# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc389461530)

[1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ 5](#_Toc389461531)

[1.1 Анализ предметной области 5](#_Toc389461532)

[1.2 Выбор специального и общесистемного программного обеспечения 5](#_Toc389461533)

[2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ 6](#_Toc389461535)

[2.1 Пользовательский интерфейс 6](#_Toc389461536)

[2.2 Функциональное обеспечение 7](#_Toc389461537)

[2.3 Структурное обеспечение 7](#_Toc389461538)

[2.4 Алгоритмическое обеспечение 8](#_Toc389461539)

[2.5 Архитектурное обеспечение 8](#_Toc389461540)

[2.6 Информационные структуры 8](#_Toc389461541)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 10](#_Toc389461542)

[3.1 Детальная реализация функциональных частей программного обеспечения 10](#_Toc389461543)

[3.1.1 Реализация функциональной части 10](#_Toc389461544)

[3.1.2 Реализация пользовательского интерфейса 12](#_Toc389461545)

[3.1.3 Реализация справочного модуля 17](#_Toc389461546)

[3.1.4 Реализация программы установки/удаления 18](#_Toc389461547)

[3.2 Сопроводительная документация программного обеспечения 18](#_Toc389461548)

[3.2.1 Минимальные аппаратные и системные требования 18](#_Toc389461549)

[3.2.2 Описание установки и запуска приложения 18](#_Toc389461550)

[3.2.3 Руководство пользователя 19](#_Toc389461551)

[3.3 Анализ программного обеспечения 19](#_Toc389461552)

[3.3.1 Анализ исходного кода приложения 19](#_Toc389461553)

[3.3.2 Анализ производительности приложения 20](#_Toc389461554)

[3.4 Тестирование программного обеспечения 21](#_Toc389461555)

[3.4.1 Разработка модульных тестов 21](#_Toc389461556)

[3.4.2 Разработка и результаты интегрированного тестирования 22](#_Toc389461557)

[ЗАКЛЮЧЕНИЕ 24](#_Toc389461558)

[СПИСОК ЛИТЕРАТУРЫ 25](#_Toc389461559)

[ПРИЛОЖЕНИЕ А 26](#_Toc389461560)

[ПРИЛОЖЕНИЕ Б 27](#_Toc389461562)

[ПРИЛОЖЕНИЕ В 28](#_Toc389461564)

[ПРИЛОЖЕНИЕ Г 30](#_Toc389461566)

[ПРИЛОЖЕНИЕ Д 31](#_Toc389461568)

# ВВЕДЕНИЕ

Сниффер – сетевой анализатор трафика, программа или программно-аппаратное устройство, предназначенное для перехвата и последующего анализа, либо только анализа сетевого трафика, предназначенного для других узлов. Такое приложение позволяет обнаружить [паразитный](http://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%B0%D1%80%D0%B0%D0%B7%D0%B8%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D1%80%D0%B0%D1%84%D0%B8%D0%BA&action=edit&redlink=1), [вирусный](http://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D1%80%D1%83%D1%81%D0%BD%D1%8B%D0%B9_%D1%82%D1%80%D0%B0%D1%84%D0%B8%D0%BA&action=edit&redlink=1) и закольцованный трафик, выявить в сети [вредоносное](http://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%B4%D0%BE%D0%BD%D0%BE%D1%81%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) и несанкционированное [ПО](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), перехватить любой незашифрованный пользовательский трафик, локализовать неисправность сети или ошибку конфигурации сетевых агентов. Реализация клиент-серверного аналога достаточно актуальна, так как слежение за сетевым трафиком широко используется в наше время, но клиент-серверные приложения отсутствуют. Целью данного курсового проекта является разработка такого приложения.

# 1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ

## 1.1 Анализ предметной области

Целью курсового проекта является создание клиент-серверного приложения «Система отслеживания трафика HTTP». Клиент должен собирать информацию о входящем и исходящем HTTP трафике; записывать полученную информацию в специальный файл; предоставлять программу в системном трее. Сервер должен видит всех подключенных клиентов и запрашивать накопленную информацию у любого из клиентов. Приложение должно предоставлять возможность просмотра HTTP трафика в интерактивном режиме с любого клиента, осуществлять поиск по значениям заголовков и просто по тексту, все настройки клиентов и сервера должны храниться в конфигурационных файлах и должна предоставляться возможность изменения настроек в конфигурационных файлах через пользовательский интерфейс.

HTTP — [протокол](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [прикладного уровня](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) передачи данных (изначально — в виде гипертекстовых документов в формате [HTML](http://ru.wikipedia.org/wiki/HTML)). Основой HTTP является [технология «клиент-сервер»](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), то есть предполагается существование потребителей ([клиентов](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9))), которые инициируют соединение и посылают [запрос](http://ru.wikipedia.org/w/index.php?title=HTTP-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81&action=edit&redlink=1), и поставщиков ([серверов](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5))), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом. HTTP в настоящее время повсеместно используется во [Всемирной паутине](http://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0) для получения информации с [веб-сайтов](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82). HTTP используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как [SOAP](http://ru.wikipedia.org/wiki/SOAP), [XML-RPC](http://ru.wikipedia.org/wiki/XML-RPC), [WebDAV](http://ru.wikipedia.org/wiki/WebDAV).

Такое приложение можно использовать как дома, так и в компаниях для отслеживания запросов сотрудников, например.

## 1.2 Выбор специального и общесистемного программного обеспечения

Для выполнения поставленных задач, нам требуется ряд различного программного обеспечения:

- Visual Studio 2012 для разработки приложения;

- Smart Install Maker для создания инсталлятора;

- Dr.Explain для разработки справочного модуля;

- Enterprise Architect и Software Ideas Modeler для построения диаграмм;

- Microsoft Word 2013 – для разработки и оформления пояснительной записки.

Платформа реализации – Windows 10.

# 2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ

## 2.1 Пользовательский интерфейс

При разработке пользовательского интерфейса за основу будет взят интерфейс HTTPNetworkSniffer. Окно сервера будет выглядеть следующим образом:

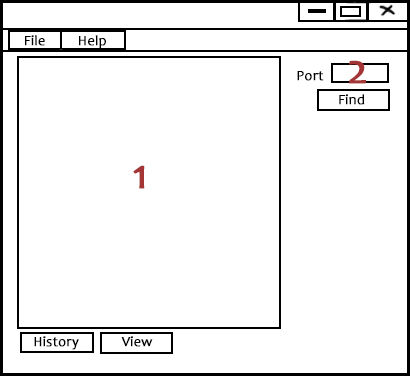


Рисунок 1 – главное окно сервера

1. Окно, в котором будет выводиться список найденных клиентов.
2. Поле, где будет задаваться порт, по которому будет производиться поиск клиентов.

