Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

09.03.01 Информатика и вычислительная техника

Профиль: Программное обеспечение средств вычислительной техники и автоматизированных систем

**Отчёт по курсовой работе**

**по дисциплине «Операционные системы»**

**по теме**

**«Многопользовательский сетевой чат»**

по направлению 09.03.01 «Информатика и вычислительная техника»,

направленность (профиль) – «Программное обеспечение средств вычислительной техники и автоматизированных систем», квалификация – бакалавр,

программа академического бакалавриата,

форма обучения – очная, год начала подготовки (по учебному плану) – 2018

Выполнил: студент 3 курса гр. ИП-813 / Бурдуковский И.А. /

Проверил: ассистент кафедры ПМиК /Нужнов А.В. /

Новосибирск, 2020

**Содержание**

[Введение 3](#_heading=h.gjdgxs)

[1. Постановка задачи 4](#_heading=h.30j0zll)

[2. Анализ предметной области 5](#_heading=h.1fob9te)

[2.1. Цель создания 5](#_heading=h.3znysh7)

[2.2. Перечень решаемых задач 5](#_heading=h.2et92p0)

[2.3. Специальное и общесистемное программное обеспечение 5](#_heading=h.tyjcwt)

[3. Программное проектирование 6](#_heading=h.3dy6vkm)

[3.1. Функциональное обеспечение 6](#_heading=h.1t3h5sf)

[3.2. Алгоритмическое обеспечение 6](#_heading=h.4d34og8)

[3.3. Архитектурное обеспечение 6](#_heading=h.2s8eyo1)

[3.4. Информационное обеспечение 7](#_heading=h.17dp8vu)

[4. Программная реализация 8](#_heading=h.3rdcrjn)

[5. Сопроводительная документация программного обеспечения 22](#_heading=h.26in1rg)

[5.1. Разработка описания программы 22](#_heading=h.lnxbz9)

[5.2. Разработка руководства пользователя 22](#_heading=h.35nkun2)

[5.3. Анализ программного обеспечения 22](#_heading=h.1ksv4uv)

[6. Заключение 23](#_heading=h.44sinio)

[7. Список использованных источников 24](#_heading=h.2jxsxqh)

# Введение

Работая в сети Ethernet, мы очень часто встречаемся с разного рода многопользовательскими программами. Ими могут быть почтовые клиенты, чаты, форумы, FTP клиенты и т.п. Все эти приложения используют для своей работы разного рода протоколы: FTP, POP, SMTP, HTTP, и т.д. Но базовым для них является единый протокол - TCP/IP. Типичное же приложение TCP/IP построено на клиент-серверной архитектуре. Примером приложения, построенного на данной архитектуре, является чат реального времени. При работе с протоколом TCP/IP используют интерфейс сокетов. Но прежде чем перейти к разработке чата, разберем, что же такое сокеты.

**Сокеты** – это интерфейс прикладного программирования для сетевых приложений TCP/IP. В переводе с английского "sockets" – гнезда, т.е. сетевые приложения используют сокеты, как виртуальные разъемы для обмена данными между собой. Сокеты бывают трех видов: клиентские, слушающие и серверные. Клиентские сокеты устанавливают связь с сервером и обмениваются с ним данными. Серверный сокет обменивается данными с клиентом по уже установленному (слушающим сокетом) соединению. Для того чтобы клиент мог установить соединение с сервером, ему необходимо указать его адрес (IP) и номер порта, через который будет происходить обмен данными.

В данной курсовой работе предполагается реализовать программное обеспечение c графическим интерфейсом для обмена сообщениями посредством локальной или беспроводной сети.

Основное предназначение реализуемого программного продукта - передача символьной информации, а также файлов. Язык, выбранный для реализации курсового проекта – С++ - мощный язык программирования, подходящий для решения любых задач.

Данный программный продукт может использоваться пользователями в офисе, а также дома.

# 1. Постановка задачи

Я выбрал в качестве курсовой работы – создание сетевого чата на основе сокетов и именованных каналов, включающее в себя следующие пункты, которые мне следовало реализовать в приложении:

Разработка сетевого приложения, реализующего:

а) сетевую игру/чат на основе сокетов.

# 2. Анализ предметной области

Предметная область проекта - передача информации по сети. Чат, позволяющий организовать общение между неограниченным количеством пользователей.

Требуется разработать приложение, которое включает в себя разработку следующих функциональных частей: разработку клиент-серверного приложения.

## 2.1. Цель создания

Главная цель курсовой работы по теме «Многопользовательский сетевой чат» – создание многопользовательского сетевого чата с возможностью обмена текстовыми сообщениями между пользователями, подключенными к одному серверу.

