Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Факультет электронно-информационных систем

Кафедра ИИТ

Лабораторная работа №4

за 5 семестр

По дисциплине: «ОСиСП»

Выполнила:

студентка 3 курса

группы ПО-4(1)

Грицак A.В.

Проверила:

Дряпко А.В.

2021

Лабораторная работа №4

Цель работы: ознакомиться с возможностями, предлагаемыми Qt для поддержки сетевого взаимодействия программ.

Вариант 7

Задание:

1) Разработать сетевую утилиту для автоматического обновления приложения, разработанного в лабораторных работах 1-3. Утилита может иметь произвольный интерфейс, определяемый ее функциональными особенностями.

2) Программа должна состоять из двух взаимодействующих частей – клиентской, устанавливаемой на компьютере с обновляемым приложением и серверной, выполняющейся на любом компьютере в локальной либо глобальной сети.

3) Клиентская часть осуществляет соединение с сервером и проверку обновлений для приложения. При наличии обновлений, все необходимые файлы загружаются и копируются в директорию с целевым приложением. В противном случае выдается соответствующее сообщение. Обработать возможные исключительные ситуации (отсутствие соединения с сервером).

4) Внести изменения в исходный проект приложения с учетом специфики загружаемых обновлений (например, хранение структуры уровня для игрового приложения в отдельном файле). То есть обновляемые ресурсы должны быть отделены от основного приложения.

5) Обновляемые компоненты по вариантам (DLL из лабораторной работы No3):

7. DLL, конфигурационный файл (количество мин, размер поля)

**Текст программы:**

1. **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

1. **myserver.cpp**

#include "myserver.h"

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

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

void MyServer::**StartServer**(){

if(this->listen(QHostAddress::Any, 1919)) {

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 == "Yes") {

Data.clear();

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

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

}

1. **main.cpp**

#include <QCoreApplication>

#include <myserver.h>

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

{

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

MyServer Server;

Server.StartServer();

return a.exec();

}

1. **server.pro**

QT += network core

CONFIG += c++11 console

CONFIG -= app\_bundle

DEFINES += QT\_DEPRECATED\_WARNINGS

SOURCES += \

main.cpp \

myserver.cpp

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

HEADERS += \

myserver.h

1. **levels.h**

#ifndef LEVELS\_H

#define LEVELS\_H

#include "levels\_global.h"

#include <QString>

extern "C" LEVELS\_EXPORT QString **countLevels**();

extern "C" LEVELS\_EXPORT QString **level1**();

extern "C" LEVELS\_EXPORT QString **level2**();

#endif // LEVEL\_H

1. **levels\_global.h**

#ifndef LEVELS\_GLOBAL\_H

#define LEVELS\_GLOBAL\_H

#include <QtCore/qglobal.h>

#if defined(LEVELS\_LIBRARY)

# define LEVELS\_EXPORT Q\_DECL\_EXPORT

#else

# define LEVELS\_EXPORT Q\_DECL\_IMPORT

#endif

#endif // LEVELS\_GLOBAL\_H

1. **levels.cpp**

#include "levels.h"

QString **countLevels**() {

return QString("2");

}

QString **level1**() {

return QString("10 8");

}

QString **level2**() {

return QString("30 11");

}

1. **levels.pro**

QT += core gui

QT += widgets

TARGET = Levels

TEMPLATE = lib

DEFINES += LEVELS\_LIBRARY

CONFIG += c++11

DEFINES += QT\_DEPRECATED\_WARNINGS=

SOURCES += \

levels.cpp

HEADERS += \

levels.h \

levels\_global.h

unix {

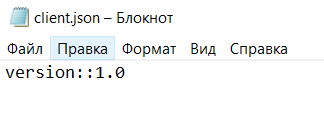
target.path = /usr/lib

}

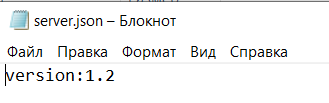
!isEmpty(target.path): INSTALLS += target

**Результаты тестирования программы:**

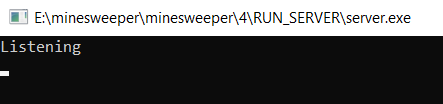
В client.json хранится текущая версия приложения:



В server.json хранится последняя версия, до которой можно обновить приложение:

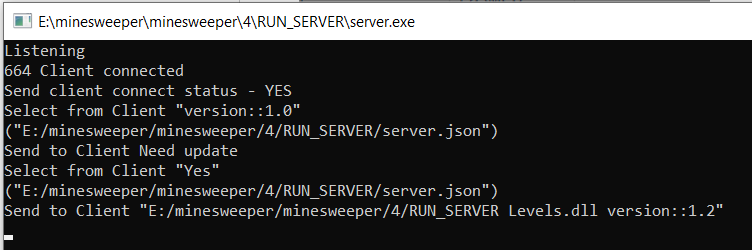
****

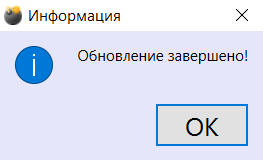
Запуск сервера:

****

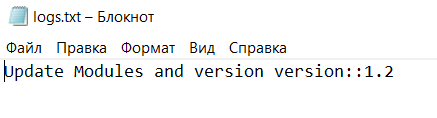
Подключение к серверу и обновление приложения до последней версии:

****

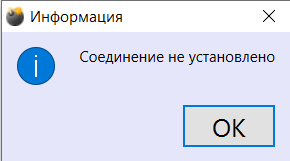
****

****

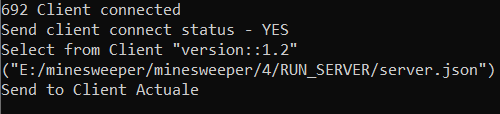
Логирование обновления:

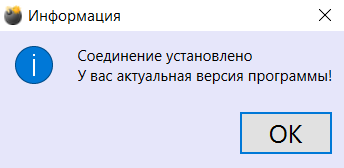


В случае отсутствия подключения к серверу:



Если приложение уже имеет последнюю версию:





**Вывод:** Ознакомилась с возможностями, предлагаемыми Qt для поддержки сетевого взаимодействия программ.