**Приложение 1. Листинг программы**

**Файл “LVS.pro” (Конфигурационный файл)**

**Автор: Агеев Н.О.**

QT += core gui

QT += core

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = LVS

TEMPLATE = app

SOURCES += main.cpp\

mainwindow.cpp \

ou.cpp \

controller.cpp \

message.cpp \

lvs.cpp \

timecounter.cpp \

logger.cpp

HEADERS += mainwindow.h \

ou.h \

controller.h \

message.h \

lvs.h \

timecounter.h \

logger.h

FORMS += mainwindow.ui

DISTFILES +=

**Файл “lvs.h” (Модуль LVS)**

**Автор: Агеев Н.О.**

#ifndef LVS\_H

#define LVS\_H

#include "ou.h"

#include "message.h"

#include "timecounter.h"

#include <QApplication>

#include <map>

using namespace std;

class LVS

{

public:

map<QString,QString> statuses;

map<QString,QString> lines;

map<int,OU\*> clients;

QString line;

QString status;

TimeCounter\* timer;

int msg\_amount;

LVS(QString line\_type);

find\_generator();

makeRound();

testRound();

};

#endif // LVS\_H

**Файл “ou.h” (Модуль ОУ)**

**Автор: Агеев Н.О.**

#ifndef OU\_H

#define OU\_H

#include "controller.h"

#include <QApplication>

#include <map>

using namespace std;

class OU

{

public:

map<QString,QString> states;

QString state;

Controller\* controller;

OU();

message\_not\_delivered();

blockMessage();

};

#endif // OU\_H

**Файл “ou.cpp”**

**Автор: Агеев Н.О.**

#include "ou.h"

OU::OU()

{

states["working"] = "working";

states["not\_working"] = "not\_working";

states["blocked"] = "blocked";

states["busy"] = "busy";

states["message\_error"] = "message\_error";

states["failure"] = "failure";

states["denial"] = "denial";

states["generator"] = "generator";

state = states["working"];

controller = new Controller;}

**Файл “lvs.cpp”**

**Авторы: Агеев Н.О., Калягина А.Д**

#include "lvs.h"

#include "ou.h"

#include <QDebug>

using namespace std;

LVS::LVS(QString line\_type)

{

if(line\_type == "A")

for(int i=1;i<=18;i++){

clients[i] = new OU();

}

lines["A"] = "A";

lines["B"] = "B";

line = line\_type;

statuses["working"] = "working";

statuses["generating"] = "generating";

statuses["off"] = "off";

status = statuses["off"];

timer = new TimeCounter;

msg\_amount = 0;

}

LVS::find\_generator()

{

int msgs = 0;

for(int i=1;i<=18;i++){

timer->addTime("block");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

}

msgs += 18;

int i = 0;

do{

i++;

timer->addTime("unblock");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

if(clients[i]->state != "generator")timer->addTime("answer");

msgs += 2;

}while(clients[i]->state != "generator");

return msgs;

}

LVS::makeRound()

{

Message\* msg = new Message;

for(int i=1;i<=18;i++){

QString encoded\_msg = msg->encodeMessage(1,i,"short-give\_response","request");

clients[i]->controller->received\_msg = encoded\_msg;

if(clients[i]->state == "working")

clients[i]->controller->received\_msg = encoded\_msg;

else clients[i]->controller->received\_msg = "none";

//qDebug() << clients[i]->controller->received\_msg << endl;

if(clients[i]->state == "not\_working")return i;

if(clients[i]->state == "failure")return i;

if(clients[i]->state == "denial")return i;

if(clients[i]->state == "generator")return i;

}

return 0;

}

LVS::testRound()

{

Message\* msg = new Message;

msg\_amount = 0;

int fail\_ou = 0;

if(status == "generating"){

for(int i=1;i<=18;i++){

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

}

msg\_amount += 18;

status = statuses["working"];

