МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

* + 1. федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

(ГУАП)

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И СЕТЕЙ

КУРСОВАЯ РАБОТА

ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. техн. наук |  |  |  | К.А. Спесивцева |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ПРОГРАММА  **Месенджер**  ОП 44.4843.19 ПЗ |
| по дисциплине: Основы программирования |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4843 |  |  |  | А.И. Чернявский  А.А. Петухов  Н.Л. Симонов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. Постановка задачи 6

2. Системные требования 7

3. Описание программы 8

3.1 Серверная часть 8

3.2 Клиентская часть 9

4. Тестирование приложения 11

ЗАКЛЮЧЕНИЕ 17

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 18

ПРИЛОЖЕНИЕ 19

ПРИЛОЖЕНИЕ А. Сервер 19

ПРИЛОЖЕНИЕ Б. Клиент 22

# **ВВЕДЕНИЕ**

На протяжении всей истории человечества люди постоянно испытывают потребность в общении. Наверное, это происходит потому, что именно через общение человеку легче понять другого человека, а также познать самого себя. Общение – сложный процесс взаимодействия между людьми, заключающийся в обмене информацией, а также в восприятии и понимании партнерами друг друга.

Казалось бы, совсем недавно люди общались на расстоянии друг с другом с помощь обычных писем. Но человек хотел чего-то большего, совершенного, быстро передаваемого и быстро принимаемого, простого в использовании и доступного. Настоящим взрывом в обществе стало изобретение в середине 19 века проводного телефона, который мог передавать и принимать звук на расстоянии. На телефонные аппараты смотрели как на сказочное устройство, с помощью которого, можно было поговорить с человеком на другом конце земли. Изобретение первых персональных компьютеров, в конце 20 века, послужило мощным толчком в развитии компьютерной техники. Однако же, прошло еще не мало времени, после изобретения первых ЭВМ, прежде чем люди научились общаться на расстоянии, используя свои недавние творения. С появлением всемирной системы объединенных компьютерных сетей стало понятно, что оперативно общаться на расстоянии можно и очень удобно. Конечно же, и сегодня есть люди, использующие «проверенную» веками систему обмена данными, но это уже скорее выглядит как исключение из правил.

В современном мире люди все больше втягиваются в сферу общения посредством электронных коммуникаций, таких как: электронная почта, чат, видеосвязь. На сегодняшний день очень просто связаться с человеком на другом континенте в любом месте и в любое время; для этого необходимы только интернет-связь и хотя бы телефон. Общение людей стало доступнее. С развитием интернет технологий постепенно происходило и развитие электронных коммуникаций. Если вначале развития электронной связи, людям достаточно было общаться, используя электронную почту, то уже несколько лет спустя этого оказалось мало. И следующими этапами в развитии связи стали чат и видеосвязь.

В последние десятилетия произошел стремительный скачок в развитии новых коммуникационных технологий. В современную жизнь прочно вошли электронная почта, интернет, цифровое телевидение. На сегодняшний день практически каждый человек пользуется тем или иным видом электронной связи, будь он в школе, на работе или дома. Коммуникация перерастает из односторонней в интерактивную, что открывает новые возможности в ее применении и представляет ее в новом свете. Поэтому данная тема актуальна в настоящее время.

Поскольку в чате, отличии от форума, нельзя оставить сообщение «впрок», чат фактически существует, только если в нем в некоторый момент времени встретились хотя бы два человека. Как правило, чат – это молодежная тусовка, а не инструмент для дискуссии. В большинстве случае пользователь относится к безопасности в сети достаточно безалаберно, считая, что им нечего скрывать и «а что тут такого секретного». На самом деле любая переписка – передача информации. Важность этой информации быть в большей или меньшей степени значима. Значимость информации определяется в первую очередь возможностью извлечь выгоду, использовав полученные данные.

Данная работа предназначена для закрепления учебного материала, изученного по курсу «Основы программирования», а также для других значительно более важных задач.