Кнопка History будет открывать окно с историей HTTP трафика.

Кнопка View будет открывать окно трафика в интерактивном режиме.

После загрузки клиента, пользователь увидит следующие окно:

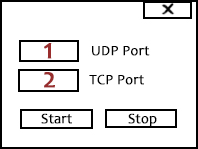


Рисунок 2 – главное окно клиента

1. Поле для ввода UDP порта.
2. Поле для ввода TCP порта.

Кнопки Start и Stop запускают и останавливают клиент.

Окно вывода информации о запросах и ответах будет выглядеть следующим образом:

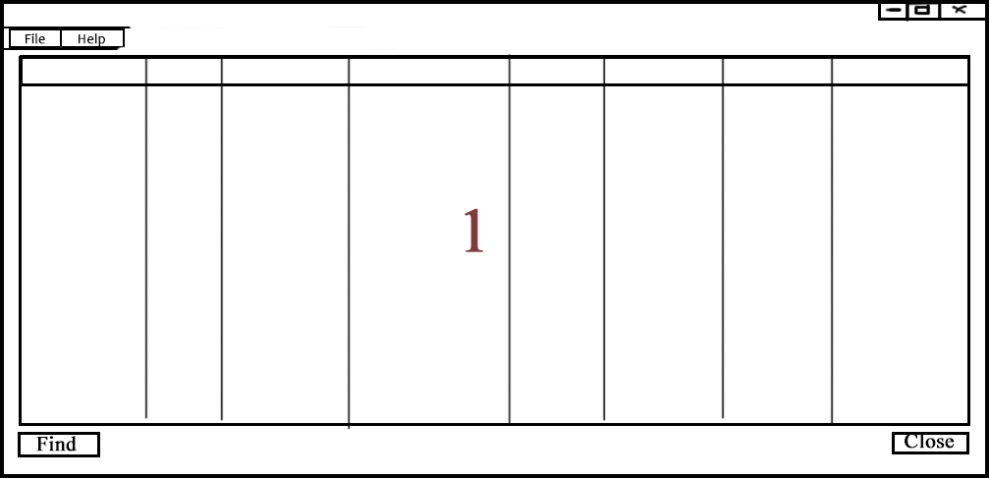


Рисунок 3 – главное окно вывода информации

1. Поле вывода информации в виде таблицы.

При нажатии на кнопку «Find» будет открываться стандартное окно поиска. Кнопка «Close» закрывает окно.

## 2.2 Функциональное обеспечение

Обозначим функции, которые должны выполняться в программе:

* Сбор информации о входящем НТТР трафике;
* Сбор информации об исходящем НТТР трафике;
* Запись информации в файл;
* Показ подключенных клиентов;
* Представление информации в системном трее;
* Запрос полученной информации у любого из клиентов;
* Поиск по значениям заголовков и просто по тексту;
* Установка и удаление клиента;
* Установка и удаление сервера;
* Изменять настройки в конфигурационных файлах.

Диаграмма вариантов использования представлена в Приложении А.

## 2.3 Структурное обеспечение

Серверное приложение содержит класс PacketsParser, который выполняет обработку принятых от клиента пакетов.  
Клиентское приложение содержит класс Sniffer, который инициирует потоки для ожидания сервера, для работы с сервером и поток для получения пакета информации. Класс Sniffer использует массив вспомогательного класса PacketsWriter, содержащий методы для записи пакета в файл и отправки его серверу.

Диаграмма классов представлена в Приложении Б.

## 2.4 Алгоритмическое обеспечение

При запуске клиентского приложения вы автоматически переводите его в режим ожидания подключения новых серверов и инициируете поток сбора информации о пакетах в файл log.db. При успешном подключении сервер можете выполнить некоторые операции над клиентом.

Алгоритм при нажатии на пункт Get History:

1. Отправка запроса клиенту;
2. Клиент отправляет файл log.db серверу;
3. Сервер принимает файл и выполняет обработку пакетов;
4. Сервер обновляет интерфейс;

Алгоритм при нажатии на пункт View:

1. Отправка запроса клиенту;
2. Клиент отправляет каждый пакет серверу;
3. Сервер выполняет обработку пакетов;
4. Сервер обновляет интерфейс;

Алгоритм при нажатии на пункт Find:

1. Вызов стандартного диалогового окна Find;
2. Поиск данных согласно запросу;

Диаграммы активности и последовательности представлены в Приложениях В-Г.

## 2.5 Архитектурное обеспечение

Для работы приложения потребуются:

1. Персональный компьютер под управлением одной из операционных систем семейства **MS Windows**: **Windows XP**, **Windows Vista**, **Windows 7**, **Windows 8**, **Windows 10**;
2. Наличие свободного места на жестком диске не менее 30 МБ;
3. Не менее 30 МБ свободного места в оперативной памяти;
4. Клавиатура, мышь, монитор;

Диаграмма развертывания представлена в Приложении Д.

## 2.6 Информационные структуры

Разрабатываемое приложение предназначено для работы с HTTP трафиком, следовательно, для работы программы требуется знать их структуру. Каждое HTTP-сообщение состоит из трёх частей, которые передаются в следующем порядке:

1. Стартовая строка — определяет тип сообщения;
2. Заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения — непосредственно данные сообщения.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа только тело сообщения.

Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так: Метод URI HTTP/Версия, где

* Метод — название запроса, одно слово заглавными буквами.
* [URI](http://ru.wikipedia.org/wiki/URI) определяет путь к запрашиваемому документу.
* Версия  — пара разделённых точкой [цифр](http://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B0%D0%B1%D1%81%D0%BA%D0%B8%D0%B5_%D1%86%D0%B8%D1%84%D1%80%D1%8B). Например: 1.0

Чтобы запросить страницу данной статьи, клиент должен передать строку (задан всего один заголовок):

GET /wiki/HTTP HTTP/1.0

Host: ru.wikipedia.org

Стартовая строка ответа сервера имеет следующий формат: HTTP/Версия КодСостояния Пояснение, где:

* Версия — пара разделённых точкой цифр как в запросе.
* Код состояния  — три цифры. По [коду состояния](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0#.D0.92.D0.B7.D0.B0.D0.B8.D0.BC.D0.BE.D0.B4.D0.B5.D0.B9.D1.81.D1.82.D0.B2.D0.B8.D0.B5_.D1.81_.D0.B2.D0.B5.D0.B1-.D1.81.D0.B5.D1.80.D0.B2.D0.B5.D1.80.D0.BE.D0.BC_.28HTTP.29) определяется дальнейшее содержимое сообщения и поведение клиента.
* Пояснение  — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

