crc32\_table();

u\_int32\_t calculate\_crc(u\_int8\_t\* buffer, int len);

u\_int8\_t DestinationMac[6] = { 0x00, 0x09, 0x73, 0x07, 0x74, 0x73 };

u\_int8\_t SourceMac[6] = { 0x00, 0x09, 0x73, 0x07, 0x73, 0xf9 };

int Mutex = 1;

int Full = 0;

int Empty = NUM\_QUE;

int ReadEndFlag = 0;

int ReadIndex = 0;

int SendIndex = 0;

u\_int8\_t Pool[NUM\_QUE][MAX\_PACKAGE\_SIZE] = { NULL };    //缓存队列

//generate table

void generate\_crc32\_table()

{

    int i, j;

    u\_int32\_t crc;

    for (i = 0; i < 256; i++)

    {

        crc = i;

        for (j = 0; j < 8; j++)

        {

            if (crc & 1)

                crc = (crc >> 1) ^ 0xEDB88320;

            else

                crc >>= 1;

        }

        crc32\_table[i] = crc;

    }

}

u\_int32\_t calculate\_crc(u\_int8\_t\* buffer, int len)

{

    int i, j;

    u\_int32\_t crc;

    crc = 0xffffffff;

    for (i = 0; i < len; i++)

    {

        crc = (crc >> 8) ^ crc32\_table[(crc & 0xFF) ^ buffer[i]];

    }

    crc ^= 0xffffffff;

    return crc;

}

void P(int\* s) {

    while ((\*s) <= 0);// printf("wait.. Mutex=%d, Full=%d, Empty=%d\n", Mutex, Full, Empty);

    (\*s)--;

}

void V(int\* s) {

    (\*s)++;

}

void load\_ethernet\_header(u\_int8\_t\* buffer)

{

    struct ethernet\_header\* hdr = (struct ethernet\_header\*)buffer;

    hdr->dest\_mac[0] = DestinationMac[0];//this is where you can define the mac address

    hdr->dest\_mac[1] = DestinationMac[1];

    hdr->dest\_mac[2] = DestinationMac[2];

    hdr->dest\_mac[3] = DestinationMac[3];

    hdr->dest\_mac[4] = DestinationMac[4];

    hdr->dest\_mac[5] = DestinationMac[5];

    hdr->src\_mac[0] = SourceMac[0];//source mac address

    hdr->src\_mac[1] = SourceMac[1];

    hdr->src\_mac[2] = SourceMac[2];

    hdr->src\_mac[3] = SourceMac[3];

    hdr->src\_mac[4] = SourceMac[4];

    hdr->src\_mac[5] = SourceMac[5];

    hdr->ethernet\_type = ETHERNET\_IP;

}

int load\_ethernet\_data(u\_int8\_t\* buffer, FILE\* fp)

{

    int size\_of\_data = 0;

    char tmp[MAX\_PACKAGE\_SIZE];

    size\_of\_data = fread(tmp, 1, 1496, fp);

    if (size\_of\_data == 0)

        return sizeof(ethernet\_header);

    while(size\_of\_data < 46 && size\_of\_data>0){

        \*(tmp + size\_of\_data) = 0;

        size\_of\_data++;

    }

    u\_int32\_t crc = calculate\_crc((u\_int8\_t\*)tmp, size\_of\_data);

    //printf("crc = %x\n", crc);

    int i;

    for (i = 0; i < size\_of\_data; i++)

    {

        \*(buffer + i) = tmp[i];

    }

    \*(u\_int32\_t\*)(buffer + i) = crc;

    return (sizeof(ethernet\_header) + size\_of\_data + 4);

}

DWORD WINAPI read\_from\_file(LPVOID pM) {

    FILE\* fp = fopen(filename, "r");

    if (fp == NULL) {

        printf("\nThe file is opened error.\n");

        system("pause");

        exit(1);

    }

    int HeaderSize = sizeof(ethernet\_header);

    int flag = 0;

    while (ReadIndex < 100) {

        P(&Empty);

        //printf("Read P Empty, M=%d,E=%d,F=%d\n",Mutex,Empty,Full);

