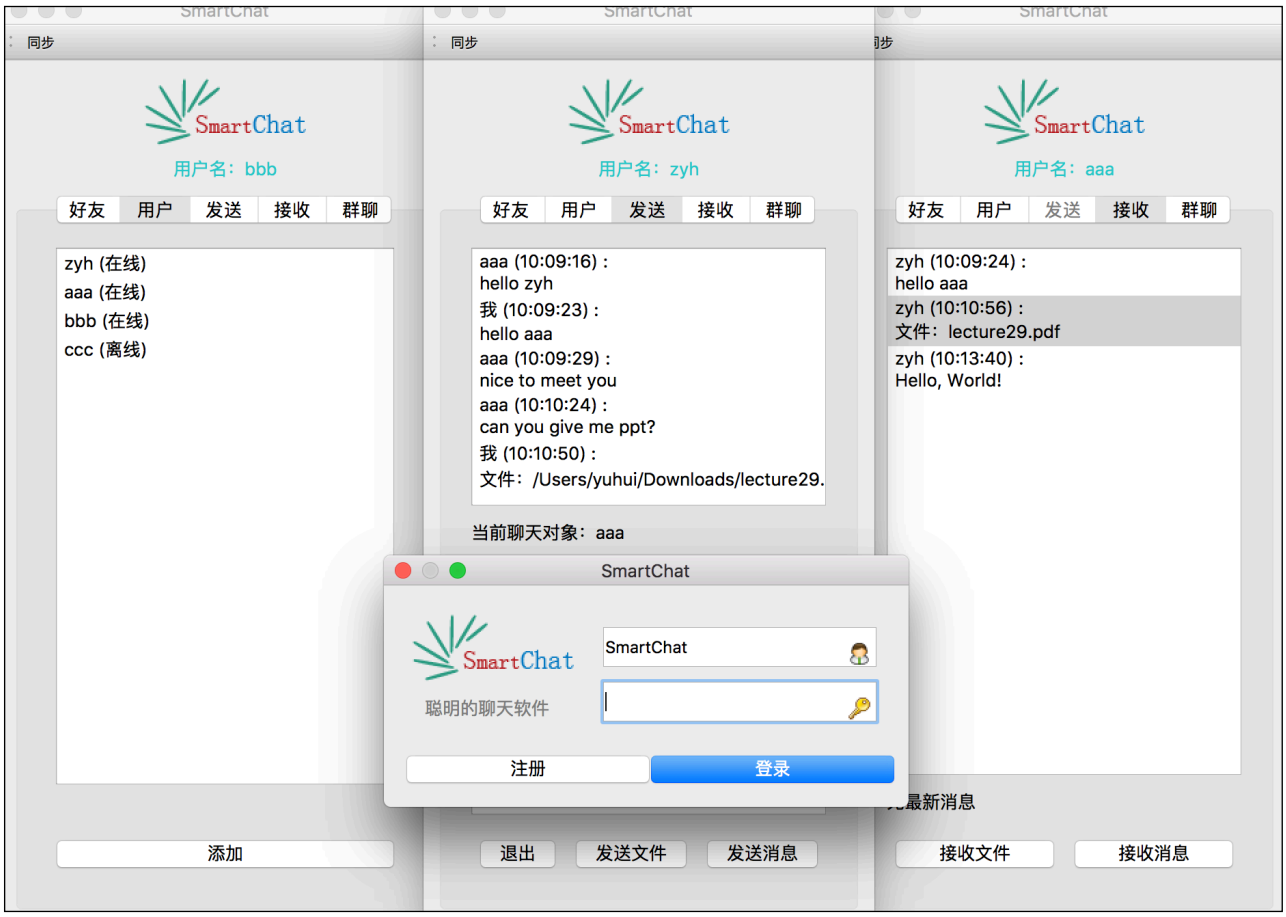
SmartChat

实验报告

张钰晖 计算机系计55班 2015011372 yuhui-zh15@mails.tsinghua.edu.cn



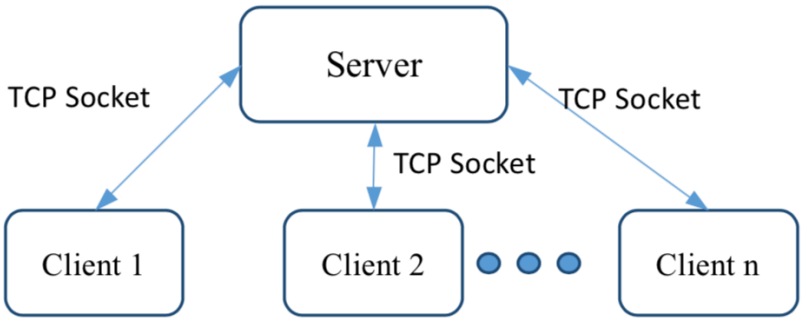
实验目的

传输消息和文件是计算机网络的一个基本功能。即时通信类软件是基于 TCP 传输的一种典型网络应用。本实验要求学生使用 socket 编程技术完成一个简易版的“微信”(一款流行的即时通信软件)。通过本实验，学生可以在掌握 socket 编程技术的基础上，深入的了解 TCP/IP 网络应用程序的基本设计方法和实现技巧。

