实验一：本地机上的聊天室

实验内容

在linux平台上使用守护进程实现服务器，用于支持宿主机和虚拟机上的客户端进行聊天。客户端使用图形界面实现。

准备工作

服务器端：运行在腾讯云轻量应用服务器上的守护进程。

客户端：运行在本地主机的 Qt C++ 程序。

通信协议：TCP协议。

代码实现

服务端

为 `server.cpp`.

宏定义部分

```
#define PORT 12345                          // 在端口 12345 监听
#define BUFFER_SIZE 1024                    // 接收缓冲区大小
#define LOG_FILE "/tmp/chat_server.log"  // 日志文件路径
```

守护进程

将程序转为后台运行的守护进程。

```
void daemonize() {
    pid_t pid = fork();        // 第一次 fork
    if (pid < 0) exit(EXIT_FAILURE);
    if (pid > 0) exit(EXIT_SUCCESS);  // 父进程退出

    if (setsid() < 0) exit(EXIT_FAILURE); // 创建新会话

    signal(SIGCHLD, SIG_IGN); // 忽略子进程信号
    signal(SIGHUP, SIG_IGN);  // 忽略挂断信号

    pid = fork();              // 第二次 fork
    if (pid < 0) exit(EXIT_FAILURE);
    if (pid > 0) exit(EXIT_SUCCESS);
```

```
    umask(0);
    chdir("/");

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);
}
```

日志记录函数

将服务端的事件记录到日志文件中。

```
void log_message(const std::string& msg) {
    std::ofstream log(LOG_FILE, std::ios::app);
    log << msg << std::endl;
}
```

主函数

详见注释，直接给出代码。

```
int main() {
    std::ofstream log(LOG_FILE, std::ios::trunc);  // 清空旧日志
    log.close();

    daemonize();  // 启动守护进程
    log_message("守护进程启动...");  // 将启动守护进程的消息写入日志

    int server_fd, client_fd;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE];

    server_fd = socket(AF_INET, SOCK_STREAM, 0);  // 创建 TCP socket
    if (server_fd < 0) {
        log_message("Socket 创建失败");
        exit(EXIT_FAILURE);
    }

    int opt = 1;
    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));  //
设置端口可复用

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
```

```
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
// 绑定端口
        log_message("绑定失败");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {  // 监听端口
        log_message("监听失败");
        exit(EXIT_FAILURE);
    }

    std::vector<int> clients;
    std::map<int, std::string> clientIdMap;

    fd_set readfds;
    while (true) {
        FD_ZERO(&readfds);
        FD_SET(server_fd, &readfds);
        int max_sd = server_fd;

        for (int client : clients) {
            FD_SET(client, &readfds);
            if (client > max_sd) max_sd = client;
        }

        int activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
        if (activity < 0) {
            log_message("select 错误");
            continue;
        }

        // 新客户端连接
        if (FD_ISSET(server_fd, &readfds)) {  // 处理新客户端连接
            client_fd = accept(server_fd, (struct sockaddr*)&address,
(socklen_t*)&addrlen);
            if (client_fd >= 0) {
                clients.push_back(client_fd);
                log_message("新客户端连接: fd=" + std::to_string(client_fd));
                // 等待客户端发送ID，放在后面循环处理
            }
        }

        // 处理客户端消息
        for (auto it = clients.begin(); it != clients.end();) {
            int sd = *it;
            if (FD_ISSET(sd, &readfds)) {
                int valread = read(sd, buffer, BUFFER_SIZE);  //接收信息
                if (valread <= 0) {
                    // 客户端断开
                    std::string userId = clientIdMap.count(sd) ?
```

```
clientIdMap[sd] : std::to_string(sd);
                    log_message("客户端断开: " + userId + " (fd=" +
std::to_string(sd) + ")");
                    close(sd);

                    // 广播用户退出消息（如果已知ID）
                    if (clientIdMap.count(sd)) {
                        std::string leaveMsg = userId + " 退出了聊天室";
                        for (int client : clients) {
                            if (client != sd) {
                                send(client, leaveMsg.c_str(),
leaveMsg.size(), 0);
                            }
                        }
                        clientIdMap.erase(sd);
                    }

                    it = clients.erase(it);
                    continue;
                }

                buffer[valread] = '\0';
                std::string recvMsg(buffer);

                if (clientIdMap.count(sd) == 0) {  // 接收到用户 ID
                    // 每个客户端第一次向服务端发送的信息都是 ID
                    clientIdMap[sd] = recvMsg;
                    log_message("用户 " + recvMsg + " 已连接 (fd=" +
std::to_string(sd) + ")");
                    std::string enterMsg = recvMsg + " 进入了聊天室";

                    // 给其他客户端广播进入消息
                    for (int client : clients) {
                        if (client != sd) {
                            send(client, enterMsg.c_str(), enterMsg.size(),
0);
                        }
                    }
                } else {
                    // 普通聊天消息，广播给其他客户端
                    std::string msg = clientIdMap[sd] + ": " + recvMsg;
                    log_message(msg);

                    for (int client : clients) {
                        if (client != sd) {
                            send(client, msg.c_str(), msg.size(), 0);
                        }
                    }
                }
            }
```

