//main.cpp

#include <view/r173.h>

#include <view/mainmenu.h>

#include <QApplication>

#include <controller/broadcastnetworkcontroller.h>

#include "view/r123.h"

#include <QThread>

#include <QDebug>

#include <thread>

#include <chrono>

#include "instruments/webCounter.h"

#include <math.h>

using namespace std;

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

{

Socket::init();

QApplication a(argc, argv);

MainMenu w;

w.show();

int ret\_val = a.exec();

QAudioFormat format;

char msg[SEMAPHORE\_MSG\_LEN];

strcpy(msg, "delete");

BroadcastSocket killSignal1, killSignal2;

killSignal1.bind(nullptr, 5001);

killSignal2.bind(nullptr, 5002);

killSignal1.send(msg, SEMAPHORE\_MSG\_LEN);

killSignal2.send(msg, SEMAPHORE\_MSG\_LEN);

Socket::clear();

return ret\_val;

}

//mainmenu.h

#ifndef MAINMENU\_H

#define MAINMENU\_H

#include <QMainWindow>

namespace **Ui** {

class **MainMenu**;

}

class **MainMenu** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainMenu**(QWidget \*parent = nullptr);

~***MainMenu***();

private slots:

void **on\_launchServer\_clicked**();

void **on\_pushButton\_clicked**();

void **on\_createServer\_clicked**();

void **on\_ipEdit\_returnPressed**();

private:

Ui::MainMenu \*ui;

};

//mainmenu.cpp

#endif // MAINMENU\_H

#include "mainmenu.h"

#include "ui\_mainmenu.h"

#include "view/r173.h"

#include "view/r123.h"

#include <regex>

#include <string>

#include <QMessageBox>

#include "controller/broadcastnetworkcontroller.h"

using namespace std;

MainMenu::MainMenu(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainMenu)

{

ui->setupUi(this);

}

MainMenu::~MainMenu()

{

delete ui;

}

void MainMenu::on\_launchServer\_clicked()

{

regex regular("^[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}$");

if (!regex\_match(this->ui->ipEdit->text().toStdString().c\_str(), regular))

{

QMessageBox::warning(this, "Ошибка ввода", "Вы ввели неправильный ip адресс");

}

else

{

#ifndef BROAD\_CAST

NetworkController \* networkController = new NetworkController(false, ui->ipEdit->text());

#else

BroadcastNetworkController \* networkController = new BroadcastNetworkController(true, "");

#endif

networkController->start();

R173 \*w = new R173(this->ui->ipEdit->text(), false, networkController);

w->show();

this->close();

}

}

void MainMenu::on\_pushButton\_clicked()

{

QMessageBox::information(this, "Авторы", "Бейда А.И.\nБурак А.А.\nДудак М.Н.\nМакатерчик А.В.\nУтин Л.Л.");

}

void MainMenu::on\_createServer\_clicked()

{

#ifndef BROAD\_CAST

NetworkController \* networkController = new NetworkController(true, "");

#else

BroadcastNetworkController \* networkController = new BroadcastNetworkController(true, "");

#endif

networkController->start();

R173 \*w = new R173(this->ui->ipEdit->text(), false, networkController);

w->show();

this->close();

}

void MainMenu::on\_ipEdit\_returnPressed()

{

on\_launchServer\_clicked();

}

//r173.h

#ifndef R173\_H

#define R173\_H

#include <QMainWindow>

#include <controller/R173Controller.h>

#include "controller/statecontroller.h"

#include "controller/broadcastnetworkcontroller.h"

#include <QMouseEvent>

namespace Ui {

class R173;

}

class R173 : public QMainWindow

{

Q\_OBJECT

friend class R173Controller;

R173Controller controller;

bool pomiechi\_reversed = false;

bool szum\_reversed = false;

bool power\_reversed = false;

bool handle\_key\_events = true;

virtual void keyPressEvent(QKeyEvent \*ev) override;

virtual void keyReleaseEvent(QKeyEvent \*ev) override;

void setEnableToAll(bool anable);

void mousePressEvent(QMouseEvent \* event) override;

void closeEvent(QCloseEvent \* event) override;

void antenna\_is\_pressed();

void zeziemlinije\_is\_pressed();

void telefon\_is\_pressed();

void pitanije\_is\_pressed();

public:

#ifndef BROAD\_CAST

explicit R173(QString IP, bool is\_server, NetworkController \* controller);

#else

explicit R173(QString IP, bool is\_server, BroadcastNetworkController \* controller);

#endif

~R173();

void set\_table(StateController::TableInfo info);

void call\_on();

void call\_off();

void prd\_on();

void prd\_off();

void wheelEvent(QWheelEvent \* event) override;

public slots:

void perest();

private slots:

void on\_button0\_clicked();

void on\_button1\_clicked();

void on\_button2\_clicked();

void on\_button3\_clicked();

void on\_button4\_clicked();

void on\_button5\_clicked();

void on\_button6\_clicked();

void on\_button7\_clicked();

void on\_button8\_clicked();

void on\_button9\_clicked();

void on\_buttonCls\_clicked();

void on\_pomiechi\_clicked();

void on\_powerLevel\_clicked();

void on\_szum\_clicked();

void on\_power\_clicked();

void on\_tableSwitch\_clicked();

void on\_ton\_pressed();

void on\_ton\_released();

private:

Ui::R173 \*ui;

};

#endif // R173\_H

//r173.cpp

#include "r173.h"

#include "ui\_r173.h"

#include <controller/statecontroller.h>

#include <QDebug>

#include <ctime>

#include <math.h>

#include <QPainter>

void R173::***keyPressEvent***(QKeyEvent \*ev)

