МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА «ИИТ»

Лабораторная работа №6

По дисциплине «ОСиСП»

Выполнила:

студент ФЭИС

группы ПО-5

Нерода А. А.

Проверил:

Дряпко А. В.

Брест 2021

**Цель: научиться создавать программы-инсталляторы, приобрести практические навыки в работе с реестром ОС Windows**

**Задание:**

Для программы, разрабатываемой в ЛР No1-5, создать с использованием программы NSIS ин-

сталлятор, который бы обеспечивал простую установку и «чистое» удаление указанной программы

из системы.

1) Создание инсталлятора в NSIS сводится к написанию скриптового сценария установки ПО, в

который вносятся все основные параметры инсталляции (в частности, директории установки,

поддерживаемый язык и т.д.)

2) Для начала работы с NSIS рекомендуется ознакомиться со справкой и примерами:

а) NSIS Users Manual

б) Simple tutorials

3) Добавить кастомную страницу установки, позволяющую выбрать тип установки «Для текущего

пользователя»/«Для всех пользователей».

**Код программы:**

**Клиент**

**main.cpp**  
  
#include "player.h"

#include <QApplication>

int **main**(int argc, char \*argv[]) {

QApplication a(argc, argv);

Player w;

w.show();

*return* a.exec();

}

**player.cpp**

#include "player.h"

#include "ui\_player.h"

#include "about.h"

#include "helper.h"

*typedef* QString (\*Helper\_set\_name\_window)();

*typedef* QString (\*Helper\_set\_first\_name)();

*typedef* QString (\*Helper\_set\_second\_name)();

Player::Player(QWidget \*parent) : QMainWindow(parent), ui(*new* Ui::Player) {

ui->setupUi(*this*);

*//* *установка* *имени* *окна* *приложения*

QLibrary \*helper\_set = *new* QLibrary("Helper");

Helper\_set\_name\_window set\_window\_name\_ = (Helper\_set\_name\_window)helper\_set->resolve("set\_window\_name");

QString windowTitle = set\_window\_name\_();

*delete* helper\_set;

*this*->setWindowTitle(windowTitle);

*//* *установка* *имени* *первого* *заголовка* *playlistview*

QLibrary \*helper\_set\_f\_n = *new* QLibrary("Helper");

Helper\_set\_first\_name set\_window\_name\_first = (Helper\_set\_first\_name)helper\_set\_f\_n->resolve("set\_first\_name\_HeaderLabels");

QString set\_name\_first = set\_window\_name\_first();

*delete* helper\_set\_f\_n;

*//* *установка* *имени* *второго* *заголовка* *playlistview*

QLibrary \*helper\_set\_s\_n = *new* QLibrary("Helper");

Helper\_set\_second\_name set\_window\_name\_second = (Helper\_set\_second\_name)helper\_set\_s\_n->resolve("set\_second\_name\_HeaderLabels");

QString set\_name\_second = set\_window\_name\_second();

*delete* helper\_set\_s\_n;

ui->just\_playingTrack->setVisible(*false*);

playListModel = *new* QStandardItemModel(*this*);

ui->playlistView->setModel(playListModel);

playListModel->setHorizontalHeaderLabels(QStringList() << tr(set\_name\_first.toStdString().data())<< tr(set\_name\_second.toStdString().data()));

ui->playlistView->hideColumn(1);

ui->playlistView->verticalHeader()->setVisible(*false*);

ui->playlistView->setSelectionMode(QAbstractItemView::SingleSelection); *//* *Разрешаем* *выделять* *только* *одну* *строку*

ui->playlistView->setEditTriggers(QAbstractItemView::NoEditTriggers); *//* *Отключаем* *редактирование*

ui->playlistView->horizontalHeader()->setStretchLastSection(*true*); *//* *по* *ширине* *строки*

player = *new* QMediaPlayer(*this*);

playlist = *new* QMediaPlaylist(player);

player->setPlaylist(playlist);

playlist->setPlaybackMode(QMediaPlaylist::Loop);

load\_mode\_plugins();

load\_interfase\_plugins();

connect(playlist, &QMediaPlaylist::currentIndexChanged, [*this*](int index) {

int delLast = count\_tracks\_num;