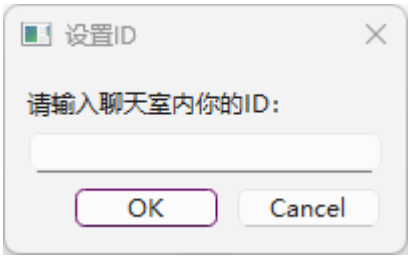
```
            ++it;
        }
    }

    return 0;
}
```

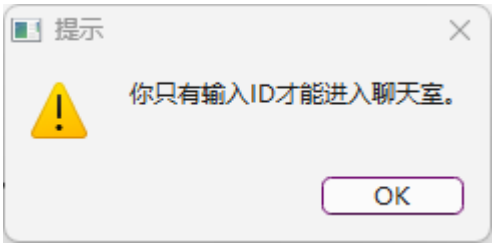
客户端

界面设计

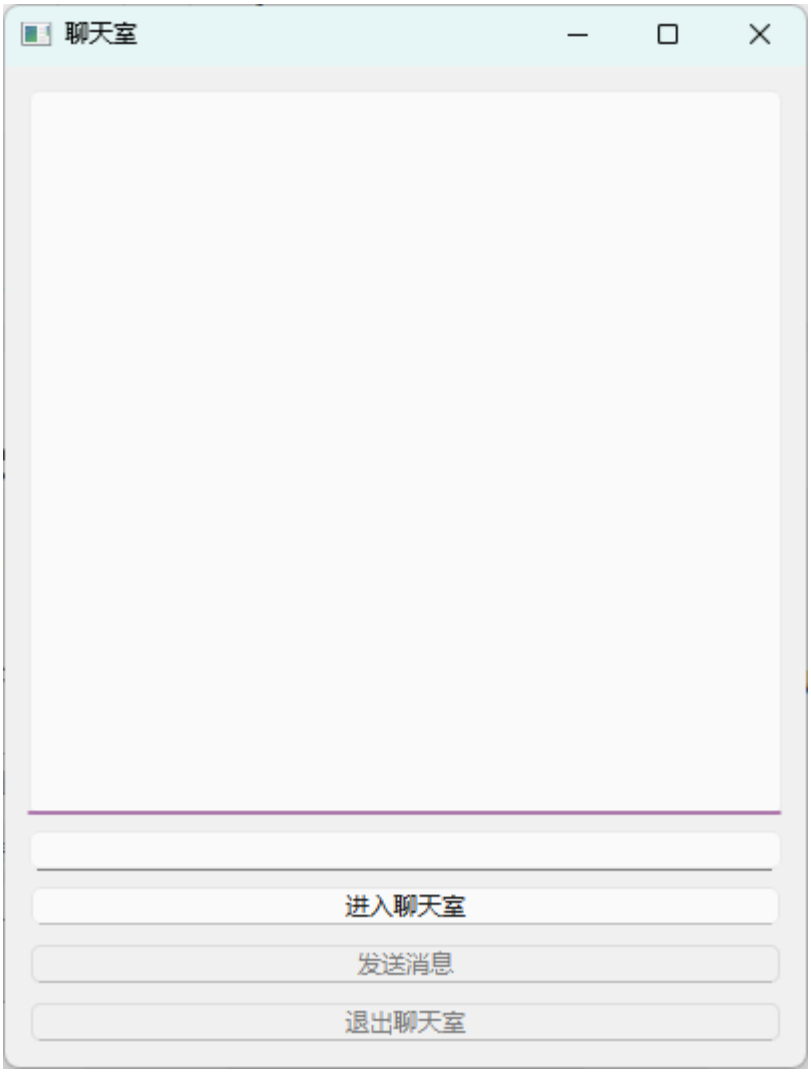
打开 `.exe` 文件后，先弹出一个 `QInputDialog` 提示设置用户 ID。‘



如果不设置 ID 则无法进入聊天室。



设置了之后即可进入聊天室。



聊天室采用了垂直布局，分为三大功能区：

- 信息显示区，`QTextEdit`
- 信息输入区，`QLineEdit`
- 功能按钮区，三个 `QPushButton`

代码

Client.h

```
#ifndef CLIENT_H
#define CLIENT_H

#include <QWidget>
#include <QTcpSocket>
#include <QPushButton>
#include <QLineEdit>
#include <QTextEdit>
#include <QVBoxLayout>

class Client : public QWidget
{
```

```
    Q_OBJECT

public:
    Client(QWidget *parent = nullptr);
    void setClientId(const QString& id);

private slots:
    void onConnectClicked();
    void onSendClicked();
    void onExitClicked();
    void onReadyRead();
    void onDisconnected();
    void onError(QAbstractSocket::SocketError);
    void onConnected();


private:
    QTcpSocket   *socket;
    QLineEdit    *inputLine;
    QTextEdit    *chatArea;
    QPushButton  *connectButton;
    QPushButton  *sendButton;
    QPushButton  *exitButton;
    QString clientId;

    QString serverIp   = "139.155.108.193";  //云服务器的公网 IP
    quint16 serverPort = 12345;  // 端口 12345
};

#endif // CLIENT_H
```

