

# 计算机网络 II

## 实验报告

班号： 1504201

学号： 150120526

姓名： 殷悦

# 实验 1 基于流式套接字的网络程序设计

姓名：殷悦

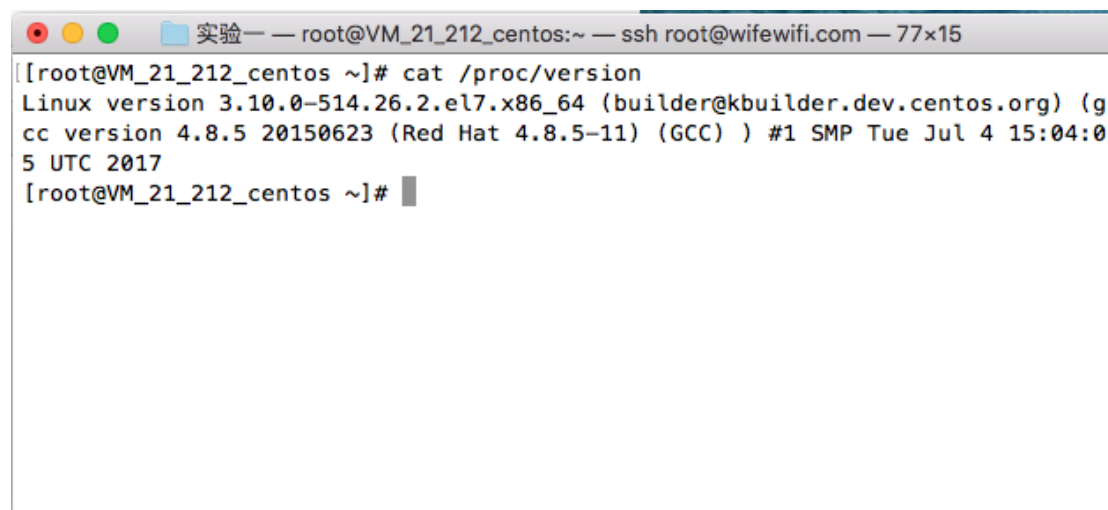
学号：150120526

班级：1504201

## 一、实验环境

Linux 平台

GUN GCC 编译器



```
实验一 — root@VM_21_212_centos:~ — ssh root@wifewifi.com — 77x15
[root@VM_21_212_centos ~]# cat /proc/version
Linux version 3.10.0-514.26.2.el7.x86_64 (builder@kbuilder.dev.centos.org) (g
cc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC) ) #1 SMP Tue Jul 4 15:04:0
5 UTC 2017
[root@VM_21_212_centos ~]#
```

## 二、实验内容

### 1.设计思路

#### 1.1 时间服务器

服务器使用 TCP 协议循环运行提供服务，客户端连接时，服务器查询当前时间，并给与回应，然后断开连接

#### 1.2.1 回射服务器

服务器循环运行，客户端向服务器发送一条信息，服务器在消息前面加上 echo:然后封装原路发回去，判断如果传过来的信息是 q, 则退出程序，若使用 recvline, 则每次 recv 一个字节，当收到\n 时则停止接受，recvline 每次接受一行，recvn 每次接受 n 个字节，若未接受够，则减去已接收的长度，继续接受，直到接收够或者 recv 返回为 0 则停止接收

#### 1.2.2 线程并发服务器

服务器循环运行，每次接受一个连接请求，则创建一个线程，将接收数据和回射任务交给线程来完成，直到线程运行结束释放，接受到数据，在数据前加上 echo:

### 2.程序清单（要求有详细的注释）

注：由于 linux 下编码不方便，源码中无注释，注释全在文字报告中

#### 1.1 时间服务器

```
#include <stdio.h>
```

```

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <time.h>

//输出错误信息

int prierr(char *msg){

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

    exit(1);

    return 0;

}


int main(int argc,char** argv){

    int sockfd;

    int port;

    int server;

    struct sockaddr_in addr;

    char msg[2000];

    if(argc != 3 && argc != 4)//使用说明

        prierr("Usage:\n \

        Server: s <port>\n \

```

```

Client: c <ip> <port>");

sockfd=socket(AF_INET,SOCK_STREAM,0);

if(!strcmp(argv[1],"s"))//判断是客户端还是服务器

    server=1;

else if (!strcmp(argv[1],"c"))//获取 IP

    server=0;

else

    prierr("Wroung input");

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

addr.sin_family=AF_INET;

if(server){

    addr.sin_addr.s_addr=htonl(INADDR_ANY); //服务器

    addr.sin_port=htons(atoi(argv[2]));//端口

    if(bind(sockfd,(struct sockaddr*)&addr,sizeof(addr))==-1)

        prierr("bind err");

    if(listen(sockfd,5)==-1)

        prierr("listen err");

    while(1){

        int cliaddr;

        socklen_t clilen=sizeof(cliaddr);

        cliaddr=accept(sockfd,(struct sockaddr*)&cliaddr,&clilen);

```

```

        if(cliaddr==-1)//接受失败

            prierr("accept err");

            time_t timep;

            time(&timep);//获取当前时间

            sprintf(msg,"Time now:%s\n",asctime(gmtime(&timep)));

            printf("send %s\n",msg);

            write(cliaddr,msg,sizeof(msg));

            close(cliaddr); //关闭客户端套接字

    }

    close(sockfd); //关闭服务器套接字

}else{

    addr.sin_addr.s_addr=inet_addr(argv[2]);//连接 IP

    addr.sin_port=htons(atoi(argv[3]));

    if(connect(sockfd,(struct sockaddr*)&addr,sizeof(addr))!=-1)

        prierr("connect error!");//连接

        int rcvlen;

        char *p=msg;

        do{

            rcvlen=read(sockfd,p,2000);//等待数据

            if(rcvlen==-1)

                printf("recv error\n");

            else if(rcvlen==0)

```

```

        printf("connection closed\n");

        p=(char *)(p+sizeof(char)*rcvlen);//循环接受

    }while(rcvlen>0);

    printf("recv %s\n",msg);

    close(sockfd);

}

return 0;

}

```

### 1. 2. 1 回射服务器

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

int prierr(char *msg){ //错误信息

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

    exit(1);

    return 0;
}

```

```
}
```

```
int recvn(int socket,char* buf,int len){

    int msglen;

    int left=len;

    while(msglen>0){

        msglen=read(socket,buf,left);

        if(msglen<0)

            printf("recv data err\n");

        else if(msglen==0)

            return len-left;

        left=msglen;//减少剩余的

        buf+=msglen;//移动存放位置

    }

    return len-left;//返回获取长度

}
```

```
int recvl(int socket,char* buf){

    int len;

    int rcvlen;

    char *p=buf;

    do{
```

```

        rcvlen=read(socket,p,sizeof(buf));

        if(rcvlen==-1)

            printf("recv error\n");

        else if(rcvlen==0)

            printf("connection closed\n");

        len+=rcvlen;

        p+=rcvlen;

        printf("%s",p);

    }while(rcvlen>0);

    return len;

}

```

```

int rcvlen(int socket,char* buf){

    int len;

    int rcvlen;

    char *p=buf;

    do{

        rcvlen=read(socket,p,1);//每次读取一个字节

        if(rcvlen==-1)

            printf("recv len\n");

        len++;

    }
}

```



```

        p++;

}while(*p!='\n');//寻找换行符标记

return len;

}

int main(int argc,char** argv){

    int sockfd;

    int port;

    int server;

    int rcvlen;

    struct sockaddr_in addr;

    char msg[2000];

    char msg2[2000];

    int choice=0;

    if(argc != 3 && argc != 4)//使用说明

        prierr("Usage:\n \

                Server: s <port>\n \

                Client: c <ip> <port>");

    sockfd=socket(AF_INET,SOCK_STREAM,0) //流式套接字

    if(!strcmp(argv[1],"s"))//判断服务类型

        server=1;

```

```

else if (!strcmp(argv[1], "c"))

    server=0;

else

    prierr("Wroung input");

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

addr.sin_family=AF_INET;

if(server){

    addr.sin_addr.s_addr=htonl(INADDR_ANY);

    addr.sin_port=htons(atoi(argv[2]));

    if(bind(sockfd,(struct sockaddr*)&addr,sizeof(addr))==-1)

        prierr("bind err");

    if(listen(sockfd,5)==-1)

        prierr("listen err");

    while(1){

        int cliaddr;

        socklen_t clien=sizeof(cliaddr);

        cliaddr=accept(sockfd,(struct sockaddr*)&cliaddr,&clilen);

        if(cliaddr==-1)

            prierr("accept err");

        //rcvlen=read(cliaddr,msg,sizeof(msg));

    }

}

choice=0;//choice one you want to use

```

```

switch(choice){//在这里选择接收方式

    case 0:

        printf("Using recvn:\n");

        rcvlen=recvn(cliaddr,msg,10);

        break;

    case 1:

        printf("Using recvl:\n");

        rcvlen=recvl(cliaddr,msg);

        break;

    case 2:

        printf("Using recvlen:\n");

        rcvlen=recvlen(cliaddr,msg);

        break;

}

/*

printf("recv:%s\n",msg);