Например, стартовая строка ответа сервера на предыдущий запрос может выглядеть так:

HTTP/1.0 200 OK

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## 3.1 Детальная реализация функциональных частей программного обеспечения

### 3.1.1 Реализация функциональной части

В ходе разработки клиентского приложения был разработан класс Sniffer. Ниже приведены основные методы данного класса:

* void Start() – запускает три потока и инициализирует сырой сокет;
* void setTCPPort(UINT port) – устанавливает значение TCP порта;
* void setUDPPort(UINT port) – устанавливает значение UDP порта;
* void Stop() – закрывает потоки и сокет;
* функция Worker() – принимает и записывает или отправляет пакеты;
* bool isRunning(){return state;} – проверяет, был ли запущен сниффер, чтобы потоки не создавались повторно.

**Листинг 3.1 – запуск сниффера**

|  |
| --- |
| 1. void Sniffer::Start() 2. { 3. state = true; 4. Init(); 5. udp = new thread(&Sniffer::UDPListener,this); 6. tcp = new thread(&Sniffer::TCPListener,this); 7. worker = new thread(&Sniffer::Worker,this); 8. } |

**Листинг 3.2 – установка значение TCP порта**

|  |
| --- |
| 1. void setTCPPort(UINT port){tcp\_port = port;} |

**Листинг 3.3 – установка значение UDP порта**

|  |
| --- |
| 1. void setUDPPort(UINT port){udp\_port = port;} |

**Листинг 3.4 – закрытие потоков и сокета**

|  |
| --- |
| 1. void Sniffer::Stop() 2. { 3. state = false; 4. lock.lock(); 5. TerminateThread(udp->native\_handle(),0); 6. TerminateThread(tcp->native\_handle(),0); 7. TerminateThread(worker->native\_handle(),0); 8. closesocket(tcpsocket); 9. closesocket(udpsocket); 10. udp = tcp = worker = nullptr; 11. Close(); 12. lock.unlock(); 13. } |

**Листинг 3.5 – работа с пакетами**

|  |
| --- |
| 1. void Sniffer::Worker() 2. { while(state) 3. { 4. PACKET \*pack = GetPacket(); 5. if (!pack) 6. continue; 7. lock.lock(); 8. for (UINT i = 0; i < writers.size() ; i++){ 9. try 10. { 11. writers[i]->Write(pack); 12. } 13. catch (...) 14. { 15. writers.erase(writers.begin() + i); 16. i--; 17. } 18. } 19. delete pack; 20. lock.unlock(); 21. } 22. } |

Сниффер использует дополнительный класс PacketWriter, объекты которого выполняют запись принятых пакетов в файл либо отправку их на сервер.

**Листинг 3.6 – запись данных в файл**

|  |
| --- |
| 1. void PacketFileWriter::Write(const PACKET \*pack) 2. { 3. FILE \*fout = fopen("log.db","ab+"); 4. fwrite((char\*)&pack->ip,sizeof(IPV4\_HDR),1,fout); 5. fwrite((char\*)&pack->tcp,sizeof(TCP\_HDR),1,fout); 6. fwrite((char\*)&pack->time,sizeof(time\_t),1,fout); 7. int n = pack->headers.size(); 8. fwrite((char\*)&n,sizeof(int),1,fout); 9. for (int i = 0; i < pack->headers.size(); i++) 10. fwrite(pack->headers[i].c\_str(),pack->headers[i].size(),1,fout); 11. fclose(fout); 12. } |

**Листинг 3.7 – передача данных на сервер**

|  |
| --- |
| 1. void PacketTCPWriter::Write(const PACKET \*pack) 2. { 3. char dummy; 4. if (SOCKET\_ERROR == send(s,(char\*)&pack->ip,sizeof(IPV4\_HDR),0)) throw "Error"; 5. if (SOCKET\_ERROR == recv(s,&dummy,1,0)) throw "Error"; 6. if (SOCKET\_ERROR == send(s,(char\*)&pack->tcp,sizeof(TCP\_HDR),0)) throw "Error"; 7. if (SOCKET\_ERROR == recv(s,&dummy,1,0)) throw "Error"; 8. if (SOCKET\_ERROR == send(s,(char\*)&pack->time,sizeof(time\_t),0)) throw "Error"; 9. if (SOCKET\_ERROR == recv(s,&dummy,1,0)) throw "Error"; 10. int n = pack->headers.size(); 11. if (SOCKET\_ERROR == send(s,(char\*)&n,sizeof(int),0)) throw "Error"; 12. if (SOCKET\_ERROR == recv(s,&dummy,1,0)) throw "Error"; 13. for (int i = 0; i < pack->headers.size(); i++){ 14. if (SOCKET\_ERROR == send(s,pack->headers[i].c\_str(),pack->headers[i].length()+1,0)) throw "Error"; 15. if (SOCKET\_ERROR == recv(s,&dummy,1,0)) throw "Error"; 16. } |

В ходе разработки серверного приложения был разработан класс PacketsParser. Метод parse обрабатывает принятые пакеты.

**Листинг 3.8 – обработка принятых пакетов**

|  |
| --- |
| 1. ReqResp\* PacketsParser::parse(const PACKET &packet) 2. { 3. ReqResp elem; 4. if (ntohs(packet.tcp.dest\_port) == 80){ 5. elem.req = packet; 6. fullpacks.push\_back(elem); 7. }else{ 8. bool ok = false; 9. int i; 10. for (i = 0 ; i < fullpacks.size(); ++i) 11. { 12. if (fullpacks[i].req.tcp.source\_port == packet.tcp.dest\_port){ 13. fullpacks[i].resp = packet; 14. fullpacks[i].full = true; 15. ok = true; 16. break; 17. } 18. } 19. if (ok){ 20. ReqResp \*full = new ReqResp; 21. \*full = fullpacks[i]; 22. fullpacks.erase(fullpacks.begin()+i); 23. if (fullpacks.size()>100) 24. fullpacks.erase(fullpacks.begin(),fullpacks.begin()+fullpacks.size()/2); 25. return full; 26. } 27. } 28. return nullptr; 29. } |

### 3.1.2 Реализация пользовательского интерфейса

Пользовательский интерфейс данной программы был разработан с помощью дизайнера диалоговых окон и использования функций WinAPI.

