Socket编程实验：聊天室

计算机网络实验一

姓名：鲁含章
学号：1811398
日期：2020年10月30日

目录

一、实验要求

1. 给出你聊天协议的完整说明。
2. 利用C或C++语言，使用基本的Socket 函数完成程序。不允许使用CSocket等封装后的类编写程序。
3. 使用流式Socket完成程序。
4. 程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。
5. 完成的程序至少应能实现两个用户之间的英文和中文聊天。
6. 编写的程序应结构清晰，具有较好的可读性。
7. 提交源码和实验报告。

二、实验环境

- 操作系统：Windows10（1909）
- 编译：x86
- IDE：Visual Studio 2019
- 语言：`c++`
- 主要库：`<WinSock2.h>`，`<iostream>`，`<string.h>`，`<string>`，`<vector>`，`<set>`，`<map>`

三、聊天协议说明

（一） 运行

1. 首先启动服务器端Sever.exe，显示`服务器初始化成功`继续；
2. 启动客户端Client.exe，输入自定义用户名，显示`连接成功`继续，可以启动多个不同名客户；
3. 客户端输入`{sendto}:{message}`格式信息，例如`user2:hello`；
4. 接收方如果已登录，则服务器会转发该信息，显示在接收方终端上，同时向发送方发送`200 已发送`信息；
5. 退出：客户端输入quit，即可断开连接，退出程序

（二）用户端

1 基本信息

IP：本机地址

端口：12000

2 输入命令格式

用户命令格式如下：

命令	含义
NAME:{username}	打开用户端程序会要求输入用户名{username}（任意字符） 客户端程序会将{username}输入拼接“NMAE:”发送给服务器 该{username}作为用户的标识，要求唯一性，对重复的用户名服务器会拒绝
{sendto}: {message}	用户进行交互的主要命令 {sendto}指代接收方用户的用户名 {message}指代消息信息
quit	用户输入的前四个字符为quit时退出程序

（三）服务器端

1 基本信息

IP：本机地址

端口：12000

2 输出说明

服务器端开始运行后无需管理，方便测试会将如下信息输出

- 接收到的连接请求
- 收到的消息
- 转发动作的消息
- 用户退出的信息

（四）返回信息

编号	错误信息
101	ERROR:加载socket库失败
401	ERROR:用户名重复
402	ERROR:未收到用户名
403	ERROR:用户未上线
404	ERROR:连接断开
200	OK:连接正常，成功发送

（五）接口

本实验中使用基于TCP的**流式套接字**（**SOCK_STREAM**）接口传输数据。

> 流式套接字用于提供面向连接、可靠的数据传输服务。该服务将保证数据能够实现无差错、无重复送，并按顺序接收[1]。

四、主要代码说明

（一）服务器端Sever.cpp

1 重要全局变量

- Users：set类型，每当新用户连接服务器，存入用户名。
- MES：map类型，key为用户名，value为该用户**收到的有序消息组**，对每个用户线程，有对应的指针指向该消息组未发送消息的开头。

```
SOCKADDR_IN addrSrv, addrClient;//端口号和地址
int len = sizeof(SOCKADDR_IN);
unsigned long dwThreadId=100; //线程号
set<string> Users; //在线的用户列表
map<string, vector<string>> MES; //消息
```

2 重要函数

handlerRequest():