}else{

for(int i=1;i<=18;i++){

QString encoded\_msg = msg->encodeMessage(1,i,"short-give\_response","request");

clients[i]->controller->received\_msg = encoded\_msg;

if(clients[i]->state == "working" || clients[i]->state == "blocked"){

clients[i]->controller->received\_msg = encoded\_msg;

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

msg\_amount += 1;

}

else clients[i]->controller->received\_msg = "none";

//qDebug() << clients[i]->controller->received\_msg << endl;

if(clients[i]->state == "not\_working")return i;

if(clients[i]->state == "failure"){

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

clients[i]->state = "working";

msg\_amount += 3;

fail\_ou = i;

}

if(clients[i]->state == "busy"){

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

timer->addTime("pause\_if\_bizy");

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("answer");

clients[i]->state = "working";

}

if(clients[i]->state == "denial"){

for(int i=0;i<3;i++){

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

timer->addTime("command");

timer->addTime("word");

timer->addTime("pause\_before\_answer");

}

msg\_amount += 3;

fail\_ou = i;

}

if(clients[i]->state == "generator"){

msg\_amount += find\_generator();

clients[i]->state = "blocked";

status = statuses["working"];

fail\_ou = i;

}

}

}

return fail\_ou;

}

**Файл “timecounter.cpp”**

**Автор: Калягина А.Д.**

#include "timecounter.h"

TimeCounter::TimeCounter()

{

total\_time = 0;

time\_types["pause\_if\_bizy"] = 5000;

time\_types["command"] = 20;

time\_types["pause\_before\_answer"] = 12;

time\_types["word"] = 12\*20;

time\_types["block"] = 20;

time\_types["unblock"] = 20;

time\_types["pause\_between\_messages"] = 1000;

time\_types["max\_word\_length"] = 800;

time\_types["answer"] = 20;

}

TimeCounter::addTime(QString type)

{

total\_time += time\_types[type];

}

int TimeCounter::getTime()

{

return total\_time;

}

**Файл “timecounter.h” (Модуль счетчика)**

**Автор: Калягина А.Д.**

#ifndef TIMECOUNTER\_H

#define TIMECOUNTER\_H

#include <QApplication>

#include <map>

using namespace std;

class TimeCounter

{

public:

int total\_time;

map<QString,int> time\_types;

TimeCounter();

addTime(QString type);

int getTime();

};

#endif // TIMECOUNTER\_H

**Файл “mainwindow.ui” (Модуль графического интерфейса)**

**Автор: Агеева А.И.**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry"> --->

<property name="windowTitle"> --->

<property name="windowIcon"> --->

<property name="styleSheet"> --->

<widget class="QWidget" name="centralWidget"> --->

<widget class="QMenuBar" name="menuBar"> --->

<widget class="QToolBar" name="mainToolBar"> --->

<widget class="QStatusBar" name="statusBar"/>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>

**Файл “mainwindow.h” (Модуль графического интерфейса)**

**Автор: Агеева А.И.**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QApplication>

#include <QMainWindow>

#include <QFrame>

#include <QRadioButton>

#include <map>

using namespace std;

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

map<int,QFrame\*> linesList;

map<int,QRadioButton\*> working\_radios;

map<int,QRadioButton\*> failure\_radios;

map<int,QRadioButton\*> denial\_radios;

map<int,QRadioButton\*> generating\_radios;

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_startButton\_clicked();

void on\_working\_clicked();

void on\_failure\_clicked();

void on\_denial\_clicked();

void on\_generating\_clicked();

void on\_roundButton\_clicked();

void colorizeLines(int fail\_ou);

void on\_changeLine\_clicked();

void on\_testButton\_clicked();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**Файл “main.cpp” (Модуль запуска графического интерфейса)**

**Автор: Агеева А.И.**

#include "mainwindow.h"

#include "OU.h"

#include "Controller.h"

#include "Message.h"

#include <string>

#include <iostream>

#include <QApplication>

using namespace std;

