

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему

УТИЛИТА УДАЛЕННОГО УПРАВЛЕНИЯ КОМПЬЮТЕРОМ

БГУИР КП 1-40 02 01 111 ПЗ

Студент: гр. 150501 Климович А. Н.

Руководитель: Поденок Л. П.

Минск 2023

Учреждение образования  
«Белорусский государственный университет информатики  
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ  
Заведующий кафедрой ЭВМ  
\_\_\_\_\_ Б.В.  
Никульшин  
(подпись)  
«\_\_\_» \_\_\_\_\_ 2023 г.

ЗАДАНИЕ  
по курсовому проектированию  
Студенту Климовичу Алексею Николаевичу

1. Тема проекта Утилита удаленного управления компьютером
2. Срок сдачи студентом законченного проекта 17.05.2023 г.
3. Исходные данные к проекту Язык программирования – C (C++), фреймворк – Qt
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)  
Введение. 1 Обзор литературы. 2 Системное проектирование. 3 Функциональное проектирование. 4 Разработка программных модулей. 5 Результаты работы. Заключение. Список использованных источников
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) 1 Схема структурная сервера. 2 Схема структурная клиента. 3 Схема алгоритма recvBasicInfoFromClient(). 4 Схема алгоритма gen\_mouse\_event().
6. Консультант по проекту Поденок Л. П.
7. Дата выдачи задания 20 февраля 2023 г.
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):  
раздел 1 к 1 марта 2023 г. – 20 %;  
разделы 2, 3 к 1 апреля 2023 г. – 30 %;  
разделы 4, 5 к 1 мая 2023 г. – 30 %;  
оформление пояснительной записки и графического материала к 15 мая 2023 г. – 20 %  
Защита курсового проекта с 29 мая 2023 г. по 10 июня 2023 г.

РУКОВОДИТЕЛЬ \_\_\_\_\_ Л. П. Поденок  
(подпись)

Задание принял к исполнению \_\_\_\_\_ *А.Н. Климович*  
(дата и подпись студента)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1 ОБЗОР ЛИТЕРАТУРЫ.....	8
1.1 Анализ существующих аналогов.....	8
1.2 Постановка задачи.....	9
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ .....	11
2.1 Серверное приложение .....	11
2.1.1 Модуль графического интерфейса .....	11
2.1.2 Модуль обработки скриншотов.....	11
2.1.3 Модуль обработки событий клавиатуры.....	12
2.1.4 Модуль обработки событий мыши.....	12
2.1.5 Модуль сети.....	12
2.1.6 Схема структурная сервера.....	12
2.2 Клиентское приложение .....	12
2.2.1 Модуль пользовательского интерфейса .....	12
2.2.2 Модуль обработки скриншотов.....	13
2.2.3 Модуль обработки событий клавиатуры.....	13
2.2.4 Модуль обработки событий мыши.....	13
2.2.5 Модуль сети.....	14
2.2.6 Модуль валидации .....	14
2.2.7 Схема структурная клиента .....	14
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....	15
3.1 Описание сервера .....	15
3.1.1 Класс MainWindow.....	15
3.1.2 Класс ServerThread.....	16
3.1.3 Класс Screenshot .....	16
3.2 Описание клиента.....	17
3.2.1 Класс MainWindow.....	17
3.2.2 Класс ClientThread.....	18
3.2.3 Класс Dialog.....	19
3.2.4 Класс Validator.....	19
3.3 Описание общих функций для сервера и клиента .....	20
3.3.1 Класс Client .....	20
3.3.2 Класс Mouse.....	20
3.3.3 Класс Keyboard .....	21
3.3.4 Функции работы с сокетами .....	22
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	23
4.1 Алгоритм по шагам функции send_file() .....	23
4.2 Алгоритм по шагам функции moveMouse() .....	24
4.3 Схема алгоритма функции recvBasicInfoFromClient().....	24
4.4 Схема алгоритма функции gen_mouse_event().....	25
5 РЕЗУЛЬТАТЫ РАБОТЫ.....	26

5.1 Тестирование .....	26
5.2 Системные и программные требования.....	29
5.3 Код программы.....	30
ЗАКЛЮЧЕНИЕ .....	31
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	32
ПРИЛОЖЕНИЕ А (обязательное) Серверная часть. Схема структурная.....	33
ПРИЛОЖЕНИЕ Б (обязательное) Клиентская часть. Схема структурная ...	34
ПРИЛОЖЕНИЕ В (обязательное) Схема алгоритма recvBasicInfoFromClient( ) .....	35
ПРИЛОЖЕНИЕ Г (обязательное) Схема алгоритма gen_mouse_event( ) ..	36
ПРИЛОЖЕНИЕ Д (обязательное) Код программы.....	37
ПРИЛОЖЕНИЕ Е (обязательное) Ведомость документов.....	38

## ВВЕДЕНИЕ

В настоящее время удаленное управление компьютером является важной технологией, позволяющей пользователям работать с компьютером удаленно, без необходимости находиться физически рядом с ним. Это особенно актуально для организаций, которые имеют несколько филиалов или для пользователей, которые работают с компьютером из дома или в пути.

Целью данной курсовой работы является разработка программы для удаленного управления компьютером, которая состоит из двух частей: клиентской и серверной. Соединение между клиентом и сервером будет реализовано с помощью сокетов, а управление компьютером будет осуществляться с помощью мышки. Обе программы будут написаны на языке C/C++.

Язык Си (англ. C) – это компилируемый язык программирования, который был разработан в 1972 году в Bell Labs. Язык Си является одним из самых популярных языков программирования в мире, и он широко используется для разработки операционных систем, драйверов устройств, компиляторов, а также для написания программного обеспечения в области встроенных систем, научных вычислений и многих других областей.

Особенности языка Си включают в себя:

- простой синтаксис и минималистичный набор ключевых слов, что делает язык Си легко читаемым и понятным для программистов;
- возможность написания быстрого и эффективного кода, что делает язык Си идеальным выбором для разработки программного обеспечения, работающего на маломощных устройствах или выполняющего сложные вычисления;
- возможности для работы с памятью, что дает программистам большую свободу и гибкость при разработке программного обеспечения.

Для написания графического интерфейса разрабатываемых программ был выбран язык C++ .

C++ – это компилируемый язык программирования, который был разработан в 1983 году Бьярном Страуструпом. Он является расширением языка Си и добавляет в него множество новых возможностей, включая объектно-ориентированное программирование, шаблоны, исключения и многие другие.

Особенности языка C++ включают в себя:

- объектно-ориентированное программирование, что позволяет легко создавать сложные программы с множеством объектов и классов;
- шаблоны, которые позволяют создавать универсальный код, который может быть использован с различными типами данных;
- управление памятью, что дает программистам большую свободу и гибкость при разработке программного обеспечения;
- исключения, которые позволяют обрабатывать ошибки и исключения в программе.

В качестве среды разработки для выбранных языков программирования будет использована среда разработки Qt Creator вместе с фреймворком Qt.

С помощью довольно популярного фреймворка Qt можно будет создать удобный пользовательский интерфейс с необходимыми пунктами меню. Поэтому для разработки графической оболочки была выбрана среда разработки Qt Creator, предназначенная для работы с самим фреймворком.

Qt Creator – кроссплатформенная свободная IDE для разработки на C, C++, JavaScript и QML. Данная IDE предоставляет программисту широкий функционал для разработки программного обеспечения.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Анализ существующих аналогов

Удаленное управление компьютером – это процесс, в котором один компьютер может управлять другим компьютером удаленно. Он используется для управления компьютерами, находящимися на больших расстояниях, а также для управления компьютерами внутри сети. Существует множество программ, которые позволяют удаленно управлять компьютером, одной из них является TeamViewer.

TeamViewer – это программное обеспечение для удаленного доступа, удаленного управления и удаленного обслуживания компьютеров и других конечных устройств, выпущенное в 2005 году. TeamViewer является популярным инструментом для администрирования компьютеров и часто используется на предприятиях, где есть большое количество ПК. По информации от самого разработчика, это ПО установлено более чем на 15 миллионах компьютеров по всему миру.

Основными возможностями программы TeamViewer являются:

- удаленное управление компьютерами и мобильными устройствами;
- скачивание и загрузка файлов на удаленный ПК;
- вывод удаленных компьютеров из режима сна, а также их перезагрузка или выключение;
- удаленная печать и работа с принтерами;
- проведение аудио и видеоконференций с пользователями удаленных ПК.

Кроме большого набора функций, у программы TeamViewer есть и другие преимущества, например, универсальность. Она работает на всех основных операционных системах включая Windows, Linux, Mac OS X, iOS и Android. При этом подключение к удаленному компьютеру возможно как напрямую, так и при использовании NAT, прокси или брандмауэра. Фактически, программа TeamViewer может обеспечить удаленный доступ в любой ситуации.



Данная программа является платной. Но, для личного использования и с некоторыми ограничениями вы можете использовать ее бесплатно. Обычно, без лицензии сеанс подключения к удаленному компьютеру длится не больше 5 минут, после чего связь обрывается и нужно подключаться заново. Также, если использовать программу слишком активно, то она может распознать вас как «коммерческого пользователя» и еще больше ограничить ваши возможности. Но, нужно отметить, что если вы действительно используете TeamViewer для личных целей, то с этой проблемой вы вряд ли столкнетесь.

К сожалению, 5 мая 2022 года TeamViewer прекратил работу в России и Беларуси в связи с событиями в Украине.

В данном проекте будет продемонстрирован упрощенный аналог этой программы, которая будет являться программой-клиентом.

## **1.2 Постановка задачи**

Требуется разработать две программы: программу-клиент и программу-сервер.

Программа-клиент будет выполнять следующие задачи:

- подключаться к программе-серверу с помощью сокетов;
- получать от сервера скриншоты экрана и выводить их на экран;
- отправлять серверу события мыши.

Также с помощью пароля можно будет создать небольшой уровень защиты от сторонних подключений.

Программа-сервер будет выполнять следующие задачи:

- создавать соединение с помощью сокетов и подключаться к клиенту;
- через заданный промежуток времени отправлять скриншоты экрана клиенту;
- получать от клиента события мышки и обрабатывать их.

Для передачи графической информации между сервером и клиентов нужно будет разработать собственный протокол для передачи изображения экрана сервера клиенту, а также событий мышки на клиенте для передачи на сервер.

В итоге, в ходе написания данного курсового проекта будут рассмотрены следующие вопросы:

- изучение основ удаленного управления компьютером, протоколов передачи данных и способы их реализации;
- изучение языка программирования C/C++ и библиотеки Qt для создания графических интерфейсов;
- тестирование программы и отладка ошибок.

Результатом выполнения курсовой работы будет полнофункциональная программа для удаленного управления компьютером, которая может быть использована в различных ситуациях.

## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

### **2.1 Серверное приложение**

#### **2.1.1 Модуль графического интерфейса**

Модуль пользовательского интерфейса предназначен для взаимодействия пользователя с приложением, основываясь на представлении и визуализации данных.

Для разрабатываемого проекта создается простейший пользовательский интерфейс с помощью фреймворка Qt. Данный пользовательский интерфейс представляет собой набор пунктов меню, с которыми может взаимодействовать пользователь:

- включение сервера;
- выключение сервера;
- установка пароля;
- вывод информации о подключенном клиенте;
- выход из приложения.

#### **2.1.2 Модуль обработки скриншотов**

Модуль работы со скриншотами необходим для генерации скриншотов на сервере, их сохранении и отправке с будущим их клиенту. Здесь осуществляется весь необходимый функционал и возможности:

- создание скриншота экрана и отправка его клиенту;
- сохранение скриншота.
- обработка входящих сообщений от клиента.

### **2.1.3 Модуль обработки событий клавиатуры**

Данный модуль служит для принятия по сети событий клавиатуры и дальнейшей генерации нажатия клавиш. Модуль не рассчитан на обработку и генерацию нажатия некоторых специальных клавиш или комбинаций клавиш.

### **2.1.4 Модуль обработки событий мыши**

Модуль обработки событий мыши работает похожим образом на предыдущий модуль: принимает от клиента событие мыши и генерирует это событие на сервере.

### **2.1.5 Модуль сети**

Модуль сети предназначен для подключения сервера и клиента друг к другу и необходим для их взаимодействия в реальном времени. Данный модуль реализуется с помощью сокетов функциями `socket()`, `bind()`, `listen()`, `accept()`, `send()` и `recv()`.

