Содержимое файла *createchoise.h*:

```
#ifndef CREATECHOICE_H
#define CREATECHOICE_H

#include <QDialog>

namespace Ui
{
    class CreateChoice;
}

class CreateChoice : public QDialog
{
    Q_OBJECT

public:
    explicit CreateChoice(QWidget* parent = nullptr);
    ~CreateChoice();
    bool get_file(); // метод, сообщающий о создании файла
    bool get_dir();  // директории
    bool get_link(); // символьной ссылки

private slots:
    void on_btnFile_clicked();

    void on_btnDir_clicked();

    void on_btnLink_clicked();

    void on_btnCancel_clicked();

private:
    Ui::CreateChoice* ui;
    QString fileName = ""; // переменная для хранения имени файла
    QString dirName = "";  // директории
    QString linkName = ""; // символьной ссылки
    bool file = 0;         // флаг выбора файла
    bool dir = 0;          // директории
    bool link = 0;         // символьной ссылки
};
#endif // CREATECHOICE_H
```

Содержимое файла *createchoise.cpp*:

```
#include "createchoice.h"
#include "ui_createchoice.h"

CreateChoice::CreateChoice(QWidget* parent) : QDialog(parent), ui(new
Ui::CreateChoice)
{
    ui->setupUi(this);        // настраивает пользовательский интерфейс для
указанного виджета
    setWindowTitle("Create"); // установка имени окна
}

CreateChoice::~CreateChoice()
{
    delete ui;
}
```

```
bool CreateChoice::get_file() // метод, сообщающий о выборе создании файла
{
    return file;
}

bool CreateChoice::get_dir() // метод, сообщающий о выборе создания
директории
{
    return dir;
}

bool CreateChoice::get_link()
{
    return link;
}

void CreateChoice::on_btnFile_clicked()
{
    file = true;
    hide();
}

void CreateChoice::on_btnDir_clicked()
{
    dir = true;
    hide();
}

void CreateChoice::on_btnLink_clicked()
{
    link = true;
    hide();
}

void CreateChoice::on_btnCancel_clicked()
{
    hide();
}
```

Содержимое файла *form.h*:

```
#ifndef FORM_H
#define FORM_H

#include "../searchResult/searchresult.h"
#include "../sysElem/syselem.h"
#include "../thToCopy/threadtocpy.h"
#include "../thToRemove/threadtoremove.h"
#include "../thToReplace/threadtoreplace.h"
#include "../thToSearch/threadtosearch.h"

#include <QDir>
#include <QFileSystemModel>
#include <QListView>
#include <QString>
#include <QThread>
#include <QWidget>

namespace Ui
{
    class Form;
```

```
}

class Form : public QWidget
{
    Q_OBJECT

    Ui::Form* ui;
    QFileSystemModel* model; //указатель для связи с моделью данных для
файловой системы
    //объекты для копирования, удаления, перемещения, поиска в отдельном
потоке
    QThread* threadCopy;
    QThread* threadRemove;
    QThread* threadReplace;
    QThread* threadSearch;

    QListView* view;
    SearchResult* window;
    // объекты потоков копирования, удаления, перемещения, поиска
    ThreadToCopy* thCopy;
    ThreadToRemove* thRemove;
    ThreadToReplace* thReplace;
    ThreadToSearch* thSearch;

    File f; //объект класса File для выполнения операций с текстовыми файлами
    Dir d;  //объект класса Dir для выполнения операций с директориями

    SysElem* file = &f;
    SysElem* dir = &d;

    QString filePath = "";

    QString dirPath = "";

    QString searchName = "";

private slots:
    void on_lvLeft_clicked(const QModelIndex& index);

    void on_lvLeft_doubleClicked(const QModelIndex& index);

    void on_btnCreate_clicked();

    void on_btnRemove_clicked();

    void on_btnCopy_clicked();

    void on_btnReplace_clicked();

    void on_btnRename_clicked();

    void on_lineSearch_textEdited(const QString& arg1);

    void on_btnSearch_clicked();

    void on_leftPath_textEdited(const QString& arg1);

    void remove_is_not_performed();

    void copy_is_not_performed();

    void replace_is_not_performed();
```

```
    void ready_to_remove();

    void ready_to_copy();

    void ready_to_replace();

    void ready_to_search(QFileInfoList list);

public:
    explicit Form(QWidget* parent = nullptr);
    ~Form();

    //методы-обертки для слотов
    void btn_create();
    void btn_remove();
    void btn_copy();
    void btn_replace();
    void btn_rename();
    void btn_search();

signals:
    void start_copy(QDir rDir, SysElem* file, SysElem* dir, QString filePath,
QString dirPath);
    void start_remove(SysElem* file, SysElem* dir, QString filePath, QString
dirPath);
    void start_replace(QDir rDir, SysElem* file, SysElem* dir, QString
filePath, QString dirPath);
    void start_search(QString, QString, QString);
};

#endif // FORM_H
```

