Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе №4

на тему

**РАЗРАБОТКА ЗАЩИЩЕННЫХ ПРИЛОЖЕНИЙ**

Студент Т. П. Власенко

Преподаватель Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_6127s8b6t8wy)

[2 Блок-схема](#_r4y5108cx8yu) алгоритма [4](#_r4y5108cx8yu)

[3 Результат выполнения лабораторной работы](#_jnfz1qbvv5j4) 5

[Выводы](#_ldsbh3e5c0xh) 6

[Приложение А (обязательное) Листинг программного кода](#_54xlhz7mfhe) 7

# 1 ПОСТАНОВКА ЗАДАЧИ

Познакомиться с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз.

Научиться разрабатывать приложения, которые используют ролевое управление доступом для разграничения полномочий пользователей. Получить навыки защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

# 2 БЛОК-СХЕМА АЛГОРИТМА

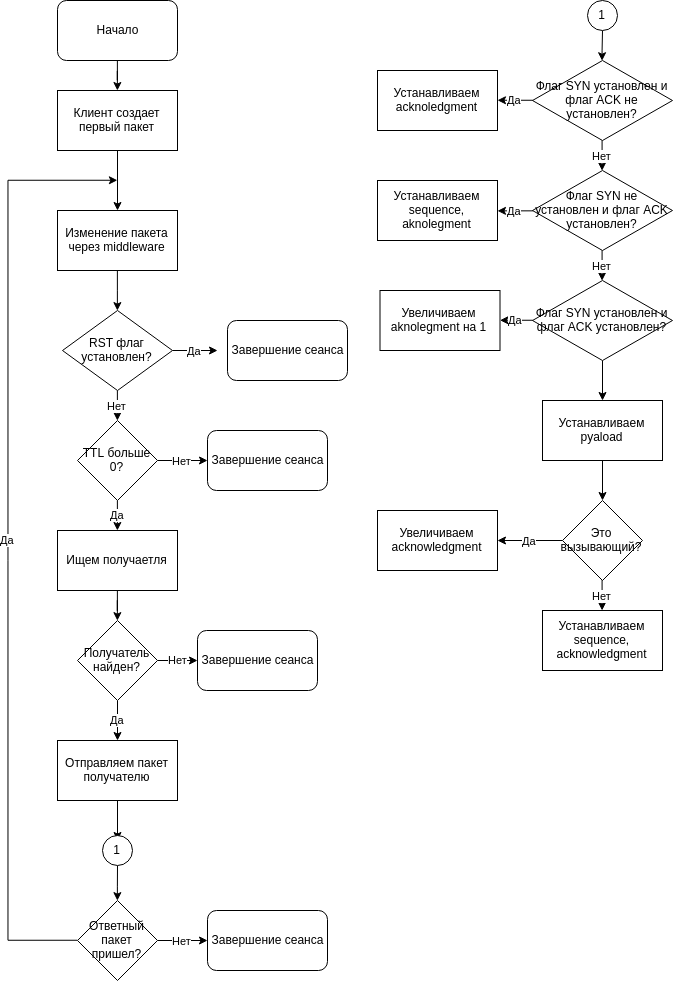
На рисунке 1 продемонстрирована блок-схема алгоритма.

Рисунок 1 – Блок-схема алгоритма программы

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

На рисунках 2, 3 изображены окна, доступные пользователю:

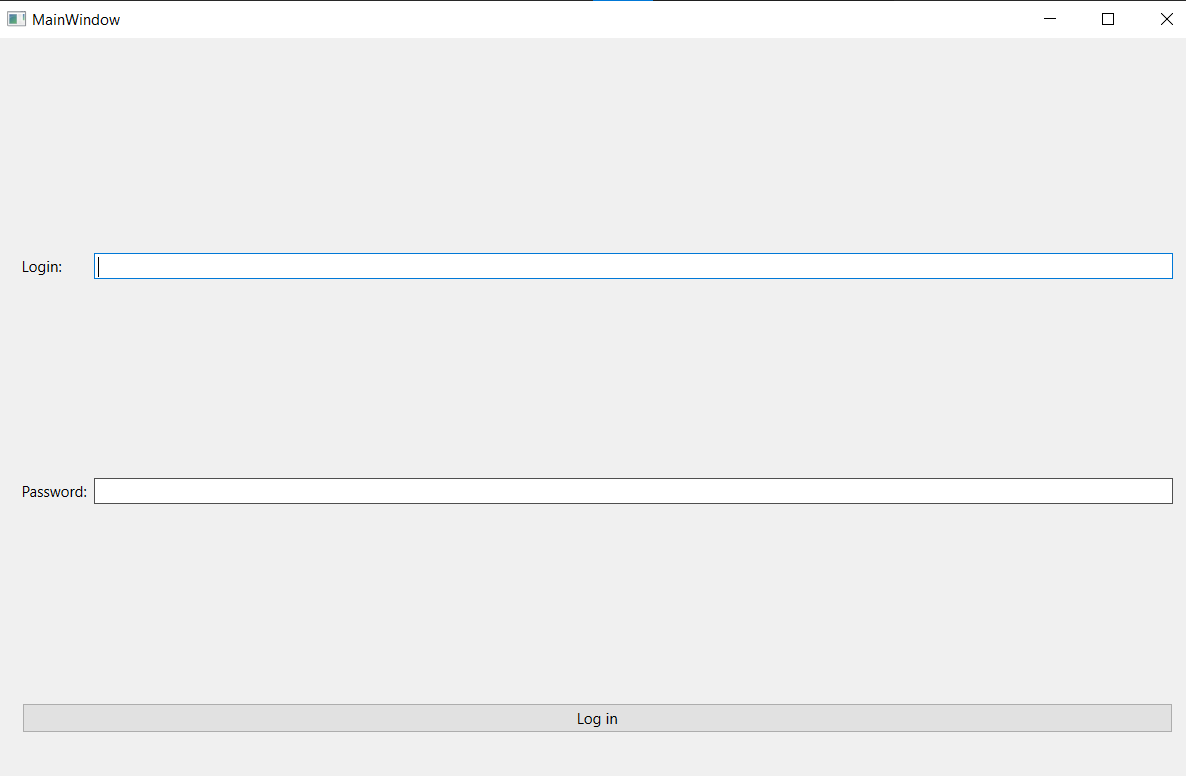


Рисунок 2 – Окно входа

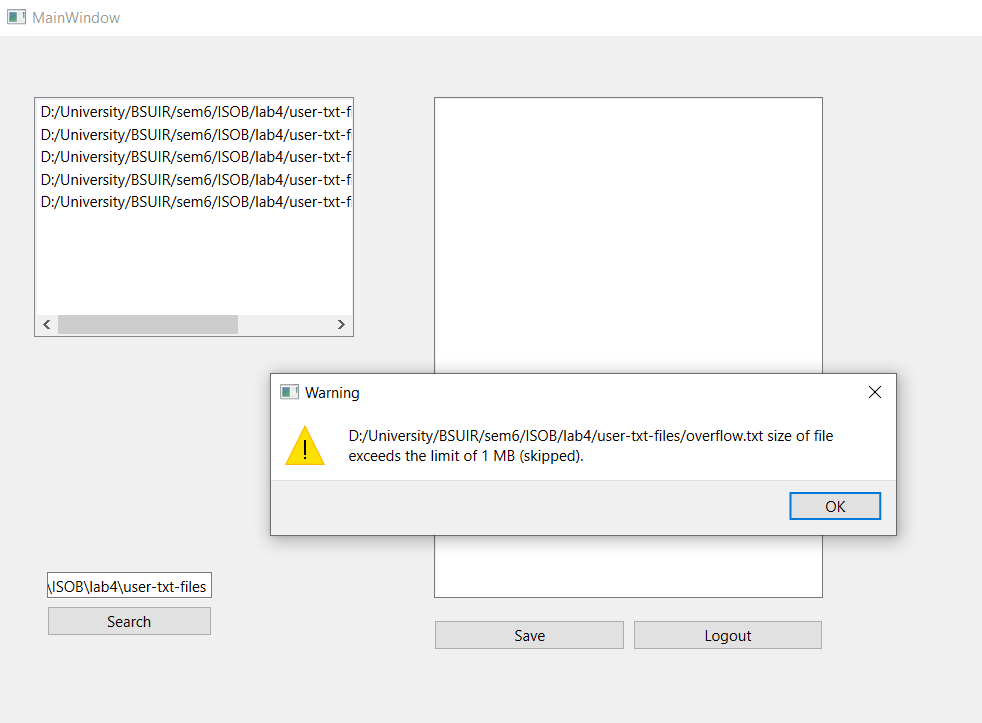


Рисунок 3 – Главное окно пользователя

# ВЫВОДЫ

В ходе выполнения лабораторной работы было разработано оконное приложение, позволяющее пользователю (в зависимости от его роли) искать файлы в папке, читать и изменять их.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

