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;
}
```