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**Файл “logger.h” (Модуль логгирования).**

**Автор: Слипкус В.Ю.**

#ifndef LOGGER\_H

#define LOGGER\_H

#include <QApplication>

#include <QDateTime>

#include <QFile>

#include <map>

using namespace std;

class Logger

{

public:

map<QString,QString> phrases;

QFile\* logsfile;

Logger(QString filename);

QString getLogsLine(QString type);

addToFile(QString logline);

~Logger();

};

#endif // LOGGER\_H

**Файл “logger.cpp”**

**Автор: Слипкус В.Ю.**

#include "logger.h"

#include <QTextStream>

#include <QTextCodec>

Logger::Logger(QString filename)

{

phrases["lvs\_start"] = "Запуск ЛВС";

phrases["lvs\_restart"] = "Перезапуск ЛВС";

phrases["status\_working"] = "Статус: Все ОУ работают";

phrases["ou\_turn\_off"] = "ОУ № искусственно выключен";

phrases["ou\_turn\_on"] = "ОУ № искусственно включен";

phrases["ou\_turn\_failure"] = "На ОУ № искусственно включен сбой";

phrases["ou\_turn\_denial"] = "На ОУ № искусственно включен отказ";

phrases["ou\_turn\_generating"] = "На ОУ № искусственно включена генерация";

phrases["make\_round"] = "Производится обход";

phrases["status\_ou\_off"] = "Статус: ОУ № выключен";

phrases["status\_ou\_failure"] = "Статус: На ОУ № произошел сбой";

phrases["status\_ou\_denial"] = "Статус: На ОУ № произошел отказ";

phrases["status\_ou\_generator"] = "Статус: ОУ № генератор";

phrases["makeround\_fail"] = "Невозможно выполнить обход на линии";

phrases["active\_line"] = "Активная линия №";

logsfile = new QFile(filename);

logsfile->*open*(QIODevice::WriteOnly);

addToFile(QDateTime::currentDateTime().toString("dd.MM.yyyy/hh:mm:ss"));

addToFile("------------------------------");

}

QString Logger::getLogsLine(QString type)

{

type = (phrases[type]!="")? phrases[type] : type;

addToFile(type);

return type;

}

Logger::addToFile(QString logline)

{

QTextStream out(logsfile);

QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF8"));

out << logline.toUtf8() << "\n";

}

Logger::~Logger()

{

addToFile("Выключение ЛВС");

addToFile("------------------------------");

logsfile->*close*();

}

**Файл “mainwindow.cpp”**

**Авторы: Агеева А.И., Агеев Н.О., Калягина А.Д., Слипкус В.Ю., Стерпу Е.К.**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "lvs.h"

#include "timecounter.h"

#include "logger.h"

#include <QDebug>

#include <string>

using namespace std;

LVS\* lvs\_base = new LVS("A");

LVS\* lvs\_reserv = new LVS("B");

LVS\* lvs;

TimeCounter\* timer;

Logger\* logger = new Logger("logs.txt");

Logger\* logger\_test = new Logger("test.txt");

QString tmp\_log = "";

int total\_time = 0;

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

linesList[1] = ui->line\_1;

linesList[2] = ui->line\_2;

linesList[3] = ui->line\_3;

linesList[4] = ui->line\_4;

linesList[5] = ui->line\_5;

linesList[6] = ui->line\_6;

linesList[7] = ui->line\_7;

linesList[8] = ui->line\_8;

linesList[9] = ui->line\_9;

linesList[10] = ui->line\_10;

linesList[11] = ui->line\_11;

linesList[12] = ui->line\_12;

linesList[13] = ui->line\_13;

linesList[14] = ui->line\_14;

linesList[15] = ui->line\_15;

linesList[16] = ui->line\_16;

linesList[17] = ui->line\_17;

linesList[18] = ui->line\_18;

working\_radios[1] = ui->working\_1;

working\_radios[2] = ui->working\_2;

