# 期末大作业——多人聊天室

小组成员：14051111陈存粮 14051112陈孟春

一、设计方案

这个作业比之前的主要增加了传文件、窗口抖动、获取在线列表功能，服务端的模型使用了完成端口。

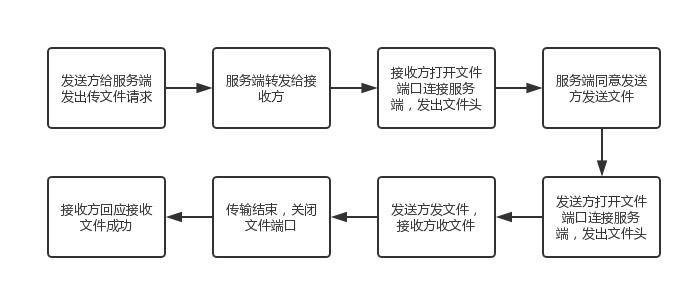
传文件功能使用了一个传文件的专用端口，和别的功能发的包分开发送，在发文件的时候还可以同时使用聊天等功能。

发送各种包的端口还是原来的8888，新增的传文件端口为8889。

客户端使用QT设计，相对MFC拥有更好的可移植性。

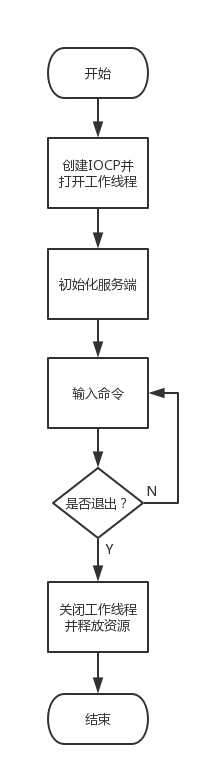
服务端的完成端口部分参考了网上的代码，把绑定的监听端口变成了两个，处理函数也做了修改。