### **2.1.6 Схема структурная сервера**

Схема структурная сервера представлена в **ПРИЛОЖЕНИИ А**.

## **2.2 Клиентское приложение**

### **2.2.1 Модуль пользовательского интерфейса**

Данный модуль предназначен, как и графический модуль сервера, для взаимодействия пользователя с приложением. С помощью средств Qt он представляет удобный интерфейс с необходимыми пунктами меню:

- соединение с сервером;
- отсоединение от сервера;
- настройка IP сервера и пароля;
- выход из приложения.

### **2.2.2 Модуль обработки скриншотов**

Модуль работы со скриншотами реализует следующий функционал:

- получение скриншота от сервера;
- сохранение скриншота в виде файла;
- вывод принятого скриншота от сервера на экран;

### **2.2.3 Модуль обработки событий клавиатуры**

Данный модуль предназначен для следующих задач:

- отлавливание событий клавиатуры;
- сохранение определенного события в специальную переменную;
- отправка события на сервер с целью его дальнейшей генерации.

### **2.2.4 Модуль обработки событий мыши**

Модуль обработки событий мыши работает в постоянном режиме, так как непрерывно отправляет серверу координаты курсора мыши. Также, если произошло какое-то событие, связанное с мышью, клиент также его обрабатывает и отправляет серверу.

### **2.2.5 Модуль сети**

Модуль сети предназначен для подключения сервера и клиента друг к другу и необходим для их взаимодействия в реальном времени. Данный модуль реализуется с помощью сокетов функциями `socket()`, `connect()`, `send()` и `recv()`.

### **2.2.6 Модуль валидации**

Данный модуль предназначен для валидации IP-адреса сервера, к которому подключается клиент. Модуль валидации реализуется с помощью регулярного выражения, используя средства и механизмы Qt.

### **2.2.7 Схема структурная клиента**

Схема структурная клиента представлена в **ПРИЛОЖЕНИИ Б**.

## 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

### 3.1 Описание сервера

#### 3.1.1 Класс MainWindow

Данный класс наследуется от библиотечных классов QMainWindow и является графическим представлением программы окно, в котором происходит работа в приложении.

Поля:

- Ui::MainWindow\* ui – пользовательский интерфейс.
- ServerThread serverThread – рабочий поток для сервера.

Функции:

- MainWindow( ) – конструктор.
- ~MainWindow( ) – деструктор.

Слоты:

- on\_actionExit\_triggered( ) – выход из приложения.
- on\_turn\_on\_pushButton\_clicked( ) – включение сервера.
- on\_turn\_off\_pushButton\_clicked( ) – выключение сервера.
- updateCursorSlot( ) – обновить рисунок курсора.
- showClientInfoSlot( ) – показ информации о подключенном клиенте.
- timeIsOverToConnectSlot( ) – время ожидания подключения сервера к клиенту вышло.

Сигналы:

- sendPasswordFromUISignal( ) – сигнал передачи пароля в рабочий поток сервера.

### 3.1.2 Класс **ServerThread**

Класс наследуется от встроенного класса Qt `QThread` и выполняет основные функции сервера по организации соединения и приема-передачи информации.

Поля:

- `struct sockaddr_in client_addr` – адрес клиента
- `int sockfd_for_listen` – сокет для прослушивания входящих запросов для соединения.

- `int sockfd_for_connect` – сокет для передачи информации.

- `int server_status` – статус сервера (включен/выключен).

- `char server_password[MAX_SIZE_STR]` – пароль на сервере.

Сигналы:

- `updateCursorSignal()` – сигнал для обновления рисунка курсора.

- `sendClientInfoSignal()` – сигнал для передачи информации на экран.

- `signalToTurnOff()` – сигнал выключить сервер.

- `timeIsOverToConnectSignal()` – сигнал о том, что время ожидания подключения истекло.

Функции:

- `ServerThread()` – конструктор.

- `run()` – запуск потока.

- `closeThread()` – закрытие потока.

- `closeSockets()` – закрытие сокетов.

- `recvBasicInfoFromClient()` – получение базовой информации от клиента.

### 3.1.3 Класс **Screenshot**

Класс является основным при работе со скриншотами и имеет следующие поля:

- `QScreen* screen` - переменная для захвата экрана.

- `QPixmap screenshot` - переменная для хранения скриншота.



Функции:

- take\_screenshot( ) - сделать скриншот и сохранить его по указанному пути.

## 3.2 Описание клиента

### 3.2.1 Класс MainWindow

Данный класс аналогичен классу MainWindow сервера, но только с тем отличием, что позволяет работать с клиентской стороной.

Поля:

- Ui::MainWindow\* ui – пользовательский интерфейс.
- ClientThread clientThread – рабочий поток для клиента.
- Dialog settings – настройки соединения (пароль и IP-адресс).

Функции:

- MainWindow( ) – конструктор.
- ~MainWindow( ) – деструктор.
- mousePressEvent( ) – переопределение обработчика нажатия кнопок мыши.
- mouseReleaseEvent( ) – переопределение обработчика отпускания кнопок мыши.

- mouseMoveEvent( ) – переопределение движения мыши.
- mouseDoubleClickEvent( ) – переопределение двойного клика мыши.
- keyPressEvent( ) – переопределение нажатия клавиш клавиатуры.

Сигналы:

- sendSettingsFromUISignal( ) – сигнал для отправки настроек с UI.
- mouseEventToClientThreadSignal( ) – сигнал для отправки события мыши.
- keyIsPressed( ) – сигнал для отправки события клавиатуры.

Слоты:

- updateCursorSlot( ) – обновить рисунок курсора
- showClientInfoSlot( ) – показ информации о подключенном клиенте.

- `timeIsOverToConnectSlot()` – время ожидания подключения сервера к клиенту вышло.
- `on_actionExit_triggered()` – выход из приложения.
- `on_actionSettings_triggered()` – зайти в настройки.
- `on_actionConnect_to_server_triggered()` – подсоединиться к серверу.
- `on_actionDisconnect_from_the_server_triggered()` – разорвать соединение с сервером.

### 3.2.2 Класс `ClientThread`

Класс является основным для работы по сети с сервером и наследуется от класса `QThread`.

Поля:

- `struct sockaddr_in server_addr` – адрес сервера.
- `int sockfd_for_connect` – сокет для передачи информации между клиентом и сервером.
- `int client_status` – статус клиента (включен/выключен).

Сигналы:

- `screenshotReady()` – сигнал готовности скриншота для вывода его на экран.
- `showPasswordStatusSignal()` – сигнал для показа правильности пароля.
- `signalToDisconnect()` – сигнал разрыва соединения.
- `connectFailedSignal()` – сигнал о невозможности подключения.

Функции:

- `ClientThread()` – конструктор.
- `~ClientThread()` – деструктор.
- `run()` – запуск потока.
- `closeThread()` – закрытие потока.
- `receiveScreenshot()` – получение скриншота.
- `sendClientInfoToServer()` – отправка информации о клиенте серверу.

Слоты:

- `receiveSettingsFromUISlot( )` – получение введенных настроек с UI.
- `getMouseEventFromWindowSlot( )` – получение события мыши.
- `getKeyEventFromWindowSlot( )` – получение события клавиатуры.

### 3.2.3 Класс **Dialog**

Данный класс реализует небольшое окно для установки настроек для подключения. Он наследуется от класса `QDialog` и имеет следующие поля:

- `Ui::Dialog` - пользовательский интерфейс.
- `Validator validator` – валидатор настроек.
- `DataConnection dataConnection` – структура для установки соединения.

В состав структуры `DataConnection` входят следующие поля:

- `QString server_ip` – IP сервера.
- `QString password` – пароль на сервере.

Функции:

- `Dialog( )` – конструктор.
- `~Dialog( )` – деструктор.

Слоты:

- `on_buttonBox_accepted( )` – сохранить изменения.
- `on_buttonBox_rejected( )` – отменить изменения.

### 3.2.4 Класс **Validator**

Данный класс предназначен для проверки введенных настроек сети. Класс не имеет полей, но имеет одну функцию для проверки правильности IP-адреса – `is_valid_IP( )`.

### 3.3 Описание общих функций для сервера и клиента

#### 3.3.1 Класс Client

Данный класс используется и клиентской, и серверной стороной. Класс имеет следующие поля:

- char local\_host\_name[MAX\_SIZE\_STR] – локальное имя хоста.
- char password[MAX\_SIZE\_STR] – пароль.
- char ip[MAX\_SIZE\_STR] – IP адрес.

Функции:

- get\_local\_host\_name( ) – получение локального имени хоста.
- set\_local\_host\_name( ) – установка локального имени хоста.
- get\_password( ) – получение пароля.
- set\_password( ) – установка пароля.
- get\_ip( ) – получение IP-адреса.
- set\_ip( ) – установка IP-адреса.

#### 3.3.2 Класс Mouse

Данный класс предоставляет возможность управлением мыши с помощью библиотеки X11.

Поля:

- int event\_flag – событие мыши.

Перечисления:

- enum MOUSE\_EVENT {
  - NONE = 0, // мышка бездействует
  - LEFT\_PRESS = 1, // нажатие ЛКМ
  - RIGHT\_PRESS = 2, // нажатие ПКМ
  - RELEASE = 3, // отпускание кнопки
  - DOUBLECLICK = 4, // двойной клик

```

        MOVE = 5                                // движение мыши
    }

```

Функции:

- getMousePos( ) – получение позиции мыши.
- moveMouse( ) – передвинуть курсор мыши.
- gen\_mouse\_event( ) – генерация события мыши.
- getEvent\_flag( ) – получить событие мыши.
- setEvent\_flag( ) – установить событие мыши.

### 3.3.3 Класс Keyboard

Данный класс предоставляет возможность управления клавиатурой с помощью библиотеки X11. Класс имеет следующие поля:

- int event\_flag – признак нажатия клавиши.
- int native\_key\_code – код клавиши.
- enum KEY\_EVENT {
  - NOT\_PRESSED = 0,           // клавиша не нажата
  - IS\_PRESSED = 1           // клавиша нажата

Функции:

- generate\_key\_event( ) – генерация нажатия клавиши.
- get\_event\_flag( ) – получение признака нажатия клавиши.
- set\_event\_flag( ) – установка признака нажатия клавиши.
- get\_native\_key\_code( ) – получить код клавиши.
- set\_native\_key\_code( ) – установить код клавиши.

### 3.3.4 Функции работы с сокетами

Функции:

- `Socket( )` – создать новый сокет и вернуть файловый дескриптор.
- `Bind( )` – связать сокет с IP-адресом и портом.
- `Listen( )` – объявить о желании принимать соединения.
- `Accept( )` – принять запрос на установку соединения.
- `Connect( )` – установить соединение.
- `Inet_pton( )` – преобразование адреса IPv4 и IPv6 из текста в бинарный вид.
- `Close( )` – закрыть файловый дескриптор (сокет).
- `send_file( )` – отправить файл по сети.
- `recv_file( )` – получить файл по сети.

## 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

### 4.1 Алгоритм по шагам функции `send_file( )`

1) Начало.

2) Входные данные:

- `sock` – дескриптор сокета для передачи файла;
- `file_name` – имя передаваемого файла.

Промежуточные данные:

- `file` – файловый поток отправляемого файла;
- `file_size` – размер файла;
- `bytes` – прочитанные байты информации из файла.

Выходные данные:

- файл, отправленный от сервера клиенту.

3) Открытие файла `file` для чтения в бинарном виде функцией `open( )`.

4) Проверка, что файл удалось открыть с помощью функции `is_open( )`.

5) Если файл открыт, то переход на шаг 6, а иначе переход на шаг 14.

6) Вычисление размера файла с помощью функции `std::filesystem::file_size( )` и запись результата в переменную `file_size`.

7) Выделение памяти под массив символов `bytes` размером `file_size+1`.

8) Читаем файл `file` в буфер `bytes` функцией `read( )`.

9) Отправляем клиенту через сокет `sock` строковое представление размера файла с помощью функции `send( )`.

10) Отправляем клиенту через сокет `sock` имя файла `file_name` с помощью функции `send( )`.

11) Отправляем через сокет `sock` содержимое файла `file`, записанное в массиве `bytes`, клиенту с помощью функции `send( )`.

12) Очистка памяти от массива `bytes`.

13) Закрытие файлового потока функцией `close( )`.

14) Конец.

## 4.2 Алгоритм по шагам функции `moveMouse( )`

1) Начало.

2) Входные данные:

- `newPos` – целочисленный массив из двух координат `x` и `y` экрана компьютера, куда нужно переместить курсор мыши.