{

if (!ev->isAutoRepeat() && this->handle\_key\_events)

this->controller.KeyPressHandler(ev);

}

void R173::***keyReleaseEvent***(QKeyEvent \*ev)

{

if (!ev->isAutoRepeat() && this->handle\_key\_events)

this->controller.KeyReleaseHandler(ev);

}

void R173::**setEnableToAll**(bool anable)

{

this->ui->ton->setEnabled(anable);

this->ui->szum->setEnabled(anable);

this->ui->power->setEnabled(anable);

this->ui->button0->setEnabled(anable);

this->ui->button1->setEnabled(anable);

this->ui->button2->setEnabled(anable);

this->ui->button3->setEnabled(anable);

this->ui->button4->setEnabled(anable);

this->ui->button5->setEnabled(anable);

this->ui->button6->setEnabled(anable);

this->ui->button7->setEnabled(anable);

this->ui->button8->setEnabled(anable);

this->ui->button9->setEnabled(anable);

this->ui->buttonCls->setEnabled(anable);

this->ui->pomiechi->setEnabled(anable);

this->ui->powerLevel->setEnabled(anable);

this->ui->tableSwitch->setEnabled(anable);

handle\_key\_events = anable;

}

void R173::***mousePressEvent***(QMouseEvent \*event)

{

static int pos\_x\_zem = 1000;

static int pos\_x\_ant = 881;

if (event->x() > 0 && event->x() < 201 &&

event->y() > 446 && event->y() < 600) // x>140, y < 537

{

this->telefon\_is\_pressed();

}

else if (event->x() > 0 && event->x() < 201 &&

event->y() > 290 && event->y() < 450) // x > 140, y < 361

{

this->pitanije\_is\_pressed();

}

else if (event->x() > 850 && event->x() < pos\_x\_ant &&

event->y() > 560 && event->y() < 621) // x < 881

{

if (pos\_x\_ant == 881)

pos\_x\_ant = 1000;

else

pos\_x\_ant = 881;

this->antenna\_is\_pressed();

}

else if (event->x() > 910 && event->x() < pos\_x\_zem &&

event->y() > 570 && event->y() < 601) // x < 961

{

this->zeziemlinije\_is\_pressed();

}

}

void R173::***closeEvent***(QCloseEvent \*event)

{

//this->controller.destroy();

event->accept();

}

void R173::**antenna\_is\_pressed**()

{

qDebug() << "antena";

static bool antena\_launched = false;

static QPixmap on(":/res/p173-lno.png"), off;

if (antena\_launched)

{

this->ui->provod4\_show->setPixmap(off);

antena\_launched = !antena\_launched;

}

else

{

this->ui->provod4\_show->setPixmap(on);

antena\_launched = !antena\_launched;

}

controller.ButtonPressed(controller.BANT);

}

void R173::**zeziemlinije\_is\_pressed**()

{

qDebug() << "zaziemlienije";

static bool zazemleno = false;

static QPixmap on(":/res/p173-lvo.png"), off;

if (zazemleno)

{

this->ui->provod3\_show->setPixmap(off);

this->setEnableToAll(false);

zazemleno = !zazemleno;

}

else

{

this->setEnableToAll(true);

this->ui->provod3\_show->setPixmap(on);

zazemleno = !zazemleno;

}

}

void R173::**telefon\_is\_pressed**()

{

qDebug() << "telefon";

static bool telefon\_launched = false;

static QPixmap on(":/res/p173-nlo.png"), off;

if (telefon\_launched)

{

this->ui->provod1\_show->setPixmap(off);

telefon\_launched = !telefon\_launched;

}

else

{

this->ui->provod1\_show->setPixmap(on);

telefon\_launched = !telefon\_launched;

}

controller.ButtonPressed(controller.BTEL);

}

void R173::**pitanije\_is\_pressed**()

{

qDebug() << "pitanije";

static bool zapitano = false;

static QPixmap on(":/res/p173-nvo.png"), off;

if (zapitano)

{

this->ui->provod2\_show->setPixmap(off);

zapitano = !zapitano;

}

else

{

this->ui->provod2\_show->setPixmap(on);

zapitano = !zapitano;

}

controller.ButtonPressed(controller.BPIT);

}

#ifndef BROAD\_CAST

R173::**R173**(QString IP, bool is\_server, NetworkController \* networkController) :

#else

**R173**::**R173**(**QString** **IP**, bool **is\_server**, **BroadcastNetworkController** \* **networkController**) :

#endif

QMainWindow(nullptr),

ui(new Ui::R173),

controller(this, IP, is\_server, networkController)

{

setFocusPolicy(Qt::FocusPolicy::StrongFocus);

this->setFixedSize(992, 671);

ui->setupUi(this);

this->setEnableToAll(false);

controller.changeVolume((24) \* 100 / 124);

}

R173::~***R173***()

{

//Socket::clear();

delete ui;

}

void R173::**set\_table**(StateController::TableInfo table)

{

if (table.field[0] == -1)

ui->field0->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field0->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[0]) + ".jpg);");

if (table.field[1] == -1)

ui->field1->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field1->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[1]) + ".jpg);");

if (table.field[2] == -1)

ui->field2->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field2->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[2]) + ".jpg);");

if (table.field[3] == -1)

ui->field3->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field3->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[3]) + ".jpg);");

if (table.field[4] == -1)

ui->field4->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field4->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[4]) + ".jpg);");

if (table.field[5] == -1)

ui->field5->setStyleSheet("background-image: url(:/res/tableNubers/field\_off.jpg);");

else

ui->field5->setStyleSheet("background-image: url(:/res/tableNubers/field" +

QString::number(table.field[5]) + ".jpg);");