        P(&Mutex);

        //printf("Read P Mutex, M=%d,E=%d,F=%d\n",Mutex, Empty, Full);

        flag = (load\_ethernet\_data(Pool[ReadIndex] + HeaderSize, fp));

        //printf("%d : flag = %d\n", ReadIndex, flag);

        if (flag == HeaderSize) {

            V(&Mutex);

            V(&Full);

            break;

        }

        size\_of\_packet[ReadIndex] = flag;

        load\_ethernet\_header(Pool[ReadIndex]);

        printf("\n生成第%d个帧\n", ReadIndex);

        ReadIndex++;

        V(&Full);

        //printf("Read V Full, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

        V(&Mutex);

        //printf("Read V Mutex, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

    }

    ReadEndFlag = 1;

}

DWORD WINAPI send(LPVOID pM) {

    pcap\_if\_t\* alldevs;

    pcap\_if\_t\* d;

    pcap\_t\* adhandle;

    int i = 0;

    char ErrBuf[PCAP\_ERRBUF\_SIZE];

    if (pcap\_findalldevs\_ex((char\*)PCAP\_SRC\_IF\_STRING, NULL, &alldevs, ErrBuf) == -1){

        printf("\nError in findalldevs\_ex function: %s\n", ErrBuf);

        system("pause");

        exit(1);

    }

    for (d = alldevs; d != NULL; d = d->next) i++;

    /\*for (d = alldevs; d != NULL; d = d->next) {

        printf("\n%d.%s\n", ++i, d->name);

        if (d->description)

            printf("---%s", d->description);

        else

            printf("No description available.\n");

        printf("\n");

    }\*/

    if (i == 0) {

        printf("\nNo interfaces found!Make sure WinPcap is installed.\n");

        pcap\_freealldevs(alldevs);

        system("pause");

        exit(1);

    }

    /\*int INum;

    printf("\nEnter the adapter id between 1 and %d: ", i);

    scanf("%d", &INum);\*/

    if (INum < 1 || INum > i){

        printf("\n Adapter id out of range.\n");

        pcap\_freealldevs(alldevs);

        system("pause");

        exit(1);

    }

    for (d = alldevs, i = 0; i < INum - 1; d = d->next, i++);

    if ((adhandle = pcap\_open\_live(d->name, 65536, 1, 1000, ErrBuf)) == NULL) {

        printf("\nUnable to open the adapter.%s is not supported by WinPcap.\n", d->name);

        //pcap\_freealldevs(alldevs);

        system("pause");

        exit(1);

    }

    while (SendIndex < ReadIndex || ReadEndFlag == 0){

        P(&Full);

        //printf("Send P Full, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

        P(&Mutex);

        //printf("Send P Mutex, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

        //printf("size\_of\_packet:%d\n", size\_of\_packet[SendIndex]);

        if (pcap\_sendpacket(adhandle, (const u\_char\*)Pool[SendIndex], size\_of\_packet[SendIndex]) != 0)

            printf("\n%d.Error sending the packet:%s\n", SendIndex, pcap\_geterr(adhandle));

        else

            printf("\nPacket %d has been sent.\n", SendIndex);

        SendIndex++;

        V(&Empty);

        //printf("Send V Empty, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

        V(&Mutex);

        //printf("Send V Mutex, M=%d,E=%d,F=%d\n", Mutex, Empty, Full);

    }

    pcap\_freealldevs(alldevs);

    printf("\n数据帧已经发送完成\n");

    system("pause");

    exit(1);

}

int main()

{

    int i;

    generate\_crc32\_table();

    printf("请设置本设备MAC地址：");

    for (i = 0; i < 6; i++)

        scanf("%x", &SourceMac[i]);

    printf("请输入目的MAC地址：");

    for (i = 0; i < 6; i++)

        scanf("%x", &DestinationMac[i]);

    printf("请设置发送网口号：");

    scanf("%d", &INum);

    printf("请输入需发送的文件：");

    scanf("%s", filename);

    printf("Mutex=%d\n", Mutex);

    Mutex = 1;

    Empty = NUM\_QUE;

    Full = 0;