Были задействованы такие элементы как:

* Button – кнопка;
* EditBox – поле ввода;
* List Control – таблица;
* Меню;
* Стандартное диалоговое окно поиска;

После запуска клиента появляется основное окно (см. рисунок 4). На данном этапе пользователю можно изменить значения портов. Затем можно запустить клиент.

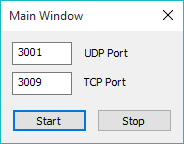


Рисунок 4 – Окно клиента

**Листинг 3.9 – реализация формы клиента**

|  |
| --- |
| 1. INT\_PTR CALLBACK MainWindow(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) 2. { 3. UNREFERENCED\_PARAMETER(lParam); 4. static bool bHideIcon=false; 5. static TCHAR ini\_path[1000]={0}; 6. switch (message) 7. {case WM\_INITDIALOG:{ 8. SetWindowPos(hDlg,0,300,300,0,0,SWP\_NOSIZE); 9. GetCurrentDirectory(1000,ini\_path); 10. lstrcat(ini\_path,TEXT("\\settings.ini")); 11. UINT port = GetPrivateProfileInt(TEXT("Global"),TEXT("UDP port"),3000,ini\_path); 12. SetDlgItemInt(hDlg,IDC\_EDIT1,port,FALSE); 13. port = GetPrivateProfileInt(TEXT("Global"),TEXT("TCP port"),3001,ini\_path); 14. SetDlgItemInt(hDlg,IDC\_EDIT2,port,FALSE); 15. return (INT\_PTR)TRUE; 16. } 17. case WM\_COMMAND: 18. if (LOWORD(wParam) == IDCANCEL) 19. { 20. EndDialog(hDlg, LOWORD(wParam)); 21. return (INT\_PTR)TRUE; 22. }else if (LOWORD(wParam) == ID\_START && !sniff.isRunning()){ 23. UINT port = GetDlgItemInt(hDlg,IDC\_EDIT1,NULL,FALSE); 24. TCHAR str[10]; 25. \_stprintf(str,TEXT("%u"),port); 26. WritePrivateProfileString(TEXT("Global"),TEXT("UDP port"),str,ini\_path); 27. sniff.setUDPPort(port); 28. port = GetDlgItemInt(hDlg,IDC\_EDIT2,NULL,FALSE); 29. \_stprintf(str,TEXT("%u"),port); 30. WritePrivateProfileString(TEXT("Global"),TEXT("TCP port"),str,ini\_path); 31. sniff.setTCPPort(port); 32. sniff.Start(); 33. MinimizeWndToTray(hDlg); 34. ShowNotifyIcon(hDlg,TRUE); 35. }else if (LOWORD(wParam) == ID\_STOP && sniff.isRunning()){ 36. sniff.Stop(); 37. } 38. break; 39. case WM\_TRAYMESSAGE: 40. switch(lParam) 41. { 42. case WM\_LBUTTONDBLCLK: 43. RestoreWndFromTray(hDlg); 44. bHideIcon=true; 45. return TRUE; 46. case WM\_LBUTTONUP: 47. if(bHideIcon) 48. { 49. ShowNotifyIcon(hDlg,FALSE); 50. bHideIcon=false; 51. } 52. return TRUE; 53. } 54. break; 55. } 56. return (INT\_PTR)FALSE; 57. } |

После запуска клиент сворачивается в системный трей:



Рисунок 5 – Клиент в системном трее

При запуске сервера пользователь видит окно сервера, где также можно указать порт, по которому сервер будет искать клиентов (см. рисунок 6).

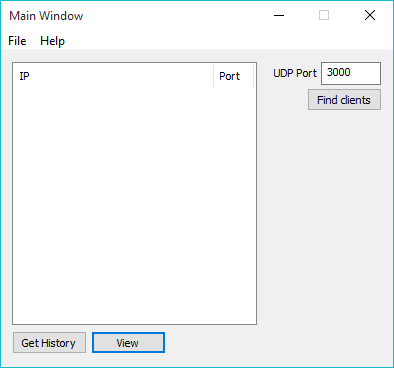


Рисунок 6 – Окно сервера

**Листинг 3.10 – реализация формы сервера**

|  |
| --- |
| 1. INT\_PTR CALLBACK MainWindow(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) 2. { 3. UNREFERENCED\_PARAMETER(lParam); 4. static TCHAR ini\_path[1000]={0}; 5. static HMENU menu; 6. switch (message){ 7. case WM\_INITDIALOG:{ 8. lv = GetDlgItem(hDlg,IDC\_LIST4); 9. SetWindowPos(hDlg,0,300,300,0,0,SWP\_NOSIZE); 10. ListView\_SetExtendedListViewStyle(lv,LVS\_EX\_SINGLEROW | LVS\_EX\_FULLROWSELECT); 11. InsertColumn(lv,TEXT("IP"),200); 12. InsertColumn(lv,TEXT("Port"),40); 13. GetCurrentDirectory(1000,ini\_path); 14. lstrcat(ini\_path,TEXT("\\settings.ini")); 15. UINT port = GetPrivateProfileInt(TEXT("Global"),TEXT("UDP port"),3000,ini\_path); 16. SetDlgItemInt(hDlg,IDC\_EDIT1,port,FALSE); 17. menu = LoadMenu(hInst,MAKEINTRESOURCE(IDC\_SERVER)); 18. SetMenu(hDlg,menu); 19. return (INT\_PTR)TRUE; 20. } 21. case WM\_COMMAND: 22. switch(LOWORD(wParam)){ 23. case IDM\_EXIT:case IDCANCEL:{ 24. EndDialog(hDlg, LOWORD(wParam)); 25. DestroyMenu(menu); 26. ExitProcess(0); 27. return (INT\_PTR)TRUE; 28. } 29. case IDM\_ABOUT: 30. MessageBox(hDlg,TEXT("Курсовая работа на тему \"HTTP-Logger\"\nВыполнила:\t\tРогайшис Т.С.\nПроверил:\t\tМатюш М.В."),TEXT("О программе"),MB\_OK|MB\_ICONINFORMATION); 31. break; 32. case IDM\_HELP: 33. ShellExecute(NULL,NULL,TEXT("help.chm"),NULL,NULL,TRUE); 34. break; 35. case ID\_HISTORY: 36. case ID\_VIEW:{ 37. if (LOWORD(wParam) == ID\_HISTORY) 38. interactive = false; 39. else 40. interactive = true; 41. if (ListView\_GetSelectedCount(lv)==0) 42. break; 43. int n = ListView\_GetNextItem(lv, -1, LVNI\_SELECTED); 44. ListView\_GetItemText(lv,n,0,ip,sizeof(ip)); 45. char prt[10]={0}; 46. ListView\_GetItemText(lv,n,1,prt,sizeof(prt)); 47. int ret = sscanf(prt,"%u",&selport); 48. DialogBox(hInst,MAKEINTRESOURCE(IDD\_HEADERS),hDlg,Headers); 49. break; 50. } 51. case ID\_CLIENTS:{ 52. if (!t){ 53. TCHAR str[10]={0}; 54. UINT port = GetDlgItemInt(hDlg,IDC\_EDIT1,NULL,FALSE); 55. \_stprintf(str,TEXT("%u"),port); 56. WritePrivateProfileString(TEXT("Global"),TEXT("UDP port"),str,ini\_path); 57. t = new std::thread(UDPThread,port); 58. } 59. break; 60. } 61. } 62. break; 63. } 64. return (INT\_PTR)FALSE; 65. } |

