МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: «ОСИСП»

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Белко В.А.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** познакомиться с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений

Вариант 2

**Задание:**

1) Основное задание заключается в доработке функционала обновления, разработка которого про-

изводилась в ЛР №4. Нужно интегрировать указанную функцию в само приложение, без ис-

пользования стороннего клиента. При этом серверная часть приложения остается без изменений(возможны некоторые доработки сервера, без изменения общей клиент-серверной архитектуры);

2) Проверка обновления должна осуществляться автоматически по таймеру (QTimer) либо по непосредственному запросу пользователя. Предусмотреть выбор из меню политики обновления (с пользовательским подтверждением, без подтверждения/автоматически);

3) Сам процесс обновления должен осуществляться с использованием отдельного потока (QThread) с минимальной вовлечённостью пользователя;

4) Необходимо отображать прогресс обновления (для этого можно использовать строку состояния – QStatusBar);

5) Для демонстрации процесса обновления и независимой работы основного и вспомогательного потоков приложения осуществлять передачу с сервера обновления помимо основных обновляемых компонентов (в соответствии с вариантом задания) одного-двух крупных файлов с произвольным содержимым (например, видео).

6) Обновляемые компоненты по вариантам (ЛР №4):

2 DLL, конфигурационный файл (внешний вид)

7) Процесс обновления логируется. При завершении обновления пользователю выдается соответствующее сообщение.

**mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <iostream>

#include <math.h>

#include <string>

#include <QDialog>

#include <QMessageBox>

#include <QLibrary>

#include <QDebug>

#include <QDir>

#include <QPluginLoader>

#include <QTcpSocket>

#include <QThread>

#include "thread.h"

#include "interface.h"

#include "interface\_theme.h"

using namespace std;

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

void load\_mode\_plugins();

void load\_interface\_plugins();

~MainWindow();

QThread \*thread\_update;

private slots:

void showProgress(int i);

void loading(int res);

void download\_video(int msg);

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

void Two\_Ten();

void Eight\_Ten();

void Two\_Six();

void Eight\_Six();

void Ten\_Six();

void on\_pushButton\_6\_clicked();

void on\_pushButton\_7\_clicked();

void sockDisc();

void sockReady();

void applyPlugin(int index);

void applyPlugin\_theme(int index);

void on\_pushButton\_8\_clicked();

private:

Ui::MainWindow \*ui;

int progress = 0;

int Flag = 0;

int Value = 0;

QVector< Interface\* > mPlugins;

QVector< Interface\_theme\* > mTheme;

QTcpSocket\* socket;

QByteArray Data;

};

#endif // MAINWINDOW\_H

**main.cpp:**

#include "mainwindow.h"

#include <QApplication>

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

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

{

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

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

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

ui->setupUi(this);

load\_mode\_plugins();

load\_interface\_plugins();

// 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 кнопки

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

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

ui->pushButton->setText(set\_btn\_name\_());

delete helper\_set\_n\_btn;

// установка имени About

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

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

ui->pushButton\_7->setText(set\_window\_name\_first());

delete helper\_set\_n\_about;

}

void MainWindow::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) {

this->mPlugins.push\_back(mode);

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

ui->comboBox->addItem(name);

connect(ui->comboBox, SIGNAL(activated(int)), this, SLOT(applyPlugin(int)));

}

}

}

void MainWindow::load\_interface\_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) {

this->mTheme.push\_back(mode1);

QString name1 = loader\_1.metaData().value("MetaData").toObject().value("interfas\_name").toString();

ui->comboBox\_2->addItem(name1);

connect(ui->comboBox\_2, SIGNAL(activated(int)), this, SLOT(applyPlugin\_theme(int)));

}

}

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::Two\_Ten() {

if(Flag==1) {

QString inputNum = ui->lineEdit->text();

QByteArray ba = inputNum.toLocal8Bit();

char \*s= ba.data();

char \*end;

long a = strtol (s, &end, 2);

QString num = QString::number(a);

ui->lineEdit->setText(num);

}

}

void MainWindow::Two\_Six() {

if(Flag==1) {

QString inputNum = ui->lineEdit->text();

QByteArray ba = inputNum.toLocal8Bit();

char \*s= ba.data();

char \*end;

long a = strtol (s, &end, 2);

string digits[16] = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

string hex;

do {

hex.insert (0, digits[a % 16]);

a /= 16;

}

while (a!=0);

QString num = QString::fromUtf8((hex).c\_str());

ui->lineEdit->setText(num);

}

}

void MainWindow::Eight\_Six(){

if(Flag==2) {

QString inputNum = ui->lineEdit->text();

string numb = inputNum.toUtf8().constData();

int res = 0;

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

res \*= 8;

res += (numb[i] - '0');

}

string digits[16] = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

string hex;

do {

hex.insert (0, digits[res % 16]);

res /= 16;

}

while (res!=0);

QString num = QString::fromUtf8((hex).c\_str());

ui->lineEdit->setText(num);

}

}

void MainWindow::Ten\_Six(){

if (Flag==3) {

QString inputNum = ui->lineEdit->text();

QByteArray ba = inputNum.toLocal8Bit();

char \*s= ba.data();

char \*end;

long a = strtol (s, &end, 10);

string digits[16] = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

string hex;

do {

hex.insert (0, digits[a % 16]);

a /= 16;

