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

Пищик А.В

Проверила:

Дряпко А.В.

Брест 2021

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

Вариант 15

*Задание:*

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

изводилась в ЛР No4. 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 архитекту-

ры);

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 обнов-

ляемых компонентов (в соответствии с вариантом задания) одного-двух крупных файлов с

произвольным содержимым (например, видео).

6) Обновляемые компоненты по вариантам (ЛР No4):

15 DLL, конфигурационный файл (внешний вид)

7) Процесс обновления логируется. При завершении обновления пользователю выдается соответ-

ствующее сообщение.

***mainwindow.h:***

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QProcess>

#include "interface.h"

#include <QTcpSocket>

#include <QMessageBox>

#include <QDir>

#include <QJsonObject>

#include <windows.h>

#include <QPluginLoader>

#include <QDebug>

#include <iostream>

#include <fstream>

#include <string>

#include <QTimer>

#include <QThread>

#include "thread.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);

~MainWindow();

void load\_mode\_plugins();

QTcpSocket\* socket;

QByteArray Data;

int progress = 0;

QTimer \*timer;

public slots:

void showProgress(int i);

void sockReady();

void sockDisc();

void loading(int res);

void download\_video(int msg);

private slots:

void on\_processor\_clicked(bool checked);

void result\_update(int size\_cols, QStringList header, QStringList mass);

void update\_str(string s);

void on\_mother\_clicked();

void on\_os\_clicked();

void on\_info\_programm\_clicked();

void on\_ram\_clicked();

void on\_hard\_clicked();

void on\_apply\_clicked();

void on\_reboot\_clicked();

void on\_check\_update\_clicked();

public:

Ui::MainWindow \*ui;

QList<Interface\*> \_mode;

QThread \*thread\_update;

};

#endif // MAINWINDOW\_H

***main.cpp:***

#include "mainwindow.h"

#include <QApplication>

#include <QLabel>

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

{

qputenv("QT\_DEBUG\_PLUGINS", QByteArray("1"));

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

***mainwindow.cpp***

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "helper.h"

#include "about.h"

#include <QThread>

#include <QDebug>

#include "thread.h"

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

{

AllocConsole();

ShowWindow(GetConsoleWindow(), SW\_HIDE); // hide console

ui->setupUi(this);

ui->tableWidget->setEditTriggers(0);

bool checked = true;

on\_processor\_clicked(checked);

load\_mode\_plugins();

//connect(&thread\_update, &QThread::started, &th\_one, &UPDATE::run);

//th\_one.moveToThread(&thread\_update);

}

string \*res = new string[100];

QStringList header, result\_data, mass;

int size\_str = 0;

MainWindow::~MainWindow()

{

delete ui;

}

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) {

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

qDebug() << name;

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

\_mode.append(mode);

}

}

}

void MainWindow::result\_update(int size\_cols, QStringList header, QStringList mass) {

string result[size\_cols];

string s;

ifstream f1;

f1.open("test1.txt");

int i = 0;

while (!f1.eof()) {

getline(f1, s);

result[i] = s;

i++;

}

f1.close();

Helper a;

a.destroy\_file("test1.txt");

for (int i = 0; i < size\_cols; i++)

{

res[i] = result[i];

}

ui->tableWidget->setRowCount(size\_cols); // Устанавливаем количество строк на size\_cols

ui->tableWidget->setColumnCount(1);

for(int i = 0; i < size\_cols; i++) {

ui->tableWidget->verticalHeader()->setSectionResizeMode(i, QHeaderView::Stretch);

}

ui->tableWidget->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);

ui->tableWidget->setVerticalHeaderLabels(header);

ui->tableWidget->horizontalHeader()->setVisible(false);

for(int i = 0; i < size\_cols; i++) {

QString data = QString::fromStdString(res[i]);

result\_data.append(data);

}

int countt = 0;

for (int i = countt; i < ui->tableWidget->rowCount(); ++i) {

QTableWidgetItem \* item;

for (int j = 0; j < ui->tableWidget->columnCount(); ++j) {

item = new QTableWidgetItem;

for(int s = 0; s < size\_str; s++) {

if(i==s)

item->setText(result\_data.value(mass[s].toInt()));

ui->tableWidget->setItem(i, j, item);

}

}

}

countt++;

ui->tableWidget->resizeColumnsToContents();

ui->tableWidget->resizeRowsToContents();

}

void MainWindow::update\_str(string s) {

string new\_str;

bool flag = false;

int flag1 = 0;

for (int i = 0; i < s.length(); i++) {

if (s[i] == '|') {

flag = false;

flag1++;

}

if (flag1 == 1) {

new\_str += '\n';

flag1++;

}

if (flag == true) {

new\_str += s[i];

}

if (s[i] == '\n' || (s[i] == ' ' && s[++i] == ' ')) {

flag = false;