Чтобы найти клиентов, нужно нажать кнопку «Find clients». Если клиенты найдены, то они будут выведены в главном поле в виде списка. Чтобы начать работу с конкретным клиентом, его нужно выделить, щелкнув один раз мышью. Тогда станут доступны еще две функции: «Get History» - для получения истории – и «View» – для интерактивного режима слежения (см рисунки 7-8).

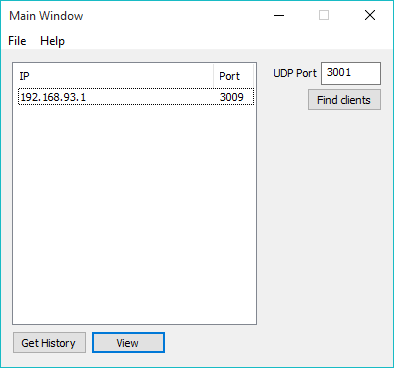


Рисунок 7 – Сервер с найденным клиентом

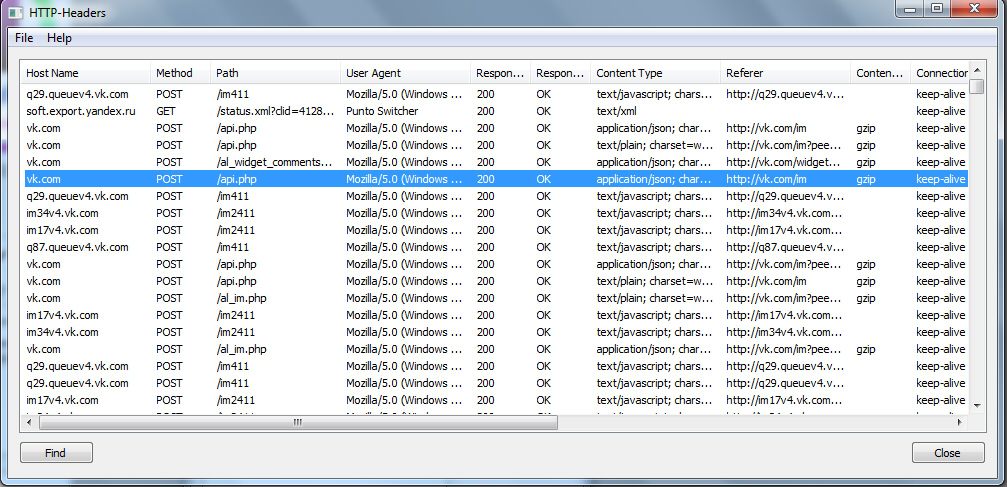


Рисунок 8 – Окно вывода информации

Вывод информации в режиме истории и интерактивном режиме практически не отличается. Разница лишь в том, что в режиме истории сервер получает файл с ранее отправленными запросами и ответами и выводит его. А в интерактивном режиме сервер получает запросы и ответы по мере их поступления. Изначально экран пуст, но затем там появляется информация.

В окне вывода информации есть кнопка «Find». При нажатии на нее вызывается стандартное окно поиска (см. рисунок 9).

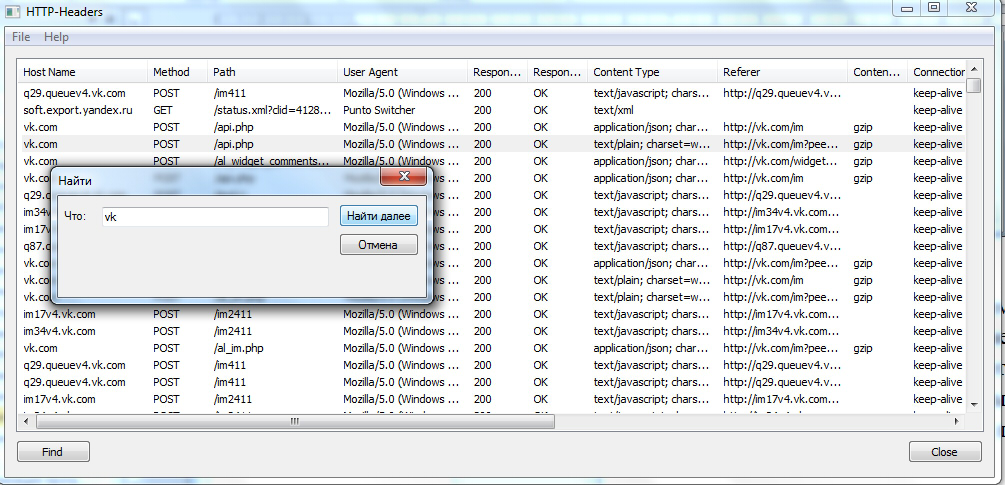


Рисунок 9 – Окно поиска

Здесь пользователь вводит слово либо часть слова и нажимает «Найти далее». Поиск может производиться как по всем записям, так и по конкретному полю, которое нужно предварительно выделить. При нахождении совпадений, строка, где оно найдено, выделяется. Если совпадения не найдены, то выводится сообщение:

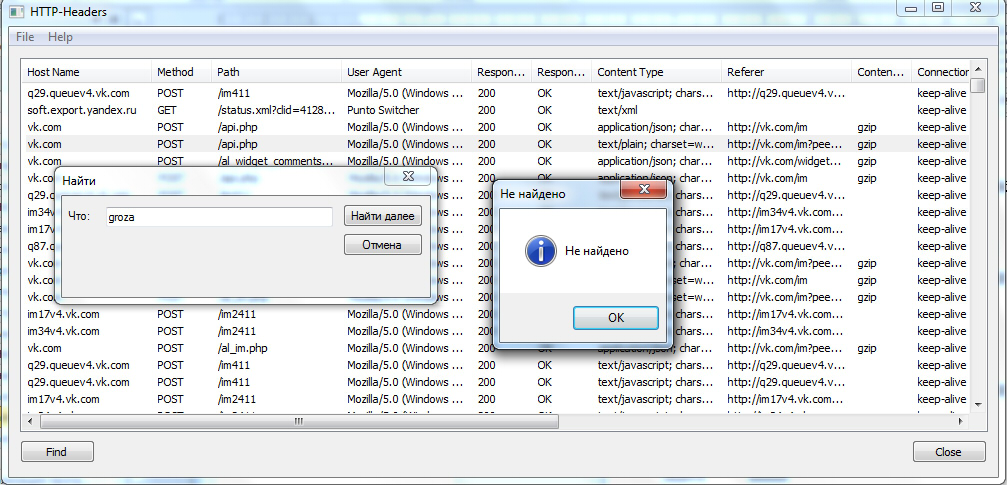


Рисунок 10 – Сообщение о ненайденных запросах

### 3.1.3 Реализация справочного модуля

Справочный модуль написан в формате chm в приложении Dr.Explain. Запуск chm файла происходит через пункт основного меню «Help -> Help» и представлен в листинге 3.11, запущенный модель справки представлен на рисунке 11.

**Листинг 3.11 – Открытие справочного модуля**

|  |
| --- |
| 1. case IDM\_HELP: 2. ShellExecute(NULL,NULL,TEXT("help.chm"),NULL,NULL,TRUE); 3. break; |

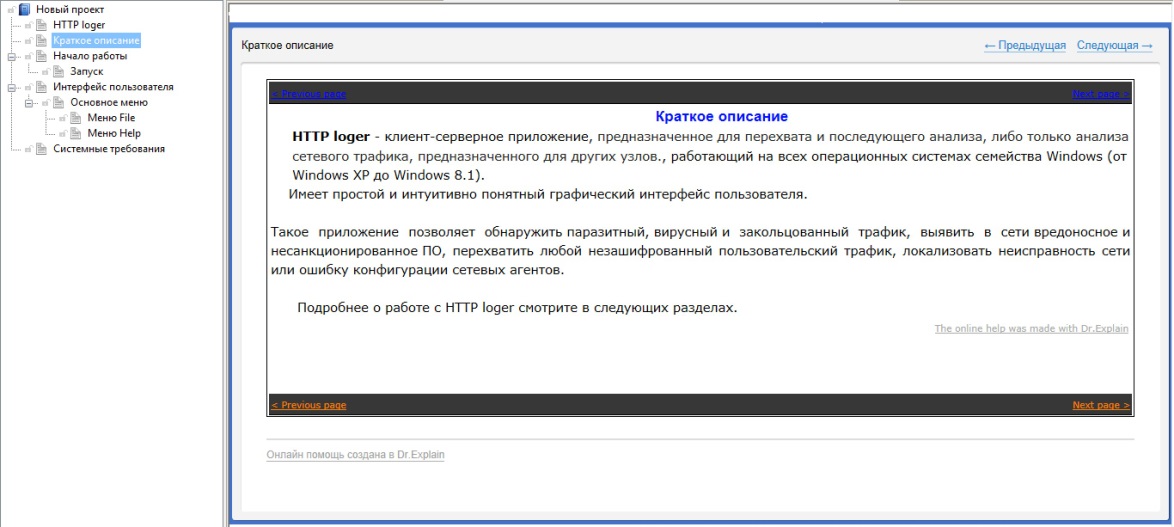


Рисунок 11 – Модуль справки

### 3.1.4 Реализация программы установки/удаления

Реализация установки\удаления программы была осуществлена с помощью программы Smart Install Maker v5.04.

## 3.2 Сопроводительная документация программного обеспечения

### 3.2.1 Минимальные аппаратные и системные требования

Для работы приложения требуются:

1. Персональный компьютер под управлением одной из операционных систем семейства **MS Windows**: **Windows XP**, **Windows Vista**, **Windows 7**, **Windows 8**, **Windows 10**;
2. Процессор с тактовой частотой не менее 1ГГц;
3. Наличие свободного места на жестком диске не менее 30 МБ;
4. Не менее 30 МБ свободного места в оперативной памяти;
5. Клавиатура, мышь, монитор;

### 3.2.2 Описание установки и запуска приложения

Для установки сервера нужно скопировать приложение Setup.exe на компьютер с диска, запустить его и проследовать инструкциям по установке. Аналогичным образом поступаем с файлом Setup1.exe для установки клиента.

Для запуска сервера необходимо запустить Server.exe. Для запуска клиента – Client.exe.

### 3.2.3 Руководство пользователя

Работа с клиентским приложением:

Клиент имеет всего две настройки – UDP и TCP порты, значения которых вводит пользователь. Для старта клиента необходимо нажать на кнопку «Start». Приложение свернется в системный трей. Для остановки клиента необходимо нажать кнопку «Stop». Для окончания работы с клиентом нажмите кнопку закрытия программы в правом верхнем углу.

Работа с серверным приложением:

1. Для начала работы нужно ввести UDP порт в поле ввода и нажать кнопку «Find clients».

2. Если серверу удастся найти хоть один клиент, то он будет выведен в главное поле окна. Для работы с клиентом, нужно выделить его в списке найденных клиентов.

3. Чтобы клиент прислал историю HTTP трафика, нужно нажать кнопку «Get History».

4. Чтобы отслеживание трафика происходило в интерактивном режиме, нужно нажать кнопку «View».

5. Чтобы осуществить поиск по запросам и ответам, нужно нажать на кнопку «Find» и ввести в строку часть слова либо слово целиком.

5. Для открытия справки нажмите в верхнем меню на вкладку «Help» и выберите формат пункт меню «Help».

6. Для окончания работы с сервером нажмите кнопку закрытия программы в правом верхнем углу или воспользуйтесь меню «File/Exit».

## 3.3 Анализ программного обеспечения

### 3.3.1 Анализ исходного кода приложения

Для анализа исходного кода приложения был использован стандартная утилита Visual Studio 2012.

