**网络协议编程实验指导书**

**实验二 UDP协议编程**

1. 实验学习目标

了解UDP协议，使用UDP协议进行程序设计，熟悉UDP的编程框架。实现以下功能。

1. 实验功能

1.udp截断测试(要求改写成客户端服务器端的形式)

2.nat及先启动udp客户端 再启动udp服务器端，通讯测试

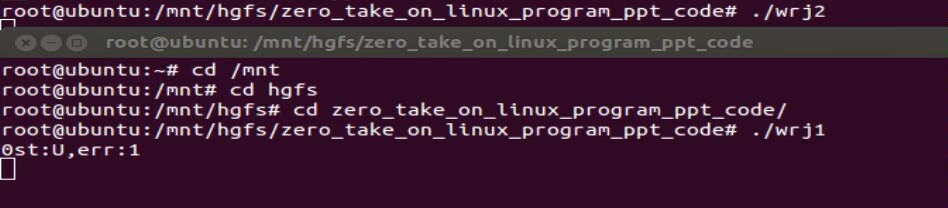
3.在udp中使用connect函数

4.可靠的基于udp的文件传输客户端、服务器端

1. 实验完成功能(效果)截图展示

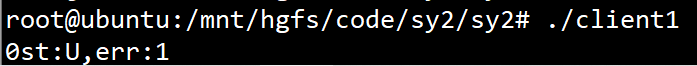
功能1 udp截断测试 截图展示

王如瑾

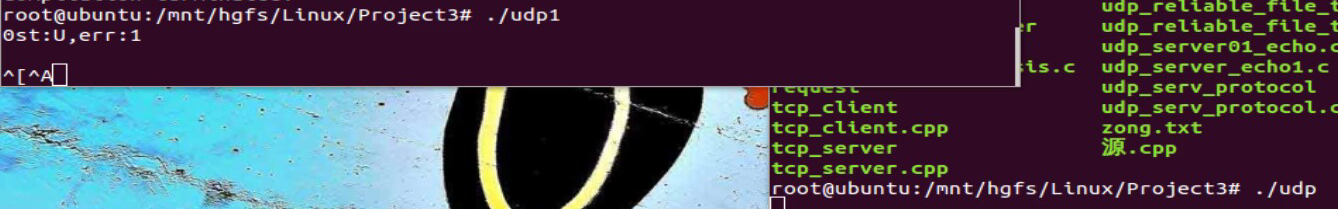


王诗婷

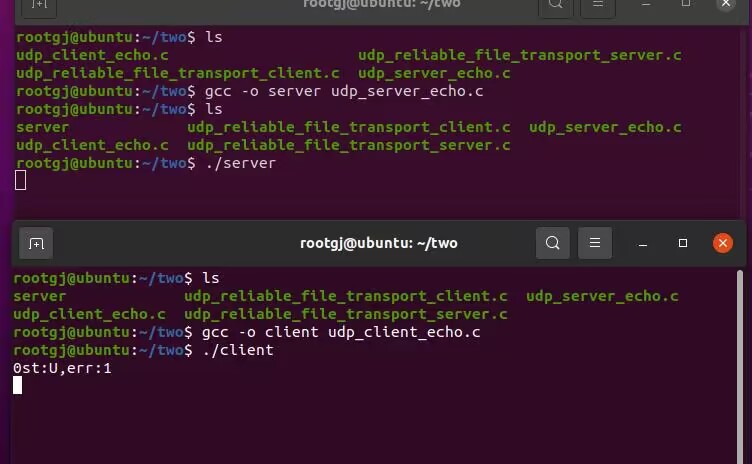




周英红



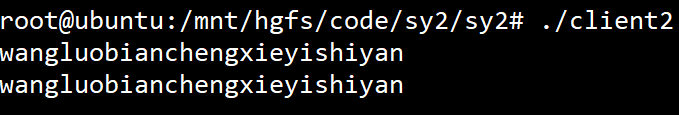
唐圪菁



功能2 nat及先启动udp客户端 再启动udp服务器端，通讯测试 截图展示

周英红

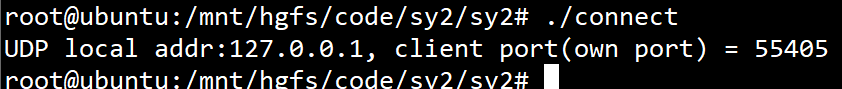






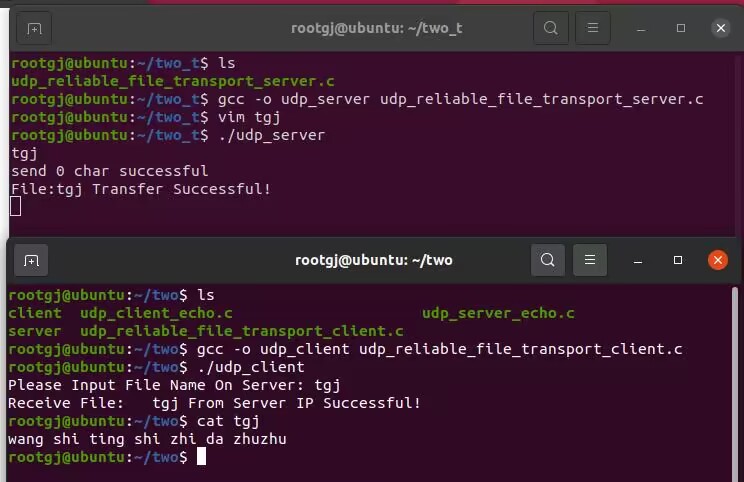
功能3在udp中使用connect函数 截图展示

王诗婷