实验说明

“微信”是腾讯公司于 2011 年 1 月 21 日推出的一款支持 Android 以及 ios 平台的即时通信软件。即时通信软件一般能够提供终端之间相互即时通信的服务， 其中大部分提供状态信息的特性——显示联系人名单、联系人是否在线以及能否 进行通信等。

即时通信软件目前一般有两种架构形式，一种是 C/S 架构，即客户/服务器结 构，用户在使用过程中需要下载安装相应的软件，C/S 架构的基本结构如图 1 所 示，客户端要与其联系人通信时，需将信息先传递给即时通信软件的服务器端， 由服务器端对状态进行管理，并将客户端发送的数据即时传递给目标联系人。另一种架构形式是 B/S 架构，即浏览器/服务器形式，这种架构可以使用户借助浏 览器直接与服务器端进行通信，一般运用于电子商务网站。本实验要求学生采用 C/S 架构，并基于 socket 编程技术来实现一个简易版的即时通信软件。



实验内容

本实验要求学生在 Linux 系统上使用 Socket 接口实现简易版“微信”服务器和客户端程序。服务器程序要求使用 C/C++语言实现，客户端运行平台及实现语 言不限制，运行平台可以使用 Linux、Windows、Android 等，语言可使用 C/C++、 Java、Python 等。多个客户端能同时连接到服务器程序，实现实时聊天、文件传 输等基本功能。本实验要求实现的基本功能如下:

1) 用户登录

2) 查找并添加好友

3) 好友间实时聊天(离线后重新登录也能收到消息)

4) 好友间文件传输(需考虑二进制文件)

5) 用户账户、联系人信息同步。

对于客户端之间通讯的交互方式，图形化界面可通过鼠标、手势等实现，非图形化界面可通过命令实现。下面给出基本功能的命令:

- login [用户名] [密码]:用户名及密码登录客户端，如 login bob passwd
- search:查找所有人
- add [用户名]:按用户名添加好友，如 add alice
- ls:显示已添加好友(包括在线和非在线)
- chat [用户名]:进入与好友通讯、传输文件界面(注意:为降低复杂度，在进入好友聊天界面后，可以不接受其他好友的消息或文件)
- sendmsg[消息]:输入消息，按回车发送，如sendmsghello
- sendfile[文件路径名]:向好友发送文件，如sendfile~/Downloads/foo.txt
- exit:退出当前聊天窗口
- recvmsg:在非聊天窗口，接受好友发送的消息(默认为最近未被接受的消息)
- recvfile:在非聊天窗口，接受好友发送的文件(默认为最近未被接受的文件，且默认存储路径为~/Downloads/)
- profile:显示用户账户信息
- sync:同步联系人信息