    CreateThread(NULL, 0, read\_from\_file, NULL, 0, NULL);

    CreateThread(NULL, 0, send, NULL, 0, NULL);

    while (1);

    system("pause");

    return 0;

}

接收方datalink\_recv.cpp：

#include<stdio.h>

#include<stdlib.h>

#define HAVE\_REMOTE

#include<pcap.h>

#include<WinSock2.h>

#pragma warning(disable:4996)

#define NUM\_QUE 100

void ethernet\_protocol\_packet\_callback(u\_char\* argument, const struct pcap\_pkthdr\* packet\_header, const u\_char\* packet\_content);

//ethernet protocol header format

struct ethernet\_header

{

    u\_int8\_t ether\_dhost[6];//destination mac

    u\_int8\_t ether\_shost[6];//src mac

    u\_int16\_t ether\_type;

};

int adapter\_id;

char filename[1514];

u\_int8\_t accept\_dest\_mac[2][6] = { { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff }, { 0x00, 0x09, 0x73, 0x07, 0x74, 0x73 } };

u\_int8\_t accept\_source\_mac[2][6] = { {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}, {0x00, 0x09, 0x73, 0x07, 0x73, 0xf9} };

u\_int32\_t crc32\_table[256];

//generate table

int Mutex = 1;

int Full = 0;

int Empty = NUM\_QUE;

int ReceiveIndex = 0;

int WriteIndex = 0;

int ReceiveEndFlag = 0;

u\_int8\_t Pool[100][1500] = { NULL };

void P(int\* s)

{

    while ((\*s) <= 0);

    (\*s)--;

}

void V(int\* s)

{

    (\*s)++;

}

DWORD WINAPI receive(LPVOID pM)

{

    pcap\_if\_t\* all\_adapters;

    pcap\_if\_t\* adapter;

    pcap\_t\* adapter\_handle;

    char error\_buffer[PCAP\_ERRBUF\_SIZE];

    if (pcap\_findalldevs\_ex((char\*)PCAP\_SRC\_IF\_STRING, NULL, &all\_adapters, error\_buffer) == -1)

    {

        fprintf(stderr, "Error in findalldevs\_ex function: %s\n", error\_buffer);

        return -1;

    }

    if (all\_adapters == NULL)

    {

        printf("\nNo adapters found! Make sure WinPcap is installed!!!\n");

        return 0;

    }

    int id = 1;

    for (adapter = all\_adapters; adapter != NULL; adapter = adapter->next) id++;

    /\*for (adapter = all\_adapters; adapter != NULL; adapter = adapter->next)

    {

        printf("\n%d.%s\n", id++, adapter->name);

        printf("--- %s\n", adapter->description);

    }

    printf("\n");

    int adapter\_id;

    printf("Enter the adapter id between 1 and %d: ", id - 1);

    scanf("%d", &adapter\_id);\*/

    if (adapter\_id<1 || adapter\_id>id - 1)

    {

        printf("\n Adapter id out of range.\n");

        pcap\_freealldevs(all\_adapters);

        return -1;

    }

    adapter = all\_adapters;

    for (id = 1; id < adapter\_id; id++)

    {

        adapter = adapter->next;

    }

    adapter\_handle = pcap\_open(adapter->name, 65535, PCAP\_OPENFLAG\_PROMISCUOUS, 5, NULL, error\_buffer);

    if (adapter\_handle == NULL)

    {

        fprintf(stderr, "\n Unable to open adapter: %s\n", adapter->name);

        pcap\_freealldevs(all\_adapters);

        return -1;

    }

    pcap\_loop(adapter\_handle, NULL, ethernet\_protocol\_packet\_callback, NULL);

    pcap\_freealldevs(all\_adapters);

    ReceiveEndFlag = 1;

}

void generate\_crc32\_table()

{

    int i, j;

    u\_int32\_t crc;

    for (i = 0; i < 256; i++)

    {

        crc = i;

        for (j = 0; j < 8; j++)

        {

            if (crc & 1)

                crc = (crc >> 1) ^ 0xEDB88320;

            else

                crc >>= 1;

        }

        crc32\_table[i] = crc;

    }

}

u\_int32\_t calculate\_crc(u\_int8\_t\* buffer, int len)

{

    int i, j;

    u\_int32\_t crc;

    crc = 0xffffffff;

    for (i = 0; i < len; i++)

    {

        crc = (crc >> 8) ^ crc32\_table[(crc & 0xFF) ^ buffer[i]];

    }

    crc ^= 0xffffffff;

    return crc;

}

//ethernet protocol analysis

void ethernet\_protocol\_packet\_callback(u\_char\* argument, const struct pcap\_pkthdr\* packet\_header, const u\_char\* packet\_content)

{

    u\_short ethernet\_type;

    struct ethernet\_header\* ethernet\_protocol;

    u\_char\* mac\_string;

    static int packet\_number = 1;

    ethernet\_protocol = (struct ethernet\_header\*)packet\_content;

    int len = packet\_header->len;

    int i, j;

    ////check the mac address

    ////if the packet is sended to my pc or broadcast

    int flag = 2;

    for (i = 0; i < 2; i++)

    {

        flag = 2;

        for (j = 0; j < 6; j++)

        {

            if (ethernet\_protocol->ether\_dhost[j] == accept\_dest\_mac[i][j])

                continue;

            else

            {

                flag = i;

                break;

            }

        }

        if (flag != 2)continue;

        else

            break;

    }

    if (flag != 2)

    {

        return;

    }

    if (i == 0)

    {

        printf("It's broadcasted.\n");

    }

    // if the source is acceptable

    for (i = 0; i < 2; i++)

    {

        flag = 1;

        for (j = 0; j < 6; j++)

        {

            if (ethernet\_protocol->ether\_shost[j] == accept\_source\_mac[i][j])

                continue;

            else

            {

                flag = 0;

                break;

            }

        }

        if (flag)

            break;

    }

    if (flag == 0)return;

    //see if the data is changed or not

    u\_int32\_t crc = calculate\_crc((u\_int8\_t\*)(packet\_content + sizeof(ethernet\_header)), len - 4 - sizeof(ethernet\_header));

    if (crc != \*((u\_int32\_t\*)(packet\_content + len - 4)))

    {

        printf("The data has been changed.\n");

        return;

    }

    printf("----------------------------\n");

    printf("capture %d packet\n", packet\_number);

    printf("capture time: %d\n", packet\_header->ts.tv\_sec);

    printf("packet length: %d\n", packet\_header->len);

    printf("-----Ethernet protocol-------\n");

    ethernet\_type = ethernet\_protocol->ether\_type;

    printf("Ethernet type: %04x\n", ethernet\_type);

    switch (ethernet\_type)

    {

    case 0x0800:printf("Upper layer protocol: IPV4\n"); break;

    case 0x0806:printf("Upper layer protocol: ARP\n"); break;

    case 0x8035:printf("Upper layer protocol: RARP\n"); break;

    case 0x814c:printf("Upper layer protocol: SNMP\n"); break;

    case 0x8137:printf("Upper layer protocol: IPX\n"); break;

    case 0x86dd:printf("Upper layer protocol: IPV6\n"); break;

    case 0x880b:printf("Upper layer protocol: PPP\n"); break;

    default:

        break;

    }

    mac\_string = ethernet\_protocol->ether\_shost;

    printf("MAC source address: %02x:%02x:%02x:%02x:%02x:%02x\n", \*mac\_string, \*(mac\_string + 1), \*(mac\_string + 2), \*(mac\_string + 3),

        \*(mac\_string + 4), \*(mac\_string + 5));

    mac\_string = ethernet\_protocol->ether\_dhost;

    printf("MAC destination address: %02x:%02x:%02x:%02x:%02x:%02x\n", \*mac\_string, \*(mac\_string + 1), \*(mac\_string + 2),

        \*(mac\_string + 3), \*(mac\_string + 4), \*(mac\_string + 5));

    /\*if (ethernet\_type == 0x0800)

    {

        ip\_protocol\_packet\_callback(argument, packet\_header, packet\_content + sizeof(ethernet\_header));