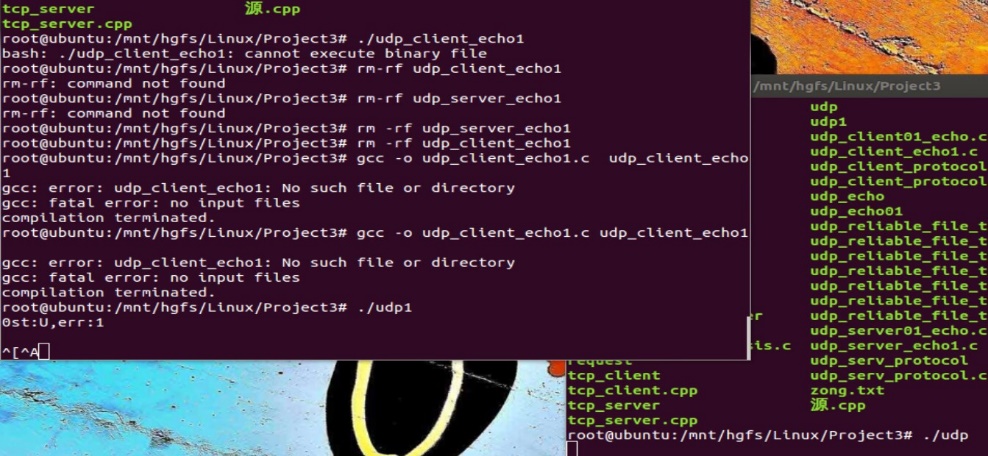


功能4 可靠的基于udp的文件传输客户端、服务器端 截图展示

唐圪菁



王如瑾



1. 关键实验步骤及关键代码（必须能按步骤重现，不含工程新建过程）

功能1 udp截断测试 实现步骤:

服务器Server：

#include <sys/types.h>

#include <sys/socket.h> /\*包含socket()/bind()\*/

#include <netinet/in.h> /\*包含struct sockaddr\_in\*/

#include <string.h> /\*包含memset()\*/

#define PORT\_SERV 5555/\*服务器端口\*/

#define BUFF\_LEN 256 /\*缓冲区大小\*/

void static udpserv\_echo(int s, struct sockaddr\* client)

{

int n; /\*接收数据长度\*/

char buff[BUFF\_LEN]; /\*接收发送缓冲区 \*/

socklen\_t len; /\*地址长度\*/

while (1) /\*循环等待\*/

{

len = sizeof(\*client);

n = recvfrom(s, buff, BUFF\_LEN, 0, client, &len);