//printf("recv len:%d\n",rcvlen);

sprintf(msg2,"echo:%s",msg);

if(!strcmp(msg,"q"))//判断结束消息

    prierr("Bye bye");

strcpy(msg,msg2);

write(cliaddr,msg,rcvlen);

```

```

        printf("send:%s\n",msg);

        close(cliaddr); //关闭客户端套接字

    }

    close(sockfd); //关闭服务器套接字

}

else{//客户端

    addr.sin_addr.s_addr=inet_addr(argv[2]); //连接 ip

    addr.sin_port=htons(atoi(argv[3])); //连接端口

    if(connect(sockfd,(struct sockaddr*)&addr,sizeof(addr))!=-1)

        prierr("connect error!");

    scanf("%s",&msg);

    write(sockfd,msg,sizeof(msg));

    int rcvlen;

    char *p=msg;

    do{

        rcvlen=read(sockfd,p,sizeof(msg));

        if(rcvlen==-1)

            printf("recv error\n");

        //    else if(rcvlen==0)

        //        printf("connection closed\n");

        p=(char *)(p+sizeof(char)*rcvlen);

    }while(rcvlen>0);

    printf("recv %s\n",msg);

```

```

        close(sockfd);

    }

    return 0;

}

```

### 1. 2. 2 线程并发服务器

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <pthread.h>

int prierr(char *msg){

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

    exit(1);

    return 0;

}

void thread(void *arg){//接受线程

    int cliaddr=*(int *)arg;

```

```

int rcvlen;

char msg[2000];

char msg2[2000];

printf("pid:%u\n", (unsigned short)pthread_self());

rcvlen=read(cliaddr, msg, sizeof(msg));

if(rcvlen== -1)

    printf("recv err\n");

printf("recv:%s\n", msg);

//printf("recv len:%d\n", rcvlen);

sprintf(msg2, "echo:%s", msg);

strcpy(msg, msg2);

write(cliaddr, msg, rcvlen);

printf("send:%s\n", msg);

close(cliaddr);

}

```

```

int main(int argc, char** argv){

    int sockfd;

    int port;

    int server;

    int rcvlen;

    char msg[2000];

```

```

pthread_t id;

struct sockaddr_in addr;

int choice=0;

if(argc != 3 && argc != 4)

    prierr("Usage:\n \

        Server: s <port>\n \

        Client: c <ip> <port>");

sockfd=socket(AF_INET,SOCK_STREAM,0);

if(!strcmp(argv[1],"s"))

    server=1;

else if(!strcmp(argv[1],"c"))

    server=0;

else

    prierr("Wroung input");

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

addr.sin_family=AF_INET;

if(server){

    addr.sin_addr.s_addr=htonl(INADDR_ANY);

    addr.sin_port=htons(atoi(argv[2]));

    if(bind(sockfd,(struct sockaddr*)&addr,sizeof(addr))== -1)

        prierr("bind err");

```

```

if(listen(sockfd,5)==-1)

    prierr("listen err");

while(1){

    int cliaddr;

    socklen_t clilen=sizeof(cliaddr);

    cliaddr=accept(sockfd,(struct sockaddr*)&cliaddr,&clilen);

    if(cliaddr==-1)

        prierr("accept err");

    int ret=pthread_create(&id,NULL,(void *)thread,&cliaddr);

    if(ret!=0)//创建线程来完成接受和发送

        prierr("create thread error");

}

close(sockfd);

}else{//客户端

    addr.sin_addr.s_addr=inet_addr(argv[2]);

    addr.sin_port=htons(atoi(argv[3]));

    if(connect(sockfd,(struct sockaddr*)&addr,sizeof(addr))== -1)

        prierr("connect error!");

    scanf("%s",&msg);

    write(sockfd,msg,sizeof(msg));

    int rcvlen;

    char *p=msg;

```



```

do{

    rcvlen=read(sockfd,p,sizeof(msg));

    if(rcvlen==-1)

        printf("recv error\n");

    //    else if(rcvlen==0)

    //        printf("connection closed\n");

    p=(char*)(p+sizeof(char)*rcvlen);

}while(rcvlen>0);

printf("recv %s\n",msg);

close(sockfd);

}

return 0;

}

```

### 3.用户使用说明（输入 / 输出规定）

注：请参考下面运行结果截图看此部分

#### 1.1 时间服务器（源码在 实验一/第一题/time.c）

客户端和服务端在一个源码中，编译 `gcc time.c`

服务器先运行，客户端后运行

服务器运行：程序名 `s` <端口>

如图所示： `./a.out s 8080`

客户端运行：程序名 `c` <IP> <端口>

如图所示： `./a.out c 127.0.0.1 8080`

客户端运行后，会连接服务器，服务器返回时间，显示在客户端中

#### 1.2.1 回射服务器（源码在 实验一/第二题第一问/recv.c）

客户端和服务端在一个源码中，编译 `gcc recv.c`

服务器先运行，客户端后运行

服务器运行：程序名 `s` <端口>

如图所示： `./a.out s 8080`

客户端运行：程序名 c <IP> <端口>

如图所示：./a.out c 127.0.0.1 8080

客户端运行后，会连接服务器，在客户端中输入任意内容，按下回车后，客户端将输入的内容发给服务器，服务器处理后返回给客户端

### 1.2.2 线程并发服务器（源码在 实验一/第二题第一问/threadrecv.c）

客户端和服务端在一个源码中，编译 gcc threadrecv.c -lpthread

注：linux 系统中需要安装 pthread 库才能支持线程，编译参数为 -lpthread 为静态连接，编译参数为 -pthread 为动态连接

服务器先运行，客户端后运行

服务器运行：程序名 s <端口>

如上图所示：./a.out s 8080

客户端运行：程序名 c <IP> <端口>

如上图所示：./a.out c 127.0.0.1 8080

客户端运行后，会连接服务器，在客户端中输入任意内容，按下回车后，客户端将输入的内容发给服务器，服务器处理后返回给客户端

## 4.运行结果（要求截图）

### 1.1 时间服务器

```
[root@VM_21_212_centos 1]# ls
a.out  time.c
[root@VM_21_212_centos 1]# gcc time.c
[root@VM_21_212_centos 1]# ./a.out s 8080
send Time now:Sat Jan 13 08:39:03 2018

send Time now:Sat Jan 13 08:39:07 2018

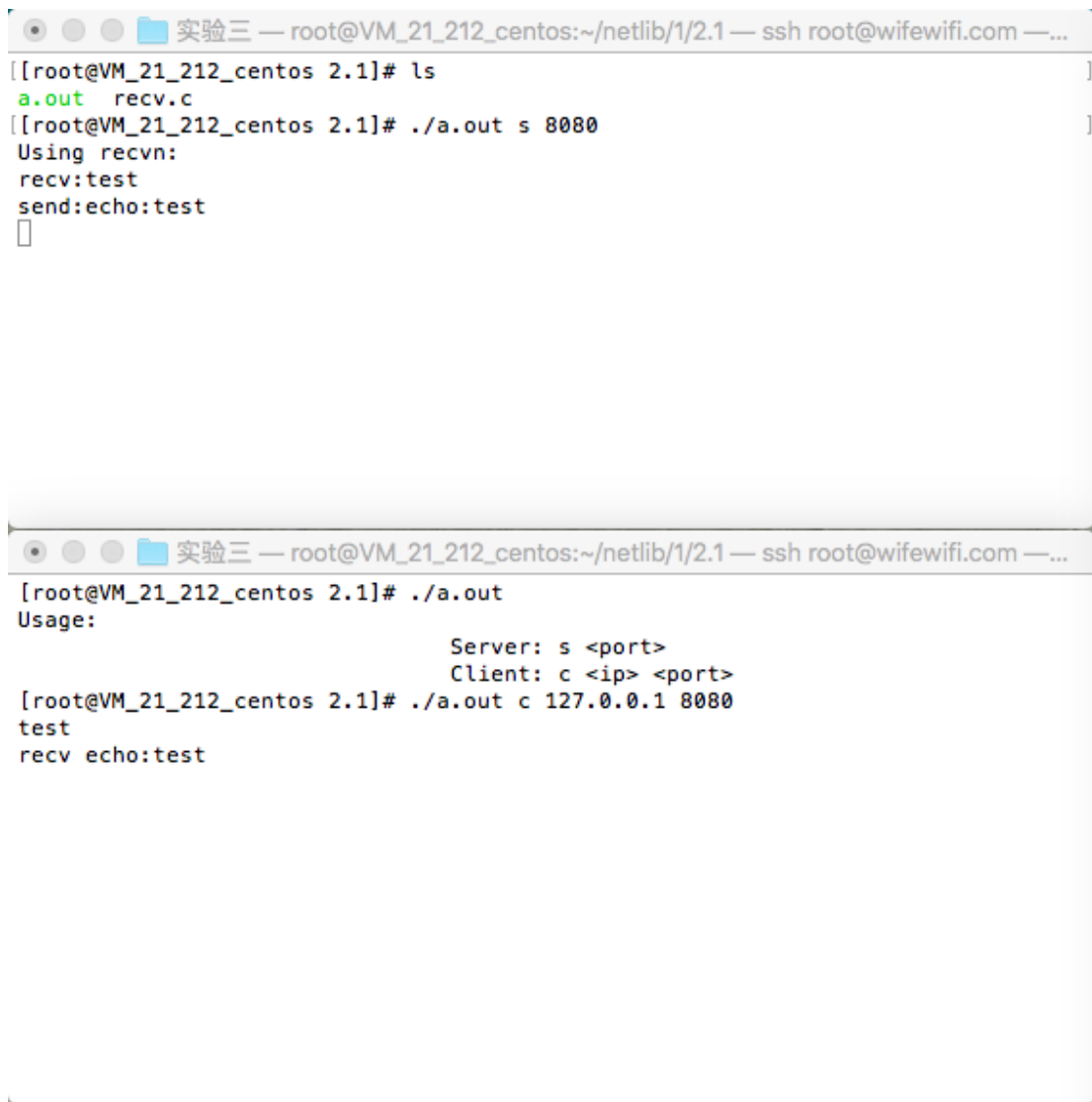
[root@VM_21_212_centos 1]# ./a.out
Usage:
    Server: s <port>
    Client: c <ip> <port>
[root@VM_21_212_centos 1]# ./a.out c 127.0.0.1 8080
connection closed
recv Time now:Sat Jan 13 08:39:03 2018

[root@VM_21_212_centos 1]# ./a.out c 127.0.0.1 8080
connection closed
recv Time now:Sat Jan 13 08:39:07 2018

[root@VM_21_212_centos 1]#
```