Промежуточные данные:

- `display` – указатель на структуру типа `Display`, которая содержит информацию о соединении с X-сервером;

- `root` – идентификатор корневого окна X-сервера.

Выходные данные:

- новые координаты курсора мыши.

3) Открытие соединения с X-сервером с помощью функции `XOpenDisplay( )` и сохранение результата в переменную `display`.

4) Если `display = NULL`, то переход на шаг 9, а иначе переход на шаг 5.

5) Получение идентификатора окна корневого уровня на X-сервере с помощью функции `DefaultRootWindow( )` и сохранение результата в переменную `root`.

6) Перемещение курсора мыши на указанные координаты на экране с помощью функции `XWarpPointer( )`.

7) Отправка запроса на сервер X с помощью функции `XFlush( )`.

8) Закрытие соединения с X-сервером функцией `XCloseDisplay( )`, которое было открыто ранее с помощью функции `XOpenDisplay( )`.

9) Конец.

## 4.3 Схема алгоритма функции `recvBasicInfoFromClient( )`

Схема алгоритма функции `recvBasicInfoFromClient( )` представлена в ПРИЛОЖЕНИИ В.



#### **4.4 Схема алгоритма функции `gen_mouse_event()`**

Схема алгоритма функции `gen_mouse_event()` представлена в **ПРИЛОЖЕНИИ Г**.

## 5 РЕЗУЛЬТАТЫ РАБОТЫ

### 5.1 Тестирование

Для начала, стоит упомянуть, что для успешной сборки проекта необходимо скачать и установить библиотеку «X11». Для этого достаточно в терминале (если вы пользователь Linux) прописать следующие команды:

```
$ sudo apt-get install libxtst-dev
```

```
$ sudo apt-get update
```

Также нужно добавить следующие флаги компиляции: `-lX11` и `-lXtst`.

Чтобы собрать программу, нужно использовать «qmake», после чего собрать проект с помощью сгенерированного файла «Makefile». После этих действий можно запустить серверное (см. рисунок 5.1) и клиентское приложение (см. рисунок 5.2).

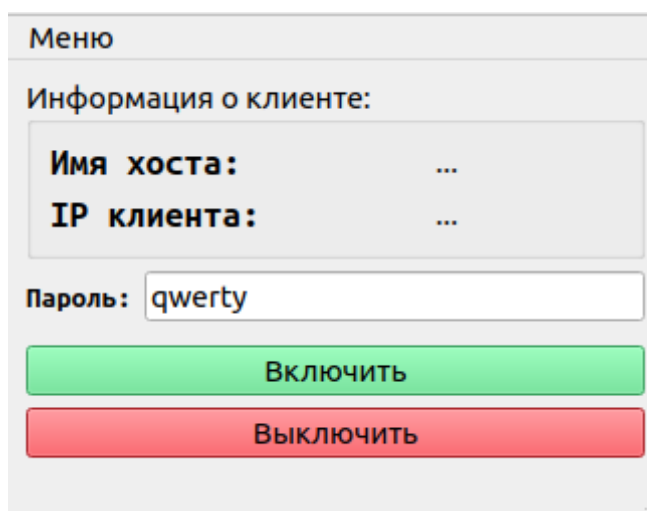


Рисунок 5.1 – Начальный экран запуска серверного приложения

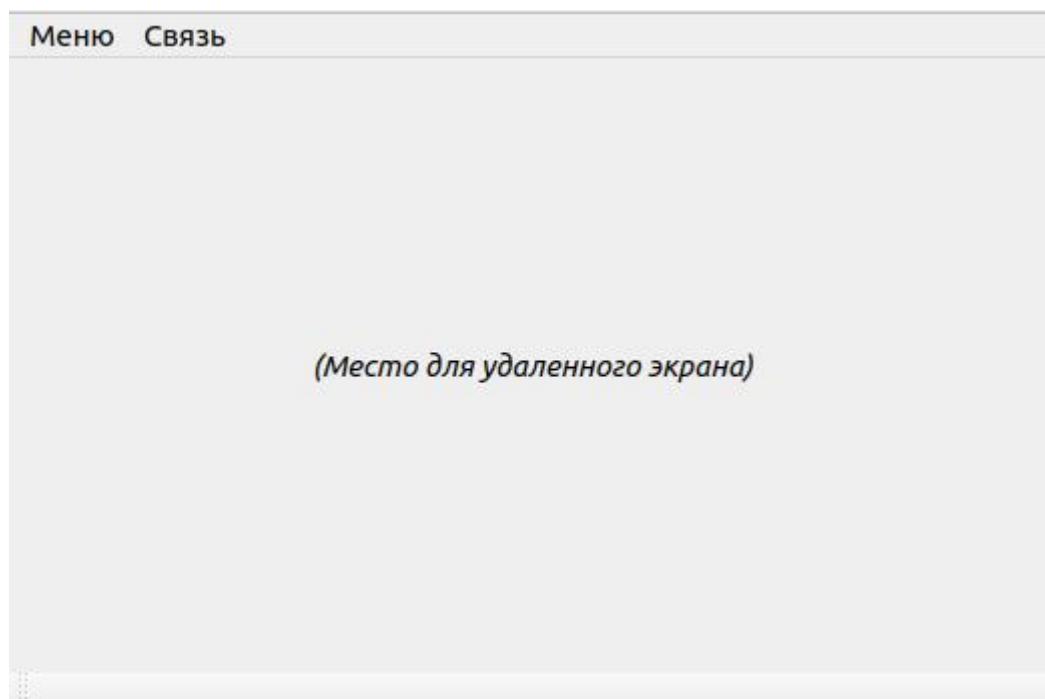


Рисунок 5.2 – Начальный экран запуска клиентского приложения

Теперь можно запустить сервер, кликнув на зеленую кнопку «Включить» (см. рисунок 5.3). Сервер в течении 10 секунд будет ожидать запросов на подключение. Если запросов не будет, сервер автоматически выключится.

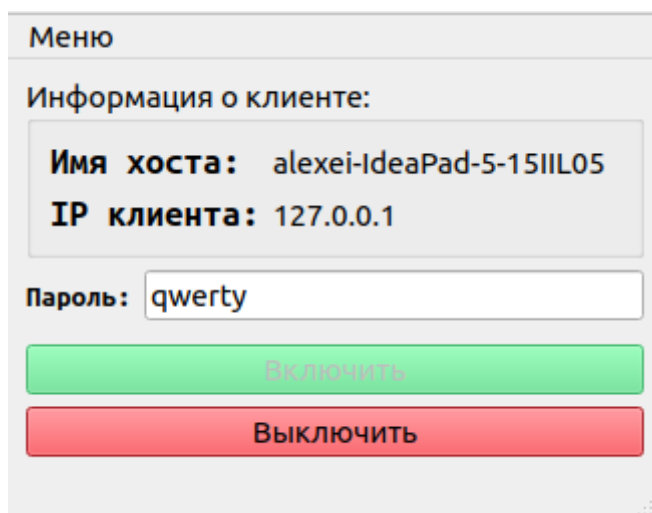


Рисунок 5.3 – Подключение клиента к серверу

Проверим настройки на клиенте для подключения к серверу (см. рисунок 5.4). Для этого нужно в главном меню кликнуть на вкладку «Настройки».

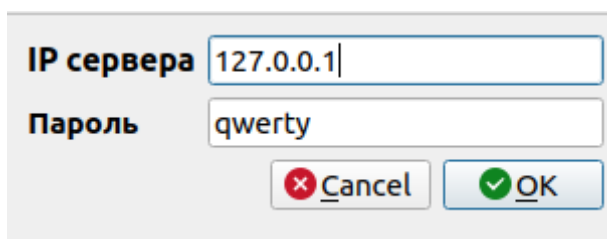


Рисунок 5.4 – Настройки сети на клиенте

Далее на клиенте нажмем на кнопку «Присоединиться к серверу», чтобы подключиться к серверу. Появится окно, сигнализирующее об успешном подключении клиента к серверу, после чего начнется передача скриншотов от сервера клиенту и обработка мыши и клавиатуры (см. рисунок 5.5).

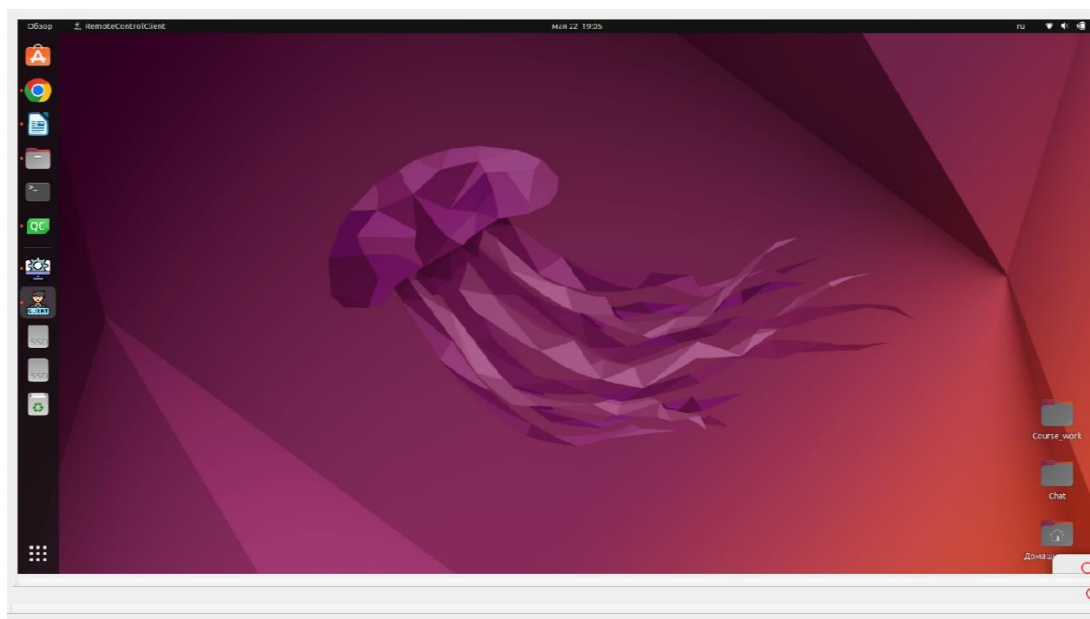


Рисунок 5.5 – Работа клиента, подключенного к серверу

Также, можно заметить, что на сервере в группе «Информация о клиенте» заполнились поля «Имя хоста» и «IP клиента».

При нажатии на клавишу F12, клиентское приложение сворачивается, после чего можно нажать на кнопку «Отсоединиться от сервера». Серверное приложение при этом перестает посылать скриншоты и обрабатывать события мыши и клавиатуры.

У обоих программ есть кнопки «Выход», нажав на которые можно выйти из приложения.

## **5.2 Системные и программные требования**

Данный проект может быть скомпилирован с помощью любого Linux-совместимого компилятора и запущен на компьютере, который соответствует следующим требованиям:

- процессор: 2.0+ MHz, 2+ ядра;
- ОЗУ: 4+ GB;
- операционная система: Linux.

Для написания кода использовалась кроссплатформенная свободная IDE – Qt Creator 6.0.2 на основе Qt 5.15.3 (GCC 11.2.0, 64 бита).

Также, в ходе написания проекта были выявлены следующие программные ограничения:

- максимальное количество подключаемых клиентов к серверу равно 1;
- некоторые клавиши и их комбинации невозможно словить на клиенте и выполнить на сервере;
- в каталоге сборки сервера и клиента заранее созданы каталоги «Screenshots», где временно хранятся переданные и полученные скриншоты на сервере и клиенте соответственно, удалять их нельзя.

Некоторые настройки заданы с самого начала по умолчанию в конфигурационном файле «config.h». При желании их можно поменять.

### **5.3 Код программы**

Код программы представлен в **ПРИЛОЖЕНИИ Д.**

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения данного курсовой проекта была создана программа на языке C/C++, которая позволяет удаленно управлять компьютером с графической оболочкой на Qt. Программа была протестирована на операционной системе Linux Ubuntu 22.04 LTS. Разработанная программа может быть использована для удаленного управления компьютерами в различных сферах деятельности, включая IT-сферу, медицину и образование.

