МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ТВЕРСКОЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

**(ФГБОУ ВО «ТГУ»)**

Математический факультет

Кафедра Компьютерной безопасности и

математических методов управления

**ОТЧЕТ**

по производственнойпрактике

студента IV курса М-44 группы

Акатьева Александра Андреевича

Специальность 10.05.01 Компьютерная безопасность

Специализация «Математические методы защиты информации»

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

*Подпись, расшифровка, ученая степень, звание*

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

*Подпись, расшифровка подписи*

Руководитель ООП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.\_\_.20\_\_

*Подпись, расшифровка подписи, ученая степень, звание*

Руководитель практики от предприятия: Андреев Виталий Юрьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

*Подпись, расшифровка подписи, ученая степень, звание*

Тверь 2017

**Содержание**

[Введение 3](#_Toc487008946)

[Задание 6](#_Toc487008947)

[Отчет по выполнению задания 7](#_Toc487008948)

[Приложение 11](#_Toc487008949)

[Заключение 28](#_Toc487008950)

[Список используемых источников информации: 29](#_Toc487008951)

# Введение

Практика проходилась в НИИ «Центрпрограммсистем» в течении 5 недель.

ЗАО НИИ ЦПС – крупная научная организация, насчитывающая более 550 специалистов, в их числе более 35 кандидатов и докторов наук, более 60 аспирантов и студентов старших курсов.

* более 10 000 кв. м. производственных площадей;
* свыше 550 единиц вычислительной техники;
* лицензионные программные средства;
* корпоративная сеть с выходом в Интернет;
* специальные приборы для научных исследований;
* опытное производство аппаратно-программных средств;
* специализированные стенды для апробации проектных решений;
* служба нормоконтроля с архивом нормативно-технической документации;
* цифровая телефонная станция.

В НИИ «Центрпрограммсистем» на сегодняшний день разрабатываются:

* автоматизированные системы управления (АСУ) для решения задач управления;
* автоматизированные информационные системы различного назначения;
* программно-технические комплексы;
* системы защиты информации;
* компьютерные тренажеры и системы обучения;
* распределенные системы хранения и обработки информации.

Научно-Исследовательский институт «Центрпрограммсистем» — ведущая организация Тверского региона в области разработки программного обеспечения. Созданный в 1971 году, НИИ ЦПС является организацией, для которой освоение и внедрение новых информационных технологий стало основным содержанием деятельности. Уже через пять лет после появления, ЦПС стал одной из ведущих организаций страны в области информатики и подготовки программистов. Серьёзный научный задел активно пополнялся прикладными разработками. Среди них — проекты по автоматизации, исследовательскому проектированию, математическому моделированию и управлению базами данных.

Основа потенциала НИИ «Центрпрограммсистем» и залог качества выполнения работ — это более 600 сотрудников, 45 из которых имеют степень кандидатов и докторов наук. Сохранив уникальный коллектив и взаимоотношения, стимулирующие творческий рост, Институт из года в год формирует грамотных, востребованных специалистов. За годы своего существования НИИ «Центрпрограммсистем» подготовил более 20000 специалистов в области программирования и в смежных областях.

Укомплектованный современной техникой, активно используя развитые коммуникационные возможности предприятия, коллектив способен решать самые сложные задачи. Сейчас НИИ «Центрпрограммсистем» — один из крупнейших в России разработчиков информационных систем и программно-аппаратных средств.

Сферой деятельности организации являются информационные технологии, поэтому информационные взаимосвязи между структурными подразделениями построены с использованием принципов обработки и передачи информации с помощью ЛВС и корпоративной интранет-сети, в первую очередь, в применении к службе качества, нормоконтроля и ведения электронного архива документации. Доступ к этой базе имеют все подразделения НИИ ЦПС.

**Основные направления деятельности:**

* Прикладные научные исследования в области информационных технологий;
* Автоматизация научных исследований;
* Создание автоматизированных систем управления, автоматизированных информационных систем, информационных систем автоматизации организационного управления жизненным циклом сложных технических комплексов;
* Разработка и производство систем защиты информации от несанкционированного доступа;
* Создание компьютерных тренажеров, тренажерных комплексов и автоматизированных систем обучения;
* Моделирование сложных технических объектов;
* Научные исследования и разработки в области распределенных систем хранения и обработки информации;
* Специальные проверки, аттестация оборудования и объектов информатизации;
* Производство компьютеров в защищенном исполнении;

**Срок прохождения практики:** 12 июня – 9 июля

# Задание

Разработка учебной статистической программы «Netmanager»

**Цель:** Оценить объем и содержимое внутреннего сетевого трафика учебного тренажера апк рок (между сервером и клиентами).