Содержимое файла *form.cpp*:

```
#include "../form/form.h"
#include "../createChoice/createchoice.h"
#include "../form/other_functions.h"
#include "../linkedPath/linkedpath.h"
#include "../newName/newname.h"
#include "ui_form.h"

#include <unistd.h>

#include <QDesktopServices>
#include <QMessageBox>

Form::Form(QWidget* parent) : QWidget(parent), ui(new Ui::Form)
{
    ui->setupUi(this); // настраивает пользовательский интерфейс для
указанного виджета

    if (!(model = new QFileSystemModel(this))) // выделение памяти под
указатель на объект этого класса
        QMessageBox::warning(this, "Memory allocation", "Model of
QFileSystemModel was not created!");

    model->setFilter(QDir::QDir::AllEntries); //отображать некоторые элементы
файловой системы(в нашем случае все)
    model->setRootPath("/"); // определить место в системе для отслеживания
изменений(указана корневая папка)
```

```
    ui->lvLeft->setModel(model); // назначение *model объектом представления
панели Source
    ui->lvRight->setModel(model);

    connect(ui->lvRight, SIGNAL(clicked(QModelIndex)), this,
SLOT(on_lvLeft_clicked(QModelIndex)));
    connect(ui->lvRight, SIGNAL(doubleClicked(QModelIndex)), this,
SLOT(on_lvLeft_doubleClicked(QModelIndex)));

    connect(ui->rightPath, SIGNAL(textEdited(QString)), this,
SLOT(on_leftPath_textEdited(QString)));

    if (!(threadCopy = new QThread(this)))
        QMessageBox::warning(this, "Memory allocation", "Object of QThread
was not created!");

    if (!(threadRemove = new QThread(this)))
        QMessageBox::warning(this, "Memory allocation", "Object of QThread
was not created!");

    if (!(threadReplace = new QThread(this)))
        QMessageBox::warning(this, "Memory allocation", "Object of QThread
was not created!");

    if (!(threadSearch = new QThread(this)))
        QMessageBox::warning(this, "Memory allocation", "Object of QThread
was not created!");

    if (!(thCopy = new ThreadToCopy()))
        QMessageBox::warning(this, "Memory allocation", "Object of
ThreadToCopy was not created!");

    if (!(thRemove = new ThreadToRemove()))
        QMessageBox::warning(this, "Memory allocation", "Object of
ThreadToRemove was not created!");

    if (!(thReplace = new ThreadToReplace()))
        QMessageBox::warning(this, "Memory allocation", "Object of
ThreadToReplace was not created!");

    if (!(thSearch = new ThreadToSearch()))
        QMessageBox::warning(this, "Memory allocation", "Object of
ThreadToSearch was not created!");

    connect_threads(this, threadCopy, threadRemove, threadReplace,
threadSearch, thCopy, thRemove, thReplace, thSearch);

    if (!(window = new SearchResult(this)))
        QMessageBox::warning(this, "Memory allocation", "Object of
SearchResult was not created!");

    // устанока информации для подсказок
    ui->btnCreate->setToolTip("Create");
    ui->btnRemove->setToolTip("Remove");
    ui->btnCopy->setToolTip("Copy");
    ui->btnReplace->setToolTip("Replace");
    ui->btnRename->setToolTip("Rename");

    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); //получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
```

```
    lDir.cd(lDir.homePath());
    ui->lvLeft->setRootIndex(model->index(lDir.homePath()));
    rDir.cd(rDir.homePath());
    ui->lvRight->setRootIndex(model->index(rDir.homePath()));

    ui->leftPath->setText(lDir.homePath());  // отображение нового пути
    ui->rightPath->setText(rDir.homePath()); // отображение нового пути

    on_lvLeft_clicked(model->index(lDir.homePath()));
    view = ui->lvLeft;
}

Form::~Form()
{
    delete ui;
    delete model;
    emit threadCopy->quit();
    threadCopy->wait();
    delete threadCopy;
    emit threadRemove->quit();
    threadRemove->wait();
    delete threadRemove;
    emit threadReplace->quit();
    threadReplace->wait();
    delete threadReplace;
    emit threadSearch->quit();
    threadSearch->wait();
    delete threadSearch;
    delete thCopy;
    delete thRemove;
    delete thReplace;
    delete thSearch;
    delete window;
}

void Form::btn_create()
{
    on_btnCreate_clicked();
}

void Form::btn_remove()
{
    on_btnRemove_clicked();
}

void Form::btn_copy()
{
    on_btnCopy_clicked();
}

void Form::btn_replace()
{
    on_btnReplace_clicked();
}

void Form::btn_rename()
{
    on_btnRename_clicked();
}

void Form::btn_search()
```

```
{
    on_btnSearch_clicked();
}

void Form::on_lvLeft_clicked(const QModelIndex& index)
{
    view = (QListView*)sender();
    QFileInfo info = model->fileInfo(index); // получение пути элемента,
который соответствует этому индексу
    if (info.isSymLink() || info.isFile()) // выбранный объект - файл
    {
        if (!dirPath.isEmpty()) // если до этого была выбрана директория
            dirPath.clear();
        init_infoBar(info, ui->lblSize, ui->lblType, ui->lblDate);
        filePath = model->filePath(index); // установка пути выбранного файла
    }
    else if (!info.fileName().compare(".") || !info.fileName().compare(".."))
    {
        filePath.clear();
        dirPath.clear();
        ui->lblSize->clear();
        ui->lblType->clear();
        ui->lblDate->clear();
    }
    else if (info.isDir()) // если выбранный объект - директория
    {
        QDir dir = QDir(model->filePath(index));
        if (!filePath.isEmpty()) // если до этого был выбран файл
            filePath.clear();    // очистка пути файл
        QString count = "";
        count = count.append(QString::number(dir.count() - 2)).append("
item");
        if (dir.count() > 3)
            count = count.append("s");
        ui->lblSize->setText(count);
        ui->lblType->setText("Type: directory");                    //
отображение типа объекта
        QString format = "dddd, d MMMM yy hh:mm:ss";                // формат
вывода даты последнего изменения
        ui->lblDate->setText(info.lastModified().toString(format)); // вывод
даты последнего изменения
        dirPath = info.absoluteFilePath(); // переменная для хранения пути
выбранной директории
    }
}

void Form::on_lvLeft_doubleClicked(const QModelIndex& index)
{
    view = (QListView*)sender(); // Возвращает указатель на объект,
отправивший сигнал, если он вызван в слоте, активированном сигналом.
    QLineEdit* var;
    if (view == ui->lvLeft)
        var = ui->leftPath;
    if (view == ui->lvRight)
        var = ui->rightPath;
    QFileInfo info = model->fileInfo(index); // получение информации
элемента, который соответствует этому индексу
    if (info.isFile()) // если выбранный элемент - файл
    {
        if (!dirPath.isEmpty())                                     // если до
этого выбрана директория
```

```
            dirPath.clear();                                        // очистка
пути директории
        QDesktopServices::openUrl(QUrl::fromUserInput(filePath)); //
открывает файл
        filePath.clear();                                       // очистка
пути файла
    }
    else if (info.isDir()) // если выбранный элемент - директория
    {
        if (!filePath.isEmpty()) // если до этого был выбран файл
            filePath.clear();    // очистка пути файла
        if (!info.fileName().compare("."))
        {
            view->setRootIndex(model->index("")); // переход в корневую папre
(показать корневую папку)
            info = model->fileInfo(model->index(""));
        }
        else if (!info.fileName().compare(".."))
        {
            QDir qDir = info.dir(); // получение объекта класса QDir
            qDir.cd(".."); // dir.cdUp();  навигация. в данном случае переход
в родительскую папку
            view->setRootIndex(model->index(qDir.absolutePath())); //
получение индекса по пути
        }
        else
            view->setRootIndex(index);         // элемент с этим индексом
становится корневым
        var->setText(info.absoluteFilePath()); // отображение нового пути
        dirPath.clear();                       //очистка пути директории
    }
}

void Form::on_btnCreate_clicked()
{
    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); //получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
    QDir qDir;
    if (view == ui->lvLeft)
        qDir = lDir;
    if (view == ui->lvRight)
        qDir = rDir;
    if (qDir.absolutePath().contains(qDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || !qDir.absolutePath().contains(qDir.homePath())) // если это
корневая директория
    {
        QMessageBox::warning(this, "Create", "There is no access to perform
any operation in this directory!");
        return;
    }
    CreateChoice window;
    window.exec(); // метод выполняет появление окна для выбора типа
создаваемого объекта
    NewName name;
    name.exec(); // выполняет появление окна для создания имени
    foreach (QFileInfo info, qDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name))
        if (info.fileName() == name.get_name()) // если файл с таким именем
найден
        {
```

```
            QMessageBox::warning(this, "Create", "A file or a directory with
this name exists!");
            return;
        }
    QString createPath =
qDir.absolutePath().append("/").append(name.get_name()); // получение пути
сооздданного файла
    if (window.get_file() || window.get_link())
// если был выбран файл
    {
        if (window.get_link())
        {
            LinkedPath window;
            window.exec();                          // выполняет появление
окна для создания имени
            QString linkedPath = window.get_path(); // получение пути
сооздданной ссылкb

           if (symlink(linkedPath.toLocal8Bit().constData(),
createPath.toLocal8Bit().constData()) != 0)
               QMessageBox::warning(this, "Create", "The operation was not