/\*接收数据放到buff中，并获得客户端地址\*/

sendto(s, buff, n, 0, client, len);/\*将接收到的n个字节发送回客户 端\*/

}

}

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

{

int s; /\*套接字文件描述符\*/

struct sockaddr\_in addr\_serv, addr\_clie; /\*地址结构\*/

s = socket(AF\_INET, SOCK\_DGRAM, 0); /\*建立数据报套接字\*/

memset(&addr\_serv, 0, sizeof(addr\_serv)); /\*清空地址结构\*/

addr\_serv.sin\_family = AF\_INET; /\*地址类型为AF\_INET\*/

addr\_serv.sin\_addr.s\_addr = htonl(INADDR\_ANY); /\*任意本地地址\*/

addr\_serv.sin\_port = htons(PORT\_SERV); /\*服务器端口\*/

bind(s, (struct sockaddr\*)&addr\_serv, sizeof(addr\_serv));

/\*绑定地址\*/

udpserv\_echo(s, (struct sockaddr\*)&addr\_clie); /\*回显处理程序\*/

return 0;

}

客户端Client：

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/socket.h> /\*包含socket()/bind()\*/

#include <netinet/in.h> /\*包含struct sockaddr\_in\*/

#include <string.h> /\*包含memset()\*/

#define PORT\_SERV 5555 /\*服务器端口\*/

#define BUFF\_LEN 256 /\*缓冲区大小\*/

static void udpclie\_echo(int s, struct *sockaddr*\* to)

{

char buff[BUFF\_LEN] = "UDP TEST"; /\*发送给服务器的测试数据05 \*/

struct sockaddr\_in from; /\*服务器地址\*/

*socklen\_t* len = sizeof(\*to); /\*地址长度\*/

*sendto*(s, buff, BUFF\_LEN, 0, to, len); /\*发送给服务器\*/

int i = 0;

for (i = 0; i < 16; i++) {

memset(buff, 0, BUFF\_LEN);

int err = *recvfrom*(s, buff, 1, 0, (struct *sockaddr*\*)&from, &len);

*printf*("%dst:%c,err:%d\n", i, buff[0], err);

}//循环截断

//recvfrom(s, buff, BUFF\_LEN, 0, (struct sockaddr\*)&from, &len);

/\*从服务器接收数据\*/

*printf*("recved:%s\n", buff); /\*打印数据\*/

}

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

{

int s; /\*套接字文件描述符\*/

struct sockaddr\_in addr\_serv; /\*地址结构\*/

s = *socket*(*AF\_INET*, *SOCK\_DGRAM*, 0); /\*建立数据报套接字\*/

memset(&addr\_serv, 0, sizeof(addr\_serv)); /\*清空地址结构\*/

addr\_serv.sin\_family = *AF\_INET*; /\*地址类型为AF\_INET\*/

addr\_serv.sin\_addr.s\_addr = htonl(INADDR\_ANY); /\*任意本地地址\*/

addr\_serv.sin\_port = htons(PORT\_SERV); /\*服务器端口\*/

// sleep(10);

udpclie\_echo(s, (struct *sockaddr*\*)&addr\_serv); /\*客户端回显程序\*/

*close*(s);

return 0;

}

功能2 nat及先启动udp客户端 再启动udp服务器端，通讯测试 实现步骤:

客户端Client：

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

#define ERR\_EXIT(m) \

do \

{ \

*perror*(m); \

*exit*(*EXIT\_FAILURE*); \

} while( 0)

void echo\_cli(int sock)

{

struct sockaddr\_in servaddr;

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

servaddr.sin\_family = *AF\_INET*;

servaddr.sin\_port = htons(5188);

servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

int ret;

char sendbuf[1024] = { 0 };

char recvbuf[1024] = { 0 };

*connect*(sock, (struct *sockaddr*\*)&servaddr, sizeof(servaddr));

while (*fgets*(sendbuf, sizeof(sendbuf), *stdin*) != *NULL*)

{

*sendto*(sock, sendbuf, strlen(sendbuf), 0, (struct *sockaddr*\*)&servaddr, sizeof(servaddr));

ret = *recvfrom*(sock, recvbuf, sizeof(recvbuf), 0, *NULL*, *NULL*);

if (ret == -1)

{

if (*errno* == *EINTR*)

continue;

ERR\_EXIT("recvfrom");

}

*fputs*(recvbuf, *stdout*);

memset(sendbuf, 0, sizeof(sendbuf));

memset(recvbuf, 0, sizeof(recvbuf));

}

*close*(sock);

}

int main(void)

{

int sock;

if ((sock = *socket*(*PF\_INET*, *SOCK\_DGRAM*, 0)) < 0)

ERR\_EXIT("socket");

echo\_cli(sock);

return 0;

}

服务器Server：

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<errno.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<string.h>

#define ERR\_EXIT(m) \

do { \

*perror*(m); \

*exit*(*EXIT\_FAILURE*); \

} while ( 0)

void echo\_ser(int sock)

{

char recvbuf[1024] = { 0 };

struct sockaddr\_in peeraddr;

*socklen\_t* peerlen;

int n;

while (1)

{

peerlen = sizeof(peeraddr);

memset(recvbuf, 0, sizeof(recvbuf));

n = *recvfrom*(sock, recvbuf, sizeof(recvbuf), 0,

(struct *sockaddr*\*)&peeraddr, &peerlen);

if (n == -1)

{

if (*errno* == *EINTR*)

continue;

ERR\_EXIT("recvfrom error");

}

else if (n > 0)

{

*fputs*(recvbuf, *stdout*);

*sendto*(sock, recvbuf, n, 0,

(struct *sockaddr*\*)&peeraddr, peerlen);

}

}

*close*(sock);

}

int main(void)

{

int sock;

if ((sock = *socket*(*PF\_INET*, *SOCK\_DGRAM*, 0)) < 0)

ERR\_EXIT("socket error");

struct sockaddr\_in servaddr;

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

servaddr.sin\_family = *AF\_INET*;

servaddr.sin\_port = htons(5188);

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

if (*bind*(sock, (struct *sockaddr*\*)&servaddr, sizeof(servaddr)) < 0)

ERR\_EXIT("bind error");

echo\_ser(sock);

return 0;

}

功能3 在udp中使用connect函数 实现步骤:

#include <sys/types.h>

#include <sys/socket.h> /\*socket()/bind()\*/

#include <netinet/in.h> /\*struct sockaddr\_in\*/

#include <string.h> /\*memset()\*/

#include <stdio.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT\_SERV 8888

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

{

int s; /\*套接字文件描述符\*/

struct sockaddr\_in addr\_serv; /\*服务器地址\*/

struct sockaddr\_in local; /\*本地地址\*/

*socklen\_t* len = sizeof(local); /\*地址长度\*/

s = *socket*(*AF\_INET*, *SOCK\_DGRAM*, 0); /\*生成数据报套接字\*/

/\*填充服务器地址\*/

memset(&addr\_serv, 0, sizeof(addr\_serv)); /\*清零\*/

addr\_serv.sin\_family = *AF\_INET*; /\*AF\_INET协议族\*/

addr\_serv.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); /\*地址为127.0.0.1\*/

addr\_serv.sin\_port = htons(PORT\_SERV); /\*服务器端口\*/

*connect*(s, (struct *sockaddr*\*)&addr\_serv, sizeof(addr\_serv));

/\*连接服务器\*/

*getsockname*(s, (struct *sockaddr*\*)&local, &len); /\*获得套接字文件描述符的地址\*/

*printf*("UDP local addr:%s, client port(own port) = %d\n", inet\_ntoa(local.sin\_addr), ntohs(local.sin\_port));

/\*打印获得的地址\*/

*close*(s);

return 0;

}

功能4 可靠的基于udp的文件传输客户端、服务器端 实现步骤:

服务器Server：

#include<sys/types.h>

#include<sys/socket.h>

#include<unistd.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<netdb.h>

#include<stdarg.h>

#include<string.h>

#define SERVER\_PORT 8000

#define BUFFER\_SIZE 1024

#define FILE\_NAME\_MAX\_SIZE 512

/\* 包头 \*/

typedef struct

{

int id;

int buf\_size;