working\_radios[3] = ui->working\_3;

working\_radios[4] = ui->working\_4;

working\_radios[5] = ui->working\_5;

working\_radios[6] = ui->working\_6;

working\_radios[7] = ui->working\_7;

working\_radios[8] = ui->working\_8;

working\_radios[9] = ui->working\_9;

working\_radios[10] = ui->working\_10;

working\_radios[11] = ui->working\_11;

working\_radios[12] = ui->working\_12;

working\_radios[13] = ui->working\_13;

working\_radios[14] = ui->working\_14;

working\_radios[15] = ui->working\_15;

working\_radios[16] = ui->working\_16;

working\_radios[17] = ui->working\_17;

working\_radios[18] = ui->working\_18;

generating\_radios[1] = ui->generating\_1;

generating\_radios[2] = ui->generating\_2;

generating\_radios[3] = ui->generating\_3;

generating\_radios[4] = ui->generating\_4;

generating\_radios[5] = ui->generating\_5;

generating\_radios[6] = ui->generating\_6;

generating\_radios[7] = ui->generating\_7;

generating\_radios[8] = ui->generating\_8;

generating\_radios[9] = ui->generating\_9;

generating\_radios[10] = ui->generating\_10;

generating\_radios[11] = ui->generating\_11;

generating\_radios[12] = ui->generating\_12;

generating\_radios[13] = ui->generating\_13;

generating\_radios[14] = ui->generating\_14;

generating\_radios[15] = ui->generating\_15;

generating\_radios[16] = ui->generating\_16;

generating\_radios[17] = ui->generating\_17;

generating\_radios[18] = ui->generating\_18;

failure\_radios[1] = ui->failure\_1;

failure\_radios[2] = ui->failure\_2;

failure\_radios[3] = ui->failure\_3;

failure\_radios[4] = ui->failure\_4;

failure\_radios[5] = ui->failure\_5;

failure\_radios[6] = ui->failure\_6;

failure\_radios[7] = ui->failure\_7;

failure\_radios[8] = ui->failure\_8;

failure\_radios[9] = ui->failure\_9;

failure\_radios[10] = ui->failure\_10;

failure\_radios[11] = ui->failure\_11;

failure\_radios[12] = ui->failure\_12;

failure\_radios[13] = ui->failure\_13;

failure\_radios[14] = ui->failure\_14;

failure\_radios[15] = ui->failure\_15;

failure\_radios[16] = ui->failure\_16;

failure\_radios[17] = ui->failure\_17;

failure\_radios[18] = ui->failure\_18;

denial\_radios[1] = ui->denial\_1;

denial\_radios[2] = ui->denial\_2;

denial\_radios[3] = ui->denial\_3;

denial\_radios[4] = ui->denial\_4;

denial\_radios[5] = ui->denial\_5;

denial\_radios[6] = ui->denial\_6;

denial\_radios[7] = ui->denial\_7;

denial\_radios[8] = ui->denial\_8;

denial\_radios[9] = ui->denial\_9;

denial\_radios[10] = ui->denial\_10;

denial\_radios[11] = ui->denial\_11;

denial\_radios[12] = ui->denial\_12;

denial\_radios[13] = ui->denial\_13;

denial\_radios[14] = ui->denial\_14;

denial\_radios[15] = ui->denial\_15;

denial\_radios[16] = ui->denial\_16;

denial\_radios[17] = ui->denial\_17;

denial\_radios[18] = ui->denial\_18;

for(int i=1;i<=18;i++){

working\_radios[i]->setToolTip("Включить/Выключить");

failure\_radios[i]->setToolTip("Сбой (вкл/выкл)");

denial\_radios[i]->setToolTip("Отказ (вкл/выкл)");

generating\_radios[i]->setToolTip("Генерация (вкл/выкл)");

connect(working\_radios[i], SIGNAL (clicked()), this, SLOT (on\_working\_clicked()));

connect(failure\_radios[i], SIGNAL (clicked()), this, SLOT (on\_failure\_clicked()));

connect(denial\_radios[i], SIGNAL (clicked()), this, SLOT (on\_denial\_clicked()));

connect(generating\_radios[i], SIGNAL (clicked()), this, SLOT (on\_generating\_clicked()));

lvs\_reserv->clients[i] = lvs\_base->clients[i];