perfomed!");
        }
        else if (window.get_file())
        {
            if (!file->create(createPath)) // если файл не создан
                QMessageBox::warning(this, "", "The operation was not
perfomed!");
        }
    }
    else if (window.get_dir()) // если выбранный объект - директория
    {
        if (!dir->create(createPath)) // если директория не создана
            QMessageBox::warning(this, "", "The operation was not
perfomed!");
    }
}

void Form::on_btnRemove_clicked()
{
    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); // получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
    QDir qDir;
    if (view == ui->lvLeft)
        qDir = lDir;
    if (view == ui->lvRight)
        qDir = rDir;
    if (qDir.absolutePath().contains(qDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || !qDir.absolutePath().contains(qDir.homePath())) // если это
корневая директория
    {
        QMessageBox::warning(this, "Remove", "There is no access to perform
any operation in this directory!");
        return;
    }
    if (filePath.isEmpty() && dirPath.isEmpty()) // если не выбран ни один
объект
    {
        QMessageBox::warning(this, "Remove", "You was not choose a file or a
directory! Please try again");
```

```
        return;
    }
    QMessageBox::StandardButton btn
        = QMessageBox::question(this, "Remove", "Do you want to perform
operation?", QMessageBox::Cancel | QMessageBox::Ok);
    if (btn == QMessageBox::Cancel)
    {
        filePath.clear(); // очистка пути файла
        dirPath.clear();  // очистка пути директории
        return;
    }
    ui->btnRemove->setEnabled(false);
    emit start_remove(file, dir, filePath, dirPath);
    filePath.clear(); // очистка пути файла
    dirPath.clear();  // очистка пути директории
}

void Form::on_btnCopy_clicked()
{
    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); // получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
    if (lDir.absolutePath().contains(lDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || rDir.absolutePath().contains(rDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || !lDir.absolutePath().contains(lDir.homePath()) ||
!rDir.absolutePath().contains(rDir.homePath()))
    {
        QMessageBox::warning(this, "Copy", "There is no access to perform any
operation in this directory!");
        filePath.clear(); // очистка пути файла
        dirPath.clear();
        return;
    }
    if (filePath.isEmpty() && dirPath.isEmpty()) // если не выбран ни один
объект
    {
        QMessageBox::warning(this, "Copy", "You was not choose a file or a
directory! Please try again");
        return;
    }
    QFileInfo data;
    if (!filePath.isEmpty()) // если выбран файл
    {
        data = QFileInfo(filePath);
    }
    else if (!dirPath.isEmpty()) // если выбрана директория
    {
        data = QFileInfo(dirPath);
    }
    foreach (QFileInfo info, rDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name))
    {
        if (info.fileName() == data.fileName()) // если файл с таким именем
есть
        {
            QMessageBox::warning(this, "Copy", "A file or a directory with
this name exists!");
            filePath.clear(); // очистка пути файла
            return;
        }
```

```
    }
    ui->btnCopy->setEnabled(false);
    emit start_copy(rDir, file, dir, filePath, dirPath);
    filePath.clear(); // очистка пути файла
    dirPath.clear();
}

void Form::on_btnReplace_clicked()
{
    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); // получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
    if (lDir.absolutePath().contains(lDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || rDir.absolutePath().contains(rDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || !lDir.absolutePath().contains(lDir.homePath()) ||
!rDir.absolutePath().contains(rDir.homePath()))
    {
        QMessageBox::warning(this, "Replace", "There is no access to perform
any operation in this directory!");
        filePath.clear(); // очистка пути файла
        dirPath.clear();
        return;
    }
    if (filePath.isEmpty() && dirPath.isEmpty()) // если не выбран ни один
объект
    {
        QMessageBox::warning(this, "Replace", "You was not choose a file or a
directory! Please try again");
        return;
    }
    QFileInfo data;
    if (!filePath.isEmpty()) // если выбран файл
    {
        data = QFileInfo(filePath);
    }
    else if (!dirPath.isEmpty()) // если выбрана директория
    {
        data = QFileInfo(dirPath);
    }
    foreach (QFileInfo info, rDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name))
    {
        if (info.fileName() == data.fileName()) // если файл с таким именем
есть
        {
            QMessageBox::warning(this, "Replace", "A file or a directory with
this name exists!");
            filePath.clear(); // очистка пути файла
            return;
        }
    }
    ui->btnReplace->setEnabled(false);
    emit start_replace(rDir, file, dir, filePath, dirPath);
    filePath.clear(); // очистка пути файла
    dirPath.clear();
}

void Form::on_btnRename_clicked()
{
```

```
    QDir lDir = QDir(model->filePath(ui->lvLeft->rootIndex())); // получение
текущей директории
    QDir rDir = QDir(model->filePath(ui->lvRight->rootIndex()));
    QDir qDir;
    if (view == ui->lvLeft)
        qDir = lDir;
    if (view == ui->lvRight)
        qDir = rDir;
    if (qDir.absolutePath().contains(qDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        || !qDir.absolutePath().contains(qDir.homePath())) // если это
корневая директория
    {
        QMessageBox::warning(this, "Rename", "There is no access to perform
any operation in this directory!");
        return;
    }
    if (filePath.isEmpty() && dirPath.isEmpty()) // если не выбран ни один
объект
    {
        QMessageBox::warning(this, "Rename", "You was not choose a file or a
directory! Please try again");
        return;
    }
    NewName name;
    name.exec();                    // метод выполняет появление окна для
переименования файла
    if (name.get_name().isEmpty()) // если имя не введено
    {
        filePath.clear();
        dirPath.clear();
        return;
    }
    foreach (QFileInfo info, qDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name))
        if (info.fileName() == name.get_name()) // если файл с таким именем
есть
        {
            QMessageBox::warning(this, "Rename", "A file or a directory with
this name exists!");
            filePath.clear();
            dirPath.clear();
            return;
        }
    QString newPath =
qDir.absolutePath().append("/").append(name.get_name()); // создание нового
пути с учетом переименования
    if (!filePath.isEmpty())
// если выбран файл
    {
        if (!file->r_name(filePath, newPath)) // если переименование не
произошло
            QMessageBox::warning(this, "Rename", "The operation was not
perfomed!");
        filePath.clear(); // очистка пути файла
    }
    else if (!dirPath.isEmpty()) // если выбрана директория
    {
        if (!dir->r_name(dirPath, newPath)) // если переименование не
произошло
            QMessageBox::warning(this, "Rename", "The operation was not
perfomed!");
```

```
        dirPath.clear(); // очистка пути директории
    }
}

void Form::on_lineSearch_textEdited(const QString& arg1)
{
    searchName = arg1;
}

void Form::on_btnSearch_clicked()
{
    QString lDirPath = model->filePath(ui->lvLeft->rootIndex());
    QString rDirPath = model->filePath(ui->lvRight->rootIndex());
    QDir lDir = QDir(lDirPath); // получение текущей директории
    QDir rDir = QDir(rDirPath);
    if (lDirPath.contains(lDir.homePath()) &&
rDirPath.contains(rDir.homePath())
        && !lDirPath.contains(lDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug"))
        && !rDirPath.contains(rDir.homePath().append("/kypck/build-
FileManager-Desktop_Qt_6_5_0_GCC_64bit-Debug")))
    {
        ui->btnSearch->setEnabled(false);
        emit start_search(lDirPath, rDirPath, searchName);
    }
    else
        QMessageBox::warning(this, "", "There is no access to perform a
search in this directory!");
}

void Form::on_leftPath_textEdited(const QString& arg1)
{
    QLineEdit* line = (QLineEdit*)sender();
    if (line == ui->leftPath)
        view = ui->lvLeft;
    else if (line == ui->rightPath)
        view = ui->lvRight;
    QFileInfo fileInfo = model->fileInfo(model->index(arg1)); // получение
информации элемента, который соответствует этому индексу
    if (fileInfo.isFile() || fileInfo.isSymLink()) // если выбранный элемент
- файл или ссылка
        view->setRootIndex(model->index(fileInfo.absolutePath()));
    else if (fileInfo.isDir())
        view->setRootIndex(model->index(arg1)); // элемент с этим индексом
становится корневым
}

void Form::remove_is_not_performed()
{
    ui->btnRemove->setEnabled(true);
    QMessageBox::warning(this, "Remove", "The operation is not performed!");
}

void Form::copy_is_not_performed()
{
    ui->btnCopy->setEnabled(true);
    QMessageBox::warning(this, "Copy", "The operation is not performed!");
}

void Form::replace_is_not_performed()
{
    ui->btnReplace->setEnabled(true);
```

```
    QMessageBox::warning(this, "Replace", "The operation is not performed!");
}

void Form::ready_to_remove()
{
    ui->btnRemove->setEnabled(true);
}

void Form::ready_to_copy()
{
    ui->btnCopy->setEnabled(true);
}

void Form::ready_to_replace()
{
    ui->btnReplace->setEnabled(true);
}

void Form::ready_to_search(QFileInfoList list)
{
    ui->btnSearch->setEnabled(true);
    window->set_ui(list);
    window->exec();
    window->reset_ui();
}
```