图 1.1 时间服务器

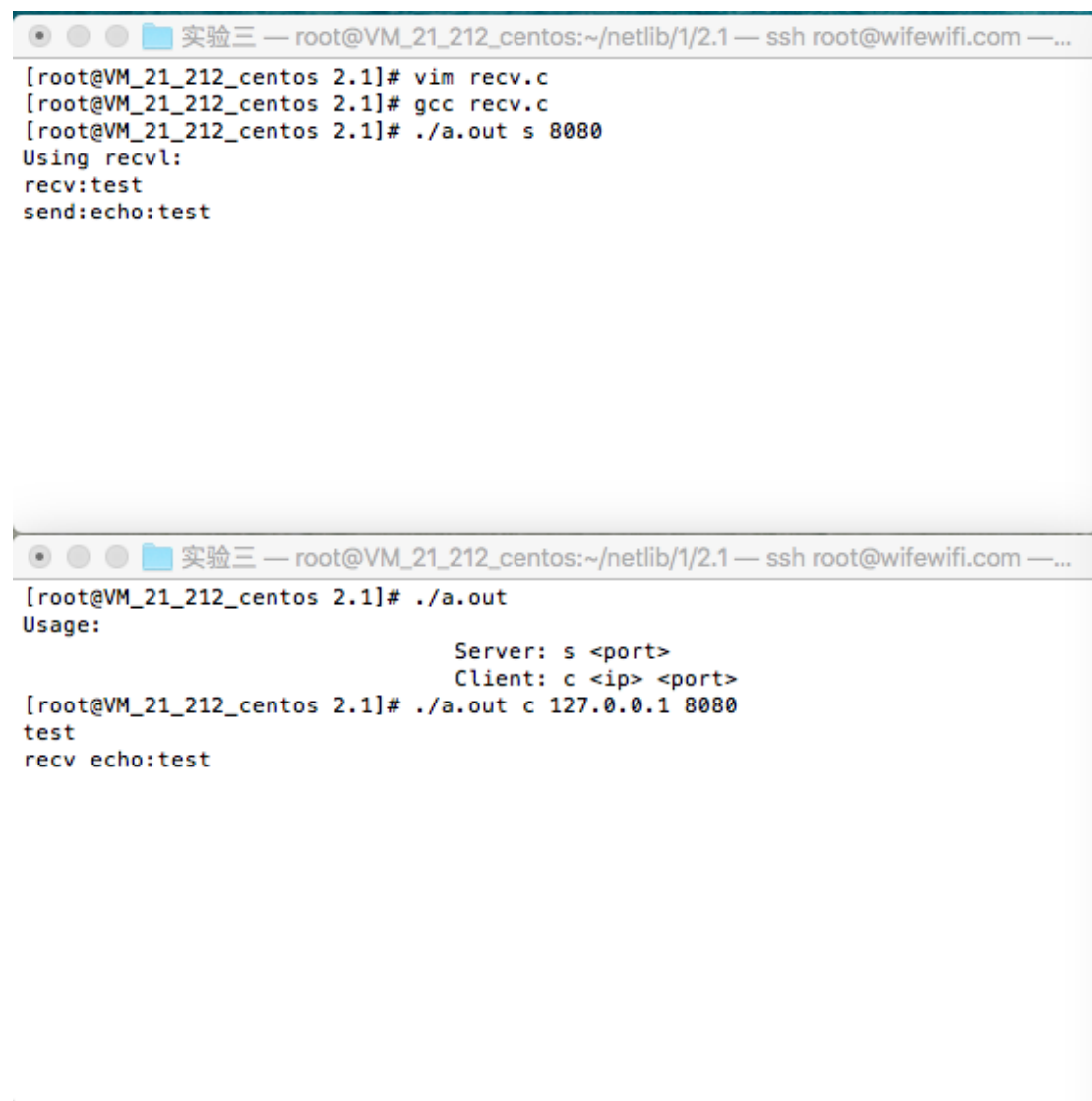
### 1.2.1 回射服务器



The image displays two terminal windows from a CentOS VM. The top window shows the compilation of 'recv.c' into 'a.out' and the execution of 'a.out s 8080' to start the server. The output indicates it is using 'recv' and is ready to receive. The bottom window shows the execution of 'a.out' with usage instructions, followed by a client connection 'a.out c 127.0.0.1 8080' which sends the message 'test', and the server's response 'recv echo:test'.

```
实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...  
[[root@VM_21_212_centos 2.1]# ls  
a.out  recv.c  
[[root@VM_21_212_centos 2.1]# ./a.out s 8080  
Using recv:  
recv:test  
send:echo:test  
□  
  
实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...  
[root@VM_21_212_centos 2.1]# ./a.out  
Usage:  
Server: s <port>  
Client: c <ip> <port>  
[root@VM_21_212_centos 2.1]# ./a.out c 127.0.0.1 8080  
test  
recv echo:test
```