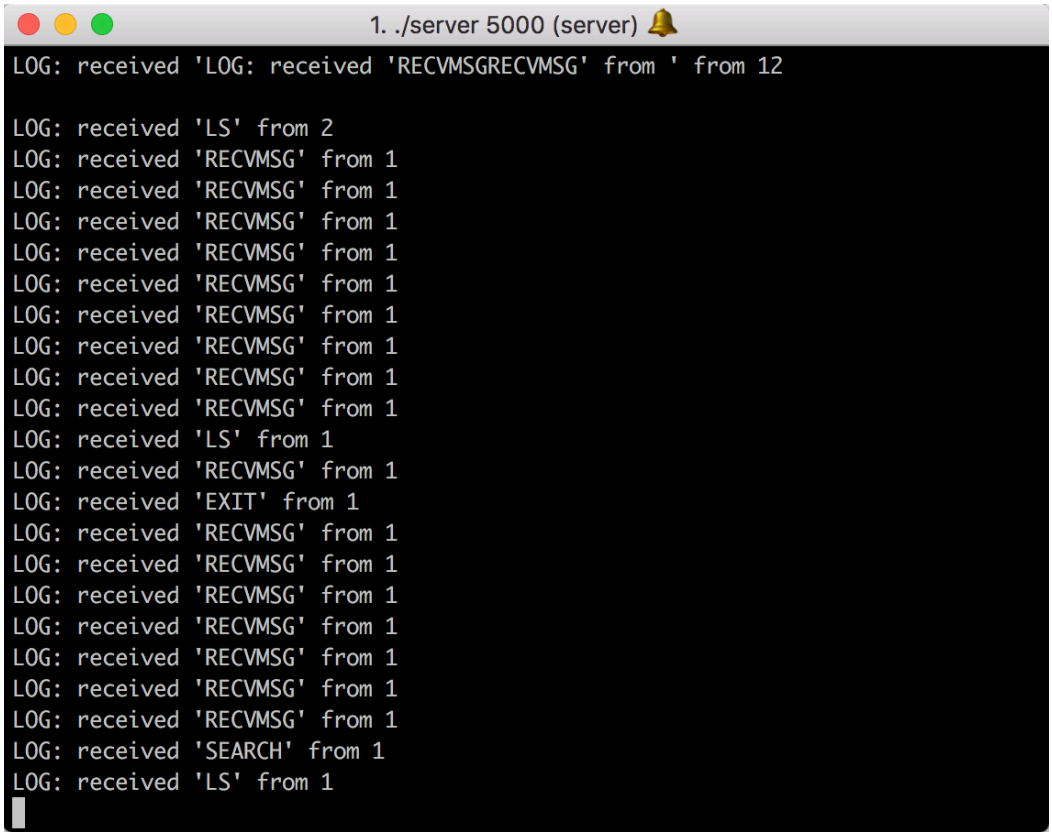
功能说明

在全部实现基本要求的基础上，增加了以下功能:

- 精美的GUI: 客户端采用Qt制作了精美的GUI，界面相对比较美观，笔者还通过PS自己设计了SmartChat的LOGO，使得界面美观性进一步提升。同时提供了Python客户端。
- 状态显示功能: 实时更新好友在线状态。
- 群聊功能: 通过群聊页面可以向所有人群发消息。
- 超级缓存功能: 设计简易文件系统，用户重新登录消息文件不丢失，服务器重启用户信息不丢失。
- 多线程支持: 服务器采用多线程设计，更高效利用资源。
- 细节完善: 错误提示信息全，程序鲁棒性强，可扩展性强，压力测试无Bug，对中文支持良好。
- 全部云端化: 服务器对信息进行维护，Qt客户端实现轻量，无需存储冗余数据。

