

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2017221102021

姓 名 彭子为

(实验) 课程名称 计算机网络编程

理论教师 张翔

实验教师 张翔

电子科技大学

实验报告

学生姓名：彭子为 学号：2017221102021 指导教师：张翔

实验地点：信软楼西 304 实验时间：2019.11.17

一、实验名称：

支持客户端之间交互式应答的 UDP 服务器与客户端设计

二、实验学时：4

三、实验目的：

- ① 掌握基于 UDP 协议的客户机与服务器间通信原理。
- ② 掌握通过 UDP 实现客户机与服务器之间的通信及程序的编写。
- ③ 掌握使用 UDP 通过服务器实现客户端之间的通信技术

四、实验原理：

服务器端

1 定义消息的结构体，消息分为登陆、广播、退出三种，分别对应 L、B、Q。当有用户连接到服务器，相当于登陆，断开服务器连接，则为退出。

```
typedef struct msg {  
    char type; /**< 消息类型 */  
    char name[32]; /**< 消息来源标识*/  
    char text[N]; /**< 消息内容*/  
}MSG;
```

2 两个客户端之间是无法直接进行通信的，需要通过服务器，设计思路是在服务器端采用链表来存储每个客户端的连接信息，通过查找链表里存储的客户端网络信息，实现客户机之间的通信。客户与服务器建立连接的时候，则将该客户端连接信息插入到链表里面，断开连接时，相应的从链表里删除。

```
typedef struct node //存储客户端网络信息结构体的链表 {  
    struct sockaddr_in addr;
```

```

    struct node *next;
}listnode,*linklist;
linklist linklist_creat() //创建链表 {
    linklist H;
    H=(linklist)malloc(sizeof(listnode));
    H->next=NULL;
    return H;
}

```

3 使用 UDP 实现客户端与服务器端的消息发送，首先创建一个套接字，

```

int main(int argc, const char *argv[]) {
    struct sockaddr_in serveraddr, clientaddr;
    socklen_t addrlen = sizeof(struct sockaddr); //创建一个套接字
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        errlog("fail to socket");
    }
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = inet_addr(argv[1]);
    serveraddr.sin_port = htons(atoi(argv[2])); /* 服务器与客户端通过 UDP 通信操作
    */ } 将套接字与客户端连接信息结构体绑定。
    //将套接字与网络信息结构体绑定
    if(bind(sockfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
        errlog("fail to bind");
    }
}

```

4 采用多进程处理方式，服务器端父进程接收处理客户端的消息并发送。

```

pid=fork(); /**父进程接收发送**/
linklist H=linklist_creat();
if(recvfrom(sockfd,&msg,sizeof(msg),0,(struct sockaddr*)&(clientaddr),&addrlen) <= 0)
{
    errlog("recvfrom error");
} /**对不同的消息类型进处理**/

```

子进程获取终端输入的内容并发送。

```
memset(&msg,0,sizeof(msg));  
strcpy(msg.name,"server");  
msg.type='B';  
while(1) {  
    fgets(msg.text, N, stdin);  
    msg.text[strlen(msg.text)-1]='\0';  
    sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr*)&(serveraddr),addrlen);  
}
```

5 客户端连接服务器，将连接信息插入到链表里面，采用头插法，sockfd 是套接字描述符，H 存储网络信息的链表头节点，msg 是传输的消息，clientaddr 是服务器端的网络消息。

```
void login(int sockfd,linklist H,MSG msg,struct sockaddr_in clientaddr) {  
    linklist p=H->next;  
    //sprintf(msg.text,"%s 上线了",msg.name);  
    strcpy(msg.text,msg.name);  
    msg.text[strlen(msg.text)]='\0';  
    strcat(msg.text," 上线了");  
    //printf(msg.text);  
    puts(msg.text);  
    while(p!=NULL) {  
        sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)&(p->addr),sizeof(p->addr));  
        // printf("send %s to port %d\n",msg.text,ntohs((p->addr).sin_port));  
        p=p->next;  
    }  
    p=(linklist)malloc(sizeof(listnode)); //使用头插法插入网络信息结构体数据  
    p->addr=clientaddr;  
    p->next=H->next;  
    H->next=p;  
}
```

```

        printf("get client port = %d.\n",ntohs((p->addr).sin_port));
    }

```

6 通过广播发送消息，遍历链表里面的存储的客户端网络信息，将消息发送 给其他客户端。

```

void send_message(int sockfd,linklist H,MSG msg,struct sockaddr_in clientaddr) {
    linklist p = H->next;
    char s[N]={0};
    sprintf(s,"%s 说: %s",msg.name,msg.text);
    strcpy(msg.text,s);
    puts(msg.text);
    while(p) //遍历链表 {
        if(memcmp(&clientaddr,&p->addr,sizeof(clientaddr))!=0) {
            if(sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr*)&(p->addr),size
of(p->addr))<0){
                errlog("fail to sendto"); }
            }
            p=p->next;
        }
    }
}

```

7 客户端断开与服务器连接，服务器从链表连删除相应的网络信息。

```

q=p->next;
p->next=q->next;
free(q);
q=NULL;

```

客户端

1 客户端同样需要定义一个消息结构体和存储网络信息的结构体链表。然后创建一个套接字，

```

int main(int argc, const char *argv[]) {
    MSG msg;
    int sockfd;

```

```

struct sockaddr_in serveraddr;

socklen_t addrlen = sizeof(struct sockaddr);

int maxfd; fd_set infds;

pid_t pid; /**第一步： 创建一个套接字*/

if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    errlog("fail to socket");
}
}

```

客户端创建 socket 后，客户端监听端口，并发送到服务器端。

```

serveraddr.sin_family = AF_INET;

serveraddr.sin_port=htons(6666);

serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);

while(1){
    FD_ZERO(&infds);

    FD_SET(fileno(stdin),&infds);

    FD_SET(sockfd,&infds);

    maxfd=max(fileno(stdin),sockfd)+1;

    if (select(maxfd,&infds,NULL,NULL,NULL)==-1){
        printf("select io error\n");

        return -1;
    }

    if(sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)&serveraddr,addrlen)<0){
        errlog("fail to sendto");
    }
}
}

```

五、实验内容：

- ① 使用 vim 等编辑工具，编写基于 UDP 的客户机之间的通信程序，服务器存储客户机的网络信息，实现客户端与客户端之间的认识及消息的发送。服务器主要工作是

维护客户端链表。

② 使用 gcc、gdb 等编译、调试工具对代码进行编译和调试，生成可执行程序。

③ 在 Linux 虚拟机上执行程序，检验服务器与客户端是否可按照预期正常工作。

1、Half Ad hoc 模式 与 Infrastructure 模式。核心问题在于协议的设计：

1) 通信节点间的通信流程

2) 通信节点间传输的 PDU 的设计（复习：PDU 包含两部分，Header & Payload）

2、无论哪种模式，用户都需要首先登录服务器登记，所以服务器需要有存储活跃节点的数据结构，并且为了保证服务器存储的信息有效，还需要心跳机制来确保活跃节点的检测。

3、Half Ad hoc 模式下，通信初始发起节点在查询了通信目的节点的 socket address 后，就会直接和通信目的节点通信。Infrastructure 模式下，通信双方节点间的通信内容都是通过服务器转发，所以服务器需要设计进行转发判断的基础数据结构（这个数据结构和 3 提到的数据结构是什么关系？）。

4、因为构建的是即时通信，所以传输层采用 UDP，但是为了避免消息丢失，还是需要可靠性保证，所以需要在应用层协议提供 ACK 机制（PDU 如何设计？）

5、除了最理想的状态，要考虑各种边界情况，比如目的节点尚未登录服务器怎么办；或者目的节点突然下线了怎么办；请服务器转发报文，没有收到服务器成功转发的应答怎么办，等等。