Client.cpp

```
#include "Client.h"
#include <QMessageBox>
#include <QInputDialog>

Client::Client(QWidget *parent)
    : QWidget(parent)
{
    // 创建控件
    chatArea      = new QTextEdit(this);
    chatArea->setReadOnly(true);

    inputLine     = new QLineEdit(this);
    connectButton = new QPushButton("进入聊天室", this);
    sendButton    = new QPushButton("发送消息", this);
    exitButton    = new QPushButton("退出聊天室", this);
```

```
    sendButton->setEnabled(false);
    exitButton->setEnabled(false);

    // 布局
    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addWidget(chatArea);
    layout->addWidget(inputLine);
    layout->addWidget(connectButton);
    layout->addWidget(sendButton);
    layout->addWidget(exitButton);
    setLayout(layout);

    setWindowTitle("聊天室");
    resize(400, 500);

    // 初始化 socket
    socket = new QTcpSocket(this);

    // 连接信号与槽
    connect(connectButton, &QPushButton::clicked, this,
&Client::onConnectClicked);
    connect(sendButton, &QPushButton::clicked, this,
&Client::onSendClicked);
    connect(exitButton, &QPushButton::clicked, this,
&Client::onExitClicked);
    connect(inputLine, &QLineEdit::returnPressed, this,
&Client::onSendClicked);
    connect(socket, &QTcpSocket::readyRead, this, &Client::onReadyRead);
    connect(socket, &QTcpSocket::disconnected, this,
&Client::onDisconnected);
    connect(socket, &QTcpSocket::errorOccurred, this, &Client::onError);
    connect(socket, &QTcpSocket::connected, this, &Client::onConnected);
}

void Client::onConnectClicked()
{
    chatArea->append("正在进入聊天室.....");
    socket->connectToHost(serverIp, serverPort);
    connectButton->setEnabled(false);
}

void Client::onSendClicked()
{
    QString msg = inputLine->text().trimmed();
    if (msg.isEmpty())
        return;

    QString fullMessage = QString("%1").arg(msg);
    socket->write(fullMessage.toUtf8());
```

```
    chatArea->append(QString("<span style=\"color:blue;\"><b>我</b>:
%1</span>").arg(msg));

    inputLine->clear();
}

void Client::onExitClicked()
{
    int ret = QMessageBox::question(this, "退出确认", "确定要退出聊天室吗？",
QMessageBox::Yes | QMessageBox::No);
    if(ret == QMessageBox::Yes) {
        if(socket->state() == QAbstractSocket::ConnectedState) {
            QString leaveMsg = QString("%1 退出了聊天室。").arg(clientId);
            socket->write(leaveMsg.toUtf8());
            socket->waitForBytesWritten(100);
            socket->disconnectFromHost();
        }
        close();
    }
}

void Client::onReadyRead()
{
    QByteArray data = socket->readAll();
    QString msg = QString::fromUtf8(data);

    chatArea->append(QString("<span
style=\"color:green;\">%1</span>").arg(msg));

    if (!sendButton->isEnabled())
        sendButton->setEnabled(true);
}

void Client::onDisconnected()
{
    chatArea->append("<span style=\"color:red;\">已退出聊天室</span>");
    connectButton->setEnabled(true);
    sendButton->setEnabled(false);
    exitButton->setEnabled(false);
}

void Client::onError(QAbstractSocket::SocketError socketError)
{
    Q_UNUSED(socketError);
    QMessageBox::critical(this, "网络错误",
                          socket->errorString());
    connectButton->setEnabled(true);
    sendButton->setEnabled(false);
    exitButton->setEnabled(false);
}
```

```
void Client::onConnected()
{
    chatArea->append("<span style=\"color:green;\">已进入聊天室</span>");
    socket->write(clientId.toUtf8());
    sendButton->setEnabled(true);
    exitButton->setEnabled(true);
}

void Client::setClientId(const QString& id)
{
    clientId = id;
}
```

main.cpp

```
#include "Client.h"

#include <QApplication>
#include <QInputDialog>
#include <QMessageBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    // 用户ID
    bool ok;
    QString clientId = QInputDialog::getText(nullptr, "设置ID", "请输入聊天室内你的ID: ", QLineEdit::Normal, "", &ok);

    if(!ok || clientId.trimmed().isEmpty()) {
        QMessageBox::warning(nullptr, "提示", "你只有输入ID才能进入聊天室。");
        return 0;
    }

    Client w;
    w.setClientId(clientId);
    w.show();
    return a.exec();
}
```

运行步骤

1. 首先为云服务器添加安全组，放行端口 12345
2. 启动服务，切换到服务端程序所在目录，编译并运行。

```
# 编译
g++ -o server server.cpp
#运行
./server
```