传文件过程介绍：

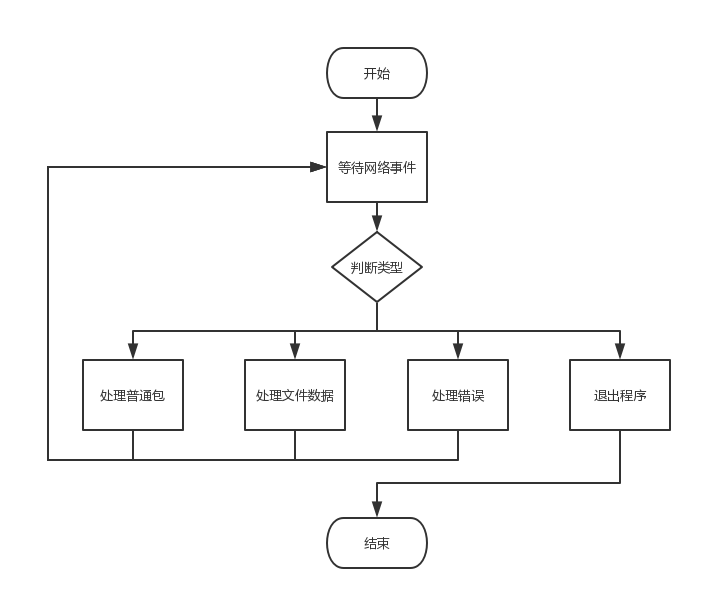


服务端程序流程图：

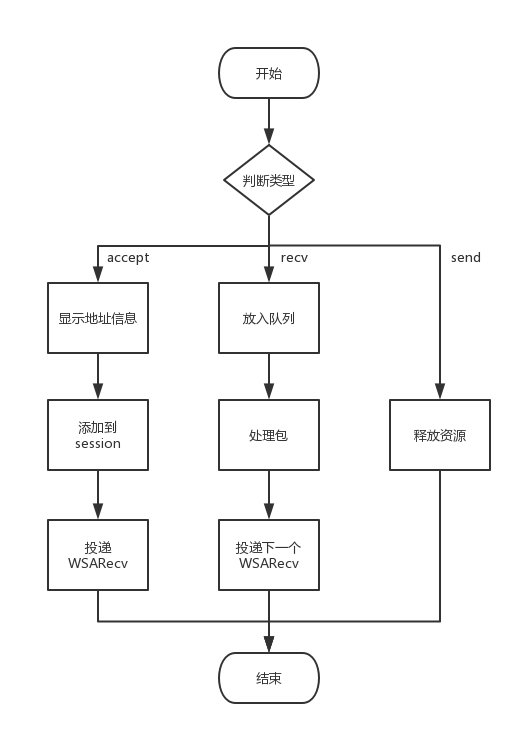
1. 主函数



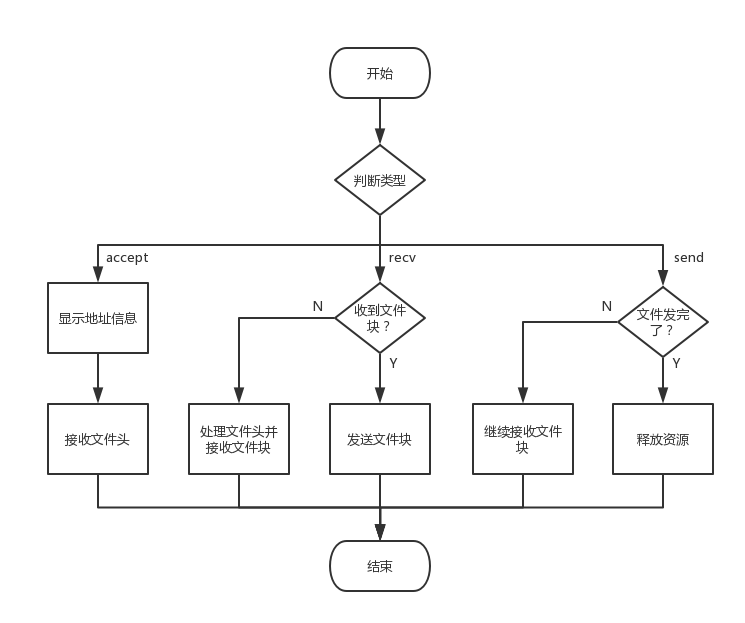
（2）完成端口工作线程



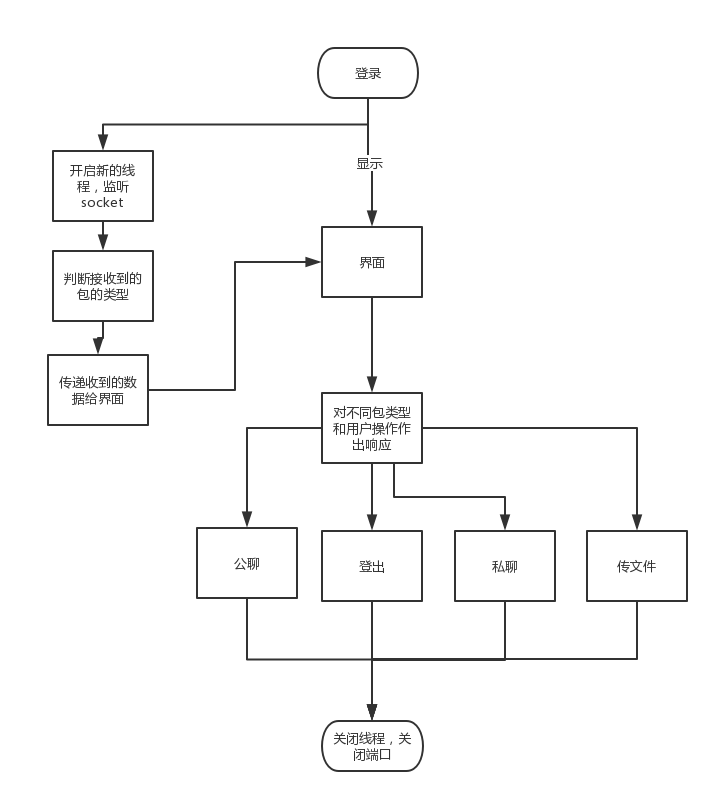
1. 处理普通包



（3）处理文件数据



客户端设计大体流程：



二、功能介绍

聊天室功能包括登录、登出、公聊、私聊、获取在线列表、窗口抖动和在线传文件。

其中登录、登出、公聊、私聊功能的实现以及协议设计较之前没有太大变化。下面主要介绍新功能的设计。

1.获取在线列表

（1）协议设计

//获取在线用户名

typedef struct RequestOnlineUserPkt

{

Header header;

int ReqNumber;

}RequestOnlineUserPkt;

//服务端返回在线用户名

typedef struct ResponseOnlineUserPkt

{

Header header;

int OnlineNum; //在线用户数

int flag; //保留（如果在线用户太多再用）

char users[1024]; //在线用户名字（用'\n'分割）

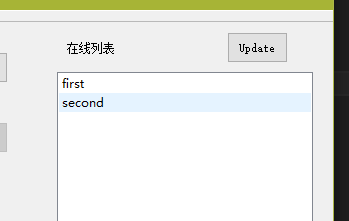
}ResponsetOnlineUserPkt;

（2）具体流程

--》客户端点击Update按钮，向服务器发送RequestOnlineUserPkt包

--》服务端接收请求后，回复一个ResponsetOnlineUserPkt包，

--》客户端接到回复包之后，进行解析将在线的用户显示在在线列表，详情如下图



（3）服务端获取在线列表处理函数

函数介绍：遍历在线用户列表，把用户名复制出来拼接起来，中间用\n间隔，最后发送。

int processUpdateOnlineUserPkt(const char \*pkt, Session \*session)