}

void R173::**prd\_on**()

{

this->ui->prd->setStyleSheet("background-image: url(:/res/pered\_o.jpg);");

}

void R173::**prd\_off**()

{

this->ui->prd->setStyleSheet("background-image: url(:/res/pered.jpg);");

}

void R173::***wheelEvent***(QWheelEvent \*event)

{

static QPixmap \* volumePrdPxmap = nullptr;

static QPixmap \* volumeSoundPxmap = nullptr;

static int anglePrd = 0;

static int angleSnd = 0;

if (event->pos().rx() > 433 && event->pos().rx() < 464

&& event->pos().ry() > 377 && event->pos().ry() < 408)

{

if (!volumePrdPxmap)

volumePrdPxmap = new QPixmap(":/res/p-173m-round.png");

QPixmap pixmap(\*volumePrdPxmap);

QMatrix rm;

if (event->angleDelta().ry() > 0 && anglePrd < 32)

anglePrd += 8;

else if (event->angleDelta().ry() < 0 && anglePrd > -92)

anglePrd -= 8;

rm.rotate(anglePrd);

pixmap = pixmap.transformed(rm);

ui->volumePRM->setPixmap(QPixmap(pixmap.transformed(rm)));

}

if (event->pos().rx() > 406 && event->pos().rx() < 437

&& event->pos().ry() > 464 && event->pos().ry() < 495)

{

if (!volumeSoundPxmap)

volumeSoundPxmap = new QPixmap(":/res/p-173m-round-sound.png");

QPixmap pixmap(\*volumeSoundPxmap);

QMatrix rm;

if (event->angleDelta().ry() > 0 && angleSnd < 100)

angleSnd += 8;

else if (event->angleDelta().ry() < 0 && angleSnd > -24)

angleSnd -= 8;

rm.rotate(angleSnd);

pixmap = pixmap.transformed(rm);

ui->volumeSound->setPixmap(QPixmap(pixmap.transformed(rm)));

controller.changeVolume((angleSnd + 24) \* 100 / 124);

}

}

void R173::**perest**()

{

static bool activate = false;

this->setEnableToAll(activate);

if (!activate)

{

prd\_on();

}

else

{

prd\_off();

}

activate = !activate;

}

void R173::**call\_on**()

{

this->ui->call->setStyleSheet("background-image: url(:/res/p-173m-call\_on.jpg);");

}

void R173::**call\_off**()

{

this->ui->call->setStyleSheet("background-image: url(:/res/p-173m-call\_off.jpg);");

}

void R173::**on\_button0\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B0);

}

void R173::**on\_button1\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B1);

}

void R173::**on\_button2\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B2);

}

void R173::**on\_button3\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B3);

}

void R173::**on\_button4\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B4);

}

void R173::**on\_button5\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B5);

}

void R173::**on\_button6\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B6);

}

void R173::**on\_button7\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B7);

}

void R173::**on\_button8\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B8);

}

void R173::**on\_button9\_clicked**()

{

this->controller.ButtonPressed(R173Controller::B9);

}

void R173::**on\_buttonCls\_clicked**()

{

this->controller.ButtonPressed(R173Controller::BCLS);

}

void R173::**on\_pomiechi\_clicked**()

{

static bool switchPomiechi = false;

if (!switchPomiechi)

{

this->ui->pomiechi->setStyleSheet("border-image: url(:/res/pomiechi\_r.jpg);");

}

else

{

this->ui->pomiechi->setStyleSheet("border-image: url(:/res/pomiechi.jpg);");

}

switchPomiechi = !switchPomiechi;

this->controller.ButtonPressed(R173Controller::BPOMIECHI);

}

void R173::**on\_powerLevel\_clicked**()

{

static bool switchPowerLevel = false;

if (!switchPowerLevel)

{

this->ui->powerLevel->setStyleSheet("border-image: url(:/res/powerLavel\_r.jpg);");

}

else

{

this->ui->powerLevel->setStyleSheet("border-image: url(:/res/powerLavel.jpg);");

}

switchPowerLevel = !switchPowerLevel;

this->controller.ButtonPressed(R173Controller::BPOWERLEVEL);

}

void R173::**on\_szum\_clicked**()

{

static bool switchSzum = false;

if (!switchSzum)

{

this->ui->szum->setStyleSheet("border-image: url(:/res/szum\_r.jpg);");

}

else

{

this->ui->szum->setStyleSheet("border-image: url(:/new/prefix1/res/szum.jpg);");

}

switchSzum = !switchSzum;

this->controller.ButtonPressed(R173Controller::BSZUM);

}

void R173::**on\_power\_clicked**()

{

static bool switchPower = false;

if (!switchPower)

{

this->ui->power->setStyleSheet("border-image: url(:/res/power\_r.jpg);");

}

else

{

this->ui->power->setStyleSheet("border-image: url(:/new/prefix1/res/power.jpg);");

}

switchPower = !switchPower;

this->controller.ButtonPressed(R173Controller::BPOWER);

}

void R173::**on\_tableSwitch\_clicked**()

{

static bool switchTable = false;

if (!switchTable)

{

this->ui->tableSwitch->setStyleSheet("border-image: url(:/res/table\_r.jpg);");

}

else

{

this->ui->tableSwitch->setStyleSheet("border-image: url(:/res/table.jpg);");

}

switchTable = !switchTable;

this->controller.ButtonPressed(R173Controller::BTABLE);

}

void R173::**on\_ton\_pressed**()