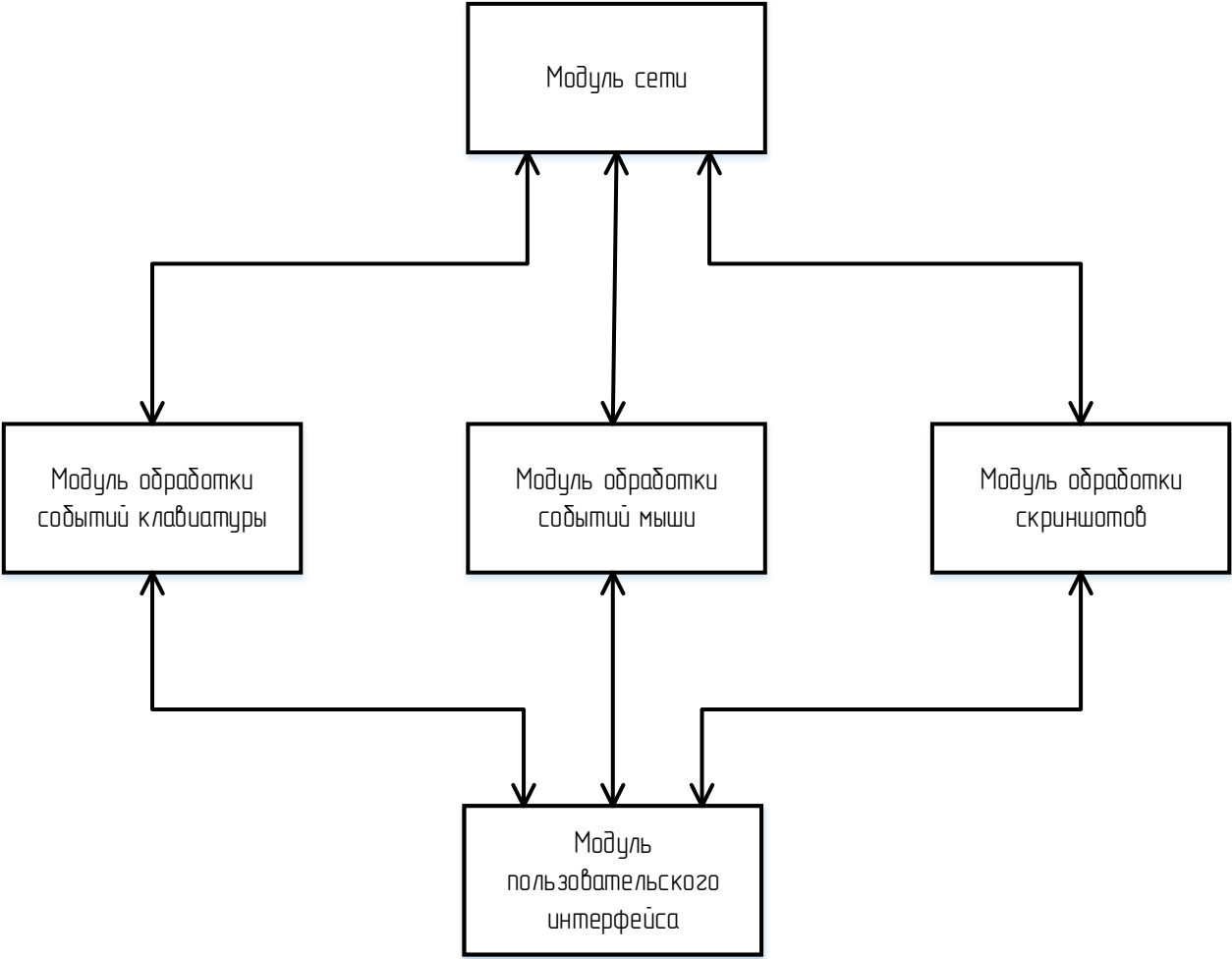
В дальнейшем возможно усовершенствование приложений, путем улучшения графического интерфейса, расширение функционала, а также добавления кроссплатформенности.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Стивенс, У. UNIX: разработка сетевых приложений / У. Стивенс. – СПб.: Питер, 2003. – 1088 с: ил. – (Серия «Мастер-класс»).
- [2] Страуструп, Б. Язык программирования C++ / Б. Страуструп; специальное издание; пер. с англ. – СПб. : BHV, 2008.
- [3] Документация фреймворка Qt. – [Электронный ресурс]. – Адрес ресурса: <https://doc.qt.io> – Дата доступа 08.05.2023.
- [4] Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Минск БГУИР 2019. – Адрес ресурса: [https://www.bsuir.by/m/12\\_100229\\_1\\_136308.pdf](https://www.bsuir.by/m/12_100229_1_136308.pdf) – Дата доступа 12.05.2023



**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Серверная часть. Схема структурная**

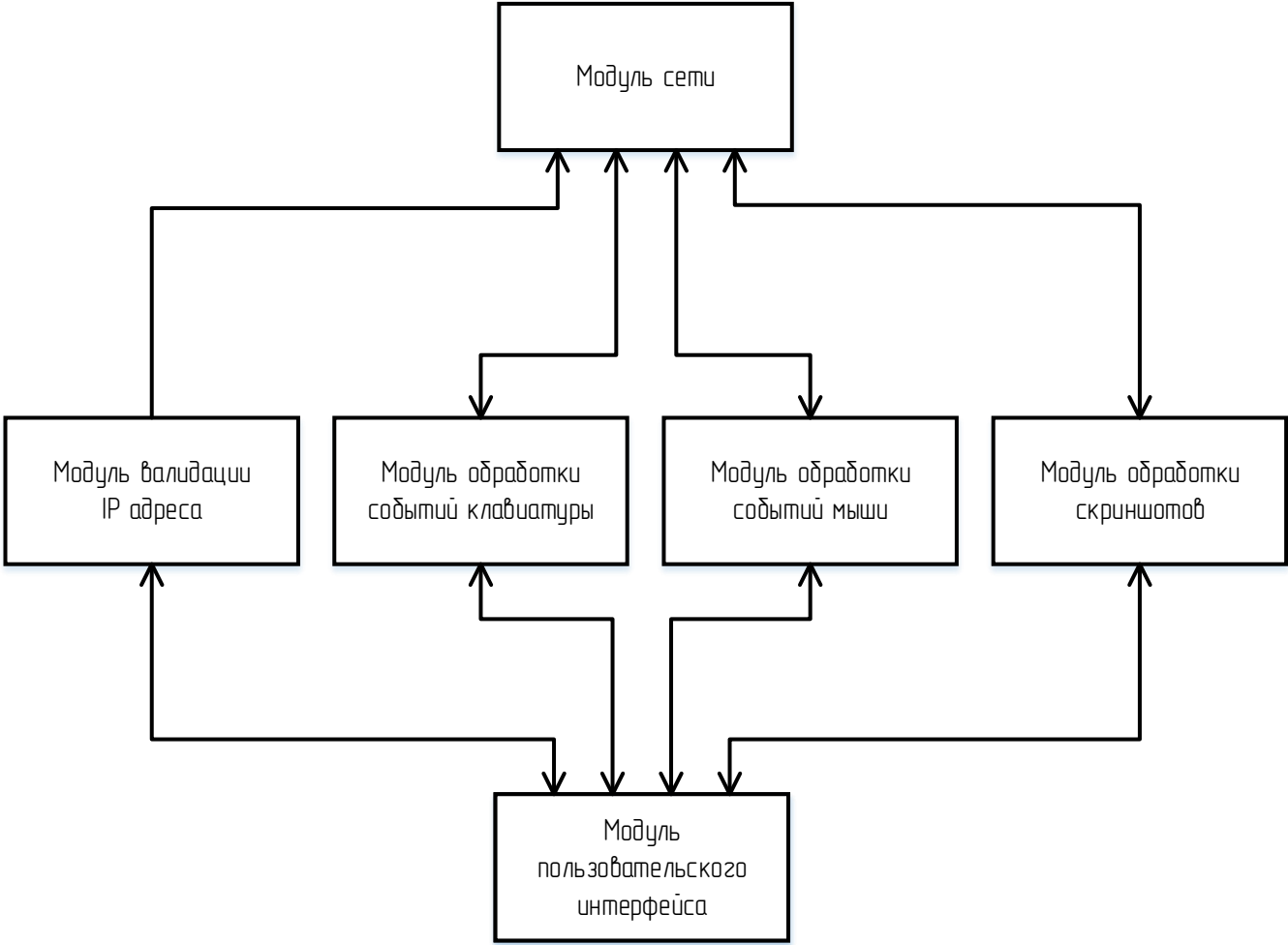


					ГЧИР.400201.111 С1			
Изм	Лист	№ докум	Подп	Дата	Серверная часть. Структурная схема	Лит.	Масса	Масштаб
Разраб.		Климович				Лист 1		Листов 1
Пров.		Поденок				ЭВМ, зр. 150501		

## **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Клиентская часть. Схема структурная**

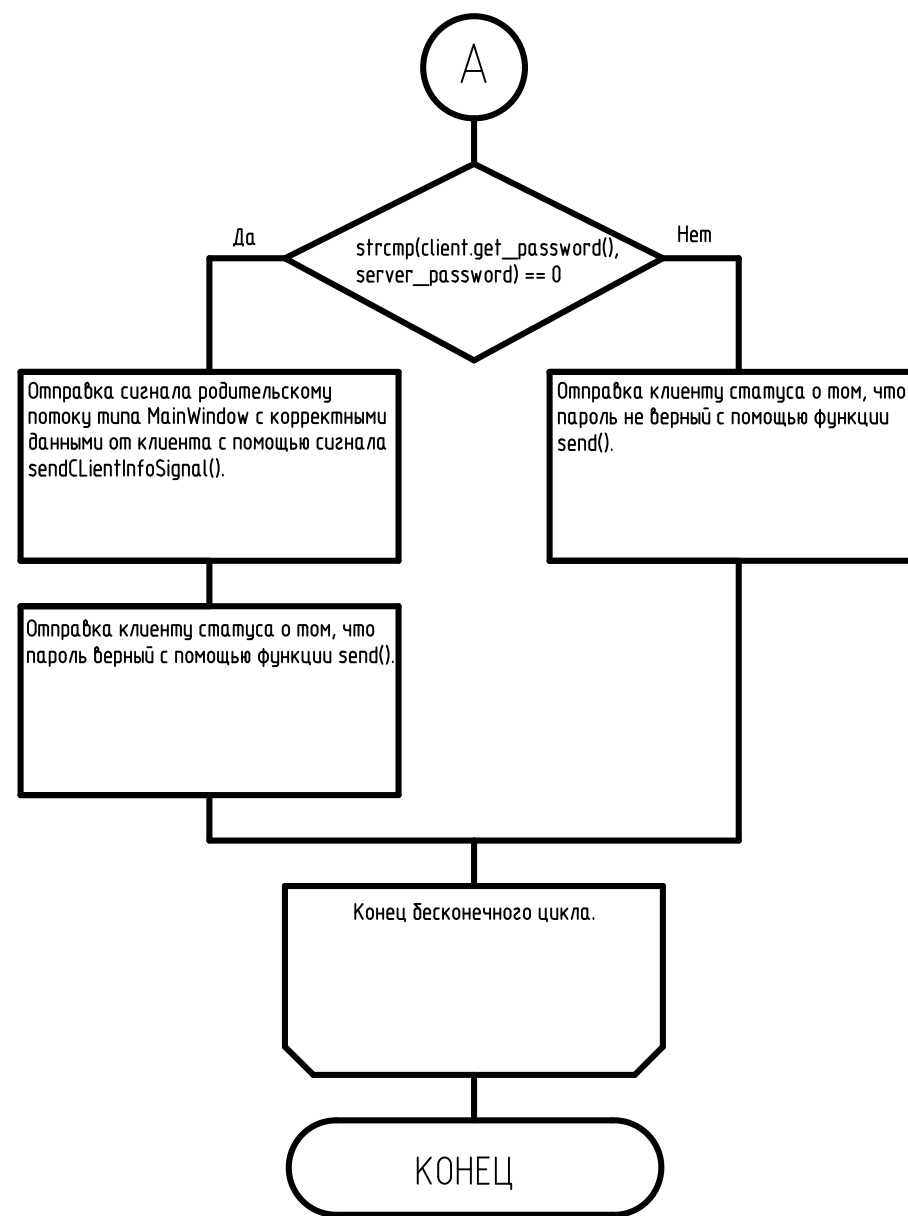
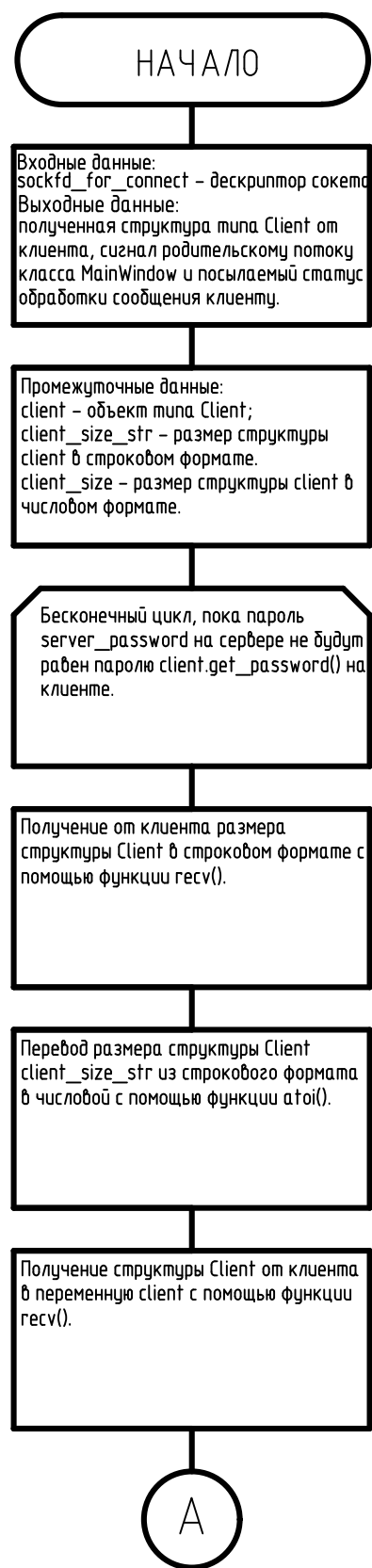