}PackInfo;

/\* 接收包 \*/

struct SendPack

{

PackInfo head;

char buf[BUFFER\_SIZE];

} data;

int main()

{

/\* 发送id \*/

int send\_id = 0;

/\* 接收id \*/

int receive\_id = 0;

/\* 创建UDP套接口 \*/

struct sockaddr\_in server\_addr;

bzero(&server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

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

server\_addr.sin\_port = htons(SERVER\_PORT);

/\* 创建socket \*/

int server\_socket\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (server\_socket\_fd == -1)

{

perror("Create Socket Failed:");

exit(1);

}

/\* 绑定套接口 \*/

if (-1 == (bind(server\_socket\_fd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr))))

{

perror("Server Bind Failed:");

exit(1);

}

/\* 数据传输 \*/

while (1)

{

/\* 定义一个地址，用于捕获客户端地址 \*/

struct sockaddr\_in client\_addr;

socklen\_t client\_addr\_length = sizeof(client\_addr);

/\* 接收数据 \*/

char buffer[BUFFER\_SIZE];

bzero(buffer, BUFFER\_SIZE);

if (recvfrom(server\_socket\_fd, buffer, BUFFER\_SIZE, 0, (struct sockaddr\*)&client\_addr, &client\_addr\_length) == -1)

{

perror("Receive Data Failed:");

exit(1);

}

/\* 从buffer中拷贝出file\_name \*/

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

bzero(file\_name, FILE\_NAME\_MAX\_SIZE + 1);

strncpy(file\_name, buffer, strlen(buffer) > FILE\_NAME\_MAX\_SIZE ? FILE\_NAME\_MAX\_SIZE : strlen(buffer));

printf("%s\n", file\_name);

/\* 打开文件 \*/

FILE \*fp = fopen(file\_name, "r");

if (NULL == fp)

{

printf("File:%s Not Found.\n", file\_name);

}

else

{

int len = 0;

/\* 每读取一段数据，便将其发给客户端 \*/

while (1)

{

PackInfo pack\_info;

if (receive\_id == send\_id)

{

++send\_id;

if ((len = fread(data.buf, sizeof(char), BUFFER\_SIZE, fp)) > 0)

{

data.head.id = send\_id; /\* 发送id放进包头,用于标记顺序 \*/

data.head.buf\_size = len; /\* 记录数据长度 \*/

if (sendto(server\_socket\_fd, (char\*)&data, sizeof(data), 0, (struct sockaddr\*)&client\_addr, client\_addr\_length) < 0)

{

perror("Send File Failed:");

break;

}

/\* 接收确认消息 \*/

recvfrom(server\_socket\_fd, (char\*)&pack\_info, sizeof(pack\_info), 0, (struct sockaddr\*)&client\_addr, &client\_addr\_length);

receive\_id = pack\_info.id;

}

else

{

break;

}

}

else

{

/\* 如果接收的id和发送的id不相同,重新发送 \*/

if (sendto(server\_socket\_fd, (char\*)&data, sizeof(data), 0, (struct sockaddr\*)&client\_addr, client\_addr\_length) < 0)

{

perror("Send File Failed:");

break;

}

/\* 接收确认消息 \*/

recvfrom(server\_socket\_fd, (char\*)&pack\_info, sizeof(pack\_info), 0, (struct sockaddr\*)&client\_addr, &client\_addr\_length);

receive\_id = pack\_info.id;

}

}

/\* 关闭文件 \*/

if(sendto(server\_socket\_fd, (char\*)&data, 0, 0, (struct sockaddr\*)&client\_addr, client\_addr\_length) < 0)

{

printf("Send 0 char Failed:\n");

break;

}

printf("send 0 char successful\n");

fclose(fp);

printf("File:%s Transfer Successful!\n", file\_name);

}

}

close(server\_socket\_fd);

return 0;

}

客户端Client：

#include<sys/types.h>

#include<sys/socket.h>

#include<unistd.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<netdb.h>

#include<stdarg.h>

#include<string.h>

#define SERVER\_PORT 8000

#define BUFFER\_SIZE 1024

#define FILE\_NAME\_MAX\_SIZE 512

/\* 包头 \*/

typedef struct

{

int id;

int buf\_size;

