

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
по курсовому проекту
на тему:

«Чат система на ОС Windows»

БГУИР КР 1-40 02 01 116 ПЗ

Выполнила:
ст. гр. 550501
Момотова Ю.О.

Руководитель:
Лавникевич Д.А.

Минск 2017

ЗАДАНИЕ

по курсовому проектированию

Студенту Момотовой Юлии Олеговне

1. Тема проекта:

Разработать клиент-серверное приложение «Чат система на ОС Windows» средствами языка C++.

2. Срок сдачи студентом законченного проекта:

10.06.2017 г.

3. Решаемые задачи и функционал разрабатываемого ПО:

Нужно разработать приложение, соответствующее следующим требованиям:

- возможность отправки сообщений
- возможность отправки файлов (музыки, картинок, документов и т.д.);
- хранение истории

Должна быть предусмотрена возможность отправки сообщений как всем пользователям, находящимся в сети, так и только одному, т.е. реализована возможность личной переписки.

Для файлов должен быть предусмотрен тот же функционал, что и для обычных сообщений.

Файлы, вместе с информацией о них, должны загружаются на сервер и там хранятся. Клиенты в любой момент времени должны иметь возможность загрузить себе любой из файлов, к которым у них есть доступ.

Отправка информации осуществляется только клиентам, находящимся в данный момент в сети (подключены к серверу).

На стороне клиента должно быть предусмотрено хранение истории всех переписок и имён отправленных файлов

На стороне сервера должна быть предусмотрено хранение имён всех пользователей и информации о всех загруженных файлов.

Также приложение должно иметь удобный графический интерфейс.

4. Средства разработки:

Язык программирования C++, Qt Creator 5.8, ОС Windows.

РУКОВОДИТЕЛЬ _____ *Д.А. Лавникович*
(подпись)

Задание принял к исполнению _____ *Ю.О. Момотова*
(дата и подпись студента)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
1 ОБЗОР АНАЛОГОВ	5
2 СТРУКТУРА ДАННЫХ.....	6
2.1 Для клиентской части	6
2.2 Для серверной части	6
3 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ.....	8
4 РАЗРАБОТКА АЛГОРИТМОВ.....	9
4.1 Алгоритм функции void MainWindow::on_pushSend_clicked()	9
4.2 Алгоритм функции void SendFileToClient(infoFile info).....	9
4.3 Алгоритм функции void InChat::on_pushButton_clicked()	9
5 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	10
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	15
ЗАКЛЮЧЕНИЕ	21
СПИСОК ИСТОЧНИКОВ.....	22

ВВЕДЕНИЕ

Не всегда есть возможность очень быстро перекинуться словом-другим с человеком, находящимся в соседнем офисе, буквально в паре десятков метров.

Работа над совместным проектом, да и личная переписка часто требуют оперативного взаимодействия нескольких человек, и чем быстрее, тем лучше.

На ум сразу приходит решение использовать для этих целей Скайп - приложение, вне всякого сомнения, весьма полезное и, часто, незаменимое.

Но как всегда есть одна проблема - отвлечение от работы. При наличии десятков контактов - родственников, близких и дальних друзей, да и просто знакомых, даже если выставить себе статус "отошёл", бывает трудно сосредоточиться и направить мысли в конкретное русло.

Потому целесообразно использовать такую программу как чат для локальной сети, способную организовать обмен сообщениями в пределах локальной сети - будь то офисная сеть, либо сеть в границах дома.

Однако существует несколько видов чат-серверов.

Например, HTML-вида. Он более доступен, так как для него не требуется специальное программное обеспечение.

Другой вид - Java - работает только со специальной программой, обеспечивающей «живого общения».

Существуют видеоверсии, где помимо обмена сообщениями, можно видеть своего собеседника, следить за его мимикой и жестами, а также специальные голосовые программы, например,

TeamSpeak, необходимые в командных играх.

Очевидно, что применение чатов весьма многообразно и их области применения ограничены лишь нашей фантазией.

Личная заинтересованность в данном проекте следующая:

- 1) Возможность на практике изучить принципы построения клиент-серверных приложений;
- 2) Получение опыта в работе с графической библиотекой Qt;
- 3) Возможность дальнейшего развития проекта (например, добавить возможность создания групп).

1 ОБЗОР АНАЛОГОВ

Существует огромное множество чатов и говорить о них можно много. Поэтому, т.к. данный проект представляет собой локальный чат, обзор аналогов будет проводится среди чатов со схожим функционалом, т.е. среди локальных чатов.

Ниже приведён краткий обзор наиболее популярных среди них:

- LAN Messenger
- Squiggle
- QChat
- Tonic

Проведя сравнительный анализ, можно сделать вывод, что все они реализуют следующий функционал:

- Не требуется сервер или дополнительные сетевые настройки.
- Позволяет соединяться с пользователями и чатиться в локальной сети.
- Позволяет создавать группы участников.
- Рассылка производится индивидуально, по группам или выборочно - по конкретным группам и пользователям.
- Поддерживается обмен файлами.
- История переписки может быть сохранена и просмотрена в любое время.
- Поддержка статусов - доступен, оффлайн, занят и др.
- Существует версия программы, не требующая установки.
- Поддерживается обмен файлами.

Естественно у них также есть и отличия.

Так **LAN Messenger** работает под Windows, Mac OS и Linux.

Squiggle имеет встроенную возможность совершать голосовые звонки в локальной сети – голосовой чат.

Qchat даёт возможность гибкой настройки профилей пользователя – ник, полное имя, изображение, адрес, телефон и т.д. Также есть возможность форматировать сообщения, изменять шрифт и добавлять смайлики.

В **Tonic** реализована автоматическое определение запущенных программ в сети, или добавление собеседников по IP адресам и диапазонам адресов. Также есть возможность настройка стиля показа сообщений чата.

Можно сделать вывод: несмотря на то, что существует множество реализаций локального чата, всем им присущи одни и те же возможности. Разница только в функциональных мелочах и интерфейсе.

В данном проекте реализована большая часть основного функционала.

2 СТРУКТУРА ДАННЫХ

2.1 Для клиентской части

В программе используется три основных основной и множество дополнительных бинарных файлов, которые являются одинаковыми по своей структуре и генерируются в процессе работы.

