计算机网络实验报告

实验名称：编程实验1

姓名：姚翔载

学号：2011273

概述

1. 使用流式Socket设计了一个两人聊天协议，扩展到多人聊天协议
2. 完成了聊天程序的设计，并在Windows下进行实现。除了两人聊天之外，运用了多线程（pthread实现），并实现了聊天室广播功能与私聊功能的切换，并且有一定的日志记录。

协议设计

基于TCP协议

int protocol=IPPROTO_TCP

三次握手

服务端调用socket()创建套接字，进行绑定套接字、监听，并调用accept()阻塞，等待接入；客户端初始化socket，通过connect()向服务器发送一个SYN并进行了阻塞，seq=x，此时完成了第一次握手；

服务器发送SYN和ACK进行应答，置发送序号为y，确认序号为x+1。客户端收到SYN和ACK，重发一个ACK。完成二次握手

服务器接收ACK，从accept返回，建立连接，完成三次握手。

四次挥手

一端调用close()关闭，向另一端发送一个FIN=M；

另一端收到FIN后被动关闭，返回ACK=M+1，之后调用close主动关闭，再发送一个FIN=N；

主动关闭的一段再发送ack确认，四次挥手。

消息类型

交互消息为一串有特殊语法和语义的字符串，长度总共不超过1024个char的大小（可以调整）。

语法

客户端向服务器发送的数据为一串字符串，其格式为：

[模式][用户名]消息（方括号包括在字符串中）

其中，模式暂时实现了广播和私聊，它的语法为

#模式字段

暂时共有三种

#BROADCAST　固定字符串，代表广播

#PRIVCHAT=DESTNAME　其中“PRIVCHAT”以及“=”为固定的，DESTNAME为私聊目标方的用户名，为变量：若根据客户端输入决定，不超过20个char的大小（可以调整）；若是由服务端向客户端发送，则固定为“server”（不包括双引号），表示是由服务器发出的消息。

#SERVER　只出现在服务器向客户端发送的消息中，代表是由服务端发送的消息这一模式。

用户名为客户端初始化时输入的用户名。不超过20个char大小（可以调整）。理论上不允许重复，也不允许为“server”（但检测机制尚未实现，但也不会出错卡死）。

服务端向客户端发送数据的格式为：

[来源IP][时间戳][模式][用户名]消息（方括号包括在字符串中）

来源IP为触发这条消息的来源（客户端或服务端）的IP地址的字符串，其命名规范即为IP地址的常见的样式：a.b.c.d

时间戳为调用time.h库中ctime函数生成的一串字符串，具体可见代码，其格式可以参考c头文件源代码，这里大致叙述：

星期 月 日 hh:mm:ss 年

例如：

Fri Oct 21 17:26:57 2022

其余部分与客户端向服务器发送的数据中的相同。

语义

为更好地说明语法，部分语义已在语法一节中进行了叙述。

[来源IP][时间戳][模式][用户名]消息

对这个语法中不同部分的语义再次进行说明：

- 通过四组方括号隔离开了五个部分：来源IP、时间戳、模式、用户名、消息
- 来源IP只出现在服务端向客户端发送的消息中，为触发这条消息的来源的IP地址，例如客户端A通过服务器向客户端B发送一条消息，那么服务器会获取A的IP地址并打印在消息中。
- 时间戳为调用time.h库中ctime函数生成的字符串，实际上为服务器端将其发出去的时间（而不是客户端发送消息的时间等，因为这样需要更多的服务端解析操作，徒增实验难度，且无太大意义）。时间戳值出现在服务端向客户端发送的消息中。
- 模式为这条消息采取的模式，可以理解为opt code。主要包括客户端的广播和私聊、服务端的服务器模式三种。当模式=广播时，调用服务端函数将这条消息广播到所有存在的客户端中；模式=私聊时，调用服务端函数将这条消息发送到目标客户端以及来源客户端中（回显）。模式=服务器模式时，通常是服务器发送一些“系统消息”。例如某个用户退出，用户退出的消息不是它主动发出的（在这个程序设计中），而是由服务器组织语言发出的。
- 用户名为客户端初始化时输入的用户名。理论上不允许重复，也不允许为“server”。
- 消息为客户端或服务端要发的消息，可以是客户端手动输入，也可以是系统固定的一些字符串。