6、具体编码之前，设计很重要，请使用 ProcessOn 等工具绘制两种场景的协议交互过程（这样思路会更清晰），应用 UML 协作图等，同时进行 PDU 的设计（Payload 自然就是要传输的文本信息，Header 中则应包括各种控制信息）

六、实验器材（设备、元器件）：

PC 机一台

六、实验步骤：

假设有两个客户端 A 和 B，服务器 S。A 和 B 的通信可以采用两种方式实现：

(1) 自组织模式：A 和 B 分别在 S 注册登记，随后 A 和 B 之间通过 S 提供的信息实现直接通信。

(2) 基础架构模式：A 和 B 分别在 S 注册登记，随后 A 和 B 之间的通信通过 S 中转。

- ProcessOn 等工具绘制两种场景的协议交互过程
- 编码实现

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）

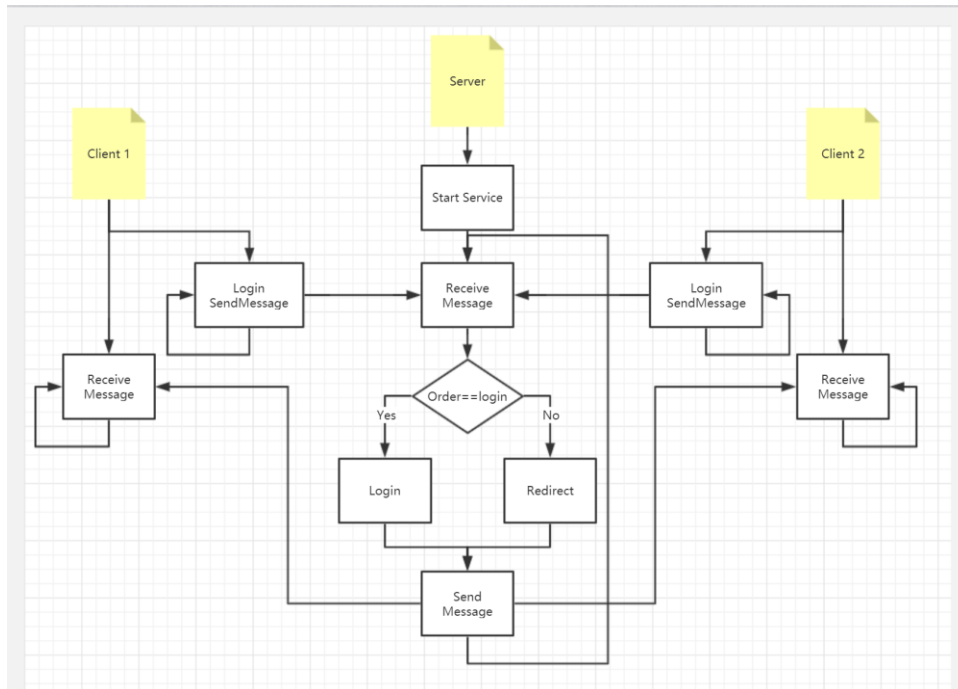


图 1 基础架构模式实现结构

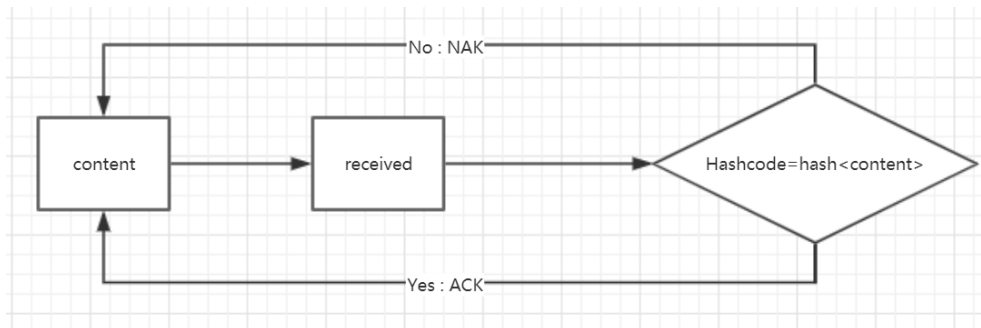


图 2 ACK 确认机制

- 1、无论哪种模式，用户都需要首先登录服务器登记，所以服务器需要有存储活跃节点的数据结构，并且为了保证服务器存储的信息有效，还需要心跳机制来确保活跃节点的检测。

注：本次实现会在用户每次更新消息时自动记录并更新用户的地址信息，一开始则会进行多次登录与同步验证。

- 2、Half Ad hoc 模式下,通信初始发起节点在查询了通信目的节点的 socker address 后,就会直接和通信目的节点通信。Infrastructure 模式下,通信双方节点间的通信内容都是通过服务器转发,所以服务器需要设计进行转发判断的基础数据结

构。

注：服务器使用 map 数据结构作用户与地址的映射。

```
map<string, pair<string, int>> users;
```

以上结构帮助我们高效完成用户信息的查找。

- 3、因为构建的是即时通信，所以传输层采用 UDP，但是为了避免消息丢失，还是需要可靠性保证，所以需要在应用层协议提供 ACK 机制（PDU 如何设计？）

注：ACK 机制由统一的 hash 函数对内容进行哈希比对完成验证完整性。如未收到 ACK 则显示 corrupted 并要求进行重传。

- 4、除了最理想的状态，要考虑各种边界情况，比如目的节点尚未登录服务器怎么办；或者目的节点突然下线了怎么办；请服务器转发报文，没有收到服务器成功转发的应答怎么办，等等。

注：节点未登录服务器则向用户返回 User not exist，表示无法实时通信。心跳机制判定节点下线即清楚数据结构中的用户信息。没有收到服务器成功转发的应答即为未收到 ACK，超时显示 Corrupted 并重传。

- 5、具体编码之前，设计很重要，请使用 ProcessOn 等工具绘制两种场景的协议交互过程（这样思路就会更清晰），应用 UML 协作图等，同时进行 PDU 的设计（Payload 自然就是要传输的文本信息，Header 中则应包括各种控制信息）

注：如下图即为用户登录时和用户发送内容时的 Json 格式设计，由于内容只有 content 一个参数，所以不再严格区分 Header 和 Payload,全部放在一个内容包内。

```
out:{
    "order":      "login",
    "username":   "A"
}
```

图 3 用户登录

```
{
    "order":      "chat",
    "username":   "B",
    "sendto":     "A",
    "content":    "214\n",
    "hashcode":   "782879189"
}
Message from B:
214
```

图 4 发送内容与显示格式

Server 端代码

```
#include "unp.h"
```

```

#undef max
#undef min
#include <map>
#include <iostream>
#include <string>
#include <cjson/cJSON.h>

using namespace std;
map<string, pair<string, int>> users;

void process(int sockfd);

unsigned int SDBMHash(char *str)
{
    unsigned int hash = 0;

    while (*str)
    {
        // equivalent to: hash = 65599*hash + (*str++);
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }

    return (hash & 0x7FFFFFFF);
}

int
main(int argc, char **argv)
{
    int listenfd, connfd, sockfd, nready, maxfdp1, pid;
    char mesg[MAXLINE];
    pid_t childpid;
    fd_set rset;
    ssize_t n;
    socklen_t len;
    const int on = 1;
    struct sockaddr_in cliaddr, servaddr;
    void sig_chld(int);

    /* 4create listening TCP socket */
    //listenfd = Socket(AF_INET, SOCK_STREAM, 0);
    //bzero(&servaddr, sizeof(servaddr));
    //servaddr.sin_family = AF_INET;
    //servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    //servaddr.sin_port = htons(SERV_PORT);