3. 验证端口监听

```
sudo netstat -tlnp | grep 12345
```

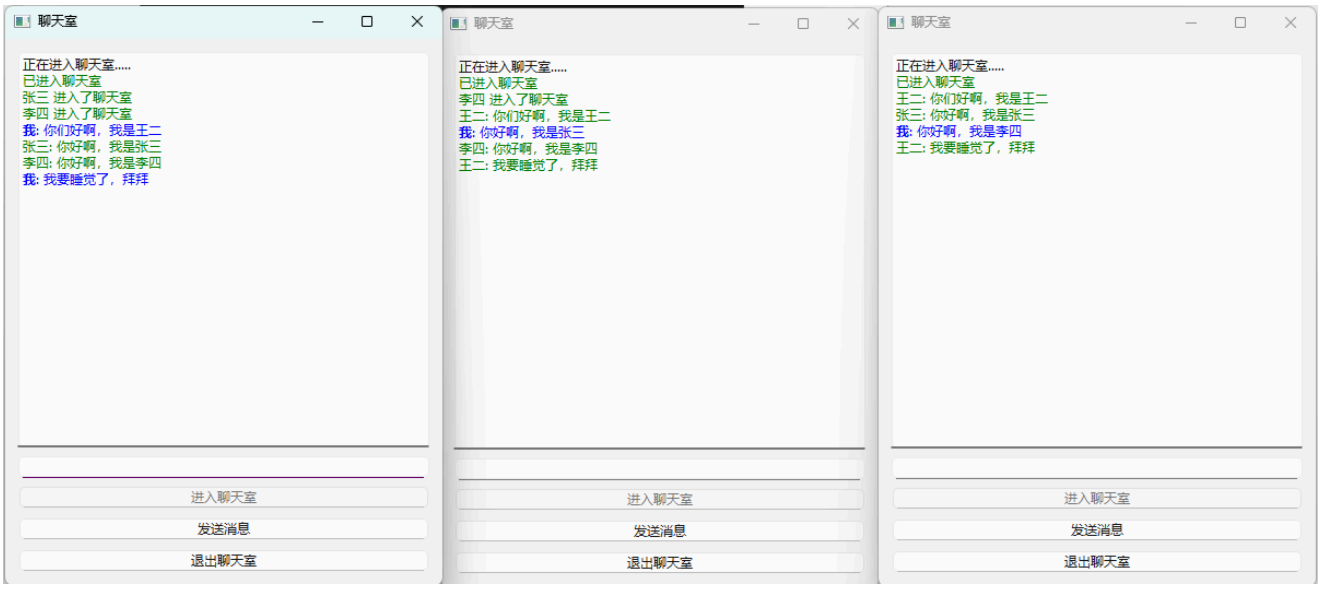
4. 实时查看日志，观察聊天室状态

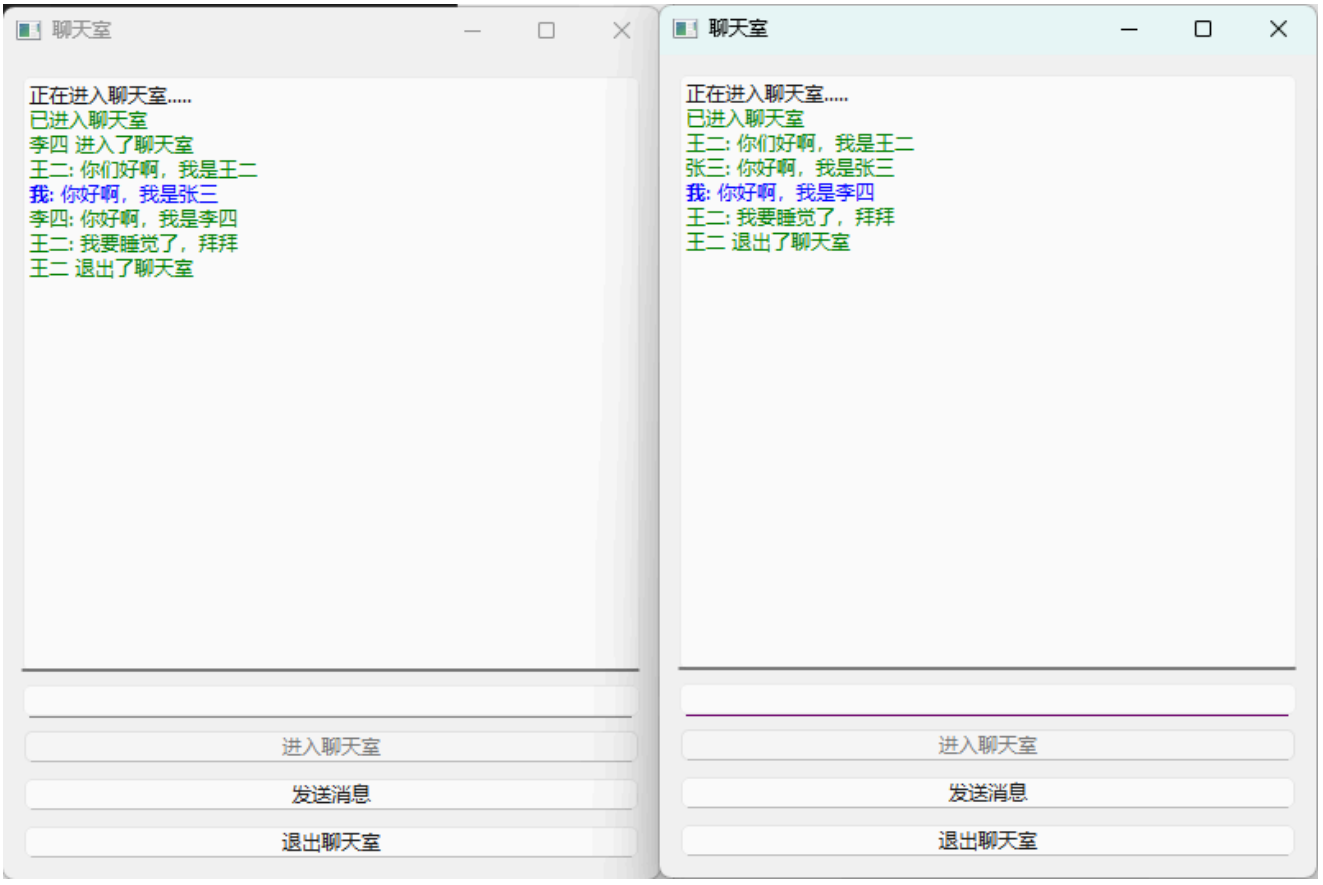
```
tail -f /tmp/chat_server.log
```

5. 主机的 Qt 客户端编译运行
6. 打开 Qt 客户端编译产生的 `.exe` 文件，设置 ID 进入聊天室即可进行聊天。

测试验证

添加了一个用户退出聊天室会向其他用户广播该用户退出了聊天室的功能。


聊天室
正在进入聊天室.....
已进入聊天室
张三 进入了聊天室
李四 进入了聊天室
我: 你们好啊，我是王二
张三: 你好啊，我是张三
李四: 你好啊，我是李四
我: 我要睡觉了，拜拜
进入聊天室
发送消息
退出聊天室
聊天室
正在进入聊天室.....
已进入聊天室
李四 进入了聊天室
王二: 你们好啊，我是王二
我: 你好啊，我是张三
李四: 你好啊，我是李四
王二: 我要睡觉了，拜拜
进入聊天室
发送消息
退出聊天室
聊天室
正在进入聊天室.....
已进入聊天室
王二: 你们好啊，我是王二