Анализ исходного кода сервера до исправления ошибок представлен на рисунке:

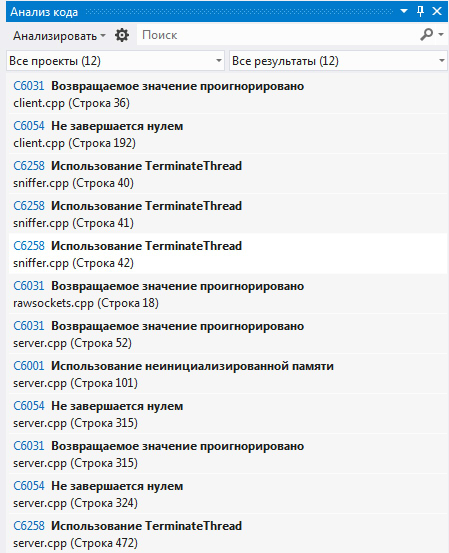


Рисунок 12 – Анализ исходного кода до исправления ошибок

Анализ исходного кода после исправления ошибок представлен на рисунке 13:

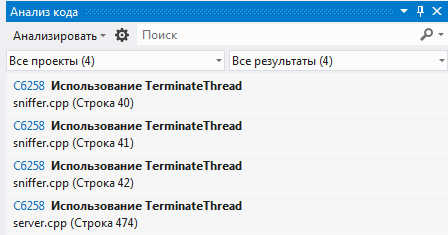


Рисунок 13 – Анализ исходного кода после исправления ошибок

### 3.3.2 Анализ производительности приложения

Для анализа производительности клиентского приложения была выбрана стандартная утилита Visual Studio 2012. Отчёт анализа производительности серверного приложения представлен на рисунке:

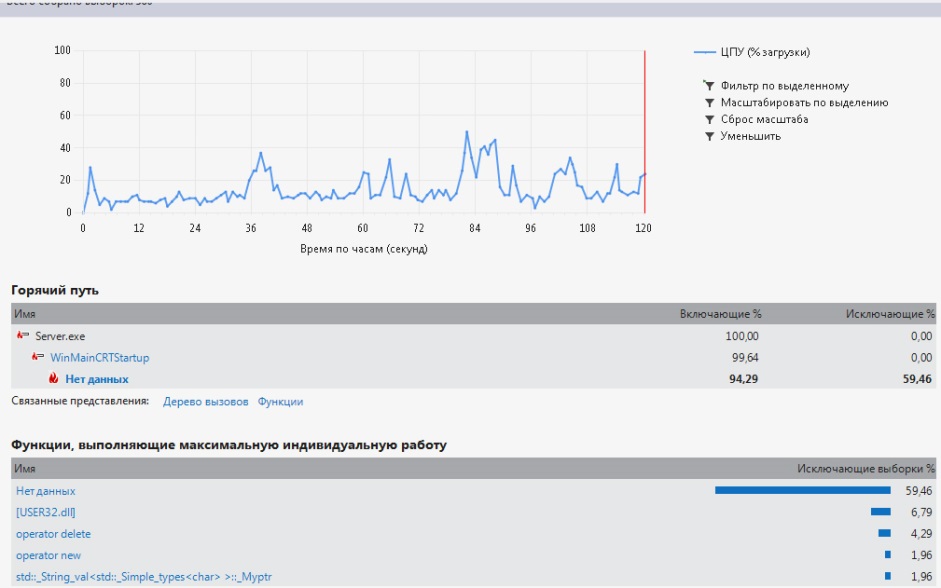


Рисунок 14 – Отчёт производительности серверного приложения

График анализа производительности клиента представлен на рисунке



Рисунок 15 – Отчёт производительности клиентского приложения

## 3.4 Тестирование программного обеспечения

### 3.4.1 Разработка модульных тестов

Для создания модульных тестов была выбрана стандартная утилита Microsoft Visual Studio 2012. Был создан тестовый класс, реализация которого представлена в листинге.

**Листинг 3.12 – Тестовый класс**

|  |
| --- |
| 1. #include "stdafx.h" 2. #include "CppUnitTest.h" 3. #include "../RawSockets/RawSockets.h" 4. using namespace Microsoft::VisualStudio::CppUnitTestFramework; 5. namespace UnitTest1 6. { 7. TEST\_CLASS(UnitTest1) 8. { 9. public: 10. TEST\_METHOD(TestGetPacket) 11. { 12. Init (); 13. PACKET \*pack=nullptr; 14. while(!pack) 15. pack=GetPacket(); 16. Close(); 17. Assert::AreEqual((unsigned char) 6,pack->ip.ip\_protocol); 18. } 19. }; 20. } |

Результаты тестирования представлены на рисунке 16:

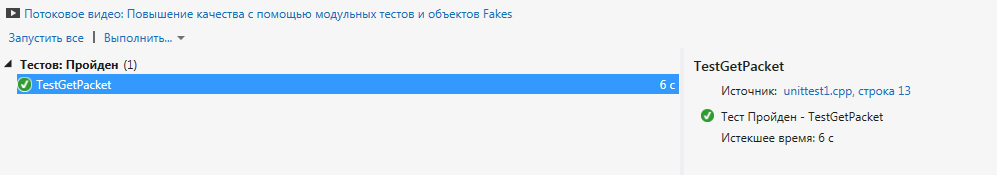


Рисунок 16 – Результаты модульного тестирования

### 3.4.2 Разработка и результаты интегрированного тестирования

Интегрированные тесты и их результаты приведены в таблицах 3.1 и 3.2 соответственно.

Таблица 3.1 – Тесты

|  |  |
| --- | --- |
| Цель теста | Действие |
| 1. Инициализация переменных клиентского приложения | Инициализация приложения |
| 1. Создание интерфейса клиентского приложения | Запуск приложения |
| 1. Инициализация переменных серверного приложения | Инициализация приложения |
| 1. Подключение клиента к серверу | Подключение |
| 1. Передача пакетов | Передача пакетов |
| 1. Закрытие сервера | Неожиданное отключение сервера |
| 1. Закрытие клиента | Неожиданное отключение клиента |
| 1. Закрытие соединения | Клиент закрывает соединение |

Таблица 3.2 – Результаты тестов

|  |  |
| --- | --- |
| Номер теста | Результат |
| 1 | Пройден |
| 2 | Пройден |
| 3 | Пройден |
| 4 | Пройден |
| 5 | Пройден |
| 6 | Пройден |
| 7 | Пройден |
| 8 | Пройден |

# ЗАКЛЮЧЕНИЕ

В процессе написания данной курсовой работы были закреплены навыки программирования на языке С++. В ходе выполнения курсовой работы было разработано два сетевых приложения. Были реализованы все основные функции необходимые при прослушивании HTTP траффика другого компьютера. А также закреплены знания, полученные в процессе изучения дисциплины «Операционные системы и системное программирование».

Программа имеет простой и интуитивно понятный интерфейс и соответствует всем заявленным требованиям и условиям курсовой работы.