使用文档

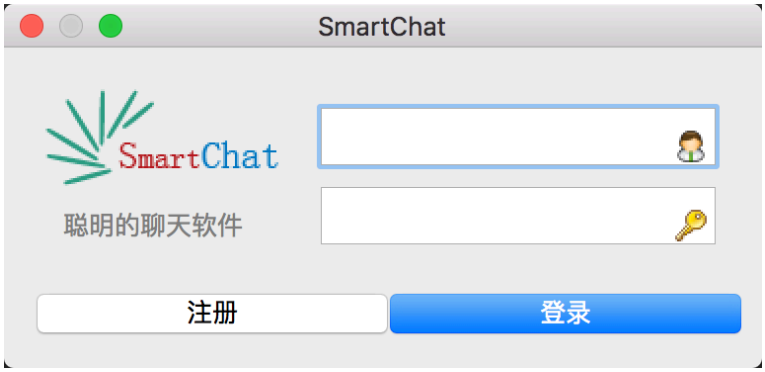
Server



终端输入``./server 端口号``开启server，开启后会输出<SERVER STARTED>，开始监听数据。

同时客户端发送的所有指令服务器会打印在屏幕上，显示发送者tid，便于调试。

Qt Client

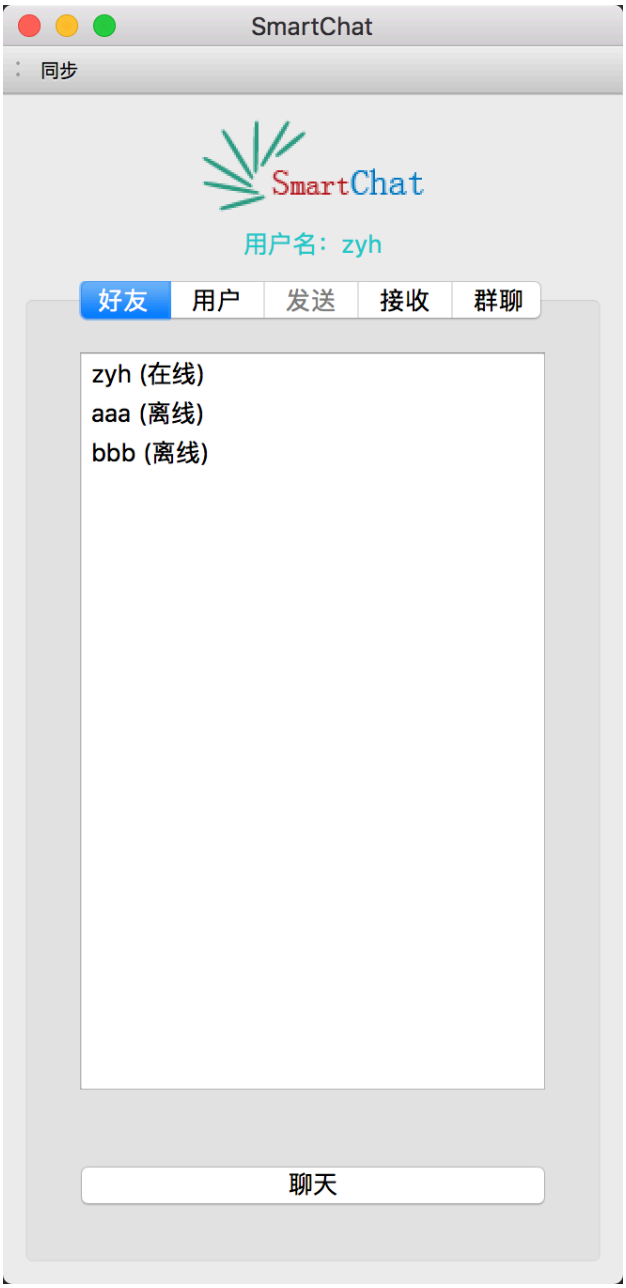


注册

若用户名未被注册，点击注册按钮便可以完成注册，否则会提示用户名已存在。

登录

若用户名已被注册，若密码正确点击登录按钮便可完成登录，否则会提示密码不正确，若用户名未被注册，会提示无此用户。



好友

好友页面中，会显示所有好友，并显示好友在线状态（在线或离线）。

个人信息

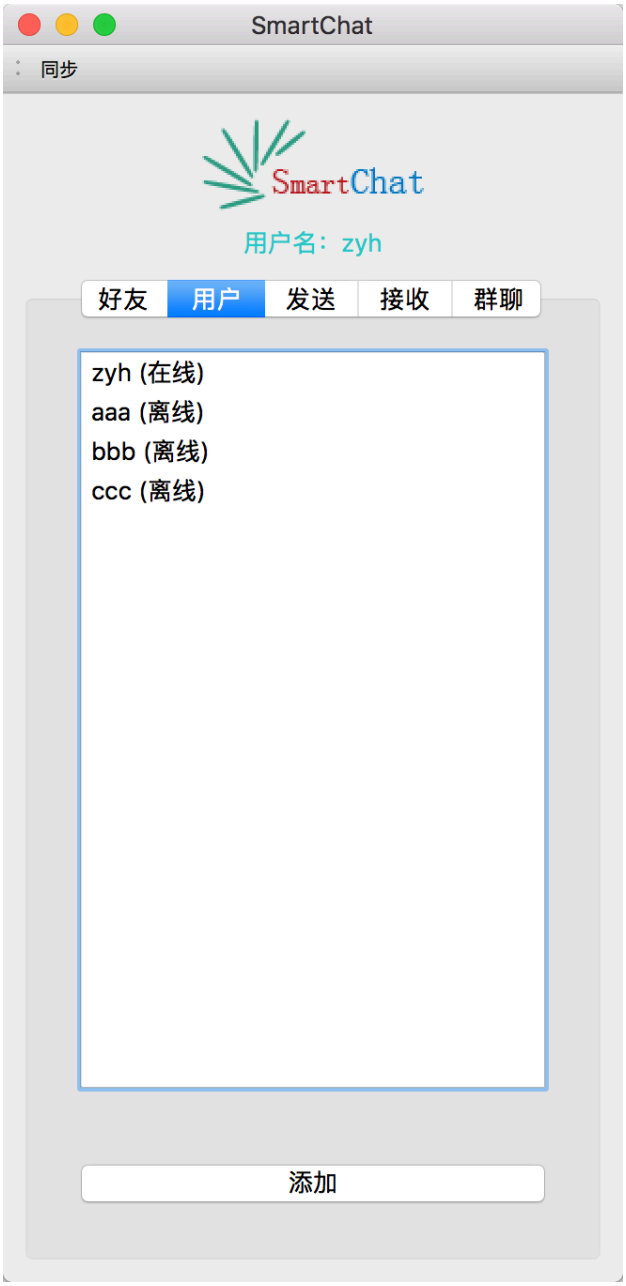
登陆后顶部将显示个人信息（用户名）。

聊天