张三: 你好啊，我是张三
我: 你好啊，我是李四
王二: 我要睡觉了，拜拜
进入聊天室
发送消息
退出聊天室


聊天室
正在进入聊天室.....
已进入聊天室
李四 进入了聊天室
王二: 你们好啊，我是王二
我: 你好啊，我是张三
李四: 你好啊，我是李四
王二: 我要睡觉了，拜拜
王二 退出了聊天室
进入聊天室
发送消息
退出聊天室
聊天室
正在进入聊天室.....
已进入聊天室
王二: 你们好啊，我是王二
张三: 你好啊，我是张三
我: 你好啊，我是李四
王二: 我要睡觉了，拜拜
王二 退出了聊天室
进入聊天室
发送消息
退出聊天室

```
ubuntu@VM-0-16-ubuntu:~/labs/lab1/chat_server$ tail -f /tmp/chat_server.log
守护进程启动...
新客户端连接：fd=1
用户 王二 已连接 (fd=1)
新客户端连接：fd=2
用户 张三 已连接 (fd=2)
新客户端连接：fd=3
用户 李四 已连接 (fd=3)
王二：你们好啊，我是王二
张三：你好啊，我是张三
李四：你好啊，我是李四
王二：我要睡觉了，拜拜
客户端断开：王二 (fd=1)
客户端断开：李四 (fd=3)
客户端断开：张三 (fd=2)
```