					ГЧИР.400201.111 С2			
Изм.	Лист	№ докум	Подп	Дата	Клиентская часть. Структурная схема	Лит.	Масса	Масштаб
Разраб.		Климович				Лист 1		Листов 1
Пров.		Поденок				ЭВМ, зр. 150501		

## **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Схема алгоритма функции `recvBasicInfoFromClient()`**

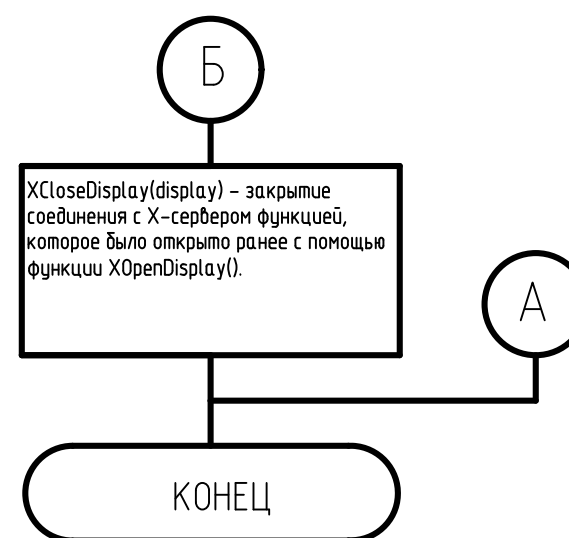
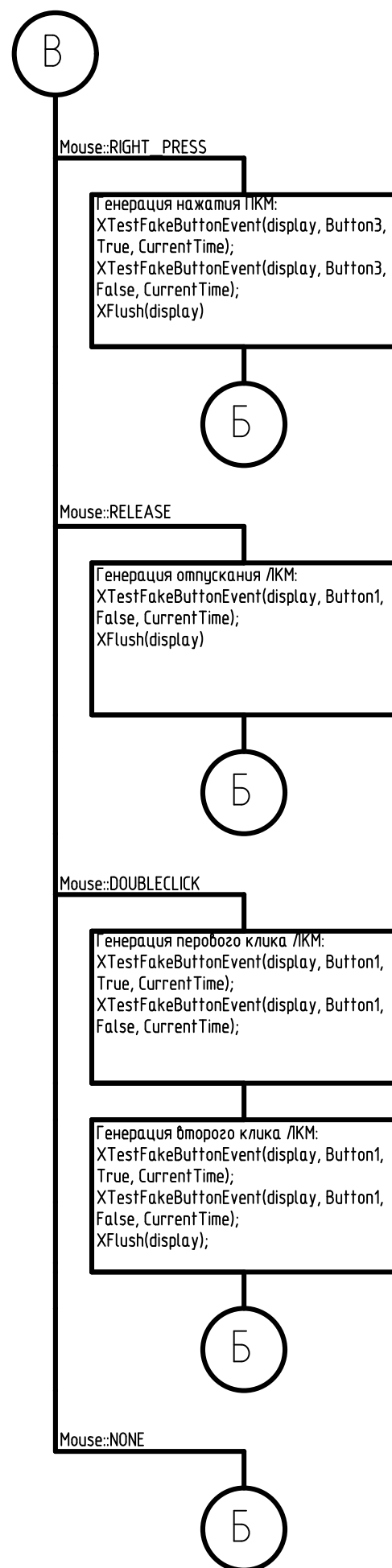
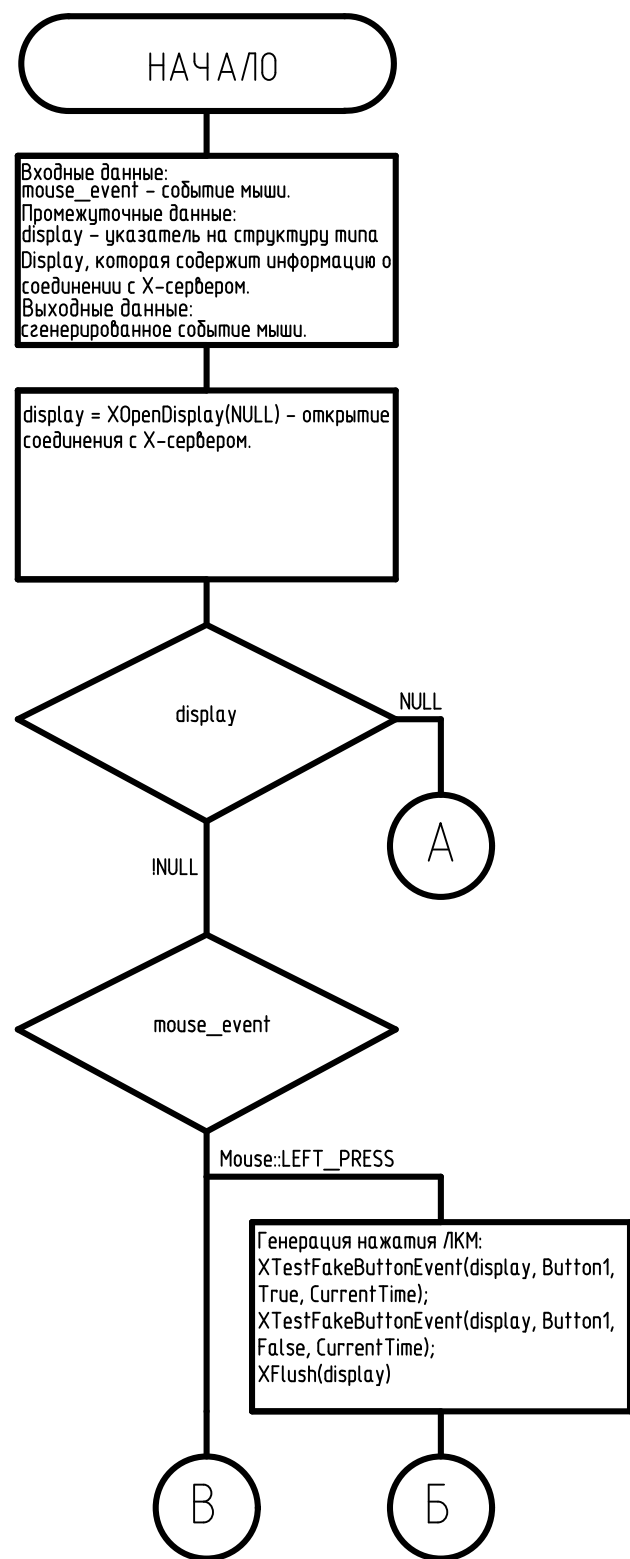


					ГЧИР.400201.104 ПД1			
					Схема функции recvBasicInfoFromClient() получения информации от клиента	Лист.	Масса	Масштаб
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Климович						
Проб.		Поденок						
						Лист 1	Листов 1	
						ЭВМ, зр. 150501		
						Формат А3		

## **ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Схема алгоритма функции `gen_mouse_event( )`**



					ГЧИР.400201.104 ПД2				
					Схема функции gen_mouse_event() генерации события мыши	Лист.	Масса	Масштаб	
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.		Климович							
Пров.		Поденок							
						Лист 1	Листов 1		
						ЭВМ, зр. 150501			



**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Код программы**

```

#ifndef CLIENT_H
#define CLIENT_H

#include <string.h>
#include "config.h"

class Client
{
public:
    char* get_local_host_name();           // получение локального имени хоста
    void set_local_host_name(const char*); // установка локального имени хоста
    char* get_password();                  // получение пароля
    void set_password(const char*);        // установка пароля
    char* get_ip();                         // получение IP
    void set_ip(const char* new_ip);       // установка IP
private:
    char local_host_name[MAX_SIZE_STR] = ""; // локальное имя хоста
    char password[MAX_SIZE_STR] = "";        // пароль
    char ip[MAX_SIZE_STR] = "";              // IP
};

#endif // CLIENT_H

```

//client.cpp:

```

#include "client.h"

char* Client::get_local_host_name() {
    return local_host_name;
}

void Client::set_local_host_name(const char *new_local_host_name) {
    strncpy(local_host_name, new_local_host_name, MAX_SIZE_STR - 1);
}

char* Client::get_password() {
    return password;
}

void Client::set_password(const char* new_password) {
    strncpy(password, new_password, MAX_SIZE_STR - 1);
}

char* Client::get_ip() {
    return ip;
}

void Client::set_ip(const char* new_ip) {
    strncpy(ip, new_ip, MAX_SIZE_STR - 1);
}

```

```

//keyboard.h:
#ifndef KEYBOARD_H
#define KEYBOARD_H

#include <stdio.h>
#include <stdlib.h>

```

```

class Keyboard
{
public:
    enum KEY_EVENT {
        NOT_PRESSED = 0,
        IS_PRESSED = 1
    };
    void generate_key_event(int key_code);           // генерация нажатия клавиши
    int get_event_flag() const;                     // получение признака нажатия
клавиши
    void set_event_flag(int newEvent_flag);         // установка признака нажатия
клавиши
    int get_native_key_code() const;                // получение кода клавиши
    void set_native_key_code(int newNative_key_code); // установка кода клавиши
private:
    int event_flag;           // признак нажатия клавиши
    int native_key_code;     // код клавиши
};

#endif // KEYBOARD_H

```

//keyboard.cpp:

```

#include "keyboard.h"
#include <X11/Xlib.h>
#include <X11/extensions/XTest.h>

void Keyboard::generate_key_event(int key_code)
{
    Display *display = XOpenDisplay(NULL);           // "привязываемся" к экрану
    if (display == NULL) {
        fprintf(stderr, "Cannot open display");
        exit(EXIT_FAILURE);
    }
    // нажимаем клавишу
    XTestFakeKeyEvent(display, XKeysymToKeycode(display, key_code), True, 0);
    // отпускаем клавишу
    XTestFakeKeyEvent(display, XKeysymToKeycode(display, key_code), False, 0);

    XCloseDisplay(display);                           // "отвязка" от экрана
}

int Keyboard::get_event_flag() const {
    return event_flag;
}

void Keyboard::set_event_flag(int new_event_flag) {
    event_flag = new_event_flag;
}

int Keyboard::get_native_key_code() const {
    return native_key_code;
}

void Keyboard::set_native_key_code(int new_native_key_code) {
    native_key_code = new_native_key_code;
}