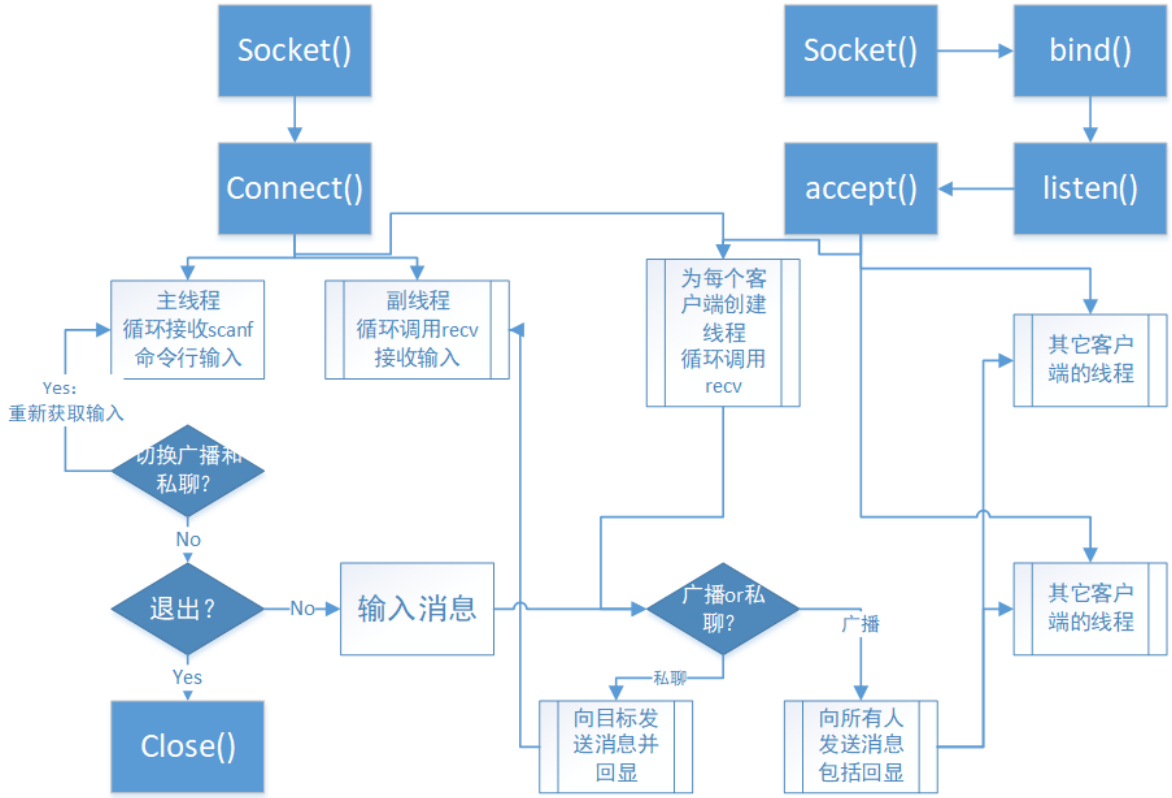
时序

所有的发送和接收频率都由while(1)设置，因此在这样正常的有限的测试操作下，不会出现丢失传输数据等问题。

系统操作的具体时间顺序为：

1. 服务端启动，调用init函数，输入IP和端口号，初始化成功
2. 客户端启动，调用init函数初始化，输入IP和端口号，连接成功则输入用户名进入下一步，否则退出客户端命令行；
3. 客户端在命令行输入，发消息。在进入到这一阶段后，可以重复步骤2，接入更多客户端。
4. 若此时服务端断开连接，客户端不会强制退出，但除了主动退出，所有操作都无作用；
5. 若此时客户端断开连接，服务端发送一个某人断开连接的提示，系统继续在步骤3运行，直到4。