将当前选中的好友设为聊天对象，激活发送窗口（没有聊天对象时发送页面不可用）。

同步

点击顶部同步按钮后，服务器端更新当前用户信息到文件，同时加载最新的用户和在线状态。

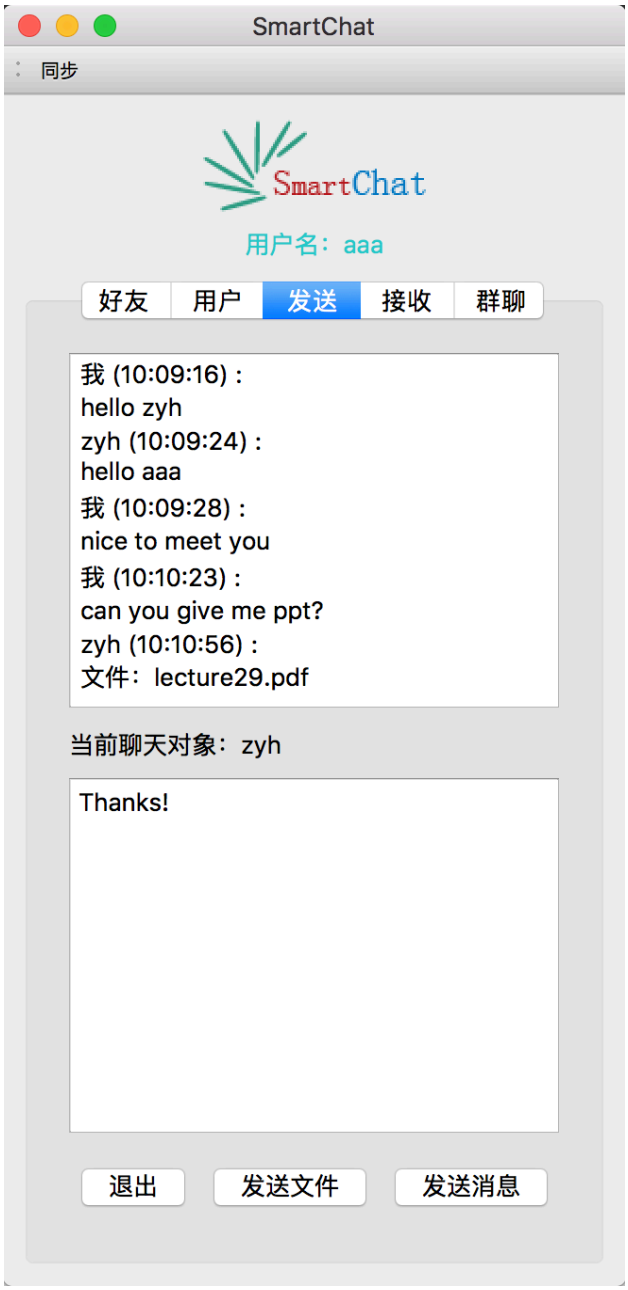


用户

用户页面中，会显示所有用户，并显示用户在线状态（在线或离线）。

加为好友

用户页面中，点击添加按钮，若选中对象尚未加为好友，则双向添加好友，若已添加，则提示好友已存在。



发送消息

发送页面中，点击发送消息按钮，发送文本框内容。

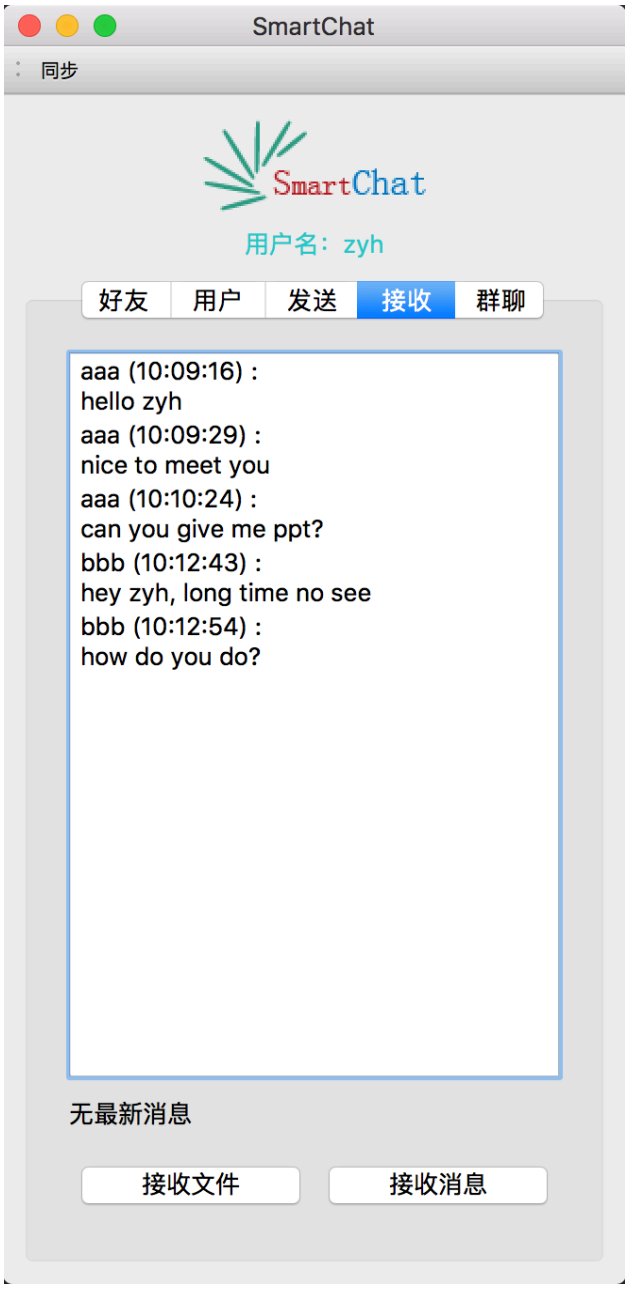
发送页面中，顶部聊天记录框会自动实时传输和聊天对象的聊天记录。

发送文件

发送页面中，点击发送文件按钮，会弹出文件选择对话框，若用户选择了文件，则发送文件，否则提示用户未选择文件，该功能支持任何类型、任何大小的文件。

退出