Next = index;

Back = index;

*if*(isNext) {

ui->playlistView->model()->setData(ui->playlistView->model()->index(Next, 0),QColor(255, 125, 0), Qt::BackgroundRole);

ui->playlistView->model()->setData(ui->playlistView->model()->index(--Back, 0),QColor(255, 255, 255), Qt::BackgroundRole);

}

*else* {

ui->playlistView->model()->setData(ui->playlistView->model()->index(Next, 0),QColor(255, 125, 0), Qt::BackgroundRole);

ui->playlistView->model()->setData(ui->playlistView->model()->index(++Back, 0),QColor(255, 255, 255), Qt::BackgroundRole);

}

ui->just\_playingTrack->setText(playListModel->data(playListModel->index(index, 0)).toString());

*if*(index == 0) ui->playlistView->model()->setData(ui->playlistView->model()->index(--delLast, 0),QColor(255, 255, 255), Qt::BackgroundRole);

*if*(index == --delLast) ui->playlistView->model()->setData(ui->playlistView->model()->index(0, 0),QColor(255, 255, 255), Qt::BackgroundRole);

});

}

void Player::**load\_mode\_plugins**()

{

QDir dir(qApp->applicationDirPath());

dir.cd("plugins");

QStringList filter;

filter << "\*.dll";

QPluginLoader loader;

*foreach*(QFileInfo info, dir.entryInfoList(filter)) {

loader.setFileName(info.absoluteFilePath());

Interface\* mode = qobject\_cast<Interface\*>(loader.instance());

*if*(mode) {

QString name = loader.metaData().value("MetaData").toObject().value("Mode\_name").toString();

ui->mode\_list->addItem(name);

\_mode.append(mode);

}

}

}

void Player::**load\_interfase\_plugins**() {

QDir dir(qApp->applicationDirPath());

dir.cd("themes\_interfases");

QStringList filter\_1;

filter\_1 << "\*.dll";

QPluginLoader loader\_1;

*foreach*(QFileInfo info, dir.entryInfoList(filter\_1)) {

loader\_1.setFileName(info.absoluteFilePath());

Interface\_theme\* mode1 = qobject\_cast<Interface\_theme\*>(loader\_1.instance());

*if*(mode1) {

QString name = loader\_1.metaData().value("MetaData").toObject().value("interfas\_name").toString();

ui->mode\_list\_2->addItem(name);

\_mode1.append(mode1);

}

}

}

Player::~**Player**() {

*delete* ui;

*delete* playListModel;

*delete* playlist;

*delete* player;

}

void Player::**on\_btn\_add\_clicked**() {

QStringList new\_files = QFileDialog::getOpenFileNames(*this*, tr("Open files"), QString(), tr("Audio Files (\*.mp3)"));

*if*(!new\_files.isEmpty()) {

QStringList write\_to\_globallist;

*if*(count\_tracks\_num != 0) {

*for*(int i = 0; i < new\_files.size(); i++) {

bool flag = *false*;

*for*(int j = 0; j < files\_list.size(); j++) {

*if*(new\_files[i] == files\_list[j]) {

flag = *true*;

}

}

*if*(!flag) {

write\_to\_globallist.append(new\_files[i]);

}

}

add\_playlist(write\_to\_globallist);

}

*else* add\_playlist(new\_files);

}

}

void Player::add\_playlist(QStringList list) {

*//* *устанавливаем* *данные* *по* *именам* *и* *пути* *к* *файлам*

*//* *в* *плейлист* *и* *таблицу* *отображающую* *плейлист*

*foreach* (QString filePath, list) {

files\_list.append(list);

count\_tracks\_num++;

QList<QStandardItem \*> items;

items.append(*new* QStandardItem(QDir(filePath).dirName()));

items.append(*new* QStandardItem(filePath));

playListModel->appendRow(items);

playlist->addMedia(QUrl(filePath));