Существуют много различных чатов, наиболее популярные из них это Вконтакте, Facebook, Whatsup, Телеграмм, Mail.ru, Одноклассники и другие. Мы решили сделать свой чат и назвать «Месенджер».

Целью курсового проекта является приобретение и закрепление навыков организации вычислительных процессов и программирования на алгоритмическом языке для решения конкретных поставленных задач.

Так же, данная работа предназначена для повышения навыков по оформлению документации на программные средства.

1. **Постановка задачи**

Чат – это общение в интернете, когда разговор ведется в реальном времени. Содержание чата существует только «здесь и сейчас» - чат выглядит как окно, в котором идет поток сообщений от всех участников чата. Для того, чтобы общаться в чате, нужно просо выбрать себе имя, под которым вы «появитесь» в нем и послать в чат сообщение, которое сразу же появится в общем потоке.

Архивы чатов в чистом виде, как правило, не имеют смысла поскольку текст выглядит сплошной мешаниной, в которой очень трудно понять, кто, кому и какое сообщение послал.

Целью данной работы является разработка чата «Месенджер». Чат состоит из двух частей: клиент и сервер. Клиент отправляет запрос на сервер, в это время сервер принимает запросы, обрабатывает и отправляет клиенту.

Эта программа предназначена для обмена информацией с другим пользователем, в некоторых случаях предназначена для хранения временных записей.

С помощью данной программы пользователь может отправлять текстовую информацию другому пользователю. Программа должна быть реализована в стиле «Cyber punk» и не должна выводить внутренних ошибок при использовании. Благодаря этой программе люди смогут обмениваться сообщениями. Так же этой программой удобно пользоваться ночью т.к. в нашем чате реализован стиль «Cyber punk», в котором мало ярких резких цветов, которые могут негативно сказаться на сетчатке глаза, что может привести к ухудшению зрения.

1. **Системные требования**

Сист. требования:

Операционная система: Mac, Windows или Linux

Процессор: I686 совместимый процессор

Доступ к сети интернет

Роутинг, открытые порты.

Средством разработки мы выбрали QT, т.к. оно является очень удобным API, исходя из того, что оно содержит достаточное количества классов для большинства повседневных задач. Так же преимуществами является простота разработки графического пользовательского интерфейса (GUI), кроссплатформенность, удобное межпроцессорное взаимодействие (слоты и сигналы). Большим плюсом является хорошая и подробная документация, которая превосходит примеры и документацию многих таких же средств – например Builder или Delphi.

1. **Описание программы**

**3.1 Серверная часть**

При запуске сервера открывается окно консоли. Создаётся объект класса QTcpServer. Вызывается метод класса startServer, который определяет автоматически значения порта в интервале [49152,65535], т.к. это порты динамически выделяемые или частные порты, которые не регистрируются. Используются временными(короткоживущими) соединениями “сервер-клиент” или в определённых частных случаях. Полученный порт записывается в переменную объекта. Если этот порт прослушивается, тогда в консоль выводится надпись “Listening” вместе с новой строчкой, которая содержит следующее: “Server’s Port = “ и полученный порт. Далее сервер ожидает подключения со стороны клиентов. По умолчанию максимальное количество подключений – 4. Создаётся UDP сокет, с любым адресом и полученным портом. Создаётся межпроцессорное взаимодействие “сигнал – вызов слота” – Если сокет readyRead(Сокет готов к чтению) , тогда вызывается слот sockReady, в котором если сокет принимает входящие датаграммы(сообщения от клиента), тогда программа обрабатывает входящий поток массива байтов. Если это сообщение содержит вид “\n”, тогда сервер записывает данные клиента: порт и IP в массив и выводит сообщение “Client \*номер клиента\* connected to server☺”. Иначе если это сообщение содержит “Code:1234”, тогда сервер пишет, что клиент отключился, т.е. – “Client disconnected☹” и удаляет порт и IP из массива данных о подключённых клиентах. Иначе если это сообщение содержит ни, то ни другое, тогда сервер принимает датаграмму, записывает его в буффер и отправляет всем активным клиентам, подключенным к серверу.

* 1. **Клиентская часть**

При запуске клиента открывается окно, которое содержит кнопки, надписи, поля для ввода, заголовок окна. У пользователя есть возможность ввести порт в поле, находящиеся под надписью “Enter port:”, и ник в поле под надписью “Your nick:”. После того как оба поля будут с введёнными значениями кнопка с надписью CONNECT TO SERVER будет доступна к нажатию (переключит режим enabled). Если порт был неверен, тогда кнопка будет недоступна для нажатий и пользователю придётся снова ввести порт и ник. При нажатии на кнопку Settings выскакивает поле ввода, надпись и кнопка “OK”. Надпись содержит – “IP”. То есть в случае необходимости можно изменить адрес, к которому пользователь хочет подключиться. По умолчанию он стоит локальный - “127.0.0.1”. После ввода нового IP и нажатия кнопки OK, клиент запоминает в переменной типа QHostAddress значение этого IP. При следующем нажатии на кнопку CONNECT TO SERVER будет уже использоваться новый IP. Также присутствует кнопка выхода из программы Exit, которая всегда активна вплоть до выключения клиентского приложения. Так же доступны горячие клавиши, такие как Escape – при нажатии на эту кнопку программа закрывается и Enter – при нажатии на эту кнопку, кнопка CONNECT TO SERVER активируется. После удачного подключения (Когда клиент при правильном IP и порте подключается к серверу, он посылает ему датаграмму со значением “\n”, чтобы сервер смог записать его в массиве данных об клиентах) скрываются кнопки подключения, настроек, полей ввода для IP и порта и надписей. Появляются новые кнопки отправления сообщений, поле ввода сообщений и недоступное для ввода поле с содержимым новых сообщений. В верхнем недоступном поле ввода появляется надпись, которая гласит – “You are connected. Welcome☺” (Она появляется при том случае, когда сервер отправляет ответную датаграмму, на ту которую клиент отправил при подключении к серверу, имеющую идентичное содержание). В нижнем поле ввода пользователь может ввести сообщение, которое хочет отправить другим клиентам, подключённым к серверу. По нажатию кнопки ENTER сообщение из нижнего поля ввода отправляется в виде датаграммы серверу, сервер принимает это сообщение и отправляет всем клиентам, подключенным к сетевому чату в данный момент. При принятии этого сообщения клиент обрабатывает данную датаграмму и выписывает её значение в виде строки в буфер, а после и записывает в верхнее недоступное для ввода поле. Так же предусмотрена горячая клавиша при вводе сообщения в нижнее поля ввода при нажатии на сочетание клавиш Alt+Enter сообщение отправляется на сервер. В течение всего это времени кнопка Exit и её горячая клавиша Escape активны всё время и доступны для нажатия. При выходе из клиентского приложения, перед самым выходом клиент отправляет серверу сообщения “Code:1234”, которое говорит серверу , что клиент отключился.

**4. Тестирование приложения**

Слот sockReady в серверном приложении, реализующий принятие и отправку сообщений рисунок 1



Рисунок 1 –sockReady в серверном приложении

Если сокет готов к чтению, тогда вызывается этот слот.

В строчке 24 проверяется имеет ли сокет не прочитанные датаграммы.

В строчках 26-30 создаётся буфер, который меняет свой размер в зависимости от размера полученной датаграммы (то есть её размер), читается датаграмма – в буфер записывается значение датаграммы, её размер и в отдельные переменные значения порта и IP.

В строчках 31-37 идёт проверка на содержимое буфера, если имеет заранее опредёлённое значение “\n”, тогда IP и порт записываются в массивы , в консоль выводится сообщение о том, что клиент подключился. Сервер отправляет по данному IP и порту сообщение из буфера, тоесть “\n”. Увеличивается количество активных клиентов в переменной i.

В строчках с 39-57 проверяется , если буфер имеет значение “Code:1234”, тогда в консоль выводится сообщение, что клиент отключился и после с помощью цикла находится в массиве значения порта и IP этого клиента и удаляется из массива.

В строчках 59-60 полученная датаграмма отправляется всем клиентам из списка подключённых клиентов.

Аналогичная функция SockReady в клиентском приложении, реализующая принятии сообщений и смены окна на рисунке 2



Рисунок 2 – SockReady в клиентском

В строчках 28-32 происходит всё так же, как и на серверном приложении, значения порта, IP, размер сообщения и самое сообщение записывается в определённые переменные.

В строчках 33-50 проверяется, является ли сообщение заранее определённым сообщением – “\n”. Если оно таковое, тогда скрываются кнопки, поля ввода, надписи и появляются необходимые для обмена сообщениями (Пропадает кнопка подключения, поля для ввода порта и ника, настройки. Появляется два окна ввода, одно для ввода сообщений, второе недоступное для вывода сообщений и так же появляется кнопка отправления сообщений). И устанавливаются горячие клавиши для кнопок отправления сообщения и выхода из программы.

В строчке 53 реализуется вставка полученных сообщений в верхнее поле недоступное для ввода .

Функции нажатия на клавишу и закрытия приложения рисунок 3



Рисунок 3 – Функции нажатия на клавишу и закрытия приложения

Первая функция keyPressEvent вызывается при активном окне, когда была нажата любая клавиша.

В строчках 124-129 проверяется является ли нажатая клавиша клавишей Escape, если да – тогда клиент отправляет сообщение серверу “Code:1234”, которое сервер воспринимает как отключение клиента от него. Меняется флаг выхода, чтобы closeEvent не отправил сообщение о отключении клиента второй раз. И клиентское приложение закрывается.

В строчках 131-139 проверяет является ли клавиша = Enter. Если да, тогда у кнопки меняется иконка, в строковую переменную texts записывается ник клиента, который отправляет сообщение и “:”, и сообщение из поля ввода , и “\n”, чтобы сообщение переносилось на новую строку. Полученная строка преобразуется в переменной data в массив байтов и отправляется датаграммой на сервер. Очищается поле ввода и кнопка отправки сообщений становится недоступной.

Вторая функция closeEvent активируется при закрытии окна.

В строчках 144-149 проверяется не был ли выход из программы уже активирован , если нет, тогда событие закрытия клиентского приложения игнорируется на время и серверу отправляется сообщение о том ,что клиент отключился. После событие снова активируется и приложение закрывается.

Заходим на сервер, в консоли автоматически появляется порт сервера рисунок 4

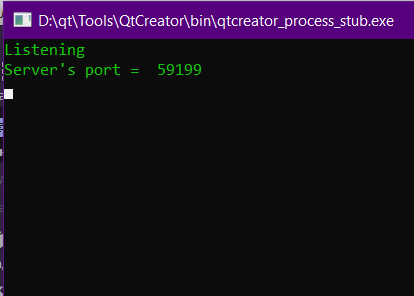


Рисунок 4 – Сервер в консоли

Заходим в клиент и появляется окно которое изображено на Рис.5

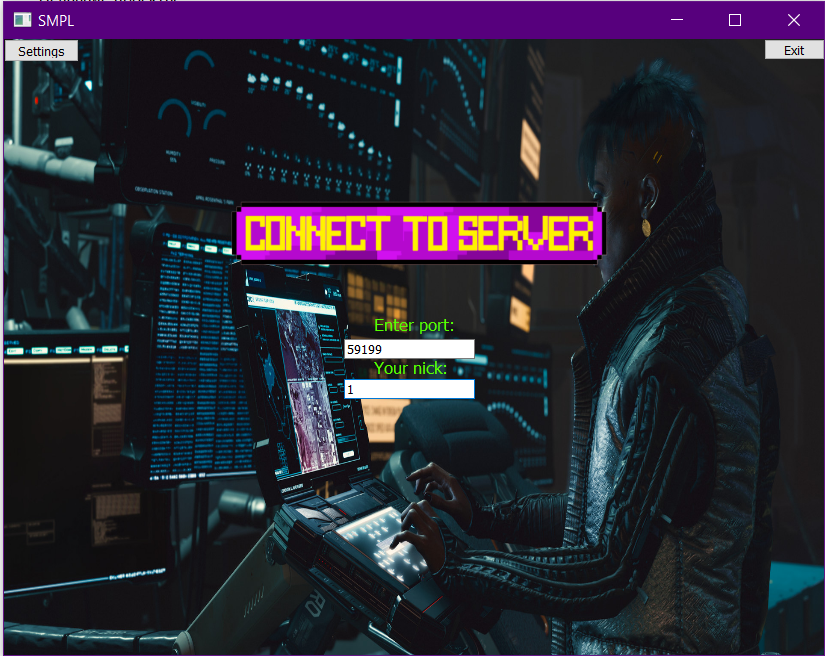


Рисунок 5 – Клиент интерфейс в стиле киберпанк

Далее мы вводим название порта и имя пользователя. Тоже самое делаем и для второго клиента на рисунке 6

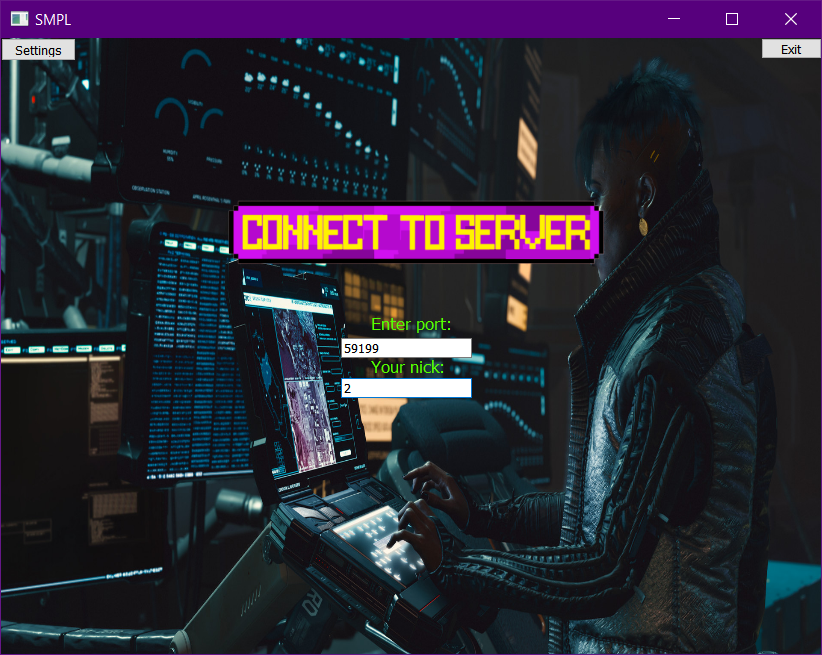


Рисунок 6 – клиент 2

На сервере в консоли можно увидеть присоединение двух клиентов на рисунке 7. И теперь можно отправлять сообщения другому клиенту.

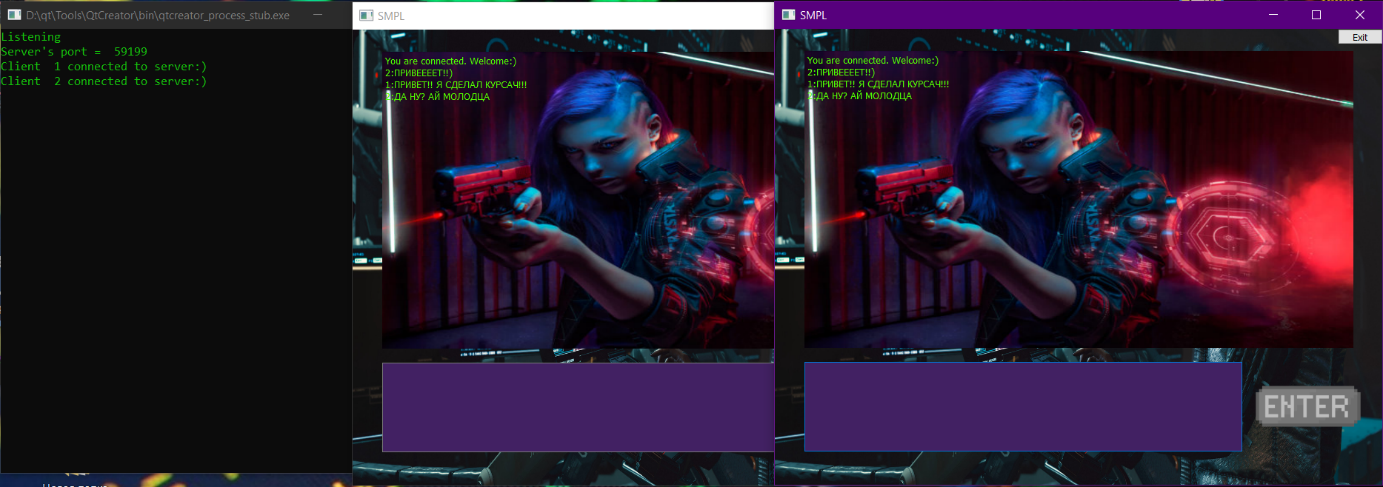


Рисунок 7 – чат готов к использованию

Далее когда выходим из клиента можно увидеть, какие клиенты вышли на сервере рисунок 8

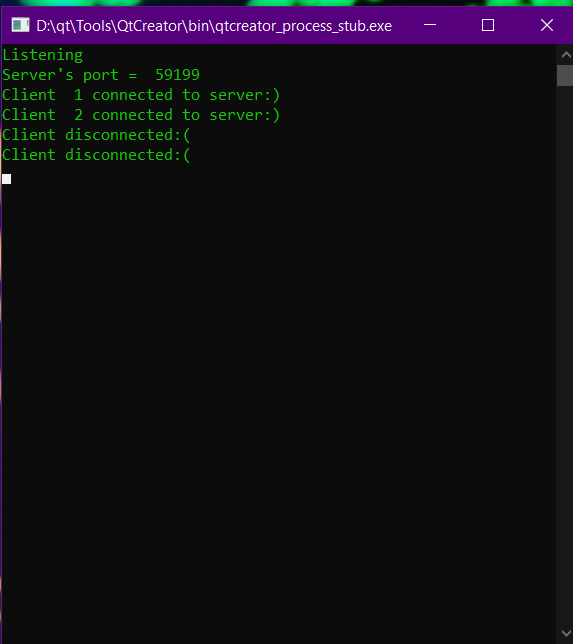


Рисунок 8 - Выход клиентов

**ЗАКЛЮЧЕНИЕ**

В данной курсовой работе был реализован чат «Месенджер».

В процессе работы над курсовым проектом были закреплены навыки программирования на языке С++.

Был получен базовый комплекс знаний по документированию программного обеспечения, согласно принятым государственным стандартам.

Таким образом, можно считать, что цель работы достигнута, а задачи, поставленные в начале – решены.

В перспективе возможны доработка и развитие созданного чата путем внедрения новых возможностей.

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Marienko L. - Язык программирования С++, 2010-2020. - URL: http://cppstudio.com/ (дата обращения 2019-11-09).
2. Документация QT Creator / Документация QT Creator сокеты, 2003-2020. - URL: https://doc.qt.io/qt-5/qsslsocket.html (дата обращения 2019-12-15).

**ПРИЛОЖЕНИЕ**

В документе приводится текст программы «Месенджер», написанной на языке C++, а так же результаты отладки и тестирования.

В процессе разработки мы использовали QtCreator. Чтобы появилась программа, на которой люди смогут отправлять информацию друг другу, мы написали две части клиент и сервер.

**ПРИЛОЖЕНИЕ СЕРВЕР**

#include <myserver.h>

#include <QDataStream>

myserver::**myserver**(){}

myserver::~***myserver***(){}

void myserver::**startServer**() //start server encrypted

{

serverPort = autoPort(); // move to port value of function autoPort

if(this->listen(QHostAddress::Any,serverPort)) // if server listening at current ip and port

{

qDebug()<<"Listening";

qDebug()<<"Server's port = " << serverPort;

}

else // if server not listening

{

qDebug()<<"Not Listening";

}

this->setMaxPendingConnections(4); // set limit to connection to host

socket = new QUdpSocket(this);

socket->bind(QHostAddress::Any,serverPort);

connect(socket,SIGNAL(readyRead()),this,SLOT(sockReady()));

}

void myserver :: **sockReady**() // function if socket is ready

{

if(socket->hasPendingDatagrams())

{

QByteArray buffer;

buffer.resize(socket->pendingDatagramSize());

QHostAddress sender;

quint16 senderPort;

socket->readDatagram(buffer.data(),buffer.size(),&sender,&senderPort);

if(buffer == "\n")

{

Clients[i] = sender;

port[i] = senderPort;

qDebug() << "Client " << i+1 << "connected to server:)" ;

socket->writeDatagram(buffer,1,Clients[i],port[i]);

i++;

}

else

if(buffer == "Code:1234")

{

qDebug() << "Client disconnected:(";

for(int k=0;k<i;k++)

{

if(Clients[k]==sender && port[k]==senderPort )

{

while(k<i)

{

Clients[k]=Clients[k+1];

port[k]=port[k+1];

k+=2;

}

i--;

break;

}

}

}

else

for(int k=0;k<i;k++)

socket->writeDatagram(buffer,Clients[k],port[k]);

}

}

quint16 myserver::**autoPort**() // function to autoport in interval 49152-65535

{

std::random\_device rd;

std::mt19937 e{rd()};

std::uniform\_int\_distribution<uint16\_t> dist{49152, 65535};

return(dist(e));

}

**ПРИЛОЖЕНИЕ КЛИЕНТ**

#include "smpl.h"

#include "ui\_smpl.h"

SMPL::**SMPL**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::SMPL) // inicilization of UI

{

ui->setupUi(this); // setting ui to current window

ui->label\_4->hide();

ui->lineEdit\_3->hide();

ui->pushButton\_5->hide();

ui->textEdit->hide(); // hide textEdit

ui->textEdit\_2->hide(); // hide textEdit\_2

ui->pushButton\_2->hide(); // hide pushButton\_2

socket = new QUdpSocket(this); // creating socket

connect(socket,SIGNAL(readyRead()),this,SLOT(SockReady()));

socket->bind(HostIP,port);

SMPL::ui->pushButton\_2->setIcon(QIcon(":/resources/img/enter.tuch.png"));//icon enter

SMPL::ui->pushButton->setIcon(QIcon(":resources/img/connect1.png"));//icon enter

ui->pushButton->setShortcut(Qt::Key\_Return);

}

SMPL::~***SMPL***() // destructor

{

delete ui;

}

void SMPL :: **SockReady**() // function if socket is ready

{

QByteArray buffer;

buffer.resize(socket->pendingDatagramSize());

QHostAddress sender;

quint16 senderPort;

socket->readDatagram(buffer.data(),buffer.size(),&sender,&senderPort);

if(buffer=="\n")

{

ui->pushButton->hide();// here hiding and showing some ui elements

ui->lineEdit->hide();

ui->label->hide();

ui->label\_2->hide();

ui->lineEdit\_2->hide();

ui->pushButton\_2->show();

ui->textEdit->show();

ui->textEdit\_2->show();

ui->textEdit->isReadOnly();

ui->textEdit->insertPlainText("You are connected. Welcome:)\n");

ui->label\_4->hide();

ui->lineEdit\_3->hide();

ui->pushButton\_5->hide();

ui->pushButton\_3->hide();

ui->pushButton\_2->setShortcut(Qt::Key\_Return);

ui->pushButton\_2->setShortcut(Qt::Key\_Enter);

}

else

ui->textEdit->insertPlainText(buffer.data());

}

void SMPL::**on\_pushButton\_clicked**() // if pushButton clicked function

{

Nick = ui->lineEdit\_2->text(); // move text from lineEdit\_2 to Nick

port = ui->lineEdit->text().toInt();

ui->lineEdit->clear();

ui->lineEdit\_2->clear();

ui->pushButton->setEnabled(false); // off the button(unclickable)

socket->writeDatagram("\n",1,HostIP,port);

}

void SMPL::**on\_lineEdit\_2\_textChanged**(const QString &arg1) //if text on lineEdit\_2 changed function

{

ui->pushButton->setEnabled(true);// make the button clickable

}

void SMPL::**on\_textEdit\_2\_textChanged**() // if text on textiEdit\_2 changed

{

if(ui->textEdit\_2->toPlainText() != "");

ui->pushButton\_2->setEnabled(true); // make the pushButton\_2 clickable

}

void SMPL::**on\_pushButton\_2\_clicked**()

{

SMPL::ui->pushButton\_2->setIcon(QIcon(":/resources/img/enter.tuch.png")); //icon enter

QByteArray data;

texts = Nick + ":" + ui->textEdit\_2->toPlainText() + "\n"; // forming the message with format : Nick:message , message taked from textEdit\_2

data.append(texts);

socket->writeDatagram(data,HostIP,port);

ui->textEdit\_2->clear();

ui->pushButton\_2->setEnabled(false);

}

void SMPL::**on\_pushButton\_3\_clicked**()

{

if (flag!=1)

{

ui->label\_4->show();

ui->lineEdit\_3->show();

ui->pushButton\_5->show();

flag=1;

}

else

{

ui->label\_4->hide();

ui->lineEdit\_3->hide();

ui->pushButton\_5->hide();

flag=0;

}

}

void SMPL::**on\_pushButton\_4\_clicked**()

{

SMPL::close();

}

void SMPL::**on\_pushButton\_5\_clicked**()

{

if(ui->lineEdit\_3->text()!="")

{

HostIP=QHostAddress(ui->lineEdit\_3->text());

}

ui->label\_4->hide();

ui->lineEdit\_3->hide();

ui->pushButton\_5->hide();

flag=0;

}

void SMPL::***keyPressEvent***(QKeyEvent \*event)

{

if(event->key() == Qt::Key\_Escape)

{

socket->writeDatagram("Code:1234",9,HostIP,port);

flagQuit=1;

SMPL::close();

}

else

if(event->key() == Qt::Key\_Enter || event->key() == Qt::Key\_Return)

{

SMPL::ui->pushButton\_2->setIcon(QIcon(":/resources/img/enter.tuch.png")); //icon enter

QByteArray data;

texts = Nick + ":" + ui->textEdit\_2->toPlainText() + "\n"; // forming the message with format : Nick:message , message taked from textEdit\_2

data.append(texts);

socket->writeDatagram(data,HostIP,port);

ui->textEdit\_2->clear();

ui->pushButton\_2->setEnabled(false);

}

}

void SMPL::***closeEvent***(QCloseEvent \*event)

{

event->ignore();

if(flagQuit!=1)

{

socket->writeDatagram("Code:1234",9,HostIP,port);

}

event->accept();

}