```

```

        //Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, s
sizeof(on));
        //Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
        //Listen(listenfd, LISTENQ);

/* 4create UDP socket */
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
bind(sockfd, (SA *) &servaddr, sizeof(servaddr));
/* end udpselect01 */

process(sockfd);
}

pair<string,int> LoginUser(string s,string ip,int port){

    cJSON* root = cJSON_Parse(s.c_str());

    if (root == NULL)
    {
        cJSON_Delete(root);
        return make_pair(string("123"),-1);
    }

    cJSON* itemContent = cJSON_GetObjectItem(root, "content");
    cJSON *hashCode = cJSON_GetObjectItem(root, "hashCode");

    if (itemContent!=nullptr && hashCode!=nullptr && SDBMHash(i
temContent->valuelstring)!=atoi(hashCode->valuelstring)){
        return make_pair(string("123"),-2);//corrupted
    }

    cJSON* itemOrder = cJSON_GetObjectItem(root, "order");
    cJSON* itemName = cJSON_GetObjectItem(root, "username");
    cJSON* itemSendto = cJSON_GetObjectItem(root, "sendto");

    users[itemName->valuelstring]=std::make_pair(ip,port);

    if (!strcmp(itemOrder->valuelstring,"login")){

```

```

        std::cout<<"Login successful:["<<ip<<":"<<port<<"]"<<itemName->valuestring;
        cJSON_Delete(root);
        return make_pair(string("123"),0);//Login
    }

    if (users.find(itemSendto->valuestring)==users.end()){
        cJSON_Delete(root);
        return make_pair(string("123"),-1);//user not exist
    }

    string temp=string(itemSendto->valuestring);
    cJSON_Delete(root);
    return users[temp];//return user information
}

void writ(pair<string,int> target,string buf, int sockfd, const
struct sockaddr *pservaddr, socklen_t servlen){

    sendto(sockfd, buf.c_str(), MAXLINE, 0, pservaddr, servlen)
;

}

void redirect(pair<string,int> target,string buff){
    int sockfd;
    struct sockaddr_in servaddr,cliaddr;

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(target.second);
    inet_pton(AF_INET, target.first.c_str(), &servaddr.sin_addr
);

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&cliaddr, sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliaddr.sin_port = htons(0); /* force assignment of ephemeral port */
    bind(sockfd, (SA *) &cliaddr, sizeof(cliaddr));

    writ(target, buff, sockfd, (SA *) &servaddr, sizeof(servaddr));
}

```

```

}

void process(int sockfd){
    printf("Server Start at Port:%d\n",SERV_PORT);
    while(1){
        char buff[1024]={0};
        SA * local;
        ssize_t s;
        socklen_t len;
        //接收数据（描述符，放到哪，内存大小，阻塞，从哪来，数据大小）
        s = recvfrom(sockfd,buff,1023,0,(struct sockaddr*)&local,&len);
        struct sockaddr_in *sock = ( struct sockaddr_in*)&local;

        int port = ntohs(sock->sin_port);
#ifdef __MINGW32__ //windows 上打印方式
            printf("ip:port  %s : %d",inet_ntoa(sock->sin_addr),port);
#else //linux 上打印方式
            struct in_addr in  = sock->sin_addr;
            char str[INET_ADDRSTRLEN]; //INET_ADDRSTRLEN 这个宏系统默认定义 16
            //成功的话此时 IP 地址保存在 str 字符串中。
            inet_ntop(AF_INET,&in, str, sizeof(str));
#endif
            printf("Server Receive Message from ip:port  %s : %d:\n%s",str,port,buff);
            pair<string,int> target;
            target>LoginUser(buff,str,port);
            if (target.second===-1){
                printf("User do not exist!\n");
                redirect(std::make_pair(str,port),"User do not exist!\n");//redirect message to sender
                continue;
            }else if (target.second==0){
                continue;
            }else if (target.second===-2){
                redirect(std::make_pair(str,port),"NAK");
                continue;
            }
            redirect(std::make_pair(str,port),"ACK");
            cout<<"redirecting\n";
            redirect(target,buff);//redirect message to receiver
    }
}

```

```
}  
}
```

Client 端代码

```
#include "unp.h"  
#undef max  
#undef min  
#include <cjson/cJSON.h>  
#include <string>  
#include <iostream>  
  
unsigned int SDBMHash(char *str)  
{  
    unsigned int hash = 0;  
  
    while (*str)  
    {  
        // equivalent to: hash = 65599*hash + (*str++);  
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;  
    }  
  
    return (hash & 0x7FFFFFFF);  
}  
  
void cli(FILE *fp, int sockfd, const struct sockaddr *pservaddr,  
        socklen_t servlen, char * myname)  
{  
    char sendLineBuf[MAXLINE];  
    char buf[10];  
    reinput:  
        printf("type in the user you want to chat with:");  
        scanf("%s",buf);  
  
    while(fgets(sendLineBuf, MAXLINE, fp) != NULL){  
        cJSON *root=cJSON_CreateObject();  
        cJSON *item1=cJSON_CreateString("chat");  
        cJSON_AddItemToObject(root,"order",item1);  
        cJSON *item2=cJSON_CreateString(myname);  
        cJSON_AddItemToObject(root,"username",item2);  
        cJSON *item3=cJSON_CreateString(buf);  
        cJSON_AddItemToObject(root,"sendto",item3);  
        cJSON *temp=cJSON_CreateString(sendLineBuf);  
        cJSON_AddItemToObject(root,"content",temp);  
        cJSON *hashcode=cJSON_CreateString(std::to_string(SDBMHash(sendLineBuf)).c_str());
```

```

        cJSON_AddItemToObject(root,"hashcode",hashcode);

        //printf("out:%s\n",cJSON_Print(root));
        //printf("%ld\n",sizeof(cJSON_Print(root)));

        sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr,
, servlen);
        cJSON_Delete(root);
    }
}

void login(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen,char *p)
{
    cJSON *root=cJSON_CreateObject();
    cJSON *item1=cJSON_CreateString("login");
    cJSON_AddItemToObject(root,"order",item1);
    cJSON *item2=cJSON_CreateString(p);
    cJSON_AddItemToObject(root,"username",item2);

    //printf("out:%s\n",cJSON_Print(root));

    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    cJSON_Delete(root);
}

void cliread(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen){
    char sendLineBuf[MAXLINE], recvLineBuf[MAXLINE + 1];
    int nrecv;

    while(1){
        nrecv = recvfrom(sockfd, recvLineBuf, MAXLINE, 0, NULL, NULL);// pay attention to the last two NULLs
        recvLineBuf[nrecv] = 0;
        //std::cout<<recvLineBuf<<std::endl;
        if (!strcmp(recvLineBuf,"User do not exist!\n")){
            printf("Error:User not exist!");
            exit(0);
        }
    }
}

```

```

        }else if (!strcmp(recvLineBuf,"ACK")){
            printf("ACK::Message send successfully!\n");
            continue;
        }else if (!strcmp(recvLineBuf,"NAK")){
            printf("NAK::Message corrupted,please send again!\n
");
            continue;
        }
        cJSON* root = cJSON_Parse(recvLineBuf);

        if (root == NULL)
        {
            cJSON_Delete(root);
        }

        cJSON* itemName = cJSON_GetObjectItem(root, "username")
;
        cJSON* itemContent = cJSON_GetObjectItem(root, "content
");

        printf("Message from %s:\n    %s",itemName->valuelstring
,itemContent->valuelstring);

        cJSON_Delete(root);
    }
}

void process(FILE *fp, int sockfd, const struct sockaddr *pserv
addr, socklen_t servlen, char * myname){
    if (fork()==0)
        cli(stdin, sockfd, pservaddr, servlen, myname);
    else
        cliread(stdin, sockfd, pservaddr, servlen);
}

int
main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in servaddr,cliaddr;
    if (argc != 3){
        printf("usage: cli <ServerIPAddress> <username>");
        exit(0);
    }
}