程序框架



程序主要运行框图如图所示。

程序主要包含四个文件：myserver.cpp/myclient.cpp/loghdr.h/log.txt。它们分别是：服务端源代码、客户端源代码、头文件（包含一些字符串处理的函数以及全局常量变量等）、日志文件。

主要模块

服务端

先调用init()初始化，这个函数封装了DLL初始化、socket创建、绑定套接字等初始化操作。包括socket、bind等socket函数不再赘述，重点讲解自主实现的功能。

通过File方法读写日志文件：

```
    char buf[MAX_BUFFER_LEN];
    FILE *logs = fopen("log.txt", "a+");
    if(logs== NULL)
    {
    printf("open file error: \n");
    }else{
    memset(buf,0,sizeof(buf));
    //********
    ...对buf进行组装
    fputs(buf,logs);
    //记录到日志
    fputs("\n",logs);
    fclose(logs);
```

init初始化成功后，调用主线程start()。这部分主要是调用accept阻塞，等待客户端接入后为其分配序号、创建线程进行处理。

```
        struct sockaddr_in clntAddr;
        int nSize = sizeof(SOCKADDR);
        SOCKET clntSock = accept(servSock,          (SOCKADDR*)&clntAddr,
&nSize);
```

若未超过最大客户端个数，为其在数组中申请一个位置以特定地进行记录这个socket。

```
for(i=0;i<MAX_CLIENT_NUM;i++)
        {
            if(clntSockArr[i]==0)
            {
                memset(clntip[i],0,sizeof(clntip[i]));
                sprintf(clntip[i], "%u.%u.%u.%u",
IP2UCHAR(clntAddr.sin_addr.s_addr));
                //用数组记录客户端socket
                clntSockArr[i] =  clntSock;
                printf("线程号= %d\n",(int)clntSock);
                //启动一个线程为该客户服务
                pthread_t tid;
                pthread_create(&tid,0,serv_thread,&clntSock);
                break;
            }
            if (MAX_CLIENT_NUM == i-1){
            //发送给客户端说聊天室满了
            char const *msg_room_full = "对不起，聊天室已经满了!";
            send(clntSock,msg_room_full,strlen(msg_room_full),0);
            closesocket(clntSock);
            }
        }
```

可以看到上述的在clntSockAddr数组中分配位置储存客户socket的操作。同时存储客户ip地址。这些都做完后，为客户端调用pthread函数创建线程。聊天室满了则发一条信息，并关闭这个客户端线程。最后回到外层的while(1)，继续回到accept。

下面解释客户端线程的主要操作。首先进行pthread函数初始化

```
void* serv_thread(void* p)//为每个客户端创建一个线程处理
{//服务端不命令行输入消息，只管接收然后全部发出去
    int clientFd = *(int*)p;
    printf("pthread = %d\n",clientFd);
    while(1){...}
```

这个while循环中不停循环，调用recv获取信息。

```
if(recv(clientFd,buf,sizeof(buf),0)<=0)
        {//客户socket退出
        ...
        }
```

如果recv返回值小于0（INVALID_SOCKET），代表有客户端退出了，这时清理记录客户端socket、ip以及用户名的数组，找到对应这个客户端socket的index，将这些数组的对应项重置为缺省状态，并向所有现存的客户端广播这一消息，并退出当前线程。一些细节部分见注释。

```
for(i=0;i<MAX_CLIENT_NUM;i++)
            {
                if(clientFd==(int)clntSockArr[i])
                {
                    printf("SOCKET=%d 退出了\n",clientFd);
                    memset(buf,0,sizeof(buf));
                    if(logs==NULL)
                    {
                        printf("OPEN LOGFILE ERROR");
                    }else{
                        sprintf(buf, "%s退出了群聊",clntname[i]);
                        char serv_name[10];
                        char serv_mode[20];
                        strcpy(serv_name,"server");
                        strcpy(serv_mode,"#SERVER");
                        addusername(buf,serv_name);
                        addusermode(buf,serv_mode);
                        //不同于客户端发出的其他消息，带有用户名
                        //这里是客户端退出后进行处理，客户端没有发消息
                        //遵守协议，在这里加上服务器的名字和模式
                        //代表从服务器发出，模式为服务器
                        SendAll(buf);
                    }
                    clntSockArr[i]=0;
                    memset(clntip[i],0,sizeof(clntip[i]));
                    memset(clntname[i],0,sizeof(clntname[i]));
                    break;
                }
            }

            pthread_exit(0);
```