Содержимое файла *linkedpath.h*:

```
#ifndef LINKEDPATH_H
#define LINKEDPATH_H

#include <QDialog>

namespace Ui
{
    class LinkedPath;
}

class LinkedPath : public QDialog
{
    Q_OBJECT;

public:
    explicit LinkedPath(QWidget* parent = nullptr);
    ~LinkedPath();
    QString get_path(); // метод передачи имени

private slots:
    void on_btnCancel_clicked();

    void on_btnOK_clicked();

    void on_path_textEdited(const QString& arg1);

private:
    Ui::LinkedPath* ui;
    QString path = ""; // переменная для хранения нового имени выбранного
файла
};

#endif // LINKEDPATH_H
```

Содержимое файла *linkedpath.cpp*:

```
#include "linkedpath.h"
#include "ui_linkedpath.h"

LinkedPath::LinkedPath(QWidget* parent) : QDialog(parent), ui(new
Ui::LinkedPath)
{
    ui->setupUi(this);
    setWindowTitle("Link"); // установка имени окна
}

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

QString LinkedPath::get_path() // метод передачи имени
{
    return path;
}

void LinkedPath::on_btnCancel_clicked()
{
    path.clear();
    hide(); // закрытие текущего окна
}

void LinkedPath::on_btnOK_clicked()
{
    hide(); // закрытие текущего окна
}

void LinkedPath::on_path_textEdited(const QString& arg1)
{
    path = arg1;
}
```

Содержимое файла *mainwindow.h*:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "../form/form.h"
#include "../searchResult/searchresult.h"

#include <QMainWindow> //предоставляет главное окно приложения

QT_BEGIN_NAMESPACE
namespace Ui
{
    class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow //класс главного окна приложения
{
    Q_OBJECT
    //теперь можем использовать сигналы-слоты в классе
```

```
    Ui::MainWindow* ui; //указатель на объект соотв типа в классе основной
формы
    Form* form;         //указатель для связи с моделью данных для файловой
системы

    ThreadToSearch* thSearch;

    SearchResult* window;

private slots:

    void on_tabWidget_tabCloseRequested(int index);

    void on_CtrlX_triggered();

    void on_CtrlC_triggered();

    void on_CtrlEsc_triggered();

    void on_CtrlN_triggered();

    void on_CtrlLeft_triggered();

    void on_CtrlRight_triggered();

    void on_CtrlDel_triggered();

    void on_CtrlT_triggered();

    void on_CtrlR_triggered();

    void on_CtrlD_triggered();

    void on_CtrlF_triggered();

public slots:
    void add_tab();

public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();
};
#endif // MAINWINDOW_H
```

Содержимое файла *mainwindow.cpp*:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QLabel>
#include <QMessageBox>
#include <QTableWidget>
#include <QToolButton>

MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent), ui(new
Ui::MainWindow)
{
    ui->setupUi(this); // настраивает пользовательский интерфейс для
указанного виджета
    setWindowTitle("FileManager"); // установка имени главного окна

    ui->tabWidget->clear();
```

```
    QToolButton* tb = new QToolButton();
    tb->setText("+");
    tb->setAutoRaise(true);
    connect(tb, SIGNAL(clicked()), this, SLOT(add_tab()));

    QLabel* lbl = nullptr;
    if (!(lbl = new QLabel("You can add tabs by pressing <b>\"+\"</b>")))
        QMessageBox::warning(this, "Memory allocation", "Object of QLabel was
not created!");
    lbl->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);

    ui->tabWidget->addTab(lbl, QString());
    ui->tabWidget->setTabEnabled(0, false);
    ui->tabWidget->tabBar()->setTabButton(0, QTabBar::RightSide, tb);

    add_tab();
}

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

void MainWindow::add_tab()
{
    form = new Form(this);
    QString tabName = "Tab " + QString::number(ui->tabWidget->count());
    ui->tabWidget->insertTab(ui->tabWidget->count() - 1, form, tabName);
    ui->tabWidget->setCurrentIndex(ui->tabWidget->count() - 2);
}

void MainWindow::on_tabWidget_tabCloseRequested(int index)
{
    if (ui->tabWidget->count() > 2 && ui->tabWidget->currentIndex() != ui-
>tabWidget->count() - 1)
        ui->tabWidget->removeTab(index);
}

void MainWindow::on_CtrlC_triggered()
{
    form->btn_copy();
}

void MainWindow::on_CtrlX_triggered()
{
    form->btn_replace();
}

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

void MainWindow::on_CtrlN_triggered()
{
    form->btn_create();
}

void MainWindow::on_CtrlLeft_triggered()
{
    ui->tabWidget->setCurrentIndex(ui->tabWidget->currentIndex() - 1);
```

```
}

void MainWindow::on_CtrlRight_triggered()
{
    if (ui->tabWidget->currentIndex() < ui->tabWidget->count() - 2)
        ui->tabWidget->setCurrentIndex(ui->tabWidget->currentIndex() + 1);
}

void MainWindow::on_CtrlDel_triggered()
{
    on_tabWidget_tabCloseRequested(ui->tabWidget->currentIndex());
}

void MainWindow::on_CtrlT_triggered()
{
    add_tab();
}

void MainWindow::on_CtrlR_triggered()
{
    form->btn_rename();
}

void MainWindow::on_CtrlD_triggered()
{
    form->btn_remove();
}

void MainWindow::on_CtrlF_triggered()
{
    form->btn_search();
}
```

Содержимое файла *newname.h*:

```
#ifndef NEWNAME_H
#define NEWNAME_H

#include <QDialog>

namespace Ui
{
    class NewName;
}

class NewName : public QDialog
{
    Q_OBJECT

    Ui::NewName* ui;
    QString name = ""; // переменная для хранения нового имени выбранного
файла

private slots:
    void on_name_textEdited(const QString& arg1);

    void on_btnCancel_clicked();

    void on_btnOK_clicked();

public:
```

```
    explicit NewName(QWidget* parent = nullptr);
    ~NewName();
    QString get_name(); // метод передачи имени
};

#endif // NEWNAME_H
```

Содержимое файла *newname.cpp*:

```
#include "newname.h"
#include "ui_newname.h"

NewName::NewName(QWidget* parent) : QDialog(parent), ui(new Ui::NewName)
{
    ui->setupUi(this);
    setWindowTitle("Rename"); // установка имени окна
}

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

QString NewName::get_name() // метод передачи имени
{
    return name;
}

void NewName::on_name_textEdited(const QString& arg1)
{
    name = arg1;
}

void NewName::on_btnCancel_clicked()
{
    name.clear();
    hide(); // закрытие текущего окна
}

void NewName::on_btnOK_clicked()
{
    hide(); // закрытие текущего окна
}
```

Содержимое файла *searchresult.h*:

```
#ifndef SEARCHRESULT_H
#define SEARCHRESULT_H

#include <QClipboard>
#include <QDialog>
#include <QDir>
#include <QListWidget> //предоставляет виджет списка на основе элементов

namespace Ui
{
    class SearchResult;
}

class SearchResult : public QDialog
{
```

```
    Q_OBJECT

    Ui::SearchResult* ui; // указатель на объект соотв типа в классе основной
формы

public:
    explicit SearchResult(QWidget* parent = nullptr);
    ~SearchResult();
    void set_ui(QFileInfoList); // метод передачи результатов поиска для
отображения
    void reset_ui();            // метод очистки окна отображения результатов
поиска

private slots:
    void on_btnOK_clicked();

    void on_leftList_itemClicked(QListWidgetItem* item);
};

#endif // SEARCHRESULT_H
```

Содержимое файла *searchresult.cpp*:

```
#include "searchresult.h"
#include "ui_searchresult.h"

#include <QClipboard>
#include <QFileSystemModel>
#include <QLabel>
#include <QMessageBox>
#include <QStandardItemModel>
#include <QStringListModel>

SearchResult::SearchResult(QWidget* parent) : QDialog(parent), ui(new
Ui::SearchResult)
{
    ui->setupUi(this);
    setWindowTitle("Search"); // установка имени окна
}

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

void SearchResult::set_ui(QFileInfoList list) // метод передачи результатов
поиска для отображения
{
    //    QStringListModel* model;
    //    if (!(model = new QStringListModel(this))) // выделение памяти под
указатель на объект этого класса
    //        QMessageBox::warning(this, "Memory allocation", "Model of
QFileSystemModel was not created!");

    //    model->setStringList(list);
    //    ui->leftList->setModel(model);

    if (list.isEmpty()) // если контейнер пуст
    {
        ui->leftList->addItem("No elements");
        return;
    }
```

```
    foreach (QFileInfo info, list)
    {
        ui->leftList->addItem(info.absoluteFilePath());
    }
}

void SearchResult::reset_ui() // метод очистки окна отображения результатов
поиска
{
    ui->leftList->clear(); // очистка окна вывода результатов поиска
}

void SearchResult::on_btnOK_clicked()
{
    hide(); // закрытие текущего окна
}

void SearchResult::on_leftList_itemClicked(QListWidgetItem* item)
{
    QClipboard* pcb = QApplication::clipboard(); //создание объекта для
взаимодействия с буфером обмена
    pcb->setText(item->text(), QClipboard::Clipboard); //копирование
выбранного текста в буфер обмена
}
```

Содержимое файла *syselem.h*:

```
#ifndef SYSELEM_H
#define SYSELEM_H

#include <cstdio>
#include <fstream>
#include <iostream>
#include <sys/stat.h>

#include <QDir>
#include <QFileSystemModel>
#include <QString>

using namespace std;

class SysElem
{
public:
    SysElem() = default;
    virtual ~SysElem() = default;
    // ниже представлены чисто виртуальные методы для работы с системными
    // объектами описаны они будут в производных классах
    virtual bool create(QString) = 0;
    virtual bool r_move(QString) = 0;
    virtual bool r_name(QString, QString) = 0;
    virtual bool c_py(QString, QString) = 0;
};

class File : public SysElem // производный класс File для работы с текстовыми
файлами
{
public:
    File() = default;
    ~File() = default;
    bool create(QString);          // метод создания текстового файла
    bool r_move(QString);          // метод удаления текстового файла
```

```
    bool r_name(QString, QString); // метод переименования текстового файла
    bool c_py(QString, QString);   // метод копирования текстового файла
};

class Dir : public SysElem // производный класс Dir для работы с директориями
{
public:
    Dir() = default;
    ~Dir() = default;
    bool create(QString);          // метод создания директории
    bool r_move(QString);          // метод удаления директории
    bool r_name(QString, QString); // метод переименования директории
    bool c_py(QString, QString);   // метод копирования директории
};

#endif // SYSELEM_H
```

Содержимое файла *syselem.cpp*:

```
#include "syselem.h"
#include <cstdio>
#include <fstream>
#include <iostream>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>

using namespace std;

bool File::create(QString filePath) // метод создания файла
{
    ofstream file;                                  //создание объекта класса
ofstream
    file.open(filePath.toLocal8Bit().constData()); //открытие файла с
указанным именем
    return file.is_open();
}

bool File::r_move(QString filePath) // метод удаления текстового файла
{
    return !remove(filePath.toLocal8Bit().constData()); // если выполнено
}

bool File::c_py(QString filePath, QString newPath) // метод копирования
текстового файла
{
    QString cmd = "cp ";
    cmd = cmd.append(filePath).append(" ").append(newPath);
    return !system(cmd.toLocal8Bit().constData());
}

bool File::r_name(QString filePath, QString newPath) // метод переименования
текстового файла
{
    return !rename(filePath.toLocal8Bit().constData(),
newPath.toLocal8Bit().constData()); // если выполнено
}

bool Dir::create(QString dirPath) // метод создания директории
{
```

```
    return !mkdir(dirPath.toLocal8Bit().constData(), S_IRWXU | S_IRWXG |
S_IROTH | S_IXOTH);
}

bool Dir::r_move(QString dirPath) // метод удаления директории
{
    return !remove(dirPath.toLocal8Bit().constData());
}

bool Dir::r_name(QString dirPath, QString newPath) // метод переименования
директории
{
    return !rename(dirPath.toLocal8Bit().constData(),
newPath.toLocal8Bit().constData());
}

bool Dir::c_py(QString dirName, QString newPath) // метод копирования
директории
{
    newPath = newPath.append("/").append(dirName);
    return !mkdir(newPath.toLocal8Bit().constData(), S_IRWXU | S_IRWXG |
S_IROTH | S_IXOTH); // если выполнено
}
```