```



```

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(SERV_PORT);
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bzero(&cliaddr, sizeof(cliaddr));
cliaddr.sin_family = AF_INET;
cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
cliaddr.sin_port = htons(0); /* force assignment of ephemeral port */
bind(sockfd, (SA *) &cliaddr, sizeof(cliaddr));

login(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr), argv[2]);

process(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr), argv[2]);

exit(0);
}

```

运行截图与说明：

```

veils@Surface:/mnt/c/wsl/ComputerNetwork$ ./server
Server Start at Port:9877
Server Receive Message from ip :port 0.0.0.0 : 0:
{
  "order": "login",
  "username": "A"
}Login successful:[0.0.0.0:0]A
Server Receive Message from ip :port 127.0.0.1 : 55881:
{
  "order": "login",
  "username": "A"
}Login successful:[127.0.0.1:55881]AServer Receive Message from ip:port 127.0.0.1 : 55881:
{
  "order": "login",
  "username": "A"
}Login successful:[127.0.0.1:55881]AServer Receive Message from ip:port 127.0.0.1 : 54489:
{
  "order": "login",
  "username": "B"
}Login successful:[127.0.0.1:54489]BServer Receive Message from ip:port 127.0.0.1 : 54489:
{
  "order": "login",
  "username": "B"
}Login successful:[127.0.0.1:54489]BServer Receive Message from ip:port 127.0.0.1 : 54489:
{
  "order": "login",
  "username": "C"
}Login successful:[127.0.0.1:54489]C

```

图 5 启动 server 与三个 client

如上图所示，我们启动一个 server 服务器以及三个 client 客户端，客户端名称为 A、B、C，我们通过这个名称标识接受消息的目标用户，程序一开始会先对自身进行登录，如图所示，此时服务器已经完成了登录的操作，客户端同时开始了监听。


```
问题 输出 调试控制台 终端
"sendto": "C",
"content": "\n",
"hashcode": "10"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 54490:
{
  "order": "chat"
,
  "username": "C",
  "sendto": "A",
  "content": "\n",
  "hashcode": "10"
}
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 55881:
{
  "order": "chat"
,
  "username": "A",
  "sendto": "B",
  "content": "Hello
,Bli'm pengziwei\n",
  "hashcode": "29521
5197"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 54489:
{
  "order": "chat"
,
  "username": "B",
  "sendto": "C",
  "content": "Hello
,C!i'm B2017221102021BBB\n",
  "hashcode": "96266
1752"
}redirecting
}

veils@Surface:/mnt/c/wsl/Compute
work$ ./client 127.0.0.1 A
type in the user you want to cha
t with:B
ACK::Message send successfully!
Message from C:
Hello,Bli'm pengziwei!
ACK::Message send successfully!
[]

veils@Surface:/mnt/c/wsl/Compute
work$ ./client 127.0.0.1 B
type in the user you want to cha
t with:Message from A:
C
ACK::Message send successfully!
Message from A:
Hello,Bli'm pengziwei!
Hello,C!i'm B2017221102021BBB
ACK::Message send successfully!
[]

veils@Surface:/mnt/c/wsl/Comput
er_Network$ ./client 127.0.0.1
C
type in the user you want to ch
at with:Message from B:
A
ACK::Message send successfully!
Message from B:
Hello,C!i'm B2017221102021B
BB
ACK::Message send successfully!
[]
```

图 8 B->C

如上图，B 向 C 发送了一条消息，即 “Hello,C!i'm B2017221102021BBB”，消息在 server 中转后 C 成功接收到这条消息。

```
问题 输出 调试控制台 终端
"hashcode": "10"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 55881:
{
  "order": "chat"
,
  "username": "A",
  "sendto": "B",
  "content": "Hello
,Bli'm pengziwei\n",
  "hashcode": "29521
5197"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 54489:
{
  "order": "chat"
,
  "username": "B",
  "sendto": "C",
  "content": "Hello
,C!i'm B2017221102021BBB\n",
  "hashcode": "96266
1752"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 54490:
{
  "order": "chat"
,
  "username": "C",
  "sendto": "A",
  "content": "Hello
Ali'm CCCCCCCCC!\n",
  "hashcode": "19724
44040"
}redirecting
}

veils@Surface:/mnt/c/wsl/Compute
work$ ./client 127.0.0.1 A
type in the user you want to cha
t with:B
ACK::Message send successfully!
Message from C:
Hello,Bli'm pengziwei!
ACK::Message send successfully!
Message from C:
Hello Ali'm CCCCCCCCC!
[]

veils@Surface:/mnt/c/wsl/Compute
work$ ./client 127.0.0.1 B
type in the user you want to cha
t with:Message from A:
C
ACK::Message send successfully!
Message from A:
Hello,Bli'm pengziwei!
Hello,C!i'm B2017221102021BBB
ACK::Message send successfully!
[]

veils@Surface:/mnt/c/wsl/Comput
er_Network$ ./client 127.0.0.1
C
type in the user you want to ch
at with:Message from B:
A
ACK::Message send successfully!
Message from B:
Hello,C!i'm B2017221102021B
BB
ACK::Message send successfully!
Hello Ali'm CCCCCCCCC!
ACK::Message send successfully!
[]
```

图 9 C->A

如上图，C 向 A 发送了一条消息，即 “Hello A!i'm CCCCCCCCC!”，消息在 server 中转后 A 成功接收到这条消息。

异常处理 1:

如果遇到目标用户不存在的情况会怎么样呢？

```
问题 输出 调试控制台 终端
veils@Surface:/mnt/c/wsl/Computer_Network$ ./server
Server Start at Port:9877
Server Receive Message from ip:port 0.0.0.0 : 0:
{
  "order": "login",
  "username": "A"
}Login successful:[0.0.0.0:0]AServer Receive Message from ip:port
127.0.0.1 : 61833:
{
  "order": "login",
  "username": "A"
}Login successful:[127.0.0.1:61833]AServer Receive Message from i
p:port 127.0.0.1 : 61833:
{
  "order": "login",
  "username": "A"
}Login successful:[127.0.0.1:61833]AServer Receive Message from i
p:port 127.0.0.1 : 61833:
{
  "order": "chat",
  "username": "A",
  "sendto": "B",
  "content": "\n",
  "hashcode": "10"
}User do not exist!
[]

2: server, bash
veils@Surface:/mnt/c/wsl/Computer_Network$ ./client 127.0.0.1 A
type in the user you want to chat with:B
Error:User not exist!veils@Surface:/mnt/c/wsl/Computer_Network$
```

图 10 User not exist

当我们联系没有注册的 B 时，服务器返回了“User do not exist!”信息，客户端无法连接而退出。

异常处理 2:

客户端重复登陆&&客户退出

```
问题 输出 调试控制台 终端
"sendto": "B",
"content": "\n",
"hashcode": "10"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 50274:
{
  "order": "chat"
,
  "username": "B",
  "sendto": "A",
  "content": "\n",
  "hashcode": "10"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 50273:
{
  "order": "chat"
,
  "username": "A",
  "sendto": "B",
  "content": "hi, B
\n",
  "hashcode": "18314
92157"
}redirecting
Server Receive Message from ip
:port 127.0.0.1 : 50274:
{
  "order": "chat"
,
  "username": "B",
  "sendto": "A",
  "content": "hi, A
\n",
  "hashcode": "18314
26558"
}redirecting
[]

veils@Surface:/mnt/c/wsl/Comput
er_Network$ ./client 127.0.0.1 A
type in the user you want to cha
t with:Message from A:
A
ACK::Message send successfully!
Message from A:
hi, B
hi, A
ACK::Message send successfully!
[]

2: server, client, client t
veils@Surface:/mnt/c/wsl/Comput
er_Network$ ./client 127.0.0.1 B
type in the user you want to cha
t with:Message from A:
A
ACK::Message send successfully!
Message from A:
hi, B
hi, A
ACK::Message send successfully!
[]
```

图 11 普通连接

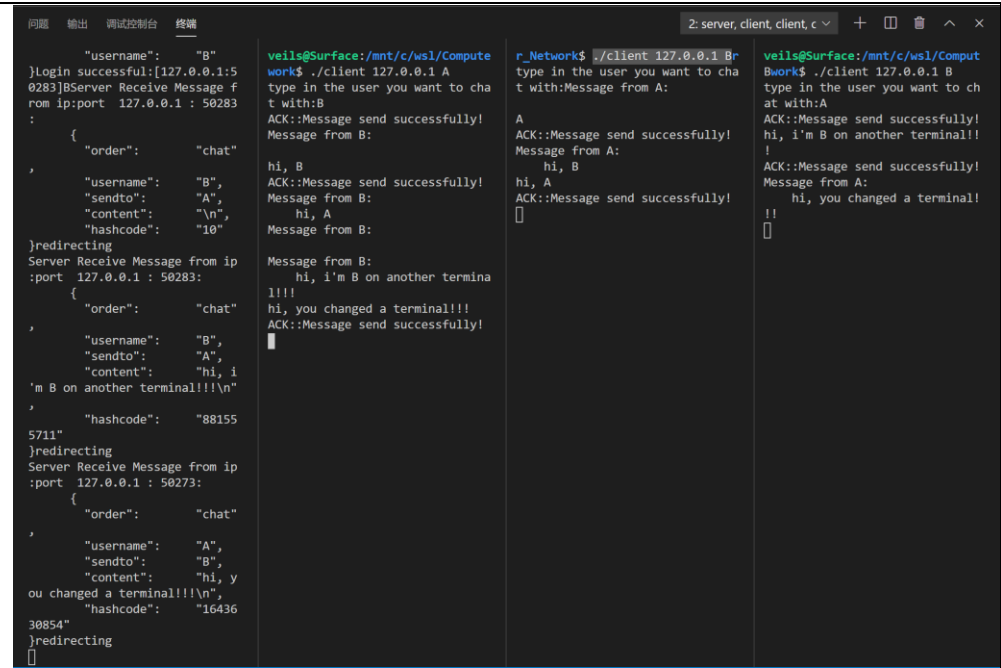


图 12 重复登陆

如图所示，A 和 B 正常通信的情况下，B 在另一个终端登录了，此时服务器的地址映射表发生更新，之后原设备退出连接。此后 A 就会与新的终端相通信。

相应的，在客户端退出后，向服务器发送 `exit` 指令，服务器清除映射表中的地址项即可。

对于自组织模式，需要新增与服务器进行目的地址请求的设计。其对应报文格式如下图所示。

```
4706]BServer Receive Message f
rom ip:port 127.0.0.1 : 55990
:
{
    "order":      "getma
p",
    "username":   "A",
    "sendto":     "B",
    "hashcode":   "66"
}{
```

图 13 用户 A 请求用户 B 地址

```

type in the user you want to cha
t with:B
getting address, please wait:ACK
::Message send successfully!
{
    "order":      "map",
    "username":   "B",
    "address":    "127.0.0
.1",
    "port": "54706"
}adding map
B127.0.0.154706
127.0.0.154706
out:{
    "order":      "chat",
    "username":   "A",
    "sendto":     "B",
    "content":    "\n",
    "hashcode":   "10"
}
B127.0.0.154706

```

图 14 客户端 A 获取 B 的地址并发送默认链接报文

如上图，通过指令为 getmap 的报文获取了指令为 map 的报文获取了用户 B 的地址与端口号信息。此外，客户端转发功能不再需要，只需要支持心跳机制和用户地址信息的映射反馈功能即可。

Server 端代码

```

#include "unp.h"
#undef max
#undef min
#include <map>
#include <iostream>
#include <string>
#include <cjson/cJSON.h>

using namespace std;
map<string,pair<string,int>> users;

void process(int sockfd);

unsigned int SDBMHash(char *str)
{
    unsigned int hash = 0;

    while (*str)
    {

```

```

        // equivalent to: hash = 65599*hash + (*str++);
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }

    return (hash & 0x7FFFFFFF);
}

int
main(int argc, char **argv)
{
    int listenfd, connfd, sockfd, nready, maxfdp1, pid;
    char mesg[MAXLINE];
    pid_t childpid;
    fd_set rset;
    ssize_t n;
    socklen_t len;
    const int on = 1;
    struct sockaddr_in cliaddr, servaddr;
    void sig_chld(int);

    /* 4create listening TCP socket */
    //listenfd = Socket(AF_INET, SOCK_STREAM, 0);
    //bzero(&servaddr, sizeof(servaddr));
    //servaddr.sin_family = AF_INET;
    //servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    //servaddr.sin_port = htons(SERV_PORT);
    //Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, size
of(on));
    //Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
    //Listen(listenfd, LISTENQ);

    /* 4create UDP socket */
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);
    bind(sockfd, (SA *) &servaddr, sizeof(servaddr));
    /* end udpselect01 */

    process(sockfd);
}

```

```

pair<string,int> LoginUser(string s,string ip,int port, string * u
sr){

    cJSON* root = cJSON_Parse(s.c_str());

    if (root == NULL)
    {
        cJSON_Delete(root);
        return make_pair(string("123"),-1);
    }

    cJSON* itemName = cJSON_GetObjectItem(root, "username");

    cJSON* itemOrder = cJSON_GetObjectItem(root, "order");
    users[string(itemName->valuelstring)]=std::make_pair(ip,port);

    if (strcmp(itemOrder->valuelstring,"getmap")){
        cJSON* itemContent = cJSON_GetObjectItem(root, "content");
        cJSON *hashcode = cJSON_GetObjectItem(root, "hashcode");

        if (itemContent!=nullptr && hashcode!=nullptr && SDBMHash(
itemContent->valuelstring)!=atoi(hashcode->valuelstring)){
            return make_pair(string("123"),-2);//corrupted
        }
    }

    cJSON* itemSendto = cJSON_GetObjectItem(root, "sendto");

    if (!strcmp(itemOrder->valuelstring,"login")){
        std::cout<<"Login successful:["<<ip<<":"<<port<<"]"<<itemN
ame->valuelstring;
        cJSON_Delete(root);
        return make_pair(string("123"),0);//Login
    }

    if (users.find(itemSendto->valuelstring)==users.end()){
        cJSON_Delete(root);
        return make_pair(string("123"),-1);//user not exist
    }

    string temp=string(itemSendto->valuelstring);
    *usr=temp;
    cJSON_Delete(root);
    return users[temp];//return user information
}