}

while (a!=0);

QString num = QString::fromUtf8((hex).c\_str());

ui->lineEdit->setText(num);

}

}

void MainWindow::Eight\_Ten() {

if(Flag==2) {

QString inputNum = ui->lineEdit->text();

string numb = inputNum.toUtf8().constData();

int res = 0;

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

res \*= 8;

res += (numb[i] - '0');

}

QString num = QString::fromUtf8(to\_string(res).c\_str());

ui->lineEdit->setText(num);

}

}

void MainWindow::on\_pushButton\_clicked()

{

Flag = 1;

}

void MainWindow::on\_pushButton\_2\_clicked() {

Flag = 2;

}

void MainWindow::on\_pushButton\_3\_clicked() {

Flag = 3;

}

void MainWindow::on\_pushButton\_4\_clicked() {

Value = 1;

if(Flag==1) {

Two\_Six();

} else if (Flag==2) {

Eight\_Six();

} else if (Flag==3) {

Ten\_Six();

}

}

void MainWindow::on\_pushButton\_5\_clicked() {

Value = 2;

if(Flag==1) {

Two\_Ten();

} else if (Flag==2) {

Eight\_Ten();

}

}

void MainWindow::on\_pushButton\_6\_clicked() {

ui->lineEdit->clear();

}

void MainWindow::on\_pushButton\_7\_clicked() {

typedef void (\*about)();

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

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

showWindowAboutProgramm();

delete aboutLib;

}

void MainWindow::sockDisc() {

socket->disconnected();

}

void MainWindow::sockReady() {

if(socket->waitForConnected(100)) {

socket->waitForReadyRead(100);

Data = socket->readAll();

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

if(Data == "Actuale") {

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

socket->disconnected();

}

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

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;

socket->write(Data);

socket->waitForBytesWritten(100);

}

else {

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 MainWindow::applyPlugin(int index) {

QFont pallete = this->mPlugins[index]->changeView();

QApplication::setFont(pallete);

}

void MainWindow::applyPlugin\_theme(int index) {

QPalette theme = this->mTheme[index]->changeInterfase();

QApplication::setPalette(theme);

}

void MainWindow::loading(int res) {

if(res == 1) {

mTheme.clear();

ui->comboBox\_2->clear();

load\_interface\_plugins();

}

}

void MainWindow::showProgress(int i) {

progress = i;

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

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

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

}

void MainWindow::download\_video(int msg) {

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

}

void MainWindow::on\_pushButton\_8\_clicked() {

QMessageBox msg;

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 {

msg.setWindowTitle("Информация");

msg.setText("Соединение не установлено");

msg.exec();

}

}

**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

**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);

}

**Mode.h**

#ifndef MODE\_H

#define MODE\_H

#include <QObject>

#include <interface.h>

class Mode : public QObject, public Interface

{

Q\_OBJECT

Q\_PLUGIN\_METADATA(IID "calc.Interface" FILE "Interface.json")

Q\_INTERFACES(Interface)

public:

Mode(QObject \*parent =0);

~Mode();

QString name();

virtual QFont changeView();

};

#endif // MODE\_H

**Mode.cpp**

#include "mode.h"

#include <QDebug>

Mode::Mode(QObject \*parent) : QObject(parent)

{

qDebug() << name() << "created";

}

Mode::~Mode()

{

qDebug() << name() << "destroy";

}

QString Mode::name()

{

return "mode 1";

}

QFont Mode::changeView()

{

QFont palette("Comic Sans MS", 10, QFont::Bold, true);

return palette;

}

**Interface.h**

#ifndef INTERFACE\_H

#define INTERFACE\_H

#include <QObject>

#include <QString>

#include <QFont>

class Interface

{

public:

virtual QFont changeView() = 0;//плагин выполняет действия над объектами в структуре Styles

};

Q\_DECLARE\_INTERFACE(Interface, "calc.Interface");

#endif // INTERFACE\_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\_name\_About();

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

#endif // HELPER\_H

**Helper.cpp**

#include "helper.h"

QString set\_window\_btn\_name() {

return QString("Двоичная");

}

QString set\_window\_name() {

return QString("CALCULATOR");

}

QString set\_name\_About() {

return QString("О программе");

}

**About.h**

#ifndef ABOUT\_H

#define ABOUT\_H

#include "about\_global.h"

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

#endif // ABOUT\_H

**About.cpp**

#include "about.h"

#include <QMessageBox>

void about()

{

QMessageBox msgBox;

msgBox.setWindowTitle("About programm");

msgBox.setText("Эту программу сделал Белко Владислав");

msgBox.exec();

}

**Server**

**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(){

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

qDebug() << "Listening";

}

else {

qDebug() << "Not Listening";

}

}

void MyServer::incomingConnection(int socketDeskriptor) {

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() {

Data = socket->readAll();

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

if(!Data.isEmpty()) {

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());

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

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') {

QStringList client\_data;

client\_data.append(QString(Data).split(" "));

Data.clear();

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

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);

}

else {

qDebug() << "Send to Client" << "Need update";

socket->write("Need update");

}

socket->waitForBytesWritten(100);

}

}

void MyServer::sockDisc(){

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

Вывод: познакомился с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений