### 实验报告：

### 一、实验目的

尝试测试210目标板与笔记本电脑Kali Linux间通过网线、网络通信程序通讯

### 二、实验器材与原料

210目标板、安装Linux操作系统的笔记本电脑、网线、USB-串口线、网络通讯程序

### 三、实验原理

同一网络下socket通讯

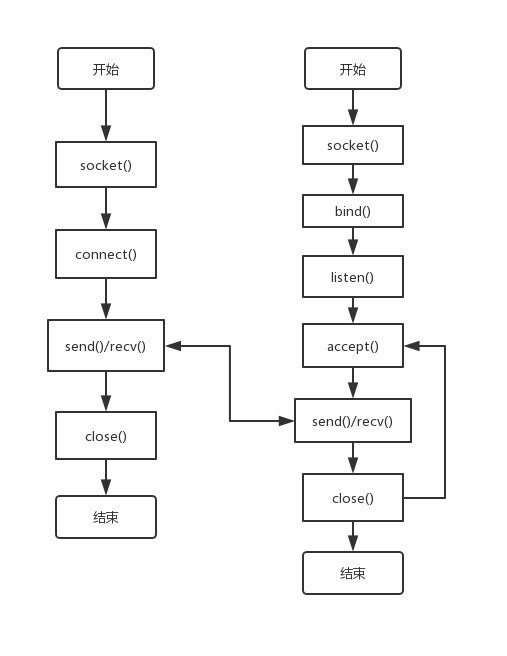
### 四、实验步骤

1、将210目标板与笔记本电脑用网线连接好，目标板与台式机用USB-串口线连接。将网络通讯程序客户端放入SD大卡中。

2、将SD大卡插入210目标板，用台式机对210发出命令，使210目标板执行网络通讯程序客户端，笔记本电脑端执行网络通讯程序客户端与服务端。

3、在笔记本电脑与210目标板上进行双方chat，测试可以通过网线进行网络通讯。

### 五、实验数据记录（或调试好的程序包括流程图）

TCP过程

//client.c

/\*

As a client, I will send ![REG]string to register a new ID,

![MSG]string to send userdata,

![SF]filepath to send a file,

![RF]filepath to receive a file,

![QUIT] to quit the chat room,

receive ![ACCREG] / ![REJREG] from the server to ensure the register.

I will receive string from server, the format is [time ID]\nuserdata.

While sending a file, I will send ![SF]filename first, then send the file.

While receiving a file, I will send ![RF]filename first, then receive the file until the buffer received is NULL.

\*/

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<string.h>

#include<unistd.h>

#include<arpa/inet.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<sys/stat.h>

#define TRUE 1

#define MAX\_USERDATA 8192 //max length of userdata string

#define MAX\_ID 1024 //max length of ID string

#define MAX\_TIME 20 //max length of time string

#define MAX\_READYTOSEND (MAX\_USERDATA + 6) //max length of ReadytoSend string

#define MAX\_RECVFROMSERV (MAX\_TIME + MAX\_ID + MAX\_USERDATA + 2) //max length of RecvfromServ string

#define MAX\_REG (MAX\_ID + 6) //max length of register string

#define MAX\_REGRESULT 10 //max length of register sendback string

#define MAX\_FILELEN (1024 \* 1024) //max length of file content

void\* th\_Recv(void \*);

struct sockaddr\_in SA; //save target address

int len\_SA; //length of SA

int fd\_server; //file descriptor of target server

pthread\_t thread\_RS, thread\_Snd;//thread IDs of recv and send

int mode\_protocol = -1; //0:TCP 1:UDP

void \*ret1 = NULL, \*ret2 = NULL; //return value of two threads

void\* th\_Send(void \*data)

{

char Userdata[MAX\_USERDATA] = { 0 }; //userdata to be filled

char ReadytoSend[MAX\_READYTOSEND] = { 0 }; //![REG]userdata

long len;

FILE \*fp = NULL; //file pointer of the file ready to be sent

struct stat FileInfo; //record file information

long ReadLen = 0; //length of file part read everytime

unsigned char Buf\_File[MAX\_READYTOSEND] = { 0 }; //buffer used to save content of file

//input string and send

while(TRUE)

{

scanf("%s", Userdata);

//client send quit, send quit msg and exit thread

if (!strncmp(Userdata, "![QUIT]", 7))

{

strcpy(ReadytoSend, Userdata);

//TCP

if (!mode\_protocol)

{

send(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

}

break;

}

//client send file

else if (!strncmp(Userdata, "![SF]", 5))

{

stat(Userdata + 5, &FileInfo);

if (FileInfo.st\_size > 1024 \* 1024)

{

printf("No larger than 1 MiB!\n");

}

else

{

//send ![SF]filename

strcpy(ReadytoSend, "![SF]");

strcat(ReadytoSend, strchr(Userdata + 5, '/') ? strchr(Userdata + 5, '/') + 1 : Userdata + 5);

//TCP

if (!mode\_protocol)

{

send(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

}

printf("Sending...\n");

//send file content

fp = fopen(Userdata + 5, "r");

memset(Buf\_File, 0, MAX\_READYTOSEND);

ReadLen = fread(Buf\_File, 1, MAX\_FILELEN, fp);

//TCP

if (!mode\_protocol)

{

send(fd\_server, Buf\_File, ReadLen, MSG\_NOSIGNAL);

send(fd\_server, Buf\_File, ReadLen, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, Buf\_File, ReadLen, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

sendto(fd\_server, Buf\_File, ReadLen, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

}

fclose(fp);

printf("OK\n");

}

}

//client send file request

else if (!strncmp(Userdata, "![RF]", 5))

{

strcpy(ReadytoSend, Userdata);

//TCP

if (!mode\_protocol)

{

send(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

}

}

//client send normal messages

else

{

strcpy(ReadytoSend, "![MSG]"); //Add ![REG] to ReadytoSend

strcat(ReadytoSend, Userdata); //connect userdata to ReadytoSend

}

//TCP

if (!mode\_protocol)

{

send(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

}

//clear buffers

memset(ReadytoSend, 0, MAX\_READYTOSEND);

memset(Userdata, 0, MAX\_USERDATA);

}

return 0;

}

void\* th\_Recv(void \*data)

{

char RecvfromServ[MAX\_RECVFROMSERV] = { 0 }; //formatted string received from the server

FILE \*fp = NULL; //file pointer of the file ready to be received

long ReadLen = 0; //length of file part written everytime

unsigned char Buf\_File[MAX\_RECVFROMSERV] = { 0 }; //buffer used to save content of file

while (TRUE)

{

//TCP

if (!mode\_protocol)

{

recv(fd\_server, RecvfromServ, MAX\_RECVFROMSERV, MSG\_NOSIGNAL);

}

//UDP

else

{

recvfrom(fd\_server, RecvfromServ, MAX\_RECVFROMSERV, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, &len\_SA);

}

if (!strncmp(RecvfromServ, "![QUIT]", 7))

{

break;

}

//server sendback success flag of receiving a file

else if (!strncmp(RecvfromServ, "![ACCRF]", 8))

{

//create file

fp = fopen(RecvfromServ + 8, "w");

//fill file content

//TCP

if (!mode\_protocol)

{

ReadLen = recv(fd\_server, Buf\_File, MAX\_RECVFROMSERV, MSG\_NOSIGNAL);

}

//UDP

else

{

ReadLen = recvfrom(fd\_server, Buf\_File, MAX\_RECVFROMSERV, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, &len\_SA);

}

//only ![ACCRF] header to save

if (!strncmp(Buf\_File, "![ACCRF]", 8))

fwrite(Buf\_File + 8, 1, ReadLen - 8, fp);

fclose(fp);

printf("File has been downloaded!\n");

}

//server sendback error flag of receiving a file

else if (!strncmp(RecvfromServ, "![REJRF]", 8))

{

printf("File received error!\n");

}

//server broadcast normal messages

else if (!strncmp(RecvfromServ, "[", 1))

{

printf("%s\n\n", RecvfromServ);

}

memset(RecvfromServ, 0, MAX\_RECVFROMSERV);

}

return 0;

}

int Register()

{

char str\_inputID[MAX\_REG] = { 0 };

char str\_REG[MAX\_REG] = { 0 }; //register string

char ReadytoSend[MAX\_READYTOSEND] = { 0 }; //readytosend string

char str\_REGresult[MAX\_REGRESULT] = { 0 }; //register sendback string

while (TRUE)

{

printf("Please register first\nformat:ID(no space, Max length=1023 Bytes):");

strcpy(str\_REG, "![REG]");

scanf("%s", str\_inputID);

strcat(str\_REG, str\_inputID);

strcpy(ReadytoSend, str\_REG);

//TCP

if (!mode\_protocol)