    }

\*/

//show the data;

    for (u\_int8\_t\* p = (u\_int8\_t\*)(packet\_content + sizeof(ethernet\_header)); p != (u\_int8\_t\*)(packet\_content + packet\_header->len - 4); p++)

    {

        printf("%c", \*p);

    }

    printf("\n");

    printf("----------------------\n");

    int k = 0;

    P(&Empty);

    P(&Mutex);

    for (u\_int8\_t\* p = (u\_int8\_t\*)(packet\_content + sizeof(ethernet\_header)); p != (u\_int8\_t\*)(packet\_content + packet\_header->len - 4); p++)

    {

        Pool[ReceiveIndex][k++] = \*p;

    }

    printf("\nPool[%d]:%s\n", ReceiveIndex, Pool[ReceiveIndex]);

    ReceiveIndex++;

    V(&Mutex);

    V(&Full);

    packet\_number++;

}

DWORD WINAPI write\_to\_file(LPVOID pM)

{

    FILE\* recvfile = fopen(filename, "w");

    if (recvfile == NULL) {

        printf("opened error\n");

        system("pause");

        exit(1);

    }

    while (WriteIndex <= ReceiveIndex || ReceiveEndFlag == 0) {

        P(&Full);

        P(&Mutex);

        fwrite(&Pool[WriteIndex], 1, sizeof(Pool[WriteIndex]), recvfile);

        WriteIndex++;

        V(&Mutex);

        V(&Empty);

    }

    fclose(recvfile);

    printf("End\n");

    system("pause");

    exit(1);

}

int main()

{

    int i;

    generate\_crc32\_table();

    printf("请设置本设备MAC地址：");

    for (i = 0; i < 6; i++)

        scanf("%x", &accept\_dest\_mac[1][i]);

    printf("请输入认可接收数据的MAC地址：");

    for (i = 0; i < 6; i++)

        scanf("%x", &accept\_source\_mac[1][i]);

    printf("请设置接收网口号：");

    scanf("%d", &adapter\_id);

    printf("请输入生成的文件名：");

    scanf("%s", filename);

    Mutex = 1;

    Empty = NUM\_QUE;

    Full = 0;

    CreateThread(NULL, 0, receive, NULL, 0, NULL);

    CreateThread(NULL, 0, write\_to\_file, NULL, 0, NULL);

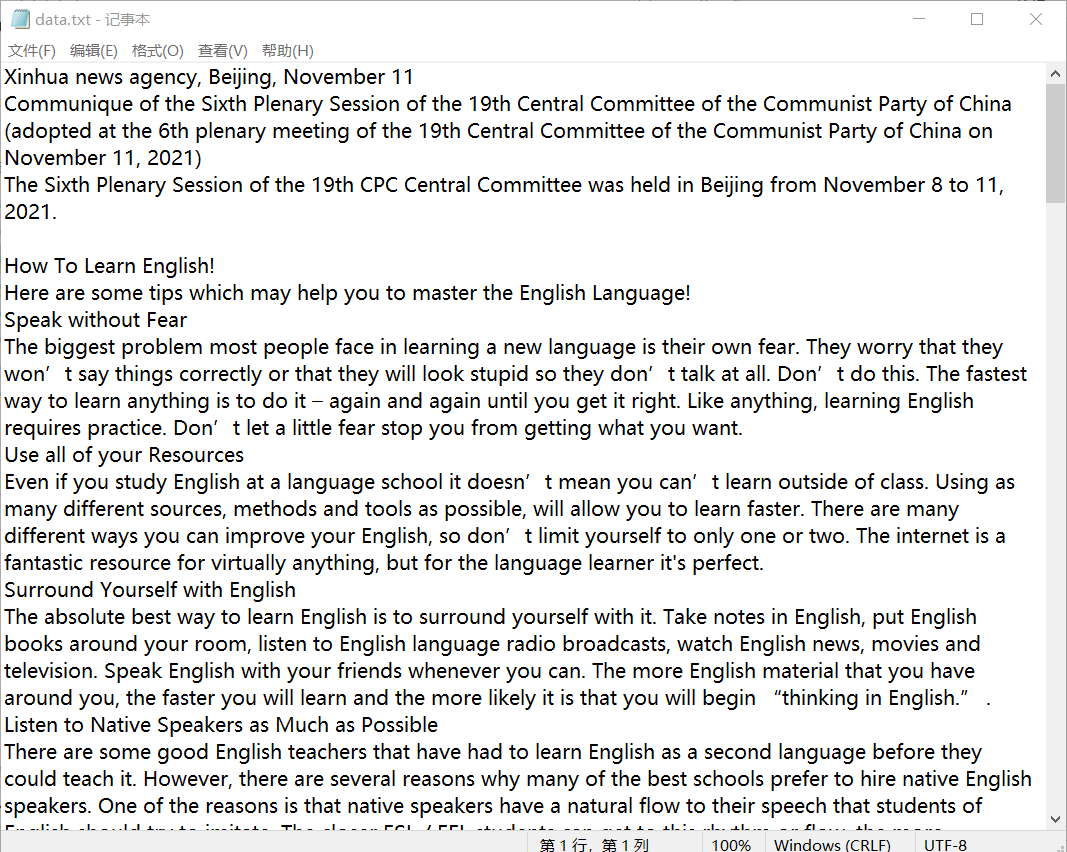
    while (1);