{

this->ui->ton->setStyleSheet("border-image: url(:/res/p-173m-ton\_pressed.jpg);");

this->controller.ButtonPressed(R173Controller::BTON);

}

void R173::**on\_ton\_released**()

{

this->ui->ton->setStyleSheet("border-image: url(:/res/p-173m-ton\_released.jpg);");

this->controller.buttonReleased(R173Controller::BTON);

}

//R173Controller.h

#pragma once

//#define BROAD\_CAST

#include <QKeyEvent>

#include "controller/statecontroller.h"

#include "controller/networkcontroller.h"

#include "controller/broadcastnetworkcontroller.h"

class **R173**;

class **R173Controller** : public QObject {

Q\_OBJECT

//SoundController soundController;

StateController stateController;

#ifdef BROAD\_CAST

**BroadcastNetworkController**& **networkController**;

#else

NetworkController& networkController;

#endif

void **setNetworkState**(bool activate);

bool is\_config;

R173 \* parent;

class **Perest** : public QThread

{

R173Controller \* parent;

void ***run***() override;

public:

**Perest**(R173Controller \* parent) : parent(parent) {}

} perest;

public:

#ifndef BROAD\_CAST

**R173Controller**(R173 \* parent, QString IP, bool is\_server, NetworkController \* networkController);

#else

**R173Controller**(**R173** \* **parent**, **QString** **IP**, bool **is\_server**, **BroadcastNetworkController** \* **networkController**);

#endif

enum **Button** {

B1, B2, B3, B4, B5, B6, B7, B8, B9, B0, BTON,

BCLS, BPOMIECHI, BSZUM, BPOWER, BPOWERLEVEL, BTABLE,

BPIT, BANT, BTEL

};

void **KeyPressHandler**(QKeyEvent \* ev);

void **ButtonPressed**(Button b);

void **KeyReleaseHandler**(QKeyEvent \* ev);

void **buttonReleased**(Button b);

void **changeVolume**(char level);

void **destroy**();

public slots:

void **set\_ton**();

void **reset\_ton**();

signals:

void **prd\_on**();

void **prd\_off**();

};

//R173Controller.cpp

#include <controller/R173Controller.h>

#include <QDebug>

#include <Qt>

#include "view/r173.h"

#include <QThread>

#ifndef BROAD\_CAST

R173Controller::R173Controller(R173 \*parent, QString IP, bool is\_server, NetworkController \* networkController) :

#else

R173Controller::R173Controller(R173 \*parent, QString IP, bool is\_server, BroadcastNetworkController \* networkController) :

#endif

networkController(\*networkController), perest(this)

{

this->parent = parent;

this->is\_config = false;

QObject::connect(networkController, SIGNAL(set\_call()), this, SLOT(set\_ton()), Qt::QueuedConnection);

QObject::connect(networkController, SIGNAL(reset\_call()), this, SLOT(reset\_ton()), Qt::QueuedConnection);

QObject::connect(this, SIGNAL(prd\_on()), this->parent, SLOT(perest()), Qt::QueuedConnection);

QObject::connect(this, SIGNAL(prd\_off()), this->parent, SLOT(perest()), Qt::QueuedConnection);

}

void R173Controller::KeyPressHandler(QKeyEvent \*ev)

{

if (ev->key() == Qt::Key\_Space)

setNetworkState(true);

}

void R173Controller::ButtonPressed(R173Controller::Button b)

{

int state = 0;

switch(b){

case B0:

state = this->stateController.numberButtonPressed(0);

break;

case B1:

state = this->stateController.numberButtonPressed(1);

break;

case B2:

state = this->stateController.numberButtonPressed(2);

break;

case B3:

state = this->stateController.numberButtonPressed(3);

break;

case B4:

state = this->stateController.numberButtonPressed(4);

break;

case B5:

state = this->stateController.numberButtonPressed(5);

break;

case B6:

state = this->stateController.numberButtonPressed(6);

break;

case B7:

state = this->stateController.numberButtonPressed(7);

break;

case B8:

state = this->stateController.numberButtonPressed(8);

break;

case B9:

state = this->stateController.numberButtonPressed(9);

break;

case BCLS:

this->stateController.clsPresed();

break;

case BPOMIECHI:

this->networkController.setSzumLevel(

this->stateController.pomiechiIsPressed()

);

break;

case BPOWER:

this->stateController.powerIsPressed();

break;

case BSZUM:

this->networkController.setSzumLevel(

this->stateController.szumIsPressed()

);

break;

case BPOWERLEVEL:

this->networkController.setRecordVolume(

this->stateController.powerLevelIsPressed()

);

break;

case BTABLE:

this->stateController.tableSwitcher();

break;

case BTON:

this->networkController.call\_on();

break;

case BTEL:

this->stateController.telSwitcher();

break;

case BANT:

this->stateController.antSwitcher();

break;

case BPIT:

this->stateController.pitSwitcher();

break;

}

if (state)

this->perest.start();

if (b != BTON)

setNetworkState(false);

this->parent->set\_table(this->stateController.getTable());

}

void R173Controller::KeyReleaseHandler(QKeyEvent \*ev)

{

if (ev->key() == Qt::Key\_Space)

this->setNetworkState(false);

}

void R173Controller::buttonReleased(R173Controller::Button b)

{

switch (b) {

case BTON:

this->networkController.call\_off();

break;

}

}

void R173Controller::changeVolume(char level)

{

networkController.setVolume(level);

}

void R173Controller::destroy()

{

delete &this->networkController;

}

void R173Controller::set\_ton()

{

this->parent->call\_on();

}

void R173Controller::reset\_ton()