Содержимое файла *threadtocpy.h*:

```
#ifndef THREADTOCPY_H
#define THREADTOCPY_H

#include "../sysElem/syselem.h"

#include <QDir>
#include <QFileInfo>
#include <QObject>

class ThreadToCopy : public QObject
{
    Q_OBJECT

public slots:
    void run_copy(QDir rDir, SysElem* file, SysElem* dir, QString filePath,
QString dirPath);

public:
    explicit ThreadToCopy(QObject* parent = nullptr);
    void c_py(QDir, SysElem*, SysElem*, QString);

signals:
    void not_performed();
    void copy_finished();
};

#endif // THREADTOCPY_H
```

Содержимое файла *threadtocpy.cpp*:

```
#include "threadtocpy.h"

ThreadToCopy::ThreadToCopy(QObject* parent) : QObject{parent} {}
```

```
void ThreadToCopy::run_copy(QDir rDir, SysElem* file, SysElem* dir, QString
filePath, QString dirPath)
{
    if (!filePath.isEmpty())
    {
        if (!file->c_py(filePath, rDir.absolutePath())) // если копирование
не произошло
        {
            emit not_performed();
            return;
        }
    }
    else
        c_py(rDir, file, dir, dirPath);
    emit copy_finished();
}

void ThreadToCopy::c_py(QDir rDir, SysElem* file, SysElem* dir, QString
dirPath)
{
    QFileInfoList copyList = QFileInfoList(); // создание контейнера для
хранения внутренних файлов выбранной директории
    QDir lDir = QDir(dirPath);
    foreach (QFileInfo info, lDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name | QDir::DirsFirst))
        copyList.append(info);                          // добавление
элемента в контейнер
    if (!dir->c_py(lDir.dirName(), rDir.absolutePath())) // если копирование
не выполнено
    {
        emit not_performed();
        return;
    }
    rDir.cd(lDir.dirName());
    QString copyPath = rDir.absolutePath();
    //цикл копирования элементов контейнера в созданную директорию
    foreach (QFileInfo info, copyList)
    {
        if (info.isFile()) // текущий элемент контейнера - файл
        {
            if (!file->c_py(info.absoluteFilePath(), copyPath)) // если
копирование не выполнено
                emit not_performed();
        }
        else if (info.isDir()) // если текущий элемент - директория
            c_py(rDir, file, dir, info.absoluteFilePath());
    }
    rDir.cdUp();
}
```

Содержимое файла *threadtoremove.h*:

```
#ifndef THREADTOREMOVE_H
#define THREADTOREMOVE_H

#include "../sysElem/syselem.h"

#include <QDir>
#include <QObject>

class ThreadToRemove : public QObject
{
```

```
    Q_OBJECT

public slots:
    void run_remove(SysElem* file, SysElem* dir, QString filePath, QString
dirPath);

public:
    explicit ThreadToRemove(QObject* parent = nullptr);
    bool rec_remove(QDir&, SysElem*, SysElem*); //функция рекурсивного
удаления содержимого выбранной папки
    void r_move(SysElem*, SysElem*, QString);

signals:
    void not_performed();
    void remove_finished();
};
#endif // THREADTOREMOVE_H
```

Содержимое файла *threadtoremove.cpp*:

```
#include "threadtoremove.h"

ThreadToRemove::ThreadToRemove(QObject* parent) : QObject{parent} {}

void ThreadToRemove::run_remove(SysElem* file, SysElem* dir, QString
filePath, QString dirPath)
{
    if (!filePath.isEmpty()) // если выбран файл
    {
        if (!file->r_move(filePath)) // удаление файла
        {
            emit not_performed();
            return;
        }
    }
    else
        r_move(file, dir, dirPath);
    emit remove_finished();
}

bool ThreadToRemove::rec_remove(QDir& qDir, SysElem* file, SysElem* dir) //
функция рекурсивного удаления содержимого выбранной папки
{
    // цикл прохода по текущей директории для удаления файлов и директорий
внутри
    foreach (QFileInfo info, qDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name | QDir::DirsFirst))
    {
        if (info.isDir()) // если директория
        {
            qDir.cd(info.fileName());                   // заходим в нее
            rec_remove(qDir, file, dir);                // рекурсивно удаляем
внутренности
            qDir.cdUp();                                // возврат
            if (!dir->r_move(info.absoluteFilePath())) // теперь папка пуста
и мы можем ее удалить
                return false;
        }
        else if (info.isFile()) // если текущий объект - файл
            if (!file->r_move(info.absoluteFilePath()))
                return false;
```

```
    }
    return true;
}

void ThreadToRemove::r_move(SysElem* file, SysElem* dir, QString dirPath)
{
    QDir qDir = QDir(dirPath); // получение выбранной директории
    if (!qDir.isEmpty())       // если директория не пуста
    {
        if (!rec_remove(qDir, file, dir)) // если внутренние файлы не удалены
        {
            emit not_performed();
            return;
        }
    }
    if (qDir.isEmpty()) // если директория пуста
    {
        if (!dir->r_move(dirPath)) // если удаление не выпонено
            emit not_performed();
    }
}
```