**Объекты:**

* Сервер: АПК РОК - аппаратно-программный комплекс руководства обучением и комплексирования. АПК РОК обеспечивает моделирование единой тактической обстановки, средства подготовки задания на тренировку, средства управления проведением тренировки и средства разбора тренировки.
* Клиент: Программа «Netmanager»

**Требования:**

1. Регистрация в качестве клиента
2. Сбор статистических данных в виде сетевых пакетов
3. Просмотр принятых пакетов
4. Отображение статистики
   1. По получателям
   2. По отдельным узлам
   3. Отображать
   4. Сортировка по отображаемым полям

Программа была реализована с помощью среды разработки Qt на базе языка С++.

# Отчет по выполнению задания

Прежде чем реализовать программу, нужно понять, каким образом происходит соединение между сервером и клиентом.

По нажатию кнопки, клиент подсоединяется к серверу (в нашем случае АПК РОК), зная его ip:port, который мы задали заранее. Программа проверяет правильность пары ip:port и выдает сообщение:

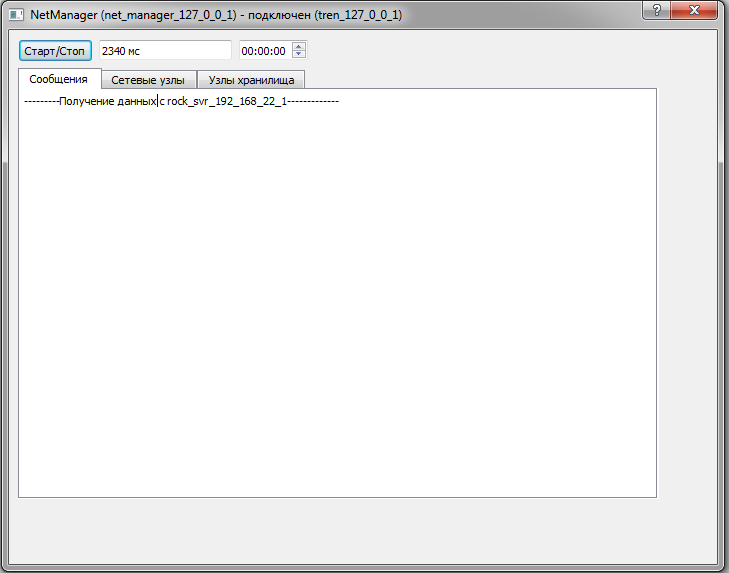


Рис 1. Окно «Сообщения»

Если же клиент не подсоединился к серверу (сервер не включен/есть проблемы на линии связи), то программа выдаст соответствующее сообщение:

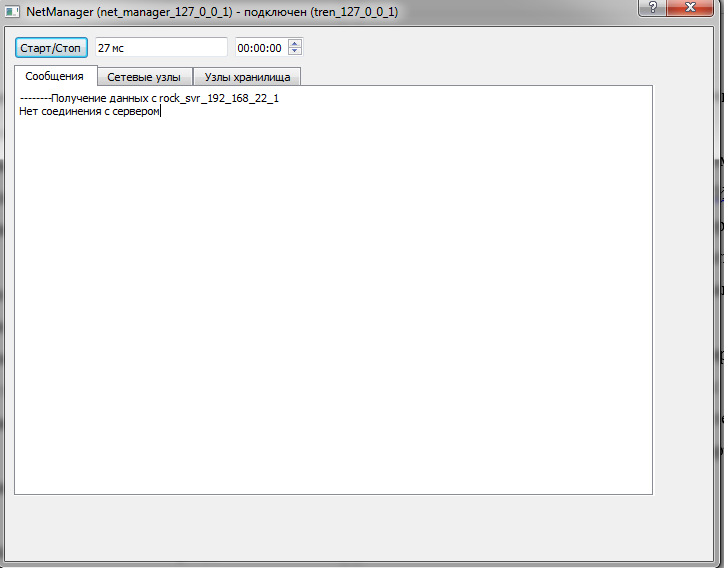


Рис 2. Окно «Сообщения» в случае проблем с сервером

Как только мы подсоединились, начинается передача сетевых пакетов в нашу программу. Нам нужно перехватить их и обработать, а результат выдать в окне «Сетевые узлы» (рис 3)