{

this->parent->call\_off();

}

void R173Controller::setNetworkState(bool activate)

{

static bool config = false;

if (activate == true)

{

if (config == false && stateController.isReadyToRecieve())

{

parent->prd\_on();

config = true;

this->networkController.config\_send(stateController.getCurrentFrequency());

}

else

{

parent->prd\_off();

config = false;

if (stateController.isReadyToRecieve())

this->networkController.config\_listen(stateController.getCurrentFrequency());

else

this->networkController.config\_kill();

}

}

else

{

parent->prd\_off();

if (this->stateController.isReadyToRecieve())

{

this->networkController.config\_listen(this->stateController.getCurrentFrequency());

}

else

{

this->networkController.config\_kill();

}

config = false;

}

}

void R173Controller::Perest::run()

{

emit parent->prd\_on();

this->msleep(3000);

emit parent->prd\_off();

}

//statecontroller.h

#ifndef STATECONTROLLER\_H

#define STATECONTROLLER\_H

#include <QAudioInput>

#include <QAudioOutput>

class **StateController**

{

int frequencies[10];

char chosen\_frequency;

QAudioInput \* audioInput;

QAudioOutput \* audioOutput;

bool is\_in\_config;

bool is\_setting\_up;

bool is\_turned\_on;

char step\_of\_setting\_up;

int is\_config\_on;

bool ant\_launched;

bool pit\_launched;

bool tel\_launched;

bool szum;

bool pomiechi;

public:

struct **TableInfo** {

char field[6];

static TableInfo **frequencyToTable**(int frequency);

static void **frequencyToTable**(TableInfo& info, int frequency);

};

**StateController**();

TableInfo **getTable**();

bool **isReadyToRecieve**();

int **numberButtonPressed**(char number);

void **clsPresed**();

void **powerIsPressed**();

char **pomiechiIsPressed**();

char **powerLevelIsPressed**();

char **szumIsPressed**();

void **tableSwitcher**();

void **pitSwitcher**();

void **antSwitcher**();

void **telSwitcher**();

int **getCurrentFrequency**();

};

#endif // STATECONTROLLER\_H

//statecontroller.cpp

#include "statecontroller.h"

#include <cmath>

#include <QDebug>

StateController::**StateController**()

{

for (int i = 0; i < 10; i++)

this->frequencies[i] = 30000;

this->chosen\_frequency = 1;

this->is\_turned\_on = false;

this->is\_in\_config = false;

this->is\_setting\_up = false;

this->is\_config\_on = -1;

this->ant\_launched = false;

this->tel\_launched = false;

this->pit\_launched = false;

this->pomiechi = false;

this->szum = false;

}

StateController::TableInfo StateController::**getTable**()

{

TableInfo info;

if(!this->is\_turned\_on || !this->pit\_launched)

{

for (int i = 0; i < 6; i++) {

info.field[i] = -1;

}

}

else if (!this->is\_in\_config)

{

for (int i = 1; i < 6; i++)

info.field[i] = -1;

info.field[0] = this->chosen\_frequency;

}

else if (!this->is\_setting\_up)

{

info.field[0] = this->chosen\_frequency;

TableInfo::frequencyToTable(*info*, this->frequencies[this->chosen\_frequency]);

}

else

{

info.field[0] = this->chosen\_frequency;

TableInfo::frequencyToTable(*info*, this->frequencies[chosen\_frequency]);

for(int i = 0; i < 5 - step\_of\_setting\_up; i++)

{

info.field[5-i] = -1;

}

}

return info;

}

bool StateController::**isReadyToRecieve**()

{

if (this->is\_config\_on != -1 && this->is\_turned\_on &&

ant\_launched && tel\_launched)

return true;

else

return false;

}

int StateController::**numberButtonPressed**(char number)

{

qDebug() << "pressed " << ((int)number);

int to\_ret = 0;

if (this->is\_turned\_on && this->pit\_launched)

{

if (!this->is\_setting\_up)

{

if(!this->is\_in\_config)

{

this->is\_config\_on = this->frequencies[number];

to\_ret = 1;

}

this->chosen\_frequency = number;

}

else

{

switch(step\_of\_setting\_up)

{

case 0:

if (number < 3 || number > 7)

return 0;

break;

case 1:

if (frequencies[chosen\_frequency] == 70000 && number > 5)

return 0;

break;

}

if(step\_of\_setting\_up == 0)

frequencies[chosen\_frequency] = 0;

frequencies[chosen\_frequency] += number \* pow(10, 4 - step\_of\_setting\_up);

step\_of\_setting\_up++;

if (step\_of\_setting\_up == 5)

{

this->is\_setting\_up = false;

}

}

}

return to\_ret;

}

void StateController::**clsPresed**()

{

qDebug() << "pressed cls";

if (this->is\_turned\_on && this->is\_in\_config)// && !this->is\_setting\_up)

{

this->is\_setting\_up = true;

this->step\_of\_setting\_up = 0;

}

}

void StateController::**powerIsPressed**()

{

this->is\_turned\_on = !this->is\_turned\_on;

this->is\_setting\_up = false;

qDebug() << "pressed power";

}

char StateController::**pomiechiIsPressed**()

{

qDebug() << "pressed pomiechi";

this->pomiechi = !this->pomiechi;

if (pomiechi && szum)

return 0;

else if (pomiechi || szum)

return 1;

else

return 2;

}

char StateController::**powerLevelIsPressed**()

{

static bool little = false;

qDebug() << "level is pressed " << !little;

if (little == false)

{

little = true;

return 50;

}

else

{

little = false;

return 100;

}

}

char StateController::**szumIsPressed**()