}

ui->count\_tracks->setText(QString::number(count\_tracks\_num));

}

void Player::**on\_btn\_play\_clicked**() {

player->play();

ui->just\_playingTrack->setVisible(*true*);

}

void Player::**on\_btn\_previous\_clicked**() {

isNext = *false*;

playlist->previous();

}

void Player::**on\_btn\_pause\_clicked**() {

player->pause();

}

void Player::**on\_btn\_stop\_clicked**() {

player->stop();

ui->just\_playingTrack->setVisible(*false*);

}

void Player::**on\_btn\_next\_clicked**() {

isNext = *true*;

playlist->next();

}

void Player::on\_playlistView\_doubleClicked(*const* QModelIndex index) {

*for* (int ind = 0; ind < count\_tracks\_num; ++ind) {

ui->playlistView->model()->setData(ui->playlistView->model()->index(ind, 0),QColor(255, 255, 255), Qt::BackgroundRole);

}

playlist->setCurrentIndex(index.row());

on\_btn\_play\_clicked();

}

void Player::**on\_Info\_clicked**() {

*typedef* void (\*About)();

QLibrary \*aboutLib = *new* QLibrary("About");

About showWindowAboutProgramm = (About)aboutLib->resolve("about");

showWindowAboutProgramm();

*delete* aboutLib;

}

void Player::**on\_reboot\_clicked**() {

qApp->quit();

QProcess::startDetached(qApp->arguments()[0], qApp->arguments());

}

void Player::**on\_apply\_clicked**() {

QApplication::setFont(\_mode.at(ui->mode\_list->currentIndex())->changeView());

}

void Player::**on\_apply\_2\_clicked**() {

QApplication::setPalette(\_mode1.at(ui->mode\_list\_2->currentIndex())->changeInterfase());

}

void Player::**sockDisc**() {

socket->disconnected();

}

void Player::**sockReady**() {

*if*(socket->waitForConnected(100)) {

socket->waitForReadyRead(100);

Data = socket->readAll();

*if*(Data == "Actuale") {

QMessageBox::information(*this*, "Информация", "Соединение установлено\nУ вас актуальная версия программы!");

socket->disconnected();

}

*else* *if*(Data == "Need update") {

QMessageBox msg;

msg.setText("New version is available");

msg.setInformativeText("Do you want to update app?");

msg.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

msg.setDefaultButton(QMessageBox::Yes);

int res = msg.exec();

*//loading*

*if* (res == QMessageBox::Yes) {

sockDisc();

socket->waitForDisconnected(1);

QThread \*thread= *new* QThread;

Thread \*my = *new* Thread("B");

my->moveToThread(thread);

connect(my, *SIGNAL*(time\_load(int)), *this*, *SLOT*(showProgress(int)));

connect(my, *SIGNAL*(load\_update(int)), *this*, *SLOT*(loading(int)));

connect(my, *SIGNAL*(load\_video(int)), *this*, *SLOT*(download\_video(int)));

connect(thread, *SIGNAL*(started()), my, *SLOT*(run()));

thread->start();

}

}

}

}

void Player::**on\_check\_update\_clicked**() {

socket = *new* QTcpSocket();

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

Data.clear();

QDir client\_version(qApp->applicationDirPath());

QStringList filter;

filter << "\*.json";

*foreach*(QFileInfo info, client\_version.entryInfoList(filter)) {

filter.clear();

filter << info.absoluteFilePath();

}

QFile file(filter.back());

*if* (!file.open(QIODevice::ReadOnly))

*return*;

Data = file.readAll();

socket->connectToHost("127.0.0.1", 5555);

socket->waitForConnected(1);

*if*(socket->state() == QTcpSocket::ConnectedState) {

*if*(socket->isOpen()) {

socket->write(Data);

socket->waitForBytesWritten(100);

}

}

*else* {

QMessageBox::information(*this*, "Информация", "Соединение не установлено");

}

}

void Player::**loading**(int res) {

*if*(res == 1) {

\_mode1.clear();

ui->mode\_list\_2->clear();

load\_interfase\_plugins();