因为调用了recv函数，返回值大于0表示客户端socket是有效的，这时对消息进行解析。

```
int i=0;
        for(i=0;i<MAX_CLIENT_NUM;i++)
        {
            if(clientFd==(int)clntSockArr[i])
```

```
            {
                parseusername(clntname[i],buf);
                printf("%s\n",clntname[i]);
                //解析客户端发出的带用户名的消息，存入数组对应项
                break;
            }
        }
        parseusermode(usermode,name_for_priv_chat_dest,buf);
//解析用户发送消息的模式
        if(strcmp(usermode,mode_bdcs)==0)
        {//广播模式则调用sendall广播
            SendAll(buf);
        }else if(strcmp(usermode,mode_priv)==0)
        {//私聊模式则记录目标和来源客户端，分别用用户名和
         //在服务器的序号表示
            SendPriv(buf,name_for_priv_chat_dest,i);
        }
```

最后重回while循环。SendAll函数实现如下：

```
void SendAll(char* msg){
    int i;
    for (i = 0;i < MAX_CLIENT_NUM;i++){
        if (clntSockArr[i] != 0){
            //写入文件
            char buf[MAX_BUFFER_LEN];
            FILE *logs = fopen("log.txt", "a+");
            if(logs== NULL)
            {
            printf("open file error: \n");
            }else{
            memset(buf,0,sizeof(buf));
            sprintf(buf, "%s",msg);
            //为消息添加时间戳和用户ip
            addtimestamp(buf);
            adduserip(buf,clntip[i]);
            fputs(buf,logs);
            fputs("\n",logs);
            fclose(logs);
            }
            send(clntSockArr[i],buf,strlen(buf),0);
        }
    }
}
```

私聊的发送函数SendPriv函数实现类似，要经过目标的用户名来在用户名数组中寻找是否存在，不存在则向来源发回一条消息说明，存在则通过这个index，在clntSockAddr找到目标socket，发送消息；同时通过传入的来源socket的index回显这条消息。

```
...上方实现类似
send(clntSockArr[i],buf,strlen(buf),0);
            send(clntSockArr[originclntid],buf,strlen(buf),0);
            //发送方和接收方都显示私聊消息
            return;
        }
    }
```

```
    //到达这里表示没有找到目标用户名，向客户发送寻找失败消息
    memset(buf,0,sizeof(buf));
    sprintf(buf, "没有找到：%s。。。",destname);
    char serv_name[10];
    char serv_mode[20];
    strcpy(serv_name,"server");
    strcpy(serv_mode,"#SERVER");
    addusername(buf,serv_name);
    addusermode(buf,serv_mode);
    addtimestamp(buf);
    adduserip(buf,clntip[originclntid]);
    send(clntSockArr[originclntid],buf,strlen(buf),0);
}
```

客户端

同样先初始化，通过正确ip端口号，调用connect连接成功服务器后，提示输入用户名，向命令行终端发送一些提示信息。初始化模式为广播模式。

之后调用主线程start()。创建pthread。

```
//每个客户端创建只需一个pthread用于接收，主线程用来发送
    pthread_t id;
    void* recv_thread(void*);
    pthread_create(&id,0,recv_thread,0);
```

