МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №4

# По дисциплине:

# « Операционные Системы и Системное Программирование»

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Прокопчик Е.А.

Проверила:

Дряпко А.В.

Брест 2021

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

Вариант 18

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):

DLL, конфигурационный файл (внешний вид, размер поля)

Код:

**Sudoku.cpp**

…

void Sudoku::**update**(bool isAutoUpdate)

{

this->isAutoUpdate = isAutoUpdate;

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

socket->write(currentVersion.toStdString().c\_str());

this->isAutoUpdate = isAutoUpdate;

}

void Sudoku::**sockDisc**()

{

socket->deleteLater();

}

void Sudoku::**sockReady**()

{

QFuture<void> future = QtConcurrent::run([=]()

{

QByteArray data = socket->readAll();

if (data == "Up-to-date" && !isAutoUpdate)

{

messageBoxText = "Already up-to-date";

return;

}

if (data == "Up-to-date")

{

return;

}

QStringList changes = QString::fromLatin1(data).split("\n\n\n");

for (int i = 1; i < changes.size(); i++)

{

QFile file(qApp->applicationDirPath() + changes[i]);

file.*open*(QIODevice::WriteOnly | QIODevice::Truncate);

file.write(changes[++i].toLatin1());

file.*close*();

}

currentVersion = changes[0];

writeSettingsToFile();

messageBoxText = "Restart application";

});

future.waitForFinished();

if (messageBoxText.length() != 0)

{

QMessageBox::information(this, "Update", messageBoxText);

}

}

void Sudoku::**setAutoUpdate**()

{

autoUpdateSetting = !autoUpdateSetting;

actionAutoUpdate->setText(getAutoUpdateMessage());

writeSettingsToFile();

}

void Sudoku::**writeSettingsToFile**()

{

QFile settingsFile(qApp->applicationDirPath() + "/settings.json");

settingsFile.*open*(QIODevice::WriteOnly | QIODevice::Text);

QJsonObject version;

version.insert("version", currentVersion);

version.insert("autoUpdate", autoUpdateSetting);

QJsonDocument settings(version);

settingsFile.write(settings.toJson());

settingsFile.*close*();

}

QString Sudoku::**getAutoUpdateMessage**()

{

return autoUpdateSetting ? "Turn off auto update" : "Turn on auto update";

}

…

**Server**

**Myserver.h**

#ifndef MYSERVER\_H

#define MYSERVER\_H

#include <QTcpServer>

#include <QtCore>

class **myserver**: public QTcpServer

{

Q\_OBJECT

public:

**myserver**();

~***myserver***();

QTcpServer \*tcpSever;

QTcpSocket\* socket;

public slots:

void **startServer**();

void ***incomingConnection***(long long socketDescriptor);

void **sockReady**();

void **sockDisc**();

};

#endif // MYSERVER\_H

**Myserver.cpp**

#include "myserver.h"

#include <QTcpSocket>

#include <QFile>

#include <QDir>

myserver::**myserver**() {}

myserver::~***myserver***() {}

void myserver::**startServer**()

{

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

{

qDebug() << "Listening";

}

else

{

qDebug() << "Not listening";

}

}

void myserver::***incomingConnection***(long long socketDescriptor)

{

socket = new QTcpSocket(this);

socket->*setSocketDescriptor*(socketDescriptor);

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

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

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

}

void myserver::**sockReady**()

{

QByteArray data = socket->readAll();

qDebug() << "Received client version: " << data;

QFile updatesFile(qApp->applicationDirPath() + "/updates.json");

updatesFile.*open*(QIODevice::ReadOnly | QIODevice::Text);

QByteArray updatesFileData = updatesFile.readAll();

updatesFile.*close*();

QJsonDocument updatesDoc = QJsonDocument::fromJson(updatesFileData);

QJsonObject rootObj = updatesDoc.object();

QString currentVersion = rootObj.value("currentVersion").toString();

if (data == currentVersion)

{

socket->write("Up-to-date");

return;

}

QJsonArray updates = rootObj.value("updates").toArray();

QByteArray dataToSend;

foreach (const QJsonValue& update, updates)

{

if (update.toObject().value("fromVersion").toString() == data)

{

dataToSend.append(currentVersion.toStdString().c\_str());

QJsonArray changes = update.toObject().value("changes").toArray();

foreach (const QJsonValue& pathToChangedFile, changes)

{

dataToSend.append("\n\n\n");

dataToSend.append(pathToChangedFile.toString().toStdString().c\_str());

dataToSend.append("\n\n\n");

QFile fileToSend(qApp->applicationDirPath() + pathToChangedFile.toString());

fileToSend.*open*(QIODevice::ReadOnly);

QByteArray fileContents = fileToSend.readAll();

dataToSend.append(fileContents);

}

qDebug() << dataToSend.size();

socket->write(dataToSend);

return;

}

}

}

void myserver::**sockDisc**()

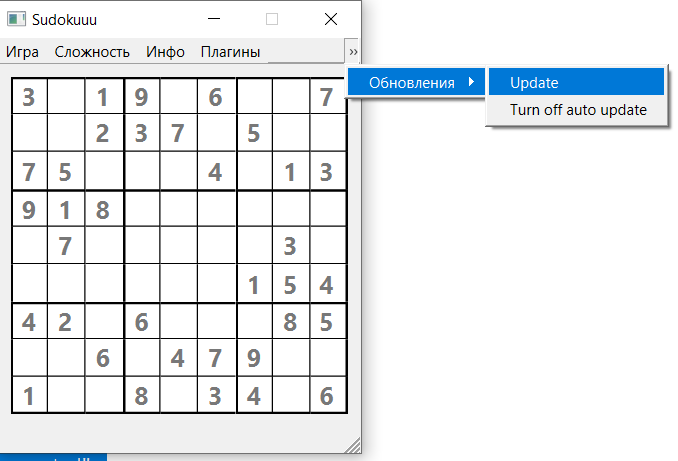
{

qDebug() << "Disconnect";

socket->deleteLater();

}

Результат:



Вывод: ознакомился с возможностями, предлагаемыми Qt для поддержки сетевого взаимодействия программ