}

lvs = lvs\_base;

}

MainWindow::~*MainWindow*()

{

logger->~Logger();

delete ui;

}

void MainWindow::on\_startButton\_clicked()

{

for(int i=1;i<=18;i++){

working\_radios[i]->setCheckable(true);

working\_radios[i]->setChecked(true);

failure\_radios[i]->setCheckable(true);

failure\_radios[i]->setChecked(false);

denial\_radios[i]->setCheckable(true);

denial\_radios[i]->setChecked(false);

generating\_radios[i]->setCheckable(true);

generating\_radios[i]->setChecked(false);

}

ui->testButton->setEnabled(false);

ui->roundButton->setEnabled(true);

ui->changeLine->setEnabled(true);

ui->lvs\_line->setText(lvs->line);

ui->line->setStyleSheet("background: green");

tmp\_log = (lvs->status == lvs->statuses["working"])? logger->getLogsLine("lvs\_restart") : logger->getLogsLine("lvs\_start");

ui->logs->append(tmp\_log);

tmp\_log = logger->phrases["active\_line"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",lvs->line)));

lvs->status = lvs->statuses["working"];

ui->lvs\_status->setText("Все ОУ работают");

ui->startButton->setText("Перезапустить");

for(int i=1;i<=18;i++)linesList[i]->setStyleSheet("background: none");

for(int i=1;i<=18;i++)lvs->clients[i]->state = "working";

}

void MainWindow::on\_working\_clicked()

{

QRadioButton\* working = qobject\_cast<QRadioButton\*>(sender());

QString ou\_num = working->objectName();

int ou\_number = (ou\_num.mid(ou\_num.indexOf("\_")+1)).toInt();

QString ou\_state = lvs->clients[ou\_number]->state;

lvs->status = "working";

ui->lvs\_status->setText("");

for(int i=1;i<=18;i++){

lvs->clients[i]->state = "working";

working\_radios[i]->setChecked(true);

failure\_radios[i]->setChecked(false);

denial\_radios[i]->setChecked(false);

generating\_radios[i]->setChecked(false);

linesList[i]->setStyleSheet("background: none");

}

if(ou\_state != "working"){

lvs->clients[ou\_number]->state = "working";

working->setChecked(true);

tmp\_log = logger->phrases["ou\_turn\_on"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

}else{

lvs->clients[ou\_number]->state = lvs->clients[ou\_number]->states["not\_working"];

working->setChecked(false);

tmp\_log = logger->phrases["ou\_turn\_off"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

}

}

void MainWindow::on\_failure\_clicked()

{

QRadioButton\* failure\_ou = qobject\_cast<QRadioButton\*>(sender());

QString ou\_num = failure\_ou->objectName();

int ou\_number = (ou\_num.mid(ou\_num.indexOf("\_")+1)).toInt();

QString ou\_state = lvs->clients[ou\_number]->state;

for(int i=1;i<=18;i++){

lvs->clients[i]->state = "working";

working\_radios[i]->setChecked(true);

failure\_radios[i]->setChecked(false);

denial\_radios[i]->setChecked(false);

generating\_radios[i]->setChecked(false);

linesList[i]->setStyleSheet("background: none");

}

if(ou\_state != "working"){

lvs->clients[ou\_number]->state = "working";

lvs->status = "working";

} else{

lvs->clients[ou\_number]->state = lvs->clients[ou\_number]->states["failure"];

failure\_ou->setChecked(true);

tmp\_log = logger->phrases["ou\_turn\_failure"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

}

ui->lvs\_status->setText("");