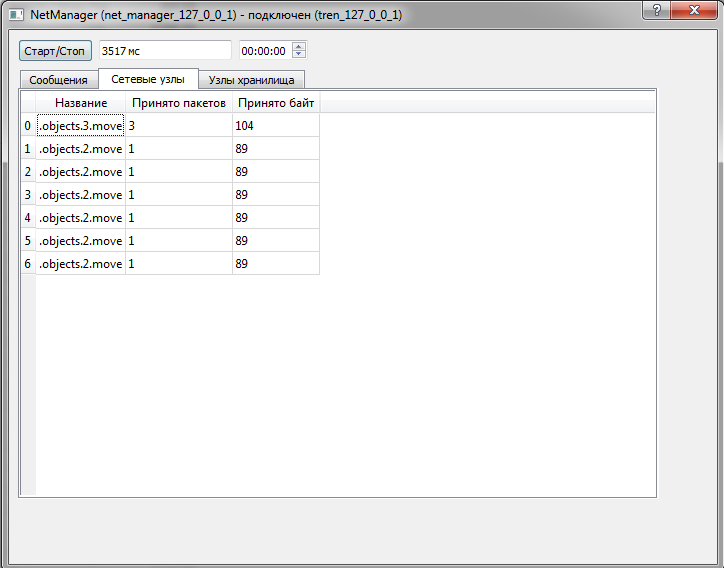


Рис 3. Окно «Сетевые узлы»

У нас есть три пункта в окне: «Название», «Принято пакетов», «Принято байт». Пункты нужно заполнить, исходя из полученных данных.

В каждом пакете присутствует имя объекта и его основные характеристики. Это имя и печатается в пункте «Название»

В соответствии с именем, считается количество принятых байт и пакетов. Эти данные помещаются в пункты «Принято байт» и «Принято пакетов»

Последнее окно «Узлы хранилища» нужно для того случая, если наша программа подсоединяется к нескольким серверам. В нашем случае там будет лишь один узел – наш сервер АПК РОК.

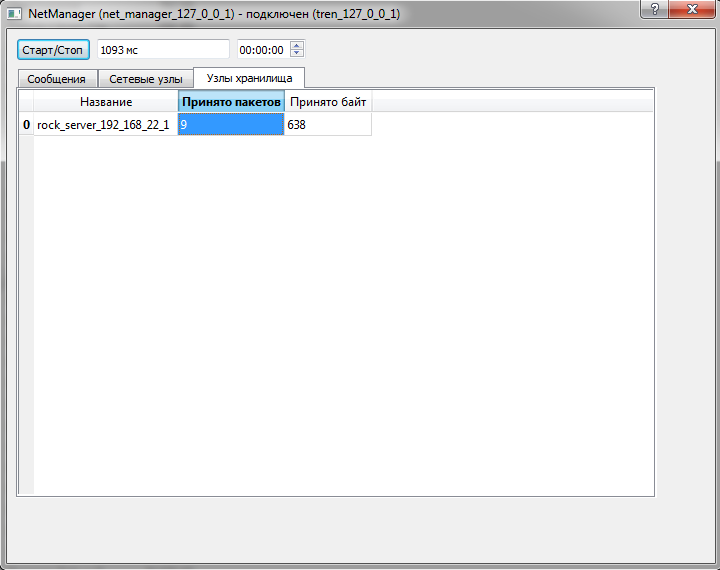


Рис 4. Окно «Узлы Хранилища»

Система разработана для платформы Windows.

Средство разработки - Qt Creator 4.2.1. Эта среда позволяет быстро и эффективно нарисовать ui-дизайн наших окон и таблиц, тем самым сэкономив нам время на написание кода. Большим преимуществом программы является кроссплатформенность и большой выбор языков программирования, встроенных в среду, таких как С++ и другие.