Первый основной файл `name.txt` содержит в себе одну единственную строку, которая является именем данного клиента в системе.

Второй основной файл `info.txt` также содержит строку, которая представляет собой следующее:

«путь сохранения загруженных файлов%ip сервера%»

Третьей основной файл `files.txt` содержит в себе имена файлов, загруженных на сервер и к которым имеет доступ данный клиент.

Информация в них представляет собой блоки:

«int sizeText, String text»

Где значение `sizeText` – размер строки `String`.

Информация из этих трёх файлов считывается при старте программы и преобразуется к нужному виду.

Остальные файлы, находящиеся в папке «Story» – это история переписки. Т.к. программа поддерживает личную переписку, то есть необходимость для каждой беседы иметь отдельный файл. Они названы по имени пользователя с кем ведётся переписка. Файл общего чата назван именем данного пользователя.

Информация в них представляет собой блоки:

«int sizeText, String text»

Для работы с каждым отдельным файлом был создан класс `Story`, а для работы со всей историей был создан класс `ListOfStory`, который содержит вектор объектов класса `Story`.

2.2 Для серверной части

В программе используется два основных файла.

Первый файл `people.txt` хранит информацию о всех зарегистрированных пользователях. Информация в нём хранится в виде следующих блоков:

«int sizeText, String text, int number»

Где `number` – номер клиента в системе.

Также в самом начале файла хранится значение типа `int`, в котором содержится общее количество всех таких блоков, т.е. количество зарегистрированных клиентов.

Для работы с информацией о каждом отдельном пользователе был создан класс `Conect`, а для работы со всей совокупностью - класс `ListOfCli`, который содержит вектор объектов класса `Conect`.

Второй файл files.txt хранит информацию о всех файлах, который пользователи загружали на сервер. Информация в нём хранится в виде следующих блоков:

«int sizeName, String name, int number, int NumberPacFiles, int sizeFile»

Где name – имя файла, number – порядковый номер в списке, NumberPacFiles – количество пакетов по 1024 байта, sizeFile – полный размер файла.

Также в самом начале файла хранится значение типа int, в котором содержится общее количество всех таких блоков, т.е. количество файлов, хранящихся на сервере.

Для работы с информацией о каждом отдельном загруженном файле был создан класс infoFile, а для работы со всей совокупностью - класс listOfInfoFile, который содержит вектор объектов класса infoFile.

3 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Данный проект было построено по архитектуре клиент-сервер.

Клиент-серверная архитектура описывает распределенные системы, состоящие из отдельных клиента и сервера и соединяющей их сети. Простейшая форма системы клиент-сервер, называемая 2-уровневой архитектурой – это серверное приложение, к которому напрямую обращаются множество клиентов.

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-сервера ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций.

На стороне клиента выполняется код приложения, в который обязательно входят компоненты, поддерживающие интерфейс с конечным пользователем, производящие отчеты, выполняющие другие специфичные для приложения функции.

Преимуществами данной архитектуры являются:

- возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети;
- все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов, а также на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
- поддержка многопользовательской работы;
- гарантия целостности данных.

4 РАЗРАБОТКА АЛГОРИТМОВ

4.1 Алгоритм функции void MainWindow::on_pushSend_clicked()

Функция считывает сообщение, введенное пользователем, анализирует номер текущего рабочего окна (в какой беседе сейчас находится клиент) и отправляет сообщение серверу.

Алгоритм представлен в Приложении А.

4.2 Алгоритм функции void SendFileToClient(infoFile info)

Функция в цикле отправляет имя файла, загруженного на сервер, всем клиентам, которые находятся в сети.

Алгоритм представлен на чертеже Приложении Б.

4.3 Алгоритм функции void InChat::on_pushButton_clicked()

Функция перезаписывает в один из основных файлов – «name.txt» - информацию (ник клиента), которую пользователь ввёл в появившееся диалоговое окно, относящееся к данному классу.

Алгоритм представлен на чертеже Приложении В.

5 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Приложение, реализующее сервер, содержит в себе следующие классы:

- Connect
- InfoFile
- ListOfCli
- ListOfInfoFiles

Приложение, реализующее клиент, содержит в себе следующие классы:

- InChat
- ListOfStory
- MainWindow
- Story
- ChServer

Класс *Connect* отвечает за создание и хранение информации об одном клиенте.

Содержит следующие protected поля:

QString name – имя клиента (ник, под которым он зарегистрирован в системе);

SOCKET ConnectCli – сокет, через который осуществляется связь с данным клиентом;

int num – личный порядковый номер (присваивается во время регистрации в системе);

bool con – подключён ли данный клиент к сети в данный момент.

Содержит следующие методы:

public explicit Connect(QString, SOCKET,int,bool) – конструктор, принимающий начальные значения для экземпляра класса.

public Connect() – конструктор по умолчанию

Помимо этих методов содержит в себе сеттеры и геттеры для каждого поля класса.

Класс *InfoFile* отвечает за создание и хранение информации об одном загруженном на сервер файле.

Содержит следующие protected поля:

int number – порядковый номер файла в системе;

int sizeFile – размер файла;

int numberPacFile – количество пакетов по 1024 байта;
QString dotFile – имя файла (с расширением).

Содержит следующие методы:

public InfoFile() – конструктор, принимающий начальные значения для экземпляра класса;
public InfoFile(int,int,int,QString) – конструктор по умолчанию.

Помимо этих методов содержит в себе сеттеры и геттеры для каждого поля класса.

Класс *ListOfCli* отвечает за хранение и работу со всеми клиентами.
Содержит следующие private поля:

QVector<Connect> List – информация обо всех клиентах.

Содержит следующие методы:

public ListOfCli() – конструктор по умолчанию;
public void add(Connect) – добавление информации о новом клиенте;
public int isNeed(QString) – проверка, существует ли в системе клиент с таким именем;
public Connect& getList(int) – извлечение из списка информации о клиенте по его номеру;
public int readFromFile() – считывание всей информации о клиентах из файла. Осуществляется при старте сервера;
public void writeToFile() – запись всей информации о клиентах в файл. Осуществляется при завершении работы сервера;
public int size() – размер списка клиентов.