## 2.2. Перечень решаемых задач

Функционал данного сетевого чата предусматривает выполнение следующих задач:

* Подключение двух и более пользователей к серверу для возможности общения друг с другом;
* Обмен сообщений между клиентами;

## 2.3. Специальное и общесистемное программное обеспечение

При написании данного курсового проекта планируется использовать следующее стороннее программное обеспечение и инструментарий:

**Qt** — кроссплатформенный IDE для разработки программного обеспечения на языке программирования C++.

**QtWidgets** — компонент библиотеки Qt, содержит классы для классических приложений на основе виджетов.

**QTcpSocket** — вспомогательный подкласс QAbstractSocket, который позволяет устанавливать TCP-соединение и передавать потоки данных.

**QTcpServer** — класс, который предназначен для управления входящими TCP-соединениями.

Разработка будет осуществляться в операционной системе Windows 10. Серверное приложение будет реализовано с интерфейсом для более комфортного и доступного использования

# 3. Программное проектирование

## 3.1. Функциональное обеспечение

Прежде, чем начинать разработку необходимо заранее определить требуемый функционал нашего программного обеспечения.

В данном проекте в качестве актеров выступают пользователь (клиентское приложение) и сервер.

Пользователь взаимодействует непосредственно с серверным приложением и должен иметь возможность выполнить следующие действия:

* Присоединиться к серверу;
* Отослать сообщение клиенту;
* Принимать сообщения от клиента.

## 3.2. Алгоритмическое обеспечение

Алгоритм обработки команд сервера на клиенте.

Клиент может получать текстовую информацию: как только клиент получает текстовое сообщение, оно отображается в поле вывода информации.

## 3.3. Архитектурное обеспечение

Для взаимодействия клиента с сервером используется протокол передачи данных TCP.

## 3.4. Информационное обеспечение

Перед реализацией программы было решено обмениваться информацией по протоколу передачи данных TCP. Такой выбор был сделан с учетом специфики задачи. Протокол TCP гарантирует доставку пакетов данных в неизменном виде, последовательности и без потерь.

Сервер с клиентом обменивается одним типом данных. Если серверу пришло сообщение строкового типа, то сервер обрабатывает сообщение и рассылает их всем остальным клиентам, в том числе и отправителю.

# 4. Программная реализация

Программа состоит из двух частей: сервер и клиент, запуск производится с помощью соответствующих .ехе-файлов.

**Описание программы**

**Клиент**

**Комментарии (dialog.cpp)**

**Dialog::Dialog(QWidget \*parent) :QDialog(parent),ui(new Ui::Dialog)** – конструктор класса Dialog. В нем происходит вывод окна с интерфейсом для подключения к чату, подключение сигналов QTcpSocket::readyRead connected, disconnected, error(QAbstractSocket::SocketError) к соответствующим слотам обработки.

**void Dialog::onSokDisplayError(QAbstractSocket::SocketError socketError)** – обработка ошибок, связанных с подключением к сокету.

**void Dialog::onSokReadyRead() -** обработчик вывода сообщений в основное окно чата на основе полученной команды и дополнительных данных.

**void Dialog::onSokConnected() -** обработка подключения к сокету и попытка авторизовать пользователя

**void Dialog::onSokDisconnected()** – обработка отключения пользователя от сокета.

**void Dialog::on\_pbConnect\_clicked() , void Dialog::on\_pbDisconnect\_clicked() –** обработка нажатия кнопки подключения / отключения

**void Dialog::on\_cbToAll\_clicked() –** смена режима отправки сообщния между «всем» и «личное»

**void Dialog::on\_pbSend\_clicked() –** обработка нажатие кнопки отправки сообщения

**void Dialog::addToLog(QString text, QColor color)** – метод вывода всех сообщений на QListWidget в нужном формате.

**Сервер**

**Комментарии (dialog.cpp)**

**Dialog::Dialog(QWidget \*parent) :QDialog(parent),ui(new Ui::Dialog)** – конструктор класса Dialog. В нем происходит вывод окна с интерфейсом сервера, подключение сигналов messageFromGui, addLogToGui к соответствующим слотам обработки и запуск сервера на локальном хосте.

**void Dialog::onAddUserToGui(QString name) –** визуальноедобавление подключенных пользователей в userlist сервера

**void Dialog::onRemoveUserFromGui(QString name)** - визуальноеудаление подключенных пользователей из userlist сервера

**void Dialog::onMessageToGui(QString message, QString from, const QStringList &users) -** обработчик вывода сообщений всех пользователей в чат сервера

**void Dialog::onAddLogToGui(QString string, QColor color) -** обработчик вывода информационных выводов

**void Dialog::on\_pbSend\_clicked()** – обработчик нажатия кнопки отправки сообщения

**void Dialog::on\_cbToAll\_clicked() –** смена режима отправки сообщений

**void Dialog::on\_pbStartStop\_toggled(bool checked)**  - обработка кнопки включения/ выключения сервера

**void Dialog::addToLog(QString text, QColor color)** – метод вывода всех сообщений на QListWidget в нужном формате.

**Комментарии (myserver.cpp)**

**bool MyServer::doStartServer(QHostAddress addr, qint16 port)** – проверка запустился ли сервер

**void MyServer::doSendToAllUserJoin(QString name) –** отправляем клиентам список всех подключенных пользователей

**void MyServer::doSendToAllUserLeft(QString name) –** отправляем клиентам список всех оставшихся пользователей

**void MyServer::doSendToAllMessage(QString message, QString fromUsername)** – отправляем клиентам сообщение от пользователя

**void MyServer::doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername) -** отправляем клиентам приватное сообщение от пользователя

**void MyServer::doSendToAllServerMessage(QString message) –** отправляем клиентам сообщение от сервера

**void MyServer::doSendToServerMessageToUsers(QString message) –** отправляем клиентам приватное сообщение от сервера

**QStringList MyServer::getUsersOnline() const -**  получение списка пользователей

**bool MyServer::isNameValid(QString name) const –** проверка имени пользователя на подходящие знаки

**bool MyServer::isNameUsed(QString name) const –** проверка имени пользователя на повторное существование в сети

**void MyServer::incomingConnection(qintptr handle) –** при входящем подключении создаём новый объект клиента и связываем с ним сигналы визуального обновления окон сервера, затем записываем нового пользователя в список пользователей

**void MyServer::onRemoveUser(MyClient \*client)** – удаляет покинувшего пользователя из списка пользователя

**void MyServer::onMessageFromGui(QString message, const QStringList &users) –** в зависимости от выбранных пользователей выбираем как отправлять сообщение клиентам

**Комментарии (myclient.cpp)**

**MyClient::MyClient(int desc, MyServer \*serv, QObject \*parent) :QObject(parent) –** аналог конструктора dialog.cpp из клиентской части.

**void MyClient::onDisconnect()** – обработка отсоединения пользователя

**void MyClient::onReadyRead() –** обработка запросов от клиента

**void MyClient::doSendCommand(quint8 comm) const -**  отправка клиенту кода ошибки в случае неудачной авторизации

**void MyClient::doSendUsersOnline() const** – отправка клиентам списка пользователей в сети

**5. Код программы**

**КЛИЕНТСКАЯ ЧАСТЬ:**

Dialog.h:

#include <QDialog>

#include <QTcpSocket>

#include <QHostAddress>

#include "D:\labs\OS\kurswork\server/myclient.h"

#include <QMessageBox>

class MyClient;

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

Q\_OBJECT

public:

explicit Dialog(QWidget \*parent = 0);

~Dialog();

private slots:

void onSokConnected();

void onSokDisconnected();

void onSokReadyRead();

void onSokDisplayError(QAbstractSocket::SocketError socketError);

void on\_pbConnect\_clicked();

void on\_pbDisconnect\_clicked();

void on\_cbToAll\_clicked();

void on\_pbSend\_clicked();

private:

Ui::Dialog \*ui;

QTcpSocket \*\_sok;

quint16 \_blockSize;

QString \_name;

void addToLog(QString text, QColor color = Qt::black);

};

Dialog.cpp:

#include "dialog.h"

#include "ui\_dialog.h"

#include <QtGui>

#include <QDebug>

Dialog::Dialog(QWidget \*parent) :QDialog(parent),ui(new Ui::Dialog)

{

ui->setupUi(this);

setWindowIcon(QIcon("favicon.PNG"));

\_name = "";

\_sok = new QTcpSocket(this);

connect(\_sok, SIGNAL(readyRead()), this, SLOT(onSokReadyRead()));

connect(\_sok, SIGNAL(connected()), this, SLOT(onSokConnected()));

connect(\_sok, SIGNAL(disconnected()), this, SLOT(onSokDisconnected()));

connect(\_sok, SIGNAL(error(QAbstractSocket::SocketError)),this, SLOT(onSokDisplayError(QAbstractSocket::SocketError)));

}

Dialog::~Dialog()

{

delete ui;

}

void Dialog::onSokDisplayError(QAbstractSocket::SocketError socketError)

{

switch (socketError) {

case QAbstractSocket::RemoteHostClosedError:

break;

case QAbstractSocket::HostNotFoundError:

QMessageBox::information(this, "Error", "The host was not found");

ui->pbConnect->setEnabled(true);

break;

case QAbstractSocket::ConnectionRefusedError:

QMessageBox::information(this, "Error", "The connection was refused by the peer.");

ui->pbConnect->setEnabled(true);

break;

default:

QMessageBox::information(this, "Error", "The following error occurred: "+\_sok->errorString());

ui->pbConnect->setEnabled(true);

break;

}

}

void Dialog::onSokReadyRead()

{

QDataStream in(\_sok);

if (\_blockSize == 0) {

if (\_sok->bytesAvailable() < (int)sizeof(quint16))

return;

in >> \_blockSize;

qDebug() << "\_blockSize now " << \_blockSize;

}

if (\_sok->bytesAvailable() < \_blockSize)

return;

else

\_blockSize = 0;

quint8 command;

in >> command;

qDebug() << "Received command " << command;

switch (command)

{

case MyClient::comAutchSuccess:

{

ui->pbSend->setEnabled(true);

addToLog("Enter as "+\_name,Qt::green);

}

break;

case MyClient::comUsersOnline:

{

addToLog("Received user list "+\_name,Qt::green);

ui->pbSend->setEnabled(true);

ui->cbToAll->setEnabled(true);

QString users;

in >> users;

if (users == "")

return;

QStringList l = users.split(",");

ui->lwUsers->addItems(l);

}

break;

case MyClient::comMessageToAll:

{

QString user;

in >> user;

QString message;

in >> message;

addToLog("["+user+"]: "+message);

}

break;

case MyClient::comMessageToUsers:

{

QString user;

in >> user;

QString message;

in >> message;

addToLog("["+user+"](private): "+message, Qt::blue);

}

break;

case MyClient::comPublicServerMessage:

{

QString message;

in >> message;

addToLog("[Server]: "+message, Qt::red);

}

break;

case MyClient::comPrivateServerMessage:

{

QString message;

in >> message;

addToLog("[Server](private) : "+message, Qt::darkMagenta);

}

break;

case MyClient::comUserJoin:

{

QString name;

in >> name;

ui->lwUsers->addItem(name);

addToLog(name+" joined", Qt::green);

}

break;

case MyClient::comUserLeft:

{

QString name;

in >> name;

for (int i = 0; i < ui->lwUsers->count(); ++i)

if (ui->lwUsers->item(i)->text() == name)

{

ui->lwUsers->takeItem(i);

addToLog(name+" left", Qt::green);

break;

}

}

break;

case MyClient::comErrNameInvalid:

{

QMessageBox::information(this, "Error", "This name is invalid.");

\_sok->disconnectFromHost();

}

break;

case MyClient::comErrNameUsed:

{

QMessageBox::information(this, "Error", "This name is already used.");

\_sok->disconnectFromHost();

}

break;

}

}

void Dialog::onSokConnected()

{

ui->pbConnect->setEnabled(false);

ui->leName->setEnabled(false);

ui->leHost->setEnabled(false);

ui->pbDisconnect->setEnabled(true);

\_blockSize = 0;

addToLog("Connected to"+\_sok->peerAddress().toString()+":"+QString::number(\_sok->peerPort()),Qt::green);

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << (quint8)MyClient::comAutchReq;

out << ui->leName->text();

\_name = ui->leName->text();

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

}

void Dialog::onSokDisconnected()

{

ui->pbDisconnect->setEnabled(false);

ui->pbSend->setEnabled(false);

ui->cbToAll->setEnabled(false);

ui->pbConnect->setEnabled(true);

ui->leName->setEnabled(true);

ui->leHost->setEnabled(true);

ui->lwUsers->clear();

addToLog("Disconnected from"+\_sok->peerAddress().toString()+":"+QString::number(\_sok->peerPort()), Qt::green);

}

void Dialog::on\_pbConnect\_clicked()

{

ui->pbConnect->setEnabled(false);

\_sok->connectToHost(ui->leHost->text(), 80);

}

void Dialog::on\_pbDisconnect\_clicked()

{

\_sok->disconnectFromHost();

}

void Dialog::on\_cbToAll\_clicked()

{

if (ui->cbToAll->isChecked())

ui->pbSend->setText("Send");

else

ui->pbSend->setText("Send To Selected");

}

void Dialog::on\_pbSend\_clicked()

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

if (ui->cbToAll->isChecked())

out << (quint8)MyClient::comMessageToAll;

else

{

out << (quint8)MyClient::comMessageToUsers;

QString s;

foreach (QListWidgetItem \*i, ui->lwUsers->selectedItems())

s += i->text()+",";

s.remove(s.length()-1, 1);

out << s;

}

out << ui->pteMessage->document()->toPlainText();

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

ui->pteMessage->clear();

}

void Dialog::addToLog(QString text, QColor color)

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+text);

ui->lwLog->item(0)->setTextColor(color);

}

**СЕРВЕРНАЯ ЧАСТЬ:**

Dialog.h:

#include <QDialog>

#include <QDebug>

#include <QtGui>

#include <QtCore>

#include "myserver.h"

#include "myclient.h"

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

friend class MyClient;

Q\_OBJECT

public:

explicit Dialog(QWidget \*parent = 0);

~Dialog();

private:

Ui::Dialog \*ui;

MyServer \*\_serv;

void addToLog(QString text, QColor color);

signals:

void messageFromGui(QString message, const QStringList &users);

public slots:

void onAddUserToGui(QString name);

void onRemoveUserFromGui(QString name);

void onMessageToGui(QString message, QString from, const QStringList &users);

void onAddLogToGui(QString string, QColor color);

private slots:

void on\_pbSend\_clicked();

void on\_cbToAll\_clicked();

void on\_pbStartStop\_toggled(bool checked);

};

Dialog.cpp:

#include "dialog.h"

#include "ui\_dialog.h"

#include <QMessageBox>

Dialog::Dialog(QWidget \*parent) :QDialog(parent), ui(new Ui::Dialog)

{

ui->setupUi(this);

setWindowIcon(QIcon("favicon.PNG"));

\_serv = new MyServer(this, this);

connect(this, SIGNAL(messageFromGui(QString,QStringList)), \_serv, SLOT(onMessageFromGui(QString,QStringList)));

connect(\_serv, SIGNAL(addLogToGui(QString,QColor)), this, SLOT(onAddLogToGui(QString,QColor)));

ui->pbStartStop->setChecked(true);

}

Dialog::~Dialog()

{

delete ui;

}

void Dialog::onAddUserToGui(QString name)

{

ui->lwUsers->addItem(name);

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+name+" joined");

ui->lwLog->item(0)->setTextColor(Qt::green);

}

void Dialog::onRemoveUserFromGui(QString name)

{

for (int i = 0; i < ui->lwUsers->count(); ++i)

if (ui->lwUsers->item(i)->text() == name)

{

ui->lwUsers->takeItem(i);

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+name+" left");

ui->lwLog->item(0)->setTextColor(Qt::green);

break;

}

}

void Dialog::onMessageToGui(QString message, QString from, const QStringList &users)

{

if (users.isEmpty())

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" message from "+from+": "+message+" to all");

else

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" message from "+from+": "+message+" to "+users.join(","));

ui->lwLog->item(0)->setTextColor(Qt::blue);

}

}

void Dialog::onAddLogToGui(QString string, QColor color)

{

addToLog(string, color);

}

void Dialog::on\_pbSend\_clicked()

{

QStringList l;

if (!ui->cbToAll->isChecked())

foreach (QListWidgetItem \*i, ui->lwUsers->selectedItems())

l << i->text();

emit messageFromGui(ui->pteMessage->document()->toPlainText(), l);

ui->pteMessage->clear();

if (l.isEmpty())

addToLog(" Sended public server message", Qt::black);

else

addToLog(" Sended private server message to "+l.join(","), Qt::black);

}

void Dialog::on\_cbToAll\_clicked()

{

if (ui->cbToAll->isChecked())

ui->pbSend->setText("Send");

else

ui->pbSend->setText("Send To Selected");

}

void Dialog::on\_pbStartStop\_toggled(bool checked)

{

if (checked)

{

QHostAddress addr;

if (!addr.setAddress(ui->leHost->text()))

{

addToLog(" invalid address "+ui->leHost->text(), Qt::red);

return;

}

if (\_serv->doStartServer(addr,80))

{

addToLog(" server started at "+ui->leHost->text()+":80", Qt::green);

ui->pbSend->setEnabled(true);

ui->cbToAll->setEnabled(true);

ui->pbStartStop->setText("Stop server");

}

else

{

addToLog(" server not started at "+ui->leHost->text()+":80", Qt::red);

ui->pbStartStop->setText("Reload");

ui->pbSend->setEnabled(false);

ui->cbToAll->setEnabled(false);

}

}

else

{

addToLog(" server stopped at "+\_serv->serverAddress().toString()+":80", Qt::green);

\_serv->close();

ui->pbSend->setEnabled(false);

ui->cbToAll->setEnabled(false);

ui->pbStartStop->setText("Start server");

}

}

void Dialog::addToLog(QString text, QColor color)

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+text);

ui->lwLog->item(0)->setTextColor(color);

}

Myclient.h:

#include <QObject>

#include <QDebug>

#include <QTcpSocket>

#include <QThreadPool>

#include <QtGui>

#include <QRegExp>

#include "dialog.h"

#include "myserver.h"

class MyServer;

class MyClient : public QObject

{

friend class MyServer;

Q\_OBJECT

public:

static const QString constNameUnknown;

static const quint8 comAutchReq = 1;

static const quint8 comUsersOnline = 2;

static const quint8 comUserJoin = 3;

static const quint8 comUserLeft = 4;

static const quint8 comMessageToAll = 5;

static const quint8 comMessageToUsers = 6;

static const quint8 comPublicServerMessage = 7;

static const quint8 comPrivateServerMessage = 8;

static const quint8 comAutchSuccess = 9;

static const quint8 comErrNameInvalid = 201;

static const quint8 comErrNameUsed = 202;

explicit MyClient(int desc, MyServer \*serv, QObject \*parent = 0);

~MyClient();

void setName(QString name) {\_name = name;}

QString getName() const {return \_name;}

bool getAutched() const {return \_isAutched;}

void doSendCommand(quint8 comm) const;

void doSendUsersOnline() const;

signals:

void addUserToGui(QString name);

void removeUserFromGui(QString name);

void removeUser(MyClient \*client);

void messageToGui(QString message, QString from, const QStringList &users);

private slots:

void onConnect();

void onDisconnect();

void onReadyRead();

void onError(QAbstractSocket::SocketError socketError) const;

private:

QTcpSocket \*\_sok;

MyServer \*\_serv;

quint16 \_blockSize;

QString \_name;

bool \_isAutched;

};

myclient.cpp:

#include "myclient.h"

#include <QWidget>

#include <QMessageBox>

const QString MyClient::constNameUnknown = QString(".Unknown");

MyClient::MyClient(int desc, MyServer \*serv, QObject \*parent) :QObject(parent)

{

\_serv = serv;

\_isAutched = false;

\_name = constNameUnknown;

\_blockSize = 0;

\_sok = new QTcpSocket(this);

\_sok->setSocketDescriptor(desc);

connect(\_sok, SIGNAL(connected()), this, SLOT(onConnect()));

connect(\_sok, SIGNAL(disconnected()), this, SLOT(onDisconnect()));

connect(\_sok, SIGNAL(readyRead()), this, SLOT(onReadyRead()));

connect(\_sok, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(onError(QAbstractSocket::SocketError)));

qDebug() << "Client connected" << desc;

}

MyClient::~MyClient()

{

}

void MyClient::onConnect()

{

}

void MyClient::onDisconnect()

{

qDebug() << "Client disconnected";

if (\_isAutched)

{

emit removeUserFromGui(\_name);

\_serv->doSendToAllUserLeft(\_name);

emit removeUser(this);

}

deleteLater();

}

void MyClient::onError(QAbstractSocket::SocketError socketError) const

{

QWidget w;

switch (socketError) {

case QAbstractSocket::RemoteHostClosedError:

break;

case QAbstractSocket::HostNotFoundError:

QMessageBox::information(&w, "Error", "The host was not found");

break;

case QAbstractSocket::ConnectionRefusedError:

QMessageBox::information(&w, "Error", "The connection was refused by the peer.");

break;

default:

QMessageBox::information(&w, "Error", "The following error occurred: "+\_sok->errorString());

}

}

void MyClient::onReadyRead()

{

QDataStream in(\_sok);

if (\_blockSize == 0) {

if (\_sok->bytesAvailable() < (int)sizeof(quint16))

return;

in >> \_blockSize;

qDebug() << "\_blockSize now " << \_blockSize;

}

if (\_sok->bytesAvailable() < \_blockSize)

return;

else

\_blockSize = 0;

quint8 command;

in >> command;

qDebug() << "Received command " << command;

if (!\_isAutched && command != comAutchReq)

return;

switch(command)

{

case comAutchReq:

{

QString name;

in >> name;

if (!\_serv->isNameValid(name))

{

doSendCommand(comErrNameInvalid);

return;

}

if (\_serv->isNameUsed(name))

{

doSendCommand(comErrNameUsed);

return;

}

\_name = name;

\_isAutched = true;

doSendUsersOnline();

emit addUserToGui(name);

\_serv->doSendToAllUserJoin(\_name);

}

break;

case comMessageToAll:

{

QString message;

in >> message;

\_serv->doSendToAllMessage(message, \_name);

emit messageToGui(message, \_name, QStringList());

}

break;

case comMessageToUsers:

{

QString users\_in;

in >> users\_in;

QString message;

in >> message;

QStringList users = users\_in.split(",");

\_serv->doSendMessageToUsers(message, users, \_name);

emit messageToGui(message, \_name, users);

}

break;

}

}

void MyClient::doSendCommand(quint8 comm) const

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << comm;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

qDebug() << "Send to" << \_name << "command:" << comm;

}

void MyClient::doSendUsersOnline() const

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << comUsersOnline;

QStringList l = \_serv->getUsersOnline();

QString s;

for (int i = 0; i < l.length(); ++i)

if (l.at(i) != \_name)

s += l.at(i)+(QString)",";

s.remove(s.length()-1, 1);

out << s;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

qDebug() << "Send user list to" << \_name << ":" << s;

}

Myserver.h

#include <QTcpServer>

#include <QDebug>

#include "dialog.h"

#include "myclient.h"

class MyClient;

class MyServer : public QTcpServer

{

Q\_OBJECT

public:

explicit MyServer(QWidget \*widget = 0, QObject \*parent = 0);

bool doStartServer(QHostAddress addr, qint16 port);

void doSendToAllUserJoin(QString name);

void doSendToAllUserLeft(QString name);

void doSendToAllMessage(QString message, QString fromUsername);

void doSendToAllServerMessage(QString message); void doSendServerMessageToUsers(QString message, const QStringList &users);

void doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername);

QStringList getUsersOnline() const;

bool isNameValid(QString name) const;

bool isNameUsed(QString name) const;

signals:

void addLogToGui(QString string, QColor color = Qt::black);

void addUserToGui(QString);

void removeUserFromGui(QString);

public slots:

void onMessageFromGui(QString message, const QStringList &users);

void onRemoveUser(MyClient \*client);

protected:

void incomingConnection(qintptr handle);

private:

QList<MyClient \*> \_clients;

QWidget \*\_widget;

};

Myserver.cpp

#include "myserver.h"

MyServer::MyServer(QWidget \*widget, QObject \*parent) :QTcpServer(parent)

{

\_widget = widget;

}

bool MyServer::doStartServer(QHostAddress addr, qint16 port)

{

if (!listen(addr, port))

{

qDebug() << "Server not started at" << addr << ":" << port;

return false;

}

qDebug() << "Server started at" << addr << ":" << port;

return true;

}

void MyServer::doSendToAllUserJoin(QString name)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comUserJoin << name;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() != name && \_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllUserLeft(QString name)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comUserLeft << name;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() != name && \_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllMessage(QString message, QString fromUsername)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comMessageToAll << fromUsername << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername)

{

QByteArray block, blockToSender;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comMessageToUsers << fromUsername << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

QDataStream outToSender(&blockToSender, QIODevice::WriteOnly);

outToSender << (quint16)0 << MyClient::comMessageToUsers << users.join(",") << message;

outToSender.device()->seek(0);

outToSender << (quint16)(blockToSender.size() - sizeof(quint16));

for (int j = 0; j < \_clients.length(); ++j)

if (users.contains(\_clients.at(j)->getName()))

\_clients.at(j)->\_sok->write(block);

else if (\_clients.at(j)->getName() == fromUsername)

\_clients.at(j)->\_sok->write(blockToSender);

}

void MyServer::doSendToAllServerMessage(QString message)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comPublicServerMessage << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendServerMessageToUsers(QString message, const QStringList &users)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comPrivateServerMessage << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int j = 0; j < \_clients.length(); ++j)

if (users.contains(\_clients.at(j)->getName()))

\_clients.at(j)->\_sok->write(block);

}

QStringList MyServer::getUsersOnline() const

{

QStringList l;

foreach (MyClient \* c, \_clients)

if (c->getAutched())

l << c->getName();

return l;

}

bool MyServer::isNameValid(QString name) const

{

if (name.length() > 20 || name.length() < 4)

return false;

QRegExp r("[A-Za-z0-9\_]+");

return r.exactMatch(name);

}

bool MyServer::isNameUsed(QString name) const

{

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() == name)

return true;

return false;

}

void MyServer::incomingConnection(qintptr handle)

{

MyClient \*client = new MyClient(handle, this, this);

if (\_widget != 0)

{

connect(client, SIGNAL(addUserToGui(QString)), \_widget, SLOT(onAddUserToGui(QString)));

connect(client, SIGNAL(removeUserFromGui(QString)), \_widget, SLOT(onRemoveUserFromGui(QString)));

connect(client, SIGNAL(messageToGui(QString,QString,QStringList)), \_widget, SLOT(onMessageToGui(QString,QString,QStringList)));

}

connect(client, SIGNAL(removeUser(MyClient\*)), this, SLOT(onRemoveUser(MyClient\*)));

\_clients.append(client);

}

void MyServer::onRemoveUser(MyClient \*client)

{

\_clients.removeAt(\_clients.indexOf(client));

}

void MyServer::onMessageFromGui(QString message, const QStringList &users)

{

if (users.isEmpty())

doSendToAllServerMessage(message);

else

doSendServerMessageToUsers(message, users);

}

**5. Пример работы**

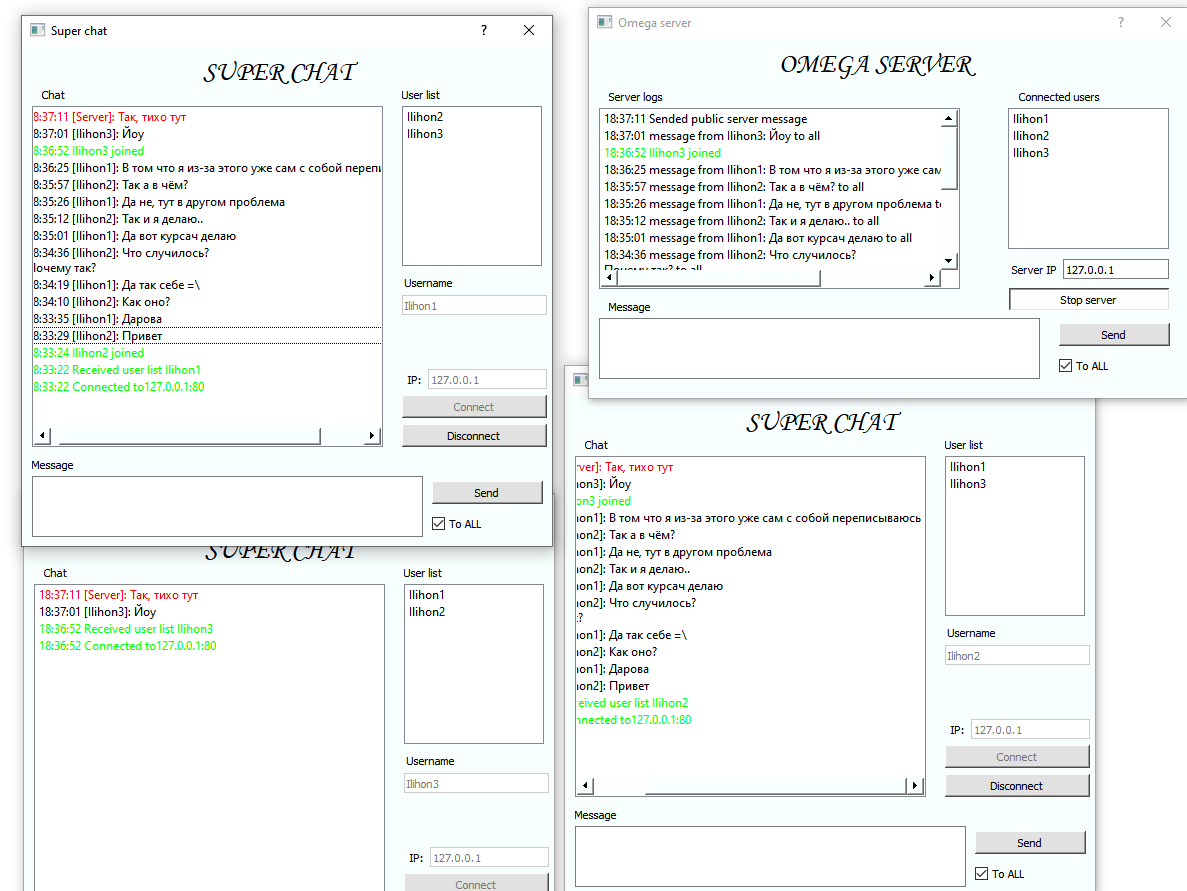


Рисунок 1. Пример работы чата.

# 5. Сопроводительная документация программного обеспечения

## 5.1. Разработка описания программы

Общее описание:

«Данный программный продукт предназначен для обмена информацией по локальной или беспроводной сети. Сервер запускается как сетевая служба и работает на компьютере в фоновом режиме. Клиентское приложение не требует никаких настроек».

## 5.2. Разработка руководства пользователя

1. Запустить программу с помощью консоли QT

Для запуска серверного и клиентского приложения необходимым и достаточным условием являются следующие требования:

* Операционная система: Windows 7 или выше;
* Объём оперативной памяти: 1,00 ГБ;
* Свободное место на диске: 1,00 МБ;
* Наличие сетевой карты.

## 5.3. Анализ программного обеспечения

Анализ исходного кода проводился с помощью встроенной утилиты в среду разработки

Qt Creator 4.5.0.

Результат анализа исходного кода:

* Предупреждений не обнаружено
* Критических ошибок в коде не обнаружено.

# 6. Заключение

На основе полученных знаний по курсу «Операционные системы» мне в результате удалось выполнить курсовую работу по теме «Многопользовательский сетевой чат на основе сокетов», было разработано приложение-клиент на основе QTcpSocket, которое имело графический интерфейс, написанный с помощью QtWidget.

Данное приложение позволяет удалённо обмениваться информацией и данными по локальной или беспроводной сети.

Считаю, что мне удалось выполнить цель, ставившуюся перед началом создания программного продукта.

# 7. Список использованных источников

1. Курс лекций Малкова Е.А. «Операционные системы».

2. Организации взаимодействия между Клиентом и Сервером с помощью протоколов TCP, UDP <http://www.intuit.ru/department/os/osintropractice/10/1.html>

3. Происхождение чата, http://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%82