# Приложение

*Приложение 1. Код программы.*

1. client.h
2. #ifndef \_ROCK\_ROCKCORE\_ROCK\_CLIENT\_H
3. #define \_ROCK\_ROCKCORE\_ROCK\_CLIENT\_H
4. #include "rockapi.h"
5. namespace rock
6. {
7. class RockClientImpl;
8. class Storage;
9. struct INetMonitor;
10. class ROCKCORE\_API RockClient: public RockApi
11. {
12. public:
13. RockClient(const std::string &id);
14. ~RockClient();
15. //присоединиться к серверу
16. bool connect(const std::string &ip, unsigned short port, unsigned flags);
17. //прием-оправка сетевых сообщений
18. void run(int dt);
19. //---------------------------------------------------------------------------------
20. //свойства
21. //---------------------------------------------------------------------------------
22. //уникальный идентификатор этого узла
23. virtual std::string *id*() const override;
24. //уникальный идентификатор узла сервера
25. virtual std::string *serverId*() const override;
26. //признак установки связи с сервером
27. virtual bool *isConnected*() const override;
28. //ip,port соединения
29. virtual std::string *ip*() const override;
30. virtual unsigned short *port*() const override;
31. //---------------------------------------------------------------------------------
32. //Сетевой транпорт
33. //---------------------------------------------------------------------------------
34. //отправить сообщение серверу
35. virtual void *sendToServer*(Value &&m) override;
36. //отправить сообщение клиенту
37. virtual void *sendToClient*(const std::string &client\_id, Value &&m) override;
38. //---------------------------------------------------------------------------------
39. //Доступ к данным - локальное хранилище клиента
40. //---------------------------------------------------------------------------------
41. virtual const Value &*get*(const ValuePath &path) override;
42. //---------------------------------------------------------------------------------
43. //Исходящие сообщения серверу
44. //---------------------------------------------------------------------------------
45. //подписаться на получение данных
46. virtual int *subscribe*(const ValueProjection &proj) override;
47. //устанановить значение
48. virtual int *set*(const ValuePath &path, Value &&v) override;
49. virtual int *set*(const ValuePath &path, const Value &v) override;
50. //обновить значение
51. virtual int *update*(const ValuePath &path, Value &&v) override;
52. virtual int *update*(const ValuePath &path, const Value &v) override;
53. //добавить узел с уникальным именем
54. virtual int *post*(const ValuePath &path, Value &&v) override;
55. virtual int *post*(const ValuePath &path, const Value &v) override;
56. //удалить узел
57. virtual int *remove*(const ValuePath &path) override;
58. //обновить метаданные узла
59. virtual int *updateMetadata*(const ValuePath &path, Value &&v) override;
60. virtual int *updateMetadata*(const ValuePath &path, const Value &v) override;
61. //событие
62. virtual int *event*(const ValuePath &path, const std::string &code, Value &&params) override;
63. virtual int *event*(const ValuePath &path, const std::string &code, const Value &params) override;
64. //команда
65. virtual int *cmd*(const ValuePath &path, const std::string &code, Value &&params) override;
66. virtual int *cmd*(const ValuePath &path, const std::string &code, const Value &params) override;
67. //отклик на запрос
68. virtual void *reply*(const std::string &sender, int request\_id, const Value &params = Value()) override;
69. //отклик - по требованию флагов
70. virtual void *demandFlag*(int request\_id, Value &&data) override;
71. //---------------------------------------------------------------------------------
72. //Обработка входящих сообщений - сигналы
73. //---------------------------------------------------------------------------------
74. //отклик на запрос
75. virtual Signal<void(int, const Value &)> &*onReply*() override;
76. //запрос
77. virtual Signal<void(int, const std::string &, const Value &)> &*onDemandFlag*() override;
78. //событие
79. virtual Signal<void(const ValuePath &, const std::string &, const Value &)> &*onEvent*() override;
80. //команда
81. virtual Signal<void(int, const std::string &, const ValuePath &, const std::string &, const Value &)> &*onCmd*() override;
82. //пользовательское сообщение
83. virtual Signal<void(const std::string &, const Value &)> &*onUserMessage*() override;
84. //присоединение к серверу
85. Signal<void(const std::string &)> &onServerAccept();
86. //мониторинг
87. void setNetMonitor(INetMonitor \*p);
88. private:
89. RockClientImpl \*p\_;
90. };
91. } //namespace rock

#endif // \_ROCK\_ROCKCORE\_ROCK\_CLIENT\_H

1. Client.cpp
2. #include "client.h"
3. #include "detail/client\_impl.h"
4. namespace rock
5. {
6. RockClient::RockClient(const std::string &id) : p\_(new RockClientImpl(id))
7. {
8. }
9. RockClient::~RockClient()
10. {
11. delete p\_;
12. }
13. std::string RockClient::*id*() const
14. {
15. return p\_->id();
16. }
17. std::string RockClient::*ip*() const
18. {
19. return p\_->ip();
20. }
21. unsigned short RockClient::*port*() const
22. {
23. return p\_->port();
24. }
25. std::string RockClient::*serverId*() const
26. {
27. return p\_->serverId();
28. }
29. bool RockClient::*isConnected*() const
30. {
31. return p\_->isConnected();
32. }
33. bool RockClient::connect(const std::string &ip, unsigned short port, unsigned flags)
34. {
35. return p\_->connect(ip, port, flags);
36. }
37. void RockClient::*sendToServer*(Value &&m)
38. {
39. p\_->*sendToServer*(std::move(m));
40. }
41. void RockClient::*sendToClient*(const std::string &client\_id, Value &&m)
42. {
43. p\_->*sendToClient*(client\_id, std::move(m));
44. }
45. void RockClient::run(int dt)
46. {
47. p\_->*run*(dt);
48. }
49. const Value &RockClient::*get*(const ValuePath &path)
50. {
51. return p\_->storage()->get(path);
52. }
53. int RockClient::*subscribe*(const ValueProjection &proj)
54. {
55. return p\_->subscribe(proj);
56. }
57. int RockClient::*set*(const ValuePath &path, Value &&v)
58. {
59. return p\_->set(path, std::move(v));
60. }
61. int RockClient::*set*(const ValuePath &path, const Value &v)
62. {
63. return p\_->set(path, Value(v));
64. }
65. int RockClient::*update*(const ValuePath &path, Value &&v)
66. {
67. return p\_->update(path, std::move(v));
68. }
69. int RockClient::*update*(const ValuePath &path, const Value &v)
70. {
71. return p\_->update(path, Value(v));
72. }
73. int RockClient::*post*(const ValuePath &path, Value &&v)
74. {
75. return p\_->post(path, std::move(v));
76. }
77. int RockClient::*post*(const ValuePath &path, const Value &v)
78. {
79. return p\_->post(path, Value(v));
80. }
81. int RockClient::*remove*(const ValuePath &path)
82. {
83. return p\_->remove(path);
84. }
85. int RockClient::*updateMetadata*(const ValuePath &path, Value &&v)
86. {
87. return p\_->updateMetadata(path, std::move(v));
88. }
89. int RockClient::*updateMetadata*(const ValuePath &path, const Value &v)
90. {
91. return p\_->updateMetadata(path, Value(v));
92. }
93. int RockClient::*event*(const ValuePath &path, const std::string &code, Value &&params)
94. {
95. return p\_->event(path, code, std::move(params));
96. }
97. int RockClient::*event*(const ValuePath &path, const std::string &code, const Value &params)
98. {
99. return p\_->event(path, code, Value(params));
100. }
101. int RockClient::*cmd*(const ValuePath &path, const std::string &code, Value &&params)
102. {
103. return p\_->cmd(path, code, std::move(params));
104. }
105. int RockClient::*cmd*(const ValuePath &path, const std::string &code, const Value &params)
106. {
107. return p\_->cmd(path, code, Value(params));
108. }
109. void RockClient::*reply*(const std::string &sender, int request\_id, const Value &params /\*= Value( )\*/)
110. {
111. p\_->reply(sender, request\_id, params);
112. }
113. void RockClient::*demandFlag*(int request\_id, Value &&data)
114. {
115. p\_->demandFlag(request\_id, std::move(data));
116. }
117. Signal<void(const std::string &)> &RockClient::onServerAccept()
118. {
119. return p\_->onConnect();
120. }
121. void RockClient::setNetMonitor(INetMonitor \*p)
122. {
123. p\_->setNetMonitor(p);
124. }
125. Signal<void(int, const Value &)> &RockClient::*onReply*()
126. {
127. return ((RockNetNode \*)p\_)->onReply();
128. }
129. Signal<void(int, const std::string &, const Value &)> &RockClient::*onDemandFlag*()
130. {
131. return p\_->onDemandFlag();
132. }
133. Signal<void(const ValuePath &, const std::string &, const Value &)> &RockClient::*onEvent*()
134. {
135. return p\_->onEvent();
136. }
137. Signal<void(int, const std::string &, const ValuePath &, const std::string &, const Value &)> &RockClient::*onCmd*()
138. {
139. return p\_->onCmd();
140. }
141. Signal<void(const std::string &, const Value &)> &RockClient::*onUserMessage*()
142. {
143. return p\_->onUserMessage();
144. }
145. } //namespace rock
146. Main.cpp
147. #include "mainwindow.h"
148. #include <QApplication>
149. #include "rockcore/storage/storage.h"
150. #include "rockcore/rock/server.h"
151. #include "rockcore/rock/client.h"
152. #include "rockcore/logging/logging.h"
153. #include "rockcore/storage/detail/value\_data.h"
154. #include "rockcore/timer.h"
155. using namespace rock;
156. int main(int argc, char \*argv[])
157. {
158. QApplication app(argc, argv);
159. QCommandLineParser parser;
160. QString id = parser.value("id");
161. if (id.isEmpty())
162. id = "simple\_client";
163. rock::RockClient client(id.toStdString());
164. client.connect( "127.0.0.1", 54301, 0 );
165. client.*subscribe*( true );
166. MainWindow \*w = new MainWindow(&client);
167. w->show();
168. int r = app.exec();
169. return r;
170. }
171. Dataclient.h
172. #ifndef DATACLIENT\_H
173. #define DATACLIENT\_H
174. #include "rockcore/rock/client.h"
175. #include "rockcore/rock/netmonitor.h"
176. namespace rock{
177. class DataClient: rock::INetMonitor
178. {
179. public:
180. //конструктор по умолчанию
181. DataClient();
183. //Соединение
184. void connect(rock::RockClient \*rclient);