}

}

void Player::showProgress(int i) {

progress = i;

ui->statusbar->showMessage(QString::number(progress)+"%");

ui->statusbar->showMessage("100%");

QMessageBox::information(*this*, "Информация", "Обновление завершено!");

}

void Player::**download\_video**(int msg) {

*if*(msg == 1) QMessageBox::information(*this*, "Информация", "Видео получено!");

}

**thread.cpp**

#include "thread.h"

QDir dir\_client(QDir::currentPath() + "/themes\_interfases");

Thread::Thread(QString s) : name(s) {

}

void Thread::sockReady() {

*if*(socket->waitForConnected(100)) {

socket->waitForReadyRead(100);

Data = socket->readAll();

QStringList rec\_data\_update;

QString new\_version;

rec\_data\_update.append(QString(Data).split(" "));

new\_version.append(rec\_data\_update.last());

path\_server = rec\_data\_update.front();

QDir new\_client\_version(QDir::current());

QStringList filter;

filter << "\*.json";

*foreach*(QFileInfo info, new\_client\_version.entryInfoList(filter)) {

filter.clear();

filter << info.absoluteFilePath();

}

QFile file(filter.back());

*if* (!file.open(QIODevice::WriteOnly))

*return*;

file.write(new\_version.toStdString().data());

*for*(int i = 1; i < rec\_data\_update.size()-1; i++) {

QFile::copy(rec\_data\_update.front()+'/'+rec\_data\_update[i], dir\_client.path()+'/'+rec\_data\_update[i]);

}

*//create* *log*

QDir write\_log(QDir::currentPath() + "/logs");

QStringList formatFile;

formatFile << "\*.txt";

*foreach*(QFileInfo info, write\_log.entryInfoList(formatFile)) {

formatFile.clear();

formatFile << info.absoluteFilePath();

}

QFile log(formatFile.front());

*if* (!log.open(QIODevice::WriteOnly))

*return*;

QString text = "Update Modules and version "+ rec\_data\_update.back();

log.write(text.toStdString().data());

}

*//* *sockDisc();*

*emit* load\_update(1);

share\_video();

}

void Thread::**sockDisc**() {

socket->disconnected();

}

void Thread::**run**() {

socket = *new* QTcpSocket();

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

socket->connectToHost("127.0.0.1", 5555);

Data.clear();

QStringList find\_filter;

bool ok = dir\_client.exists();

*if* (ok) {

dir\_client.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);

dir\_client.setSorting(QDir::Name);

QFileInfoList list = dir\_client.entryInfoList();

*for* (int i = 0; i < list.size(); ++i) {

QFileInfo fileInfo = list.at(i);

find\_filter.append(fileInfo.fileName());

}

qDebug() << find\_filter << endl;

}

*foreach* (*const* QString &str, find\_filter) {

Data.append(str);

*if*(&str != find\_filter.last()) {

Data.append(" ");

}

}

qDebug() << Data << endl; *//get* *list* *themes\_interfases* *client*

socket->write(Data);

socket->waitForBytesWritten(100);

*emit* time\_load(58);

}

void Thread::**share\_video**() {

*if*(QFile::copy(path\_server+"/video/DCs.Legends.of.Tomorrow.S06E11.1080p.rus.LostFilm.TV.mkv", dir\_client.path() + "/video/DCs.Legends.of.Tomorrow.S06E11.1080p.rus.LostFilm.TV.mkv")) *emit* load\_video(1);

}

**player.h**

#ifndef PLAYER\_H

#define PLAYER\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QStandardItemModel>

#include <QMessageBox>

#include <QMediaPlayer>

#include <QMediaPlaylist>

#include <QProcess>

#include <QLibrary>

#include <QPluginLoader>

#include <QTcpSocket>

#include <QThread>

#include "thread.h"

#include "interface.h"

#include "interface\_theme.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **Player**; }

QT\_END\_NAMESPACE

*class* Player : *public* QMainWindow {

Q\_OBJECT

*public*:

Player(QWidget \*parent = *nullptr*);

~Player();

void load\_mode\_plugins();

void load\_interfase\_plugins();

*private* *slots*:

void showProgress(int i);

void loading(int res);

void download\_video(int msg);

void add\_playlist(QStringList list);

void on\_btn\_add\_clicked();

void on\_btn\_play\_clicked();

void on\_btn\_previous\_clicked();

void on\_btn\_pause\_clicked();

void on\_btn\_stop\_clicked();

void on\_btn\_next\_clicked();

void on\_playlistView\_doubleClicked(*const* QModelIndex index);

void on\_Info\_clicked();

void on\_reboot\_clicked();

void on\_apply\_clicked();

void on\_apply\_2\_clicked();

void sockDisc();

void sockReady();

void on\_check\_update\_clicked();

*private*:

Ui::Player \*ui;

QList<Interface\*> \_mode;

QList<Interface\_theme\*> \_mode1;

QStringList files\_list;

int count\_tracks\_num = 0;

QStandardItemModel \*playListModel;

QMediaPlayer \*player;

QMediaPlaylist \*playlist;

int Next, Back;

bool isNext = *true*;

QTcpSocket\* socket;

QByteArray Data;

QThread \*thread\_update;

int progress = 0;

};

#endif *//* *PLAYER\_H*

**about.h**

#ifndef ABOUT\_H

#define ABOUT\_H

#include "about\_global.h"

*extern* "C" ABOUT\_EXPORT void about();

#endif *//* *ABOUT\_H*

**about\_global.h**

#ifndef ABOUT\_GLOBAL\_H

#define ABOUT\_GLOBAL\_H

#include <QtCore/qglobal.h>

#if defined(ABOUT\_LIBRARY)

# define ABOUT\_EXPORT Q\_DECL\_EXPORT

#else

# define ABOUT\_EXPORT Q\_DECL\_IMPORT

#endif

#endif *//* *ABOUT\_GLOBAL\_H*

**helper.h**

#ifndef HELPER\_H

#define HELPER\_H

#include "helper\_global.h"

#include <QString>

*extern* "C" HELPER\_EXPORT QString set\_window\_name();

*extern* "C" HELPER\_EXPORT QString set\_first\_name\_HeaderLabels();

*extern* "C" HELPER\_EXPORT QString set\_second\_name\_HeaderLabels();

#endif *//* *HELPER\_H*

**helper\_global.h**

#ifndef HELPER\_GLOBAL\_H

#define HELPER\_GLOBAL\_H

#include <QtCore/qglobal.h>

#if defined(HELPER\_LIBRARY)

# define HELPER\_EXPORT Q\_DECL\_EXPORT

#else

# define HELPER\_EXPORT Q\_DECL\_IMPORT

#endif

#endif *//* *HELPER\_GLOBAL\_H*

**thread.h**

#ifndef THREAD\_H

#define THREAD\_H

#include <QTcpSocket>

#include <QThread>

#include <QTimer>

#include <QDir>

#include <QFile>

*class* Thread : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* Thread(QString name);

QTcpSocket\* socket;

QByteArray Data;

int progress = 0;

QTimer \*timer;

*public* *slots*:

void sockReady();

void **sockDisc**();

void **run**();

void **share\_video**();

*signals*:

void time\_load(int);

void **load\_update**(int);

void **load\_video**(int);

*private*:

QString name;

QString path\_server;

};

#endif *//* *THREAD\_H*

**Сервер**

**main.cpp**  
  
#include <QCoreApplication>

#include <myserver.h>

int **main**(int argc, char \*argv[])

{

QCoreApplication a(*argc*, *argv*);

MyServer Server;

Server.StartServer();

*return* a.exec();

}

**myserver.cpp**

#include "myserver.h"

MyServer::**MyServer**(){}

MyServer::~***MyServer***(){}

void MyServer::**StartServer**(){ *//если* *он* *стартовал,* *то* *сервер* *готов* *к* *работе*

*if*(*this*->listen(QHostAddress::*Any*, 5555)) {

qDebug() << "Listening";