发送页面中，点击退出按钮，会结束与当前聊天对象的聊天，同时把发送页面置为不可用。

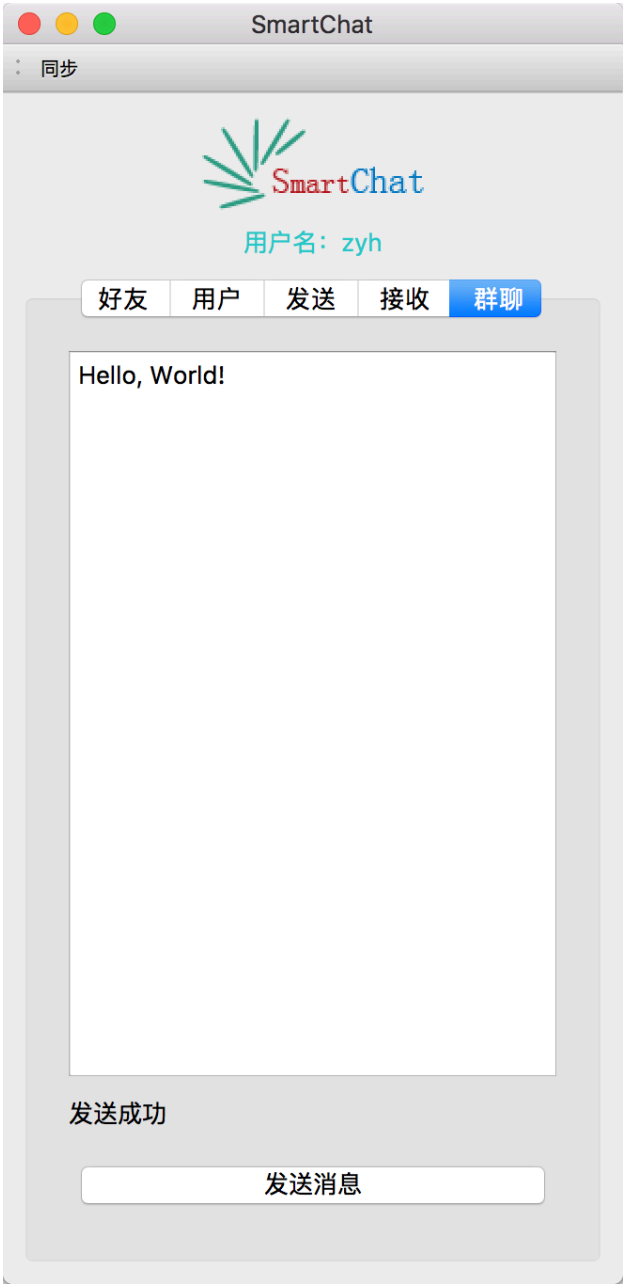


接收消息

接收页面中，点击接收消息按钮，会立刻更新消息。
接收页面中，聊天记录框会自动实时更新收到的消息，理论上无需手动点击按钮。

接收文件

接收页面中，点击接收文件按钮，会立刻更新文件。



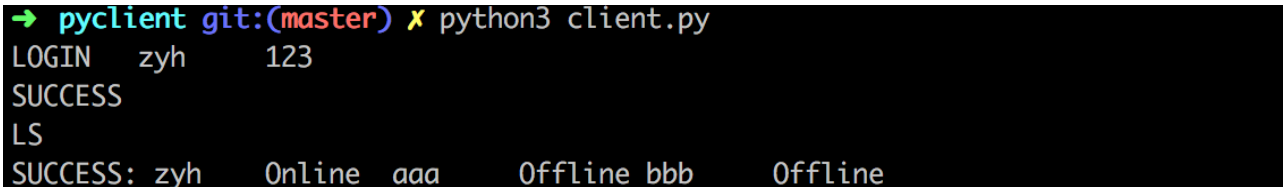
群聊

群聊页面中，点击发送按钮，会向所有用户广播信息。

超级缓存

对用户而言，即使用户不在线，也可以上线后接收到之前好友发的消息和文件。
对服务器而言，即使服务器重启，用户信息都依然保存。

Python Client



终端输入``python3 client.py —port=端口号``开启client，连接后输入指令即可，指令字段为大写，指令参数以Tab分割，指令格式完全同实验说明，同时增加了SENDALL群发指令。