187. int getNumberReceiveBytes();
188. int getNumberReceivePackage();
189. std::string getName();
190. //функция очистки
191. void clear();
192. virtual void *onReceive*(const std::string &sender, const Value &m);
193. virtual void *onSend*(const std::string &receiver, const Value &m);
195. private:
196. std::string NamePackage;
197. int NumberReceivePackage;
198. int NumberReceiveByte;
199. };
200. }
201. #endif // DATACLIENT\_H
202. Dataclient.cpp
203. #include "dataclient.h"
204. #include "rockcore/storage/value\_utils.h"
205. namespace rock {
206. DataClient::DataClient(){
207. NamePackage = "";
208. NumberReceiveByte = 0;
209. NumberReceivePackage = 0;
210. }
211. void DataClient::connect(RockClient \*rclient)
212. {
213. rclient->setNetMonitor(this);
214. }
215. int DataClient::getNumberReceiveBytes()
216. {
217. return NumberReceiveByte;
218. }
219. int DataClient::getNumberReceivePackage()
220. {
221. return NumberReceivePackage;
222. }
223. std::string DataClient::getName()
224. {
225. return NamePackage;
226. }
227. void DataClient::clear()
228. {
229. NamePackage = "";
230. NumberReceiveByte = 0;
231. NumberReceivePackage = 0;
232. }
233. void DataClient::*onReceive*(const std::string &sender, const Value &m)
234. {
235. //количество пакетов
236. if(NamePackage == m["path"].toString()){
237. ++NumberReceivePackage;
238. auto s=WriteJson(m);
239. NumberReceiveByte += s.size();
240. }
242. //Имя пакета
243. NamePackage = m["path"].toString();
244. }
245. void DataClient::*onSend*(const std::string &/\*receiver\*/, const Value &/\*m\*/){
246. }
247. }
248. Mainwindow.h
249. #ifndef MAINWINDOW\_H
250. #define MAINWINDOW\_H
251. #include <QMainWindow>
252. #include "rockcore/rock/client.h"
253. #include "ui\_mainwindow.h"
254. #include <QDialog>
255. #include <QTime>
256. #include "dataclient.h"
257. #include "rocklink/rocklink.h"
258. #include <QTimer>
259. #include "QCommandLineParser"
260. #include "QStandardItemModel"
261. #include "QStandardItem"
262. #include <QString>
263. namespace Ui {
264. class MainWindow;
265. }
266. namespace rock{
267. class MainWindow : public QDialog
268. {
269. Q\_OBJECT
270. public:
271. explicit MainWindow(rock::RockClient \*rclient, QWidget \*parent = 0);
272. public slots:
273. void Start();
274. void SlotTimer();
275. private:
276. DataClient data\_; //наши данные
278. //Объект ui и его компоненты
279. QStandardItemModel \*model = new QStandardItemModel;
280. QStandardItemModel \*model1 = new QStandardItemModel;
281. QStandardItem \*item;
282. QStandardItem \*item1;
283. QScopedPointer<Ui::MainWindow> ui;
284. QStringList horizontalHeader;
285. QStringList verticalHeader;
286. QStringList horizontalHeader1;
287. QStringList verticalHeader1;
289. //Клиент
290. rock::INetMonitor \*p;
291. rock::RockClient \*client\_;
293. int a = 0;
294. QTimer \*timer\_;
295. int CountRow = 1;
297. };
298. }
299. #endif // MAINWINDOW\_H
300. Mainwindow.cpp
301. #include "mainwindow.h"
302. #include "ui\_mainwindow.h"
303. #include "rockcore/rock/netmonitor.h"
304. #include "rockcore/logging/logging.h"
305. #include "rockcore/storage/storage.h"
306. #include "rocklink/node.h"
307. #include "rockcore/timer.h"
308. namespace Ui {
309. class MainWindow;
310. }
311. using namespace rock;
312. MainWindow::MainWindow(rock::RockClient \*rclient, QWidget \*parent) :QDialog(parent), ui(new Ui::MainWindow), client\_(rclient)
313. {
314. ui->setupUi(this);
315. timer\_ = new QTimer(this);
316. setWindowTitle("NetManager (net\_manager\_127\_0\_0\_1) - подключен (tren\_127\_0\_0\_1)");
317. connect(timer\_, SIGNAL(timeout()), this, SLOT(SlotTimer()));
318. connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(Start()));
319. timer\_->start(0);
320. model->clear();
322. //Окно "Сетевые узлы
323. //Заголовки столбцов
324. horizontalHeader.append("Название");
325. horizontalHeader.append("Принято пакетов");
326. horizontalHeader.append("Принято байт");
328. //Заголовки строк
329. verticalHeader.append("0");
330. item = new QStandardItem("");
331. model->setItem(0, 0, item);
333. //Узлы хранилища.
334. //Заголовки столбцов
335. horizontalHeader1.append("Название");
336. horizontalHeader1.append("Принято пакетов");
337. horizontalHeader1.append("Принято байт");
338. //Заголовки строк
339. verticalHeader1.append("0");