Класс *ListOfInfoFiles* отвечает за хранение и работу со всеми файлами, загруженными на сервер.

Содержит следующие private поля:

QVector<InfoFile> ListFiles – информация обо всех файлах

Содержит следующие методы:

public ListOfInfoFile()
public void add(InfoFile)
public InfoFile& getList(int) -
public int readFromFile();

```
public void writeToFile();  
public int size();
```

Назначение методов аналогично классу *ListOfCli*.

Класс *InChat* отвечает за авторизацию пользователя. Вызывается, если пользователь первый раз зашёл в систему.

Содержит следующие private поля:

Ui::InChat *ui – пользовательский интерфейс окна ввода имени.

Содержит следующие методы:

```
explicit InChat(QWidget *parent = 0) – конструктор по умолчанию;  
~InChat() – деструктор;  
private slots void on_pushButton_clicked() – подтверждение ввода, либо  
выход.
```

Класс *ListOfStory* отвечает за хранение и использование всех историй сообщений. При смене беседы подгружается нужная история.

Содержит следующие private поля:

QVector <Story> listOfStory – список всех историй

Содержит следующие методы:

```
public ListOfStory() – конструктор по умолчанию;  
public void add(Story) – добавление новой истории  
public Story& getStory(int) – возвращает историю по порядковому  
номеру;  
public int find(QString ) – поиск по имени беседы;  
public void close() – сохранение изменение всех бесед в  
соответствующие файлы.
```

Класс *MainWindow* отвечает за хранение и использование всех историй сообщений. При смене беседы подгружается нужная история.

Содержит следующие private поля:

Ui::MainWindow *ui – для общения с пользователем интерфейс представлен окном QMainWindow.

Содержит следующие public поля:

SOCKET my_sock – сокет для связи с сервером;
 char nameP[150] – имя клиента;
 ListOfStory listOfStory – список историй для каждой беседы. При выборе нужной беседы главное окно очищается и выводится история нужной беседы;
 int workWindow – номер активной беседы;
 QString trail – путь сохранения файлов;
 char SERVERADDR[15] – IP сервера;
 int numberOfGtntralStory – номер общей беседы (номер клиента в системе).

Содержит следующие методы:

public void WriteMassageToServer(char *) – вывод сообщения на главное окно;
 public void WriteName(char *) – записать имя в окне пользователей;
 public void AddNameFile(char *) – записать имя файла в окне файлов;
 public void *closeEvent*(QCloseEvent *event) – выполняет определённые действия перед закрытием окна. Отправляет сообщение серверу, что данный клиент отсоединился о сети;
 public void beforeWork() – перед подключение к серверу загружает из файлов имя клиента и ip сервера;
 public void WriteMassageToServerWithoutStory(char *) – отображение сообщений в главном окне без записи их в историю. Необходимо для отображения этой самой истории, чтобы повторно её не записывать;
 public void setBackgroundNamePeople() – установка постоянного отличительного фона для общей беседы;
 public void readListOfFiles() – считывание из файла имён всех доступных файлов и занесение их в окно файлов.
 public void writeListOfFiles() – считывание из окна файлов имён всех доступных файлов и их сохранение в файл;
 public void setColor(int row) – изменение цвета имени беседы, если тута пришло новое не прочитанное сообщение;
 public SOCKET startConnect() – функция установки соединения с сервером.
 private slots void on_pushSend_clicked() – отправки сообщения серверу;
 private slots void on_pushButton_clicked() – отправка файла серверу;
 private slots void on_lisfFiles_doubleClicked(const QModelIndex &index) – скачивание файла сервера;
 private slots void on_lustOfPeople_doubleClicked(const QModelIndex &index) – смена беседы;
 private slots void on_connect_clicked() – попытка подключения к серверу;
 private slots void on_actionIP_triggered() – изменение ip сервера.

Класс *Story* отвечает за хранение и использование одной конкретной беседы.

Содержит следующие private поля:

QVector <QString> listOfMes – история всех сообщений данной беседы;

QString name – имя беседы

Содержит следующие методы:

public Story() – конструктор по умолчанию;

public Story(QString) – конструктор, принимающий имя беседы.

Создаётся новый файл для неё;

public void add(QString) – добавление сообщения к общей истории;

public QString gerStr(int number) – возвращает строку по её номеру;

public void setNameFile(QString) – изменение имени беседы;

public int readFromFile() – считывание всех сообщений из файла;

public void writeToFile() – запись всех изменений беседы в файл;

public int size() – размер (количество сообщений);

public QString getName() – возвращает имя файла;

Класс *ChServer* представляет собой класс для изменения ip сервера.

Содержит следующие private поля:

Ui::ChServer *ui – для общения с пользователем интерфейс представлен окном QDialog.

Содержит следующие методы:

explicit ChServer (QWidget *parent = 0) – конструктор по умолчанию;

~ ChServer () – деструктор;

private slots void on_okBytton_clicked() – подтверждение ввода, либо Выход.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Запускаем сервер. Так как ни одного клиента к нему не подключено и ни одного файла на него не загружено, то мы видим просто сообщение о начале работы (рис 6.1).

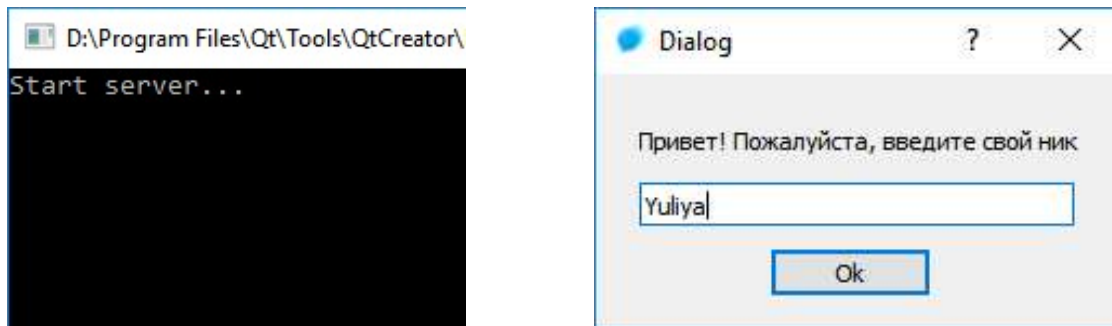


Рисунок 6.1

Запускаем клиент. При первоначальном запуске пользователь предлагают вести имя (ник) под которым он будет зарегистрирован в системе (рис 6.1). К сожалению, имя изменить нельзя.