由于有Qt版本，Python版本制作相对简略。

设计文档

报文格式

报文格式一致采用“指令 + <TAB> + 指令参数列表”的形式，其中指令参数列表以<TAB>分隔。

例如向用户zyh发送消息，报文为”SENDMSG + <TAB> + zyh + <TAB> + message”。

Server

Server端整体采用C++实现。

User类记录了用户的所有信息，例如id，name，status，friends，messages，files等字段。

```
class User {
public:
    // 构造函数
    User(int id, std::string name, std::string password);
    User(int id);
    // 修改函数
    void SetName(std::string name);
    void SetPassword(std::string password);
    void SetChatId(int chatid);
    void SetStatus(int status);
    void AddFriend(int id);
    void AddMessage(std::string name, std::string message);
    void AddFile(std::string name, std::string message);
    // 获取函数
    std::string GetName();
    std::string GetPassword();
    int GetChatId();
    std::vector<int> GetFriends();
    std::vector<std::string> GetMessages();
    std::vector<std::string> GetFile();
    std::string GetStatus();
    // 功能函数
    void CleanMessages();
    void CleanFile();
    int IsFrientExisted(int id);
    // 数据输出输入函数
    void StoreUser();
    void LoadUser();
private:
    std::string name; // 用户名
    std::string password; // 密码
    int chatid; // 当前聊天对象
    std::string status; // 在线状态
    int id; // 用户唯一标识ID
    std::vector<int> friends; // 好友列表
    std::vector<std::string> messages; // 消息列表
    std::vector<std::string> file; // 文件列表
    int fd; // 文件指针
};
```

Data类实现了简易的数据库管理，支持增、删、改用户信息。

```
class Data
{
public:
    // 构造函数
    Data();
    // 修改函数
    void AddUser(std::string name, std::string password);
    // 获取函数
    std::vector<User> GetUsers();
    User GetUser(int index);
    int GetUserCount();
    // 功能函数
    int IsUserExisted(std::string name);
    void SetUser(User user, int index);
    // 数据存储读入函数
    void LoadData();
    void StoreData();
private:
    std::vector<User> users; // 所有用户，用户id作为vector的index
    int usercnt; // 用户数量
};
```

Utils类自己实现了一些常用的字符串处理函数。

```
std::vector<std::string> Split(std::string s, char c) // 字符串分割函数
```

Server利用标准库socket实现通讯协议，利用标准库pthread实现多线程，利用User和Data类实现简易文件系统。

每个用户连接Server时，会给该用户新建一个线程，分配一个唯一的tid，连接处理函数，将其插入队列中。

```
int main(int argc, char *argv[])
{
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addr;
    pthread_t thread;
    /* Socket settings */
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));
    /* Ignore pipe signals */
    signal(SIGPIPE, SIG_IGN);
    /* Bind */
    if (bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("Socket binding failed");
        return 1;
    }
    /* Listen */
    if (listen(listenfd, 10) < 0) {
        perror("Socket listening failed");
        return 1;
    }
    std::cerr << "<SERVER STARTED>" << std::endl;
    /* Accept clients */
    while (true) {
        socklen_t clilen = sizeof(cli_addr);
        connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);
        /* Check if max clients is reached */
        if ((cli_count + 1) == MAX_CLIENTS) {
            std::cerr << "<<MAX CLIENTS REACHED\n";
            std::cerr << "<<REJECT ";
            PrintClientAddr(cli_addr);
            std::cerr << "\n";
            close(connfd);
            continue;
        }
        /* Client settings */
        Client *cli = new Client;
        cli->addr = cli_addr;
        cli->connfd = connfd;
        cli->tid = tid++;
        cli->uid = -1;
        /* Add client to the queue and fork thread */
        QueueAdd(cli);
        pthread_create(&thread, NULL, &HandleClient, (void*)cli);
    }
}
```

每个用户离开时，会根据该用户的tid，从队列中将其删除。

```
void *HandleClient(void *arg) {
    /* Receive input from client */
    while (true) {...}
    /* Close connection */
    close(cli->connfd);
    /* Delete client from queue and yeild thread */
    QueueDelete(cli->tid);
    printf("Leave ");
    PrintClientAddr(cli->addr);
    printf(" referenced by %d\n", cli->tid);
    free(cli);
    cli_count--;
    pthread_detach(pthread_self());
}
```

处理函数会根据指令类型，进入不同的子处理函数处理指令。

```
void *HandleClient(void *arg) {
    /* Receive input from client */
    while (true) {
        if (command == "LOGIN") {
            std::string username = str_in_list[1];
            std::string password = str_in_list[2];
            msg = HandleLogin(username, password, cli);
        }
        else if (command == "REGISTER") {
            std::string username = str_in_list[1];
            std::string password = str_in_list[2];
            msg = HandleRegister(username, password);
        }
        else if (command == "SEARCH") {
            msg = HandleSearch();
        }
        else if (command == "LS") {
            msg = HandleLs(cli->uid);
        }
        else if (command == "ADD") {
            std::string username = str_in_list[1];
            msg = HandleAdd(cli->uid, username);
        }
        ...
        else {
            msg = "FAIL: Unknown Command";
        }
        SendMessageSelf(msg.c_str(), cli->connfd, msg.length());
    }
}
```