343. }
344. void MainWindow::SlotTimer()
345. {
346. //Таймер в мс
347. a++;
348. ui->lineEdit->setText(QString::number(a) + " мс");
349. client\_->run(50);
350. if (client\_->*isConnected*())
351. {
352. data\_.connect(client\_);
353. //Просматриваем таблицу в цикле
354. for(int i = 0; i < CountRow; i++){
356. QModelIndex index1 = model->*index*(i,0);
357. //Если такое название было, добавляем туда информацию
358. if(QString::fromStdString(data\_.getName()) == model->*data*(index1) || model->*data*(index1) == QString("")){
359. item = new QStandardItem(QString::fromStdString(data\_.getName()));
360. model->setItem(i, 0, item);
361. item = new QStandardItem(QString::number(data\_.getNumberReceivePackage()));
362. model->setItem(i, 1, item);
363. item = new QStandardItem(QString::number(data\_.getNumberReceiveBytes()));
364. model->setItem(i, 2, item);
365. break;
366. }
367. else{//Если нет - создаем новое поле
368. QString Row("");
369. Row += QString("%1").arg(QString::number(CountRow));
370. verticalHeader.append(Row);
371. item = new QStandardItem(QString::fromStdString(data\_.getName()));
372. model->setItem(CountRow, 0, item);
373. item = new QStandardItem(QString::number(data\_.getNumberReceivePackage()));
374. model->setItem(CountRow, 1, item);
375. item = new QStandardItem(QString::number(data\_.getNumberReceiveBytes()));
376. model->setItem(CountRow, 2, item);
377. CountRow+=1;
378. break;
379. }
380. }
382. //Заносим данные для третьего окна
383. item1 = new QStandardItem(QString::fromStdString("rock\_server\_192\_168\_22\_1"));
384. model1->setItem(0, 0, item);
386. item1 = new QStandardItem(QString::number(data\_.getNumberReceivePackage()));
387. model1->setItem(0, 1, item);
389. item1 = new QStandardItem(QString::number(data\_.getNumberReceiveBytes()));
390. model1->setItem(0, 2, item);

393. //Передаем на форму
394. model->setHorizontalHeaderLabels(horizontalHeader);
395. model->setVerticalHeaderLabels(verticalHeader);
397. model1->setHorizontalHeaderLabels(horizontalHeader1);
398. model1->setVerticalHeaderLabels(verticalHeader1);
400. ui->tableView->*setModel*(model);
401. ui->tableView->resizeRowsToContents();
402. ui->tableView->resizeColumnsToContents();
404. ui->tableView\_2->*setModel*(model1);
405. ui->tableView\_2->resizeRowsToContents();
406. ui->tableView\_2->resizeColumnsToContents();
407. }
408. else
409. ui->textEdit->setText("Нет связи с сервером...");
410. }
411. void MainWindow::Start(){
412. if(timer\_->isActive()){
413. timer\_->stop();
414. a = 0;
415. }
416. Else
417. {
418. timer\_->start(0);
419. CountRow = 1;
420. }
421. }

*Приложение 2.*

**Требования техники безопасности**

Утверждена

Приказом

Министерства

Российской Федерации

по связи и информатизации

от 2 июля 2001 г. N 162

Согласовано

письмом

Центрального комитета

Общероссийского профсоюза

работников связи

Российской Федерации

от 20 апреля 2001 г. N 4-380

Дата введения - 1 июля 2001 года

**Типовая инструкция по охране труда при работе на персональном компьютере тои Р-45-084-01**

**1. Общие требования безопасности**

1.1. К работе на персональном компьютере допускаются лица, прошедшие обучение безопасным методам труда, вводный инструктаж, первичный инструктаж на рабочем месте.