{

qDebug() << "szum is pressed";

this->szum = !this->szum;

if (pomiechi && szum)

return 0;

else if (pomiechi || szum)

return 1;

else

return 2;

}

void StateController::**tableSwitcher**()

{

qDebug() << "table switcher is pressed";

this->is\_in\_config = !this->is\_in\_config;

this->is\_setting\_up = false;

}

void StateController::**pitSwitcher**()

{

pit\_launched = !pit\_launched;

}

void StateController::**antSwitcher**()

{

ant\_launched = !ant\_launched;

}

void StateController::**telSwitcher**()

{

tel\_launched = !tel\_launched;

}

int StateController::**getCurrentFrequency**()

{

return this->is\_config\_on;

}

StateController::TableInfo StateController::TableInfo::**frequencyToTable**(int frequency)

{

TableInfo info;

TableInfo::frequencyToTable(*info*, frequency);

return info;

}

void StateController::TableInfo::**frequencyToTable**(StateController::TableInfo &info, int frequency)

{

for (int i = 0; i < 5; i++)

{

info.field[5-i] = frequency % 10;

frequency/=10;

}

}

//networkcontroller.h

#ifndef NETWORKCONTROLLER\_H

#define NETWORKCONTROLLER\_H

#include "model/client.h"

#include "model/server.h"

#include "model/message.h"

#include <QThread>

#include <QAudioInput>

#include <QAudioOutput>

class NetworkController : public QThread

{

Q\_OBJECT

Server \* server;

Client \* client;

bool n\_exit;

QMutex change\_state;

Message nothing;

Message sendSound;

Message callingSound;

Message recieveSound;

bool is\_send;

bool is\_call;

bool is\_config;

QAudioInput \* audioInput;

QAudioOutput \* audioOutput;

char szum\_level;

void run() override;

Message getSzum();

void prepareToAudioOutput(Message& msg);

public:

NetworkController(bool server, QString ip = "127.0.0.1");

void config\_send(int frequency);

void config\_listen(int frequency);

void config\_kill();

void call\_on();

void call\_off();

void setVolume(char level);

void setSzumLevel(char level);

void setRecordVolume(char level);

~NetworkController();

signals:

void set\_call();

void reset\_call();

};

#endif // NETWORKCONTROLLER\_H

//networkcontroller.cpp

#include "networkcontroller.h"

#include <QAudioDeviceInfo>

#include <QDebug>

#include <math.h>

void NetworkController::run()

{

auto audioOut = this->audioOutput->start();

auto audioIn = this->audioInput->start();

bool switched = false;

bool call = false;

qDebug("network thread is run");

while(n\_exit)

{

this->change\_state.lock();

if (this->is\_config)

{

if (this->is\_send)

{

if (this->is\_call)

{

if (call != true)

{

switched = true;

}

call = true;

client->send(callingSound);

client->recieve(callingSound);

audioOut->write(callingSound.audio\_data, MESSAGE\_SIZE);

}

else

{

if (call != false)

{

switched = true;

}

call = false;

audioIn->read(sendSound.audio\_data, MESSAGE\_SIZE);

client->send(sendSound);

client->recieve(sendSound);

audioOut->write(sendSound.audio\_data, MESSAGE\_SIZE);

}

}

else

{

memset(this->recieveSound.audio\_data, '\0', MESSAGE\_SIZE);

client->send(this->recieveSound);

client->recieve(this->recieveSound);

prepareToAudioOutput(recieveSound);

audioOut->write(recieveSound.audio\_data, MESSAGE\_SIZE);

if (recieveSound.call)

{

if (call == false)

{

switched = true;

}

call = true;

recieveSound.call = false;

}

else

{

if (call == true)

{

switched = true;

}

call = false;

}

}

}

else

{

client->send(nothing);

client->recieve(nothing);

}

this->change\_state.unlock();

if(switched)

{

switched = false;

if (call)

{

emit set\_call();

}

else

{

emit reset\_call();

}

}

}

}

Message NetworkController::getSzum()

{

Message message;

for (int i = 0; i <MESSAGE\_SIZE; i ++)

message.audio\_data[i] = rand() % 100;

return message;

}

void NetworkController::prepareToAudioOutput(Message &msg)

{

Message szum = getSzum();

switch (szum\_level) {

case 0:

return;

case 1:

for (int i = 0; i < MESSAGE\_SIZE; i++)

{

szum.audio\_data[i] \*= 0.1;

msg.audio\_data[i] += szum.audio\_data[i];

}

return;

case 2:

for (int i = 0; i < MESSAGE\_SIZE; i++)

{

szum.audio\_data[i] \*= 0.5;

msg.audio\_data[i] += szum.audio\_data[i];

}

return;

}

}

NetworkController::NetworkController(bool server, QString ip)

{

if (server)

this->server = new Server();

else

this->server = nullptr;

nothing.call = false;

nothing.sending = false;

nothing.disconect = false;

nothing.frequency = -1;

sendSound.call = false;

sendSound.sending = true;

sendSound.disconect = false;

callingSound.call = true;

callingSound.sending = true;

callingSound.disconect = false;

for (int i = 0; i < MESSAGE\_SIZE; i++)

callingSound.audio\_data[i] = char(50 \* sin(i \* MESSAGE\_SIZE));

recieveSound.call = false;

recieveSound.sending = false;

recieveSound.disconect = false;

this->n\_exit = true;

QAudioDeviceInfo infoInput = QAudioDeviceInfo::defaultInputDevice();

QAudioDeviceInfo infoOutput = QAudioDeviceInfo::defaultOutputDevice();