Содержимое файла *threadtoreplace.h*:

```
#ifndef THREADTOREPLACE_H
#define THREADTOREPLACE_H

#include "../sysElem/syselem.h"

#include <QDir>
#include <QFileInfo>
#include <QObject>

class ThreadToReplace : public QObject
{
    Q_OBJECT
public slots:
    void run_replace(QDir rDir, SysElem* file, SysElem* dir, QString
filePath, QString dirPath);

public:
    explicit ThreadToReplace(QObject* parent = nullptr);
    void c_py(QDir, SysElem*, SysElem*, QString);
    bool rec_remove(QDir&, SysElem*, SysElem*);
    void r_move(SysElem*, SysElem*, QString);

signals:
    void not_performed();
    void replace_finished();
};

#endif // THREADTOREPLACE_H
```

Содержимое файла *threadtoreplace.cpp*:

```
#include "threadtoreplace.h"

ThreadToReplace::ThreadToReplace(QObject* parent) : QObject{parent} {}

void ThreadToReplace::run_replace(QDir rDir, SysElem* file, SysElem* dir,
QString filePath, QString dirPath)
```

```
{
    if (!filePath.isEmpty())
    {
        if (!file->c_py(filePath, rDir.absolutePath())) // если копирование
не произошло
        {
            emit not_performed();
            return;
        }
        if (!file->r_move(filePath))
        {
            emit not_performed();
            return;
        }
    }
    else
    {
        c_py(rDir, file, dir, dirPath);
        r_move(file, dir, dirPath);
    }
    emit replace_finished();
}

void ThreadToReplace::c_py(QDir rDir, SysElem* file, SysElem* dir, QString
dirPath)
{
    QFileInfoList copyList = QFileInfoList(); // создание контейнера для
хранения внутренних файлов выбранной директории
    QDir lDir = QDir(dirPath);
    foreach (QFileInfo info, lDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name | QDir::DirsFirst))
        copyList.append(info);                              // добавление
элемента в контейнер
    if (!dir->c_py(lDir.dirName(), rDir.absolutePath())) // если копирование
не выполнено
    {
        emit not_performed();
        return;
    }
    rDir.cd(lDir.dirName());
    QString copyPath = rDir.absolutePath();
    //цикл копирования элементов контейнера в созданную директорию
    foreach (QFileInfo info, copyList)
    {
        if (info.isFile()) // текущий элемент контейнера - файл
        {
            if (!file->c_py(info.absoluteFilePath(), copyPath)) // если
копирование не выполнено
                emit not_performed();
        }
        else if (info.isDir()) // если текущий элемент - директория
            c_py(rDir, file, dir, info.absoluteFilePath());
    }
    rDir.cdUp();
}

bool ThreadToReplace::rec_remove(QDir& qDir, SysElem* file, SysElem* dir) //
функция рекурсивного удаления содержимого выбранной папки
{
    // цикл прохода по текущей директории для удаления файлов и директорий
внутри
```

```
    foreach (QFileInfo info, qDir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name | QDir::DirsFirst))
    {
        if (info.isDir()) // если директория
        {
            qDir.cd(info.fileName());                   // заходим в нее
            rec_remove(qDir, file, dir);                // рекурсивно удаляем
внутренности
            qDir.cdUp();                                // возврат
            if (!dir->r_move(info.absoluteFilePath())) // теперь папка пуста
и мы можем ее удалить
                return false;
        }
        else if (info.isFile()) // если текущий объект - файл
            if (!file->r_move(info.absoluteFilePath()))
                return false;
    }
    return true;
}

void ThreadToReplace::r_move(SysElem* file, SysElem* dir, QString dirPath)
{
    QDir qDir = QDir(dirPath); // получение выбранной директории
    if (!qDir.isEmpty())       // если директория не пуста
    {
        if (!rec_remove(qDir, file, dir)) // если внутренние файлы не удалены
        {
            emit not_performed();
            return;
        }
    }
    if (qDir.isEmpty()) // если директория пуста
    {
        if (!dir->r_move(dirPath)) // если удаление не выпонено
            emit not_performed();
    }
}
```

Содержимое файла *threadtosearch.h*:

```
#ifndef THREADTOSEARCH_H
#define THREADTOSEARCH_H

#include <QDir>
#include <QFileInfo>
#include <QObject>
#include <QThread>

class ThreadToSearch : public QObject
{
    Q_OBJECT

public slots:
    void run_search(QString lDirPath, QString rDirPath, QString searchName);

public:
    explicit ThreadToSearch(QObject* parent = nullptr);
    void search(QDir&, QString, QFileInfoList&);

signals:
    void search_finished(QFileInfoList);
};
```

```
#endif // THREADTOSEARCH_H
```

Содержимое файла *threadtosearch.cpp*:

```
#include "threadtosearch.h"

#include <cstdio>
#include <fstream>
#include <iostream>

#include <QDesktopServices>
#include <QList>

using namespace std;

ThreadToSearch::ThreadToSearch(QObject* parent) : QObject{parent} {}

void ThreadToSearch::search(QDir& dir, QString searchName, QFileInfoList&
list)
{
    // проход по текущему диску для поиска объектов с введенным именем
    foreach (QFileInfo info, dir.entryInfoList(QDir::Files | QDir::Dirs |
QDir::NoDotAndDotDot, QDir::Name | QDir::DirsFirst))
    {
        if (info.fileName() == searchName) // сравнение элементов по имени
            list.append(info);
        if (info.isFile() || info.isSymLink()) // если элемент - файл,
переходим к след файлу
            continue;
        if (info.isDir()) // если дошли до этого момента, значит элемент -
директория
        {
            dir.cd(info.fileName());       // заходим в нее
            search(dir, searchName, list); // рекурсивно ищем различия
            dir.cdUp();                    // возврат
        }
    }
}

void ThreadToSearch::run_search(QString lDirPath, QString rDirPath, QString
searchName)
{
    QDir lDir = QDir(lDirPath);
    QDir rDir = QDir(rDirPath);
    QFileInfoList list = QFileInfoList();
    if (!list.isEmpty())
        list.clear();
    if (lDirPath.contains("/home"))
        search(lDir, searchName, list);
    if (rDirPath.contains("/home"))
        search(rDir, searchName, list);
    emit search_finished(list);
}
```