在主函数中为每一个连接的用户会新建一个**Socket接口**与一个**该函数控制的线程**，实现以下功能：

- 接收用户端发来的第一个消息：用户名，并为该用户名创建消息组放于全局变量`MES`中；
- 通过`iter`确定有序消息组`MES[userName]`中未转发给该用户的消息，遍历转发；
- 创建接收该用户消息的线程handlerRev，与发送线程同步运行

```
DWORD WINAPI handlerRequest(LPVOID lparam) {
    SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
    string userName="";
    char recvbuf[BUF_SIZE]="";
    char sendbuf[BUF_SIZE]="";

    ...//接收用户名操作，省略

    //初始化
    Users.insert(userName);
    vector<string> wait_mes;
    MES.insert(make_pair(userName, wait_mes));
    int iter = 0;

    cout << "用户" << userName << "建立连接" << endl;

    memset(recvbuf, 0, sizeof(recvbuf) / sizeof(char)); //及时清空缓存
    recvbuf[0] = '\0';

    para P = { ClientSocket ,userName };
    dwThreadId += 1;
    HANDLE hThread = CreateThread(NULL, NULL, handlerRev, LPVOID(&P), 0,
&dwThreadId);
    CloseHandle(hThread);       //创建接收的线程

    //发送消息
    while (1) {
        //发送未发送消息
        int sizeMes = MES[userName].size();
```

```
        while (iter < sizeMes) {
            string t = MES[userName][iter];
            t += '\n';
            cout<< "\n发送消息给" << userName << ":\"" << t <<"\""<< endl;
            str2char(sendbuf, t);
            send(ClientSocket, sendbuf, strlen(sendbuf), 0);
            memset(sendbuf, 0, sizeof(sendbuf) / sizeof(char)); //及时清空缓存
            sendbuf[0] = '\0';
            ++iter;
        }
    }
    closesocket(ClientSocket);
    return 0;
}
```

handlerRev()

由handlerRequest()创建的线程，用于接收消息模块，实现如下功能：

- 接收并解析用户传来的信息
- 将有效信息存入对应的有序消息组
- 向用户返回错误信息或发送成功的信息

```
DWORD WINAPI handlerRev(LPVOID lparam) {        //接收模块，独立线程
    char recvbuf[BUF_SIZE] = "";
    char sendbuf[BUF_SIZE] = "";
    para* P= (para*)(LPVOID)lparam;
    SOCKET ClientSocket = P->sock;
    string userName = P->userName;
    while (1) {
        //接收消息
        if (recv(ClientSocket, recvbuf, 50, 0) == 0) {
            cout << ClientSocket << "404 ERROR:连接断开" << endl;
            break;
        }

        if (recvbuf[0] != '\0') {
            cout << "\n收到来自" << userName << "的消息:\"" << recvbuf << "\"" <<
endl;

            if (strcmp_n(recvbuf, "quit", 4)) {  //退出
                cout << "用户" << userName << "退出" << endl;
                cout << "关闭Socket" << ClientSocket << endl;
                break;
            }
            string sendto;
            if ((sendto = add_to_MES(recvbuf, userName)) != "") {
                //无该用户
                string t = "403 用户" + sendto + "未上线";
                str2char(sendbuf, t);
                send(ClientSocket, sendbuf, strlen(sendbuf), 0);
                memset(sendbuf, 0, sizeof(sendbuf) / sizeof(char)); //及时清空缓存
                sendbuf[0] = '\0';
            }
            else {
                strcpy_s(sendbuf, "200 已发送");
                send(ClientSocket, sendbuf, strlen(sendbuf), 0);
```

```
                memset(sendbuf, 0, sizeof(sendbuf) / sizeof(char)); //及时清空缓存
                sendbuf[0] = '\0';
            }
        }
        memset(recvbuf, 0, sizeof(recvbuf) / sizeof(char)); //及时清空缓存
        recvbuf[0] = '\0';

    }
    return 0;
}
```

（二）客户端Client.cpp

1 主要函数

main()

主函数，发送模块位于主线程中，将接收模块创建另一个线程。

```
...//连接部分省略

while (1)
{
    if (connect(sockCli, (SOCKADDR*)&addrSer, sizeof(addrSer)) != SOCKET_ERROR)
    {//如果连接成功
        string nameMes = "NAME:" + userName;    //初始化发送姓名
        str2char(sendBuf,nameMes);
        if (send(sockCli, sendBuf, strlen(sendBuf), 0) < 0) {
            cout << "error: 连接断开，发送失败" << endl;
            break;
        }
        HANDLE hThread = CreateThread(NULL, NULL, handlerRecver,
LPVOID(sockCli), 0, &dwThreadId);
        CloseHandle(hThread);       //接收消息线程

        while (1) {
            //发送模块置于主线程
            char sendM[BUF_SIZE] = "";
            cin >> sendM;

            strcpy_s(sendBuf, sendM);
            if (send(sockCli, sendBuf, strlen(sendBuf), 0) < 0) {
                cout << "404 ERROR:连接断开" << endl;
                break;
            }

            if (strcmp_n(sendM, "quit", 4)|QUIT) {  //退出
                cout << "关闭Socket" << endl;
                QUIT = 1;
                closesocket(sockCli);
                WSACleanup();//卸载socket库
                cout << "退出程序中" << endl;
                return 0;
            }
        }
    }
}
```

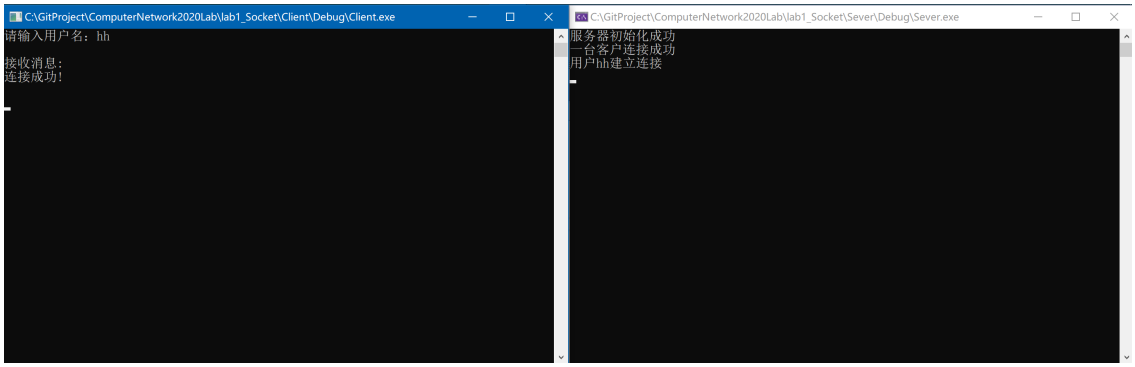
handlerRecver()

接收消息的线程，由main()函数创建。

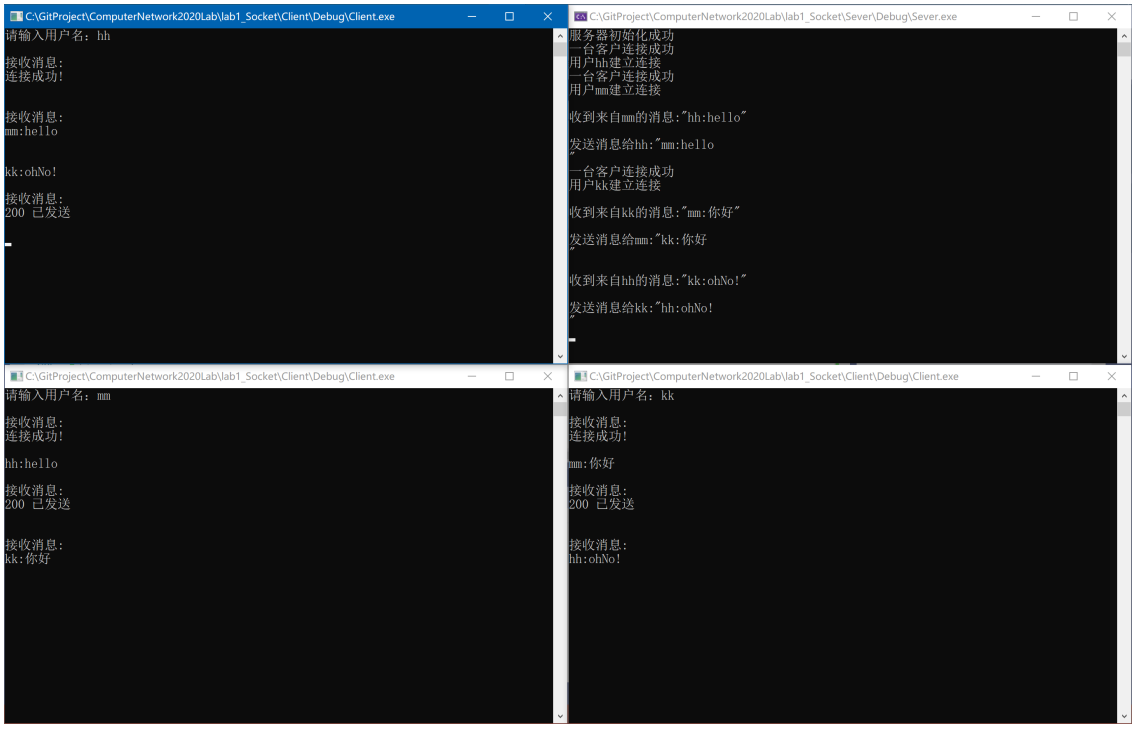
```
DWORD WINAPI handlerRecver(LPVOID lparam) {      //接收模块，独立线程
    while (1) {
        if (QUIT)return 0;

        if (recv(sockCli, recvBuf, sizeof(recvBuf), 0) == 0) {
            cout<< "404 ERROR:连接断开" << endl;
            return 0;
        }

        if(recvBuf[0]!='\0')cout <<"\n接收消息:\n"<< recvBuf << endl<<endl;
        memset(recvBuf, 0, BUF_SIZE); //及时清空缓存
        recvBuf[0] = '\0';
    }
    return 0;
}
```

五、运行截图

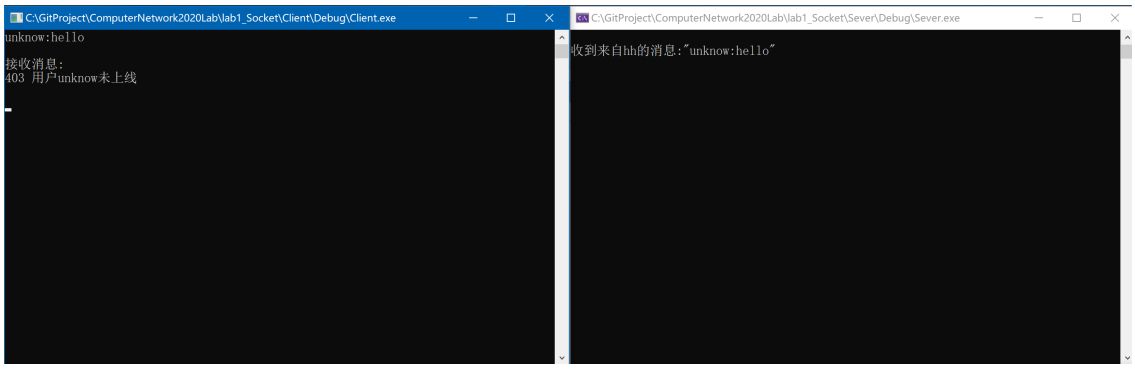
1. 输入用户名，建立连接



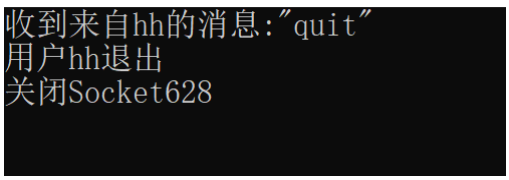
2. 多个用户间相互收发消息



3. 向不存在的用户发送消息



4. 用户退出，服务器端输出：



参考文献

[1] 百度百科：套接字