}

void MainWindow::on\_denial\_clicked()

{

QRadioButton\* denial = qobject\_cast<QRadioButton\*>(sender());

QString ou\_num = denial->objectName();

int ou\_number = (ou\_num.mid(ou\_num.indexOf("\_")+1)).toInt();

QString ou\_state = lvs->clients[ou\_number]->state;

for(int i=1;i<=18;i++){

lvs->clients[i]->state = "working";

working\_radios[i]->setChecked(true);

failure\_radios[i]->setChecked(false);

denial\_radios[i]->setChecked(false);

generating\_radios[i]->setChecked(false);

linesList[i]->setStyleSheet("background: none");

}

if(ou\_state != "working"){

lvs->clients[ou\_number]->state = "working";

lvs->status = "working";

} else{

lvs->clients[ou\_number]->state = lvs->clients[ou\_number]->states["denial"];

denial->setChecked(true);

tmp\_log = logger->phrases["ou\_turn\_denial"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

}

ui->lvs\_status->setText("");

}

void MainWindow::on\_generating\_clicked()

{

QRadioButton\* generating = qobject\_cast<QRadioButton\*>(sender());

QString ou\_num = generating->objectName();

int ou\_number = (ou\_num.mid(ou\_num.indexOf("\_")+1)).toInt();

QString ou\_state = lvs->clients[ou\_number]->state;

for(int i=1;i<=18;i++){

lvs->clients[i]->state = "working";

working\_radios[i]->setChecked(true);

failure\_radios[i]->setChecked(false);

denial\_radios[i]->setChecked(false);

generating\_radios[i]->setChecked(false);

linesList[i]->setStyleSheet("background: none");

}

if(ou\_state != "working"){

lvs->clients[ou\_number]->state = "working";

lvs->status = "working";

//tmp\_log = logger->phrases["ou\_turn\_on"];

//ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

} else{

lvs->clients[ou\_number]->state = lvs->clients[ou\_number]->states["generator"];

lvs->status = "generating";

generating->setChecked(true);

tmp\_log = logger->phrases["ou\_turn\_generating"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(ou\_number))));

}

ui->lvs\_status->setText("");

}

void MainWindow::on\_roundButton\_clicked()

{

int fail\_ou = lvs->makeRound();

ui->logs->append(logger->getLogsLine(logger->phrases["make\_round"]));

colorizeLines(fail\_ou);

if(lvs->status != "generating"){

if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "not\_working"){

ui->lvs\_status->setText("ОУ " + QString::number(fail\_ou) + " выключен");

tmp\_log = logger->phrases["status\_ou\_off"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(fail\_ou))));

}

else if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "failure"){

lvs->clients[fail\_ou]->state = "working";

failure\_radios[fail\_ou]->setChecked(false);

ui->lvs\_status->setText("На ОУ " + QString::number(fail\_ou) + " произошел сбой");

tmp\_log = logger->phrases["status\_ou\_failure"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(fail\_ou))));

}

else if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "denial"){

ui->lvs\_status->setText("На ОУ " + QString::number(fail\_ou) + " произошел отказ");

tmp\_log = logger->phrases["status\_ou\_denial"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(fail\_ou))));

}

else if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "generator"){

lvs->clients[fail\_ou]->state = "not\_working";

ui->lvs\_status->setText("ОУ " + QString::number(fail\_ou) + " генератор, будет выключен");

working\_radios[fail\_ou]->setChecked(false);

generating\_radios[fail\_ou]->setChecked(false);

tmp\_log = logger->phrases["status\_ou\_generator"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(fail\_ou))));

tmp\_log = logger->phrases["ou\_turn\_off"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",QString::number(fail\_ou))));

}

else{

ui->lvs\_status->setText("Все ОУ работают");

ui->logs->append(logger->getLogsLine(logger->phrases["status\_working"]));

}

} else{

ui->lvs\_status->setText("Невозможно выполнить обход");

ui->logs->append(logger->getLogsLine(logger->phrases["makeround\_fail"]));