}

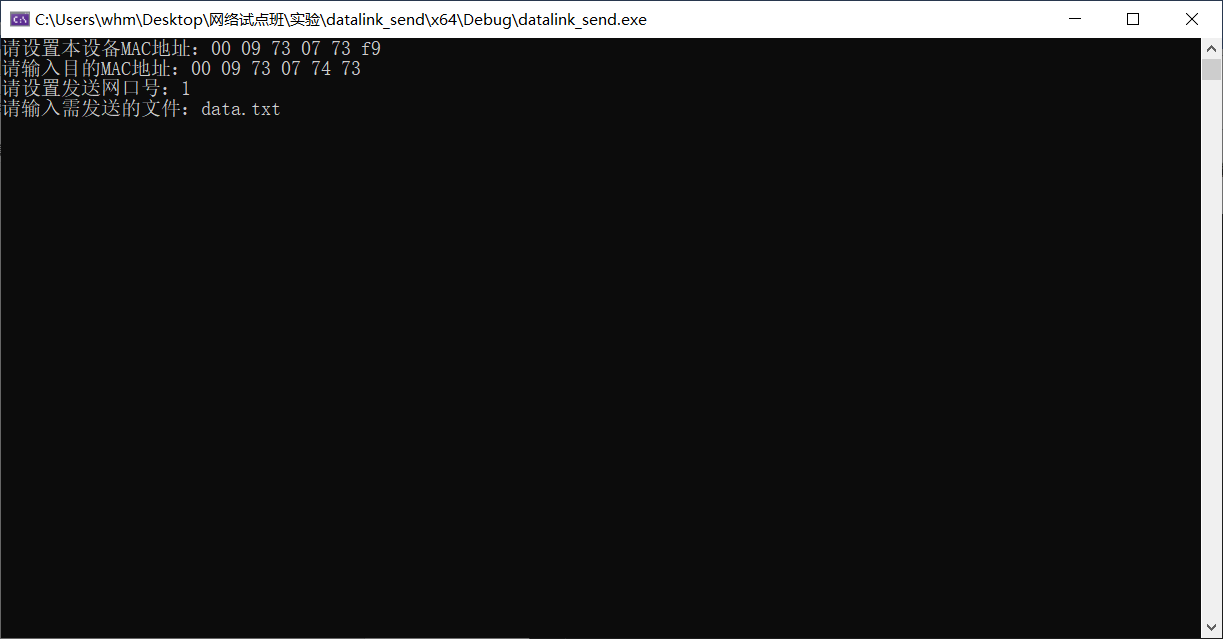
# 6、运行结果与分析

**发送方：**

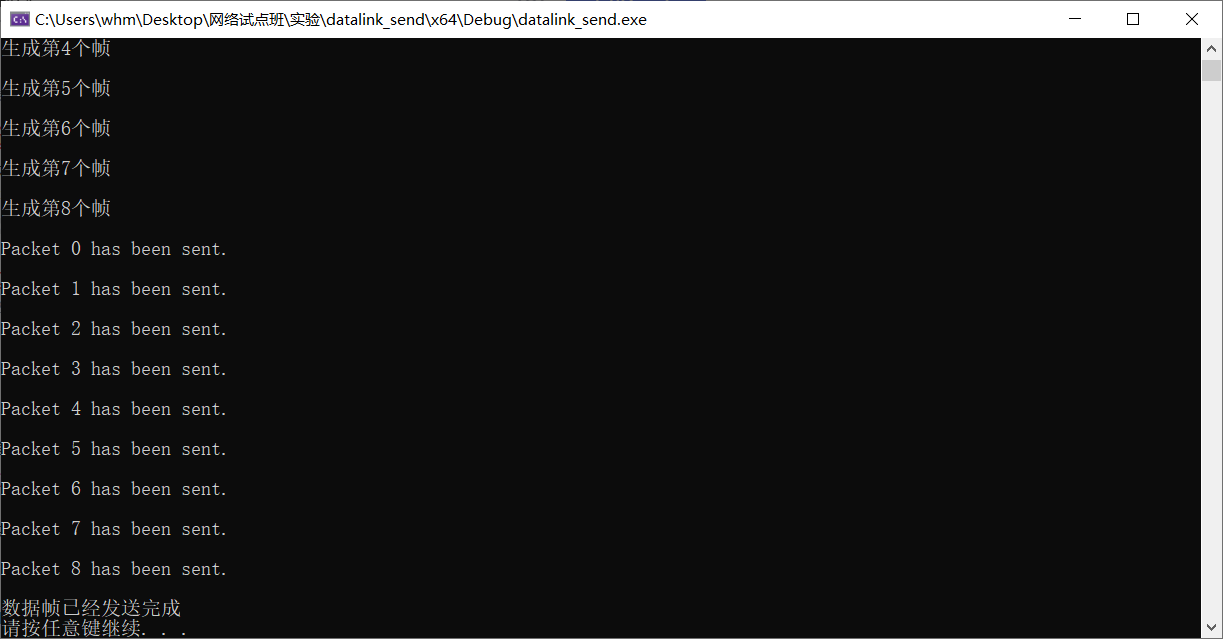
①测试样例data.txt内容如下：