QAudioFormat format;

format.setSampleRate(8000);

format.setChannelCount(1);

format.setSampleSize(RATE\_BYTES);

format.setSampleType(RATE\_BYTES == 8 ? QAudioFormat::UnSignedInt : QAudioFormat::SignedInt);

format.setByteOrder(QAudioFormat::LittleEndian);

format.setCodec("audio/pcm");

this->audioInput = new QAudioInput(infoInput, format);

this->audioOutput = new QAudioOutput(infoOutput, format);

if (server)

this->client = new Client("127.0.0.1");

else

this->client = new Client(ip);

this->is\_call = false;

this->is\_send = false;

this->is\_config = false;

this->szum\_level = 2;

}

void NetworkController::config\_send(int frequency)

{

this->change\_state.lock();

this->is\_config = true;

this->is\_send = true;

sendSound.frequency = frequency;

callingSound.frequency = frequency;

this->change\_state.unlock();

}

void NetworkController::config\_listen(int frequency)

{

this->change\_state.lock();

this->is\_config = true;

this->is\_send = false;

recieveSound.frequency = frequency;

this->change\_state.unlock();

}

void NetworkController::config\_kill()

{

this->change\_state.lock();

this->is\_config = false;

this->change\_state.unlock();

}

void NetworkController::call\_on()

{

this->change\_state.lock();

this->is\_call = true;

this->change\_state.unlock();

}

void NetworkController::call\_off()

{

this->change\_state.lock();

this->is\_call = false;

this->change\_state.unlock();

}

void NetworkController::setVolume(char level)

{

change\_state.lock();

this->audioOutput->setVolume(qreal(level) / 100);

change\_state.unlock();

}

void NetworkController::setSzumLevel(char level)

{

change\_state.lock();

this->szum\_level = level;

change\_state.unlock();

}

void NetworkController::setRecordVolume(char level)

{

change\_state.lock();

this->audioInput->setVolume(qreal(level) / 100);

change\_state.unlock();

}

NetworkController::~NetworkController()

{

if (server)

delete server;

delete client;

delete audioInput;

delete audioOutput;

this->n\_exit = false;

}

//client.h

#pragma once

#include <QString>

#include "model/message.h"

#include "instruments/socket.h"

class Client {

Socket socket;

public:

Client(QString ip\_address);

void send(Message& msg);

void recieve(Message& msg);

~Client();

};

//client.cpp

#include "model/client.h"

#include <string>

Client::Client(QString ip\_address)

{

if (this->socket.connect(ip\_address.toStdString().c\_str(), 5000))

{

//throw std::string("Либо сервера не существует, либо проверьте подключение!");

}

}

void Client::send(Message &msg)

{

this->socket.send(reinterpret\_cast<char\*>(&msg), sizeof(msg));

}

void Client::recieve(Message &msg)

{

this->socket.receive(reinterpret\_cast<char\*>(&msg), sizeof(msg));

}

Client::~Client()

{

Message m;

m.disconect = true;

m.sending = false;

send(m);

}

//server.h

#pragma once

#include "instruments/socket.h"

#include <QThread>

#include <list>

#include <QMutex>

#include "model/message.h"

class Server

{

bool exit\_flag;

QMutex list\_mutex;

Socket socket;

std::list<Socket\*> sockets;

std::list<Message> messages;

friend class ListenningThread;

friend class SendRecieveThread;

friend class SendThread;

class ListenningThread : public QThread

{

Server \* parentServer;

void run() override;

public:

ListenningThread(Server\* parent);

} listenningThread;

class SendRecieveThread : public QThread

{

Server \* parentServer;

void run() override;

public:

SendRecieveThread(Server \* parent);

} recieveThread;

class IpRequest : public QThread

{

Server \* parentServer;

void run() override;

public:

IpRequest(Server \* parent);

}; //ipRequest;

public:

Server();

static QString getServerIp();

void close();

};

//server.cpp

//#include "model/server.h"

#include "model/client.h"

Server::Server() :

listenningThread(this),

recieveThread(this)

{

this->socket.bind(5000);

this->exit\_flag = true;

listenningThread.start();

recieveThread.start(QThread::TimeCriticalPriority);

}

QString Server::getServerIp()

{

QString str("224.0.0.0");

Client client(str);

return "";

}

void Server::close()

{

this->exit\_flag = false;

}

void Server::ListenningThread::run()

{

while(parentServer->exit\_flag)

{

parentServer->socket.listen();

Socket \* client = parentServer->socket.accept();

qDebug("launched");

parentServer->list\_mutex.lock();

parentServer->sockets.push\_front(client);

parentServer->messages.push\_front(Message());

parentServer->list\_mutex.unlock();

}

}

Server::ListenningThread::ListenningThread(Server \*parent)

{

this->parentServer = parent;

}

void Server::SendRecieveThread::run()

{

while(parentServer->exit\_flag)

{

parentServer->list\_mutex.lock();

auto clientPtr = parentServer->sockets.begin();

auto messagePtr = parentServer->messages.begin();

for(;messagePtr != parentServer->messages.end(); clientPtr++, messagePtr++)

{

if ((\*clientPtr)->receive(reinterpret\_cast<char\*>(&\*messagePtr),

sizeof(Message)) == -1)

{

messagePtr->call = false;

messagePtr->sending = false;

messagePtr->disconect = true;

messagePtr->frequency = -1;

}

}

clientPtr = parentServer->sockets.begin();

messagePtr = parentServer->messages.begin();

for(;messagePtr != parentServer->messages.end(); clientPtr++, messagePtr++)

{

if(!messagePtr->sending && !messagePtr->disconect)

