电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220304031

姓 名 邓巧

（实验）课程名称 嵌入式网络编程

理论教师 兰刚

实验教师 冯月

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：邓巧 学号：2016220304031 指导教师：冯月**

**实验地点：信软楼304 实验时间：2018.10.18**

1. 实验名称：

Linux 平台下的基于 UDP 的文本聊天程序

1. 实验学时：4
2. 实验目的：

·掌握Linux下的Socket编程相关的基本数据结构；

·掌握UDP数据通信的函数使用；

·实现基于UDP数据传输的文本聊天程序。

1. 实验原理：
2. 套接字地址结构类型

struct sockaddr {

unsigned short sa\_family;/\*地址族：AF\_INET\*/

char sa\_data[14];/\*协议地址：IP地址和端口号\*/

};

1. 套接字地址结构类型

struct in\_addr {

in\_addr\_t s\_addr;/\*IP地址,in\_addr\_t一般为32位的unsigned long\*/

};

1. 套接字地址结构类型

struct sockaddr\_in {

short int sin\_family; /\*地址族：AF\_INET\*/

unsigned short int sin\_port;/\*端口号\*/

struct in\_addr sin\_addr;/\*IP地址\*/

unsigned char sin\_zero[8];/\*零数据（bzero或memset设置）\*/

};

1. 主机转网络：

uint16\_t htons(uint16\_t hostshort)

uint32\_t htonl(uint32\_t hostlong)

1. 网络转主机：

uint16\_t ntohs(uint16\_t netshort)

uint32\_t ntohl(uint32\_t netlong)

1. IP地址网络字节和字符串转换函数

inet\_addr()、inet\_aton或inet\_pton();/\*将IP字符串转换成网络字节序的二进制\*/

inet\_ntoa()或inet\_ntop();/\*将网络地址转换成IP字符串\*/

1. 创建套接字函数

格式：

int socket（int domain， int type，int protocol）

功能：根据给定的参数创建一个套接字描述子，成功返回一个套接字描述符，不成功返回-1。

参数：

domain指示网络协议族，AF\_INET或AF\_INET6标识TCP/IP协议；

type指示协议类型，如SOCK\_STREAM（TCP）、SOCK\_DRAM（UDP）和SOCK\_RAW；

protocol指示特定的协议类型，type为SOCK\_STREAM和SOCK\_DGRAM时protocol设置为0，type为SOCK\_RAW时可设置不同的值（IPPROTO\_IP、IPPROTO\_ICMP等）

1. 绑定套接字函数

格式：

int bind（int sockfd, const struct sockaddr \*addr, socklen\_t addrlen）

功能：绑定IP地址和端口号到套接字sockfd。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

addr指示本地的套接字地址；

addrlen指示套接字地址长度；

1. 发送数据套接字函数

格式：

int sendto（int sockfd, const void \*buf, size\_t len, int flags, const struct sockaddr \*dest\_addr, socklen\_t addrlen）

功能：向指定的目的地址发送数据。成功返回发送的字符数，否则返回-1。

参数：

sockfd指示套接字描述符；

buf要发送的数据缓冲区；

len指示要发送数据的长度；

flags指示发送数据的属性，典型的MSG\_OOB指示带外数据发送；

dest\_addr和addrlen分别指示目的地址和目的地址的长度。

1. 接收数据套接字函数

格式：

ssize\_t recvfrom（int sockfd, void \*buf, size\_t len, int flags, struct sockaddr \*src\_addr, socklen\_t \*addrlen）

功能：接收指定套接字的数据。成功返回接收的字节数，否则返回-1。

参数：

sockfd指示套接字描述符；

buf用于存放接收数据的缓冲区；

len指示接收数据的长度；

flags指示接收数据的属性，典型的MSG\_OOB指示接收带外数据；

src\_addr和addrlen分别指示接收数据的来源地址及其地址长度

1. 关闭套接字函数

格式：

int close（int sockfd）

功能：关闭网络套接字。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

1. 实验内容：

文本聊天程序的双方分别是服务器方和客户方

1. 服务器方

·以命令行的方式启动，要求输入参数：服务器端口号。

·服务器程序建立等待接收数据的套接字，等待客户的连接。要求服务器可以连续发送多条文本信息，而不用等待收到客户端的信息后，再发送。服务器也可以连续接收客户端的多条文本信息，而不用等待服务器键盘输入后再接收。