}

*else* {

qDebug() << "Not Listening";

}

}

void MyServer::**incomingConnection**(int socketDeskriptor) { *//инициализация* *подключаемого* *клиента*

socket = *new* QTcpSocket(*this*);

socket->*setSocketDescriptor*(socketDeskriptor);

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

connect(socket,SIGNAL(disconnected()),*this*,SLOT(sockDisc()));

qDebug()<<socketDeskriptor<<"Client connected";

qDebug()<<"Send client connect status - YES";

}

void MyServer::**sockReady**() { *//что* *получает,* *когда* *происходит* *получение* *от* *клиента*

Data = socket->readAll();

qDebug() << "Select from Client" << Data;

*if*(!Data.isEmpty()) { *//если* *что-то* *получило,* *то*

QDir server\_version(QDir::currentPath());

QStringList filter;

filter << "\*.json";

foreach(QFileInfo info, server\_version.entryInfoList(filter)) {

filter.clear();

filter << info.absoluteFilePath();

}

qDebug() << filter;

QFile file(filter.back()); *//сервер* *заходит* *в* *свою* *папку* *и* *получает* *свою* *версию*

*if* (!file.*open*(QIODevice::*ReadOnly*))*//если* *версии* *равные,* *то* *он* *возвр* *клиенту,* *что* *версия* *актуальна*

*return*;

path\_to\_Download = file.readAll();

*if*(QString(Data) == path\_to\_Download) {

qDebug() << "Send to Client" << "Actuale";

socket->write("Actuale");

}

*else* *if*(Data[0] == 'T') { *//* *если* *клиент* *отпр* *список* *своих* *файлов*

QStringList client\_data;

client\_data.append(QString(Data).split(" ")); *//помещение* *всего* *списка* *клиента*

Data.clear(); *//очистить* *для* *отправки*

QDir dir\_server(QDir::currentPath() + "/themes\_interfases"); *//сервер* *заходит* *в* *свои* *темы*

QStringList find\_filter;

Data.append(dir\_server.path()+' ');

bool ok = dir\_server.exists();

*if* (ok)

{

dir\_server.setFilter(QDir::*Files* | QDir::*Hidden* | QDir::*NoSymLinks*);

dir\_server.setSorting(QDir::*Name*);

QFileInfoList list = dir\_server.entryInfoList();

*for* (int i = 0; i < list.size(); ++i)

{

QFileInfo fileInfo = list.at(i);

find\_filter.append(fileInfo.fileName());

}

}

QStringList sen\_to\_clien;

*for*(int i = client\_data.size(); i < find\_filter.size(); i++) {

sen\_to\_clien.append(find\_filter[i]);

}

foreach (*const* QString &str, sen\_to\_clien)

{

Data.append(str);

*if*(&str != find\_filter.last()) {

Data.append(" ");

}

*else* {

Data.append(" ");

Data.append(path\_to\_Download);

}

}

qDebug() << "Send to Client" << Data;

socket->write(Data); *//записывает* *свои* *и* *помещает* *в* *переменную,* *которую* *нужно* *отправить* *клиенту*

}

*else* {

qDebug() << "Send to Client" << "Need update"; *//если* *неактуальные,* *то* *отправляет* *клиенту* *сообщение* *об* *этом*

socket->write("Need update");

}

socket->*waitForBytesWritten*(100);

}

}

void MyServer::**sockDisc**(){ *//если* *отключился* *клиент*

qDebug()<<"Disconnect";

socket->deleteLater();

}

**myserver.h**

#ifndef MYSERVER\_H

#define MYSERVER\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QDir>

#include <QFile>

*class* MyServer: *public* QTcpServer{

Q\_OBJECT

*public*:

MyServer();

~MyServer();

QTcpSocket \* socket;

QByteArray Data;

QString path\_to\_Download;

*public* *slots*:

void StartServer();

void incomingConnection(int socketDeskriptor);

void sockReady();

void sockDisc();

};

#endif *//* *MYSERVER\_H*