图 1.2.1.1recv 回射服务器

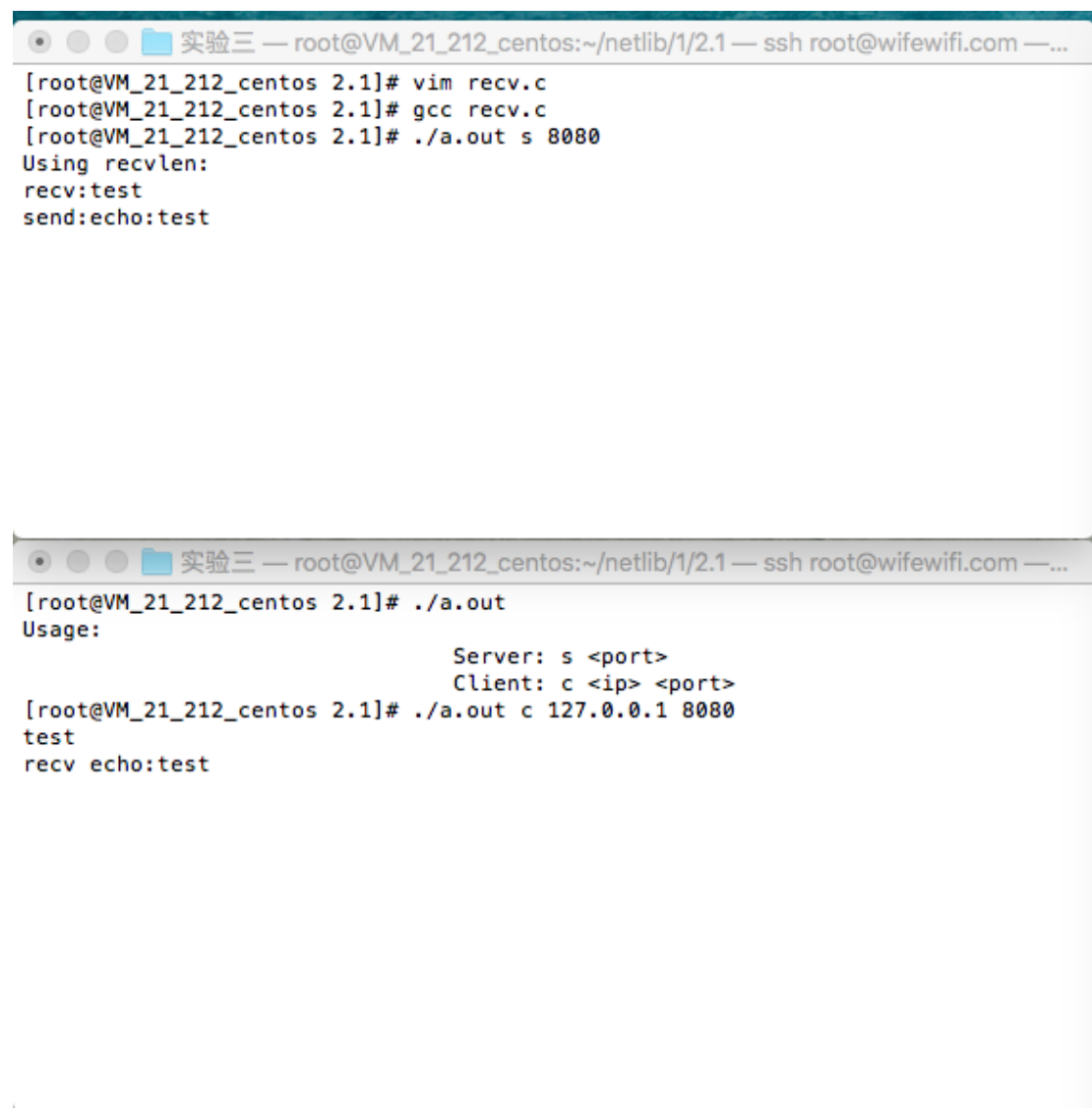


The image displays two terminal window screenshots from a CentOS virtual machine. The top window shows the user editing 'recv.c' with vim, compiling it with gcc, and running the resulting binary './a.out' as a server on port 8080. The output shows it is using 'recv' and has received the message 'test'. The bottom window shows the user running './a.out' as a client, connecting to 127.0.0.1 on port 8080, and sending the message 'test', which is echoed back.

```
实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...
[root@VM_21_212_centos 2.1]# vim recv.c
[root@VM_21_212_centos 2.1]# gcc recv.c
[root@VM_21_212_centos 2.1]# ./a.out s 8080
Using recv:
recv:test
send:echo:test

实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...
[root@VM_21_212_centos 2.1]# ./a.out
Usage:
                Server: s <port>
                Client: c <ip> <port>
[root@VM_21_212_centos 2.1]# ./a.out c 127.0.0.1 8080
test
recv echo:test
```

图 1.2.1.2 recv 回射服务器



The image consists of two terminal window screenshots. The top window shows the user editing a file named 'recv.c' with vim, compiling it with gcc, and running the resulting binary 'a.out' as a server listening on port 8080. It receives a connection from 'test' and echoes back 'test'. The bottom window shows the user running 'a.out' as a client, connecting to the server at 127.0.0.1:8080, and sending the message 'echo:test'.

```
实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...  
[root@VM_21_212_centos 2.1]# vim recv.c  
[root@VM_21_212_centos 2.1]# gcc recv.c  
[root@VM_21_212_centos 2.1]# ./a.out s 8080  
Using recvlen:  
recv:test  
send:echo:test  
  
实验三 — root@VM_21_212_centos:~/netlib/1/2.1 — ssh root@wifewifi.com —...  
[root@VM_21_212_centos 2.1]# ./a.out  
Usage:  
Server: s <port>  
Client: c <ip> <port>  
[root@VM_21_212_centos 2.1]# ./a.out c 127.0.0.1 8080  
test  
recv echo:test
```

图 1.2.1.3 recvlen 回射服务器

## 1.2.2 线程并发服务器

```
实验三 — root@VM_21_212_centos:~/netlib/1/2.2 — ssh root@wifewifi.com —...
[[root@VM_21_212_centos 2.2]# ls
a.out  threadrecv.c
[[root@VM_21_212_centos 2.2]# gcc threadrecv.c -lpthread
[[root@VM_21_212_centos 2.2]# ./a.out s 8080
tid:46848
recv:test
send:echo:test
tid:42752
recv:hello
send:echo:hello
tid:38656
recv:yinyue
send:echo:yinyue
[]

实验三 — root@VM_21_212_centos:~/netlib/1/2.2 — ssh root@wifewifi.com —...
[[root@VM_21_212_centos 2.2]# ./a.out
Usage:
                Server: s <port>
                Client: c <ip> <port>
[[root@VM_21_212_centos 2.2]# ./a.out c 127.0.0.1 8080
[test
recv echo:test
[[root@VM_21_212_centos 2.2]# ./a.out c 127.0.0.1 8080
[hello
recv echo:hello
[[root@VM_21_212_centos 2.2]# ./a.out c 127.0.0.1 8080
[yinyue
recv echo:yinyue
[[root@VM_21_212_centos 2.2]# []
```

图 1.2.2 线程并发服务器

## 实验 2 基于数据报套接字的网络程序设计

姓名：殷悦

学号：150120526

班级：1504201

### 一、实验环境

Linux 平台

GUN GCC 编译器



```
实验一 — root@VM_21_212_centos:~ — ssh root@wifewifi.com — 77x15
[root@VM_21_212_centos ~]# cat /proc/version
Linux version 3.10.0-514.26.2.el7.x86_64 (builder@kbuilder.dev.centos.org) (g
cc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC) ) #1 SMP Tue Jul 4 15:04:0
5 UTC 2017
[root@VM_21_212_centos ~]#
```

### 二、实验内容

#### 1. 设计思路

##### 2.1.1 数据报循环服务器与客户端

服务器使用 UDP 协议循环运行提供服务，先运行服务器，再运行客户端。服务器循环处理一个一个客户的请求，当每个客户的请求处理完后，才能处理下一个客户的请求。服务器每收到一个数据，在数据前加上 echo：后返回给客户端。

##### 2.1.2 数据包并发服务器与客户端

服务器使用 UDP 协议并发运行提供服务，先运行服务器，再运行客户端（注：由于第一个实验已经使用过线程处理，所以为了学习 fork，此并发采用 fork 来实现）。服务器创建一个套接字，然后接收数据，每次接收完后，fork 一下，然后父进程进入下一轮接收循环；子进程重新创建一个新的套接字，然后将刚刚收到的数据在数据前加上 echo：后用新的套接字发送回去。

##### 2.2 丢包率测试服务器与客户端

服务器使用 UDP 协议提供服务，先运行服务器，再运行客户端。选择一段数据，用 setsockopt 设置好超时，然后循环发送接收 n 次，统计发送的次数和失败的次数，最后用 丢包率=(失败次数/成功次数)\*100%

#### 2. 程序清单（要求有详细的注释）

注：由于 linux 下编码不方便，源码中无注释，注释全在文字报告中

##### 2.1.1 数据报循环服务器与客户端

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <unistd.h>
```

```
#include <arpa/inet.h>
```

```
#include <sys/socket.h>
```

```
#include <time.h>
```

```
int prierr(char *msg){ //错误输出
```

```
    printf("%s\n",msg);
```

```
    exit(1);
```

```
    return 0;
```

```
}
```

```
int main(int argc,char** argv){
```

```
    int sockfd;
```

```
    int port;
```

```
    socklen_t len;
```

```
    int server;
```

```
    struct sockaddr_in saddr;
```

```
    struct sockaddr_in raddr;
```

```
    char msg[1500];
```

```
    char msg2[1500];
```



```

int msglen;

if(argc != 3 && argc != 4) //使用帮助

    prierr("Usage:\n \

        Server: s <port>\n \

        Client: c <ip> <port>");


sockfd=socket(AF_INET,SOCK_DGRAM,0); //数据包套接字

if(!strcmp(argv[1],"s"))//判断服务类型

    server=1;

else if (!strcmp(argv[1],"c"))

    server=0;

else

    prierr("Wroung input");

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

saddr.sin_family=AF_INET;

if(server){ //服务器

    saddr.sin_addr.s_addr=htonl(INADDR_ANY);

    saddr.sin_port=htons(atoi(argv[2]));

    if(bind(sockfd,(struct sockaddr*)&saddr,sizeof(saddr))== -1)

        prierr("bind err");

    while(1){

```

```

        if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,&len))<0) //接受数据

            prierr("recvfrom error");

            msg[msglen]=0;

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

            sprintf(msg2,"echo:%s",msg); //添加 echo:

            strcpy(msg,msg2);

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

            len=sizeof(raddr);

            if((msglen=sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr*)&raddr,len))<0)
//发送数据

                prierr("sendto error");

            }

            close(sockfd);

    }else{//客户端

        saddr.sin_addr.s_addr=inet_addr(argv[2]); //连接 IP

        saddr.sin_port=htons(atoi(argv[3])); //连接端口

        while(1){

            scanf("%s",msg);

            if(!strcmp(msg,"q"))

                close(sockfd);

            len=sizeof(saddr);

            if((msglen=sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr*)&saddr,len))<0)

```

```

        prierr("sendto error");

        if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,&len))<0)

            prierr("recvfrom error");

        msg[msglen]=0;

        fputs(msg,stdout);

        //printf("%s\n",msg);

    }

}

return 0;

}

```

### 2. 1. 2 数据包并发服务器与客户端

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <time.h>

```

```

int prierr(char *msg){

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

```

```

    exit(1);

    return 0;
}

```

```

int main(int argc, char** argv){

    int sockfd;

    int sockfd2;

    int port;

    socklen_t len;

    int server;

    struct sockaddr_in saddr;

    struct sockaddr_in raddr;

    char msg[1500];

    char msg2[1500];

    int msglen;

    pid_t fpid;

    if(argc != 3 && argc != 4)

        prierr("Usage:\n \

                Server: s <port>\n \

                Client: c <ip> <port>");

    sockfd=socket(AF_INET, SOCK_DGRAM, 0);

```

```

if(sockfd<0)

    prierr("father socket error");

if(!strcmp(argv[1],"s"))

    server=1;

else if (!strcmp(argv[1],"c"))

    server=0;

else

    prierr("Wroung input");

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

saddr.sin_family=AF_INET;

if(server){

    saddr.sin_addr.s_addr=htonl(INADDR_ANY);

    saddr.sin_port=htons(atoi(argv[2]));

    if(bind(sockfd,(struct sockaddr*)&saddr,sizeof(saddr))== -1)

        prierr("bind err");

    while(1){

        if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,&len))<0)

            prierr("recvfrom error");

        msg[msglen]=0;

        fpid=fork();//复制另一个自身

        if(fpid<0)//执行失败