}

}

void MainWindow::colorizeLines(int fail\_ou)

{

if(lvs->status != "generating"){

ui->line->setStyleSheet("background: green");

for(int i=1;i<=18;i++)linesList[i]->setStyleSheet("background: green");

if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "not\_working")linesList[fail\_ou]->setStyleSheet("background: red");

if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "failure")linesList[fail\_ou]->setStyleSheet("background: blue");

if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "denial")linesList[fail\_ou]->setStyleSheet("background: black");

if(fail\_ou != 0 && lvs->clients[fail\_ou]->state == "generator")linesList[fail\_ou]->setStyleSheet("background: yellow");

} else ui->line->setStyleSheet("background: orange");

}

void MainWindow::on\_changeLine\_clicked()

{

lvs = (lvs == lvs\_base)? lvs\_reserv : lvs\_base;

tmp\_log = logger->phrases["active\_line"];

ui->logs->append(logger->getLogsLine(tmp\_log.replace("№",lvs->line)));

if(lvs->status == "generating"){

int line\_generated = 0;

for(int i=1;i<=18;i++)if(lvs->clients[i]->state == "generator")line\_generated = 1;

if(line\_generated == 1)lvs->status = "generating";

else lvs->status = "working";

}

ui->lvs\_line->setText(lvs->line);

if(lvs->status != "generating"){

lvs->status = lvs->statuses["working"];

ui->line->setStyleSheet("background: green");

for(int i=1;i<=18;i++)linesList[i]->setStyleSheet("background: none");

}

}

void MainWindow::on\_testButton\_clicked()

{

int N = 1;

int total\_generating, total\_failure, total\_denial, total\_busy, total\_group, total\_count;

QString logger\_str = "Сеанс Сообщений: Сбой | Отказ | Занят | Генерация | Время";

logger\_test->addToFile(logger\_str);

qDebug() << "Группа Сообщений:" << "Сбой" << "Отказ" << "Занят" << "Генерация" << "Время" << endl;

for(int j=0;j<N;j++){

int session = 20000;

int busy, failure, denial, busy\_number, failure\_number, denial\_number, generating, gen\_number, group\_time;

total\_count = busy = failure = denial = generating = busy\_number = failure\_number = denial\_number = gen\_number = 0;

total\_generating = total\_failure = total\_denial = total\_busy = total\_group = group\_time = 0;

int count1k = 1000;

int count4k = 4000;

int count20k = 20000;

lvs->timer->total\_time = 0;

lvs->status = lvs->statuses["working"];

for(int i=1;i<=18;i++)lvs->clients[i]->state = "working";

while(total\_count <= (session)){

if(count20k >= 20000){

generating = rand() % 2;

if(generating == 1){

total\_generating += 1;

lvs->status = lvs->statuses["generating"];

gen\_number = rand() % 18;

lvs->clients[gen\_number+1]->state = "generator";

}

count20k = 0;

}

if(count4k >= 4000){

denial = rand() % 2;

if(denial == 1){

total\_denial += 1;

do{denial\_number = rand() % 18;}

while(denial\_number == gen\_number);

lvs->clients[denial\_number+1]->state = "denial";

}

else denial\_number = -1;

count4k = 0;

}

if(count1k >= 1000){

group\_time = lvs->timer->getTime() - group\_time;

total\_group += 1;

failure = rand() % 2;

busy = rand() % 2;

if(failure == 1){

total\_failure += 1;

do{failure\_number = rand() % 18;}

while(failure\_number == gen\_number && failure\_number == denial\_number);

lvs->clients[failure\_number+1]->state = "failure";

}

else failure\_number = -1;

if(busy == 1){

total\_busy += 1;

do{busy\_number = rand() % 18;}

while(busy\_number == gen\_number && busy\_number == denial\_number && busy\_number == failure\_number);

lvs->clients[busy\_number+1]->state = "busy";