②设置发送端地址和网口信息

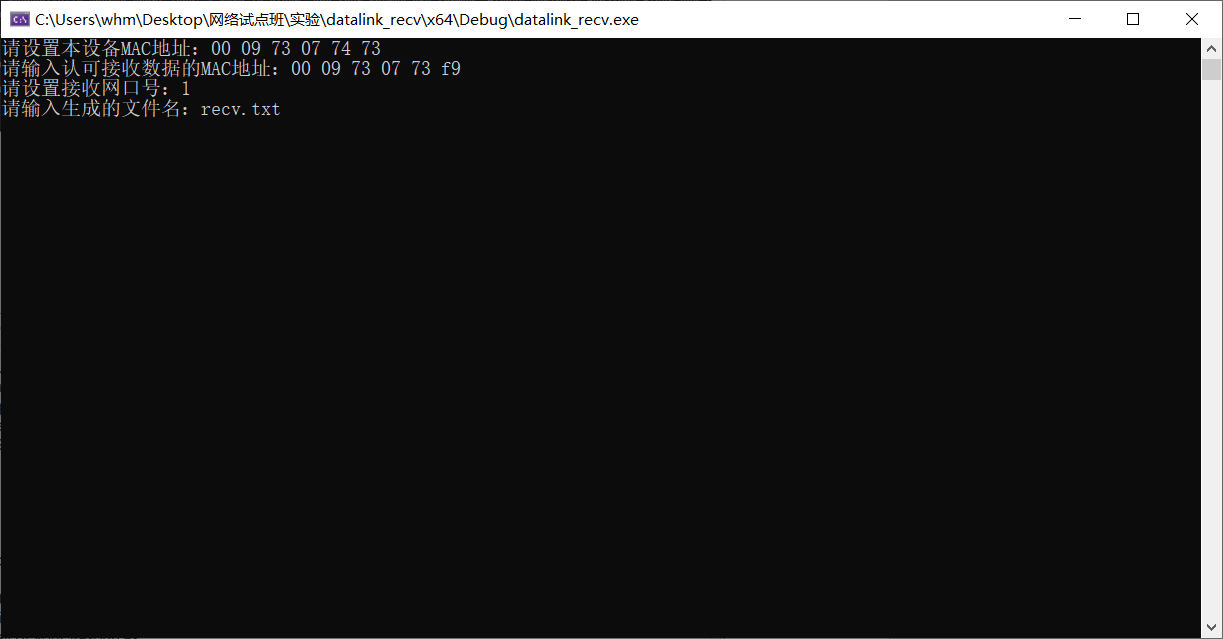


1. 发送数据帧，可以看到生成并发送了8个数据帧。



**接收方：**

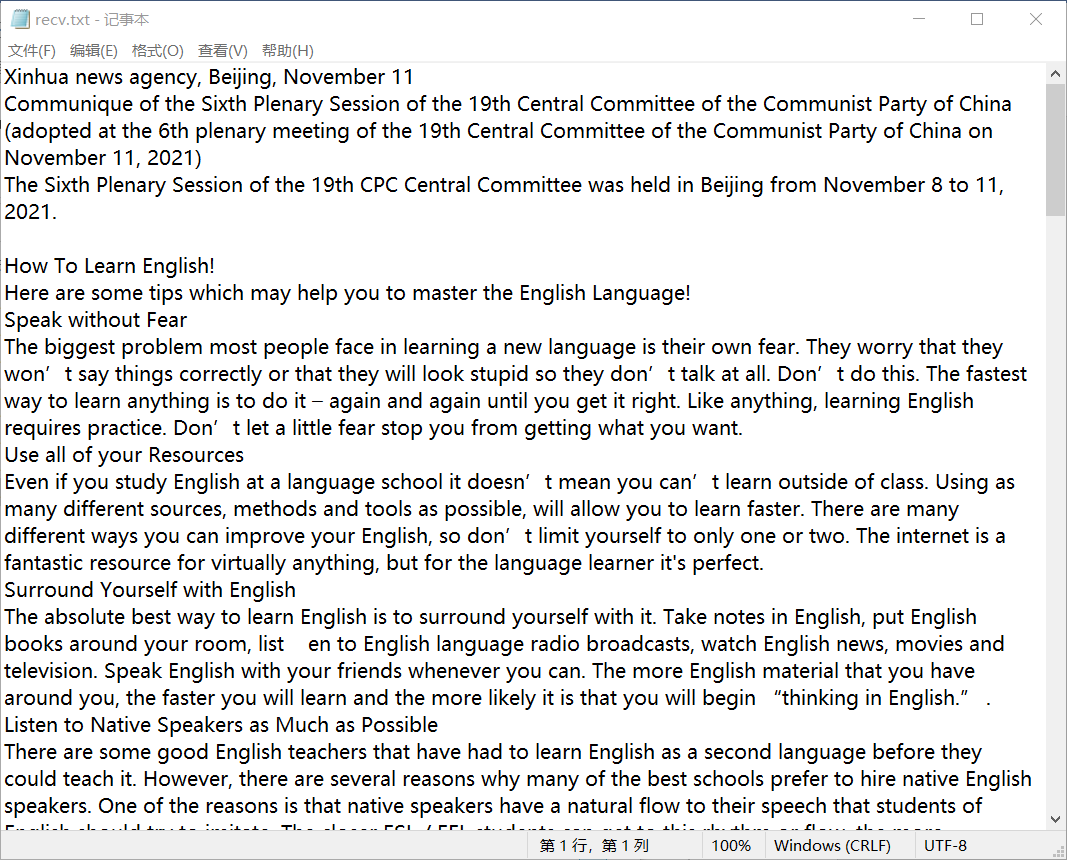
①设置发送端地址和网口信息



②接收到8个数据帧，并打印出了相关信息。



③查看文件recv.txt，数据正确，内容如下：



# 7、参考文献

* 《计算机网络协议分析与实践》姚烨,朱怡安 电子工业出版社