```

```

        prierr("fork error\n");

    else if(fpid==0)//Son 子程序

        break;//父程序不跳出，循环到上面继续执行

    }

    printf("fork\n");//子进程执行

    sockfd2=socket(AF_INET,SOCK_DGRAM,0);//创建新的 UDP 套接字

    if(sockfd2<0)

        prierr("son socket error");

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

    sprintf(msg2,"echo:%s",msg);

    strcpy(msg,msg2);

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

    len=sizeof(raddr);

    if((msglen=sendto(sockfd2,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,len))<0)//用新的套接字发送出去

        prierr("sendto error");

    return 0;

    close(sockfd);

}

else{

    saddr.sin_addr.s_addr=inet_addr(argv[2]);

    saddr.sin_port=htons(atoi(argv[3]));

```

```

while(1){

    scanf("%s",msg);

    if(!strcmp(msg,"q"))

        close(sockfd);

    len=sizeof(saddr);

    if((msglen=sendto(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&saddr,len))<0)

        prierr("sendto error");

    if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,&len))<0)

        prierr("recvfrom error");

    msg[msglen]=0;

    fputs(msg,stdout);

    //printf("%s\n",msg);

}

}

return 0;

}

```

## 2.2 丢包率测试服务器与客户端

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

```

```
#include <unistd.h>
```

```
#include <arpa/inet.h>
```

```
#include <sys/socket.h>
```

```
#include <sys/time.h>
```

```
#include <time.h>
```

```
int prierr(char *msg){  
  
    printf("%s\n",msg);  
  
    exit(1);  
  
    return 0;  
  
}
```

```
int main(int argc,char** argv){  
  
    int sockfd;  
  
    int port;  
  
    socklen_t len;  
  
    int server;  
  
    struct sockaddr_in saddr;  
  
    struct sockaddr_in raddr;  
  
    char msg[1500];  
  
    char msg2[1500];  
  
    int msglen;
```



```

if(argc != 3 && argc != 4)

    prierr("Usage:\n \

        Server: s <port>\n \

        Client: c <ip> <port>");

sockfd=socket(AF_INET,SOCK_DGRAM,0);

if(!strcmp(argv[1],"s"))

    server=1;

else if(!strcmp(argv[1],"c"))

    server=0;

else

    prierr("Wroung input");

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

saddr.sin_family=AF_INET;

if(server){//服务器

    saddr.sin_addr.s_addr=htonl(INADDR_ANY);

    saddr.sin_port=htons(atoi(argv[2]));

    if(bind(sockfd,(struct sockaddr*)&saddr,sizeof(saddr))== -1)

        prierr("bind err");

    while(1){

        if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct

sockaddr*)&raddr,&len))<0)

```

```

        printf("recvfrom error\n");

    msg[msglen]=0;

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

    sprintf(msg2,"echo:%s",msg);

    strcpy(msg,msg2);

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

    len=sizeof(raddr);

    sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr*)&raddr,len);

}

close(sockfd);

}else{//客户端

    saddr.sin_addr.s_addr=inet_addr(argv[2]);

    saddr.sin_port=htons(atoi(argv[3]));

    //while(1){

        //fgets(msg,sizeof(msg),stdin);

        //strcpy(msg,"test message");

        if(!strcmp(msg,"q"))

            close(sockfd);

        len=sizeof(saddr);

        int i,j=0,k;

        printf("How many times do you want to test:\n");

        scanf("%d",&k); //获取重复执行次数

```

```

struct timeval tv={1,0};//设置延时

setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO,(char*)&tv, sizeof(tv));

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

    strcpy(msg,"test message");

    printf("trying the %dst time:\t",i);

    sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr*)&saddr,len);

    printf("sendto:%s\t",msg);

    if((msglen=recvfrom(sockfd,msg,sizeof(msg),0,(struct
sockaddr*)&raddr,&len))<0){

        printf("recvfrom error\n");

        j++;

    }

    msg[msglen]=0;

    printf("recvfrom:%s\n",msg);

}

printf("Send poket:%d,Lost poket:%d,Lost rate:%0.2f\n",i,j,(float)j*100/i);

//}

}

return 0;

}

```

### 3.用户使用说明（输入 / 输出规定）

注：请参考下面运行结果截图看此部分

2.1.1 数据报循环服务器与客户端（源码在 实验二 / 第一题第一问

/echo\_udp.c)

客户端和服务端在一个源码中，编译 `gcc echo_udp.c`

服务器先运行，客户端后运行

服务器运行：程序名 `s` <端口>

如图所示： `./a.out s 8080`

客户端运行：程序名 `c` <IP> <端口>

如图所示： `./a.out c 127.0.0.1 8080`

客户端运行后，在客户端中输入任意内容，按下回车后，客户端将输入的内容发给服务器，服务器处理后返回给客户端

2.1.2 数据包并发服务器与客户端（源码在 实验二/第一题第二问/echo\_udp.c）

客户端和服务端在一个源码中，编译 `gcc echo_udp.c`

服务器先运行，客户端后运行

服务器运行：程序名 `s` <端口>

如图所示： `./a.out s 8080`

客户端运行：程序名 `c` <IP> <端口>

如图所示： `./a.out c 127.0.0.1 8080`

客户端运行后，在客户端中输入任意内容，按下回车后，客户端将输入的内容发给服务器，服务器处理后返回给客户端

2.2 丢包率测试服务器与客户端（源码在 实验一/第二题/threadrecv.c）

客户端和服务端在一个源码中，编译 `gcc test.c`

注：由于 Windows 不支持 `fork`，因此交叉编译到 windows 平台上需要用线程处理

服务器先运行，客户端后运行

服务器运行：程序名 `s` <端口>

如上图所示： `./a.out s 8080`

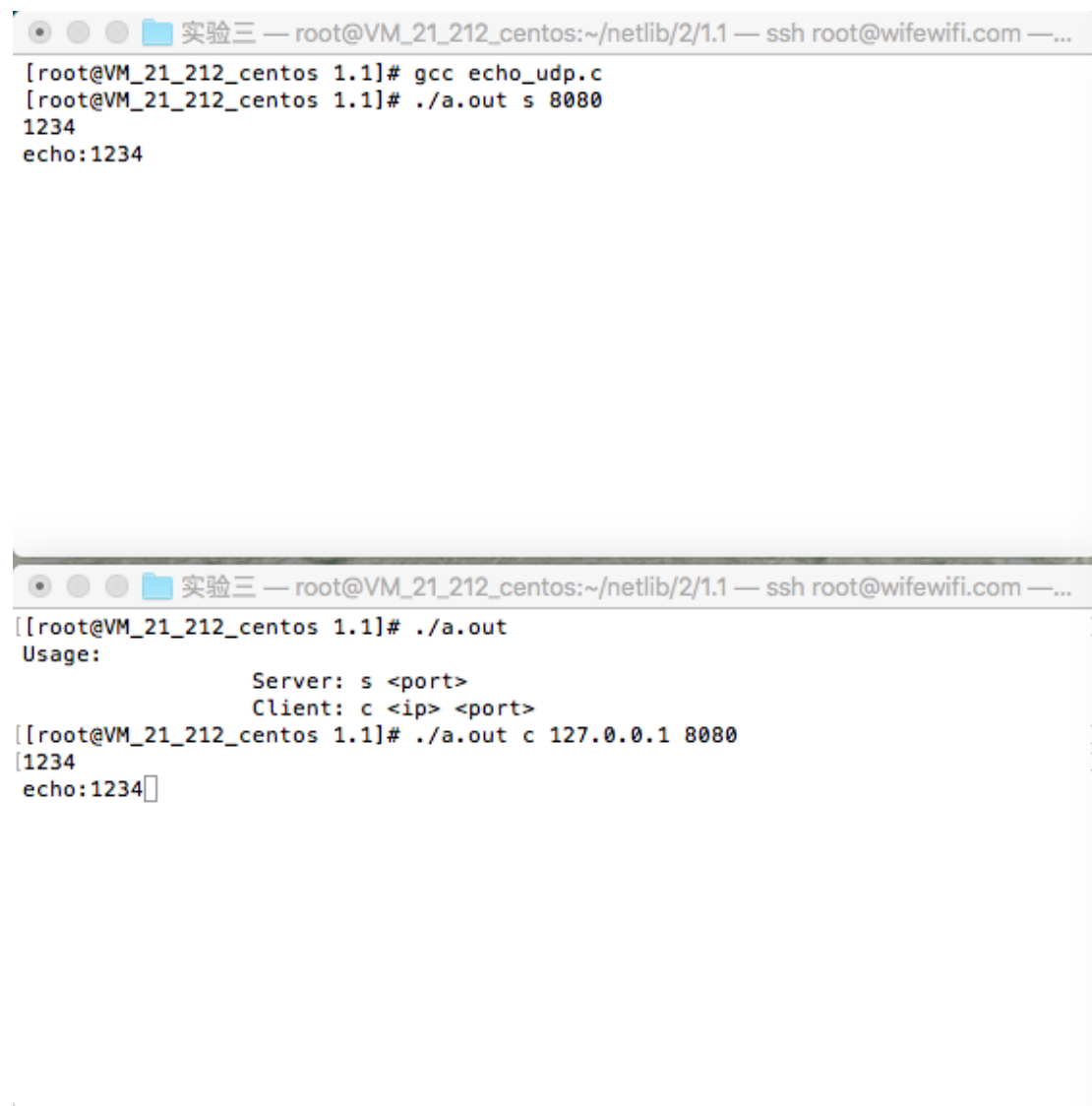
客户端运行：程序名 `c` <IP> <端口>

如上图所示： `./a.out c 127.0.0.1 8080`

客户端运行后，输入要测试的发送和接收循环的次数，然后按下回车进行测试（注：如果测试次数上十万次后可能比较慢，请耐心等待），测试好后显示测试结果

#### 4.运行结果（要求截图）

2.1.1 数据报循环服务器与客户端