}

else busy\_number = -1;

logger\_str = QString::number(total\_group) + " Группа Сообщений: " + QString::number(failure) + " " + QString::number(denial) + " " + QString::number(busy) + " " + QString::number(generating) + " " + QString::number(group\_time);

logger\_test->addToFile(logger\_str);

qDebug() << total\_group << "Группа Сообщений:" << failure << denial << busy << generating << group\_time << endl;

//qDebug() << "Сообщений= " << total\_count << endl;

//qDebug() << "Генерация:" << generating << ":" << gen\_number+1 << endl;

//qDebug() << "Отказ:" << denial << ":" << denial\_number+1 << endl;

//qDebug() << "Сбой:" << failure << ":" << failure\_number+1 << endl;

//qDebug() << "Занят:" << busy << ":" << busy\_number+1 << endl;

//qDebug() << "Время:" << lvs->timer->getTime() << endl;

//qDebug() << "--------------------------------" << endl;

count1k = 0;

group\_time = lvs->timer->getTime();

}

lvs->testRound();

total\_count += lvs->msg\_amount;

count1k += lvs->msg\_amount;

count4k += lvs->msg\_amount;

count20k += lvs->msg\_amount;

}

}

total\_time += lvs->timer->getTime();

logger\_str = "Всего: " + QString::number(total\_failure) + " " + QString::number(total\_denial) + " " + QString::number(total\_busy) + " " + QString::number(total\_generating) + " " + QString::number(lvs->timer->getTime());

logger\_test->addToFile(logger\_str);

logger\_str = "Общее время: " + QString::number(total\_time);

logger\_test->addToFile(logger\_str);

logger\_test->addToFile("-------------------------------------------");

qDebug() << "Всего: " << total\_count << total\_failure << total\_denial << total\_busy << total\_generating << lvs->timer->getTime() << endl;

qDebug() << "Общее время:" << total\_time << endl;

qDebug() << "-------------------------------------------" << endl;

}

**Файл “message.h” (Модуль сообщений)**

**Автор: Стерпу Е.К.**

#ifndef MESSAGE\_H

#define MESSAGE\_H

#include <QApplication>

#include <map>

using namespace std;

class Message

{

public:

QString state;

map<QString,QString> types;

map<int,QString> modes;

QString type;

int address\_from;

int address\_to;

Message();

QString encodeMessage(int address\_from, int address\_to, QString mode, QString command);

};

#endif // MESSAGE\_H

**Файл “message.cpp”**

**Автор: Стерпу Е.К.**

#include "message.h"

Message::Message()

{

types["short-command"] = "command";

types["short-give\_response"] = "give\_response";

types["short-block"] = "block";

types["short-unblock"] = "unblock";

types["long-give\_info"] = "give\_info";

modes[0] = "00000"; modes[1] = "00001"; modes[2] = "00010"; modes[3] = "00011"; modes[4] = "00100"; modes[5] = "00101";

modes[6] = "00110"; modes[7] = "00111"; modes[8] = "01000"; modes[9] = "01001"; modes[10] = "01010";

modes[11] = "01011"; modes[12] = "01100"; modes[13] = "01101"; modes[14] = "01110"; modes[15] = "01111";

modes[16] = "10000"; modes[17] = "10001"; modes[18] = "10010";

}

QString Message::encodeMessage(int address\_from, int address\_to, QString mode\_name, QString command)

{

QString message;

int lastbit = 1;

QString synchr = "111";

map<QString,QString>::iterator iter = types.find(mode\_name);

int mode = (distance(types.begin(), iter)) - 1;

QStringList command\_words = command.split(' ');

int words\_amount = command\_words.count();

for(int i=1;i<19;i++){

int bit = (message[i] == '1')? 1 : 0;

lastbit = lastbit ^ bit;

}

message = message + synchr + modes[address\_to] + "0" + modes[mode] + modes[words\_amount] + QString::number(lastbit);

return message;

}