主线程一开始马上通过服务端广播"欢迎进入群聊"的带用户名的信息，方便服务端记录对应用户名。

```
char buf2[MAX_BUFFER_LEN] = {};
    memset(buf2,0,sizeof(buf2));
    sprintf(buf2,"欢迎%s进入群聊",name);
    addusername(buf2,name);
    addusermode(buf2,mode);
    send(sock,buf2,strlen(buf2),0);
```

while(1)循环检测scanf，直到有下一个输入：若为切换模式，则切换模式后continue，再次输入消息；若为退出则调用closesocket退出；否则为发送消息，加上模式和用户名信息后调用send，之后回到循环。

```
while(1){//主线程，主要用来发送
    //接收服务器传回的数据
    //向服务器发送数据
        char buf[MAX_BUFFER_LEN] = {};
        scanf("%s",buf);
        if(strcmp(buf,"QUIT")==0){//输入quit退出
            //不用发quit消息，server的recv会检测到
            break;
        }else if(strcmp(buf,"PRIVCHAT")==0){
            strcpy(mode,mode_priv);
            printf("请输入您想私聊的用户名：");
            scanf("%s",destname);
            printf("请输入您要发送的消息\n");
            strcat(mode,"=");
            strcat(mode,destname);
            continue;
```

```
        }else if(strcmp(buf,"BROADCAST")==0){
            strcpy(mode,mode_bdcs);
            printf("请输入您要发送的消息\n");
            continue;
        }
        char msg[MAX_BUFFER_LEN] = {};
        sprintf(msg,"%s",buf);
        //在消息前端添加用户名，消息=[@名字]消息
        addusername(msg,name);
        addusermode(msg,mode);
        send(sock,msg,strlen(msg),0);

    }
    //关闭套接字
    closesocket(sock);
}
```

接收线程较简单，一直调用recv函数，收到信息打印即可。

```
void* recv_thread(void* p){
    while(1){
        char buf[MAX_BUFFER_LEN] = {};
        if (recv(sock,buf,sizeof(buf),0) <=0){
            break;
        }
        printf("%s\n",buf);
    }
}
```

loghdr.h

头文件主要包括了include代码，全局定义了代表模式的字符串常量，并实现了一些字符串操作方法，包括为消息添加时间戳、用户i、用户名、用户模式，以及通过处理符合协议的字符串，从消息字符串中获取用户名、用户模式等信息。

添加类

以添加时间戳方法为例

```
//为消息添加时间戳
void addtimestamp(char* msg)
{
    char msg2[1024];
    memset(msg2,0,sizeof(msg2));
    strcpy(msg2,msg);
    memset(msg,0,sizeof(msg));
    time(&nowtime1);
    //ctime最后一个为换行符，将它替换，美观
    char* timestamp = ctime(&nowtime1);
    char *tmp = NULL;
    if ((tmp = strstr(timestamp, "\n")))
    {
        *tmp = '\0';
    }
    sprintf(msg,"[%s]%s",timestamp,msg2);
}
```

主要通过strcpy函数、sprint函数来处理得到**[时间戳]源字符串**。注意到调用ctime获取的字符串最后是一个换行符，将其换为结束符。其它几个添加消息字符串组成部分的函数实现类似。

解析类

以解析用户名为例，通过strchr、指针操作获取想要的字符串的相对位置，再进行复制等操作。一些详细说明见代码注释。

```
void parseusername(char* nameaddr,char* buf)
{
    //协议中，客户端和服务端发送出的消息都是以方框包住的一些信息开头
    //只有用户名前有@
    //strchr能得到字符串中某个字符第一次出现的位置
    //解析出@到@下一个]之前的字符串即为用户名
    char tmp[1024];
    memset(tmp,0,sizeof(tmp));
    char rst[22];
    memset(rst,0,sizeof(rst));
    strcpy(tmp,buf);
    char* dest_at=strchr(tmp,'@');
    char* dest_rightc=strchr(dest_at,']');
    int begin = dest_at-tmp;
    int end = dest_rightc-dest_at+begin;
    int cnt=0;
    for(int i=begin+1;i<end;i++)
    {
        rst[i-begin-1] = tmp[i];
        cnt++;
    }
    rst[cnt]='\0';
    memset(nameaddr,0,sizeof(nameaddr));
    sprintf(nameaddr,"%s",rst);
}
```