{

send(fd\_server, str\_REG, MAX\_REG, MSG\_NOSIGNAL);

recv(fd\_server, str\_REGresult, MAX\_REGRESULT, MSG\_NOSIGNAL);

}

//UDP

else

{

sendto(fd\_server, str\_REG, MAX\_REG, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, len\_SA);

recvfrom(fd\_server, str\_REGresult, MAX\_REGRESULT, MSG\_NOSIGNAL, (struct sockaddr \*)&SA, &len\_SA);

}

//get accept from server

if (!strncmp(str\_REGresult, "![ACCREG]", MAX\_REGRESULT - 1))

{

printf("Register successfully!\n");

break;

}

printf("\nRetry another ID to register!\n");

//clear buffers

memset(str\_REG, 0, MAX\_REG);

memset(str\_inputID, 0, MAX\_REG);

memset(ReadytoSend, 0, MAX\_READYTOSEND);

memset(str\_REGresult, 0, MAX\_REGRESULT);

}

return 0;

}

int TCPChatRoom()

{

fd\_server = socket(AF\_INET, SOCK\_STREAM, 0); //set file descriptor of TCP

connect(fd\_server, (struct sockaddr \*)&SA, len\_SA);

Register();

pthread\_create(&thread\_RS, NULL, th\_Recv, 0);

pthread\_create(&thread\_Snd, NULL, th\_Send, 0);

pthread\_join(thread\_Snd, &ret1); //wait for send thread to quit

pthread\_join(thread\_RS, &ret2); //wait for recv thread to quit

close(fd\_server);

return 0;

}

int UDPChatRoom()

{

pthread\_t thread\_RS, thread\_Snd;//thread IDs of recv and send

void \*ret1 = NULL, \*ret2 = NULL;

fd\_server = socket(AF\_INET, SOCK\_DGRAM, 0); //set file descriptor of UDP

Register();

pthread\_create(&thread\_RS, NULL, th\_Recv, 0);

pthread\_create(&thread\_Snd, NULL, th\_Send, 0);

pthread\_join(thread\_Snd, &ret1); //wait for send thread to quit

pthread\_join(thread\_RS, &ret2); //wait for recv thread to quit

return 0;

}

int main(int argc, char \*argv[])

{

char sel;

if (argc == 3)

{

memset(&SA,0,sizeof(SA));

SA.sin\_family = AF\_INET;

SA.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET,argv[1],&SA.sin\_addr);

len\_SA = sizeof(SA);

printf("In the ChatRoom:\n\"![SF][FilePath]\" to send a file.\n\"![RF][FileName]\" to recv a file and save to the current folder.\n\"![QUIT]\" to exit the chatroom\n1.TCP ChatRoom\n2.UDP ChatRoom\n");

do

{

sel = getchar();

getchar();

if (sel == '1')

{

mode\_protocol = 0;

TCPChatRoom();

continue;

}

else if (sel == '2')

{

mode\_protocol = 1;

UDPChatRoom();

continue;

}

system("clear");

}while (sel != '1' && sel != '2');

}

return 0;

}

//server.c

/\*

As a server, I will send ![ACCREG]/![REJREG] to accept or reject a register,

message string to client(format is [time ID]\nuserdata),

receive ![REG]stringID from the client,

![MSG]string to broadcast userdata,

![SF]filename to save the file from server,

![RF]filename to send the file to target client,

![QUIT] to close the connection between server and client.

\*/

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

#include<string.h>

#include<time.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<errno.h>

#define TRUE 1

#define MAX\_USERDATA 8192 //max length of userdata string

#define MAX\_ID 1024 //max length of ID string

#define MAX\_TIME 20 //max length of time string

#define MAX\_RECVFROMCLIENT (MAX\_USERDATA + 6) //max length of ReadytoSend string

#define MAX\_READYTOSEND (MAX\_TIME + MAX\_ID + MAX\_USERDATA + 2) //max length of RecvfromServ string

#define MAX\_REG (MAX\_ID + 6) //max length of register string

#define MAX\_REGRESULT 10 //max length o fregister sendback string

#define MAX\_CONNECT 100 //max user connected to this host

#define MAX\_FILELEN (1024 \* 1024) //max length of file content

//information of client link

typedef struct

{

//UDP

struct sockaddr\_in SA\_Client;//sock address info of client

int len\_SA\_Cient; //length of SA\_Client

//TCP

int fd\_client; //file descriptor of socket

char ID[1024]; //client ID

int flag\_protocol; //0:TCP 1:UDP

}ClientNode;

ClientNode ClientList[MAX\_CONNECT]; //the list record client information

int fd\_TCP, fd\_UDP; //file descriptor of TCP and UDP

struct sockaddr\_in SA\_server; //server address configuration

int len\_SA\_server; //length of SA\_server

//deal with the data received from the client

void\* th\_Proc(void \*data)

{

char RecvfromClient[MAX\_RECVFROMCLIENT] = { 0 }; //data received

char ReadytoSend[MAX\_READYTOSEND] = { 0 }; //formatted string ready to send to the client

int pos\_fd = \*(int\*)data; //index of clientlist

char ch;

int i;

time\_t t = 0;

struct tm \*stu\_t;

char str\_time[MAX\_TIME] = { 0 }; //string of formatted time

FILE \*fp = NULL; //file pointer of the file ready to be received

long ReadLen = 0; //length of file part written everytime

unsigned char Buf\_File[MAX\_RECVFROMCLIENT] = { 0 }; //buffer used to save content of file

unsigned char tBuf\_File[MAX\_RECVFROMCLIENT] = { 0 }; //buffer used to save content of file

//input string and send

while(TRUE)

{

recv(ClientList[pos\_fd].fd\_client, RecvfromClient, MAX\_RECVFROMCLIENT, MSG\_NOSIGNAL);

//receive quit from client

if (!strncmp(RecvfromClient, "![QUIT]", 7))

{

ClientList[pos\_fd].flag\_protocol = -1;

send(ClientList[pos\_fd].fd\_client, "![QUIT]", MAX\_READYTOSEND, MSG\_NOSIGNAL);

break;

}

//receive register request from client

else if (!strncmp(RecvfromClient, "![REG]", 6))

{

for (i = 0;i<MAX\_CONNECT;i++)

{

//not setted or different string length

if (ClientList[i].flag\_protocol == -1 || strlen(RecvfromClient) - 6 != strlen(ClientList[i].ID))

continue;

//have same ID, send reject message

if (!strcmp(RecvfromClient + 6, ClientList[i].ID))

{

strcpy(ReadytoSend, "![REJREG]");

break;

}

}

if (i == MAX\_CONNECT)

{

strcpy(ReadytoSend, "![ACCREG]");

strcpy(ClientList[pos\_fd].ID, RecvfromClient + 6);

}

send(ClientList[pos\_fd].fd\_client, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

//client send file

else if (!strncmp(RecvfromClient, "![SF]", 5))

{

//create file

fp = fopen(RecvfromClient + 5, "w");

//fill file content

memset(Buf\_File, 0, MAX\_READYTOSEND);

ReadLen = recv(ClientList[pos\_fd].fd\_client, Buf\_File, MAX\_FILELEN, MSG\_NOSIGNAL);

fwrite(Buf\_File, 1, ReadLen, fp);

fclose(fp);

}

//client receive file

else if (!strncmp(RecvfromClient, "![RF]", 5))

{

//open existed file

fp = fopen(RecvfromClient + 5, "r");

//open file from error

if (!fp)

send(ClientList[pos\_fd].fd\_client, "![REJRF]", MAX\_READYTOSEND, MSG\_NOSIGNAL);

else

{

while ((ReadLen = fread(tBuf\_File, MAX\_READYTOSEND - 10, 1, fp)) > 0)

{

memset(Buf\_File, 0, MAX\_READYTOSEND);

strcpy(Buf\_File, "![ACCRF]");

strcat(Buf\_File, tBuf\_File);

send(ClientList[pos\_fd].fd\_client, Buf\_File, ReadLen + 8, MSG\_NOSIGNAL);

memset(tBuf\_File, 0, MAX\_RECVFROMCLIENT);

}

fclose(fp);

}

}

//normal message

else if (!strncmp(RecvfromClient, "![MSG]", 6))

{

strcpy(ReadytoSend, "[");

//connect time string

time(&t);

stu\_t = localtime(&t);

strftime(str\_time, MAX\_TIME, "%Y-%m-%d %H:%M:%S", stu\_t);

strcat(ReadytoSend, str\_time);

strcat(ReadytoSend, " ");

//connect ID string

strcat(ReadytoSend, ClientList[pos\_fd].ID);

strcat(ReadytoSend, "]\n");

//connect userinput string

strcat(ReadytoSend, RecvfromClient + 6);

for (i = 0;i < MAX\_CONNECT;i++)

{

if (ClientList[i].flag\_protocol != -1)

send(ClientList[i].fd\_client, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL);

}

}

//clear every string buffer

memset(RecvfromClient, 0, MAX\_RECVFROMCLIENT);

memset(ReadytoSend, 0, MAX\_READYTOSEND);

memset(str\_time, 0, MAX\_TIME);

}

return 0;

}

int TCPChatRoom()

{

int tfd\_client; //temp file descriptor of client

pthread\_t thread\_proc[MAX\_CONNECT] = { 0 };//thread IDs of process info

struct sockaddr\_in tsa;

int tl = sizeof(struct sockaddr\_in);

int i;

if (listen(fd\_TCP, MAX\_CONNECT) == -1)

{

printf("Fail to listen: %s\n", strerror(errno));

return -1;

}

while (TRUE)

{

memset(&tsa, 0, sizeof(tsa));

tl = sizeof(tsa);

//new client

if ((tfd\_client = accept(fd\_TCP, (struct sockaddr \*)&tsa, &tl)) != -1)

{

//!!!dangerous!!!

//if number of conntion is large, fd may be wrong.

//find NULL connection

for (i = 0;i < MAX\_CONNECT;i++)

{

if (ClientList[i].flag\_protocol == -1)

{

ClientList[i].fd\_client = tfd\_client;

ClientList[i].flag\_protocol = 0;

break;

}

}

//connection number is not full

if (i != MAX\_CONNECT)

{

//one thread per client

pthread\_create(&thread\_proc[i], NULL, th\_Proc, (void \*)&i);

pthread\_detach(thread\_proc[i]);

}

}

}

return 0;

}

int UDPChatRoom()

{

char RecvfromClient[MAX\_RECVFROMCLIENT] = { 0 }; //receivefromclient string

char ReadytoSend[MAX\_READYTOSEND] = { 0 }; //the formatted string ready to send

pthread\_t thread\_proc[MAX\_CONNECT] = { 0 };//thread IDs of process info

struct sockaddr\_in tsa;

int tl = sizeof(struct sockaddr\_in);

int i;

time\_t t = 0;

struct tm \*stu\_t;

char str\_time[MAX\_TIME] = { 0 }; //string of formatted time

FILE \*fp = NULL; //file pointer of the file ready to be received

long ReadLen = 0; //length of file part written everytime

unsigned char Buf\_File[MAX\_RECVFROMCLIENT] = { 0 }; //buffer used to save content of file

unsigned char tBuf\_File[MAX\_RECVFROMCLIENT] = { 0 }; //buffer used to save content of file

while (TRUE)

{

memset(&tsa, 0, sizeof(tsa));

tl = sizeof(tsa);

recvfrom(fd\_UDP, RecvfromClient, MAX\_RECVFROMCLIENT, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, &tl);

//receive quit from client

if (!strncmp(RecvfromClient, "![QUIT]", 7))

{

sendto(fd\_UDP, "![QUIT]", MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, tl);

//set clientlist

for (i = 0;i < MAX\_CONNECT;i++)

{

if (ClientList[i].flag\_protocol != -1 && ClientList[i].len\_SA\_Cient == tl && !memcmp(&ClientList[i].SA\_Client, &tsa, tl))

{

ClientList[i].flag\_protocol = -1;

break;

}

}

}

else if (!strncmp(RecvfromClient, "![REG]", 6))

{

//check same ID

for (i = 0;i < MAX\_CONNECT;i++)

{

//not setted or different string length

if (ClientList[i].flag\_protocol == -1 || strlen(RecvfromClient) - 6 != strlen(ClientList[i].ID))

continue;

//have same ID, send reject message

if (!strcmp(RecvfromClient + 6, ClientList[i].ID))

{

strcpy(ReadytoSend, "![REJREG]");

break;

}

}

//no same ID

if (i == MAX\_CONNECT)

{

//check NULL client

for (i = 0;i < MAX\_CONNECT;i++)

{

//find a NULL client

if (ClientList[i].flag\_protocol == -1)

{

ClientList[i].len\_SA\_Cient = tl;

memcpy(&ClientList[i].SA\_Client, &tsa, tl);

ClientList[i].flag\_protocol = 1;

strcpy(ReadytoSend, "![ACCREG]");

strcpy(ClientList[i].ID, RecvfromClient + 6);

break;

}

}

//client number is full

if (i == MAX\_CONNECT)

{

strcpy(ReadytoSend, "![REJREG]");

}

}

sendto(fd\_UDP, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, tl);