```
实验三 — root@VM_21_212_centos:~/netlib/2/1.1 — ssh root@wifewifi.com —...
[root@VM_21_212_centos 1.1]# gcc echo_udp.c
[root@VM_21_212_centos 1.1]# ./a.out s 8080
1234
echo:1234

实验三 — root@VM_21_212_centos:~/netlib/2/1.1 — ssh root@wifewifi.com —...
[[root@VM_21_212_centos 1.1]# ./a.out
Usage:
    Server: s <port>
    Client: c <ip> <port>
[[root@VM_21_212_centos 1.1]# ./a.out c 127.0.0.1 8080
1234
echo:1234
```

图 2.1.1 数据报循环服务器与客户端

## 2.1.2 数据包并发服务器与客户端

```
实验三 — root@VM_21_212_centos:~/netlib/2/1.2 — ssh root@wifewifi.com —...
[[root@VM_21_212_centos 1.2]# ls
a.out echo_udp.c
[[root@VM_21_212_centos 1.2]# gcc echo_udp.c
[[root@VM_21_212_centos 1.2]# ./a.out s 8080
fork
hello
echo:hello
fork
yinyue
echo:yinyue
fork
123123
echo:123123
[]

实验三 — root@VM_21_212_centos:~/netlib/2/1.2 — ssh root@wifewifi.com —...
[[root@VM_21_212_centos 1.2]# ./a.out
Usage:
                                Server: s <port>
                                Client: c <ip> <port>
[[root@VM_21_212_centos 1.2]# ./a.out c 127.0.0.1 8080
hello
echo:hello
yinyue
echo:yinyue
123123
echo:123123[]
```

图 2.1.2 数据包并发服务器与客户端

## 2.2 丢包率测试服务器与客户端

```
实验三 — root@VM_21_212_centos:~/netlib/2/2 — ssh root@wifewifi.com — 7...
[[root@VM_21_212_centos 2]# ls
a.out test.c
[[root@VM_21_212_centos 2]# gcc test.c
[[root@VM_21_212_centos 2]# ./a.out s 8080
[

实验三 — root@VM_21_212_centos:~/netlib/2/2 — ssh root@wifewifi.com — 7...
[[root@VM_21_212_centos 2]# ./a.out
Usage:
                                Server: s <port>
                                Client: c <ip> <port>
[[root@VM_21_212_centos 2]# ./a.out c 127.0.0.1 8080
How many times do you want to test:
1000[

实验三 — root@VM_21_212_centos:~/netlib/2/2 — ssh root@wifewifi.com — 7...
echo:test message
test message
echo:test message
test message
echo:test message
test message
echo:test message
test message
echo:test message
test message
echo:test message
[

实验三 — root@VM_21_212_centos:~/netlib/2/2 — ssh root@wifewifi.com — 7...
trying the 987st time: sendto:test message      rcvfrom:echo:test message
trying the 988st time: sendto:test message      rcvfrom:echo:test message
trying the 989st time: sendto:test message      rcvfrom:echo:test message
trying the 990st time: sendto:test message      rcvfrom:echo:test message
trying the 991st time: sendto:test message      rcvfrom:echo:test message
trying the 992st time: sendto:test message      rcvfrom:echo:test message
trying the 993st time: sendto:test message      rcvfrom:echo:test message
trying the 994st time: sendto:test message      rcvfrom:echo:test message
trying the 995st time: sendto:test message      rcvfrom:echo:test message
trying the 996st time: sendto:test message      rcvfrom:echo:test message
trying the 997st time: sendto:test message      rcvfrom:echo:test message
trying the 998st time: sendto:test message      rcvfrom:echo:test message
trying the 999st time: sendto:test message      rcvfrom:echo:test message
Send poket:1000,Lost poket:1,Lost rate:0.10%
[[root@VM_21_212_centos 2]# [
```

图 2.2 丢包率测试服务器与客户端

## 实验3 原始套接字编程

姓名：殷悦

学号：150120526

班级：1504201

### 一、实验环境

Linux 平台

GUN GCC 编译器



```
[[root@VM_21_212_centos ~]# cat /proc/version
Linux version 3.10.0-514.26.2.el7.x86_64 (builder@kbuilder.dev.centos.org) (gcc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC) ) #1 SMP Tue Jul 4 15:04:05 UTC 2017
[root@VM_21_212_centos ~]#
```

### 二、实验内容

#### 1. 设计思路

##### 3.1 使用 icmp 协议实现 ping 程序

程序采用原始套接字实现客户端部分。首先创建原始套接字，然后用 `setsockopt` 设置超时，接着构造一个 icmp 数据包，填写 icmp 的参数，计算校验和，然后调用 `sendto` 将 icmp 数据包发送出去。然后循环调用 `select` 函数读取 `sockfd` 文件描述符的消息，当收到信息后调用 `recvfrom` 接收，若 `select` 超时，则提示超时错误，接收后判断是否是自己发送的数据包，若不是则丢弃重新接收，直到接收到自己的 icmp 包为止。

##### 3.2 使用原始套接字捕获数据包分析 ftp 协议

程序采用原始套接字实现服务器部分。首先创建原始套接字，然后调用 `recvfrom` 循环接受数据，接收到后，分析 ip 头，接着 tcp 头或 udp 头，然后将其中信息提取显示出来。

#### 2. 程序清单（要求有详细的注释）

注：由于 linux 下编码不方便，注释全在实验报告中，源码中无注释

##### 3.1 使用 icmp 协议实现 ping 程序

```
#include <stdio.h>
```

```
#include <sys/socket.h>
```



```

#include <sys/types.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/ip_icmp.h>

#include <netdb.h>

#include <string.h>

//数据包大小

#define PACKET_SIZE 4096

#define ERROR 0

#define SUCCESS 1

//计算校验和，发送包的时候需要用到

unsigned short cal_chksum(unsigned short *addr,int len){

    int nleft=len;

    int sum=0;

    unsigned short *w=addr;

    unsigned short answer=0;

    while(nleft > 1){

        sum += *w++;

        nleft -= 2;

    }

```

```

if(nleft==1){

    *(unsigned char *)&answer=*(unsigned char *)w;

    sum += answer;

}

sum=(sum >> 16) +(sum & 0xffff);

sum +=(sum >> 16);

answer=~sum;


return answer;

}

```

```

int ping(char *ips,int timeout){

    struct timeval *tval;

    int maxfds=0;

    fd_set readfds;


    struct sockaddr_in addr;

    struct sockaddr_in from;

    bzero(&addr,sizeof(addr));

    addr.sin_family=AF_INET;

    addr.sin_addr.s_addr=inet_addr(ips);

    int sockfd;

```

```
sockfd=socket(AF_INET,SOCK_RAW,IPPROTO_ICMP); //使用原始套接字
```

```
if(sockfd < 0){  
  
    printf("ip:%s,socket error\n",ips);  
  
    return ERROR;  
  
}
```

```
//设置超时时间
```

```
struct timeval timeo;  
  
timeo.tv_sec=timeout / 1000;  
  
timeo.tv_usec=timeout % 1000;  
  
if(setsockopt(sockfd,SOL_SOCKET,SO_SNDTIMEO,&timeo,sizeof(timeo))== -1){  
  
    printf("ip:%s,setsockopt error\n",ips);  
  
    return ERROR;  
  
}
```

```
char sendpacket[PACKET_SIZE];  
  
char recvpacket[PACKET_SIZE];  
  
memset(sendpacket,0,sizeof(sendpacket));
```

```
pid_t pid;  
  
pid=getpid();
```

```

struct ip *iph;

struct icmp *icmp;


icmp=(struct icmp*)sendpacket;

icmp->icmp_type=ICMP_ECHO;

icmp->icmp_code=0;

icmp->icmp_cksum=0;

icmp->icmp_seq=0;

icmp->icmp_id=pid;

tval=(struct timeval *)icmp->icmp_data;

gettimeofday(tval,NULL);

icmp->icmp_cksum=cal_chksum((unsigned short *)icmp,sizeof(struct icmp)); //计算校
验和


int n; //发送数据包

n=sendto(sockfd,(char *)&sendpacket,sizeof(struct icmp),0,(struct sockaddr
*)&addr,sizeof(addr));

if(n < 1){

    printf("ip:%s,sendto error\n",ips);

    return ERROR;

}

```

```

while(1){

    FD_ZERO(&readfds);

    FD_SET(sockfd,&readfds);

    maxfds=sockfd + 1;

    n=select(maxfds,&readfds,NULL,NULL,&timeo); //等待数据到达，超时用

    if(n <= 0){ //超时

        printf("ip:%s,Time out error\n",ips);

        close(sockfd);

        return ERROR;