}

if (s[i] == '=') {

size\_str++;

flag = true;

}

}

ofstream ff;

ff.open("test1.txt");

ff << new\_str;

ff.close();

}

void MainWindow::on\_processor\_clicked(bool checked) {

size\_str = 0;

ui->tableWidget->setVisible(true);

ui->switch\_info->setText("Информация о процессоре");

header.clear(), result\_data.clear(), mass.clear();

Helper d;

const char \*take = d.set\_apply\_do(0).data(); //proc

system(take);

Helper a;

update\_str(a.read\_file\_info());

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

for(int i = 0; i < size\_str; i++) {

QString m = QVariant(i).toString();

mass.append(m);

}

result\_update(size\_str, header<<"Частота на ядро"<<"Имя процессора"<<"Количество ядер"<<"Количество потоков", mass);

}

void MainWindow::on\_mother\_clicked() {

size\_str = 0;

ui->tableWidget->setVisible(true);

header.clear(), result\_data.clear(), mass.clear();

ui->switch\_info->setText("Информация о Материнской плате");

Helper d;

const char \*take = d.set\_apply\_do(1).data(); //mather

system(take);

Helper a;

update\_str(a.read\_file\_info());

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

for(int i = 0; i < size\_str; i++) {

QString m = QVariant(i).toString();

mass.append(m);

}

result\_update(size\_str, header<<"Производитель"<<"Модель", mass);

}

void MainWindow::on\_os\_clicked() {

size\_str = 0;

ui->tableWidget->setVisible(true);

header.clear(), result\_data.clear(), mass.clear();

ui->switch\_info->setText("Информация об Операционной системе");

Helper d;

const char \*take = d.set\_apply\_do(4).data(); //OS

system(take);

Helper a;

update\_str(a.read\_file\_info());

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

for(int i = 0; i < size\_str; i++) {

QString m = QVariant(i).toString();

mass.append(m);

}

result\_update(size\_str, header<<"Производитель OS"<<"Версия", mass);

}

void MainWindow::on\_ram\_clicked() {

size\_str = 0;

ui->tableWidget->setVisible(true);

header.clear(), result\_data.clear(), mass.clear();

ui->switch\_info->setText("Информация об Модулях оперативной памяти");

Helper d;

const char \*take = d.set\_apply\_do(2).data(); //ram

system(take);

Helper a;

update\_str(a.read\_file\_info());

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

int r = size\_str/2;

for(int i = 0; i < r; i++) {

header<<"Партийный номер"<<"Частота";

}

for(int i = 0; i < size\_str; i++) {

QString m = QVariant(i).toString();

mass.append(m);

}

result\_update(size\_str, header, mass);

}

void MainWindow::on\_hard\_clicked() {

size\_str = 0;

ui->tableWidget->setVisible(true);

header.clear(), result\_data.clear(), mass.clear();

ui->switch\_info->setText("Информация об Накопителях");

Helper d;

const char \*take = d.set\_apply\_do(3).data(); //hard

system(take);

Helper a;

update\_str(a.read\_file\_info());

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

int r = size\_str/2;

for(int i = 0; i < r; i++) {

header<<"Модель"<<"Размер";

}

for(int i = 0; i < size\_str; i++) {

QString m = QVariant(i).toString();

mass.append(m);

}

result\_update(size\_str, header, mass);

}

void MainWindow::on\_info\_programm\_clicked() {

About ab(this);

ab.exec();

}

void MainWindow::on\_apply\_clicked() {

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

}

void MainWindow::on\_reboot\_clicked() {

qApp->quit();

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

}

void MainWindow::on\_check\_update\_clicked() {

socket = new QTcpSocket();

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

Data.clear();

string result\_client;

ifstream f;

f.open("C:\\Users\\Toketsu\\Desktop\\QT\_3 \_lab\\lab1\\debug\\client.json");

getline(f, result\_client);

f.close();

QString data\_client = QString::fromStdString(result\_client);

Data.append(data\_client);

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 MainWindow::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 MainWindow::sockDisc() {

socket->disconnected();

}

void MainWindow::loading(int res) {

if(res == 1) {

\_mode.clear();

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

load\_mode\_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!");

}

**Thread.h**

#ifndef THREAD\_H

#define THREAD\_H

#include <QTcpSocket>

#include <QThread>

#include <QTimer>

#include <iostream>

#include <fstream>

#include <QDir>

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;

};

#endif // THREAD\_H

**Thread.cpp**

#include "thread.h"

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

}

void Thread::sockReady() {

if(socket->waitForConnected(100)) {

socket->waitForReadyRead(100);

Data = socket->readAll();

QStringList rec\_data\_update;

std::ofstream write\_json\_clien;

QString new\_version;