{

ResponseOnlineUserPkt rouPkt;

int offset = 0;

int len = 0;

rouPkt.flag = 0;

rouPkt.OnlineNum = 0;

memset(rouPkt.users, 0, sizeof(rouPkt.users));

//遍历在线用户列表，把名字拷贝出来

for (int i = 0; i < g\_online\_session\_num;i++)

{

if (STATE\_LOGIN == g\_online\_sessions[i].state)

{

strcpy(rouPkt.users + offset, g\_online\_sessions[i].userName);

offset += strlen(g\_online\_sessions[i].userName) + 1;

rouPkt.users[offset - 1] = '\n';

rouPkt.OnlineNum++;

}

}

len = RE\_ONLINEUSER\_HEAD + offset;

rouPkt.header.type = htonl(UPDATE\_ONLINEUSER\_PKT);

rouPkt.header.len = htonl(len);

rouPkt.flag = htonl(rouPkt.flag);

rouPkt.OnlineNum = htonl(rouPkt.OnlineNum);

myWSASend(session->sock, (const char \*)&rouPkt, len, 0);

return 0;

}

2.窗口抖动包

（1）协议设计

typedef struct WindowShakingPkt

{

Header header;

char from[20];

char to[20];

}WindowShakingPkt;

（2）具体流程

--》客户端点击抖动按钮，向服务器发送WindowShakingPkt，并自己先进行抖动操作

--》服务端收到后根据包内to字段，将WindowShakingPkt转发给to字段指向的用户

--》某个用户接收到WindowShakingPkt包后，如果已经打开和from字段指向用户私聊窗口的话，抖动该窗口，否则新建一个私聊窗口进行抖动操作

详情如下图（一个gif文件）



（3）服务端窗口抖动处理函数

函数介绍：过程和私聊包类似，就是把发送方的包转发给接收方。

int processWindowShakingPkt(const char \*pkt, Session \*session)

{

WindowShakingPkt \*windowShakingPkt = (WindowShakingPkt \*)pkt;

SOCKET sockTo;

if (FIND\_SESSION\_SUCCESS == findSocket(windowShakingPkt->to, &sockTo))

{

windowShakingPkt->header.len = htonl(windowShakingPkt->header.len);

windowShakingPkt->header.type = htonl(windowShakingPkt->header.type);

myWSASend(sockTo, pkt, sizeof(WindowShakingPkt), 0);

}

return 0;

}

3.文件传输功能

（1）协议设计

//这里是传文件包命令字段的定义

#define FILE\_SEND 101 //请求发送文件

#define FILE\_RECV 102 //请求接收文件

#define FILE\_RECVSUCCESS 200 //接收文件成功

#define FILE\_EXIST 304 //文件已经存在

#define FILE\_NOTFOUND 404 //文件已经过期

#define FILE\_SERVERERROR 500 //服务端不想收

//在命令端口的包格式

typedef struct TransmitFlieCmdPkt

{

Header header;

int cmd; //命令字

int fileSize; //文件大小

int recvSize; //保留做断点续传用

char from[20];

char to[20];

char md5[32]; //32位MD5校验码

char filename[256];

}TransmitFlieCmdPkt;

//在数据端口的包格式(在发数据之前发)

//这里的cmd只能用FILE\_SEND和FILE\_RECV

typedef struct FileHeader

{

int cmd; //命令字（收or发）

int fileSize; //文件大小

int recvSize; //保留做断点续传用

char from[20];

char to[20];

char md5[32]; //32位MD5校验码

}FileHeader;

（2）具体流程

--》发送方客户端点击发送文件按钮，弹出文件对话框，用户选择要发送的文件，向服务器发送一个TransmitFlieCmdPkt包，并生成一个传输文件的对话框

--》服务端收到TransmitFlieCmdPkt包之后，将包中的文件信息添加到服务端维护的文件传输表中，然后发一个TransmitFlieCmdPkt包发给接收方

--》接收方收到TransmitFlieCmdPkt包，判断如果与发送方的私聊窗口是否打开，如果未打开，新建一个私聊窗，并新建一个文件传输窗，询问用户是否接收该文件，如果选择接收则点击Recive按钮，将新建一个数据端口连接到服务器上的数据端口，并通过该端口向服务器发送一个同意接收文件的FileHeader包，

--》服务器接收到接收方的FileHeader包之后，服务器经过简单比对包内文件信息，向发送方发一个TransmitFlieCmdPkt包

--》发送方接收到回复的TransmitFlieCmdPkt包之后，connect服务器按钮可用，用户点击该按钮后，新建一个数据端口，连接服务器，并通过该端口向服务器发送一个FileHeader包，然后就开启一个线程通过数据端口开始发送文件了，直至接收完毕，关闭线程，关闭数据端口，关闭文件传输窗口

--》服务端接收到后就将包转发给接收方，重复该操作，直至文件发送完毕

--》接收方接受完完整文件后，向服务器发送一个成功的TransmitFlieCmdPkt包，关闭线程，关闭数据端口，关闭文件传输窗口

--》服务器收到转发给发送方

--》发送方接收到后，弹窗提示文件发送成功

（3）完成端口中传文件时使用的结构体

//收发文件用的结构体

typedef struct

{

OVERLAPPED overlapped;

OPERATION\_TYPE operationType;

WSABUF databuff;

char buffer[FILEBUFFSIZE];

DWORD bufferLen;

DWORD flags;

SOCKET socket; //在建立连接时临时存储

SOCKET fsockfrom; //发送端的socket

SOCKET fsockto; //接收端的socket

int fileSize; //文件大小

int recvSize; //已接收大小

int state; //传文件的状态

char md5[36]; //32位MD5校验码

}PER\_FILE\_DATA, \*LPPER\_FILE\_DATA;

（4）服务端处理传文件命令包的函数

函数介绍：收到包时，先通过addFileTable函数把文件信息添加到表中，如果添加成功了就通知接收端。

int processTransmitFileCmdPkt(const char \*pkt, Session \*session)

{

TransmitFlieCmdPkt \*transmitFileCmdPkt = (TransmitFlieCmdPkt \*)pkt;

transmitFileCmdPkt->cmd = ntohl(transmitFileCmdPkt->cmd);

transmitFileCmdPkt->fileSize = ntohl(transmitFileCmdPkt->fileSize);

transmitFileCmdPkt->recvSize = ntohl(transmitFileCmdPkt->recvSize);

int ret = 0;

int cmd = transmitFileCmdPkt->cmd;

if (FILE\_SEND == cmd)//客户端想发文件

{

ret = addFileTable(\*transmitFileCmdPkt);//添加到待传输文件列表

//创建响应包

TransmitFlieCmdPkt replyPkt = \*transmitFileCmdPkt;

int pktlen = replyPkt.header.len+4;

replyPkt.fileSize = htonl(replyPkt.fileSize);

replyPkt.recvSize = htonl(replyPkt.recvSize);

replyPkt.header.len = htonl(replyPkt.header.len);

replyPkt.header.type = htonl(replyPkt.header.type);

if (-FILE\_SERVERERROR == ret)//文件列表满了

{

replyPkt.cmd = htonl(FILE\_SERVERERROR);

myWSASend(session->sock, (const char \*)&replyPkt, pktlen, 0);

}

else//通知接收端接收文件

{

replyPkt.cmd = htonl(FILE\_SEND);

SOCKET sockto;

findSocket(replyPkt.to, &sockto);

myWSASend(sockto, (const char \*)&replyPkt, pktlen, 0);

}

}

else if (FILE\_RECVSUCCESS == cmd)//文件成功接收提示

{

//创建响应包

TransmitFlieCmdPkt replyPkt = \*transmitFileCmdPkt;

int pktlen = replyPkt.header.len + 4;

replyPkt.fileSize = htonl(replyPkt.fileSize);

replyPkt.recvSize = htonl(replyPkt.recvSize);

replyPkt.header.len = htonl(replyPkt.header.len);

replyPkt.header.type = htonl(replyPkt.header.type);

SOCKET sockfrom;

findSocket(replyPkt.from, &sockfrom);

//给发送方回应

replyPkt.cmd = htonl(FILE\_RECVSUCCESS);

myWSASend(sockfrom, (const char \*)&replyPkt, pktlen, 0);

}

return 0;

}

（5）服务端文件端口处理文件头的函数

函数介绍：当收到接收端的文件头时，先保存下接收端的socket，状态从STATE\_START转换到STATE\_WAITTRANS；当收到发送端的文件头时，状态从STATE\_WAITTRANS转换到STATE\_TRANS，传输文件就可以开始了。

int processFileHeader(SOCKET sock, SOCKET \*sockto, FileHeader fileHeader)

{

\*sockto = INVALID\_SOCKET;

if (WAIT\_OBJECT\_0 == WaitForSingleObject(g\_fileInformation\_mutex, INFINITE))

{

int i;

if (fileHeader.cmd == FILE\_RECV)//接收端连接我了

{

for (i = 0; i < g\_fileInformation\_num; i++)

{

if (0 == memcmp(g\_fileInformation[i].md5, fileHeader.md5, sizeof(fileHeader.md5))

&& 0 == strcmp(g\_fileInformation[i].to, fileHeader.to))

{

g\_fileInformation[i].state = STATE\_WAITTRANS;

g\_fileInformation[i].fsockto = sock;

break;