    }


    memset(recvpacket,0,sizeof(recvpacket));

    int fromlen=sizeof(from);

    n=recvfrom(sockfd,recvpacket,sizeof(recvpacket),0,(struct sockaddr

*&from,(socklen_t *)&fromlen); //接受数据

    if(n < 1) {

        break;

    }


    iph=(struct ip *)recvpacket;

    icmp=(struct icmp *) (recvpacket +(iph->ip_hl<<2));

```

```

        printf("ip:%s\ticmp_type:%d\ticmp_id:%d\n",ips,icmp->icmp_type,icmp->icmp_id);

//通过判断进程 ID 号判断是不是自己 Ping 的包，如果不是，不要接受

        if(icmp->icmp_type==ICMP_ECHOREPLY && icmp->icmp_id==pid)

                break;

        else

                continue;

    }

}

```

```

int main(){

    char cPing[16];

    printf("please input ping ip:");

    scanf("%s",cPing);

    if(ping(cPing,10000))//10000 是超时时间，单位毫秒

        printf("ping succeed!\n");

    else

        printf("ping wrong!\n");

}

```

### 3.2 使用原始套接字捕获数据包分析 ftp 协议

```
#include <stdio.h>
```

```
#include <errno.h>
```

```
#include <unistd.h>
```

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
#include <linux/in.h>
```

```
#include <linux/if_ether.h>
```

```
int main(int argc, char **argv){
```

```
    int sock, n;
```

```
    char buffer[2048];
```

```
    unsigned char *iphead, *ethhead;
```

```
    if((sock=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP)))<0){
```

```
        perror("socket");//创建原始套接字
```

```
        return 1;
```

```
    }
```

```
    while(1){
```

```
        n = recvfrom(sock, buffer, 2048, 0, NULL, NULL); ;//接受数据
```

```
        if(n<42){ //数据包比 14+20+8(无数据的 UDP 包大小)还小，肯定不是 TCP 包或者
```

```
UDP 包，丢弃
```

```
            perror("recvfrom():");
```

```
            printf("Incomplete packet(errno is %d)\n",
```

```
                errno);
```

```

        close(sock);

        return 0;

    }

//跳过以太网头就是 IP 头

    iphead = buffer+14; /* Skip Ethernet header */

//判断协议类型

    if(*iphead==0x45){ /* Double check for IPv4

        * and no options present */

//第十位为 6 是 TCP 包，源端口或目的端口是 21 的是 FTP 服务

        if(iphead[9]==6 && (

            (iphead[20]<<8)+iphead[21]==21 ||

            (iphead[22]<<8)+iphead[23]==21)

        ){

            printf("%d bytes read\n",n);

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

            /*//输出源 MAC 和目的 MAC，题目未要求，不用输出

            ethhead = buffer;

            printf("Source MAC address: "

                "%02x:%02x:%02x:%02x:%02x:%02x\n",

                ethhead[0],ethhead[1],ethhead[2],

                ethhead[3],ethhead[4],ethhead[5]);

            printf("Destination MAC address: "

```



```

        "%02x:%02x:%02x:%02x:%02x:%02x\n",

        ethhead[6],ethhead[7],ethhead[8],

        ethhead[9],ethhead[10],ethhead[11]);

*///输出 IP 地址

iphead = buffer+14; /* Skip Ethernet header */

printf("Source host %d.%d.%d.%d\n",

        iphead[12],iphead[13],

        iphead[14],iphead[15]);

printf("Dest host %d.%d.%d.%d\n",

        iphead[16],iphead[17],

        iphead[18],iphead[19]);

printf("Source, Dest ports %d,%d\n",

        (iphead[20]<<8)+iphead[21],

        (iphead[22]<<8)+iphead[23]);

printf("Layer-4 protocol %d\n",iphead[9]);

/*(buffer+n)=0;

switch(iphead[9]){

    case 6://tcp

        printf("TCP protocol,data:\n%s",buffer+14+20+20);//跳过包头,

到数据部分

        break;

    case 17://udp

        printf("UDP protocol,data:\n%s",buffer+14+20+8);//同上

```

```

        break;

    }

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

}

}

}

```

### 3. 用户使用说明（输入 / 输出规定）

注：请参考下面运行结果截图看此部分

#### 3.1 使用 icmp 协议实现 ping 程序（源码在 实验三/第一题/icmp.c）

编译 gcc icmp.c

程序运行：程序名

如图所示：./a.out

程序运行后，输入目标 ip（你要 ping 的目标），然后回车，等待片刻，将会显示程序执行的结果

#### 3.2 使用原始套接字捕获数据包分析 ftp 协议（源码在 实验三/第二题/rawsniffer.c）

程序编译 gcc rawsniffer.c

程序运行：程序名

如图所示：./a.out

程序运行后便开始抓包，直接使用 ftp 就可查看抓包结果

### 4. 运行结果（要求截图）

#### 3.1 使用 icmp 协议实现 ping 程序

```
实验三 — root@VM_21_212_centos:~/netlib/3/1 — ssh root@wifewifi.com — 7...
[[root@VM_21_212_centos 1]# gcc icmp.c
[[root@VM_21_212_centos 1]# ./a.out
please input ping ip:127.0.0.1
ip:127.0.0.1    icmp_type:8    icmp_id:9313
ip:127.0.0.1    icmp_type:0    icmp_id:9313
ping succeed!
[[root@VM_21_212_centos 1]# ./a.out
please input ping ip:115.159.155.95
ip:115.159.155.95    icmp_type:8    icmp_id:9319
ip:115.159.155.95    icmp_type:0    icmp_id:9319
ping succeed!
[[root@VM_21_212_centos 1]# ./a.out
please input ping ip:114.114.114.114
ip:114.114.114.114    icmp_type:0    icmp_id:9325
ping succeed!
[[root@VM_21_212_centos 1]# ./a.out
```

图 3.1 ping 程序

### 3.2 使用原始套接字捕获数据包分析 ftp 协议





```
实验三 — root@VM_21_212_centos:~/netlib/3/2 — ssh root@wifewifi.com — 82x35

<FuncFlag>0</FuncFlag>
</xml>-----

90 bytes read
-----
Source host 121.42.73.232
Dest host 10.105.21.212
Source, Dest ports 21, 55106
Layer-4 protocol 6
TCP protocol, data:

]??^??0?226 Transfer complete.
ata connection.

12-30-17 04:00PM          383 conn.asp
12-31-17 04:00AM      <DIR>      HttpErrors
12-30-17 05:26PM          3688 index.asp
12-30-17 04:29PM      <DIR>      login
12-30-17 03:16PM          1842 opr.asp
12-30-17 05:19PM          3514 order.asp
12-30-17 05:19PM          4525 shoppingCart.asp
12-30-17 04:13PM      <DIR>      test
12-30-17 03:34PM          221 test.asp
12-30-17 03:14PM      <DIR>      wwwlogs
08-13-14 05:28PM          306 ???t????.txt
?你感兴趣的,回复 @书本编号 (例如 @123 ),我就把这本书推送到你的Kindle推送邮箱上(ㄐ'ω`ㄐ)
回复 帮助 阅读使用帮助
]]></Content>

<FuncFlag>0</FuncFlag>
</xml>-----

226 Transfer complete.
ftp> 54 bytes read
-----
Source host 121.42.73.232
Dest host 10.105.21.212
```

图 3.2.3 查看目录

## 实验4 Winsock I/O 模型的使用

姓名：殷悦

学号：150120526

班级：1504201

### 一、实验环境

Linux 平台

GUN GCC 编译器



```
实验一 — root@VM_21_212_centos:~ — ssh root@wifewifi.com — 77x15
[root@VM_21_212_centos ~]# cat /proc/version
Linux version 3.10.0-514.26.2.el7.x86_64 (builder@kbuilder.dev.centos.org) (g
cc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC) ) #1 SMP Tue Jul 4 15:04:0
5 UTC 2017
[root@VM_21_212_centos ~]#
```

### 二、实验内容

#### 1. 设计思路

##### 4. 使用 select 实现支持多协议回射服务器

先创建 UDP 套接字，然后使用 setsockopt 设置 SO\_REUSEADDR 重复使用该端口，接着创建 TCP 套接字，接着 TCP 监听并绑定，然后将 UDP 套接字和 TCP 套接字设置到同一个 fd\_set 中，使用 select 阻塞等待消息的到达，当消息到达时，select 放行，然后判断是 TCP 的数据还是 UDP 的数据还是出错了。若是 TCP 的数据，则先接受连接，（第一个实验已经学习了创建线程实现并发，第二个实验已经学习了调用 fork 实现 UDP 的并发，此处学习调用 fork 实现 TCP 的并发），然后 fork 一下，父进程 close 接受的连接套接字，然后循环继续。子进程 close 父进程的套接字，然后接收数据，处理数据，回发给客户端，然后退出。若是 UDP 的数据，则接收数据，然后处理后，回发给客户端。

#### 2. 程序清单（要求有详细的注释）

注：由于 Linux 下编码不方便，注释全在实验报告中，源码中无注释

##### 4. 使用 select 实现支持多协议回射服务器

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// #include <sys/types.h>
```

```

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <sys/select.h>

#include <string.h>

#define PORT 8000

int prierr(char *msg){ //输出错误信息

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

    exit(1);

    return 0;