```



```

}

void writ(pair<string,int> target,string buf, int sockfd, const st
ruct sockaddr *pservaddr, socklen_t servlen){

    sendto(sockfd, buf.c_str(), MAXLINE, 0, pservaddr, servlen);

}

void redirect(pair<string,int> target,string buff){
    int sockfd;
    struct sockaddr_in servaddr,cliaddr;

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(target.second);
    inet_pton(AF_INET, target.first.c_str(), &servaddr.sin_addr);
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&cliaddr, sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliaddr.sin_port = htons(0); /* force assignment of ephemeral
port */
    bind(sockfd, (SA *) &cliaddr, sizeof(cliaddr));

    writ(target, buff, sockfd, (SA *) &servaddr, sizeof(servaddr))
;

}

string make_Address(pair<string,int> t,string user){
    cJSON *root=cJSON_CreateObject();
    cJSON *item1=cJSON_CreateString("map");
    cJSON_AddItemToObject(root,"order",item1);
    cJSON *item2=cJSON_CreateString(user.c_str());
    cJSON_AddItemToObject(root,"username",item2);
    cJSON *item3=cJSON_CreateString(t.first.c_str());
    cJSON_AddItemToObject(root,"address",item3);
    cJSON *item4=cJSON_CreateString(std::to_string(t.second).c_str
());
    cJSON_AddItemToObject(root,"port",item4);
    string temp=string(cJSON_Print(root));
    cout<<temp<<endl;
    cJSON_Delete(root);
}

```

```

        return temp;
    }

void process(int sockfd){
    printf("Server Start at Port:%d\n",SERV_PORT);
    while(1){
        char buff[1024]={0};
        SA * local;
        ssize_t s;
        socklen_t len;
        //接收数据（描述符，放到哪，内存大小，阻塞，从哪来，数据大小）
        s = recvfrom(sockfd,buff,1023,0,(struct sockaddr*)&local,&
len);

        struct sockaddr_in *sock = ( struct sockaddr_in*)&local;
        int port = ntohs(sock->sin_port);
        #ifdef __MINGW32__ //windows 上打印方式
            printf("ip:port  %s : %d",inet_ntoa(sock->sin_addr),po
rt);
        #else //linux 上打印方式
            struct in_addr in = sock->sin_addr;
            char str[INET_ADDRSTRLEN]; //INET_ADDRSTRLEN 这个宏系
统默认定义 16
            //成功的话此时 IP 地址保存在 str 字符串中。
            inet_ntop(AF_INET,&in, str, sizeof(str));
        #endif
        printf("Server Receive Message from ip:port  %s : %d:\n
%s",str,port,buff);
        pair<string,int> target;
        string usr;
        target=LoginUser(buff,str,port,&usr);
        if (target.second== -1){
            printf("User do not exist!\n");
            redirect(std::make_pair(str,port),"User do not exist!\
n");//redirect message to sender
            continue;
        }else if (target.second==0){
            continue;
        }else if (target.second== -2){
            redirect(std::make_pair(str,port),"NAK");
            continue;
        }
        redirect(std::make_pair(str,port),"ACK");
        redirect(std::make_pair(str,port),make_Address(target,usr)
.c_str());
    }
}

```

```
}  
}
```

Client 端代码

```
#include "unp.h"  
#undef max  
#undef min  
#include <cjson/cJSON.h>  
#include <string>  
#include <iostream>  
#include <map>  
  
using namespace std;  
  
map<string, pair<string, int>> users;  
  
unsigned int SDBMHash(char *str)  
{  
    unsigned int hash = 0;  
  
    while (*str)  
    {  
        // equivalent to: hash = 65599*hash + (*str++);  
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;  
    }  
  
    return (hash & 0x7FFFFFFF);  
}  
  
void writ(pair<string, int> target, string buf, int sockfd, const struct  
sockaddr *pservaddr, socklen_t servlen){  
  
    sendto(sockfd, buf.c_str(), MAXLINE, 0, pservaddr, servlen);  
  
}  
  
void redirect(pair<string, int> target, string buff){  
    int sockfd;  
    struct sockaddr_in servaddr, cliaddr;  
  
    bzero(&servaddr, sizeof(servaddr));  
    servaddr.sin_family = AF_INET;  
    servaddr.sin_port = htons(target.second);  
    inet_pton(AF_INET, target.first.c_str(), &servaddr.sin_addr);  
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

```

    bzero(&cliaddr, sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliaddr.sin_port = htons(0); /* force assignment of ephemeral
port */
    bind(sockfd, (SA *) &cliaddr, sizeof(cliaddr));

    writ(target, buff, sockfd, (SA *) &servaddr, sizeof(servaddr))
;
}

void cli(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen, char * myname, char * buf)
{
    char sendLineBuf[MAXLINE];

    while(fgets(sendLineBuf, MAXLINE, fp) != NULL){
        cJSON *root=cJSON_CreateObject();
        cJSON *item1=cJSON_CreateString("chat");
        cJSON_AddItemToObject(root,"order",item1);
        cJSON *item2=cJSON_CreateString(myname);
        cJSON_AddItemToObject(root,"username",item2);
        cJSON *item3=cJSON_CreateString(buf);
        cJSON_AddItemToObject(root,"sendto",item3);
        cJSON *temp=cJSON_CreateString(sendLineBuf);
        cJSON_AddItemToObject(root,"content",temp);
        cJSON *hashcode=cJSON_CreateString(std::to_string(SDBMHash
(sendLineBuf)).c_str());
        cJSON_AddItemToObject(root,"hashcode",hashcode);

        printf("out:%s\n",cJSON_Print(root));
        //printf("%ld\n",sizeof(cJSON_Print(root)));
        cout<<buf<<users[string(buf)].first<<users[string(buf)].second<<endl;
        redirect(std::make_pair(users[string(buf)].first,users[string(buf)].second),cJSON_Print(root));
        //sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
        cJSON_Delete(root);
    }
}

```

```

void login(FILE *fp, int sockfd, const struct sockaddr *pservaddr,
socklen_t servlen, char *p)
{
    cJSON *root=cJSON_CreateObject();
    cJSON *item1=cJSON_CreateString("login");
    cJSON_AddItemToObject(root,"order",item1);
    cJSON *item2=cJSON_CreateString(p);
    cJSON_AddItemToObject(root,"username",item2);

    printf("out:%s\n",cJSON_Print(root));

    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, servlen);
    cJSON_Delete(root);
}

void cliread(FILE *fp, int sockfd, const struct sockaddr *pservaddr, socklen_t servlen){
    char sendLineBuf[MAXLINE], recvLineBuf[MAXLINE + 1];
    int nrecv;

    while(1){
        nrecv = recvfrom(sockfd, recvLineBuf, MAXLINE, 0, NULL, NULL); // pay attention to the last two NULLs
        recvLineBuf[nrecv] = 0;
        //std::cout<<recvLineBuf<<std::endl;
        if (!strcmp(recvLineBuf,"User do not exist!\n")){
            printf("Error:User not exist!");
            exit(0);
        }else if (!strcmp(recvLineBuf,"ACK")){
            printf("ACK::Message send successfully!\n");
            continue;
        }else if (!strcmp(recvLineBuf,"NAK")){
            printf("NAK::Message corrupted,please send again!\n");
            continue;
        }
        cJSON* root = cJSON_Parse(recvLineBuf);
        cout<<cJSON_Print(root);

        if (root != NULL){

```

```

        cJSON* itemOrder = cJSON_GetObjectItem(root, "order");

        if (!strcmp(itemOrder->valuestring, "map")){
            cJSON* itemName = cJSON_GetObjectItem(root, "usern
ame");
            cJSON* itemAddr = cJSON_GetObjectItem(root, "addre
ss");
            cJSON* itemPort = cJSON_GetObjectItem(root, "port"
);
            cout<<"adding map\n"<<string(itemName->valuestring
)<<string(itemAddr->valuestring)<<atoi(itemPort->valuestring)<<endl;

            users[string(itemName->valuestring)]=make_pair(str
ing(itemAddr->valuestring),atoi(itemPort->valuestring));
            cout<<users[string(itemName->valuestring)].first<<
users[string(itemName->valuestring)].second<<endl;
        }else{
            cJSON* itemName = cJSON_GetObjectItem(root, "usern
ame");
            cJSON* itemContent = cJSON_GetObjectItem(root, "co
ntent");
            printf("Message from %s:\n    %s",itemName->values
tring,itemContent->valuestring);
        }

        cJSON_Delete(root);
    }
}

void process(FILE *fp, int sockfd, const struct sockaddr *pservadd
r, socklen_t servlen, char * myname){
    char buf[10];

    printf("type in the user you want to chat with:");
    scanf("%s",buf);

    printf("getting address, please wait:");