·服务器如果收到客户方的exit文本，则退出服务器程序。

·服务器端程序收到从键盘输入的shutdown文本，也退出程序。

1. 客户方

·以命令行的方式启动，命令行要求有服务器地址，服务器端口号。

·建立套接字，可以输入文本与服务器进行通信。要求客户端可以连续发送多条文本信息，而不用等待收到服务器端的信息后再发送。客户端也可以连续接收服务器的多条文本信息，而不用等待客户端键盘输入后再接收。

·客户方通过输入exit文本结束服务器的通信并关闭服务器。

·客户方通过输入shutdown文本结束与服务器的通信并退出客户端程序。

六、 实验器材（设备、元器件）：

Linux 操作系统、带有 C 语言编译插件的 Eclipse 开发工具。

七、 实验步骤：



图7-1 实验原理图

八、 实验结果与分析（含重要数据结果分析或核心代码流程分析）

1. server.c

1.1流程图

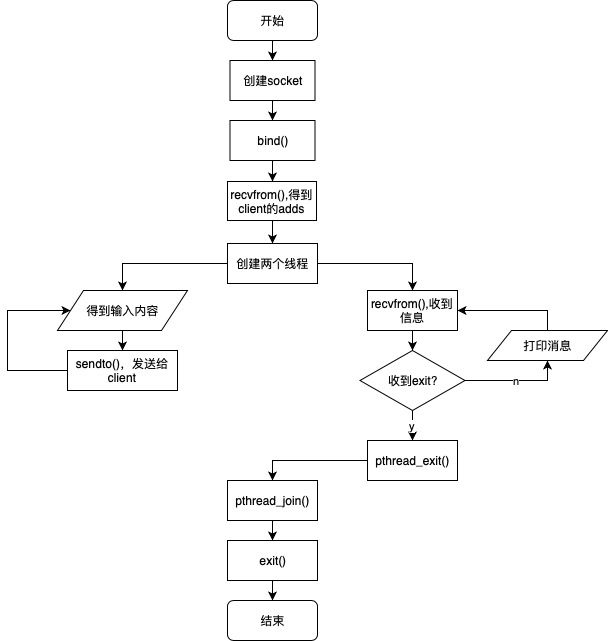


图8-1 Sever.c 流程图

1.2 代码

/\*邓巧 2016220304031\*/

/\*date: 18-10-17\*/

/\*A simple udp chat program,\*/

/\*Server.c\*/

#include<sys/socket.h>

#include<stdio.h>

#include<string.h>

#include<arpa/inet.h>

#include<stdlib.h>

#include <unistd.h>

#include<pthread.h>

#include <signal.h>

#define BUFF\_SIZE 100 //the size of the input and output buffer

struct sockaddr\_in server\_addr,client\_addr;

int addr\_len = sizeof(struct sockaddr\_in);

int serverfd,clientfd;

int get\_line(char \*buffer,int maxlen);

char exit\_message[] = "exit\n";

void\* recv\_message(void\* socketfd);

void\* send\_message(void\* socketfd);

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

void \*retval;

char line[BUFF\_SIZE];

pthread\_t send\_pid,recv\_pid;

char send\_buffer[BUFF\_SIZE];

char recev\_buffer[BUFF\_SIZE];

char ip\_addr[] = "0.0.0.0"; //the default ip address of server

socklen\_t len = sizeof(struct sockaddr);

//if user don’t type the port number,exit

if(!argv[1]){

perror("Lack port number!\n");

exit(EXIT\_FAILURE);

}

//create a socket

if((serverfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1){

perror("Failed to get socketfd!\n");

exit(EXIT\_FAILURE);

}

//initialize

memset(&server\_addr, 0, sizeof(server\_addr));

//the server addr

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

server\_addr.sin\_port = htons(atoi(argv[1]));

server\_addr.sin\_addr.s\_addr = inet\_addr(ip\_addr);

//bind the serverfd to the addr

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

perror("Failed to bind!\n");

exit(EXIT\_FAILURE);

}

//get the client addr

if((recvfrom(serverfd,(void \*)recev\_buffer,

sizeof(recev\_buffer),0,

(struct sockaddr \*)&client\_addr,(socklen\_t \*)&addr\_len)) > 0) {

printf("Connected to client...\n");

printf("Receive from client: %s",recev\_buffer);

memset(recev\_buffer,0,BUFF\_SIZE);

}

//create two pthread to recv and send at the same time

if(pthread\_create(&send\_pid,NULL,send\_message,(void \*)&client\_addr) != 0){

perror("Failed to create send thread!\n");

exit(EXIT\_FAILURE);

}

if(pthread\_create(&recv\_pid,NULL,recv\_message,(void \*)&client\_addr) != 0)

perror("Failed to create recv thread!\n");

pthread\_join(recv\_pid,&retval);

//the recv thread end,so close the socket and exit the process,

printf("Server closed.\n");

close(serverfd);

exit(EXIT\_SUCCESS);

}

//a safe way to get message consisting blank spaces

int get\_line(char \*line, int max) {

char c;

int len;

for (len = 0; len < max && (c = getchar()) != EOF && c != '\n'; ++len)

line[len] = c;

if (c == '\n') {

line[len] = c;

len++;

}

line[len] = '\0';

return len;

}

//recv thread. If get the message “exit” then exit the thread

void\* recv\_message(void\* client\_addr){

void \*retval = NULL;

char recev\_buffer[BUFF\_SIZE];

memset(recev\_buffer,0,BUFF\_SIZE);

while(1){

if(recvfrom(serverfd,(void \*)recev\_buffer,

sizeof(recev\_buffer),0,

(struct sockaddr \*)client\_addr,(socklen\_t \*)&addr\_len) > 0){

if(strcmp(recev\_buffer,exit\_message) == 0){

pthread\_exit(retval);

printf("exit\n");

}

printf("Receive from client: %s",recev\_buffer);

memset(recev\_buffer,0,BUFF\_SIZE);

}

}

}

//the send thread

void\* send\_message(void\* client\_addr){

char send\_buffer[BUFF\_SIZE];

memset(send\_buffer,0,BUFF\_SIZE);

if(pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL) != 0)

perror("Failed to set cancel!\n");

while(1){

if(get\_line(send\_buffer,BUFF\_SIZE) > 0){

sendto(serverfd,send\_buffer,strlen(send\_buffer),0,(struct sockaddr \*)client\_addr,addr\_len);

memset(send\_buffer,0,BUFF\_SIZE);

}

pthread\_testcancel();

}

}

2. clent.c

2.1 流程图

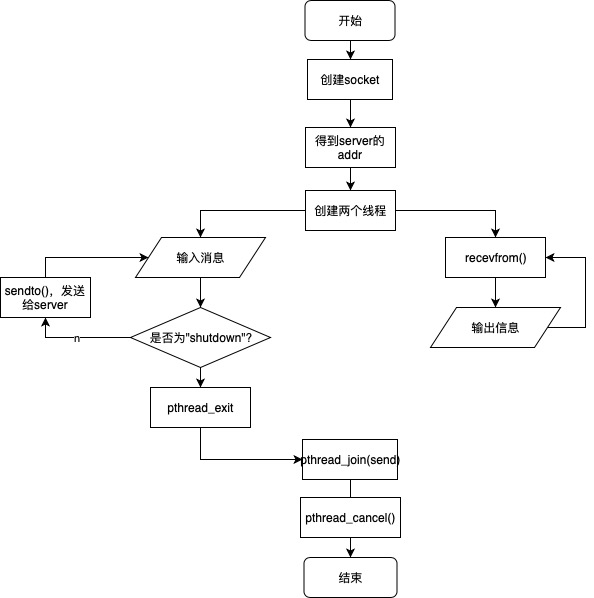


图8-2 client流程图

2.2 代码

/\*邓巧 2016220304031\*/

/\*date: 18-9-26\*/

/\*A simple udp chat program\*/

/\*Client.c\*/

#include<string.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<stdio.h>

#include<arpa/inet.h>

#include<unistd.h>

#include<pthread.h>

#include<signal.h>

#define BUFF\_SIZE 100

int get\_line(char \*buffer,int maxlen);

char exit\_message[] = "exit\n";

char shutdown\_message[] = "shutdown\n";

void\* send\_message(void\* socketfd);

void\* recv\_message(void\* socketfd);

int serverfd,clientfd;

struct sockaddr\_in server\_addr;

socklen\_t addr\_len = sizeof(struct sockaddr);

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

void\* ret;

pthread\_t send\_pid,recv\_pid;

//if the user didn’t type the correct arguments,then exit

if(argc != 3){

perror("Lack port and address!\n");

exit(EXIT\_FAILURE);

}

char send\_buffer[BUFF\_SIZE];

char recev\_buffer[BUFF\_SIZE];

//create the socket

if((clientfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1){

perror("Failed to get socketfd!\n");

exit(EXIT\_FAILURE);

}

memset(&server\_addr, 0, sizeof(server\_addr));

//create the server’s addr

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

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

server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

//create two thread so that can both send and recev

if(pthread\_create(&send\_pid,NULL,send\_message,(void \*)&clientfd) != 0){

perror("Failed to create send pthread!\n");

exit(EXIT\_FAILURE);

}

if(pthread\_create(&recv\_pid,NULL,recv\_message,(void \*)&clientfd) != 0){

perror("Failed to create recv pthread!\n");

}

pthread\_join(send\_pid,&ret);

//if the user type shutdown, then the send thread exit. Then cancel the

//recv thread

if(pthread\_cancel(recv\_pid) != 0){

perror("Failed to cancel thread!\n");

}

pthread\_join(recv\_pid,&ret);

printf("Conversation closed.\n");

close(clientfd);

exit(EXIT\_SUCCESS);

}

//a safe way to get message consisting blank spaces

int get\_line(char \*line, int max) {

char c;

int len;

for (len = 0; len < max && (c = getchar()) != EOF && c != '\n'; ++len)

line[len] = c;

if (c == '\n') {

line[len] = c;

len++;

}

line[len] = '\0';

return len;

}

void\* recv\_message(void\* socketfd){

char recev\_buffer[BUFF\_SIZE];

memset(recev\_buffer,0,BUFF\_SIZE);

if(pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL) != 0){

perror("Failed to set cancel!\n");

}

while(1){

pthread\_testcancel();

if(recvfrom(clientfd,recev\_buffer,sizeof(recev\_buffer),0,(struct sockaddr \*)&server\_addr,&addr\_len) > 0){

printf("Receive from server: %s",recev\_buffer);

memset(recev\_buffer,0,BUFF\_SIZE);

}

}

}

void\* send\_message(void \*socketfd){

void \*retval = NULL;

char send\_buffer[BUFF\_SIZE];

memset(send\_buffer,0,BUFF\_SIZE);

//printf("The pid for recv thread: %d\n",((struct parameter\*)send\_parameter)->pid);

while(1){

if(get\_line(send\_buffer,BUFF\_SIZE) > 0){

sendto(clientfd,send\_buffer,sizeof(send\_buffer),0,(struct sockaddr \*)&server\_addr,addr\_len);

if(strcmp(send\_buffer,shutdown\_message) == 0){

pthread\_exit(retval);

}

memset(send\_buffer,0,BUFF\_SIZE);