}

int main(int argc,char **argv)

{

    int listenfd,connfd,udpfid,nready;

    char msg[2000];

    pid_t childpid;

    fd_set rset;

    int rcvlen;

    socklen_t len;

    const int on=1;

```



```

    struct sockaddr_in cliaddr,servaddr;

//流式套接字

    if((listenfd=socket(AF_INET,SOCK_STREAM,0))==-1)

        prierr("tcp socket error");


    bzero(&servaddr,sizeof(servaddr)); //初始化变量

    servaddr.sin_family=AF_INET;

    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

    servaddr.sin_port=htons(PORT);

//让 TCP 和 UDP 可以复用 8000 端口

    setsockopt(listenfd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));

    if(bind(listenfd,(struct sockaddr*)&servaddr,sizeof(servaddr))==-1) //绑定 TCP 服务器
端口等信息

        prierr("tcp bind error");


    listen(listenfd,10); //监听 TCP

//创建流式套接字

    if((udpfdf=socket(AF_INET,SOCK_DGRAM,0))==-1)

        prierr("upd socket error");


    bzero(&servaddr,sizeof(servaddr));

    servaddr.sin_family    =AF_INET;

    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

```

```
servaddr.sin_port = htons(PORT);
```

```
if(bind(udpfd,(struct sockaddr*) &servaddr,sizeof(servaddr))==-1) //绑定 UDP 服务器  
端口等信息
```

```
    prierr("udp bind error");
```

```
FD_ZERO(&rset); //清空 fd_set
```

```
while(1){
```

```
    FD_SET(listenfd,&rset); //将 TCP 套接字加入监听
```

```
    FD_SET(udpfd,&rset); //将 UDP 套接字加入监听
```

```
    if((nready=select((listenfd>udpfd)?listenfd+1:udpfd+1
```

```
        ,&rset,NULL,NULL,NULL)) < 0){
```

```
//第一个参数表示文件描述符号中的取值范围，上面的意思是去两个中最大的加 1，那么肯定  
在范围内，然后程序阻塞在这里，有消息就向下走
```

```
    }
```

```
if(FD_ISSET(listenfd,&rset)){ //判读是 TCP 的消息
```

```
    len=sizeof(cliaddr);
```

```
    if((connfd=accept(listenfd,(struct sockaddr*) &cliaddr,&len))==-1)
```

```
        prierr("accept error");
```

```
    if((childpid=fork())==0){ //让子进程来完成 echo 数据工作
```

```

        close(listenfd);

        if((rcvlen=recv(connfd,msg,sizeof(msg),0))==-1)

            prierr("recv error");

        printf("tcp recv:%s\n",msg);

        send(connfd,msg,rcvlen,0);

        printf("tcp send:%s\n",msg);

        exit(0);

    }

    close(connfd);

}

if(FD_ISSET(udpfd,&rset)){ //判断是 UDP 的消息

    len=sizeof(cliaddr);

    if((rcvlen=recvfrom(udpfd,msg,sizeof(msg),0,(struct sockaddr*)

&cliaddr,&len))==-1)

        prierr("recvfrom error");

    printf("udp recv:%s\n",msg);

    sendto(udpfd,msg,rcvlen,0,(struct sockaddr*) &cliaddr,len);

    printf("udp send:%s\n",msg);

}

}

}

```

### 3.用户使用说明（输入 / 输出规定）

注：请参考下面运行结果截图看此部分

#### 4. 使用 select 实现支持多协议回射服务器（源码在 实验四/select.c）

编译 gcc select.c

服务器先运行，客户端再运行

服务器运行：程序名

如图所示：./a.out

服务器已经使用 8000 端口，若要更改端口，则更改源码中#define PORT 8000 的值

UDP 客户端使用第二个实验第一大题的第一个小题或第二题小题的程序

UDP 客户端运行：程序名 c <IP> <端口>

如图所示：./a.out c 127.0.0.1 8000

程序运行后，输入任意字符，按下回车发送数据，然后显示回复的结果

TCP 客户端使用第一个实验第二大题的第一个小题或第二题小题的程序

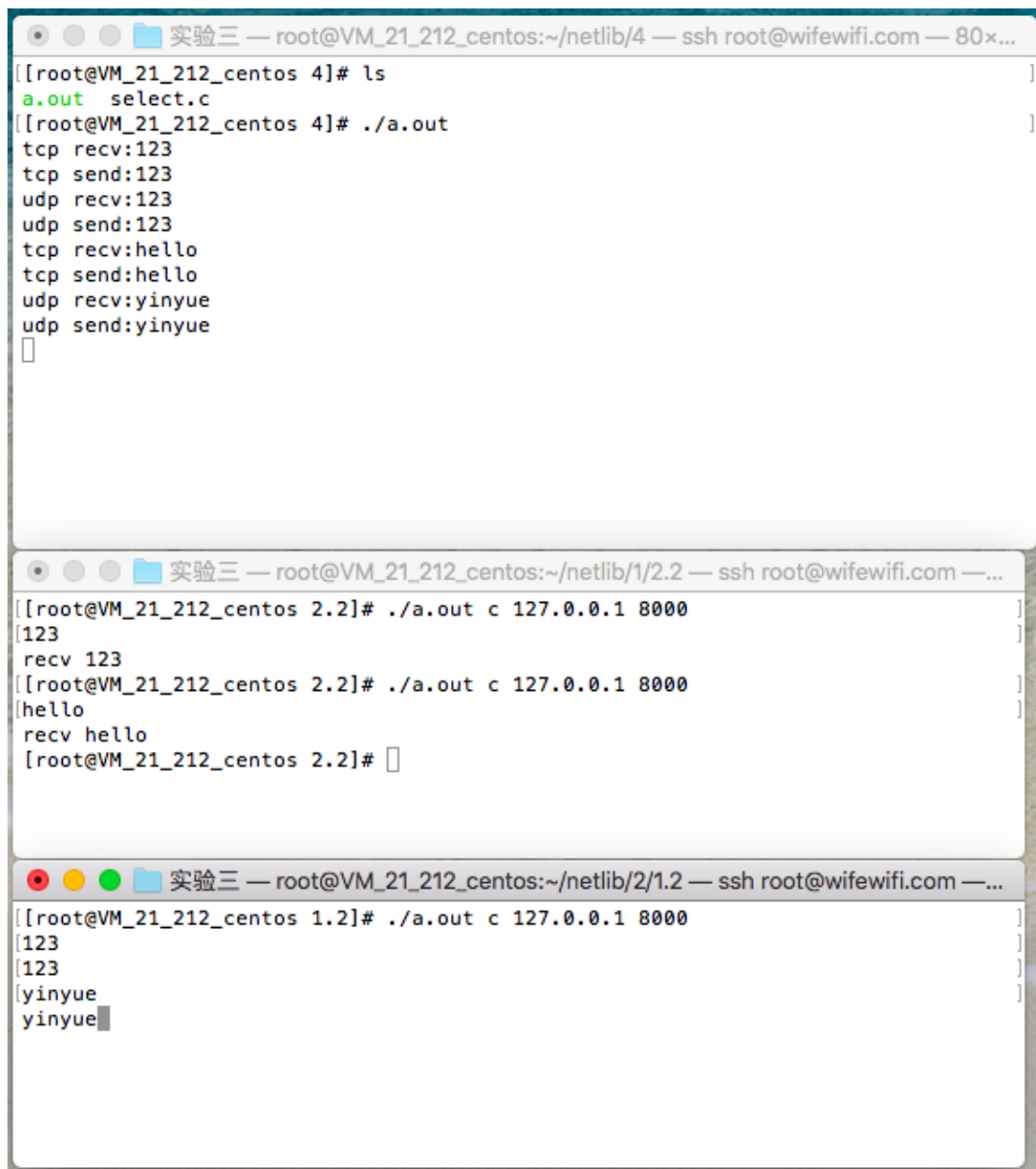
UDP 客户端运行：程序名 c <IP> <端口>

如图所示：./a.out c 127.0.0.1 8000

程序运行后，连接服务器，输入任意字符，按下回车发送数据，然后显示回复的结果

### 4.运行结果（要求截图）

#### 4. 使用 select 实现支持多协议回射服务器



```
实验三 — root@VM_21_212_centos:~/netlib/4 — ssh root@wifewifi.com — 80x...
[[root@VM_21_212_centos 4]# ls
a.out select.c
[[root@VM_21_212_centos 4]# ./a.out
tcp recv:123
tcp send:123
udp recv:123
udp send:123
tcp recv:hello
tcp send:hello
udp recv:yinyue
udp send:yinyue
[]

实验三 — root@VM_21_212_centos:~/netlib/1/2.2 — ssh root@wifewifi.com — ...
[[root@VM_21_212_centos 2.2]# ./a.out c 127.0.0.1 8000
[123
recv 123
[[root@VM_21_212_centos 2.2]# ./a.out c 127.0.0.1 8000
[hello
recv hello
[[root@VM_21_212_centos 2.2]# []

实验三 — root@VM_21_212_centos:~/netlib/2/1.2 — ssh root@wifewifi.com — ...
[[root@VM_21_212_centos 1.2]# ./a.out c 127.0.0.1 8000
[123
[123
[yinyue
[yinyue
[]
```

图 4 使用 select 实现支持多协议回射服务器效果