Qt Client

图形界面通过Qt Designer绘制。

Qt通过信号与槽(slots)机制，将用户在GUI的操作（例如点击按钮，更换页面等）绑定到函数上，从而实现响应。

```
class LoginDialog : public QDialog
{
    Q_OBJECT
public:
    explicit LoginDialog(QWidget *parent = 0);
    ~LoginDialog();
private slots:
    void on_loginButton_clicked();
    void on_registerButton_clicked();
private:
    Ui::LoginDialog *ui;
    QTcpSocket *socket;
};
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QTcpSocket *socket, QWidget *parent = 0);
    ~MainWindow();
private slots:
    void on_tabWidget_currentChanged(int index);
    void on_addButton_clicked();
    void on_chatButton_clicked();
    void on_sendButton_clicked();
    void on_recvButton_clicked();
    void on_exitButton_clicked();
    void on_syncAction_triggered();
    void on_sendFileButton_clicked();
    void on_recvFileButton_clicked();
    void on_sendAllButton_clicked();
private:
    void closeEvent(QCloseEvent *event);
private:
    Ui::MainWindow *ui;
    QTcpSocket *socket;
    QString currentChatName;
    QTimer *timer;
};
```

具体槽函数实现和C++函数实现没有本质区别。

```
void LoginDialog::on_loginButton_clicked()
{
    QString name = ui->nameEdit->text();
    QString password = ui->passwordEdit->text();
    if (name.length() == 0 || password.length() == 0) {
        ui->infoLabel->setText("账号密码不能为空");
    }
    else {
        QString req = QString("LOGIN") + QString("\t") + name + QString("\t") +
password;
        this->socket->write(req.toStdString().c_str());
        char cans[BUFFER_SIZE];
        memset(cans, 0, sizeof(cans));
        this->socket->waitForReadyRead();
        this->socket->read(cans, BUFFER_SIZE);
        QString ans = QString(cans);
        QStringList anslist = ans.split(':');
        if (anslist[0] == "SUCCESS") {
            MainWindow *w = new MainWindow(this->socket);
            this->close();
            w->show();
        } else {
            ui->infoLabel->setText("登录失败");
            QMessageBox::information(this, anslist[0], anslist[1]);
        }
    }
}
```

其中有两个相对比较难实现的问题：

其一是实时消息，笔者通过设置QTimer轮询实现。

其二是文件发送，将文件读入到Qt数据结果QByteArray中，在发送的时候将该数据分段，每次发送包大小为4096Byte，同时服务器设置一个fd状态机记录发送进度，从而解决了大文件发送问题。

Python Client

Python Client主要用于服务器调试，实现较为简单，直接将指令发送给服务器，不再赘述。

思考题

1）Linux 系统里，Socket 与文件的关系。

答：Socket最先应用于Linux操作系统，Linux的设计思想是”一切皆文件“，Socket是种特殊的文件，Socket数据传输其实就是一种特殊文件读写。Socket有文件描述符，文件描述符的本质是一个非负整数，只是用于区分，因此可对Scoket进行文件操作。在Linux系统中，Socket和普通文件一样对待，它可以像普通文件一样被读和写，但是它还有一些自己独特的特点，例如，文件的读写位置可以设置，但是Socket只能被顺序的读写等等。

2) 即时通信时，服务器程序的角色。

答：即时通信中，服务器扮演着管理协调者的角色。一方面，服务器记录所有用户信息，解析用户请求，根据用户请求完成连接的建立。另一方面，服务器存储消息内容，确保用户离线也可以收到服务器转发的信息。

3) 服务器端口与客户端连接个数的关系。

答：服务器在某个端口上监听，当新客户端加入时，生成一个新的Socket与客户端通讯。同一时刻，一个端口只能建立一个连接，但是服务器在监听端口的同时，生成一个等待队列，每一个来自客户端的连接都会送入等待队列中，服务器利用一定的算法进行选择相应的连接请求处理，所以在一个端口可以监听多个请求。因此没有必要每个客户分配一个端口，会造成无谓的浪费。但如果同时的连接过多，服务器相应每个连接的时间就会相应的变长，就会变慢。

实验心得

本次实验代码量较大，但完成该工程后收获颇丰。

通过本次实验，我更加深入的理解了网络通信协议，初步掌握了Socket编程技术的基础，也进一步学习了Qt网络编程的相关技术。

在实现的过程中，我充分考虑了大部分细节，对细节做了优化，使得最终程序美观度、易用性都较为良好。