}

}

//创建响应包

TransmitFlieCmdPkt replyPkt;

replyPkt.cmd = htonl(FILE\_RECV);

strcpy(replyPkt.filename, g\_fileInformation[i].filename);

replyPkt.fileSize = htonl(g\_fileInformation[i].fileSize);

strcpy(replyPkt.from, g\_fileInformation[i].from);

strcpy(replyPkt.to, g\_fileInformation[i].to);

memcpy(replyPkt.md5, g\_fileInformation[i].md5, sizeof(replyPkt.md5));

replyPkt.header.type = htonl(TRANSMITFILECMD\_PKT);

int len = TRANSMITFILE\_HEAD + strlen(replyPkt.filename) + 1;

replyPkt.header.len = htonl(len - 4);

SOCKET sockfrom;

findSocket(g\_fileInformation[i].from, &sockfrom);

myWSASend(sockfrom, (const char\*)&replyPkt, len, 0);

}

else//发送端连接我了

{

for (int i = 0; i < g\_fileInformation\_num; i++)

{

if (0 == memcmp(g\_fileInformation[i].md5, fileHeader.md5, sizeof(fileHeader.md5))

&& 0 == strcmp(g\_fileInformation[i].from, fileHeader.from))

{

g\_fileInformation[i].state = STATE\_TRANS;

g\_fileInformation[i].fsockfrom = sock;

\*sockto = g\_fileInformation[i].fsockto;

break;

}

}

}

ReleaseMutex(g\_fileInformation\_mutex);

}

return 0;

}

4.服务端简化使用的WSASend函数

函数介绍：因为WSASend函数在使用的时候初始化的代码比较多，所以做成了一个函数，把原来send函数的地方替换成这个就可以使用了。

int myWSASend(SOCKET s, const char \*buf, int len, DWORD flags)

{

LPPER\_IO\_DATA lpPerIOData = (LPPER\_IO\_DATA)GlobalAlloc(GPTR, sizeof(PER\_IO\_DATA));

ZeroMemory(&(lpPerIOData->overlapped), sizeof(OVERLAPPED));

ZeroMemory(&(lpPerIOData->buffer), sizeof(lpPerIOData->buffer));

memcpy(lpPerIOData->buffer, buf, len);

lpPerIOData->bufferLen = len;

lpPerIOData->databuff.len = len;

lpPerIOData->databuff.buf = lpPerIOData->buffer;

lpPerIOData->flags = flags;

lpPerIOData->operationType = OPERATION\_SEND;

lpPerIOData->socket = s;

int ret = WSASend(s, &(lpPerIOData->databuff), 1,

&(lpPerIOData->bufferLen), lpPerIOData->flags, &(lpPerIOData->overlapped), NULL);

if (-1 == ret&& GetLastError() != ERROR\_IO\_PENDING)

{

MYERROR("WSASend");

removeSession(s);

GlobalFree(lpPerIOData);

lpPerIOData = NULL;

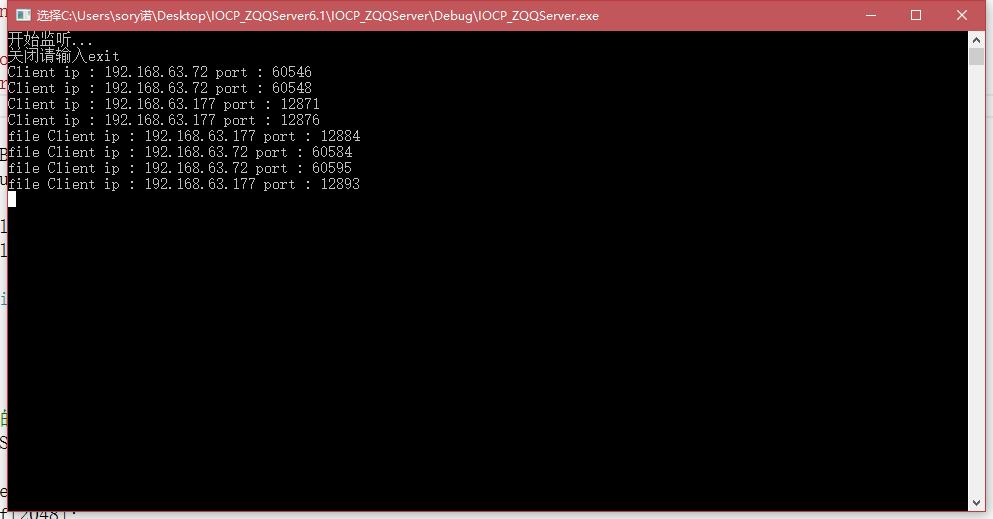
}

return 0;