    {
        cJSON *root=cJSON_CreateObject();
        cJSON *item1=cJSON_CreateString("getmap");
        cJSON_AddItemToObject(root,"order",item1);
    }
}

```

```

        cJSON *item2=cJSON_CreateString(myname);
        cJSON_AddItemToObject(root,"username",item2);
        cJSON *item3=cJSON_CreateString(buf);
        cJSON_AddItemToObject(root,"sendto",item3);
        cJSON *hashcode=cJSON_CreateString(std::to_string(SDBMHash
(buf)).c_str());
        cJSON_AddItemToObject(root,"hashcode",hashcode);
        sendto(sockfd, cJSON_Print(root), MAXLINE, 0, pservaddr, s
ervlen);
        cJSON_Delete(root);
    }

    char sendLineBuf[MAXLINE], recvLineBuf[MAXLINE + 1];
    int nrecv;
    while(1){
        nrecv = recvfrom(sockfd, recvLineBuf, MAXLINE, 0, NULL, NU
LL); // pay attention to the last two NULLs
        recvLineBuf[nrecv] = 0;
        //std::cout<<recvLineBuf<<std::endl;
        if (!strcmp(recvLineBuf,"User do not exist!\n")){
            printf("Error:User not exist!");
            exit(0);
        }else if (!strcmp(recvLineBuf,"ACK")){
            printf("ACK::Message send successfully!\n");
            continue;
        }else if (!strcmp(recvLineBuf,"NAK")){
            printf("NAK::Message corrupted,please send again!\n");
            continue;
        }
        cJSON* root = cJSON_Parse(recvLineBuf);
        cout<<cJSON_Print(root);

        if (root != NULL){

            cJSON* itemOrder = cJSON_GetObjectItem(root, "order");

            if (!strcmp(itemOrder->valuestring,"map")){
                cJSON* itemName = cJSON_GetObjectItem(root, "usern
ame");
                cJSON* itemAddr = cJSON_GetObjectItem(root, "addre
ss");
                cJSON* itemPort = cJSON_GetObjectItem(root, "port"
);
            }
        }
    }

```

```

        cout<<"adding map\n"<<string(itemName->valuelstring)
)<<string(itemAddr->valuelstring)<<atoi(itemPort->valuelstring)<<endl;

        users[string(itemName->valuelstring)]=make_pair(string(itemAddr->valuelstring),atoi(itemPort->valuelstring));
        cout<<users[string(itemName->valuelstring)].first<<
users[string(itemName->valuelstring)].second<<endl;
        break;
    }else{
        cJSON* itemName = cJSON_GetObjectItem(root, "username");
        cJSON* itemContent = cJSON_GetObjectItem(root, "content");

        printf("Message from %s:\n    %s",itemName->valuelstring,itemContent->valuelstring);
    }

    cJSON_Delete(root);
}

if (fork()==0)
    cli(stdin, sockfd, pservaddr, servlen, myname,buf);
else
    cliread(stdin, sockfd, pservaddr, servlen);
}

int
main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in servaddr,cliaddr;
    if (argc != 3){
        printf("usage: cli <ServerIPaddress> <username>");
        exit(0);
    }

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&cliaddr, sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;

```



```

cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
cliaddr.sin_port = htons(0); /* force assignment of ephemeral
port */
bind(sockfd, (SA *) &cliaddr, sizeof(cliaddr));

login(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr), argv[2
]);

process(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr), argv
[2]);

exit(0);
}

```

功能测试（Json 显示用作调试用，蓝色框内为实际效果）：

```

}Login successful:[127.0.0.1:5
4706]BServer Receive Message f
rom ip:port 127.0.0.1 : 54706
:
{
  "order":      "login
",
  "username":    "B"
}
}Login successful:[127.0.0.1:5
4706]BServer Receive Message f
rom ip:port 127.0.0.1 : 55990
:
{
  "order":      "getma
p",
  "username":    "A",
  "sendto":      "B",
  "hashcode":    "66"
}
{
  "order":      "map",
  "username":    "B",
  "address":     "127.0
.0.1",
  "port":       "54706"
}
Server Receive Message from ip
:port 127.0.0.1 : 54706:
{
  "order":      "getma
p",
  "username":    "B",
  "sendto":      "A",
  "hashcode":    "65"
}
{
  "order":      "map",
  "username":    "A",
  "address":     "127.0
.0.1",
  "port":       "55990"
}
}

type in the user you want to cha
t with:B
getting address, please wait:ACK
::Message send successfully!
{
  "order":      "map",
  "username":    "B",
  "address":     "127.0.0
.1",
  "port":       "54706"
}
}adding map
B127.0.0.154706
127.0.0.154706
out:{
  "order":      "chat",
  "username":    "A",
  "sendto":      "B",
  "content":     "\n",
  "hashcode":    "10"
}
}
B127.0.0.154706
{
  "order":      "chat",
  "username":    "B",
  "sendto":      "A",
  "content":     "\n",
  "hashcode":    "10"
}
}Message from B:
hi, bli'm a 2017221102021
out:{
  "order":      "chat",
  "username":    "A",
  "sendto":      "B",
  "content":     "hi, bli'
m a 2017221102021\n",
  "hashcode":    "1783983
675"
}
B127.0.0.154706