}PackInfo;

/\* 接收包 \*/

struct RecvPack

{

PackInfo head;

char buf[BUFFER\_SIZE];

} data;

int main()

{

int id = 1;

/\* 服务端地址 \*/

struct sockaddr\_in server\_addr;

bzero(&server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(SERVER\_PORT);

socklen\_t server\_addr\_length = sizeof(server\_addr);

/\* 创建socket \*/

int client\_socket\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_socket\_fd < 0)

{

perror("Create Socket Failed:");

exit(1);

}

/\* 输入文件名到缓冲区 \*/

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

bzero(file\_name, FILE\_NAME\_MAX\_SIZE + 1);

printf("Please Input File Name On Server: ");

scanf("%s", file\_name);

char buffer[BUFFER\_SIZE];

bzero(buffer, BUFFER\_SIZE);

strncpy(buffer, file\_name, strlen(file\_name)>BUFFER\_SIZE ? BUFFER\_SIZE : strlen(file\_name));

/\* 发送文件名 \*/

if (sendto(client\_socket\_fd, buffer, BUFFER\_SIZE, 0, (struct sockaddr\*)&server\_addr, server\_addr\_length) < 0)

{

perror("Send File Name Failed:");

exit(1);

}

/\* 打开文件，准备写入 \*/

FILE \*fp = fopen(file\_name, "w");

if (NULL == fp)

{

printf("File:\t%s Can Not Open To Write\n", file\_name);

exit(1);

}

/\* 从服务器接收数据，并写入文件 \*/

int len = 0;

while (1)

{

PackInfo pack\_info;

printf("read to receive file ....");

if ((len = recvfrom(client\_socket\_fd, (char\*)&data, sizeof(data), 0, (struct sockaddr\*)&server\_addr, &server\_addr\_length)) > 0)

{

if (data.head.id == id)

{

pack\_info.id = data.head.id;

pack\_info.buf\_size = data.head.buf\_size;

++id;

/\* 发送数据包确认信息 \*/

if (sendto(client\_socket\_fd, (char\*)&pack\_info, sizeof(pack\_info), 0, (struct sockaddr\*)&server\_addr, server\_addr\_length) < 0)

{

printf("Send confirm information failed!");

}

/\* 写入文件 \*/

if (fwrite(data.buf, sizeof(char), data.head.buf\_size, fp) < data.head.buf\_size)

{

printf("File:\t%s Write Failed\n", file\_name);

break;

}

}

else if (data.head.id < id) /\* 如果是重发的包 \*/

{

pack\_info.id = data.head.id;

pack\_info.buf\_size = data.head.buf\_size;

/\* 重发数据包确认信息 \*/

if (sendto(client\_socket\_fd, (char\*)&pack\_info, sizeof(pack\_info), 0, (struct sockaddr\*)&server\_addr, server\_addr\_length) < 0)

{

printf("Send confirm information failed!");

}

}

else

{

}

}

else

{

break;

}

}

printf("Receive File:\t%s From Server IP Successful!\n", file\_name);

fclose(fp);

close(client\_socket\_fd);

return 0;

}

1. 实验总结（回顾关键要点、易犯错误之处）

1、服务器与客户端要进行配合，接收的缓冲区要比发送的数据大一些，防止造成报文截断的可能。

2、UDP的socket允许多个socket绑定到同一个本地端口，即打洞，通过打洞能互相获取对方发送的数据。通过UDP与服务器通讯后，NAT网关默认会做一个外网IP和端口号与内网IP和端口号的映射。

3、Connect()的功能时将套接字文件描述符与一个网络地址结构进行绑定，在地址结构中所设置的值是发送接受数据时套接字采用的IP地址和端口。

4、编程中需注意ip地址问题，即客户端所对应的服务器端地址，如程序内IP地址未改，会导致客户端无法连接服务器，从而无法获取数据。服务器发送数据后应再发送一次数据将缓冲区数据推出，否则数据在缓冲区没有发至客户端，导致数据丢失。