```

```

//mouse.h:

#ifndef MOUSE_H
#define MOUSE_H

#include <stdio.h>
#include <stdlib.h>

class Mouse
{
public:
    enum MOUSE_EVENT
    {
        NONE = 0,                // никакая кнопка мыши не нажата
        LEFT_PRESS = 1,          // нажатие ЛКМ
        RIGHT_PRESS = 2,         // нажатие ПКМ
        RELEASE = 3,             // отпущание ЛКМ
        DOUBLECLICK = 4,         // двойной клик ЛКМ
        MOVE = 5                 // движение мышки
    };
    void getMousePos(int** pos);    // получение позиции мышки
    void moveMouse(int* newPos);   // передвинуть мышь
    void generate_mouse_event(int mouse_event); // сгенерировать событие мыши
    int getEvent_flag() const;     // получение события мыши
    void setEvent_flag(int newEvent_flag); // установка события мыши
private:
    int event_flag;               // событие мыши
};

#endif // MOUSE_H

//mouse.cpp:

#include "mouse.h"
#include <X11/Xlib.h>
#include <X11/extensions/XTest.h>

void Mouse::getMousePos(int** pos)
{
    Display *display = XOpenDisplay(NULL);
    if (display == NULL) {
        fprintf(stderr, "Cannot open display");
        exit(EXIT_FAILURE);
    }
    Window root = DefaultRootWindow(display);
    int rootX, rootY, winX, winY;
    unsigned int mask;

    XQueryPointer(display, root, &root, &root, &rootX, &rootY, &winX, &winY, &mask);

    (*pos)[0] = rootX; // Переменные rootX и rootY будут содержать координаты
    (*pos)[1] = rootY; // указателя мыши относительно корневого окна экрана
                        // (обычно это экранная панель или рабочий стол).
    (*pos)[2] = winX;  // winX и winY - относительно окна, на котором
    (*pos)[3] = winY;  // находится указатель
    XCloseDisplay(display);
}

```

```

void Mouse::moveMouse(int* newPos)
{
    Display *display = XOpenDisplay(NULL);
    if (display == NULL) {
        fprintf(stderr, "Cannot open display");
        exit(EXIT_FAILURE);
    }
    Window root = DefaultRootWindow(display);

    XWarpPointer(display, None, root, 0, 0, 0, 0, newPos[0], newPos[1]);
    XFlush(display); // необходимо вызвать XFlush для того, чтобы изменения
    XCloseDisplay(display); // применились немедленно
}

```

```

void Mouse::generate_mouse_event(int mouse_event)
{
    Display *display = XOpenDisplay(NULL);
    if (display == NULL) {
        fprintf(stderr, "Cannot open display");
        exit(EXIT_FAILURE);
    }

    switch(mouse_event)
    {
        case Mouse::NONE: break;
        case Mouse::LEFT_PRESS : // генерация нажатия ЛКМ
            XTestFakeButtonEvent(display, Button1, True, CurrentTime);
            XTestFakeButtonEvent(display, Button1, False, CurrentTime);
            XFlush(display);
            break;
        case Mouse::RIGHT_PRESS: // генерация нажатия ПКМ
            XTestFakeButtonEvent(display, Button3, True, CurrentTime);
            XTestFakeButtonEvent(display, Button3, False, CurrentTime);
            XFlush(display);
            break;
        case Mouse::RELEASE: // генерация отпускания ЛКМ
            XTestFakeButtonEvent(display, Button1, False, CurrentTime);
            XFlush(display);
            break;
        case Mouse::DOUBLECLICK: // генерация двойного клика ЛКМ
            XTestFakeButtonEvent(display, Button1, True, CurrentTime);
            XTestFakeButtonEvent(display, Button1, False, CurrentTime);
            XTestFakeButtonEvent(display, Button1, True, CurrentTime);
            XTestFakeButtonEvent(display, Button1, False, CurrentTime);
            XFlush(display);
            break;
    }

    XCloseDisplay(display);
}

```

```

int Mouse::getEvent_flag() const { return event_flag; }

```

```

void Mouse::setEvent_flag(int newEvent_flag) { event_flag = newEvent_flag; }

```

```

//sockets.h:

```

```

#ifndef SOCKETS_FUNCTIONS_H
#define SOCKETS_FUNCTIONS_H

```

```

#include <sys/types.h>
#include <sys/socket.h>

```

```

#include <netinet/in.h>
#include <arpa/inet.h>
#include <ifaddrs.h>
#include <netdb.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string.h>
#include <string>

#include <QDebug>

#include "config.h"

int Socket(int domain, int type, int protocol);    // создать новый сокет и вернуть
                                                    // файловый дескриптор

void Bind(int, const struct sockaddr*, socklen_t); // связать сокет с IP-адресом
                                                    // и портом
void Listen(int sockfd, int backlog);    // объявить о желании принимать соединения,
                                                    // слушает порт и ждет, когда будет
                                                    // установлено соединение
int Accept(int, struct sockaddr*, socklen_t*);    // принять запрос на установку
                                                    // соединения
int Connect(int, const struct sockaddr*, socklen_t); // установить соединение
void Inet_pton(int, const char*, void*); // преобразование адреса IPv4 и IPv6 из
                                                    // текста в бинарный вид
void Close(int &fd); // закрыть файловый дескриптор (сокет)
void send_file(int* sock, const char* file_name); // отправить файл по сети
void recv_file(int* sock, int *status); // получить файл из сети

#endif // SOCKETS_FUNCTIONS_H

```

//sockets.cpp:

```

#include "sockets_functions.h"
int Socket(int domain, int type, int protocol) {
    int res_sockfd = socket(domain, type, protocol);
    if (res_sockfd == -1) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Установка опции SO_REUSEADDR для повторного использования адреса
    int opt = 1;
    if (setsockopt(res_sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0)
    {
        // Ошибка установки опции сокета
        perror("Setsockopt failed");
        exit(EXIT_FAILURE);
    }

    // Установка тайм-аута в 10 секунд
    struct timeval tv;
    tv.tv_sec = TIMEOUT; // TIMEOUT = 10 sec
    tv.tv_usec = 0;
}

```

```

    if (setsockopt(res_sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof(tv)) < 0)
    {
        // Ошибка установки опции сокета
        perror("Setsockopt failed");
        exit(EXIT_FAILURE);
    }

    return res_sockfd;
}

void Bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen) {
    int res = bind(sockfd, addr, addrlen);
    if (res == -1) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
}

void Listen(int sockfd, int backlog) {
    int res = listen(sockfd, backlog);
    if (res == -1) {
        perror("Listen failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }
}

int Accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen) {
    int res = accept(sockfd, addr, addrlen);
    if (res == -1) {
        if (errno == EWOULDBLOCK || errno == EAGAIN) {
            // Истек тайм-аут, соединение не установлено
            errno = 0;
            return 0;
        } else {
            // Ошибка принятия входящего соединения
            perror("Accept failed");
            close(sockfd);
            exit(EXIT_FAILURE);
        }
    }
    return res;
}

int Connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen) {
    int res = connect(sockfd, addr, addrlen);
    if (res == -1) {
        perror("Connect failed");
        Close(sockfd);
        //exit(EXIT_FAILURE);
    }
    return res;
}

void Inet_pton(int af, const char *src, void *dst) {
    int res = inet_pton(af, src, dst);
    if (res == 0) {
        perror("Inet_pton failed: src does not contain a character"
            " string representing a valid network address in the specified"
            " address family\n");
        exit(EXIT_FAILURE);
    }
}

```

```

        if (res == -1) {
            perror("Inet_pton failed");
            exit(EXIT_FAILURE);
        }
    }

void Close(int& fd)
{
    shutdown(fd, SHUT_RDWR);
    close(fd);
}

void send_file(int* sock, const char* file_name)
{
    std::fstream file;
    file.open(file_name, std::ios_base::in | std::ios_base::binary);

    if(file.is_open())
    {
        int file_size = std::filesystem::file_size(file_name) + 1;
        char* bytes = (char*)malloc(file_size*sizeof(char));

        file.read(bytes, file_size);

        qDebug("size: %d", file_size);
        qDebug("name: %s", file_name);
        qDebug("data: %s", bytes);

        send(*sock, std::to_string(file_size).c_str(), MAX_SIZE_INT_STR, 0);
        send(*sock, file_name, MAX_SIZE_STR, 0);
        send(*sock, bytes, file_size, 0);

        free(bytes);
    }
    else
        perror("Error file open\n");
    file.close();
}

void recv_file(int* sock, int *status)
{
    char file_size_str[MAX_SIZE_INT_STR];
    char file_name[MAX_SIZE_STR];

    recv(*sock, file_size_str, MAX_SIZE_INT_STR, MSG_WAITALL);
    int file_size = atoi(file_size_str);
    char* bytes = (char*)malloc(file_size*sizeof(char));

    recv(*sock, file_name, MAX_SIZE_STR, MSG_WAITALL);

    qDebug("size: %d", file_size);
    qDebug("name: %s", file_name);

    if(file_size == 0) // сервер был отключен
    {
        *status = 0;
        free(bytes);
        return;
    }

    std::fstream file;

```





```
#endif // MAINWINDOW_H
```

```
//mainwindow.cpp:
```

```
#include "mainwindow.h"  
#include "ui_mainwindow.h"  
#include <QThread>
```

```
MainWindow::MainWindow(QWidget *parent)  
    : QMainWindow(parent)  
    , ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
  
    // Соединение сигналов и слотов  
    connect(&serverThread, &ServerThread::updateCursorSignal, this,  
    &MainWindow::updateCursorSlot);  
    connect(&serverThread, &ServerThread::sendClientInfoSignal, this,  
    &MainWindow::showClientInfoSlot);  
    connect(this, &MainWindow::sendPasswordFromUISignal, &serverThread,  
    &ServerThread::receivePasswordFromUISlot);  
    connect(&serverThread, &ServerThread::signalToTurnOff, this,  
    &MainWindow::on_turn_off_pushButton_clicked);  
    connect(&serverThread, &ServerThread::timeIsOverToConnectSignal, this,  
    &MainWindow::timeIsOverToConnectSlot);  
}
```

```
MainWindow::~MainWindow()  
{  
    serverThread.closeThread();  
    serverThread.wait();  
    serverThread.closeSockets();  
    delete ui;  
}
```

```
void MainWindow::on_actionExit_triggered()  
{  
    this->close();  
    this->~MainWindow();  
}
```

```
void MainWindow::on_turn_on_pushButton_clicked()  
{  
    // Установка всех признаков включенного сервера  
    ui->turn_on_pushButton->setEnabled(false);  
    this->statusBar()->showMessage("Is working...");  
    this->setCursor(Qt::BusyCursor);  
  
    // Проверка пароля  
    if(ui->password_lineEdit->text().isEmpty()) {  
        QMessageBox::warning(this, "Warning", "Password field is empty");  
        return;  
    }  
  
    // Отправка пароля в рабочий поток сервера  
    emit sendPasswordFromUISignal(ui->password_lineEdit->text().toString().c_str());  
  
    // Запуск потока сервера  
    serverThread.start();  
}
```

```

void MainWindow::on_turn_off_pushButton_clicked()
{
    // Установка всех признаков выключенного сервера
    this->statusBar()->showMessage("Server is off", STATUSBAR_DELAY);
    ui->turn_on_pushButton->setEnabled(true);
    ui->hostName->setText("...");
    ui->hostName->setAlignment(Qt::AlignCenter);
    ui->clientIP->setText("...");
    ui->clientIP->setAlignment(Qt::AlignCenter);

    // Смена рисунка курсора
    this->setCursor(Qt::ArrowCursor);

    // Закрытие потока
    serverThread.closeThread();
}

void MainWindow::updateCursorSlot(int cursor_type)
{
    switch(cursor_type)
    {
        case 0: this->setCursor(Qt::ArrowCursor); break; // курсор "Стрелка"
        case 1: this->setCursor(Qt::BusyCursor); break; // курсор "Загрузка"
    }
}

void MainWindow::showClientInfoSlot(QString local_host_name, const char* ip)
{
    qDebug("Received hostname from client = %s",
    local_host_name.toStdString().c_str());
    qDebug("Received IP from client = %s\n", ip);
    ui->hostName->setText(local_host_name);
    ui->clientIP->setText(ip);
}

void MainWindow::timeIsOverToConnectSlot()
{
    QMessageBox::warning(this, "Warning", "Connection timed out");
    on_turn_off_pushButton_clicked();
    serverThread.closeThread();
    serverThread.wait();
}

//screenshot.h:

#ifndef SCREENSHOT_H
#define SCREENSHOT_H

#include <QApplication>
#include <QPixmap>
#include <QScreen>
#include <QBuffer>
#include <QCursor>
#include <QPoint>
#include <QPainter>

class Screenshot
{
public:
    void take_screenshot(const char* screenshot_pathname); // сделать скриншот
    void save_in_buffer(QBuffer* buffer); // сохранить скриншот в буфер