{

memset(messagePtr->audio\_data, '\0', MESSAGE\_SIZE);

for (auto &message : parentServer->messages)

{

if (message.sending && !message.disconect &&

message.frequency == messagePtr->frequency)

{

for (int i = 0; i < MESSAGE\_SIZE / 8; i++)

{

reinterpret\_cast<long long \*>(messagePtr->audio\_data)[i] +=

reinterpret\_cast<long long \*>(message.audio\_data)[i];

}

if (message.call)

messagePtr->call = true;

}

}

}

}

clientPtr = parentServer->sockets.begin();

messagePtr = parentServer->messages.begin();

for(;messagePtr != parentServer->messages.end(); clientPtr++, messagePtr++)

{

if(messagePtr->disconect)

{

qDebug("disconnect");

// auto \* socket = \*clientPtr;

// delete socket;

parentServer->messages.remove(\*messagePtr);

parentServer->sockets.remove(\*clientPtr);

messagePtr = parentServer->messages.begin();

clientPtr = parentServer->sockets.begin();

}

}

clientPtr = parentServer->sockets.begin();

messagePtr = parentServer->messages.begin();

for(;messagePtr != parentServer->messages.end(); clientPtr++, messagePtr++)

{

(\*clientPtr)->send(reinterpret\_cast<char\*>(&\*messagePtr), sizeof(Message));

}

parentServer->list\_mutex.unlock();

}

}

Server::SendRecieveThread::SendRecieveThread(Server \*parent)

{

this->parentServer = parent;

}

void Server::IpRequest::run()

{

Socket listener;

listener.connect("224.0.0.0", 5000);

char responce[16], request[16];

memset(responce, '\0', 16);

memset(request, '\0', 16);

while(parentServer->exit\_flag)

{

listener.receive(request, 16);

if (strcmp(request, "ip req") == 0)

{

}

}

}

Server::IpRequest::IpRequest(Server \*parent)

{

this->parentServer = parent;

}

//socket.h

#ifndef SOCKETCLIENT\_H

#define SOCKETCLIENT\_H

#include <WinSock2.h>

class Socket

{

SOCKET sock;

sockaddr\_in address;

bool crack;

Socket(int);

public:

static void init();

static void clear();

int send(char\* data, int len);

int receive(char\* buffer, int len);

int connect(const char \* ip\_addr, int port);

int bind(int port);

int listen();

Socket \* accept();

Socket();

~Socket();

};

#endif // SOCKETCLIENT\_H

//socket.cpp

#include "socket.h"

#include <winsock2.h>

#include <QDebug>

#include <string.h>

//#pragma comment(lib, "ws2\_32.lib")

// TODO UDP SERVER

// google it: broadcast

// bind 0.0.0.0

// or use multicast

Socket::Socket()

{

crack = true;

if ((this->sock = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET)

qDebug() << "error sock init";

qDebug() << "sock is inited";

}

Socket::~Socket()

{

if (crack)

closesocket(this->sock);

else

crack = true;

}

int Socket::connect(const char \* ip\_addr, int port)

{

memset(&this->address, '\0', sizeof(this->address));

this->address.sin\_addr.s\_addr = inet\_addr(ip\_addr);

this->address.sin\_family = AF\_INET;

this->address.sin\_port = htons(port);

if (::connect(

this->sock,

(sockaddr\*)&this->address,

sizeof(address)

) < 0)

{

qDebug() << "error connection";

return 2;

}

qDebug() << "conetion is good";

return 0;

}

int Socket::send(char\* data, int len)

{

if (::send(this->sock, data, len, 0) < 0)//is\_server ? 0 : MESSAGE\_SIZE) < 0)

return 1;

else

return 0;

}

int Socket::receive(char\* buffer, int len)

{

int recv\_size;

recv\_size = recv(this->sock, buffer, len, 0);//is\_server ? MESSAGE\_SIZE : 0);

if (recv\_size == SOCKET\_ERROR)

return -1;

else {

//buffer[recv\_size] = 0;

return recv\_size;

}

}

int Socket::bind(int port)

{

this->address.sin\_family = AF\_INET;

this->address.sin\_addr.s\_addr = INADDR\_ANY;

this->address.sin\_port = htons(port);

auto resp = ::bind(this->sock, (sockaddr\*)&this->address, sizeof(address));

return resp < 0;

}

int Socket::listen()

{

auto resp = ::listen(this->sock, 1);

return resp < 0;

}

Socket \* Socket::accept()

{

Socket \*sock = new Socket(6);

int size = sizeof(address);

sock->sock = ::accept(this->sock, (sockaddr\*)&this->address, &size);

qDebug() << (sock->sock == INVALID\_SOCKET);

return sock;

}

Socket::Socket(int)

{

crack = false;

}

void Socket::init() {

WSADATA wsa;

WSAStartup(MAKEWORD(2,2), &wsa);

}

void Socket::clear() {

WSACleanup();

}

//message.h

#ifndef MESSAGE\_H

#define MESSAGE\_H

#define MESSAGE\_SIZE 1024

#define RATE\_BYTES 8

struct **Message**

{

char audio\_data[MESSAGE\_SIZE];

int frequency;

bool disconect;

bool call;

bool sending;

int id;

public:

bool operator==(const Message& other) const

{

if (frequency != other.frequency)

return false;

if (disconect != other.disconect)

return false;

if (call != other.call)

return false;

if (sending != other.sending)

return false;

for(int i = 0; i < MESSAGE\_SIZE; i++)

if (audio\_data[i] != other.audio\_data[i])

return false;

return true;

}

};

#endif // MESSAGE\_H