"username":    "B"
}
type in the user you want to cha
t with:A
getting address, please wait:{
  "order":      "chat",
  "username":    "A",
  "sendto":      "B",
  "content":     "\n",
  "hashcode":    "10"
}
}Message from A:
ACK::Message send successfully!
{
  "order":      "map",
  "username":    "A",
  "address":     "127.0.0
.1",
  "port":       "55990"
}
}adding map
A127.0.0.155990
127.0.0.155990
out:{
  "order":      "chat",
  "username":    "B",
  "sendto":      "A",
  "content":     "\n",
  "hashcode":    "10"
}
}
A127.0.0.155990
{
  "order":      "chat",
  "username":    "A",
  "sendto":      "B",
  "content":     "hi, bli'
m a 2017221102021\n",
  "hashcode":    "1783983
675"
}
}Message from A:
hi, bli'm a 2017221102021

```

图 15 用户 A 向用户 B 发送数据

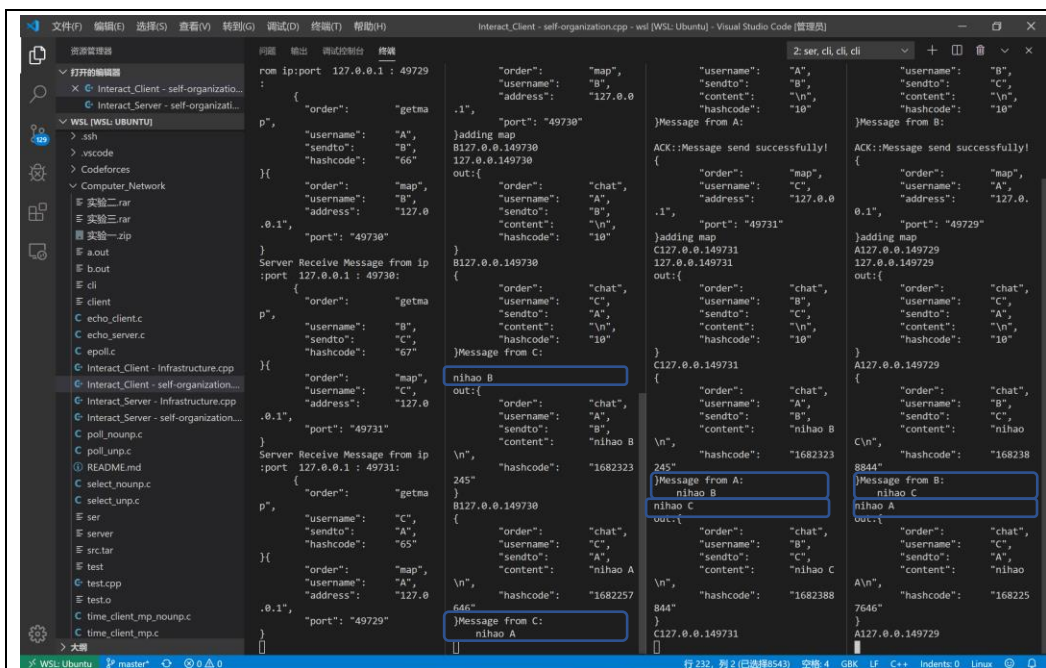


图 16 三用户数据发送

注: Json 显示用作调试用, 蓝色框内为实际效果。Json 调试内容可删去。

异常处理 1:

如果遇到目标用户不存在的情况会怎么样呢?

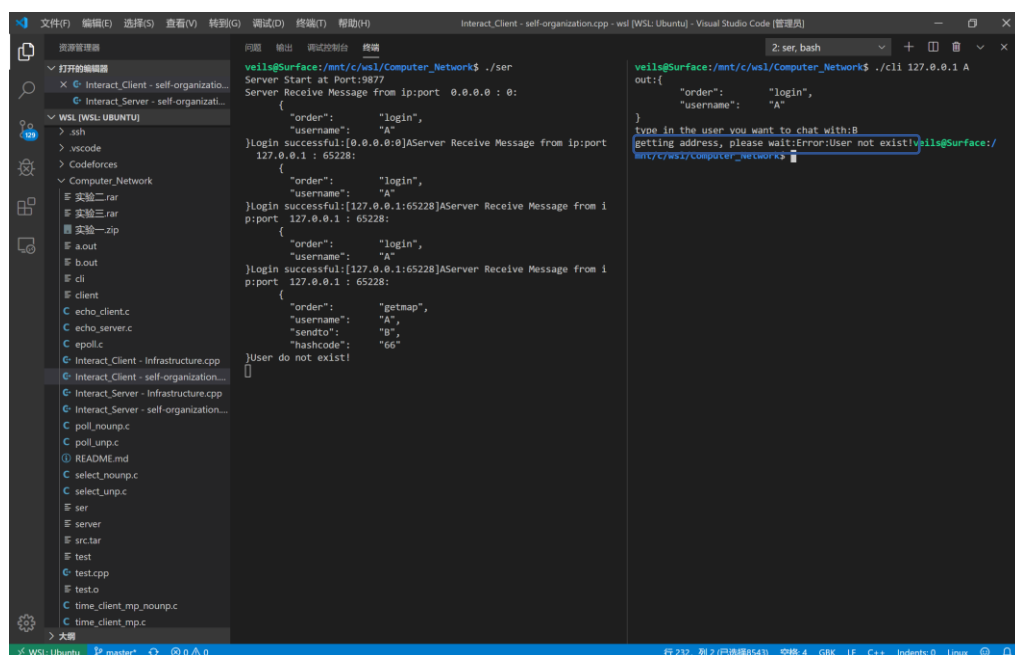


图 17 用户不存在

当我们联系没有注册的 B 时, 服务器返回了 “User do not exist!” 信息, 客户端无法连接而退出。

异常处理 2:

客户端重复登陆&&客户退出

此时与组织架构模式情况相同。

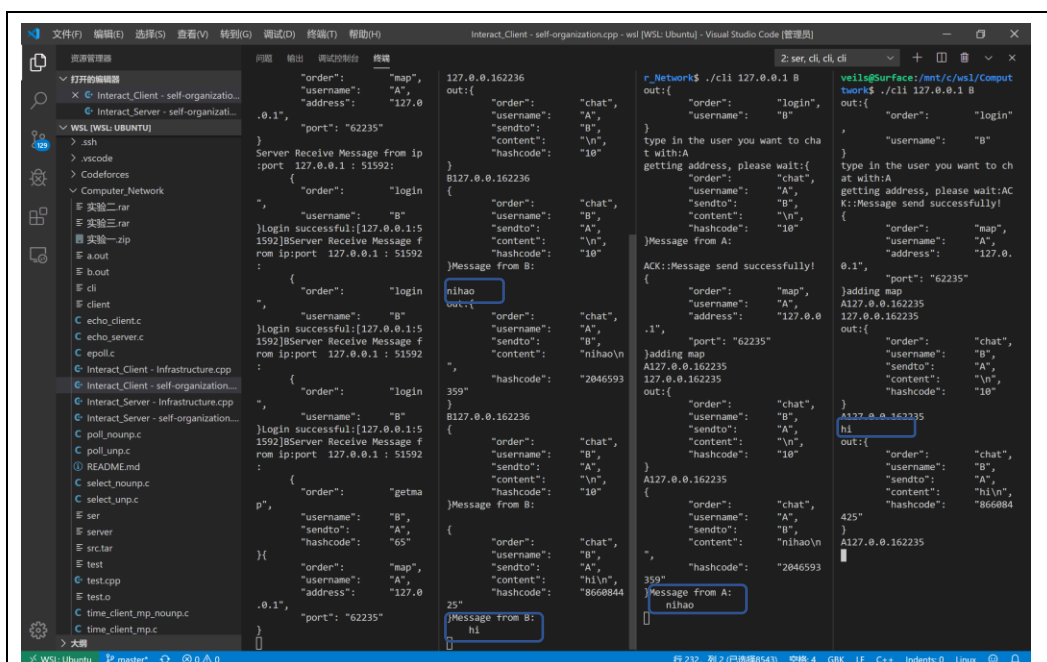


图 18 客户端重复登陆&&客户退出

如图所示，A 和 B 正常通信的情况下，B 在另一个终端登录了，此时服务器的地址映射表发生更新，之后原设备退出连接。此后 A 就会与新的终端相通信。

相应的，在客户端退出后，向服务器发送 exit 指令，服务器清除映射表中的地址项即可。

九、总结及心得体会：

通过本次实验，我掌握了基于 UDP 协议的客户机与服务器间通信原理。通过 UDP 实现客户机与服务器之间的通信及程序的编写，并掌握使用 UDP 通过服务器实现客户端之间的通信技术。

本次实验综合性较强，考虑到 UDP 不可靠传输的机制，我们需要考虑如何保证数据完整性与可用性，并设计相应的协议方法来进行可靠的实现。出于时间关系我只完成了基础架构模式的设计与实现。基于自组织模式的实现方法相比于基础架构模式并无太大差别，要注意保证对服务器用户地址信息的实时更新手段。

十、对本实验过程及方法、手段的改进建议：

暂无特别改进建议。

报告评分：

指导教师签字：