#include "user\_actions.h"

#include "ui\_user\_actions.h"

#include <QRegularExpression>

#include <QRegularExpressionValidator>

#include <QFileInfo>

#include <QFile>

#include <QDir>

#include <QMessageBox>

#include <QDirIterator>

#include <QTimer>

#include "textEditWithInsertChecking.h"

user\_actions::user\_actions(size\_t level, QWidget \*parent) :

QWidget(parent),

privilege\_lvl(level),

ui(new Ui::user\_actions)

{

ui->setupUi(this);

ui->dirPath->setMaxLength(100);

MyTextEdit\* mte = new MyTextEdit(ui->frame\_4);

delete ui->textEdit;

mte->setObjectName("textEdit");

mte->setEnabled(true);

mte->setGeometry(QRect(20, 20, 311, 401));

mte->setMaxLength(10000);

setGeometry(0,0, 500, 500);

}

void user\_actions::set\_privilege\_lvl(int lvl) {

privilege\_lvl = lvl;

if (privilege\_lvl > 0) {

ui->saveBtn->hide();

} else {

ui->saveBtn->show();

}

}

user\_actions::~user\_actions()

{

delete ui;

}

void user\_actions::on\_searchFilesBtn\_clicked()

{

QString dirPath = ui->dirPath->text();

QFileInfo fileInfo(dirPath);

QString canonicalDirPath = fileInfo.canonicalFilePath();

if (canonicalDirPath.startsWith("D:/University/BSUIR/sem6/ISOB/lab4/user-txt-files")) {

const qint64 maxSize = 1024 \* 1024;

int maxFileCnt = 100;

QDirIterator it(canonicalDirPath, QStringList() << "\*.\*", QDir::Files, QDirIterator::Subdirectories);

while (it.hasNext()) {

QString filePath = it.next();

QFileInfo fileInfo(filePath);

qint64 fileSize = fileInfo.size();

if (fileSize > maxSize) {

QMessageBox::warning(this, "Warning", filePath + " size of file exceeds the limit of 1 MB (skipped).");

continue;

}

ui->foundFiles->addItem(filePath);

maxFileCnt--;

if(maxFileCnt == 0) {

QMessageBox::warning(this, "Warning", "Maximum number of files is 100 (other skipped).");

}

}

} else {

QMessageBox::critical(this, "Error", "The specified directory path is invalid.");

}

}

void user\_actions::on\_foundFiles\_itemClicked(QListWidgetItem \*item)

{

if (privilege\_lvl > 1) {

QMessageBox::critical(this, "Error", "You are not allowed to read files.");

} else {

QString fileName = item->text();

QFile file(fileName);

if (file.open(QIODevice::ReadOnly)) {

qDebug() << "Open file";

QTextStream stream(&file);

ui->textEdit->setPlainText(stream.readAll());

file.close();

}

}

}

void user\_actions::on\_saveBtn\_clicked()

{

ui->saveBtn->setEnabled(false);

QTimer\* timer = new QTimer(this);

timer->setSingleShot(true);

timer->setInterval(1000);

QObject::connect(timer, &QTimer::timeout, [timer, this](){

ui->saveBtn->setEnabled(true);

timer->deleteLater();

});

timer->start();

if (ui->foundFiles->currentItem() == nullptr) return;

QString fileName = ui->foundFiles->currentItem()->text();

QFile file(fileName);

if (file.open(QIODevice::WriteOnly)) {

QTextStream stream(&file);

stream << ui->textEdit->toPlainText();

file.close();

}

}

void user\_actions::on\_signOutBtn\_clicked()

{

emit sign\_out();

}