# СПИСОК ЛИТЕРАТУРЫ

1. **Страуструп, Б.**   Язык программирования С++ / Б. Страуструп - 3-е изд., перераб. и доп. - М. : Бином, 2010. - 1136 с. - ISBN 978-5-9518-0425-9.
2. **Лафоре, Р.**   Объектно-ориентированное программирование в С++ / Р. Лафоре - 1-е изд. - М. : Питер, 2010. - 928 с. - ISBN 978-5-4237-0038-6.

### [Википедия*—* свободная энциклопедия](http://ru.wikipedia.org/)[Электронный ресурс]*.* URL: http://ru.wikipedia.org/

# ПРИЛОЖЕНИЕ А

# Диаграмма вариантов использования



Рисунок 17 – Диаграмма вариантов использования

# ПРИЛОЖЕНИЕ Б

# Диаграмма классов

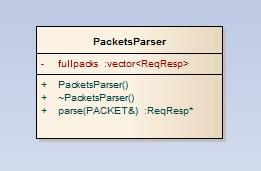


Рисунок 18 – Диаграмма классов сервера

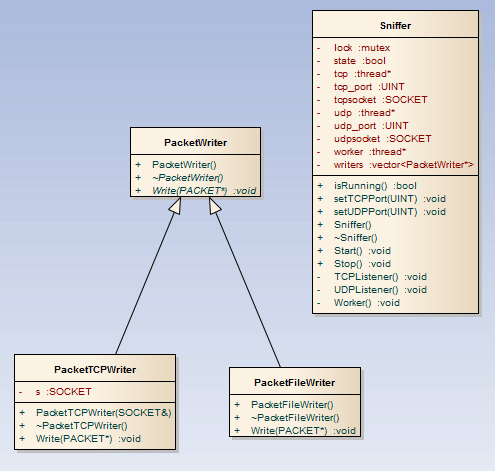


Рисунок 19 – Диаграмма классов клиента

# ПРИЛОЖЕНИЕ В

# Диаграмма активности

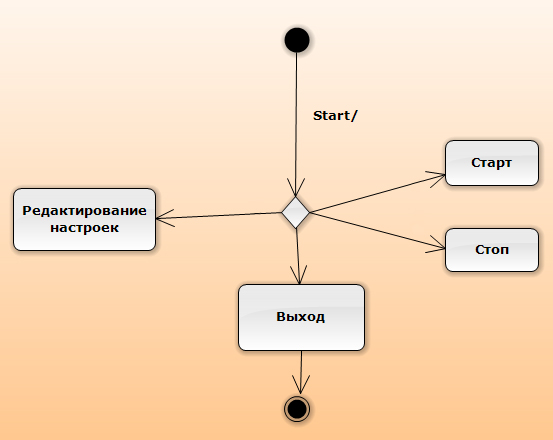


Рисунок 20 – Диаграмма активности клиента

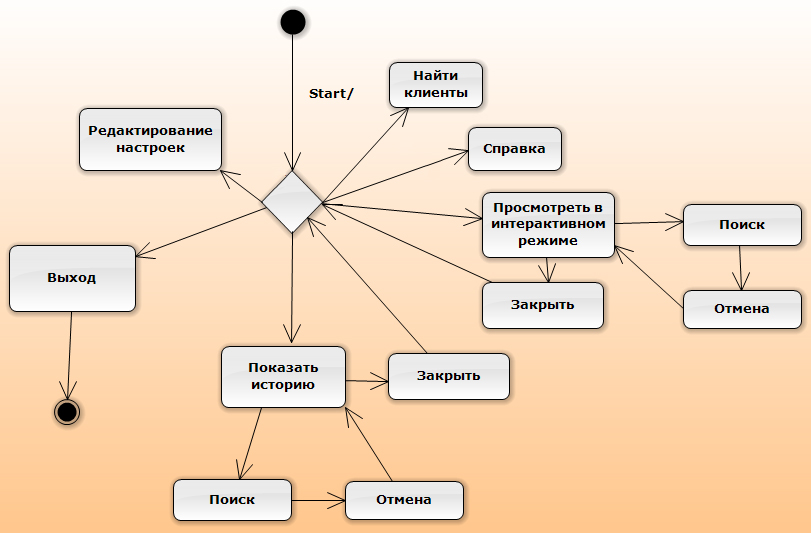


Рисунок 21 – Диаграмма активности сервера

# ПРИЛОЖЕНИЕ Г

# Диаграмма последовательности

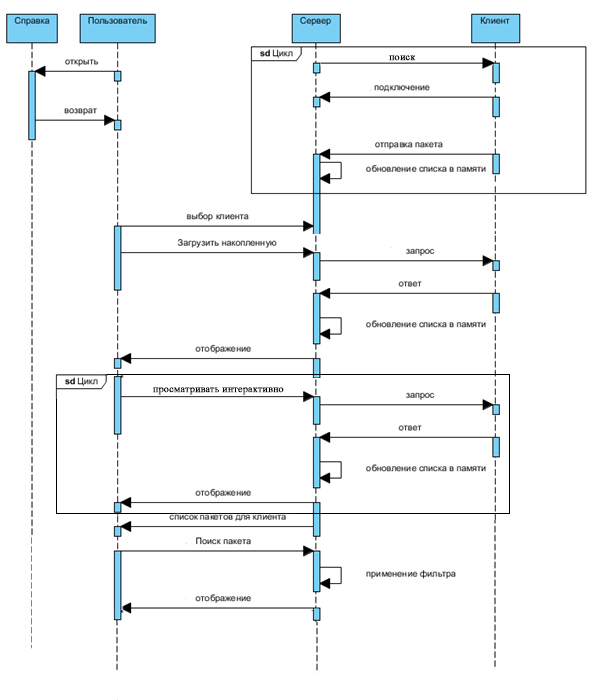


Рисунок 22 – Диаграмма последовательности

# ПРИЛОЖЕНИЕ Д

# Диаграмма развертывания

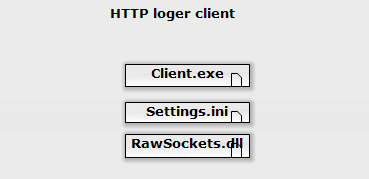


Рисунок 23 – Диаграмма развертывания клиента

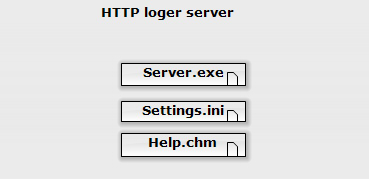


Рисунок 24 – Диаграмма развертывания сервера