1.2. Работник обязан:

1.2.1. Выполнять только ту работу, которая определена его должностной инструкцией.

1.2.2. Содержать в чистоте рабочее место.

1.2.3. Соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности (Приложение 1).

1.2.4. Соблюдать меры пожарной безопасности.

1.3. Рабочие места с компьютерами должны размещаться таким образом, чтобы расстояние от экрана одного видеомонитора до тыла другого было не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов - не менее 1,2 м.

1.5. Рабочие места с персональными компьютерами по отношению к световым проемам должны располагаться так, чтобы естественный свет падал сбоку, преимущественно слева.

1.6. Оконные проемы в помещениях, где используются персональные компьютеры, должны быть оборудованы регулируемыми устройствами типа: жалюзи, занавесей, внешних козырьков и др.

1.7. За невыполнение данной Инструкции виновные привлекаются к ответственности согласно правилам внутреннего трудового распорядка или взысканиям, определенным Кодексом законов о труде Российской Федерации.

**2. Требования безопасности перед началом работы**

2.1. Подготовить рабочее место.

2.2. Отрегулировать освещение на рабочем месте, убедиться в отсутствии бликов на экране.

2.3. Проверить правильность подключения оборудования к электросети.

2.4. Проверить исправность проводов питания и отсутствие оголенных участков проводов.

2.5. Убедиться в наличии заземления системного блока, монитора и защитного экрана.

2.6. Протереть антистатической салфеткой поверхность экрана монитора и защитного экрана.

2.7. Проверить правильность установки стола, стула, подставки для ног, пюпитра, угла наклона экрана, положение клавиатуры, положение "мыши" на специальном коврике, при необходимости произвести регулировку рабочего стола и кресла, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительных напряжений тела.

**3. Требования безопасности во время работы**

3.1. Работнику при работе на ПК запрещается:

- прикасаться к задней панели системного блока (процессора) при включенном питании;

- переключать разъемы интерфейсных кабелей периферийных устройств при включенном питании;

- допускать попадание влаги на поверхность системного блока (процессора), монитора, рабочую поверхность клавиатуры, дисководов, принтеров и других устройств;

- производить самостоятельное вскрытие и ремонт оборудования;

- работать на компьютере при снятых кожухах;

- отключать оборудование от электросети и выдергивать электровилку, держась за шнур.

3.2. Продолжительность непрерывной работы с компьютером без регламентированного перерыва не должна превышать 2-х часов.

3.3. Во время регламентированных перерывов с целью снижения нервно - эмоционального напряжения, утомления зрительного анализатора, устранения влияния гиподинамии и гипокинезии, предотвращения развития познотонического утомления выполнять комплексы упражнений.

**4. Требования безопасности по окончании работы**

5.1. Отключить питание компьютера.

5.2. Привести в порядок рабочее место.

5.3. Выполнить упражнения для глаз и пальцев рук на расслабление.

# Заключение

В ходе производственной практики была разработана учебная программа для предприятия согласно заданию. Были созданы основные объекты программы, нарисованы окна. «Netmanager»» грамотно обрабатывает сетевые данные и собирает статистику.

Для соединения программы по сети был разработан класс «Client», содержащий основные функции по принятию сетевых сообщений, их обработке и хранению в памяти.

Для обработки самих данных был создан класс «Dataclient», хранящий получаемые данные в удобном виде.

Класс «Mainwindow» содержит команды для графического представления и вывода данных на экран.

В ходе практики получен опыт по программированию на языке C++, а также использованию среды разработки Qt.

**Список используемых источников информации:**

1. Мейерс, Скотт. **Эффективный и современный С++: 42 рекомендации по использованию С++ 1 1 и С++14.** : Пер. с англ. - М. : ООО "ИЛ. Вильяме", 2016. - 304 с. ил.
2. **Алан Купер об интерфейсе. Основы проектирования взаимодействия**. – Пер. с англ. – СПб.: Символ'Плюс, 2009. – 688 с., ил.
3. Таненбаум Э., Уэзеролл Д. Т18 **Компьютерные сети. 5-е изд**. — СПб.: Питер, 2012. — 960 с.