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

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

write\_json\_clien.open("C:\\Users\\Toketsu\\Desktop\\QT\_3 \_lab\\lab1\\debug\\client.json");

write\_json\_clien << new\_version.toStdString();

write\_json\_clien.close();

for(int i = 0; i < rec\_data\_update.size()-1; i++) {

QFile::copy("C:\\Users\\Toketsu\\Desktop\\test server\\server\\server\\debug\\plugins\\"+rec\_data\_update[i], "C:\\Users\\Toketsu\\Desktop\\QT\_3 \_lab\\lab1\\debug\\plugins\\"+rec\_data\_update[i]);

}

//create log

std::ofstream write\_log;

write\_log.open("C:/Users/Toketsu/Desktop/test server/server/server/debug/logs/log.txt", std::ios::app);

write\_log << std::endl;

write\_log << "Update Modules and version " << QString(Data).toStdString();

write\_log.close();

}

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);

QByteArray wr;

QStringList find\_filter;

QDir dir\_client(QDir::current());

dir\_client.cd("plugins");

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) {

wr.append(str);

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

wr.append(" ");

}

}

qDebug() << wr << endl; //get list plugins client

socket->write(wr);

socket->waitForBytesWritten(100);

emit time\_load(58);

}

void Thread::share\_video() {

if(QFile::copy("D:\\DCs.Legends.of.Tomorrow.S06E11.1080p.rus.LostFilm.TV.mkv", "C:\\Users\\Toketsu\\Desktop\\QT\_3 \_lab\\lab1\\debug\\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 "aida.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()

{

qDebug() << name() << "print";

QFont newFont("Courier", 6, QFont::Bold, true);

return newFont;

}

***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, "aida.Interface");

#endif // INTERFACE\_H

***Helper.h***

#ifndef HELPER\_H

#define HELPER\_H

#include "helper\_global.h"

#include <QString>

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class HELPER\_EXPORT Helper

{

public:

Helper();

void destroy\_file(QString s);

string read\_file\_info();

string set\_apply\_do(int num);

};

#endif // HELPER\_H

***Helper.cpp***

#include "helper.h"

Helper::Helper()

{

}

void Helper::destroy\_file(QString s)

{

string name;

name = s.toStdString().c\_str();

const char \*r = name.data();

remove(r);

}

string Helper::read\_file\_info() {

ifstream f;

string s;

f.open("test.txt");

while (!f.eof()) {

char c = f.get();

s += c;

}

f.close();

destroy\_file("test.txt");

return s;

}

string Helper::set\_apply\_do(int num)

{

string \*mass = new string[5];

mass[0] = "wmic CPU get Name, NumberOfCores, NumberOfLogicalProcessors, Maxclockspeed /format:list >> test.txt"; //proc

mass[1] = "wmic baseboard get manufacturer, product /format:list >> test.txt"; //mather

mass[2] = "wmic memorychip get speed, PartNumber /format:list >> test.txt"; //ram

mass[3] = "wmic diskdrive get model, size /format:list >> test.txt"; //hard

mass[4] = "wmic OS get Manufacturer , Version /format:list >> test.txt"; //OS

return mass[num];

}

***About.h***

#ifndef ABOUT\_H

#define ABOUT\_H

#include "about\_global.h"

#include <QDialog>

class ABOUT\_EXPORT About : public QDialog

{

public:

explicit About(QWidget \*parent = nullptr);

};

#endif // ABOUT\_H

***About.cpp***

#include "about.h"

#include <QLabel>

#include <QGridLayout>

About::About(QWidget\* parent): QDialog(parent)

{

QGridLayout\* gridLayout = new QGridLayout(this);

setLayout(gridLayout);

gridLayout->addWidget(new QLabel("Программу сделал студент группы ПО-5, Пищик Алексей \nДанная программа выводит информацию о компьютере", this));

}

**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()) {

string json\_server;

ifstream f1;

f1.open("C:\\Users\\Toketsu\\Desktop\\test server\\server\\server\\debug\\server.json");

getline(f1, json\_server);

f1.close();

QString data = QString::fromStdString(json\_server);

if(QString(Data) == data) {

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

socket->write("Actuale");

}

else if(Data[0] == 'M') {

QStringList client\_data;

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

QByteArray wr;

QDir dir\_server(QDir::currentPath());

QStringList find\_filter;

dir\_server.cd("plugins");

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)

{

wr.append(str);

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

wr.append(" ");

}

else {

wr.append(" ");

wr.append(data);

}

}

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

socket->write(wr);

}

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>

#include <fstream>

#include <iostream>

using namespace std;

class MyServer: public QTcpServer{

Q\_OBJECT

public:

MyServer();

~MyServer();

QTcpSocket \* socket;

QByteArray Data;

QList<qintptr> sessions\_pending;

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