Запускается главное окно клиента. На данный момент нам нужно ввести ip сервера, к которому мы будем подключаться. Для этого переходим во вкладку Options - ip сервера (рис 6.2)

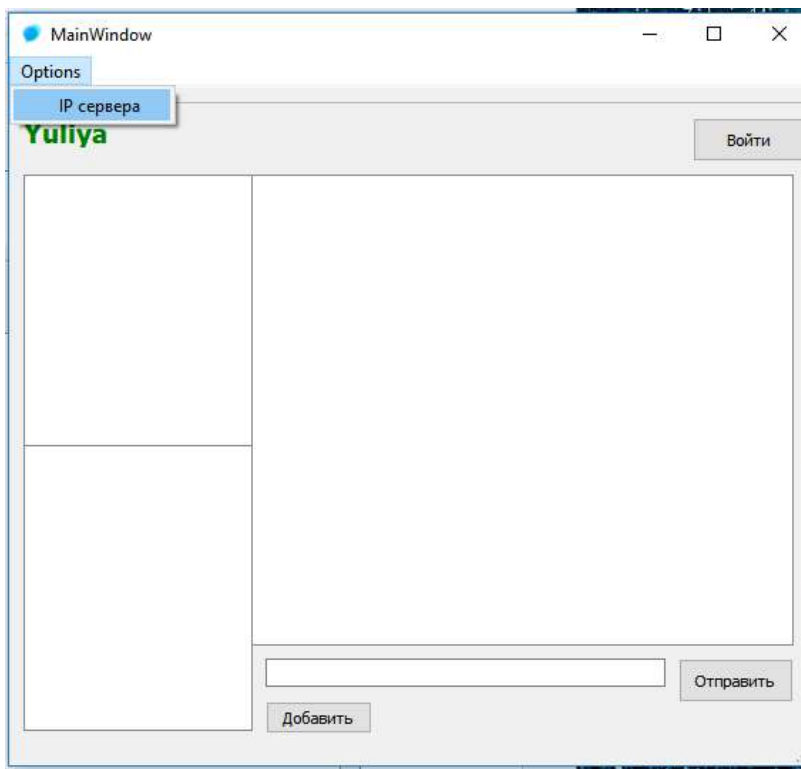


Рисунок 6.2

Далее пользователь предлагают ввести ip сервера (рис 6.3).

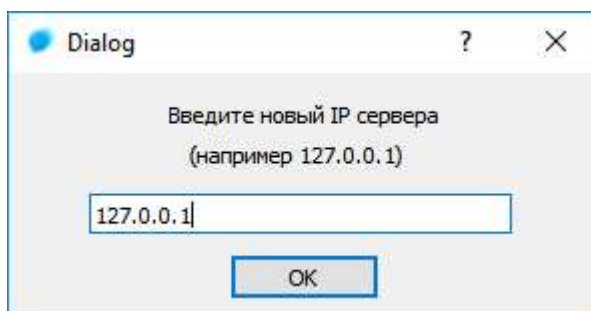


Рисунок 6.3

После ввода он отобразится на главном окне. Далее нажимаем кнопку «Войти». Клиент пытается соединиться с сервером. Если у него не получилось, то он выведет сообщение. В данном случае соединение установлено (рис 6.4).

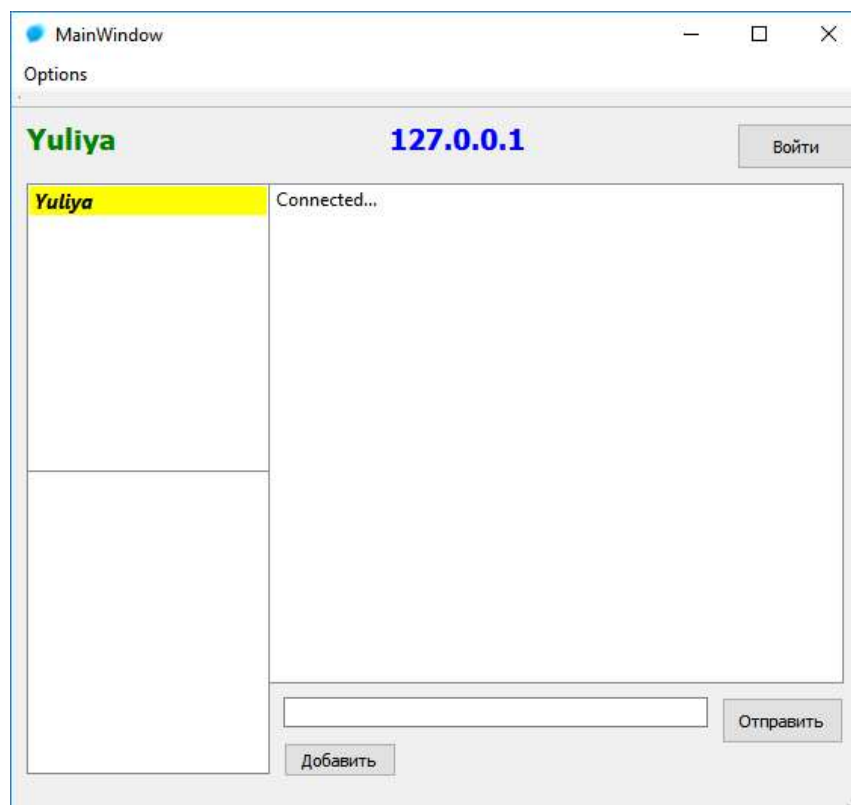


Рисунок 6.4

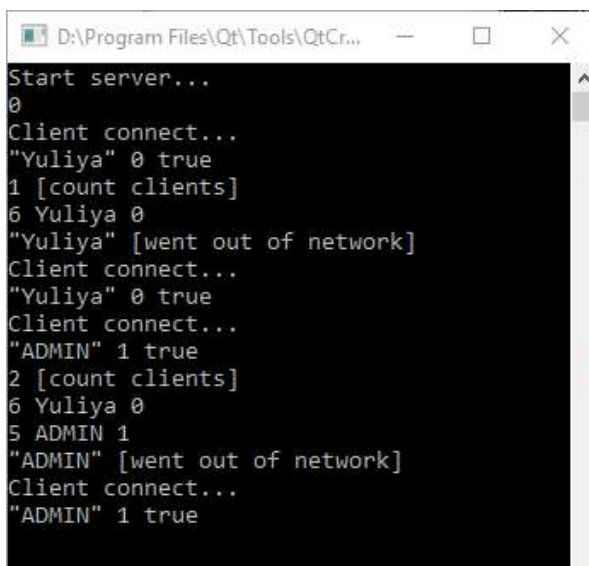
На сервере отобразилось подключение. В качестве демонстрации того, что отображается на сервере, закроем и вновь откроем нашего клиента, а затем подключим ещё одного клиента. (рис 6.5)

Следует сказать пару слов об организации клиентской части. Основное рабочее поле разделено на три области.

По центру размещается окно вывода сообщений, т.е. переписка.

Верхнее левое – окно бесед. Там отображаются беседы, названные именами клиентов, с которыми они связаны. Беседе, названная именем пользователя является общей, и она всегда выделена жёлтым цветом. Для перехода между беседами достаточно дважды щёлкнуть по имени нужной.

В левом нижнем угле располагается окно файлов. Там отображаются все файлы, которые загружены на сервер и доступны для скачивания данному клиенту. Их можно скачивать в любое время.



```
D:\Program Files\Qt\Tools\QtCr...
Start server...
0
Client connect...
"Yuliya" 0 true
1 [count clients]
6 Yuliya 0
"Yuliya" [went out of network]
Client connect...
"Yuliya" 0 true
Client connect...
"ADMIN" 1 true
2 [count clients]
6 Yuliya 0
5 ADMIN 1
"ADMIN" [went out of network]
Client connect...
"ADMIN" 1 true
```

Рисунок 6.5

Теперь подключим ещё одного клиента. Они отправят несколько сообщений в общую беседу (рис. 6.6 и рис. 6.7).

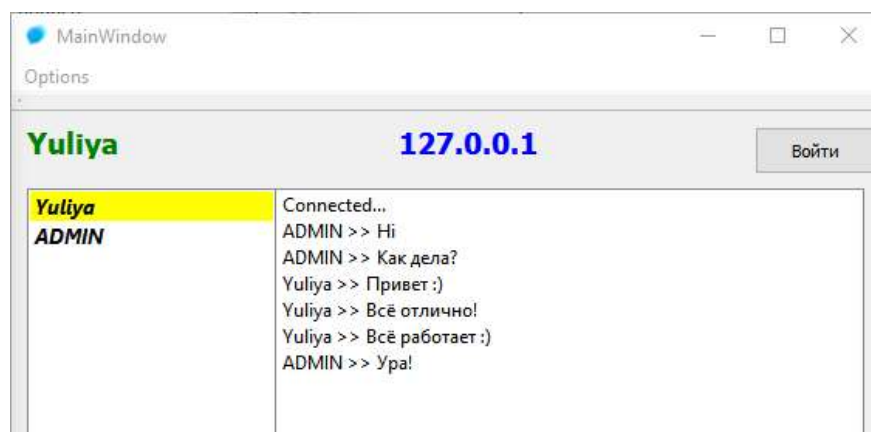


Рисунок 6.6

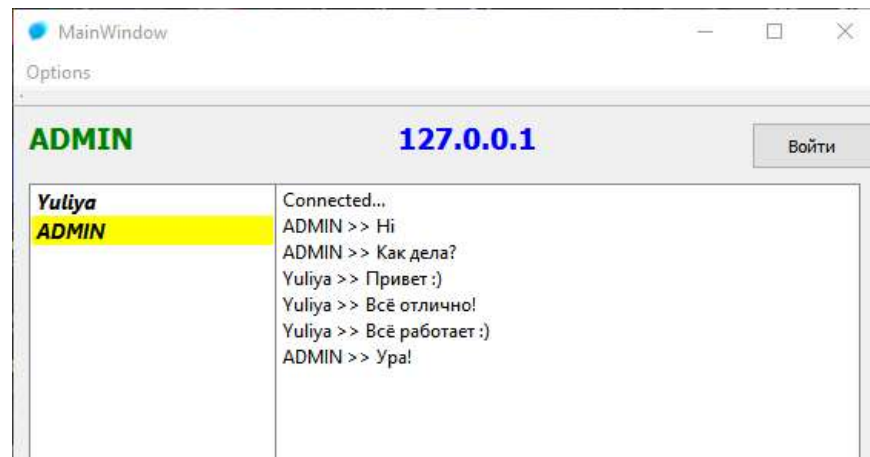


Рисунок 6.7

Теперь проверим приватные сообщения. Для этого двойным щелчком по имени беседы «ADMIN» сменим беседу. Затем введём сообщение, адресованное только данному пользователю (рис 6.8). У клиента «ADMIN» беседа «Yuliya» станет зелёного цвета. Значит, там есть непрочитанные сообщения (рис.6.9). В общей беседе это сообщение не отобразилось.

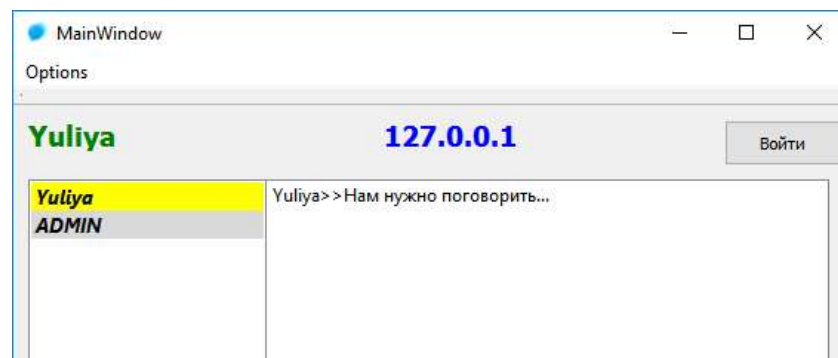


Рисунок 6.8

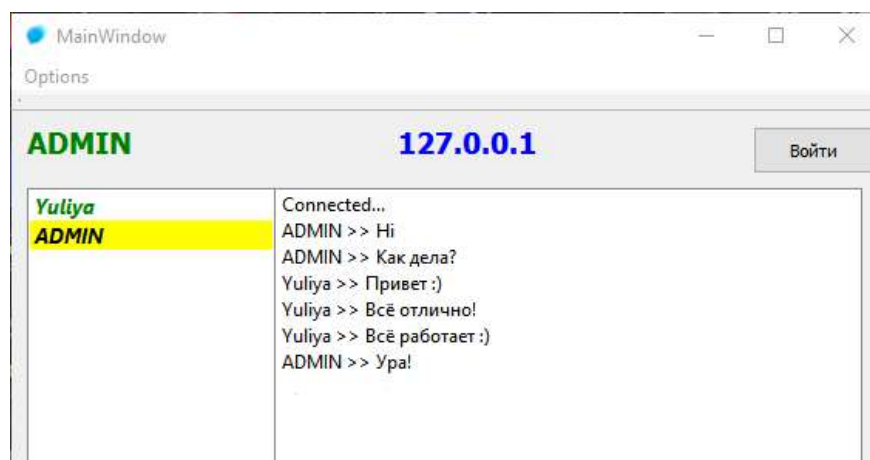


Рисунок 6.9

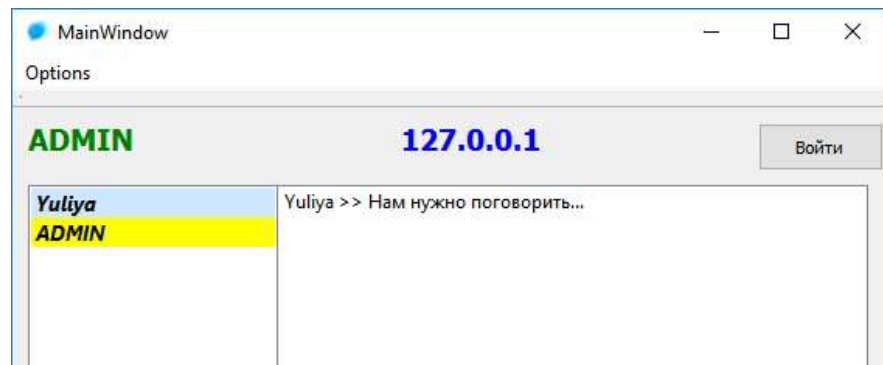


Рисунок 6.10

Теперь загрузим на сервер несколько файлов, затем скачаем один из них. Для загрузки файлов достаточно нажать на кнопку «Добавить». Откроется стандартный проводник, и пользователь сможет выбрать нужный файл. Как только файл загружается, то его имя рассылается клиентам (одному или всем) которые находятся в сети.

Имена файлов отображаются в левом нижнем окне. И оттуда его можно скачать, дважды кликнув на имя нужного файла (рис 6.11).

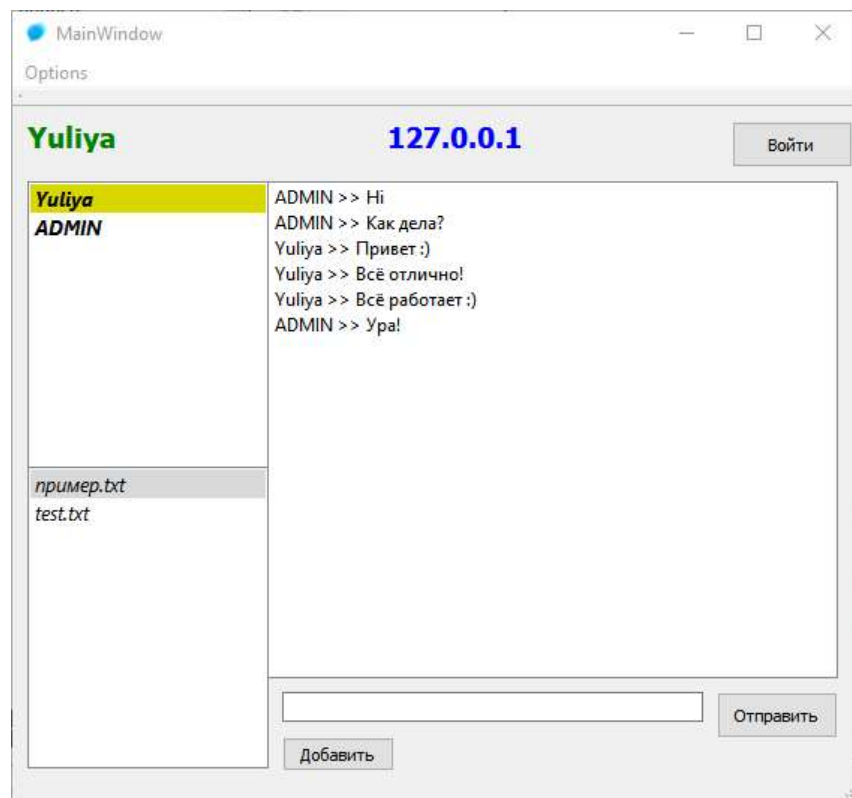
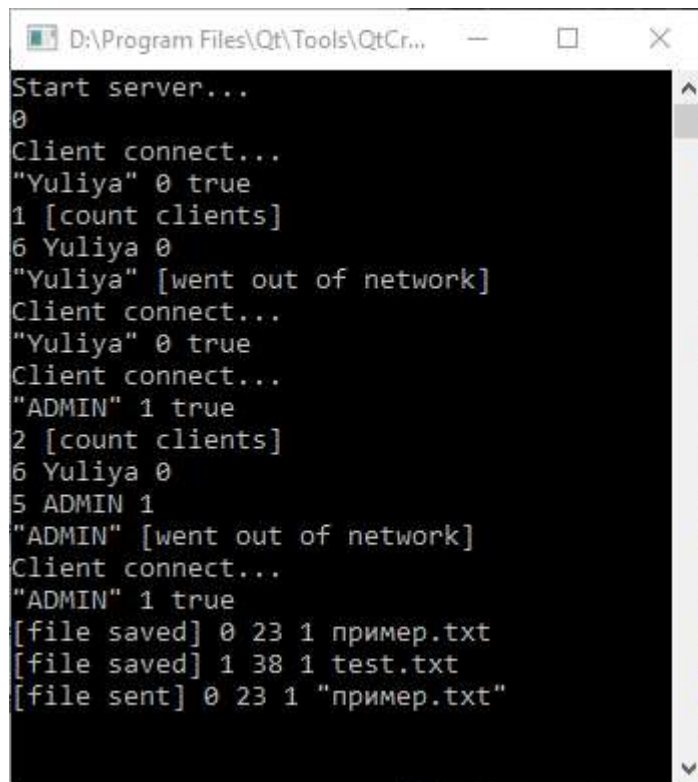


Рисунок 6.11

На сервере отображается история загрузки, изменения и скачивания файлов в пределах текущего сеанса работы сервера (рис. 6.12).

A screenshot of a Qt Creator console window. The title bar shows the file path "D:\Program Files\Qt\Tools\QtCr...". The console output is as follows:

```
Start server...
0
Client connect...
"Yuliya" 0 true
1 [count clients]
6 Yuliya 0
"Yuliya" [went out of network]
Client connect...
"Yuliya" 0 true
Client connect...
"ADMIN" 1 true
2 [count clients]
6 Yuliya 0
5 ADMIN 1
"ADMIN" [went out of network]
Client connect...
"ADMIN" 1 true
[file saved] 0 23 1 пример.txt
[file saved] 1 38 1 test.txt
[file sent] 0 23 1 "пример.txt"
```

Рисунок 6.12

ЗАКЛЮЧЕНИЕ

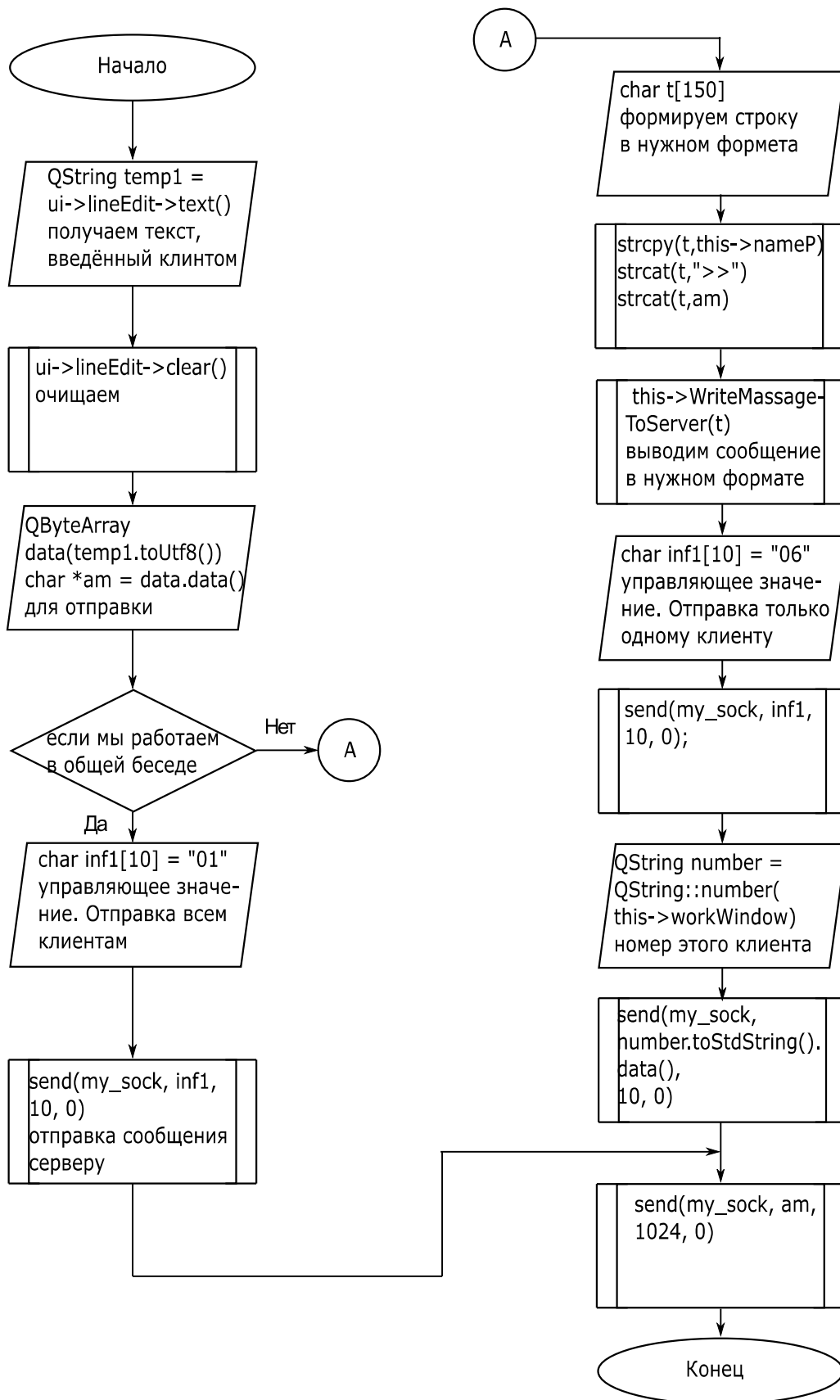
На момент защиты курсовой удалось создать приложение, которое выполняет минимальный функционал для обмена сообщениями и файлами. Клиентская часть приложения имеет достаточно удобный и интуитивно понятный интерфейс. Серверная часть сообщает все необходимые сведения об активности клиентов.

Не обошлось и без недостатков: отсутствие разбиения клиентов на группы, отсутствие индикатора активности (в сети, не в сети). Реализовать всё это помешало отсутствие опыта в разработке приложений, а также маленький срок реализации курсового проекта.

Однако, на данный момент реализация этих функций находится в разработке и в будущем они будут добавлены. В конечном итоге может получиться хороший продукт.

СПИСОК ИСТОЧНИКОВ

1. Лафоре Р. Объектно-ориентированное программирование в C++/ 4-е издание М.:Питер, 2004. – 923 с.
2. Страуструп, Б. Программирование. Принципы и практика с использованием C++/Б. Страуструп, второе издание: Пер. с англ. – М.: ООО «И.Д. Вильямс». 2016. – 1328 с.
3. Jasmin Blanchette C++ GUI Programming with Qt 4/2006 – 560 с.
4. https://github.com/BlackFox147/SPO_BM.git Исходный код



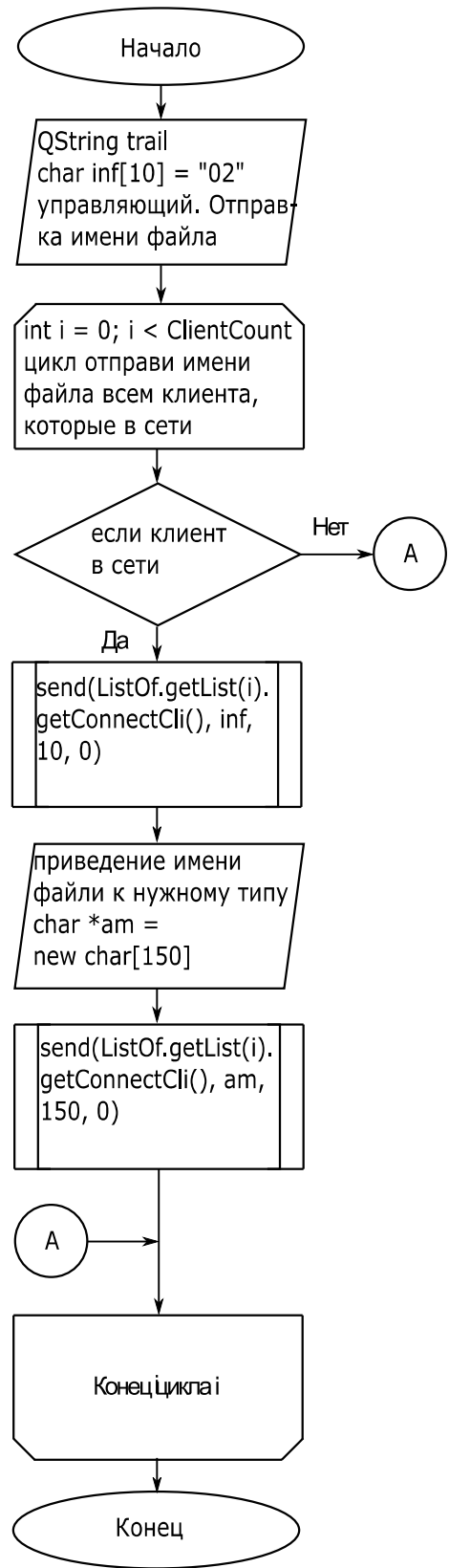
ГУИР.400201.116. ПД.1

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.	Момотова			
Пров.	Лавникович			

Схема
Алгоритма работы функции
on_pushSend_clicked()

Лит.	Масса	Масштаб
Лист 1	Листов 1	

ЭВМ, зр. 550501

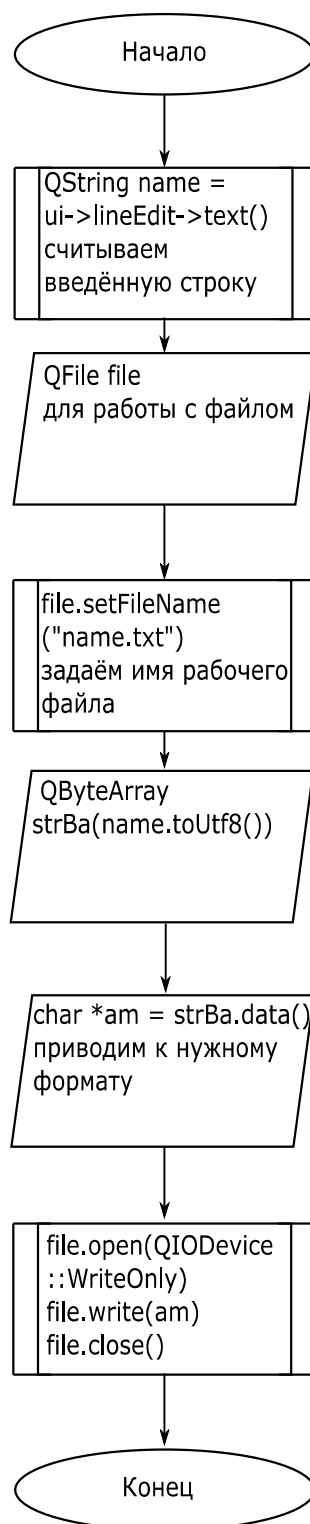


Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Момотова		
Пров.		Лавникович		

ГУИР.400201.116. ПД.2

Схема
Алгоритма работы функции
SendFileToClient(infoFile info)

Лит.	Масса	Масштаб
Лист 1	Листов 1	



ГУИР.400201.116. ПД.3

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Момотова		
Пров.		Лавникович		

Схема
Алгоритма работы функции
on_pushButton_clicked()

Лит.	Масса	Масштаб
Лист	1	Листов 1

ЭВМ, зр. 550501

Обозначение					Наименование					Примечание				
					<u>Графические документы</u>									
ГУИР.400201.116 ПЗ.1					Схема алгоритма работы функции on_pushSend_clicked()					А4				
ГУИР.400201.116 ПЗ.2					Схема алгоритма работы функции SendFileToClient(infoFile info)					А4				
ГУИР.400201.116 ПЗ.3					Схема алгоритма работы функции on_pushButton_clicked()					А4				
					<u>Текстовые документы</u>									
БГУИР КП 1-40 02 01 116 ПЗ*					Пояснительная записка					22 с.				