}

//client send file

else if (!strncmp(RecvfromClient, "![SF]", 5))

{

//create file

fp = fopen(RecvfromClient + 5, "w");

//fill file content

memset(Buf\_File, 0, MAX\_READYTOSEND);

ReadLen = recvfrom(fd\_UDP, Buf\_File, MAX\_FILELEN, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, &tl);

fwrite(Buf\_File, 1, ReadLen, fp);

fclose(fp);

}

//client receive file

else if (!strncmp(RecvfromClient, "![RF]", 5))

{

//open existed file

fp = fopen(RecvfromClient + 5, "r");

//open file from error

if (!fp)

sendto(fd\_UDP, "![REJRF]", MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, tl);

else

{

while ((ReadLen = fread(tBuf\_File, MAX\_READYTOSEND - 10, 1, fp)) > 0)

{

memset(Buf\_File, 0, MAX\_READYTOSEND);

strcpy(Buf\_File, "![ACCRF]");

strcat(Buf\_File, tBuf\_File);

sendto(fd\_UDP, Buf\_File, ReadLen + 8, MSG\_NOSIGNAL, (struct sockaddr \*)&tsa, tl);

memset(tBuf\_File, 0, MAX\_RECVFROMCLIENT);

}

fclose(fp);

}

}

//normal message

else if (!strncmp(RecvfromClient, "![MSG]", 6))

{

strcpy(ReadytoSend, "[");

//connect time string

time(&t);

stu\_t = localtime(&t);

strftime(str\_time, MAX\_TIME, "%Y-%m-%d %H:%M:%S", stu\_t);

strcat(ReadytoSend, str\_time);

strcat(ReadytoSend, " ");

//connect ID string

for (i = 0;i < MAX\_CONNECT;i++)

{

if (ClientList[i].flag\_protocol != -1 && ClientList[i].len\_SA\_Cient == tl && !memcmp(&ClientList[i].SA\_Client, &tsa, tl))

{

strcat(ReadytoSend, ClientList[i].ID);

break;

}

}

strcat(ReadytoSend, "]\n");

//connect userinput string

strcat(ReadytoSend, RecvfromClient + 6);

for (i = 0;i < MAX\_CONNECT;i++)

{

if (ClientList[i].flag\_protocol != -1)

sendto(fd\_UDP, ReadytoSend, MAX\_READYTOSEND, MSG\_NOSIGNAL, (struct sockaddr \*)&ClientList[i].SA\_Client, ClientList[i].len\_SA\_Cient);

}

}

//clear buffer

memset(RecvfromClient, 0, MAX\_RECVFROMCLIENT);

memset(ReadytoSend, 0,MAX\_READYTOSEND);

}

return 0;

}

int InitClientList()

{

int i;

for (i = 0;i < MAX\_CONNECT;i++)

{

ClientList[i].fd\_client = 0;

memset(ClientList[i].ID, 0, MAX\_ID);

ClientList[i].flag\_protocol = -1;

ClientList[i].len\_SA\_Cient = sizeof(ClientList[i].SA\_Client);

memset(&ClientList[i].SA\_Client,0,len\_SA\_server);

}

return 0;

}

int main(int argc, char \*argv[])

{

pid\_t pid;

if (argc == 2)

{

InitClientList();

memset(&SA\_server,0,sizeof(SA\_server));

SA\_server.sin\_family = AF\_INET;

SA\_server.sin\_addr.s\_addr = htonl(INADDR\_ANY);

SA\_server.sin\_port = htons(atoi(argv[1]));

len\_SA\_server = sizeof(SA\_server);

fd\_TCP = socket(AF\_INET, SOCK\_STREAM, 0);

if (bind(fd\_TCP, (struct sockaddr \*)&SA\_server, len\_SA\_server) == -1)

{

printf("Fail to bind: %s\n", strerror(errno));

return -1;

}

fd\_UDP = socket(AF\_INET, SOCK\_DGRAM, 0);

if (bind(fd\_UDP, (struct sockaddr \*)&SA\_server, len\_SA\_server) == -1)

{

printf("Fail to bind: %s\n", strerror(errno));

return -1;

}

pid = fork();

//fail to fork

if (pid < 0)

exit(-1);

//child process, manage TCP

if (pid == 0)

{

TCPChatRoom();

}

//parent process, manage UDP

else

{

UDPChatRoom();

}

}

return 0;

}

### 六、结论

socket通讯方便快捷，成功实现210目标板与笔记本电脑间的网络通讯。