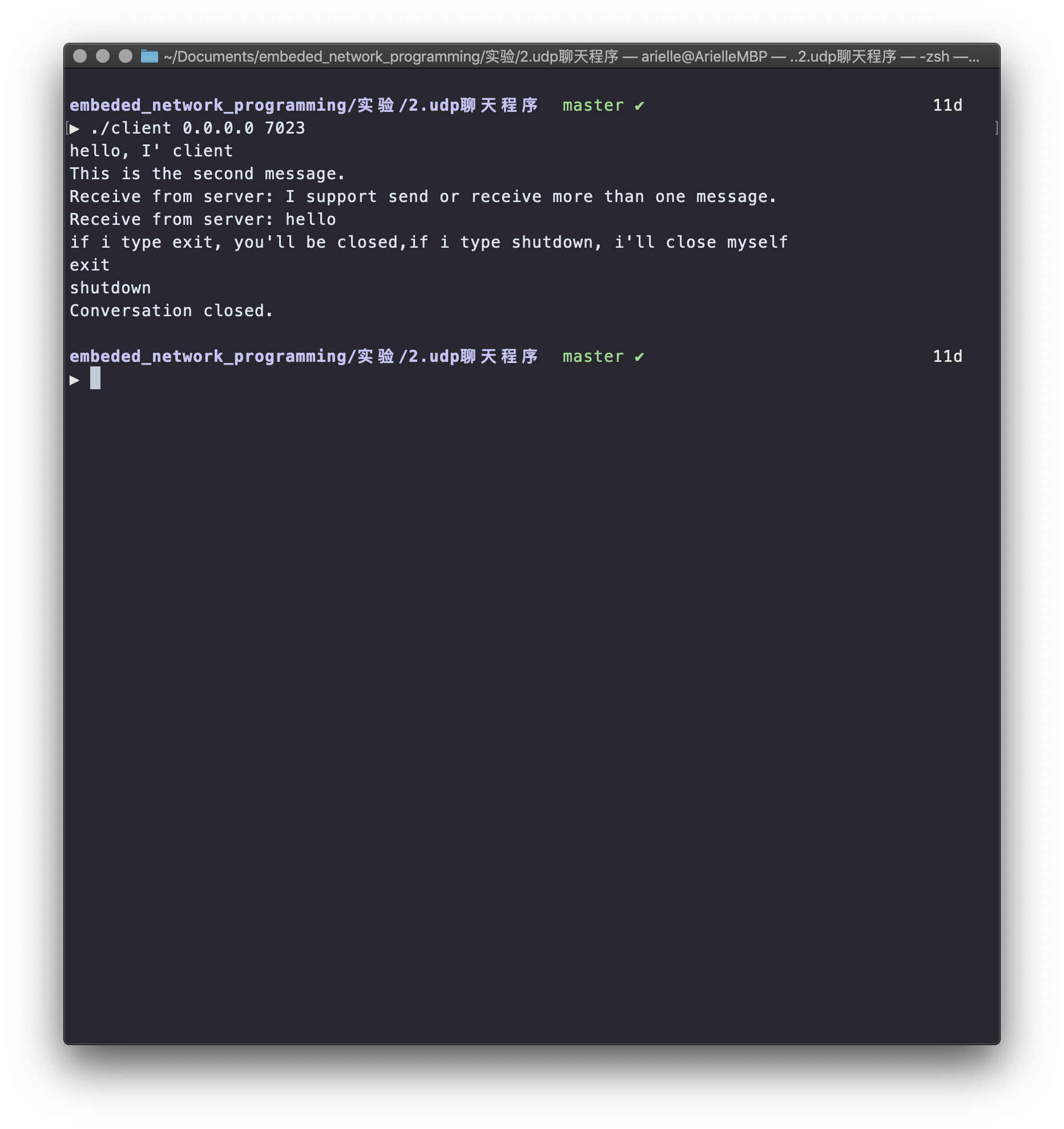
}

}

}



Server运行结果



Client运行结果

九、 总结及心得体会：

Linux手册上的内容最权威，应该多参考。

用户数据报协议（英语：User Datagram Protocol，缩写为UDP），又称用户数据报文协议，是一个简单的面向数据报的[传输层](https://zh.wikipedia.org/wiki/%E4%BC%A0%E8%BE%93%E5%B1%82)协议，正式规范为RFC 768。

在TCP/IP模型中，UDP为[网络层](https://zh.wikipedia.org/wiki/%E7%BD%91%E7%BB%9C%E5%B1%82)以上和[应用层](https://zh.wikipedia.org/wiki/%E5%BA%94%E7%94%A8%E5%B1%82)以下提供了一个简单的接口。UDP只提供数据的不可靠传递，它一旦把应用程序发给网络层的数据发送出去，就不保留数据备份（所以UDP有时候也被认为是不可靠的数据报协议）。UDP在IP数据报的头部仅仅加入了复用和数据校验（字段）

socket是在应用层和传输层之间的一个抽象层，它把TCP/IP层复杂的操作抽象为几个简单的接口供应用层调用已实现进程在网络中通信。

sockaddr和sockaddr\_in包含的数据都是一样的，但他们在使用上有区别：程序员不应操作sockaddr，sockaddr是给操作系统用的。程序员应使用sockaddr\_in来表示地址，sockaddr\_in区分了地址和端口，使用更方便。

十、 对本实验过程及方法、手段的改进建议：

可以多增加一些变化，提高难度。

报告评分：

指导教师签字：