```

```

private:
    QScreen* screen;        // для захвата экрана
    QPixmap screenshot;     // переменная для хранения скриншота
};

#endif // SCREENSHOT_H

//screenshot.cpp:
#include "screenshot.h"
void Screenshot::take_screenshot(const char* screenshot_pathname)
{
    // Захват экрана
    screen = QApplication::primaryScreen();
    screenshot = screen->grabWindow(0);

    // Рисуем изображение курсора поверх скриншота
    QPainter painter(&screenshot);
    QPen pen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap);
    painter.setPen(pen);
    painter.setBrush(Qt::NoBrush);
    QPoint mousePos = QCursor::pos();
    int radius = 10;
    painter.drawEllipse(mousePos, radius, radius);
    painter.end();

    // Сохраняем изображение скриншота с курсором в файл
    if(screenshot.save(screenshot_pathname) == false)
    {
        fprintf(stderr, "Error: screenshot is not created");
        exit(EXIT_FAILURE);
    }
}

void Screenshot::save_in_buffer(QBuffer* buffer)
{
    QByteArray bytes;
    buffer->setBuffer(&bytes);
    buffer->open(QIODevice::WriteOnly);
    if(screenshot.save(buffer, "PNG") == false)
        fprintf(stderr, "Error: screenshot is not saved to the buffer");
}

//serverthread.h:
#ifndef SERVERTHREAD_H
#define SERVERTHREAD_H

#include <QApplication>
#include <QThread>
#include <QString>

#include "sockets_functions.h"
#include "screenshot.h"
#include "client.h"
#include "mouse.h"
#include "keyboard.h"

class ServerThread : public QThread {
    Q_OBJECT

```

```

signals:
    void updateCursorSignal(int cursor_type); // сигнал для обновления рисунка курсора
    void sendClientInfoSignal(QString s, char* ip); // сигнал для передачи
                                                    // информации на экран
    void signalToTurnOff(); // сигнал выключить сервер
    void timeIsOverToConnectSignal(); // сигнал о том, что время ожидания подключения вышло

private:
    struct sockaddr_in client_addr; // клиент
    int sockfd_for_listen; // прослушивающий сокет
    int sockfd_for_connect; // рабочий сокет
    int server_status; // статус сервера (вкл/выкл)
    char server_password[MAX_SIZE_STR]; // пароль на сервере

public:
    explicit ServerThread(QObject *parent = nullptr); // конструктор
    void run() override; // запуск потока
    void closeThread(); // закрытие потока
    void closeSockets(); // закрытие сокетов
    void receiveBasicInfoFromClient(); // получение базовой информации от клиента

public slots:
    void sendScreenshot(); // отправка скриншота на экран
    void receivePasswordFromUISlot(const char*); // получение пароля с UI
};

#endif // SERVERTHREAD_H

#include "serverthread.h"
ServerThread::ServerThread(QObject *parent) : QThread(parent)
{
    // Сервер выключен
    server_status = false;

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

    // Создаем прослушивающий сокет:
    sockfd_for_listen = Socket(AF_INET, SOCK_STREAM, 0);
    qDebug("Сокет создан");

    // Связываем сокеты:
    Bind(sockfd_for_listen, (struct sockaddr*)&client_addr, sizeof client_addr);
    qDebug("Связь установлена");
}

void ServerThread::closeThread(){
    this->server_status = 0;
}

void ServerThread::closeSockets() {
    Close(sockfd_for_connect);
    Close(sockfd_for_listen);
}

void ServerThread::run()
{
    // Сервер включен
    server_status = 1;

```

```

socklen_t addrlen = sizeof(client_addr);

// Слушаем:
Listen(sockfd_for_listen, 1); // backlog = 1
qDebug("Слушаем...");

emit updateCursorSignal(1); // тип курсора: loading

// Подключаемся:
sockfd_for_connect = Accept(sockfd_for_listen, (struct sockaddr*)&client_addr, &addrlen);
if(sockfd_for_connect == 0) {
    emit timeIsOverToConnectSignal();
    return;
}
qDebug("Клиент успешно подключен\n");

emit updateCursorSignal(0); // тип курсора: normal

// Принимаем информацию от клиента - его localhostName и введенный пароль
receiveBasicInfoFromClient();

// Отправляем скриншоты:
int i = 0;
char response[2];
Mouse mouse;
Keyboard keyboard;
while(server_status)
{
    // Отправка очередного скриншота
    sendScreenshot();

    // Получение ответа от клиента
    recv(sockfd_for_connect, response, 2, MSG_WAITALL);
    if(strcmp(response, "") == 0) {
        server_status = 0;
    }
    qDebug("RESPONSE = %s", response);

    // Получение координат мышки
    char message[32], len[16];
    int* pos = (int*)malloc(2 * sizeof(int)); // [0] = x, [1] = y
    memset(message, 0, sizeof(message));
    recv(sockfd_for_connect, len, 16, MSG_WAITALL);
    recv(sockfd_for_connect, message, atoi(len), MSG_WAITALL);
    sscanf(message, "%d %d", &pos[0], &pos[1]);

    // Обработка координат мыши
    mouse.moveMouse(pos);
    free(pos);

    // Получение события мышки
    char mouse_event_str;
    recv(sockfd_for_connect, &mouse_event_str, 1, MSG_WAITALL);
    int mouse_event = mouse_event_str - '0';
    qDebug() << "MOUSE EVENT: " << mouse_event;
    mouse.generate_mouse_event(mouse_event);

    // Получение события клавиатуры
    strcpy(message, "");
    recv(sockfd_for_connect, message, 16, MSG_WAITALL);
    qDebug() << "KEY CODE: " << message;
}

```

```

        if(strcmp(message, "0") != 0)
        {
            int key_code = atoi(message);
            keyboard.generate_key_event(key_code);
            keyboard.set_event_flag(Keyboard::NOT_PRESSED);
        }

        msleep(1000/FPS);
        qDebug("iterations = %d", i);
        strcpy(response, "");
        i++;
    }

    // Сервер был выключен
    emit signalToTurnOff();
}

void ServerThread::sendScreenshot()
{
    Screenshot screenshot;
    if(server_status)
    {
        screenshot.take_screenshot(FILENAME);
        send_file(&sockfd_for_connect, FILENAME);
        qDebug("Скриншот отправлен\n");
    }
}

void ServerThread::receiveBasicInfoFromClient()
{
    Client client;
    char client_size_str[MAX_SIZE_INT_STR];
    int client_size;

    while(1)
    {
        recv(sockfd_for_connect, client_size_str, MAX_SIZE_INT_STR, MSG_WAITALL);
        client_size = atoi(client_size_str);
        qDebug("client struct size = %d", client_size);

        recv(sockfd_for_connect, &client, client_size, MSG_WAITALL);
        qDebug("client password = %s", client.get_password());
        qDebug("client ip = %s", client.get_ip());
        qDebug("client local host name = %s\n", client.get_local_host_name());

        if(strcmp(client.get_password(), server_password) == 0)
        {
            QString local_host_name(client.get_local_host_name());
            emit sendClientInfoSignal(local_host_name, client.get_ip());
            send(sockfd_for_connect, "1", 1, 0); //1 - all good;
            break;
        }
        else
            send(sockfd_for_connect, "0", 1, 0); //0 - all bad;
    }
}

void ServerThread::receivePasswordFromUISlot(const char* password)
{
    qDebug("password from ui = %s", password);
    strncpy(server_password, password, MAX_SIZE_STR - 1);
}

```

```

//main:

/*
 *  REMOTE CONTROL SERVER
 *
 *  Created by Alexei Klimovich on 16.04.2023.
 *  Copyright © 2023 Alexei Klimovich. All rights reserved.
 *
 */

#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Remote Control Server");
    w.show();
    return a.exec();
}

//clientthread.h:

#ifndef CLIENTTHREAD_H
#define CLIENTTHREAD_H

#include <QApplication>
#include <QMainWindow>
#include <QThread>
#include <QHostInfo>

#include "dialog.h"
#include "sockets_functions.h"
#include "client.h"
#include "mouse.h"
#include "keyboard.h"

class ClientThread : public QThread
{
    Q_OBJECT

signals:
    void screenshotReady(QImage screenshot);    // сигнал готовности скриншота вывода
                                                // на экран
    void showPasswordStatusSignal(int status); // сигнал для показа статуса
                                                // правильности пароля
    void signalToDisconnect();                 // сигнал для разрыва соединения
    void connectFailedSignal();                 // сигнал о невозможности подключения

private:
    struct sockaddr_in server_addr;             // сервер
    int sockfd_for_connect;                     // рабочий сокет
    int client_status;                          // статус клиента (вкл/выкл)
    DataConnection dataConnection;             // информация для создания соединения
    Mouse mouse;                               // мышь
    Keyboard keyboard;                          // клавиатура

public:
    explicit ClientThread(QObject* parent = nullptr); // конструктор
    ~ClientThread();                                // деструктор
    void run() override;                           // запуск потока
    void closeThread();                             // закрытие потока

```



```

        void receiveScreenshot();           // получение скриншота
        void sendClientInfoToServer();      // отправка клиентской информации серверу

public slots:
    void receiveSettingsFromUISlot(DataConnection data); // получение введенных
                                                         // настроек с UI
    void getMouseEventFromWindowSlot(int event_pos);      // получение события мыши
    void getKeyEventFromWindowSlot(int is_pressed, int key_code); // получения события
                                                         // клавиатуры
};

#endif // CLIENTTHREAD_H

//clientthread.cpp:

#include "clientthread.h"
#include <X11/Xlib.h>
#include <X11/extensions/XTest.h>
#include "mouse.h"

ClientThread::ClientThread(QObject* parent)
    : QThread(parent)
{
    // Клиент не подключен
    client_status = false;

    mouse.setEvent_flag(Mouse::NONE);
    keyboard.set_event_flag(Keyboard::NOT_PRESSED);

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr =
inet_addr(dataConnection.server_ip.toStdString().c_str()); // server IP
}

ClientThread::~ClientThread() {
    closeThread();
    Close(sockfd_for_connect);
}

void ClientThread::closeThread(){
    this->client_status = 0;
}

void ClientThread::receiveSettingsFromUISlot(DataConnection data) {
    dataConnection.password = data.password;
    dataConnection.server_ip = data.server_ip;
}

void ClientThread::receiveScreenshot() {
    QPixmap screenshot(FILENAME);
    if(client_status)
    {
        recv_file(&sockfd_for_connect, &client_status);
        emit screenshotReady(screenshot.toImage());
        qDebug("Скриншот получен");
    }
}