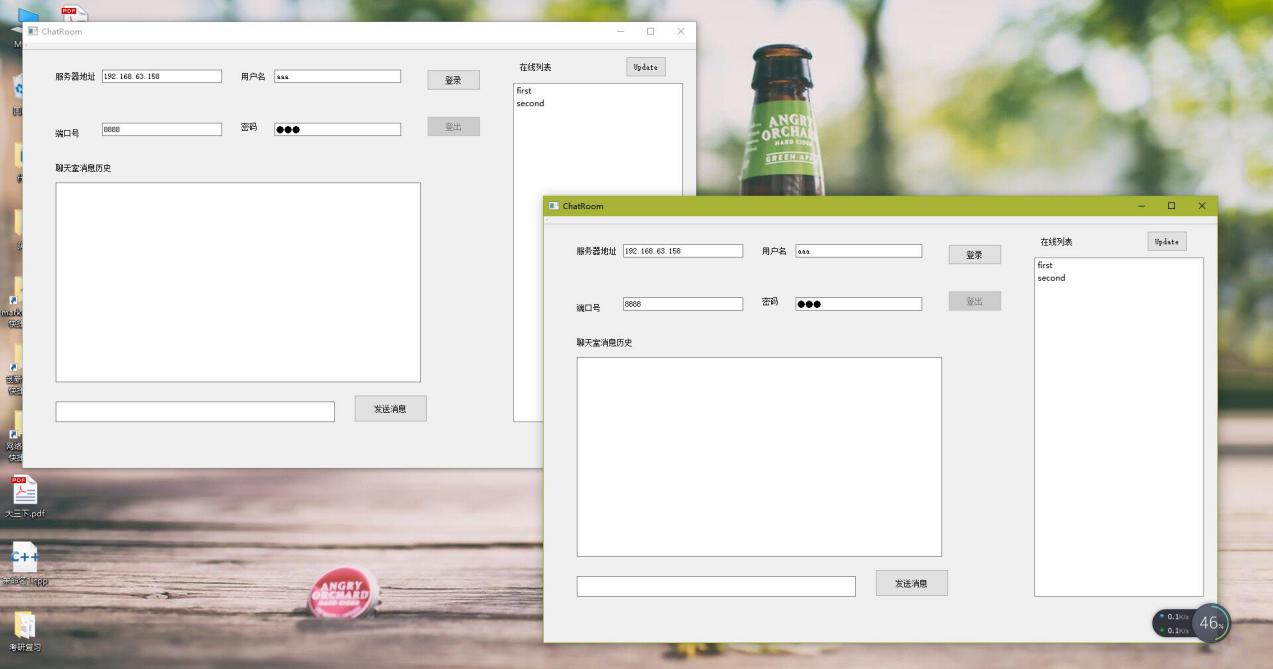
}

三、程序运行截图

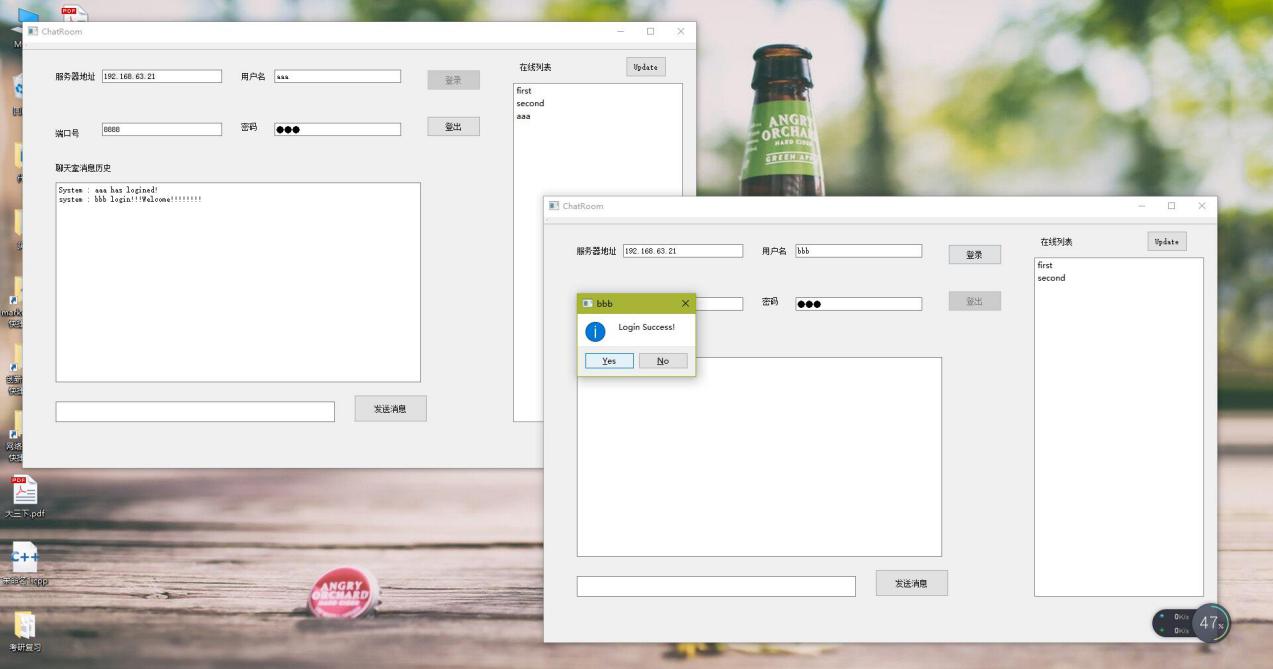
1.服务端



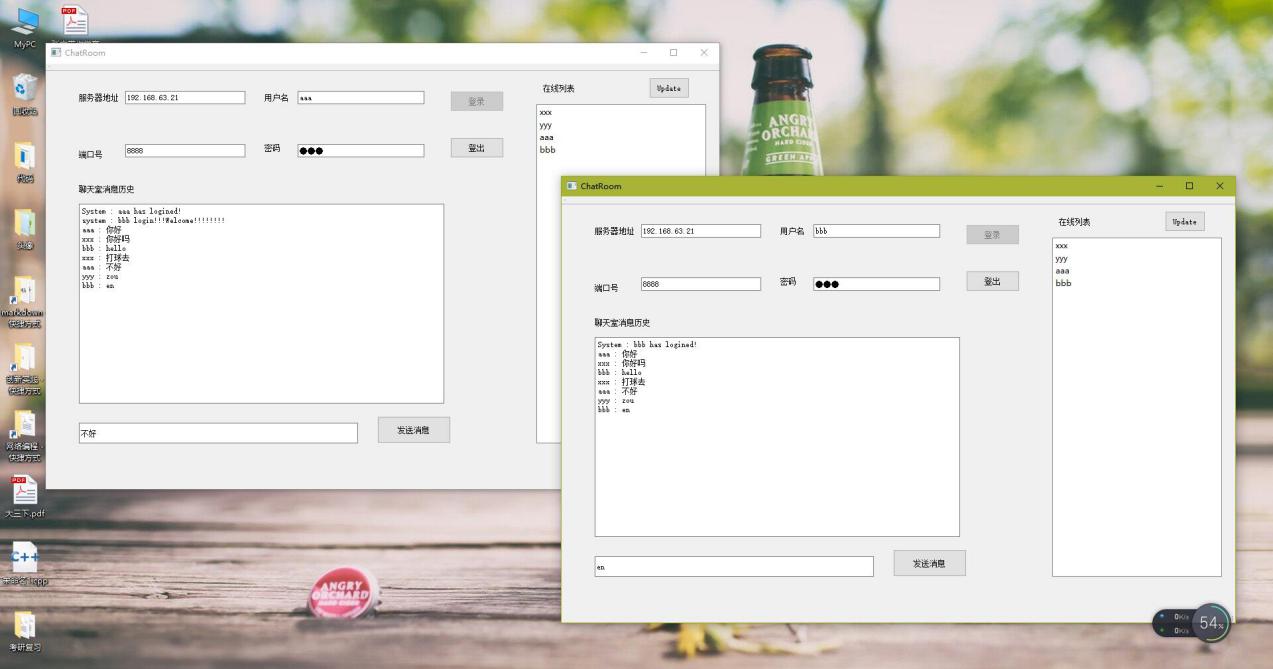
1. 客户端
2. 打开



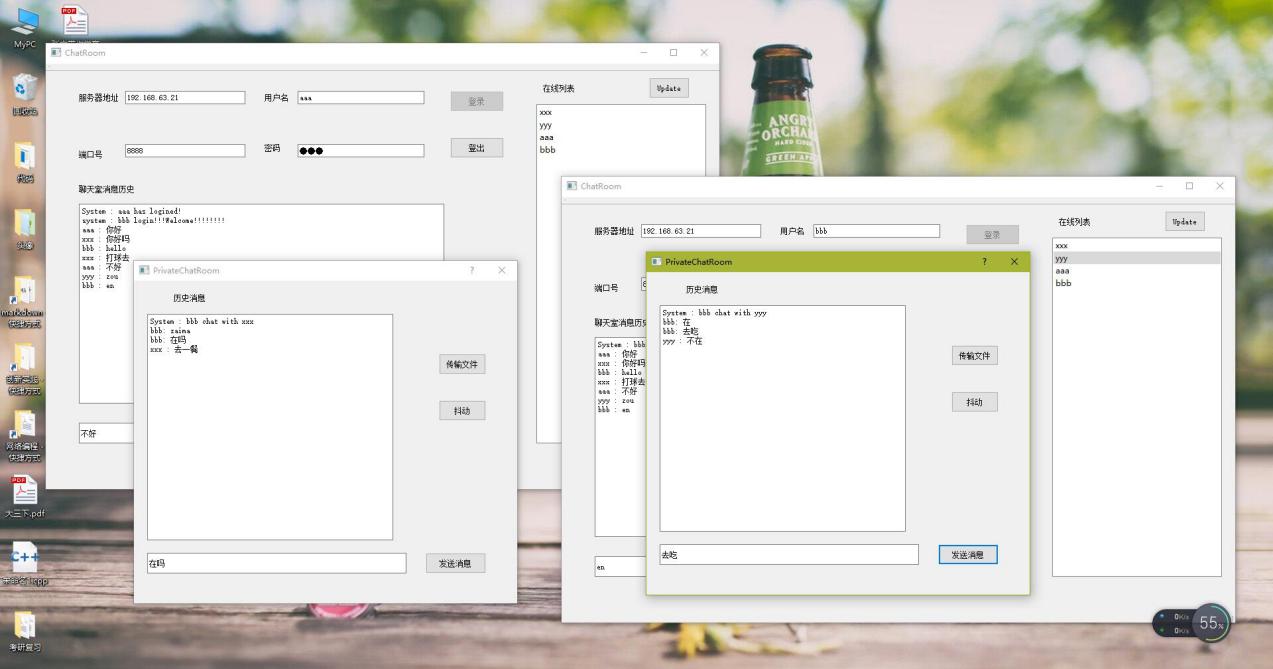
（2）登录



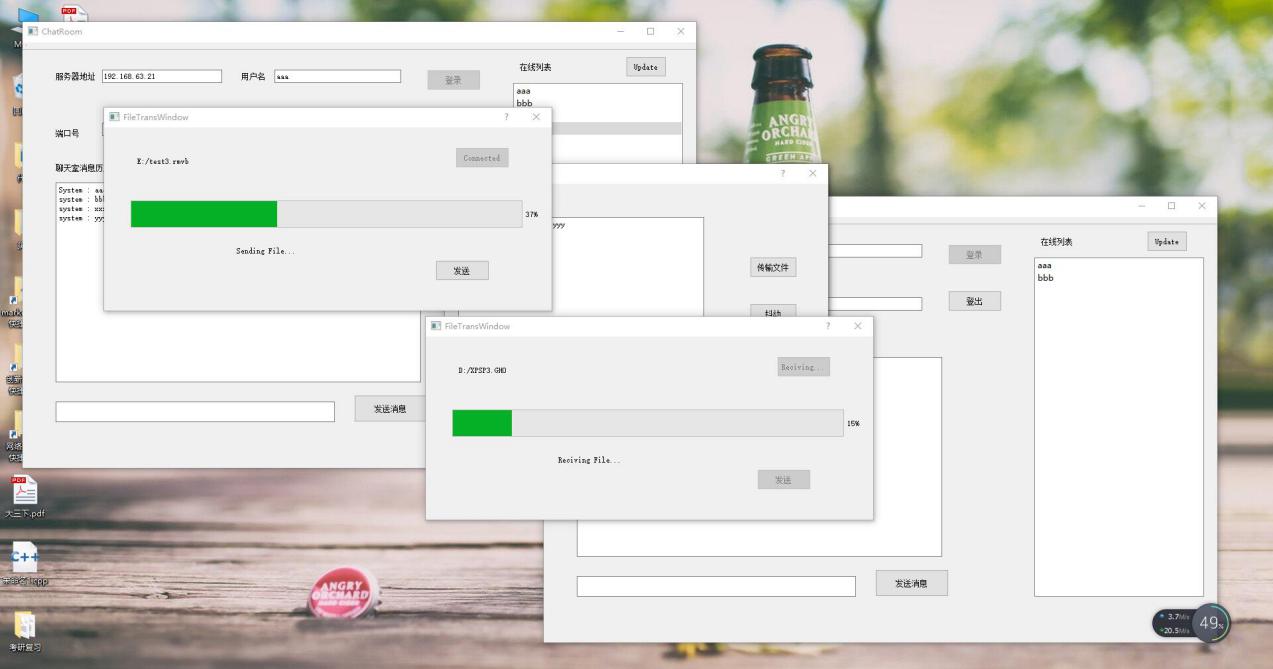
1. 公聊



1. 私聊



（5）传文件



四、系统测试

1. 功能测试

多个客户端交替使用登录、登出、公聊、私聊、窗口抖动、查询在线用户、传文件功能均正常。

1. 性能测试

（1）正常使用时，多个客户端登录、登出、公聊、私聊、窗口抖动、查询在线用户、传文件功能都没问题。

（2）使用一个四线程死循环登录登出程序（握手、登录、登出、挥手），测试了基于完成端口的服务端。在运行时服务端不会发生崩溃，并且当测试程序关闭时，服务端的session中的用户数量能回到原来正常的数量。

（3）在线传输大文件的时候，三台电脑用交换机连在一起，一台做服务端，两台做客户端。在传一个文件的时候平均速度10几MB/s，最高的时候有30MB/s；在传两个文件的时候加起来大概有20MB/s；更多文件同时传输目前还没有测试，陈孟春表示接收文件的时候客户端的电脑很卡。