```

```
void ClientThread::run() {

    // Включение клиента:
    client_status = 1;

    //Настройка и создание сокетов:
    sockfd_for_connect = Socket(AF_INET, SOCK_STREAM, 0);
    Inet_pton(AF_INET, dataConnection.server_ip.toString().c_str(),
    &server_addr.sin_addr);
    int connect_status = Connect(sockfd_for_connect, (struct sockaddr*)&server_addr,
    sizeof(server_addr));
    if(connect_status == -1)
    {
        emit connectFailedSignal();
        emit signalToDisconnect();
        return;
    }

    //Отправка информации о клиенте серверу
    sendClientInfoToServer();

    Display* display = XOpenDisplay(NULL);
    if (display == NULL) {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
    }

    // Обработка скриншотов
    int i = 0;
    char message[100];
    int* pos = (int*)malloc(4 * sizeof(int)); //позиция мышки
    while(client_status)
    {
        // Получение очередного скриншота
        receiveScreenshot();

        // Отправка статуса, что скриншот получен
        send(sockfd_for_connect, "OK", 2, 0);

        // Получение координат мыши
        mouse.getMousePos(&pos); // rootX = pos[0], rootY = pos[1], winX = pos[2],
        // winY = pos[3]
        sprintf(message, "%d %d", pos[0], pos[1]);
        send(sockfd_for_connect, std::to_string(strlen(message)).c_str(), 16, 0);
        send(sockfd_for_connect, message, strlen(message), 0);
        strcpy(message, "");

        // Отправка события мыши
        char mouse_event = mouse.getEvent_flag() + '0';
        send(sockfd_for_connect, &mouse_event, 1, MSG_NOSIGNAL);
        mouse_event = '0';
        mouse.setEvent_flag(Mouse::NONE);

        // Отправка события клавиатуры
        if(keyboard.get_event_flag() == Keyboard::IS_PRESSED) {
            sprintf(message, "%d", keyboard.get_native_key_code());
            send(sockfd_for_connect, message, 16, 0);
            keyboard.set_event_flag(Keyboard::NOT_PRESSED);
        } else {
            send(sockfd_for_connect, "0", 16, 0); // отправляем, что никакая клавиша
            // не нажата
        }
    }
}
```

```

        msleep(1000/FPS);
        qDebug("iterations = %d", i);
        i++;
    }

    // Соединение прервано:
    Close(sockfd_for_connect);
    XCloseDisplay(display);
    free(pos);
    emit signalToDisconnect();
}

void ClientThread::sendClientInfoToServer()
{
    Client client;
    QHostInfo info;
    QByteArray data = info.localHostName().toUtf8();

    client.set_local_host_name(data.toStdString().c_str());
    client.set_password(dataConnection.password.toStdString().c_str());
    client.set_ip(inet_ntoa(server_addr.sin_addr));

    qDebug() << "Local host=" << data;
    qDebug() << "password=" << client.get_password();
    qDebug() << "ip=" << client.get_ip();

    unsigned char* client_ptr = (unsigned char*)&client;
    send(sockfd_for_connect, std::to_string(sizeof(client)).c_str(), MAX_SIZE_INT_STR,
MSG_NOSIGNAL);
    qDebug("length of Client struct was sent");
    send(sockfd_for_connect, client_ptr, sizeof(client), MSG_NOSIGNAL);
    qDebug("Client struct was sent");

    // Получаем статус пароля от сервера:
    char password_status;
    recv(sockfd_for_connect, &password_status, 1, MSG_WAITALL);
    if(password_status == '1') {
        emit showPasswordStatusSignal(1); // all good
    } else {
        emit showPasswordStatusSignal(0); // all bad
        closeThread();
        Close(sockfd_for_connect);
    }
}

void ClientThread::getMouseEventFromWindowSlot(int event_flag)
{
    qDebug() << "Event flag = " << event_flag;
    mouse.setEvent_flag(event_flag);
}

void ClientThread::getKeyEventFromWindowSlot(int is_pressed, int key_code)
{
    qDebug() << "is_pressed = " << is_pressed << " key_code = " << key_code;
    keyboard.set_event_flag(is_pressed);
    keyboard.set_native_key_code(key_code);
}

```

```

//dialog.h:

#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QMessageBox>

#include "ui_dialog.h"
#include "validator.h"

struct DataConnection // структура для установки соединения
{
    QString server_ip; // IP сервера
    QString password; // пароль на сервере
};

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = nullptr); // конструктор
    ~Dialog(); // деструктор

    bool isValidParametres(); // проверка валидности настроек
    DataConnection getData() const; // получить структуру для создания соединения
    void setDataFromUI(); // инициализировать структуру введенными данными с UI

private slots:
    void on_buttonBox_accepted(); // сохранить изменения
    void on_buttonBox_rejected(); // отменить изменения

private:
    Ui::Dialog *ui; // пользовательский интерфейс
    Validator validator; // валидатор настроек
    DataConnection data; // структура для создания сети
};

#endif // DIALOG_H

//dialog.cpp:

#include "dialog.h"
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
}

Dialog::~Dialog() {
    delete ui;
}

```

```
bool Dialog::isValidParametres() {
    setDataFromUI();
    if(validator.is_valid_IP(data.server_ip)) {
        return true;
    } else
        return false;
}

void Dialog::on_buttonBox_accepted()
{
    if(isValidParametres())
        QMessageBox::information(this, "Information", "Settings have been saved");
    else
        QMessageBox::warning(this, "Warning", "Data entered incorrectly");
}

void Dialog::on_buttonBox_rejected() {
    this->close();
}

void Dialog::setDataFromUI()
{
    data.server_ip = ui->IP_lineEdit->text();
    data.password = ui->password_lineEdit->text();
}

DataConnection Dialog::getData() const {
    return data;
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include "dialog.h"
#include "clientthread.h"
#include "ui_mainwindow.h"
#include "mouse.h"
#include "keyboard.h"
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

[illegible]

```

protected:
    void mousePressEvent(QMouseEvent* event) override;    // перегрузка встроенных
                                                         // методов для отлавливания
                                                         // событий мыши и клавиатуры

    void mouseReleaseEvent(QMouseEvent* event) override;
    void mouseMoveEvent(QMouseEvent* event) override;
    void mouseDoubleClickEvent(QMouseEvent* event) override;
    void keyPressEvent(QKeyEvent *event) override;

public:
    MainWindow(QWidget *parent = nullptr);              // конструктор
    ~MainWindow();                                       // деструктор

private slots:
    void on_actionExit_triggered();                     // выход из приложения
    void on_actionSettings_triggered();                 // зайти в настройки
    void on_actionConnect_to_server_triggered();        // подсоединиться к серверу
    void on_actionDisconnect_from_the_server_triggered(); // отсоединиться от сервера

public slots:
    void updateScreenshot(QImage screenshot);          // обновить скриншот на экране
    void showPasswordStatusSlot(int status);           // показать статус правильности пароля
    void connectFailedSlot();                          // не удалось соединиться с сервером
private:
    Ui::MainWindow *ui;                               // пользовательский интерфейс
    Dialog settings;                                   // настройки
    ClientThread clientThread;                         // рабочий поток клиента
};
#endif // MAINWINDOW_H

// mainwindow.cpp

#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    settings.setDataFromUI(); // получение настроек с UI

// Соединение сигналов и слотов
    connect(this, &MainWindow::sendSettingsFromUISignal, &clientThread,
&ClientThread::receiveSettingsFromUISlot);
    connect(&clientThread, &ClientThread::showPasswordStatusSignal, this,
&MainWindow::showPasswordStatusSlot);
    connect(&clientThread, &ClientThread::signalToDisconnect, this,
&MainWindow::on_actionDisconnect_from_the_server_triggered);
    connect(&clientThread, &ClientThread::screenshotReady, this,
&MainWindow::updateScreenshot);
    connect(&clientThread, &ClientThread::connectFailedSignal, this,
&MainWindow::connectFailedSlot);
    connect(this, &MainWindow::mouseEventToClientThreadSignal, &clientThread,
&ClientThread::getMouseEventFromWindowSlot);
    connect(this, &MainWindow::keyIsPressed, &clientThread,
&ClientThread::getKeyEventFromWindowSlot);
}

MainWindow::~MainWindow() {
    clientThread.closeThread();
    clientThread.wait();
    delete ui;
}

```

```

void MainWindow::on_actionExit_triggered() {
    this->close();
}

void MainWindow::on_actionSettings_triggered() {
    settings.setWindowTitle("Settings");
    settings.show();
    settings.exec();
}

void MainWindow::updateScreenshot(QImage screenshotImage) {
    QPixmap screenshot = QPixmap::fromImage(screenshotImage);
    ui->remoteScreen_label->setScaledContents(true);
    QPixmap scaledPixmap = screenshot.scaled(ui->remoteScreen_label->size(),
Qt::KeepAspectRatio);
    ui->remoteScreen_label->setPixmap(scaledPixmap);
}

void MainWindow::on_actionConnect_to_server_triggered()
{
    //Проверка валидности настроек для подключения
    if(settings.isValidParametres() == false)
    {
        QMessageBox::warning(this, "Warning", "Incorrect settings data entered");
        return;
    }
    if(settings.getData().password.isEmpty())
    {
        QMessageBox::warning(this, "Warning", "Password field is empty");
        return;
    }

    // Включение клиента
    ui->actionConnect_to_server->setEnabled(false);
    ui->statusbar->showMessage("Is working...");

    // Отправка настроек для связи
    emit sendSettingsFromUISignal(settings.getData());

    // Запуск потока для клиента
    clientThread.start();

    // Спрятать лишние менюшки
    this->setWindowState(Qt::WindowFullScreen);
    this->menuWidget()->setVisible(false);
    this->menuBar()->setVisible(false);
}

void MainWindow::on_actionDisconnect_from_the_server_triggered() {
    // Установка всех признаков отсоединенного клиента от сервера
    ui->actionConnect_to_server->setEnabled(true);
    ui->remoteScreen_label->clear();
    ui->remoteScreen_label->setText("(Here is a remote screen)");
    ui->remoteScreen_label->setStyleSheet("font: italic");
    ui->remoteScreen_label->setAlignment(Qt::AlignCenter);
    this->statusBar()->showMessage("Connection interrupted", STATUSBAR_DELAY);

    // Закрытие потока
    clientThread.closeThread();
    clientThread.wait();
}

```

```

        // Изменение размеров окна
        this->setWindowState(Qt::WindowMaximized);
        this->menuBar()->setVisible(true);
        this->menuWidget()->setVisible(true);
        this->resize(800, 600);
    }

void MainWindow::showPasswordStatusSlot(int status)
{
    if(status == 1) {
        QMessageBox::information(this, "Success", "You are connected
successfully.\nPress F12 to show the menu.");
    } else {
        QMessageBox::warning(this, "Failure", "Wrong password");
        on_actionDisconnect_from_the_server_triggered();
    }
}

void MainWindow::connectFailedSlot() {
    QMessageBox::critical(this, "Error", "Connect failed\nTry again");
    on_actionDisconnect_from_the_server_triggered();
}

void MainWindow::keyPressEvent(QKeyEvent *event)
{
    QString keyText = event->text();
    qDebug() << "\nKEY = " << keyText << " qt_key_code=" << event->key() << "
native_key_code=" << event->nativeVirtualKey() << "\n";
    emit keyIsPressed(Keyboard::IS_PRESSED, event->nativeVirtualKey());

    // Обработка символа, связанного с нажатой клавишей
    if (event->key() == Qt::Key_F12) {
        if(this->windowState() == Qt::WindowFullScreen) {
            this->setWindowState(Qt::WindowMaximized);
            this->menuBar()->setVisible(true);
            this->menuWidget()->setVisible(true);
            this->resize(800, 600);
        } else {
            this->setWindowState(Qt::WindowFullScreen);
            this->menuWidget()->setVisible(false);
            this->menuBar()->setVisible(false);
        }
    } else
        QWidget::keyPressEvent(event);
}

void MainWindow::mousePressEvent(QMouseEvent* event) {
    if (event->button() == Qt::LeftButton) {
        qDebug() << "=====LEFT PRESS=====";
        emit mouseEventToClientThreadSignal(Mouse::LEFT_PRESS);
    } else if (event->button() == Qt::RightButton) {
        qDebug() << "=====RIGHT PRESS=====";
        emit mouseEventToClientThreadSignal(Mouse::RIGHT_PRESS);
    }
    event->accept();
}

void MainWindow::mouseReleaseEvent(QMouseEvent* event) {
    qDebug() << "=====RELEASE=====";
    emit mouseEventToClientThreadSignal(Mouse::RELEASE);
    event->accept();
}

```



```

void MainWindow::mouseMoveEvent(QMouseEvent* event) {
    qDebug() << "=====MOVE=====";
    emit mouseEventToClientThreadSignal(Mouse::MOVE);
    event->accept();
}

void MainWindow::mouseDoubleClickEvent(QMouseEvent* event) {
    qDebug() << "=====DOUBLECLICK=====";
    emit mouseEventToClientThreadSignal(Mouse::DOUBLECLICK);
    event->accept();
}

```

// validator.h:

```

#ifndef VALIDATOR_H
#define VALIDATOR_H

#include <QString>
#include <QRegularExpression>

class Validator
{
public:
    bool is_valid_IP(QString);    // проверка валидности IP
};

#endif // VALIDATOR_H

```

//validator.cpp:

```

#include "validator.h"
bool Validator::is_valid_IP(QString ip)
{
    static QRegularExpression example_of_ip("[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}");
    if(ip.isEmpty() || !example_of_ip.match(ip).hasMatch())
        return false;
    return true;
}

```

**ПРИЛОЖЕНИЕ Е**  
**(обязательное)**  
**Ведомость документов**

Обозначение					Наименование					Дополнительные сведения				
					Текстовые документы									
ГУИР КП 1-40 02 01 111 ПЗ					Пояснительная записка					38 с.				
ГУИР КП 1-40 02 01 111 КП					Код программы					24 с.				
					Графические документы									
ГУИР.1-40 02 01 111 С1					Серверная часть. Схема структурная					Формат А4				
ГУИР.1-40 02 01 111 С2					Клиентская часть. Схема структурная					Формат А4				
ГУИР.1-40 02 01 111 ПД1					Схема функции getvBasicInfoFromClient()					Формат А3				
ГУИР.1-40 02 01 111 ПД2					Схема функции gen_mouse_event()					Формат А3				